Abstract of "Quantifying Uncertainty in Data Exploration" by Yeounoh Chung, Ph.D., Brown University, April 2019.

In the age of big data, uncertainty in data constantly grows with its volume, variety and velocity. Data is noisy, biased and error-prone. Compounding the problem of uncertain data is uncertainty in data analysis. A typical end-to-end data analysis pipeline involves cleaning and processing the data, summarizing different characteristics and running more complex machine learning algorithms to extract interesting patterns. The problem is that all the steps are error-prone and imperfect. From the input data to the output results, uncertainty propagates and compounds. This thesis addresses a subset of the uncertainty problems in data exploration.

First, it is shown how uncertainty in the form of missing unknown data items can affect aggregate query results, which are very common in exploratory data analysis. It is challenging to make sure that we have collected all the important data items to derive correct data analysis, especially when we deal with real-world big data; there is always a chance that some items of unknown impacts are missing from the collected data set. To this end, sound techniques to derive aggregate queries with the *open-world assumption* (the data set may or may not be complete) are proposed.

Next, uncertainty in the form of data errors is examined. It is almost guaranteed that any real-world data sets contain some forms of data error. This is an important source of uncertainty in data analysis because those errors would almost surely corrupt the final analysis results. Unfortunately, there has not been much work to measure the quality of data in terms of the number of remaining data errors in the data set. The data cleaning best practices basically employ a number of (orthogonal) data cleaning techniques, algorithms or human/crowd workers in a hope that the increased cleaning efforts would result in a perfectly clean data set. To guide such cleaning efforts, techniques to estimate the number of undetected remaining errors in a data set are proposed.

Lastly, an uncertainty problem related to machine learning (ML) model quality is addressed. ML is one of the most popular tools for learning and making predictions on data. For its use, ensuring good ML model quality leads to more accurate and reliable data analysis results. The most common practice for model quality control is to consider various test performance metrics on separate validation data sets; however, the problem is that the overall performance metrics can fail to reflect the performance on smaller subsets of the data. At the same time, evaluating the model on all possible subsets of the data is prohibitively expensive, which is one of the key challenges in solving this uncertainty problem.

Quantifying Uncertainty in Data Exploration

by
Yeounoh Chung
B.S., Cornell University 2008
M. Eng., Cornell University, 2009

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
April 2019

This dissertation by Yeounoh Chung is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Tim Kraska, Director

Recommended to the Graduate Council

Date _____ _____

Peter J. Haas, Reader
(University of Massachusetts, Amherst)

Date _____ _____

Eli Upfal, Reader
(Brown University)

Approved by the Graduate Council

Date _____ _____

Andrew G. Campbell
Dean of the Graduate School

Vitae

Yeounoh Chung attended Cornell University, graduating with a B.S. in Electrical and Computer Engineering and a M.Eng. in Computer Science. Before coming to Brown, he worked in the Korean Information and Communications Technology (ICT) industy for several years. In 2014, he started his Ph.D. program at Brown working towards "Quantifying Uncertainty in Data Exploration."

Publications related to this thesis

"Automated Data Slicing for Model Validation: A Big data - AI Integration Approach, " Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. IEEE Transactions on Knowledge and Data Engineering (TKDE), 2019.

"Slice Finder: Automated Data Slicing for Model Validation," Yeounoh Chung, Tim Kraska, Steven E. Whang, and Neoklis Polizotis. IEEE International Conference on Data Engineering (ICDE), 2019.

"Towards Quantifying Uncertainty in Data Analysis & Exploration," Yeounoh Chung, Sacha Servan-Schreiber, Emanuel Zgraggen, and Tim Kraska. IEEE Data Engineering Bulletin, 2018.

"Estimating the Impact of Unknown Unknowns on Aggregate Query Results," Yeounoh Chung, Michael L. Mortensen, Carsten Binnig, and Tim Kraska. ACM Transactions on Database Systems (TODS), 2018.

"A Data Quality Metric (DQM): How to Estimate the Number of Undetected Errors in Data Sets," Yeounoh Chung, Sanjay Krishnan, and Tim Kraska. Very Large Data Bases (VLDB), 2017.

"Estimating the Impact of Unknown Unknowns on Aggregate Query Results," Yeounoh Chung, Michael L. Mortensen, Carsten Binnig, and Tim Kraska. ACM Special Interest Group on Management of Data (SIGMOD), 2016. [Best of SIGMOD 2016]

Dedicated to my parents.
They have loved and raised me the best way they knew.

Acknowledgements

My small success today is attributed to many, including my advisor Tim. It is a very reflection of Tim's guidance and the skills that he has passed on. After studying for years under him, I now better see how brilliant my advisor was. In a very real sense, In addition to his support throughout my study, I would like to take this opportunity to thank him again for your VLDB'17 presentation; "life is full of the unexpected." I truly admire Tim's commitments to excellence in research and teaching. For that, I will always be grateful.

I am also thankful to the community of researchers, friends and mentors at Brown and Rhode Island Central Korean Church. I'd like to thank Carsten Binnig, Eli Upfal and all the members of Brown Data Management Research group for sharing their brilliant minds in my research work. And special thanks to Peter J. Haas from UMass Amherst and Jee-Hyong Lee from Sungkyunkwan University for your generous advice.

And I am ever grateful to my friends, family and God.

Contents

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the age of big data, uncertainty in data constantly grows with its volume, variety and velocity. Data is noisy, biased and error-prone. Compounding the problem of uncertain data is uncertainty in data analysis. A typical end-to-end data analysis pipeline involves cleaning and processing the data, summarizing different characteristics and running more complex machine learning algorithms to extract interesting patterns. The problem is that all the steps are error-prone and imperfect. From the input data to the output results, uncertainty propagates and compounds.

In this thesis, several potential uncertainty problems in data analysis are discussed along with the proposed solutions. The works presented here are part of a project called, "Quantifying the Uncertainty in Data Exploration" (QUDE; pronounced "cute"), which aims to automatically quantify different types of uncertainty and errors within data exploration pipelines. On one hand, there are the obvious types of uncertainty, such as outliers, which can be easily detected via simple visualizations (e.g., error bars, scatter plots). On the other hand, there are various types of uncertainty that are critical for reliable analysis results, but often overlooked in practice. The goal is to provide techniques for automatically detecting and quantifying such missed types of uncertainty. This can help users without deep statistical or machine learning backgrounds to derive more safe and reliable data analytical conclusions.

1.1 Types of Uncertainty in Data Analysis

A typical end-to-end data analysis pipeline involves collecting and cleaning data, summarizing different characteristics and running complex machine learning algorithms. At a high level, a data analysis and exploration workflow involves multiple stages illustrated in Figure 1.1. Namely, it is necessary to extract high quality data from multiple data sources (or samples), clean the data to remove the inconsistency and merge the same entity in heterogeneous formats (i.e., de-duplication), apply data analysis techniques (e.g., simple aggregate queries or more complex machine learning algorithms) to extract useful information and extract interesting insights. Finally, one must present the results in the right context for the interpretation.

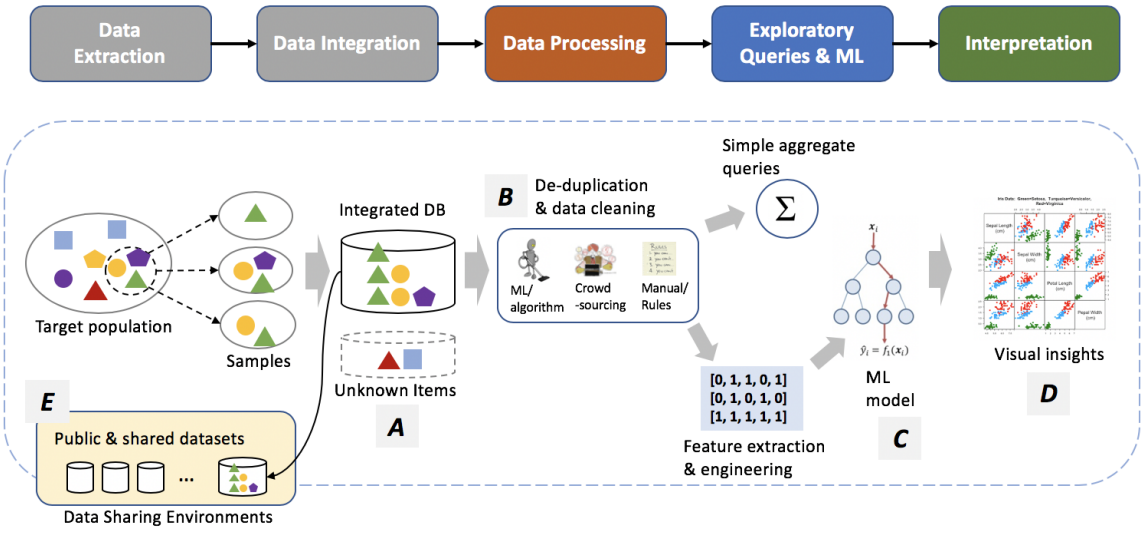


Figure 1.1: Data analysis & exploration workflow (top) and an example pipeline (bottom).

While such data analysis best practices can produce actionable insights and discoveries, *one should not take anything at face value*. The size and complexity, noise and incompleteness with big data not only impede the progress of the pipeline, but also make each step in the pipeline more error-prone. The uncertainty around the quality of the intermediate results propagates and compounds, making it even more difficult to validate the output results. Obviously, many quality-related concerns exist in the entire pipeline from entity resolution problems up to Simpson-Paradox problems for aggregated results.

As part of QUDE, this thesis focuses on the following forms of uncertainty around the data analysis pipeline, that are less explored in the literature and/or no automatic technique exists to control the uncertainty type. Addressing these forms of uncertainty, in turn, should provide various measures to correct and validate the output results and discoveries.

A. Missing Unknown Data Items [8–10]: Incompleteness of the data is one of the most common sources of uncertainty in practice. For instance, if unknown data items are missing (i.e., *one cannot tell if the database is complete or not*) from the unknown target population, even a simple aggregate query result, like SUM, can be questionable. It is challenging to make sure that we have collected all the important data items to derive correct data analysis.

B. Undetected Data Errors [11]: Complicating the challenges of incomplete data is also the quality of the collected data items. Real-world data is noisy and almost always comes with a variety of errors. Such "dirty" data must be removed or corrected because errors can and will bias the results. There are many techniques to identify and repair the errors, but no single technique can guarantee a perfect error coverage. The challenge is that one would use a number of orthogonal cleaning techniques or hire a lot of crowd-workers without knowing

Table 1.1: More commonly-studied data analysis errors/uncertainty [1–7]. QUDE focuses on other types of uncertainty that are crucial for safe and reliable data analysis, but often overlooked in practice.

Data Extraction	Data Integration	Data processing	Exploratory Queries & ML	Interpretation
Sample selection bias	Entity resolution	Combination of error types	Model selection	Human involvement
Data source validity	Un-/semi-structured data	Data cleaning & coverage	Hyper-parameter tuning	Visualization selection
Disparate data sources	Data enrichment	Human involvement	Model bias & variance	Deceptive visualizations
		Missing information	Data under-/over-fitting	
			Feature engineering	
			Concept/distribution shift	

when to stop. Thus, we want to estimate *how many errors are still remaining in the data set*, without knowing the ground truth (a complete/perfect set of constraint rules or the true number of errors in the data set).

C. Model Quality [12]: ML is one of the most popular tools for learning and making predictions on data. For its use, ensuring good ML model quality leads to more accurate and reliable data analysis results. The most common practice for model quality control is to consider various test performance metrics on separate validation data sets (e.g., cross-validation); however, the problem is that the overall performance metrics can fail to reflect the performance on smaller subsets of the data. At the same time, evaluating the model on all possible subsets of the data is prohibitively expensive, which is one of the key challenges in solving this uncertainty problem.

This thesis focuses on these uncertainty types for two reasons: One, they are important for safe and reliable data analysis; Two, they are overlooked by the common data analysis and data wrangling systems [13–17]. Table 1.1 lists other types of errors or uncertainty that are more commonly considered in practice. The initial prototype of QUDE [18] also looks at false discovery control and data sharing environment to enable more reliable data analysis.

1.2 Uncertainty as Missing Unknown Data Items

First, it is shown how uncertainty in a form of missing unknown data items can affects aggregate query results, which are very common in exploratory data analysis. It is challenging to make sure that we have collected all the important data items to derive correct data analysis, especially when we deal with real-world big data; there is always a chance that some items of unknown impacts are missing from the collected data set. To this end, sound techniques to derive aggregate queries with the *open-world assumption* (the data set may or may not be complete) are proposed.

1.2.1 The Impact of Unknown Unknowns

In the past few years, the number of data sources has increased exponentially because of the ease of publishing data on the web, the proliferation of data-sharing platforms (e.g., Google Fusion Table [19]

or Freebase [20]), and the adoption of open data access policies, both in science and government. The success of crowdsourcing provides another virtually unlimited source of information [21–28]. This deluge of data has enabled data scientists, both in commercial enterprises and in academia, to acquire and integrate data from multiple data sources, achieving higher quality results than ever before. It is therefore not surprising that industry and academia alike have developed highly sophisticated systems and tools to assist data scientists in the process of data integration [29]. However, even with a perfectly cleaned and integrated data set, two fundamental questions remain: (1) do the data sources cover the complete data set of interest and (2) what is the impact of any unknown (i.e., unobserved) data on query results?

To answer such fundamental quetions, techniques to estimate the impact of the unknown data on aggregate queries of the form `SELECT AGGREGATE(attr) FROM table WHERE predicate` are developed.

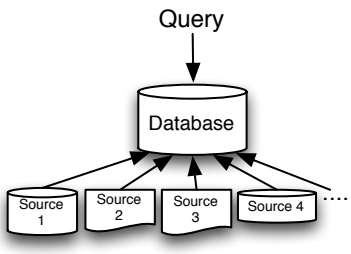


Figure 1.2: Simple data integration scenario where multiple data sources overlap but are not necessarily complete.

For the purpose of this work, a simple data integration scenario is assumed, as depicted in Figure 1.2. Several domain-related data sources are integrated into one database, preserving the lineage information for each data item or record. Naturally, these data sources overlap with each other, but even when put together they might not be complete. For example, all data sources in Figure 1.2 might list U.S. tech companies but some smaller companies might not be mentioned in any of the sources. This data integration scenario applies to a wide range of use cases ranging from crowdsourcing (where every crowdworker can be considered a single data source [22]) to data extraction from web pages.

Estimating the impact of the unknown data (data items that are not observed in any data source) is particularly difficult as both the number of missing unique data items and their values are unknown; thus, we deal with **unknown unknowns**. This characteristic distinguishes our work from what is generally known as *missing data*, or *known unknowns*, estimation in Statistics [30–32], which tries to estimate the value of unknown (missing) attributes for known records. At a first glance, it may seem impossible to estimate the impact of *unknown unknowns*; however, for a large class of data integration scenarios, the analysis of overlap of multiple data sources makes it feasible.

1.2.2 An Illustrative Example

To demonstrate the impact of *unknown unknowns*, let us asssume a simple aggregate query to calculate the number of all employees in the U.S. tech industry, `SELECT SUM(employees) FROM us_tech_companies`, over a crowdsourced data set. For this experiment, techniques from [22] are

used to collect employee numbers from U.S. tech companies[1] using Amazon Mechanical Turk (AMT). The data was manually cleaned before processing (e.g., entity resolution, removal of partial answers). Figure 1.3 shows the result.

The red line represents the ground truth (i.e., the total number of employees in the U.S. tech sector) for the query, whereas the grey line shows the result of the observed SUM query over time with the increasing number of received crowd-answers. As the ground-truth, we used the US tech sector employment report from the Pew Research Center [33]. The gap between the observed and the ground truth is due to the impact of the *unknown unknowns*, which gets smaller at a diminishing rate as more crowd-answers arrive.

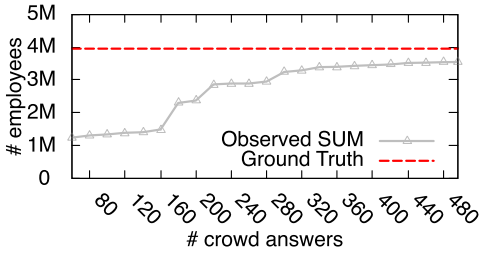


Figure 1.3: Employees in the U.S. tech sector. The original SUM query result (Observed SUM) never reaches the ground truth.

While the experiment was conducted in the context of crowdsourcing, the same behavior can be observed with other types of data sources, such as web pages. For instance, suppose a user searches the Internet to create a list of all solar energy companies in the U.S. The first few web pages will provide the greatest benefit (i.e., more new solar companies), while after a dozen web pages the benefit of adding another web page diminishes as the likelihood of duplicates increases. The rate of increasing overlap of data sources is indicative of the completeness of the data set.

1.2.3 Goal and Approach

This work is a first step towards developing techniques to estimate the impact of the *unknown unknowns* on query results.

We build upon the previous results [34] that proposed techniques to estimate the *number of unknowns*, and extend it to be also able to estimate the value of the missing items for simple aggregate queries, especially *SUM*-aggregates, but also other aggregates like *AVG*, *MIN*, and *MAX*. We design techniques that can deal with the peculiarities of the data integration scenarios discussed before, such as uneven contributions from different sources (bias of data sources).

In this work, we use crowdsourced data sets because they are easier to collect, but the techniques are general and apply to almost all data integration scenarios that combine overlapping data sources. While we do not argue that the proposed techniques can predict black-swan-like data items (i.e., extremely rare data items), we will show that our techniques can provide useful estimates under more "normal" circumstances, which we will define more formally. For instance, in the example of Figure 1.3, we can get an almost perfect estimate of the impact of the *unknown unknowns* after only 350 crowd-answers. In addition, by building upon recent work on the Good-Turing estimator [35],

[1] More precisely, we only asked for companies with a presence in Silicon Valley, as we found it provides more accurate results.

we are able to provide an upper bound for our estimates under easy to understand conditions.

In this proposal, we make the following contributions:

- We formalize the problem of estimating the impact of *unknown unknowns* on query results (Section 2.1.2) and describe why existing techniques for species estimation and missing data estimation are not sufficient (Section 1.2.2).
- We develop techniques to estimate the impact of the *unknown unknowns* on aggregate query results (Section 2.2).
- We derive a first upper bound for *SUM*-aggregate queries (Section 2.3).
- We examine the effectiveness of our techniques via experiments using both real and synthetic data sets (Section 2.5).

1.3 Uncertainty as Undetected Data Errors

Next, uncertainty in the form of data errors is examined. It is almost guaranteed that any real-world data sets contain some forms of data error. This is an important source of uncertainty in data analysis because those errors would almost surely corrupt the final analysis results. Unfortunately, there has not been much work to measure the quality of data in terms of the number of remaining data errors in the data set. The data cleaning best practices basically employ a number of (orthogonal) data cleaning techniques, algorithms or human/crowd workers in a hope that the increased cleaning efforts would result in a perfectly clean data set. To guide such cleaning efforts, techniques to estimate the number of undetected remaining errors in a data set are proposed.

1.3.1 Quantifying the Remaining Errors

While this is a seemingly simple question, it is actually extremely challenging to define data quality without knowing the ground truth [36–41]; previous works define data quality through counting the losses to gold standard data or violations of the constraint rules set forth by domain-specific heuristics and experts [42–46]. In practice, however, such ground truth data or rules are not readily available and are incomplete (i.e., there exists a "long tail" of errors).

In this work, we set out to design a statistical estimator to address both of the issues. That is, we need to estimate the number of remaining errors without knowing the ground truth in the presence of false negative and false positive errors. A simple approach is to extrapolate the number of errors from a small *"perfectly clean"* sample (as in [47]): (1) we take a small sample, (2) perfectly clean it manually or with the crowd, and (3) extrapolate our findings to the entire data set. For example, if we found 10 new errors in a sample of 1000 records out of 1M records, we would assume that the total data set contains 10000 additional errors. However, this naïve approach presents a *chicken-and-egg* paradox. If we clean a very small sample of data, it may not be representative and

thus will give an inaccurate extrapolation or estimates based off it. For larger samples, how can the analyst know that the sample itself is perfectly clean without a quality metric?

To this end, we design a statistical estimator based on the principle of diminishing returns, which basically states that every additional error is more difficult to detect. For example, with experts or crowd sourcing, the first (crowd-)worker to pass over the data set finds more new errors than every subsequent worker, and so on. The key insight is to estimate this diminishing return rate (i.e., fewer errors are found in each pass) and project this rate forward to estimate the number of errors if there were an infinite number of workers.

Note that, while we use crowd-sourcing as our motivation, the same observation also applies for purely algorithmic cleaning techniques [48] (e.g., machine learned error classifiers). Each additional data cleaning algorithm applied to a data set will have a diminishing utility to the analyst. However, in this paper, we focus on crowd-sourced cleaning techniques because they empirically achieve high quality results and are widely used in the industry [49]. Exploring pure algorithmic techniques, which make some problems easier but others harder as explained later, is beyond the scope of this paper. In this work, we present examples on entity resolution and malformed entities, although our statistical techniques can also be applied to other types of data errors as long as the errors are countable (i.e., independent workers provide the amount of dirty data or errors they could find in the dataset).

1.3.2 An Illustrative Example

For instance, take a simple data cleaning task where we want to identify (and manually fix) malformed US home addresses in the database, shown in Figure 1.4. As in Guided Data Repair (GDR) [42], we might have a set of rules to propose repairs for missing values ($r1$, $r2$) and functional dependency ($r1$, $r3$, $r6$), but not for US state/city name misspellings ($r3$, $r4$) or wrong home addresses ($r5$, $r6$), in a seemingly valid format that only the most observant crowd-workers might catch. Once errors are identified, a human can verify the proposed errors and automatic repairs. Similarly, as in CrowdER [50], we can run inexpensive heuristics to identify errors and ask crowd-workers to confirm. In both of these cases, the fallibility of the system in the form of false negative (e.g., "long tail" or missed errors) and false positive (e.g., even humans can make mistakes) errors is a big concern.

1.3.3 Goal and Approach

Our goal is to estimate the number of all (eventually) detectable errors. We are not concerned with the errors that are not detectable even with infinite resources. In the case of our two-stage data cleaning with crowd-sourcing, infinite resources means an infinite number of crowd-workers, who would eventually reach the correct consensus decisions (i.e., dirty or clean) for all candidate matches.

Maybe surprisingly, this problem is related to estimating the completeness of query results using

	address	city	state	zip
r1:	15440 Southwest Mallard Drive Appartment # 101	Portland		97007
r2:	15440 SW Mallard Drive Apt # 102	Portland	OR	
r3:	12855 Southwest Dipper Lane Apartment # 101	Patland	OR	97007
r4:	289 Angell steet, unit 1H	Providence	RA	2912
r5:	Boston House, 239 S Indian River Dr	Fort Pierce	FL	34950
r6:	12345 ABCD street # EFGH	New York	NY	97007

Figure 1.4: Erroneous US home addresses: $r1$ and $r2$ contain missing values; $r3$ and $r4$ contain invalid city names and zip codes; $r1$, $r3$, and $r6$ violate a functional dependency ($zip \rightarrow city, state$); $r5$ is not a home address, and $r6$ is a fake home address in a valid format.

species estimation techniques as first proposed in [51]. We can think of our data quality problem as estimating the size of the set containing all distinct errors that we would discover upon adding more workers. Each worker marks a subset of records as dirty, with some overlaps with other workers. The idea is to estimate the number of distinct records that will be marked erroneous if an infinite number of workers, $K \rightarrow \infty$, are added.

Unfortunately, existing species estimation techniques [9, 51] to estimate the completeness of a set, do not consider that workers can make mistakes. At the same time in any real data cleaning scenario, workers can make both false negative errors (a worker fails to identify a true error) and false positive errors (a worker misclassifies a clean item as dirty). It turns out that false positives have a profound impact on the estimation quality of how many errors the data set contains. This is because species estimators rely on the number of observed "rare" items as a proxy for the number of remaining species, and this number can be highly sensitive to a small number of false positive errors.

To address this issue, we reformulate the estimation problem to estimate the number of distinct changes in the majority consensus. To the best of our knowledge, this is the first research done on the diminishing return effect, in the context of consensus, and to develop techniques to estimate the number of remaining errors in a data set, without knowing the true number of errors in the original dataset or a complete set of rules to define a perfect dataset.

In this proposal, we make the following contributions:

- We formalize the link between data quality metrics and species estimation, which enables estimation of the number of remaining errors without the knowledge of the ground truth (Section 3.3).
- Analytically and empirically, we show that traditional species estimators are very sensitive to errors from the crowd (Section 3.3.2, 3.6.2).
- We propose a variant of the species estimation problem that estimates the number of distinct majority switching events, find that this new estimator is more accurate in the presence of

noise (e.g., misclassified errors), and show how this estimate can be used to determine the quality of the data set (Section 3.4).

- We evaluate our switch-based quality metric using real- and synthetic data sets and find that they provide much more accurate and reliable estimates with fallible crowds (Section 3.6).

1.4 Uncertainty as Model Quality

ML is one of the most popular tools for learning and making predictions on data. For its use, ensuring good ML model quality leads to more accurate and reliable data analysis results. The most common practice for model quality control is to consider various test performance metrics on separate validation data sets; however, the problem is that the overall performance metrics can fail to reflect the performance on smaller subsets of the data. At the same time, evaluating the model on all possible subsets of the data is prohibitively expensive, which is one of the key challenges in solving this uncertainty problem.

1.4.1 Automated Data Slicing for Model Validation

Machine learning (ML) systems [52] are becoming more prevalent thanks to a vast number of success stories. However, the data tools for interpreting and debugging models have not caught up yet and many important challenges exist to improve our model understanding after training [53]. One such key problem is to understand if a model performs poorly on certain parts of the data, hereafter also referred to as a *slice*.

The problem is that the overall model performance can fail to reflect that of smaller data slices. Thus, it is important that the performance of a model is analyzed on a more granular level. While a well-known problem [54], current techniques to determine under-performing slices largely rely on domain experts to define important sub-populations (or at least specify a feature dimension to slice by) [55, 56]. Unfortunately, ML practitioners do not necessarily have the domain expertise to know all important under-performing slices in advance, even after spending a significant amount of time exploring the data.

An underlying assumption here is that the dataset is large to the extent that enumerating all possible data slices and validating model performance for each is not practical due to the sheer number of possible slices. Worse yet, simply searching for the most under-performing slices can be misleading because the model performance on smaller slices can be noisy, and without any safeguard, this leads to slices that are too small for meaningful impact on the model quality or that are false discoveries (i.e., non-problematic slices appearing as problematic). Ideally, we want to identify the largest and true problematic slices from the smaller slices that are not fully reflected on by the overall model performance metric.

There are more generic clustering-based algorithms in model understanding [57–59] that group

Slice	Log Loss	Size	Effect Size
All	0.35	30k	n/a
Sex = Male	0.41	20k	0.47
Sex = Female	0.21	10k	-0.47
Workclass = Local-gov Race = White	0.43	1.7k	0.19
Education = HS-grad	0.32	9.8k	-0.09
Education = Bachelors	0.44	0.5k	0.27
Education = Masters	0.49	1.6k	0.40
Education = Doctorate	0.47	5k	0.32

Table 1.2: UCI Census data slices. The overall metrics may be considered acceptable, since the overall log loss is low for the entire data (see the "All" row). However, the individual slices tell a different story. When slicing data by gender, the model is more accurate for Female than Male

similar examples together as clusters and analyze model behavior locally within each cluster. Similarly, we can cluster examples by their similarities and check model performances across the clusters. However, clusters of similar examples can still have high variance and high cardinality of feature values, which are hard to summarize and interpret. In comparison, a data slice with a few common feature values (e.g., Female slice contains all examples with Sex = Female) is much easier to interpret. In practice, validating and reporting model performance on interpretable slices are much more useful than validating on arbitrary (non-interpretable) slices (e.g., a cluster of similar examples with mixed properties).

1.4.2 An Illustrative Example

Consider a Random Forest classifier that predicts whether a person's income is above or below $50,000 (UCI Census data [60]). Looking at Table 1.2, the overall metrics may be considered acceptable, since the overall log loss (a widely-used loss metric for binary classification problem) is low for all the data (see the "All" row). However, the individual slices tell a different story. When slicing data by gender, the model is more accurate for Female than Male (the *effect size* defined in Section 2.1 captures this relation by measuring the normalized loss metric difference between the Male slice and its counterpart, the Female slice). The Local-gov White slice is interesting because the average loss metric is on par with Male, but the effect size is much smaller (by convention, $d \leq 0.3$ is small). A small effect size means that the loss metric on Local-gov White is similar to the loss metric on other demographics (defined as *counterparts* in Section 4.1.2).

Once again, the main challenge is the scalability of the search. It is not feasible to evaluate every possible data subset defined by different feature-value combinations.

1.4.3 Goal and Approach

A good technique to detect problematic slices for model validation thus needs to find easy-to-understand subsets of data and ensure that the model performance on the subsets is meaningful and

not attributed to chance. Each problematic slice should be immediately understandable to a human without the guesswork. The problematic slices should also be large enough so that their impact on the overall model quality is non-negligible. Since the model may have a high variance in its prediction quality, we also need to be careful not to choose slices that are false discoveries. Finally, since the slices have an exponentially large search space, it is infeasible to manually go though each slice. Instead, we would like to guide the user to a handful of slices that satisfy the conditions above. In this paper we propose Slice Finder, which efficiently discovers large possibly-overlapping slices that are both interpretable and actually problematic.

A slice is defined as a conjunction of feature-value pairs where having fewer features is considered more interpretable. A problematic slice is identified based on testing of a significant difference of model performance metrics (e.g., loss function) of the slice and its counterpart. That is, we treat each problematic slice as a hypothesis and perform a principled hypothesis testing to check if it is a true problematic slice and not a false discovery by chance. We discuss the details in Section 4.1.2. One problem with performing many statistical tests (due to a large number of candidate slices) is an increased number of false positives. This is what is also known as Multiple Comparisons Problem (MCP) [61]: imagine a test of Type-I error (false positive: recommending a non-problematic slice as problematic) rate of 0.05 (a common α-level for statistical significance testing); the probability of having any false positives blows up exponentially with the number of comparisons (e.g., $1 - (1 - 0.05)^8 = 0.34$, even for just 8 tests, but then, we may end up exploring hundreds and thousands of slices even for a modest number of examples). We address this issue in Section 4.2.2.

In addition to testing, the slices found by Slice Finder can be used to evaluate model fairness or in applications such as fraud detection, business analytics, and anomaly detection, to name a few. While there are many definitions for fairness, a common one is that a model performs poorly (e.g., lower accuracy) on certain sensitive features (which define the slices), but not on others. Fraud detection also involves identifying classes of activities where a model is not performing as well as it previously did. For example, some fraudsters may have gamed the system with unauthorized transactions. In business analytics, finding the most promising marketing cohorts can be viewed as a data slicing problem. Although Slice Finder evaluates each slice based on its losses on a model, we can also generalize the data slicing problem where we assume a general scoring function to assess the significance of a slice. For example, data validation is the process of identifying training or validation examples that contain errors (e.g., values are out of range, features are missing, and so on). By scoring each slice based on the number or type of errors it contains, it is possible to summarize the data errors through a few interpretable slices rather than showing users an exhaustive list of all erroneous examples.

The high-level contribution of this work is applying data mining algorithms to an important ML problem. In summary, we make the following contributions:

- We define the data slicing problem and the use of hypothesis testing for problematic slice identification (Section 4.1.2) and false discovery control (Section 4.2.2).
- We describe the Slice Finder system and propose three automated data slicing approaches,

including a naïve clustering-based approach as a baseline for automated data slicing (Section 4.2).

- We present model fairness as a representative use case for Slice Finder (Section 4.3).
- We evaluate the three automated data slicing approaches using real datasets (Section 4.4).

1.5 Thesis Organization

The outline of this thesis will be as follows. Chapter 2 presents the case of unknown missing data items in more detail and its impact on aggregate query results. The challenges of having missing unknown data items are elaborated and novel techniques to quantify, and thus, correct for the impact of unknown unknowns are proposed. Chapter 3 focuses on uncertainty as undetected data errors. Detecting remaining errors is a critical in both data cleaning and data analysis in general. In this work, a novel data quality metric based on the estimated number of undetected data errors is proposed. The next two chapters discuss the uncertainty challenges related to model quality. Chapter 4 presents an automated data slicing tool for model validation. Chapters 2, 3 and 4 are self-contained with relevant experimental results, related work and summary for the specific topic. Finally, Chapter 5 concludes with some interesting ideas for future work.

Chapter 2

Uncertainty as Missing Unknown Data Items

It is common practice for data scientists to acquire and integrate disparate data sources to achieve higher quality results. But even with a perfectly cleaned and merged data set, two fundamental questions remain: (1) is the integrated data set complete and (2) what is the impact of any unknown (i.e., unobserved) data on query results?

In this chapter, we formalize the problem and present techniques to estimate the impact of the unknown data (a.k.a., unknown unknowns) on simple aggregate queries. The key idea is that the overlap between different data sources enables us to estimate the number and values of the missing data items. Our main techniques are parameter-free and do not assume prior knowledge about the distribution; we also propose a parametric model that can be used instead when the data sources are imbalanced. Through a series of experiments, we show that estimating the impact of unknown unknowns is invaluable to better assess the results of aggregate queries over integrated data sources.

2.1 The Impact of Unknown Unknowns

In this Section, we define unknown unknowns, explain how data integration over multiple sources can be regarded as a sampling process and formally define our estimation goal. For convenience Appendix A.1 contains a symbol-table.

For the purpose of this work (Chapter 2), we treat data cleaning (e.g., entity resolution, data fusion, etc.) as an orthogonal problem. Any data cleaning techniques can be applied to our problem without altering the problem context [30, 62–66]. While data quality can influence the estimation quality, studying it goes beyond the scope of this paper [34]. We assume that after a proper data cleaning process we have one instance per observed entity and know exactly how many times the entity was observed across multiple data sources.

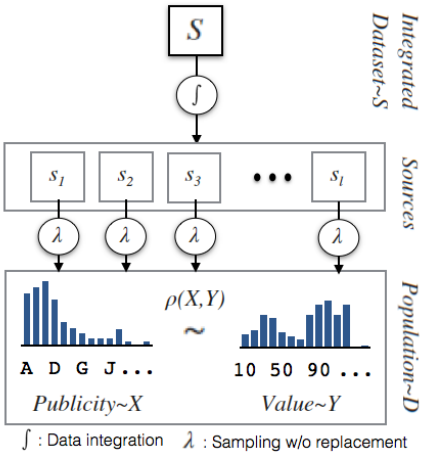


Figure 2.1: A sampling process for the integrated database.

2.1.1 Unknown Unknowns

We assume that queries are of the form `SELECT AGGREGATE(attr) FROM table WHERE predicate`, that *table* only contains records about a single entity class (e.g., companies) and that a record in *table* corresponds to exactly one real-world entity (e.g., IBM). Thus, in the remainder of the paper we use record, entity and data item interchangeably.

Definition 2.1. (Unknown Unknowns) Let Ω be the universe of unknown size of all valid unique entities r for a given entity class, and let $attr_A(r)$ be the value of attribute A of r. Then the ground truth $D \subseteq \Omega$ is defined as a set of entities that satisfy the predicate, i.e., $D = \{r \in \Omega \mid predicate(r)\}$, where its size $N = |D|$ is not known. Let S be a sample with replacement from D and c be the number of unique entities in S. Unknown unknowns U refers to any unobserved entity r that exists in D but not in S: $U = D - S$ with size $N - c$.

For our running example, Ω would be the universe of all companies in the world, D all tech companies in the US and $\sum_{r \in D} attr_{empl}(r)$ be the true number of U.S. tech sector employees. S would be a sample with duplicates and unknown unknowns would be every company which is not in S.

What we aim to achieve is a good estimate of the ground truth: `SELECT AGGREGATE(attr) FROM D`, when we only have S. Note, that we drop the *predicate* from the query, since every item in D already has to fulfill the *predicate*. In this work, we assume that we neither know all entities in D nor its size (i.e., *open world* assumption). This distinguishes our problem from the problem of *missing data* [2, 31, 67], which refers to incomplete data or missing attribute values.

2.1.2 Data Integration As Sampling Process

Data integration refers to the process of combining different data sources under a common schema [62]. For the purpose of this work, we assume that data sources are independent samples (e.g., data

sources are not copies from each other and instead are independently created), and we model the data integration process as a multi-stage sampling process as shown in Figure 2.1.

We assume l data sources $s_1 ... s_l$, each sampling $n_j = |s_j|$ data items from the ground truth D (e.g., the complete set of tech companies in the US with their respective number of employees), **without replacement**, since a data source typically only mentions a data item once. We also assume that every data item $d_i \in D$ has a publicity likelihood p_i of being sampled, following some distribution X. Likewise, the attribute values (e.g., the number of employees) have a certain likelihood to appear in the ground truth, referred to as *value likelihood*, again following some distribution Y. These two distributions are possibly correlated (i.e,. the publicity-value correlation ρ is bigger or smaller than 0: $\rho \neq 0$). For instance, more popular items are likely to have more frequent values; if there is no correlation, more popular items could have less frequent or any values. Note that each item appears only once in D, but multiple times in S depending on its publicity; value frequency in D can be uncorrelated to the publicity distribution.

The l data sources are then integrated into a single integrated data set S of size $n_S = \sum_{j=1}^{l} n_j$. Although each source samples without replacement from $N = |D|$ different classes (i.e., unique data item), S contains duplicates because every data source is sampling from the same underlying truth D. This overlap between the data sources is important as it allows to estimate the frequency of the unique items in S; a key requirement to be able to use species estimation techniques. Furthermore, the larger the value of l and the smaller the size n_j of every data source, the better S approximates a sample with replacement, and the more accurate the resulting frequency estimate. We analyze the effects of smaller l in Section 2.2.4 and 2.5.

Whereas S contains duplicates, the end-user only sees a view of S, referred to as the integrated database K (for *Known* data), which contains only one entity per unique entity. Lastly, it is interesting to observe that S is not necessarily a simple random sample, in that some data items are more likely to appear in S due to the publicity-value correlation. The techniques proposed in Section 2.2 account for this and we illustrate how different techniques perform in the face of such a non-random sample in Section 2.5.2.

This data integration model covers a large class of use cases from web integration to crowdsourcing. In the latter case, each crowd worker can be regarded as a separate data source s_j since it is known that workers also sample without replacement from D [34]. While extremely powerful, there are scenarios where this sampling model does not apply. Most importantly, data sources are not always independent [68]. Furthermore, the number of data sources l has to be large enough to have sufficient overlap between the sources (see Section 2.5). If any of these assumptions are violated, then only low-quality estimations are possible.

2.1.3 Problem Statement

We are interested in estimating the impact of unknown unknowns (U) to adjust aggregate query results.

Definition 2.2. (The Impact of Unknown Unknowns) Given an integrated database K, the impact of unknown unknowns is defined as the difference between the current answer ϕ_K of the aggregate query over the database K and the answer over the ground-truth ϕ_D:

$$\Delta = \phi_D - \phi_K \tag{2.1}$$

Our goal is to estimate the answer on the ground-truth by estimating Δ based on S:

$$\hat{\phi}_D = \phi_K + \hat{\Delta}(S) \tag{2.2}$$

Note that this definition works for all common aggregates including *MIN* and *MAX*, where $\hat{\Delta}$ would be the positive or negative adjustment to the observed *MIN*/*MAX* value.

2.2 Sum Query

In this section, we focus on *SUM*-aggregates to illustrate our estimation techniques. We first formalize the naïve estimator (Section 2.2.1) and highlight its drawbacks. We then develop the *frequency* estimator by making *naïve* estimator more robust to the *publicity-value correlation* (Section 2.2.2). Afterwards, we describe the more sophisticated *bucket* estimator (Section 2.2.3). Finally, we develop a *Monte-Carlo* estimator which is better suited for a smaller set of data sources (Section 2.2.4).

2.2.1 Naïve Estimator

Estimating the impact of unknown unknowns for SUM queries is equivalent to solving two subproblems: (1) estimating how many unique data items are missing (i.e., the unknown unknowns count estimate), and (2) estimating the attribute values of the missing data items (i.e., the unknown unknowns value estimate). The *naïve* estimator uses the $Chao92$ [69] species estimation technique to estimate the number of the missing data items, and *mean substitution* [2] to estimate the values of them.

Let $\phi_K = \sum_{r \in K} attr(r)$ be the current sum over the integrated database, then we can more formally define our *naïve* estimator for the impact of unknown unknowns as:

$$\Delta_{naive} = \underbrace{\frac{\phi_K}{c}}_{\text{Value estimate}} \cdot \underbrace{(\hat{N} - c)}_{\text{Count estimate}} \tag{2.3}$$

$\hat{N}$ is the estimate of the number of unique data items in the ground truth D, and c is the number of unique entities in our integrated database K (thus, $\hat{N} - c$ is our estimate of the number of the unknown data items). ϕ_K/c is the average attribute value of all unique entities in our database K.

Chao92 estimator

Throughout the paper, we use the popular $Chao92$ estimator. Many species estimation techniques exist [70, 71], but we choose $Chao92$ since it is more robust to a skewed publicity distribution (we

illustrate this further in Section 2.5.10). The *Chao92* estimator uses *sample coverage* to predict $\hat{N}$. The sample coverage C is defined as the sum of the probabilities p_i of the observed classes. Since the true distribution $p_1 ... p_N$ is unknown, we estimate C using the Good-Turing estimator [72]:

$$\hat{C} = 1 - f_1/n \tag{2.4}$$

The f-statistics, e.g., f_1, represent the frequencies of observed data items in the sample, where f_j is the number of data items with exactly j occurrences in the sample. f_1 is referred as *singletons*, f_2 *doubletons*, and f_0 as the *missing data* [73]. Sample coverage measures the ratio between the number of singletons (f_1) and the sample size (n). This ratio changes with the amount of duplicates in the sample. The high-level idea is that the more duplicates that exist in our sample S compared to the number of singletons f_1, the more complete the sample is (i.e., higher sample coverage).

In addition, the *Chao92* estimator explicitly incorporates the skewness of the underlying distribution using *coefficient of variation* (CV) γ, a metric that is used to describe the dispersion in a probability distribution [69]. A higher CV indicates a higher variability among the p_i values, while a $CV = 0$ indicates that each item is equally likely (i.e., the items follow a uniform distribution).

Given the publicity distribution $(p_1 \cdots p_N)$ that describes the probability of the i-th class being sampled from D, with mean $\bar{p} = \sum_i p_i/N = 1/N$, CV can be expressed as follows:

$$\gamma = \left[\sum_i (p_i - \bar{p})^2 / N \right]^{1/2} / \bar{p} \tag{2.5}$$

However, since p_i is not available for all data items, CV has to be estimated using the f-statistic:

$$\hat{\gamma}^2 = \max \left\{ \frac{\frac{c}{\hat{C}} \sum_i i(i-1) f_i}{n(n-1)} - 1 \, , \, 0 \right\} \tag{2.6}$$

There is a bias-corrected estimator for CV, which works better than $\hat{\gamma}$ when the true CV is relatively large; however, $\hat{\gamma}$ is superior when CV is moderate [69]. For this reason, we deem $\hat{\gamma}$ a better fit for real-world use cases, especially, when the true CV is not known in advance. The final *Chao92* estimator for $\hat{N}_{Chao92}$ can then be formalated as:

$$\hat{N}_{Chao92} = \frac{c}{\hat{C}} + \frac{n(1-\hat{C})}{\hat{C}} \cdot \hat{\gamma}^2 \tag{2.7}$$

The Estimator

$\hat{N}_{Chao92}$ is our estimate for N, and comparing this to c provides us with a means of evaluating the completeness of S. By substituting $\hat{N}_{Chao92}$ for $\hat{N}$, the final *naìve* estimator can be written as:

$$\Delta_{naive} = \frac{\phi_K}{c} \cdot (\hat{N}_{Chao92} - c) = \frac{\phi_K \cdot f_1 \cdot \left(c + \hat{\gamma}^2 n\right)}{c \cdot (n - f_1)} \tag{2.8}$$

Note, that the *naìve* estimator does not consider any *publicity-value correlation* and thus tends to over-estimate (or under-estimate) the ground truth.

Limitations

The naìve approach has a couple of drawbacks. First, species estimation has very strict requirements on how data is collected. Almost every data integration scenario violates these requirements, causing the estimator to significantly over/underestimate the number of missing data items.

Second, it ignores the fact that the attribute values of the missing items may be correlated to the likelihood of observing certain data items. For example, large tech companies like Google with many employees are often more well known and thus, appear more often in data sources than smaller start-ups, creating a biased data set. This is problematic since it also biases the mean and with it the estimate.

In the statistics literature, this second problem is referred to as *Missing Not At Random* (MNAR) [2, 31], where the missingness of a data item depends on its value. There are many statistical inference techniques dealing with MNAR [30, 32, 74–76], but nearly all the techniques require at least partial knowledge of the record. For example, in the case of surveys, people with a high salary might be more reluctant to report their salary but have no problem stating their home address or how many children they have. Existing MNAR techniques use the reported values (e.g., the address) to infer the missing attributes. Unfortunately, this is not possible in the case of unknown unknowns, as we miss the entire record.

2.2.2 Frequency Estimator

We developed a simple variation of the *naìve* estimator, which makes direct use of the frequency statistics to improve estimation quality. All coverage-based species estimation methods give special attention to the singletons f_1; the data items observed exactly once. The idea is that those rare items, in relation to the sample size n, give a clue about how well the complete population is covered (i.e., f_1/n is a proxy to the number of unknown unknowns, see Equation 2.4). A ratio of f_1/n close to 1 means that almost every sample is unique, indicating that many items might still be missing. Conversely, a ratio close to 0 indicates all unique values have been observed several times, decreasing the likelihood of any unknown data. We use a similar reasoning to improve our value estimation. The key idea is that singletons are the best indicator of missing data items, and that their average value might be a better representation of the values of the missing items. Let ϕ_{f_1} be the sum of all singletons, $\sum_{r \in singletons} attr(r)$ and $\hat{N}_{Chao92}$ again be the $Chao92$ count estimate. Then the estimator can be defined as:

$$\Delta_{freq} = \frac{\phi_{f_1}}{f_1} \cdot (\hat{N}_{Chao92} - c) = \frac{\phi_{f_1}\left(c + \hat{\gamma}^2 n\right)}{n - f_1} \tag{2.9}$$

While this estimator still does not directly consider the publicity-value correlation, it is more robust against popular high-impact data items (i.e., data items with extreme *attribute* values). For example, in our running employee example, big companies that are highly visible like Google or IBM can significantly impact the known value estimate ϕ_K/c. However, through using the average value of the singletons, ϕ_{f_1}/f_1, it is reasonable to assume that those companies will not stay as singletons

very long in any sample and thus will not impact the average value for the unknown unknowns. This estimator is surprisingly simple and becomes even simpler if we assume $\hat{\gamma}^2 = 0$:

$$\Delta_{freq} = \frac{\phi_{f_1} \cdot c}{n - f_1} \tag{2.10}$$

Note, that $\hat{\gamma}^2 = 0$ makes it a Good-Turing estimate, which also converges to the ground truth even for skewed publicity values; it might just take a bit longer [69]. While Δ_{freq} is not the best estimator (see Section 2.5) the simplicity makes it still useful to quickly test if an aggregate query result might be impacted by any unknown unknowns.

2.2.3 Bucket Estimator

The problem with the previous two estimators is that they do not directly consider a correlation between publicity and attribute values. We designed the *bucket* as a first estimator designed for unknown unknowns with publicity-value correlation. The idea of the estimator is to divide the attribute value range into smaller sub-ranges called buckets, and treat each bucket as a separate data set. We can then estimate the *impact of unknown unknowns* per bucket (e.g., large, medium, or small companies) and aggregate them to the overall effect:

$$\Delta_{bucket} = \sum_i \Delta(b_i) \tag{2.11}$$

Here $\Delta(b_i)$ refers to the estimate per bucket and both the *frequency* or *naïve* estimator could be used. Using buckets has two effects: first, it provides more detailed estimates of what types of companies are missing; second, it decreases the variance of item values per bucket, making the estimate less prone to outliers (e.g., items with extreme low and high values can be "contained" in separate buckets).

The challenge with the *bucket* estimator is to determine the right size for each bucket. If the bucket size is too small, the bucket contains almost no data items. In an extreme case of having a single data item per bucket, no count or proper value estimation is possible. If the bucket size is too big, then the publicity-value correlation can still bias the estimate. In fact, the case with a single bucket is equivalent to using just the *naïve* or *frequency* estimator. In the following we describe two bucketing strategies.

Static Bucket

An easy way to define buckets is to divide the observed value range into a fixed n_b number of buckets of size w_i:

$$w_i = \frac{(a_{max} - a_{min})}{n_b} \tag{2.12}$$

where a_{min} (a_{max}) refers to the min (max) observed attribute value. Afterwards we apply Δ_{naive} per bucket. It is important to note that the estimate goes to infinity with buckets which only contain singletons due to division-by-zero ($n - f_1 = 0$, see equation 2.8), which can significantly increase the error of the estimate for very small buckets.

The optimal number of buckets varies depending on the underlying publicity distribution (see Appendix A.2); unfortunately, the true publicity distribution is not known and we cannot predetermine the right number (or size) of static buckets. To this end, we found that the estimation approach using static buckets is of little practical value.

Dynamic Bucket

To overcome the previously mentioned issues, we developed several alternative statistical approaches to determine the optimal bucket boundaries over time. The most notable are our uses of the error estimate/upper bounds from Section 2.3 and of treating f_1 as a random variable (see also Section 2.2.5). Surprisingly, we achieve the best performance across all our real-world use cases and simulations using a rather simple conservative approach, referred to as $\Delta_{Dynamic}$.

The core idea behind our dynamic strategy $\Delta_{Dynamic}$ is to sort the *attribute* values of S and then recursively split the range into smaller buckets only if it reduces the estimated impact of unknown unknowns, i.e., the absolute Δ value. Intuitively, this is controversial since either under- or overestimation could be better for different use cases. However, there is a more fundamental reason behind this strategy.

The Foundation: Whenever we split a data set into buckets, each bucket contains less data than before the split, and the chance of an estimation error increases. To illustrate this, we consider the simplest case of a uniform publicity distribution ($\hat{\gamma} = 0$) and an even bucket split. In this case, the $Chao92$ estimate for $\hat{N}$ is greater than or equal to the estimate before the split:

$$\begin{aligned} \hat{N}_{Chao92} = \frac{c}{1 - f_1/n} &= \overbrace{\frac{n \cdot c}{n - f_1}}^{\text{Before split}} \\ &\leq \underbrace{\frac{n_{b1} \cdot c_{b1}}{n_{b1} - f_{1_{b1}}} + \frac{n_{b2} \cdot c_{b2}}{n_{b2} - f_{1_{b2}}}}_{\text{After split}} \end{aligned} \tag{2.13}$$

When we split the data exactly into halves, it follows that $c_{b1} = c_{b2} = c/2$ (i.e., we split in regard to the unique values). With a uniform publicity distribution, every item is equally likely, and therefore we can assume that both buckets contain roughly the same amount of data after the split: $n_{b1} = n_{b2} \approx n/2$. However, in contrast to n and c, the number of singletons (f_1) can vary significantly between the buckets. In fact, we know that the estimators only stabilize if every item was observed several times [69] and as a consequence n has to be significantly larger than c and c significantly larger than f_1 ($n \gg c \gg f_1$); if we split, there is a higher chance that we unevenly distribute the f_1 among the buckets.

To model the uneven distribution of f_1 we introduce another parameter $\alpha \in [0, 1]$ and set $f_{1_{b1}} = \alpha \cdot f_1$ and $f_{1_{b2}} = (1 - \alpha) \cdot f_1$. As a result the inequality in equation 2.13 becomes:

$$\underbrace{\frac{n \cdot c}{n - f_1}}_{\text{Before split}} \leq \underbrace{\frac{\frac{n}{2} \cdot \frac{c}{2}}{\frac{n}{2} - \alpha \cdot f_1} + \frac{\frac{n}{2} \cdot \frac{c}{2}}{\frac{n}{2} - (1 - \alpha) \cdot f_1}}_{\text{After split}} \tag{2.14}$$

ALGORITHM 1: Dynamic bucket generation

Input : S
Output: List of buckets

```
1  b_0 = (minValue(S), maxValue(S)) ;                          /* init bucket b_0 */
2  todo = [b_0];                                               /* list with b_0 */
3  δ_min = abs(Δ(b_0)) ;                                       /* Δ estimate over b_0 */
4  bkts = [] ;                                                 /* final bucket list */
5  while !todo.empty do
6  |  b = todo.pop ;                                          /* pop first bucket */
7  |  δ_tmp = δ_min − abs(Δ(b));
8  |  tmp = (null, null) ;                                    /* empty pair */
9  |  for unique r ∈ b do
10 |  |  (t_1, t_2) = split(b, r.value) ;                     /* split b by r.value */
11 |  |  if δ_min > δ_tmp + abs(Δ(t_1)) + abs(Δ(t_2)) then
12 |  |  |  δ_min = δ_tmp + abs(Δ(t_1)) + abs(Δ(t_2)) ;
13 |  |  |  tmp = (t_1, t_2);
14 |  |  end
15 |  end
16 |  if tmp ≠ (null, null) then
17 |  |  todo.add(t_1, t_2) ;                                 /* replace b by t_1, t_2 to minimize δ_min */
18 |  else
19 |  |  bkts.add(b) ;                                        /* put b back:  don't split */
20 |  end
21 end
22 return bkts;
```

Appendix A.3 shows that the right hand side of the above inequality has its global minimum at $\alpha = 0.5$, which evaluates to $nc/(n - f_1)$ ($\hat{N}$ before split), and that the inequality always holds. Thus, it can be seen that splitting a data set into buckets not only potentially increases the error, but it does so in a monotonic way.

Yet, this does not mean that the *sum* estimate Δ always increases as well. Especially with a publicity-value correlation, the overall estimate of Δ over all buckets can still decrease as the average *attribute* values per bucket differ. This is in line with our original motivation to use buckets, as we wanted to get a more detailed unknown unknowns estimate (e.g., how many small companies vs. large companies are missing). Bringing these two observations together, we can assume for many real-world use cases that whenever our estimate of the impact of unknown unknowns Δ increases after a split, it has a significant chance of being caused by the increasing error in $\hat{N}$, whereas when it decreases it potentially improves the estimate due to the more detailed unknown estimate. While it does not always have to be the case (e.g., if the publicity-value correlation is negative) it is still an indicator for many real-world use cases (see Section 2.5). Based on the observations, we have devised the conservative bucket splitting strategy: only split a bucket if it reduces the overall estimate for Δ.

The Algorithm: Algorithm 1 shows the final algorithm. First we add a bucket which covers the complete value range of S to the *todo* list (line 2) and calculate the current Δ over S (line 3). Note that we take the absolute values of all estimates (Δ) to underestimate the impact of unknown

unknowns even for the case of having negative attribute values (e.g., net losses of companies). Afterwards, we check recursively if we can split the bucket to reduce Δ until no further "underestimation" is possible (line 5-21).

We therefore remove the first bucket from the *todo* list (line 6) and calculate the Δ over S without the impact of this bucket b (line 7). Note, that during the first iteration δ_{tmp} will be 0. Afterwards, for every unique record in b, we split the current bucket b into two temporary buckets t_1 and t_2 based on the record's attribute value (line 10). If the resulting estimate using this split is smaller than any previously observed minimums (line 11), we set the new minimum to this value (line 12) and store the new buckets (line 13). When the for-loop of line 9-15 finishes and if at least one new bucket was found (line 16), *tmp* will contain the new split point, which reduces δ for the bucket, and δ_{min} the new minimum value of δ. These buckets are then added to the *todo* list (line 17) to check, if splitting them again would further lower the estimate. On the other hand, if *tmp* is empty, the algorithm wasn't able to further split the bucket and the current bucket without any additional splits is added to the final bucket list (line 19). If no buckets are left in the *todo* list, the algorithm terminates and *bkts* contains the final list of buckets.

Discussion: The idea behind the *Bucket* estimator is to only split a bucket if it reduces the estimated value. The reasoning behind it is two fold: on one hand, splitting increases the estimation error and with it the estimate as explained above. On the other hand, many use cases have a positive publicity-value correlation. For example, the more known companies are also the bigger ones, the less known (missing) are smaller. In that case, a more pessimistic estimate is often more desired as explained above.

However, it is also possible that there might be no or even a negative correlation, contradicting our previous assumption. Interestingly, our bucket estimator in these cases should still be able to provide an accurate estimate. The reason is, that — in expectation — the *Bucket* will not split any buckets and act similar to the baseline estimator (e.g., naìve estimator). This can be explained by the fact that the value and count estimate with no or a negative correlation will — in expectation — result in an higher estimate compared to before the split, and thus, prevent the split in the first place. Yet, it is conceivable that from time to time, and by chance, a split might happen, which reduces the estimate (e.g., an uneven distribution of values). We will study this effect in more detail in Section 2.5.4.

2.2.4 Monte-Carlo Estimator

As our experiments show, the previous estimator actually performs very well (see Section 2.5). However, what it does not consider is the effect of uneven contributions from data sources (i.e., one data source contains much more data than another) and the peculiarities of the sampling process itself. The *Chao*92 species estimation, like almost all other estimators, assumes sampling *with* replacement, whereas our data sources sample *without* replacement from the underlying ground truth. The reason why the *Chao*92 still works is that with a reasonably high number of data sources, the integrated data source S approximates a sample with replacement [34]. However, with

either a small number of data sources or uneven contributions from sources (i.e., some sources are significantly bigger than others), S diverges significantly from a sample done with replacement, resulting in significant over- or under-estimation. In the case of crowd-sourcing, workers who provide significantly more data items than other workers are referred to as *streakers* [34].

To address these issues, we present a *Monte Carlo*-based (MC) estimator for $\hat{N}$. The idea is that we simulate the sampling process to find the most likely distribution with population size N, which best explains the observed sample including how many items s_j every data source j contributes. More formally, given $(s_1, ..., s_l)$ what we seek is a set of parameters Θ (e.g., the distribution parameters) for the MC simulation, which minimize some distance function Γ between the observed data S and the simulated data Q_Θ:

$$\underset{\Theta}{\operatorname{argmin}}\, \Gamma(S, Q_\Theta | l, [s_1, ..., s_l]) \tag{2.15}$$

In the following we first describe the MC method for generating Q_Θ given a distance function Γ and a search strategy to find the optimal parameter Θ [77].

Monte-Carlo Method

In contrast to the other estimators, the *Monte-Carlo* estimator requires an assumption about the shape of the underlying publicity distribution; in this work, we use an exponential distribution for publicity, from which data source j samples n_j data items. Accordingly, the parameter Θ has two components: θ_N specifies the assumed number of data items, and θ_λ governs the shape (skew) of publicity distribution. Note, that the assumption of the exponential distributions makes the MC method a parametric model. For future work, we could try a more general parametric model that captures publicity-value correlation in any distributional shapes. The goal of the MC simulation is to determine how well θ_N and θ_λ help to explain the observed S.

Algorithm 2 outlines our MC algorithm. First, we use a scaled exponential distribution with skew θ_λ to sample publicity $(p_1 \cdots p_{\hat{N}})$ for $\theta_{\hat{N}}$ items (line 1); the exponential distribution has normalized and scaled publicity probabilities of at least 0.1 for $\hat{N}$ unique items (support size). And then we initialize the distance to 0 (line 2). Afterwards, we repeat the following procedure $nbRuns = 5000$ times. For every data source (line 5) we sample n_j data items according to E, but also without replacement (line 6). The sampled items are added to Q to form a histogram (line 7) for the particular run. After simulating l sources, Q contains the simulated version of S.

To finally compare the simulated sample Q with the observed sample S, we make use of the discrete KL-divergence metric [78]. However, this requires transforming S and Q into a frequency statistic and indexing them to ensure that the right items are compared with each other (line 9).

After the indexing we have two comparable frequency statistics for S and the simulation: F_S and F_Q. However, S might contain less than $\hat{N}$ unique data items, for which the KL-divergence is not defined. We therefore adjust F_S and assign a small non-zero probability to the missing extra unique items (line 10). Finally, the two frequency statistics can be compared using the standard KL-Divergence metric and added to the total distance (line 11). After all the simulation runs the average distance is returned (line 13).

ALGORITHM 2: Monte Carlo method

Input : $\theta_{\hat{N}}$, θ_λ, S, $[n_1, ..., n_l]$, $nbRuns$
Output: Average distance

```
1  E = dist(θ_N̂, θ_λ);                                   /* publicity of N̂ items */
2  Γ = 0.0;                                              /* default value */
3  for i = 1 to nbRuns do
4  |   Q = [];                                           /* simulated model */
5  |   for j = 1 to l do
6  |   |   s_i = sample(n_j, E);                        /* w/o repl */
7  |   |   Q.add(s_i);
8  |   end
9  |   (F_S, F_Q) = indexing(S, Q);
10 |   F'_S = smooth(F_S, F_Q);
11 |   Γ += klDiv(F'_S, F_Q);                            /* KL-divergence */
12 end
13 return Γ/nbRuns;
```

Search Strategy

We can now simulate the observed sampling process leading to S, but we still need a way to find the optimal Θ, which best explains the observed sample S. The difficulty is that even though the KL-divergence cost function is convex, the integer variable $\hat{N}$ prevents us from using tractable optimization algorithms (e.g., gradient descent). Furthermore, the distance function can be quite sensitive to small amounts of noise in D.

We therefore make the estimator more robust by first performing a grid search for Θ (line 5-10). We vary θ_N between $c \leq \hat{N} \leq \hat{N}_{Chao92}$ with a step-size $(\hat{N}_{Chao92} - c)/10$ and θ_λ between $-0.4 \leq \lambda \leq 0.4$ (i.e., almost no to heavy skew) with a step-size 0.1 (line 2 and 3). The step sizes are chosen to be small enough to efficiently model the convex curve, but large enough to be robust to any noise. Afterwards, we fit a two-dimensional curve using least-squares curve fitting (line 11) and return the $\hat{N}_{MC}$ with the minimum D_{KL} on the fitted curve as the final count estimate (line 11). As future work, we want to investigate more efficient search strategies, like discrete stochastic optimization algorithms [79].

Finally, to estimate the total difference, we use our *naìve* estimation technique with $\hat{N}_{MC}$. The estimate is more robust and over-estimates less than the original *naìve* estimator as our MC method always penalizes any unmatched unique items in Q. In other words, the MC estimator favors solutions where $\hat{N}$ is closer to the number of observed unique items c.

2.2.5 Other Estimators

During the course of developing the above estimators, we explored various alternatives. For example, we experimented with alternative static bucket strategies (see also Appendix A.2). Furthermore, we noticed that many proposed techniques can be combined. For instance, we can use the *frequency* estimator, instead of the *naìve* estimator, with the *bucket* (i.e., *Dynamic Bucket* approach) estimator

ALGORITHM 3: Monte-Carlo based $\hat{N}$ estimation

Input : $[s_1, ...s_l]$,c,$N_{\hat{Chao92}}$,$nbRuns$
Output: Estimated number of unique data items, $\hat{N}$

$D_{KL} = []$; /* KL-divergence */
$n = sizes([s_1, ..., s_l])$; /* $[n_1, ..., n_l]$ */
$\Theta_{\hat{N}} = [c : \frac{(\hat{N}_{Chao92} - c)}{10} : N_{Chao92}]$;
$\Theta_{\lambda} = [-0.4 : 0.1 : 0.4]$;
for $\theta_{\hat{N}} \in \Theta_{\hat{N}}$ **do**
  for $\theta_{\hat{N}} \in \Theta_{\lambda}$ **do**
    $\Gamma = monteCarlo(\theta_{\hat{N}}, \theta_{\lambda}, n, n_r)$; /* Alg 2 */
    $D_{KL}.add(\Gamma)$;
  end
end
$p = curveFit(\Theta_{\hat{N}}, \Theta_{\lambda}, D_{KL}, 2)$; /* 2-D curve fit */
$[\hat{N}, \lambda] = \underset{\lambda \in [-0.4, 0.4], \hat{N} \in [c, \hat{N_{Chao92}}]}{\arg\min}\{p(\hat{N}, \lambda)\}$; /* min on the curve */
return $\hat{N}$;

or the *Monte-Carlo* estimator; furthermore, we can also combine the *Monte-Carlo* estimator with the *bucket* estimator.

However, since the *Monte-Carlo* estimator requires larger sample size for accurate estimates, applying it to a smaller bucket often degrades the estimation quality. Also, our experiences tell us that the difference between the *naìve* and *frequency* estimators does not help much for the *bucket* approach (see Appendix A.4). Therefore, we focus on the original techniques for the evaluation of this work.

2.3 Estimation Upper Bound

Because the foregoing estimators are based on a sample S, they are subject to uncertainty, which should be taken into account. One approach is to supplement the estimators with a probabilistic upper bound Δ such that $P(\phi_D \leq \Delta) = \alpha$, where α is a specified probability. In this section, we provide an approximate estimation upper bound based on the *naìve* estimator (the product of count estimate upper bound and value estimate upper bound). We treat this estimation upper bound as an upper bound on the ground truth sum query result (ϕ_D) because *naìve* estimator tends to highly overestimate the ground truth (empirically shown in Section 2.5); the asymptotic estimate is at least ϕ_D given non-zero publicity probabilities ($p_1 ... p_N$) and positive item values (e.g., number of employees).

The $Chao92$ count estimation is based on *sample coverage* plus a correction for the skew $\hat{\gamma} > 0$. Recent work proposed an error bound of the Good-Turing estimator for the ground truth unknown unknowns distribution mass ($M_0 = \sum_{d_i \in U} p_i$) [35]:

$$M_0 \leq \frac{f_1}{n} + (2\sqrt{2} + \sqrt{3}) \cdot \sqrt{\frac{\log 3/\epsilon}{n}} \tag{2.16}$$

which holds with **probability at least** $1-\epsilon$ over the choice of the sample with $n = |S|$. The confidence parameter ϵ governs the tightness of this bound (we use $\epsilon = 0.01$ for 99% confidence). Based on equation 2.16, we bound $Chao92$:

$$\begin{aligned}\hat{N}_{Chao92} &= \frac{c}{\hat{C}} + \frac{n(1-\hat{C})}{\hat{C}} \cdot \hat{\gamma}^2 \\ &\approx \frac{c}{\hat{C}} = \frac{c}{1-M_0} \\ &\leq \frac{c}{1-(\frac{f_1}{n} + (2\sqrt{2}+\sqrt{3}) \cdot \sqrt{\frac{\log\log 3/\delta}{n}})}\end{aligned} \tag{2.17}$$

Notice, that we omit $\hat{\gamma}$ as we consider approximate upper bound for large n. It is as if we assume non-zero publicity probabilities or that all items are eventually discovered. For future work, we want to relax such asymptotic assumption and incorporate heterogeneous publicity probabilities (i.e., $\hat{\gamma} > 0$) to more tightly bound the count estimate.

To provide an approximate bound on the *mean substitution* ($\frac{\phi_K}{c}$), we assumed that $\frac{\phi_K}{c}$ is normally distributed and $c \to N$ as $|S|$ gets large [9]; the worst case estimate of the ground truth *attribute* mean value ($\frac{\phi_D}{N}$) is then defined with the help of the sample standard deviation (σ_K):

$$\frac{\phi_D}{N} \leq \frac{\phi_K}{c} + z \cdot \sigma_K \tag{2.18}$$

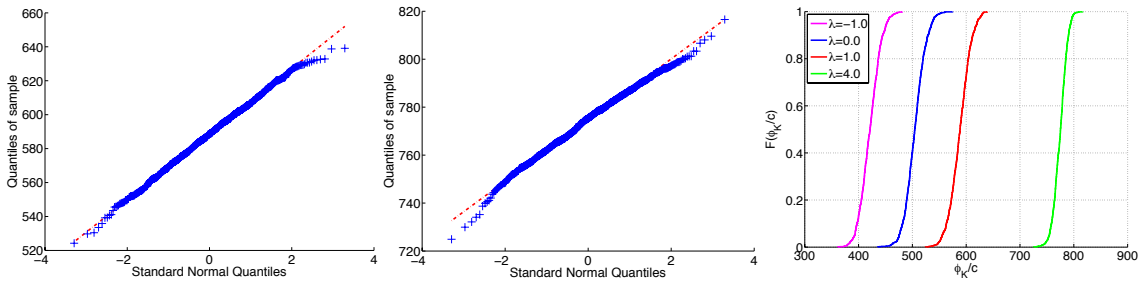
Here z controls the confidence of the bound, and we use $z = 3$ to have nearly all values with 99.87% confidence lie below the upper bound. However, it is highly non-trivial to establish the normality assumption for a number of reasons. First, the standard *Central Limit Theorem* is not applicable because c distinct items in K are not i.i.d. random variables. Even if we assume that c distinct items are sampled independently into K, as K is integrated over a large number of sources independently sampling a small subset of D, each item has different publicity to be sampled. Second, c is bounded above by N which is finite, and thus, large-sample approximation is not possible unless N goes off to ∞. Lastly, there is no guarantee that $c \to N$ as the sample size $|S|$ gets larger, i.e., publicity $(p_1 \cdots p_N)$ could be heavily long-tailed. We therefore empirically justify the normality assumption via Monte-Carlo simulation (Figure 2.2) using the same synthetic dataset from Section 2.5.2. The sample size is 200 and it is integrated over 10 sources; the simulation is repeated 5000 times.

The final approximate upper bound is then the simple multiplication of the two worst case estimators:

$$\Delta_{bound1} = \frac{(\frac{\phi_K}{c} + z \cdot \sigma_K) \cdot c}{1-(\frac{f_1}{n} + (2\sqrt{2}+\sqrt{3}) \cdot \sqrt{\frac{\log\log 3/\delta}{n}})} \tag{2.19}$$

This final bound holds if both the count estimate bound (Equation 2.17) and the value estimate bound (Equation 2.18) hold with high probability. Let $p_{count} = 0.99$ ($\epsilon = 0.01$) denote the probability of the count estimate bound holds true and $p_{value} = 0.9987$ ($z = 3$) the probability of the value estimate bound holds true. Applying Bonferroni's inequality, $P(A \cap B) \geq 1 - P(A^c) - P(B^c)$, the classical approximate confidence is

$$1 - (1 - p_{count}) - (1 - p_{value}) = 0.9887. \tag{2.20}$$



(a) $\lambda = 1.0$ (b) $\lambda = 4.0$ (c) Empirical CDF

Figure 2.2: Monte-Carlo simulation results to empirically justify the normality of the mean substitution (ϕ_K/c): quantile-quantile plots, (a) and (b), show that samples drawn from skewed *publicity* distribution have approximately normally distributed mean values; it can be seen in (c) that positive *publicity-value correlation* shifts the average value distribution (e.g., negative publicity-value correlation shifts the distribution to the left as smaller values are more likely to be sampled), without changing the shape of CDF too much. The uniform *publicity* ($\lambda = 0.0$) samples have mean average value, $F(x = 504.87) = 0.5$, close to the ground truth 505.

2.3.1 Using Confidence Interval of Chao Estimator

Unfortunately, our experiences show that the previous "upper" bound estimate is very loose. To this end, we developed another approximate estimation upper bound based on the confidence interval of $Chao84$, which assumes that occurrence for each species follows a binomial distribution. The variance estimate presented in the original paper of $Chao84$ is then as follows [80]:

$$\hat{\sigma}^2 = f_2 \left[\frac{1}{2} \left(\frac{f_1}{f_2} \right)^2 + \left(\frac{f_1}{f_2} \right)^3 + \frac{1}{4} \left(\frac{f_1}{f_2} \right)^4 \right] \tag{2.21}$$

Assuming the normality of $\hat{N}_{Chao84}$ with variance $\hat{\sigma}^2$, the classical approximate confidence interval for the estimate is $[\hat{N}_{Chao84} - z' \cdot \hat{\sigma}, \hat{N}_{Chao84} + z' \cdot \hat{\sigma}]$. With $z' = 3$, the confidence for the desired one-sided bound $(-\infty, \hat{N}_{Chao84} + z' \cdot \hat{\sigma}]$ is 99.87%. However, in [81] an improved asymmetric confidence interval was presented; assuming the normality of $\log(\hat{N} - c)$ instead of $\hat{N}$ the author shows that the following interval holds true:

$$\left[c + \frac{(\hat{N} - c)}{T}, c + (\hat{N} - c) \cdot T \right] \tag{2.22}$$

with

$$T = exp(z' \cdot \sqrt{log(1 + \hat{\sigma}^2/(\hat{N} - c)^2)]}) \tag{2.23}$$

The asymmetric confidence interval holds with high confidence (e.g., $z' = 3$ for 99.73% confidence), and so does the one-sided interval, $(-\infty, c + (\hat{N} - c) \cdot T]$ (e.g., $z' = 3$ for confidence greater than 99.73%). Using this improved one-sided interval as count estimate upper bound, a tighter approximate upper bound is derived:

$$\Delta_{bound2} = \left(\frac{\phi_K}{c} + z \cdot \sigma_K \right) \cdot \left(c + (\hat{N} - c) \cdot T \right) \tag{2.24}$$

This bound holds true if both the count estimate confidence interval and the value estimate bound hold true; we set $z' = 3$ (in T) and $z = 3$, to have Δ_{bound2} holds with at least 98% confidence (by Equation 2.20). In Section 2.5.8, we show by means of simulation that this achieves a much tighter bound than the previous method.

2.4 Other Aggregate Queries

In the following, we describe how the same techniques for *SUM*-aggregates can be applied to other aggregates for estimating the impact of unknown unknowns.

COUNT: Estimating *COUNT* is easier than *SUM* as it only requires estimating the number of unknown data items, but not their values. For instance, one could either directly use the *Chao*92 estimator or the techniques proposed in [34]. In addition, the *bucket* and *Monte-Carlo* approaches can be used simply by skipping the second step, i.e., not multiplying the estimated count with the value estimates.

AVG: The simplest way to estimate the *AVG* with unknown unknowns is to use the *AVG* over the observed sample S (i.e., the law of large numbers). This is reasonable because of the law of large numbers. However, S might be biased due to a publicity-value correlation and need to be corrected. One way to deal with the bias is to use our *bucket* approach with a simple modification on how the Δ_b per bucket are aggregated (e.g., weighted average of averages by the number of unique data items ($\hat{N}_{Chao92}$) per bucket).

MAX/MIN: At a first glance, it seems impossible to estimate *MIN* or *MAX* in the presence of unknown unknowns. However, we can still do better than simply returning the observed extreme values by reporting when we believe that the observed minimum or maximum value is the true extreme value. This is already very helpful in many integration scenarios and easy to do with our *bucket* estimator. The strategy divides the observed value range of S into consecutive sub-ranges (i.e., buckets); the number of unknown unknowns as well as their values are estimated per bucket. If the estimated unknown unknowns count in the highest (lowest) value range bucket is zero, then we say that we have observed the true maximum (minimum) value and only then report the highest (lowest) value.

2.5 Experiments

We evaluated our algorithms on several crowdsourced and synthetic data sets to test their predictive power. Crowdsourcing allowed us to generate many real data sets and avoided the licensing issues which often comes with other data sources. The source code and the crowdsourced data sets are available at https://github.com/yeounoh/Query-Estimation. We designed our experiments to answer the following questions:

- How does the estimation quality between the different estimators compare on real-world data sets?

- What is the sensitivity of our estimators in regard to data skew (publicity distribution and publicity-value correlation), and streakers/imbalance of data sources?
- How useful is the upper bound?
- How early are accurate *MIN*/*MAX* estimates possible?

2.5.1 Real Crowdsourced Data

We evaluated the estimation techniques on a number of real-world data sets, each gathered independently using Amazon Mechanical Turk, following the guidelines in [22]. Here we chose four representative data sets and four aggregate queries, which show different characteristics we encountered during the evaluation.

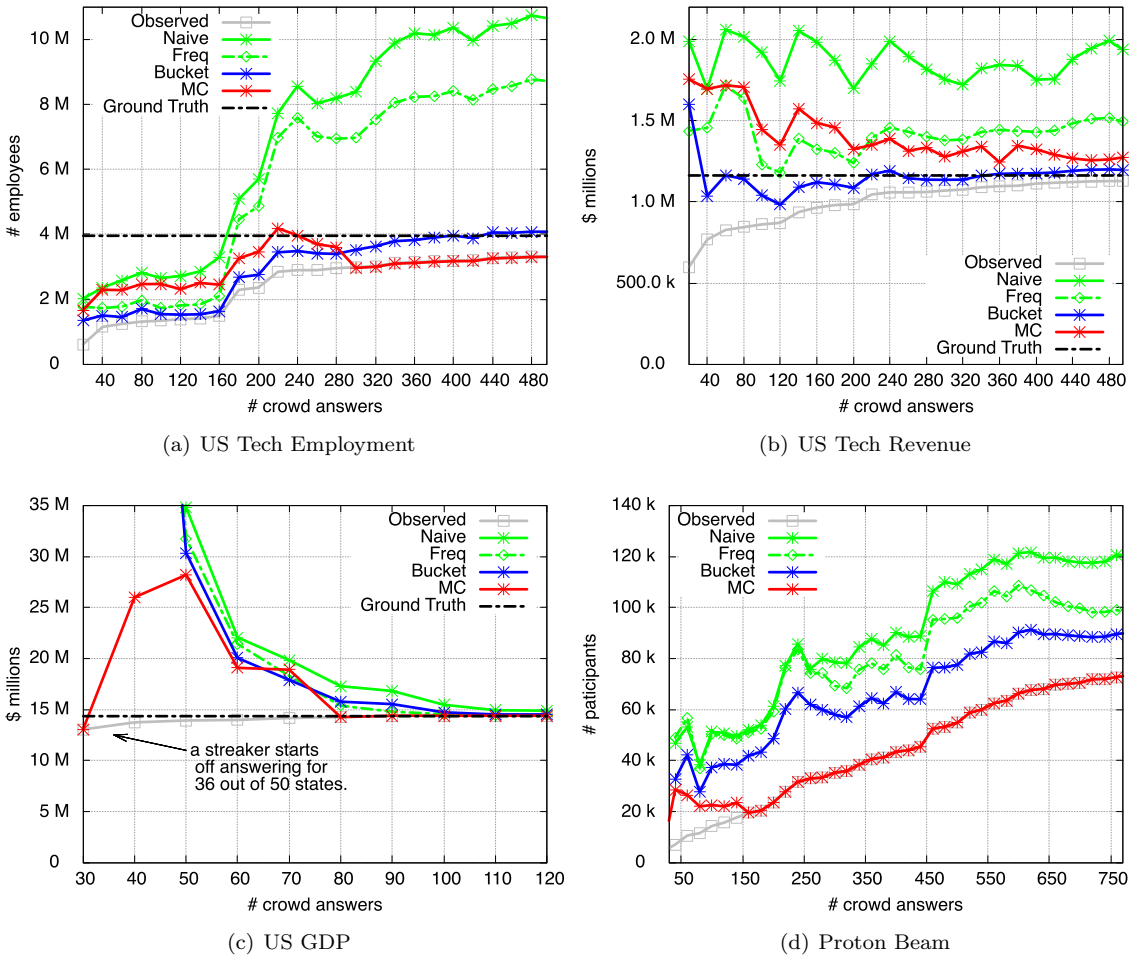
1. **US tech revenue & employment**: For the query: *how much revenue does the US tech industry produce?*, i.e., `SELECT SUM(revenue) FROM us_tech_companies`, we used the crowd to collect US[1] tech company names and revenues. Similarly, in an independent experiment we asked for US tech company names and number of employees, in order to answer the question: *how many people does the US tech industry employ?*, i.e., `SELECT SUM(employees) FROM us_tech_companies`. We selected the two data sets as they exhibit a steady arrival of unique answers from crowd workers.

2. **US GDP**: As a proof-of-concept experiment, we asked crowd workers to enter a US state with its GDP. This data set suffered from streakers.

3. **Proton beam**: Together with researchers from the field of Evidence Based Medicine (EBM) at Brown university we created a platform for abstract screening and fact extraction and spent over $6,000 on AMT, to screen articles about 4 different topics. Here we utilize the results on one of these, namely Proton beam: a set of articles on the benefits and harms of charged-particle radiation therapy for patients with cancer. Part of the abstract screening asked workers to supply the number of patients being studied. The question we aim to answer is *how many people, in total, participated in these types* of studies: `SELECT SUM(participants) FROM proton_beam_studies`. This data set and research question is grounded in a real world problem and unlike the other queries, this one does not have a known answer.

We paid between 2 and 35 cents per task. For the Proton beam experiment we designed a qualification test to filter out bad workers, the other experiments were done without qualification tests. Data cleaning was done manually: if workers disagreed on the value (e.g., the number of employees of a company) we used the average.

[1] We asked for companies in Silicon Valley to get a representative sample of US tech companies; without restrictions we received too many tiny computer shops and even non-US based companies.



(a) US Tech Employment

(b) US Tech Revenue

(c) US GDP

(d) Proton Beam

Figure 2.3: Real data experiments with aggregate SUM query

In the following we describe the results for every data set and the following estimators: *Naïve* (naive) (Section 2.2.1), *frequency* (Freq) (Section 2.2.2), *bucket* (Bucket) (Section 2.2.3), and *Monte-Carlo* (MC) (Section 2.2.4) estimators (other estimators did not perform that well or had the same performance and are only shown in Appendix A.2 and A.4).

US Tech-Sector Employment

Figure 2.3(a) shows the *SUM* estimates from the different estimators (colored lines) for our running example `SELECT SUM(employees) FROM us_tech_companies` as well as the observed *SUM* (grey line) over time (i.e., with an increasing number of crowd-answers). As the ground-truth (dotted black line) we used the US tech sector employment report from the Pew Research Center [33].

Both the *naïve* and *frequency* estimators heavily overestimate the impact of unknown unknowns. The *frequency* estimator does slightly better than the *naïve* estimator, which indicates that some

big companies have a high publicity likelihood and were observed early on by several sources.

In contrast, the *MC* estimator does well until it falls back to the observed query result. This can be explained by a peculiarity of this experiment. After roughly 280 crowd-provided data items, all remaining companies have a rather uniform publicity likelihood. In such a case, the *MC* estimator has a tendency to favor count estimates that are similar to the number of observed items: $\hat{N}_{MC} \sim c$. This tendency is a major drawback of our *MC* estimation technique.

Finally, the *bucket* estimator provides the best estimate (4053160.57 at 500 crowd answers), which is only $\sim$ 2.5% above the ground truth (3951730). While it is possible that the *bucket* estimator might require more data to converge, it is also possible that the ground truth is inaccurate: the employment statistics can vary widely based on many factors (e.g., inclusion or exclusion of part-time employees). We also speculate that there exist many smaller start-ups overlooked by survey agencies, due to the high data collection cost. In contrast, a school of crowd workers can more easily find smaller start-ups and their number of employees on web-pages. Thus, the *bucket* estimate could be closer to the ground truth than the one by the Pew Research Center. This is an astonishing result as the cost of crowdsourcing (e.g., $50.00 per 500 crowd-answers for US tech revenue & employment experiments) is probably only a small fraction of the cost of survey research by any major agency.

US Tech-Sector Revenue

Figure 2.3(b) shows the results for the US tech-sector revenue. In this data set, both the *naïve* and the *frequency* techniques overestimate the ground truth significantly because of the publicity-value correlation. While both estimators will eventually converge to the ground truth, it requires significantly more crowd-answers than what we collected.

Again, both *Monte-Carlo* and *bucket* estimators provide better estimates than *naïve* and *frequency* estimators. Yet, *Monte-Carlo* still overestimates, whereas *bucket* gives an almost perfect estimate after 240 answers. However, it can also be observed that the bucket estimator slightly over-estimates at the end of the experiment. This happens because one crowd worker suddenly reported a few unique companies causing the estimator to believe that there were more. Again, we cannot say with 100% certainty that our assumed ground-truth is actually the real ground truth and the *bucket* estimate might or might not be the real value.

GDP per US State

Figure 2.3(c) shows the estimate quality for our GDP experiment. To clean the data, we substituted the crowd reported GDP values with the values from [82]. This experiment suffered from streakers, i.e., uneven contributions from crowd workers. A single crowd-worker reported almost all answers in the beginning; this kind of aggressive behavior results in unusually high f_1, which throws off the estimators.

As the figure shows, only the *Monte-Carlo* based technique can actually deal with streakers and provides a reasonable estimate even in the beginning. However, it should also be noted that

all estimators converge after 100 samples (for $N = 50$). Furthermore, except for the *Monte-Carlo* estimator, there is no difference between the other estimators.

Proton Beam

Finally, results for Proton beam are shown in Figure 2.3(d). Again the *Monte-Carlo* estimator follows the observed line, which makes the estimates less interesting. Furthermore, we suspect that the *naìve* and the *frequency* estimators overestimate with constantly increasing number of unique data items (reviewed articles). By manually examining the data set, we confirm that this crowdsourcing experiment did not encounter any streakers, which may cause our estimators (e.g., *bucket*) to fail. Note that the *bucket* estimator converges to roughly $95k$, which we consider to be the best estimate of the number of participants for this particular type of cancer therapy effectiveness study.

Discussion

Overall, our *bucket* estimator has the highest accuracy. The only exception is when streakers are present, making the *Monte Carlo* perform better. However, it should also be noted, that the run-time of the *Monte-Carlo* estimator is significantly higher than the other estimators. While not a serious issue for our experiments (roughly 3.5s for *Monte-Carlo* vs. 0.2s for *bucket*), it could be significant for larger data sets, as the run-time scales linearly with sample size (the inner loop in Algorithm 2 depends on the sample size). In the remainder we analyze the different estimators in more depths using simulations and make final recommendations about which estimator to use at the end of the section.

2.5.2 Synthetic Data Experiment

To explore the estimation quality more systematically, we used a synthetic data set with $N = 100$ unique items, each having a single attribute-value ranging from 10 to 1000 ($attr = 10, 20, 30, ..., 1000$). We further simulated the sampling process outlined in Section 2.1 and used a scaled exponential distribution with parameter λ to model various publicity distributions ($\lambda = 0$: uniform; $\lambda = 4$: highly skewed). Finally, our simulation allowed us to vary the publicity-value correlation ($\rho = 0$: no correlation; $\rho = 1$: perfect correlation - the most frequent item also has the largest value).

Figure 2.4 shows the results for various synthetic data experiments, each of which is repeated 50 times and the results averaged (we omit the error bars for better readability). From left to right, we vary the number of simulated crowd-workers (i.e., sources) from $w = 100$, 10 to 5. From top to bottom, we first assume no publicity skew and no publicity-value correlation ($\lambda = 0, \rho = 0$), a for species estimation techniques often ideal scenario, we then show the more realistic scenario with skew and publicity-value correlation ($\lambda = 4, \rho = 1$), and finally simulate an environment where some rare items might contain high values ($\lambda = 4, \rho = 0$).

Ideal: Looking at the top-left figure with a uniform publicity distribution and a hundred workers, we can see that all estimators perform very well from the beginning. This is not surprising since all

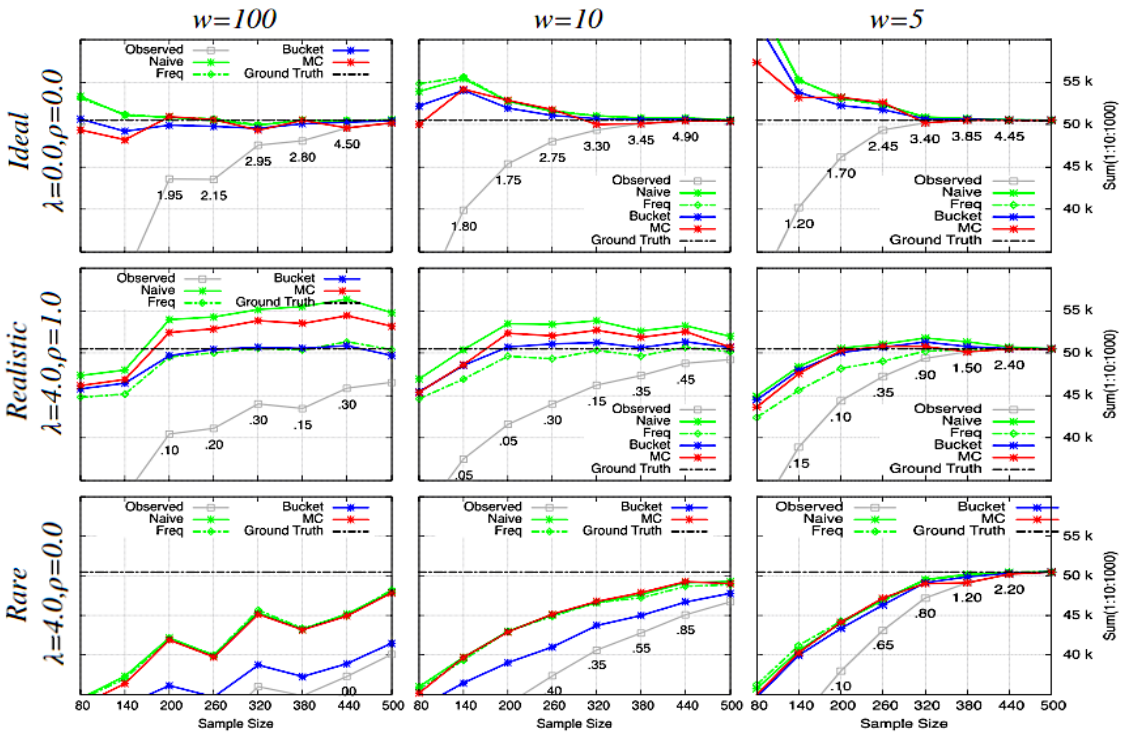
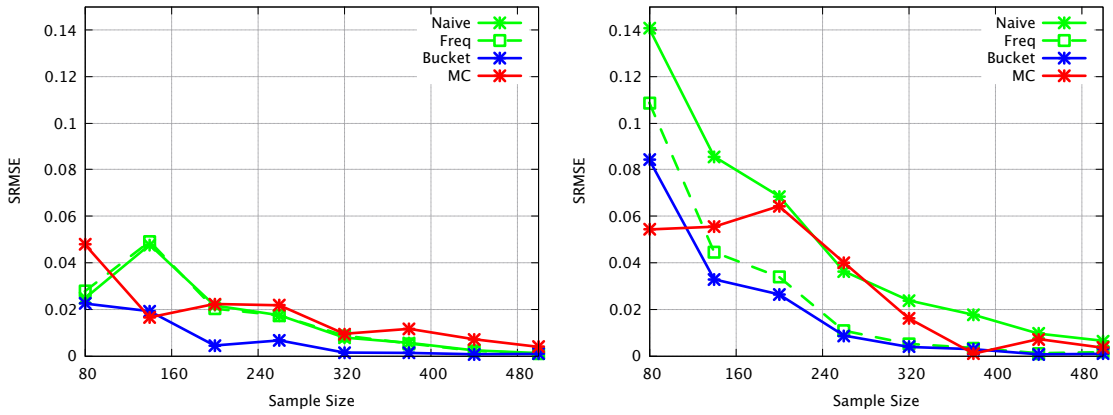


Figure 2.4: Synthetic data with varying number of sources (w), degrees of publicity skew (λ) & publicity-value correlation (ρ).

estimators work best with sampling with replacement from a uniform publicity distribution; having many workers sampling without replacement from a uniform distribution approximates sampling with replacement. With fewer numbers of workers sampling from the uniform distribution (top row), all estimators start to overestimate slightly. We conclude that *under the ideal conditions (i.e., the original assumptions of species estimation technique) all estimators perform equally well.*

Realistic: The middle row shows the scenarios which best resemble real-world use cases with a skewed publicity distribution and positive publicity-value correlation. In this case, the *bucket* estimator always provides the best estimates. However, in contrast to the real-world experiments the *frequency* estimator also performs well. This is due to a couple of reasons: Firstly, the publicity is highly skewed and perfectly correlated to the values. Secondly, the item values are evenly spaced. This helps the *frequency* estimator to under-estimate as *singletons* consist of only rare low-valued items from the tail – a peculiarity of this simulation. Also, with 5 evenly contributing workers almost all estimators perform about the same. However, the *bucket* estimator has less variance (not shown). We conclude that *under the more realistic conditions the bucket estimator performs the best and does not over-estimate.*

Rare events: Finally, we see in the bottom row that the *bucket* estimator is not the best choice. This is the case where we have skewed publicity, but no publicity-value correlation. In fact, all estimators perform poorly in this scenario, even with a lot of data sources (d). As the publicity



(a) $\lambda = 0.0$ (b) $\lambda = 1.0$

Figure 2.5: The accuracy of the estimators are measured in SRMSE, which reflects the bias and the variance of the estimates. *Bucket* is the most accurate estimator in both uniform and skewed publicity cases.

distribution tail can take on any values (i.e., no publicity-value correlation, the tail (i.e., *singletons*) can contain many high-impact values or "black-swan" events. In this case, because it conservatively favors underestimation, the *bucket* estimator performs worse. In summary, *none of the estimators are able to predict black-swan events or the long tail; all the estimators underestimate the ground truth.*

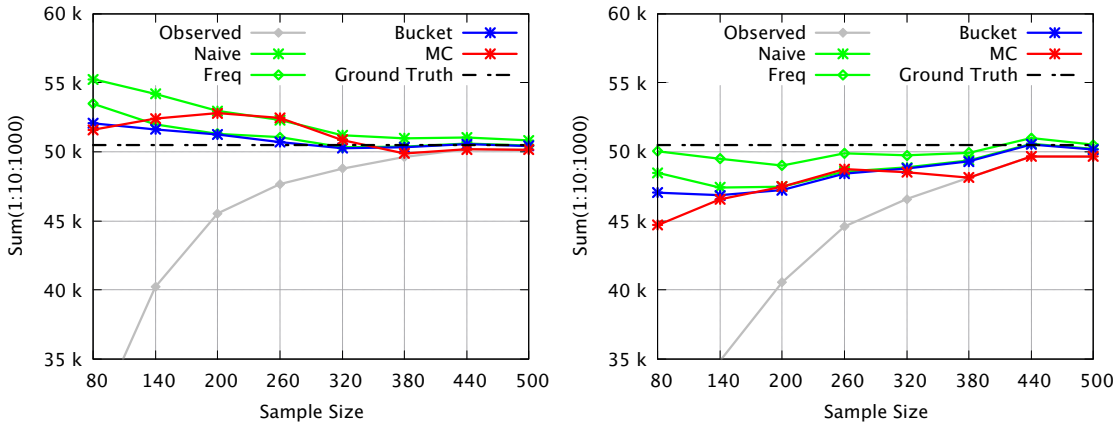
2.5.3 Estimation Accuracy

Bias and precision define the performance of estimators. The more biased and the less precise an estimator is, the worse is the performance. Accuracy (error) measures the overall/average distances between the estimated and the true point estimates; there are accuracy measures, e.g., Scaled Root Mean Squared Error (SRMSE), that explicitly combine bias and precision (variance): $SRMSE = \frac{1}{D}\sqrt{variance + bias^2}$.

As shown in Figure 2.4 most estimators, except *Bucket*, fails with increasing skew; this is again shown in Figure 2.5, using a synthetic data set with $w = 20$, $\lambda = 1.0$ and $\rho = 1.0$. Figure 2.5 illustrates how different estimators perform in terms of accuracy, and we see that *Bucket* is the best performing estimator, and the gaps between *Bucket* and the others grow larger with the skewed population.

2.5.4 Negative Publicity-Value Correlation

Many real-world scenarios assume positive publicity-value correlation, e.g., larger companies are more well-known and likely to be sampled by sources; our *Bucket* estimator benefits the most when the underlying population complies with such an assumption (i.e., *Bucket* splits to further



(a) Positive correlation

(b) Negative correlation

Figure 2.6: Simulation ($\lambda = 1.0$, $w = 20$) results with positive (a) and negative (b) publicity-value correlation. With the negative correlation, *Bucket* still converges to the ground truth faster than the observed aggregates, but at a slower rate than the positive correlation case.

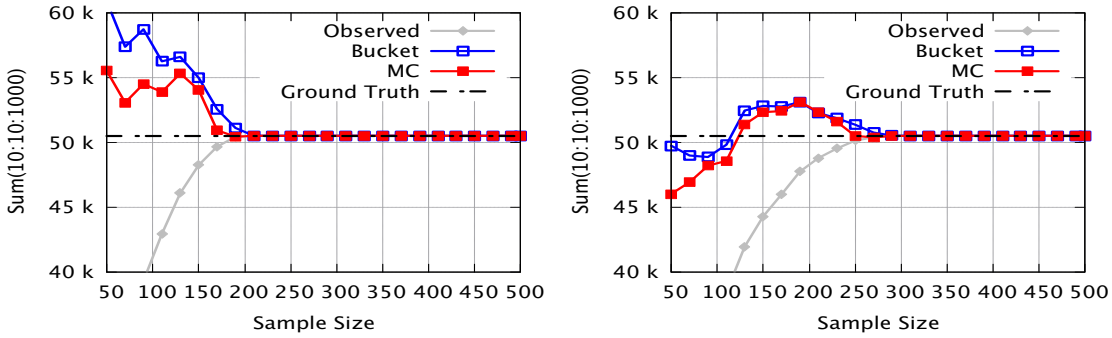
under-estimate).

In Figure 2.6, we illustrate how the proposed techniques perform with the negative publicity-value correlation; we show that *Bucket* still provides valuable estimates over the observed aggregates. However, notice how *Bucket* converges at a slower rate than in the case of positive correlation; this is expected, as rare items would have larger values so more aggressive estimates would benefit the most (e.g., *Freq* performs the best in the negative correlation scenario, with the largest value estimate solely based on the rare items). In other words, *Bucket* is more conservative and tend not to split, i.e., *Bucket* and *Naive* estimates overlap for the most part if the rare items tend to have large extreme values.

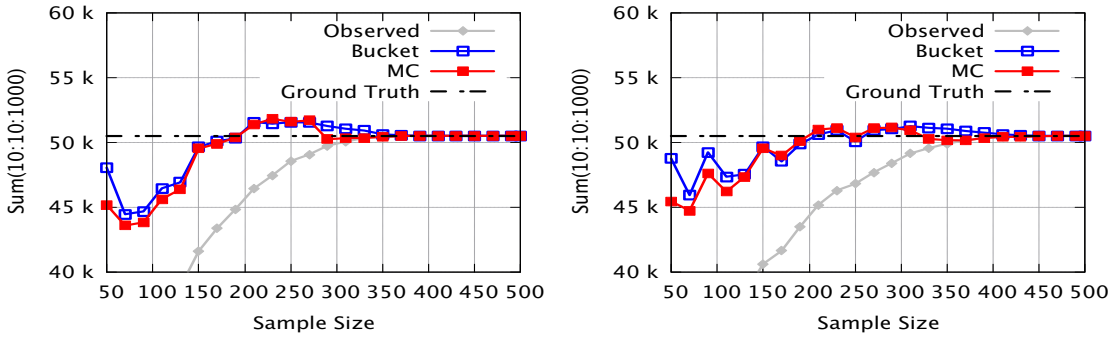
2.5.5 Number of Sources

Bucket estimator is non-parametric and works well with with both uniform and skewed distributions; however, it assumes a sample S sampled with replacement. This assumption is appropriate as long as enough independent data sources contribute evenly to S.

In Figure 2.7, we illustrate this with a synthetic data (skewed publicity correlated to item *attribute* values). In this particular example, more than 5 sources result in enough overlaps for *bucket* to estimate accurately; however, the minimum number of sources would vary with the data set. In addition, *Monte-Carlo* estimator converges faster as it does not assume sampling with replacement.

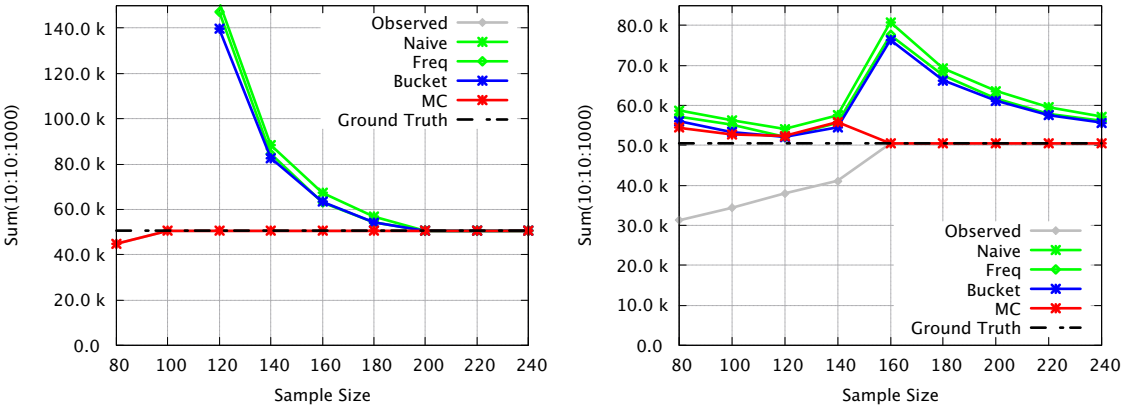


(a) $w = 2$ (b) $w = 3$



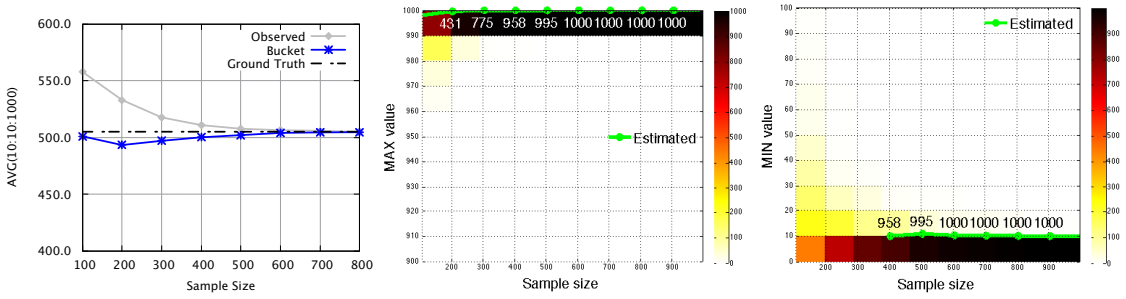
(c) $w = 4$ (d) $w = 5$

Figure 2.7: Synthetic data ($\lambda = 4.0$, $\rho = 1.0$: larger values are more likely) with varying number of sources (w). *Bucket* estimator performs better with more independent sources due to more overlaps.



(a) streakers only (b) a streaker injected at $n = 160$

Figure 2.8: Streaker effect experiments using a synthetic data ($\lambda = 1.0$, $\rho = 1.0$); *Monte-Carlo* is the most robust estimator against streakers.



(a) AVG (b) MAX (c) MIN

Figure 2.9: AVG query (d) and aggregate MAX/MIN queries (e)(f) experiments using a synthetic data ($\lambda = 1.0$, $\rho = 1.0$)

2.5.6 Streakers

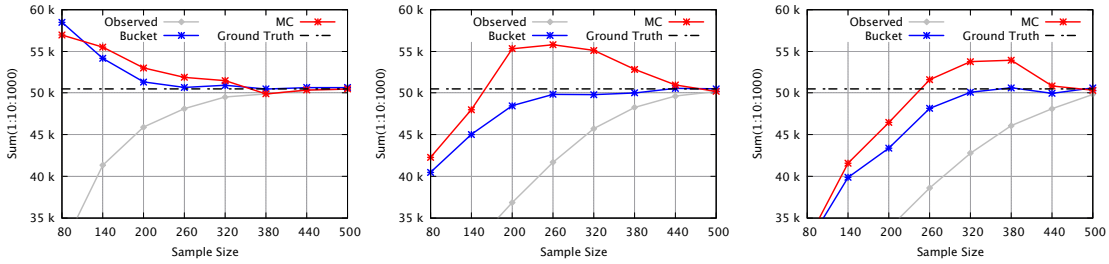
We have seen in Section 2.5.1 that the estimators can heavily overestimate in the presence of streakers. We now examine the effects of streakers using the synthetic data set with $w = 20$, $\lambda = 1.0$ and $\rho = 1.0$.

First, we consider an extreme case where each source successively provides all $N = 100$ data items; first, one data source contributes $n = 100$ items and then the second source starts to contribute its $n = 100$ items, and so on. Figure 2.8(a) shows that *Monte-Carlo* simply defaults to the observed sum from one source ($n = 100$), whereas all other estimators fail. This is because of the fact that all *Chao*92-based estimators assume a sample with replacement; an assumption which is strongly violated in this case. Only *Monte-Carlo* is more robust against streakers as it tries to best explain the observed S using simulation.

Next, we consider a more moderate case where we inject a single streaker (i.e., an overly ambitious crowd-worker). In Figure 2.8(b), a streaker is injected at the sample size $n = 160$, contributing all $N = 100$ unique data items directly afterwards. Similar to the previous case, all estimators, except *Monte-Carlo*, heavily overestimate in the presence of a streaker. Again, the reason is that *Monte-Carlo* uses simulation to explain the observed sample S instead of assuming that S was created using sampling *with* replacement.

2.5.7 Robustness of Monte-Carlo Estimator

Monte-Carlo is more robust to imbalanced data sources (e.g., streakers), as shown in Section 2.5.6. However, unlike *bucket* estimator, *Monte-Carlo* estimator is a parametric model in which the underlying item publicity is expected to follow an exponential distribution. In this case, the simple publicity-value correlation model (e.g., larger values are more likely or all item values are equally likely) is overly favorable to the parametric assumption of the estimator. Here, we assess robustness of *Monte-Carlo* under a non-exponential publicity distribution, namely, the gamma distribution.



(a) $\alpha = 1$ (b) $\alpha = 20$ (c) $\alpha = 50$

Figure 2.10: Robustness of *Monte-Carlo* estimator under the gamma publicity distribution with different shape parameter α. (a) $\alpha = 1$ results in an exponentially distributed publicity, and *Monte-Carlo* performs comparable to *Bucket*; (b) $\alpha = 20$ and (c) $\alpha = 50$ result in non-exponential publicity distributions with a mode in the middle of the item value range. *Monte-Carlo* performs worse than before as the (exponential) parametric assumption fails. Notice that our non-parametric *Bucket* estimator outperforms in all three scenarios.

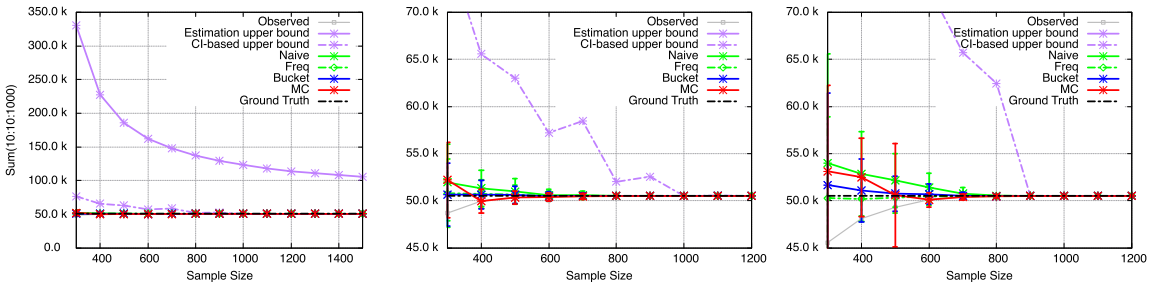
Figure 2.10 illustrates that *Monte-Carlo* performs better when the underlying publicity distribution is exponentially distributed (e.g., larger companies are more well-known). Such a parametric assumption is brittle (Figure 2.10(b)(c)), but still reasonable for our real crowdsourced data experiments. It would have been better for the *Monte-Carlo* estimator to use a more flexible parametric model. Developing a robust estimator in any circumstances is an important area of future work.

2.5.8 Other Queries

In this subsection we present results for other aggregate queries than *SUM* using the techniques from Section 2.4. As before we use synthetic data with 100 unique data items (e.g., with values $\{10, 20, 30, ..., 1000\}$) integrated over 20 sources with $\lambda = 1.0$ and a publicity-value correlation $\rho = 1.0$. The experiments are repeated 1000 times.

AVG: Figure 2.9(a) shows the observed (gray line) and estimated (blue line) for a simple average query of the form `SELECT AVG(attr) FROM table`. We only show the *bucket* estimation, as other estimates exactly overlap the observed *AVG* query results (i.e., when all unknown unknowns assume the same observed mean value, the *AVG* query result is the same as the observed). As with the sum-aggregates, our dynamic bucket estimator is able to correct the bias of the average because of the publicity-value correlation and provides an almost perfect estimate in this scenario.

MIN/MAX: Figure 2.9(b)(c) compactly visualizes the observed *MIN or* MAX query results. The heat-map shows when the real MIN/MAX value was observed in the data set (the darker the color the more often the result was observed given a number of samples over the 1000 repetitions). The green line shows on average, which value was reported if the unknown unknowns count estimate for the highest (*MAX*) / lowest (*MIN*) bucket was zero. The text next to the green line shows how often over the 1000 repetitions the MIN/MAX value was reported for a given sample size. As it is shown, the average is almost perfect for both *MAX* and *MIN* (note the actual minimum value is 10).



(a) $\lambda = 1.0$ (b) $\lambda = 1.0$ (c) $\lambda = 4.0$

Figure 2.11: We show the upper-bounds from Section 2.3 using a synthetic data set: (a) shows both bounds on the same figure for a weak publicity-value correlation skew, (b) a closer look at the results shown in (a), and (c) shows that the variances of the estimates are still more tightly bounded by the CI-based upper bound, even with a highly skewed data set.

That is, whenever our estimation technique for *MAX/MIN* reports a value, the user can have more trust in it. It should be noted, though, that it is impossible to estimate rare extreme values (black swans). Thus, it is only possible to improve upon the confidence but not eliminate any doubts in the results.

2.5.9 Upper Bound

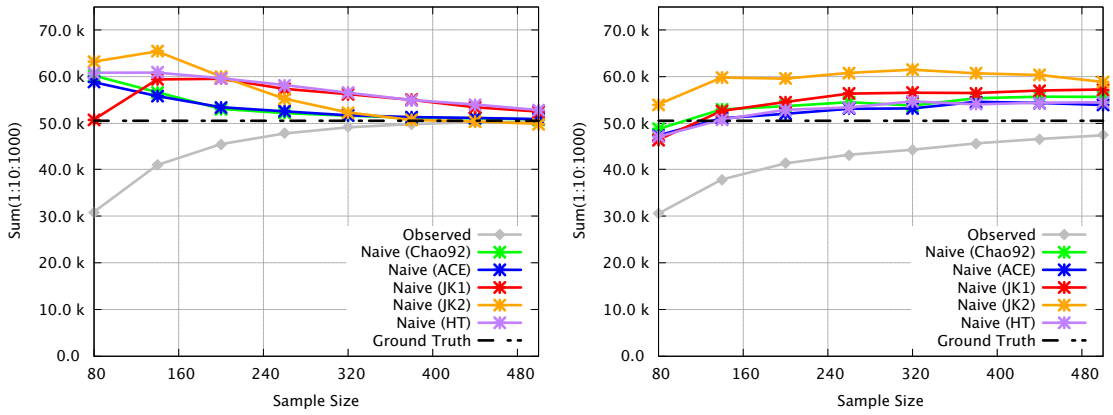
Finally in Figure 2.11(a) we show the upper-bounds from Section 2.3 using the same synthetic data set. As it can be seen, the worst-case estimation upper bound is very loose (i.e., very large compared to our estimates) and becomes tighter as we observe more data (the real data set results are similar and omitted for space constraints); while the upper bound provides a valuable insight, it may still be too loose for many real-world scenarios and we hope to improve it in the future.

On the other hand, the improved confidence interval-based upper bound is tighter as shown in Figure 2.11(a-c). Once again, such intervals would capture the ground truth with a high confidence (e.g., 99.95% of the times if repeatedly generated).

2.5.10 Other Base Estimators

There are a number of techniques for asymptotic species richness estimation; a handful of nonparametric estimators (i.e., model or underlying population assumption free), including *Chao*92, have gained increasing popularity over the past years [83].

In Figure 2.12, we compare the more widely-used (nonparametric) species estimators [69, 83, 84] and show how different base estimators perform at different skew-levels (e.g., low and high) with *Bucket*. For the comparison, we combine *Bucket* with *Chao*92 (described in Section 2.2.1; *Chao*84 is excluded for it performs worse than *Chao*92 for skewed data), improved Horvitz-Thompson estimator (*HT*) [84, 85], Abundance-based Coverage Estimator (*ACE*) [86], or First-Order Jackknife (*JK*1)



(a) $\lambda = 1.0$ (b) $\lambda = 4.0$

Figure 2.12: *Bucket* **estimation results with different base estimators;** *Chao*92 **is known to be one of the more robust estimators, and the simulation results show that it is relatively more robust even against high skew ($\lambda = 4.0$).**

and Second-Order Jackknife ($JK2$) estimators [87]. In the simulation results, we see that *Chao*92-based *Bucket* (in Green) is most accurate and efficient (i.e., converges faster with a smaller sample); we chose *Chao*92 as the base estimator for the proposed approaches, since it is one of the more robust estimators that explicitly considers data skew (e.g., publicity-value correlation). It is also interesting to see that the Jackknife estimators ($JK1$ and $JK2$) are not robust to data skew and either heavily over- or under-estimate.

Additionally, a recent study [88] found that estimating species richness using incidence-data (i.e., each species is noted if it is presented in a sample) instead of abundance-data (i.e., each species is noted how many times it is presented in a sample) yield more accurate estimates with lower sample coverage. This is interesting for *Bucket* since it estimates per bucket (i.e., a smaller value range with possibly lower sample coverage); however, our sampling model (see Section 2.1.2) assumes that each source is sampling without replacement. That is, the incidence-data is not distinguishable from the abundance-data in our integrated data set S, i.e., the aggregated incidences is the abundances of species.

2.5.11 Discussion

Which Estimator To Use: While the *Monte Carlo* or *bucket* estimators always dominate all the others, there is no clear winner between them. The *bucket* estimator performs exceptionally well unless the data sources are imbalanced. It provides the best performance on the real-world use cases (except on the GDP experiment, which suffers from streakers); furthermore, it performs at least as good as other estimators in the simulations from Section 2.5.2 (except for the *rare event* case, in which all estimators fail to predict black-swan events). However, when the data sources are imbalanced the *Monte Carlo* estimator wins.

Bucket estimator is a *Chao*92 based method, and thus, a *nonparametric model*, which does not require assumptions about the underlying distribution. However, it assumes a single sample without replacement, which is not an issue as long as enough independent data sources exist (in our simulations, 5 sources are often sufficient, see Section 2.5.5) and contribute evenly (i.e., no streakers). Also, the authors of [69] found that *Chao*92 estimator is inaccurate with very low sample coverage C (i.e., observed items are mostly *singletons*) and reported results for cases with $C \geq 0.395$ only. On the other hand, *Monte-Carlo* estimator is a 'data analytic method' [77] and really good at adjusting to the specifically observed sampling scenario (i.e., streakers), but at a cost of being a *parametric model*. The method assumes an exponential distribution to model the publicity distribution, which can be good or bad depending on the true shape of the underlying distribution.

Thus, our recommendation is to use *bucket* estimator, if the predicted sample coverage $\hat{C}$ (Equation 2.4) is greater than 40%, and when the analyst knows that enough data sources contribute evenly to the sample, and, otherwise, to use the more conservative *Monte Carlo* method.

Trust In The Results: With any types of estimators the main question arises: *How can we trust the estimate?* In 1953, Good, who worked with Turing on the estimators, already pointed out that "I don't believe it is usually possible to estimate the number of species ... but only an appropriate lower bound to that number. This is because there is nearly always a good chance that there are a very large number of extremely rare species"[70]. In estimating the Impact of unknown unknowns, this statement is even more critical as the rare items can have extreme values.

Nonetheless, species estimation techniques are extensively used in biology and even helped to decipher the Enigma machine [72]. We actually believe that it comes down to a simple question: *What do you trust more? A potentially wrong answer as no missing data is considered or a potentially wrongly corrected result.* Now, knowing that with enough data sources and no *streakers*, our *bucket* estimator tends to under-estimate the ground truth; in this case, one can generally say that it can only improve the estimates (see the simulations and real-world experiments). Unfortunately, without enough sources or in the presence of *streakers*, the answer is less obvious since the estimators might over-estimate more often. Thus, the answer to the question actually lies somewhere in the middle; to this end, the estimation upper bound can guide the decision. Overall, we believe that user should not stay on the blind-side; instead, user should be informed of the hidden ground truth (or a best guess) when a query result is incomplete.

In this work, we made a first step in the direction, while a lot remains to be done including developing tighter bounds, better ways to deal with the imbalance of sources, and easier ways to convey the meaning (and assumptions) of the estimates to the user.

2.6 Related Work

Traditional query processing assumes the database to be complete (i.e., closed world assumption). Furthermore, nearly all sampling-based query processing techniques assume knowledge of the population size [89]; hence, none of these are suitable for our problem with unknown unknowns. To

the best of our knowledge, [9] is the first work on estimating the impact of the unknown unknowns on query results (i.e., aggregate query processing in the *open world*). Most related to this work is [34], which first proposed to use species estimation techniques to estimate the completeness (i.e., the number of missing items) on query results. In this work, we extended these techniques and went further to also estimate the impact of the value of the missing data items (i.e., the unknown unknowns).

Species estimation: This work leverages the vast amount of work on species estimation techniques, like $Chao92$ [69, 70, 90]. Recent work [91] in this area even tries to estimate the shape of the population (e.g., support size, N). We could use these techniques in place of $Chao92$ to estimate the number of unknown unknowns, but not to directly estimate the *impact of unknown*, as the shape does not concern the values of unknown unknowns.

Species estimation techniques have also been used to estimate the size of search engine indexes and the deep web [92]. The problem is similar to our unknown unknowns count estimation [93]. Yet again, this line of work does not consider the unknown unknowns value.

Similar species estimation techniques have been used in the context of distinct value estimation for a database table [89, 94]. However those techniques leverage the knowledge of the table size to avoid over-estimation.

Survey Methodology & Missing Data: There is a vast body of literature on sampling-based statistical inferences to estimate population statistics [95–97] or techniques to deal with missingness of values [30, 31, 74–76].

However, unknown unknowns is different from the missing data; missingness refers to the case when the existence of record is known, but one or more attribute values are missing; most of the techniques assume the knowledge of population size to categorize something as missing (e.g., among this many people asked, only that many people responded, etc.). In addition, the cause of missingness (e.g., missing completely at random, missing at random, missing not at random) needs to be known to select appropriate techniques. The statistical inference techniques (e.g., multiple imputation based EM/maximum likelihood estimation [30, 74], propensity score estimation [75], or *Markov Chain Monte Carlo* simulation [30, 76]) used to fill the missing attributes make their inference based on the known non-missing attributes of the record. In the case of unknown unknowns, these assumptions are violated as the entire record (i.e., all attributes) are missing.

Missing data is also studied in the database community with respect to data cleaning [2, 32, 98]; however, since traditional RDBMS query processing function under the *closed world* assumption, they do not consider unknown unknowns as part of query processing.

Recent works defined database completeness in a partly *open world* semantic (i.e., database can be incomplete, which causes incorrect query results) and use the completeness information to denote the completeness of query [98, 99]. Similar in spirit to our work, they investigate the impact on query results of entire database records that may be missing [99]; however, they also assume the knowledge of population size (e.g., there are 7 days in a week) to define the completely missing records and measure the completeness.

Sampling-Based Query Processing: To cope with aggregates over large data sets, sampling based estimation techniques have been proposed as part of query processing [100–102]. One limiting aspect of any sampling based estimation techniques, though, is that they assume a complete database (i.e., *closed world*).

2.7 Summary

Integrating various data sources into a unified data set is one of the most fundamental tools to achieve high quality answers. However, even with the best data integration techniques, some relevant data might be missing from the integrated data set. In this work, we have developed techniques to quantify the impact of any such missing data on simple aggregate query results. The challenge lies in the fact that the existence and the value of the missing data is unknown. To our knowledge, this is the first work on estimating the impact of unknown unknowns on query results.

By nature, our techniques cannot predict black swan events (i.e., extremely rare data items). However, based on our evaluation results, we believe that the proposed techniques can provide valuable insights for users; rather than blindly believing a query result over an integrated data source, the user gets an idea of what the true answer might be.

There are several interesting future directions. Currently, none of our estimators works best under all circumstances. The *Monte-Carlo* estimator is robust against streakers, whereas the *bucket* estimator provides the most accurate results without streakers. Developing a robust estimator in any scenario is an important area of future work. Similarly, developing tighter error-bounds for the techniques is important. Finally, extending the results to more complex aggregate queries (e.g., with joins) also remains for future work.

This work is an important step towards providing higher quality query results. After all, we live in a big data world where even an integrated data set over multiple sources is possibly incomplete.

Chapter 3

Uncertainty as Undetected Data Errors

Data scientists report that data cleaning, including resolving duplicates and fixing inconsistencies, is one of the most time-consuming steps when starting a new project [1, 103]. A persistent challenge in data cleaning is coping with the "long tail" of errors, which can affect algorithmic, manual, and crowdsourced cleaning techniques. For example, data integrity rules can be incomplete and miss rare problems, and likewise, crowd workers that are not familiar with a particular domain might miss subtle issues. Consequently, even after spending significant time and/or money to clean and prepare a dataset, there may still be a large number of undetected errors.

3.1 Data Quality Metric (DQM)

We want to estimate the number of undetected errors in a data set and use the estimate as a data quality metric. In this section, we formalize our assumptions about the cleaning process and the problem setup. For convenience, Appendix B.1 contains a symbol-table.

3.1.1 Crowdsourced Data Cleaning

Most practical data cleaning tools rely on humans to verify the errors or repairs [16], and Crowd-based cleaning techniques use humans to identify errors in conjunction with algorithmic pre-processing. To overcome human error, the prevailing wisdom is to hire a sufficient number of workers to guarantee that a selected set of records is reviewed by multiple workers. This redundancy helps correct for false positives and false negatives, and there are many well-studied techniques to do so (e.g., Expectation Maximization and Voting [104, 105]). The core idea of all these techniques is that with more and more workers, the majority of the workers will eventually agree on the ground truth.

Consider a two-stage entity resolution algorithm as proposed in CrowdER [50]. During the first stage, an algorithmic similarity-based technique determines pairs that are likely matches (potential

errors), unlikely matches, and critical candidates for crowd verification. For example, likely matches could be the records pairs with a Jaccard distance of 1, unlikely matches as the ones with a similarity below 0.5, and everything in between as uncertain (i.e., the candidate matches). During the second stage, a crowd-worker is assigned to one of the candidate matches and determines if the pair is a match or not. However, assigning a pair to a single worker is risky since he or she can make a mistake. To improve the quality, we could assign a second and third worker to check the same items again. Although the quality of the cleaned data set improves, the impact of every additional worker decreases.

In this work, we also assign multiple workers to each item to verify and fix the errors, but at random. This enables our technique to leverage the diminishing utility of workers for the estimation. In other words, instead of assigning a fixed number of workers (e.g., three to form a quorum) to all items, we distribute a small subset of the dataset to each worker uniformly at random. Such redundancy in worker assignment gives rise to a reliable quality metric based on the number of remaining error estimations; however, it seems wasteful from the data cleaning perspective. In Section 3.6, we empirically show that the added redundancy, due to the random assignment, is marginal compared to the fixed assignment (exactly three votes per item).

3.1.2 Problem Statement

Every data set R consists of a set of records $\{r_1, ..., r_N\}$. Some subset of R is erroneous $R_{dirty} \subseteq R$. Let $W = \{w_1, ..., w_K\}$ be the set of *"fallible"* workers, and each worker $w_i \in W$ observes a subset of records $S^{(i)} \subseteq R$ and identifies those records that are dirty $S^{(i)}_{dirty}$ and clean $S^{(i)}_{clean}$, with some unknown error rates. Workers might have different error rates as well as a different set of internal rules to identify errors, but a sufficient number of workers would correctly identify errors in consensus [104, 105].

Therefore, we have a collection of sets $\mathcal{S}_{dirty} = \{S^{(1)}_{dirty}, ..., S^{(K)}_{dirty}\}$ of all the records that the crowd identified as dirty and a collection of sets of all the records that the crowd identified as clean $\mathcal{S}_{clean} = \{S^{(1)}_{clean}, ..., S^{(K)}_{clean}\}$. The worker responses can be concisely represented in a $N \times K$ matrix $\mathcal{I}$, where every column represents the answers from a worker k, and which entries are $\{1, 0, \emptyset\}$ denoting dirty, clean, unseen respectively.

Problem 1 (Data Quality Estimation)**.** *Let $\mathcal{I}$ be a $N \times K$ worker-response matrix with entries $\{1, 0, \emptyset\}$ denoting dirty clean, unseen respectively. The data quality estimation problem is to estimate $|R_{dirty}|$ given $\mathcal{I}$.*

This model is general enough to handle a variety of crowd-sourced cleaning operations such as missing value filling, fixing common inconsistencies, and removing irrelevant or corrupted records; the errors only need to be identifiable and countable by some of the fallible workers in W. Notice that the model does not require the ground truth (e.g., a complete set of rules/constraints to identify all errors in the dataset), and simply collects the votes from somewhat reliable, independent sources.

It turns out that we can also describe entity resolution problems in this way by defining R to be

a relation whose records are pairs of possibly duplicate records. Suppose we have a relation Q where some records refer to the same real-world entity. We can define $R = Q \times Q$ to be the set of all pairs of records from Q. A pair of records $(q_1 \in Q, q_2 \in Q)$ is defined as "dirty" if they refer to the same entity, and are clean if otherwise. Therefore, R_{dirty} is the set of all duplicated pairs of records (remove commutative and transitive relations to avoid double-counting, e.g., $\{q_1-q_2, q_1-q_4, q_2-q_1, q_2-q_4\} \mapsto \{q_1 - q_2, q_1 - q_4\}$). To quantify the number of entity resolution errors in a dataset, we estimate the cardinality of this set. In this paper, we use items, records, and pairs interchangeably.

Some pairs will be obvious matches or non-matches and do not require crowd-sourcing, and in Section 3.5, we describe how to integrate this basic model with algorithmic prioritization. Furthermore, the above problem definition concerns estimating the ground truth number of dirty items in a data set, but does not necessarily correct them. While in practice, identifying and correcting are often done together as part of this work we consider them as orthogonal problems.

3.2 Baselines

Based on Problem 1, we describe some existing approaches as our baselines. We consider crowd-sourced approaches, in the context where the true number of errors (or a complete set of constraints to identify all violations) is not available. We categorize these approaches as descriptive or predictive. Descriptive approaches only consider the first K workers (or worker response or task) to clean the dataset, and predictive approaches consider the impact of the future K' (possibly infinite) workers.

Nominal (descriptive): The most basic estimate for the number of errors, is to count the number of records marked as error by at least one worker:

$$c = nominal(\mathcal{I}) = \sum_{i}^{N} \mathbb{1}[n_i^+ > 0]$$

The number of positive votes on record i (marking it as dirty or an error), n_i^+, as well as the number of total votes n_i are at most $K > 0$. We refer to the estimate as *nominal* estimate. However, this estimate is neither forward looking nor tolerant of inconsistency, cognitive biases, and fatigue of workers, and this estimate may be far from the true number $|R_{dirty}|$.

Voting (descriptive): A more robust descriptive estimate is the majority consensus:

$$c_{majority} = majority(\mathcal{I}) = \sum_{i=1}^{N} \mathbb{1}[n_i^+ - \frac{n_i}{2} > 0]$$

If the number of workers who marked record i as dirty (n_i^+) is more than those who marked it as clean ($n_i^- = n_i - n_i^+$), then we label the record as dirty. If we assume that workers are better than a random-guesser (i.e., correctly identify dirty or clean records with probability $> 50\%$), then this procedure will converge as we add more workers. However, for a small number of workers, especially if each worker sees only a small sample of records, even the majority estimate may differ greatly

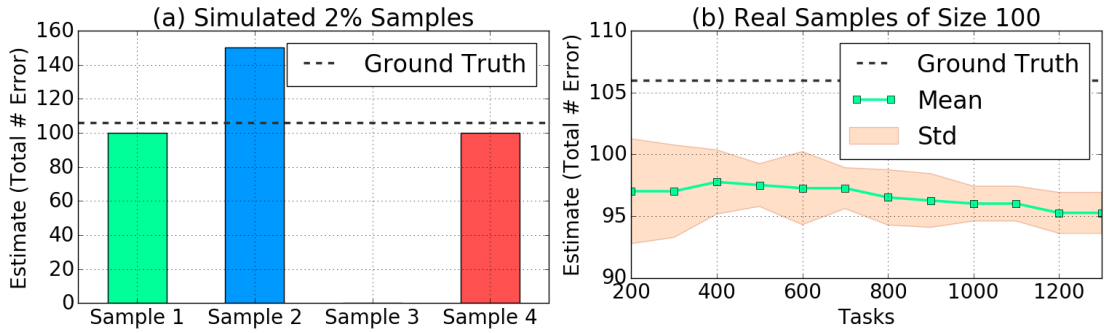


Figure 3.1: Estimating the total number of errors: (a) Extrapolation results using four different perfectly cleaned 2% samples; (b) extrapolation results with an increasing effort in cleaning the sample.

from $|R_{dirty}|$. Adding more workers may further improve the quality of the dataset, and accordingly, the data quality estimation problem is to estimate the value of adding more workers given $\mathcal{I}$.

Extrapolation (predictive): Unlike the previous descriptive methods, extrapolation is a simple technique to predict how many more errors exsit in the dataset if more workers were added. The core idea of the extrapolation technique is to "perfectly" clean a small sample of data and to extrapolate the error rate for the whole data set. For example, if a sample of $s = 1\%$ would contain $err_s = 4$ errors, we would assume that the whole data set has $err_{total} = \frac{1}{s} err_s = 400$ errors or $err_{remaining} = \frac{1}{s} err_s - err_s = 396$ remaining/undetected errors as we already have identified the 4 from the sample.

However, this technique has fundamental limitations: (1) How can one determine that the sample is perfectly clean (i.e., a chicken and egg problem) and (2) the sample might not be representative. More surprisingly, these two problems are even related. If the sample is small enough, one can probably put enough effort and resources (e.g., leverage several high-quality experts) to clean the sample. However, the smaller the size, the higher the chance that the sample is not representative of the original dataset. Worse yet, de-duplication requires comparing every item with the every other in the data set (we refer to one comparison between two tuples as an *entity pair*). This makes it even harder to determine a representative subset as we need to retrieve a representative (large enough) sample from an exponential number of combinations ($N * (N-1)/2$).

To illustrate such limitations, we ran an initial experiment using the *restaurant* data set (also used in [50, 106]). The restaurant data set contains 858 records of restaurants with some additional location information:

$$\textbf{Restaurant}(id, name, address, city, category)$$

Some of the rows of this dataset refer to the same restaurant, e.g., "Ritz-Carlton Cafe (buckhead)" and "Cafe Ritz-Carlton Buckhead". Each restaurant was duplicated at most once. For this data set, we also have the ground truth (the true number of errors), so we are able to quantify the estimation error of different techniques.

First, we run a simulated experiment of directly applying the extrapolation approach to 858 $\times$

858 = 367653 entity-pairs (Figure 3.1(a)). Of the 367653 entity pairs, only 106 of them are duplicate errors, which makes sampling a representative sample (with the representative number of errors) very difficult. This is related to the known problem of query selectivity in approximate query processing [107, 108]. We randomly sampled 2% (about 7300 pairs) four times and used an "oracle" to perfectly clean the data and extrapolated the found errors. Figure 3.1(a) shows how for relatively rare errors, this estimate is highly dependent on the particular sample that is drawn.

This is clearly impractical as cleaning 7300 pairs perfectly is not trivial, since workers are fallible. Worse yet, even with a perfectly clean sample, the perfect estimate is not guaranteed. Figure 3.1(b) depicts a more realistic scenario with the previously mentioned 2-stage CrowdER algorithm [50]. We used a normalized edit distance-based similarity and defined the candidate pairs as the ones with similarity between 0.9 and 0.5. Then, we took 4 random samples of size 100 out of the remaining 1264 pairs; we used Amazon Mechanical Turk to have each worker reviewing 10 to 100 random pairs from the candidate pairs, taking the majority consensus on items as true labels. Unfortunately for this particular experiment, the (average) estimate moves away from the ground truth as we use more workers to clean the sample. This is because the earlier false positive errors are corrected with more workers.

3.3 Species Estimation Approach

The previous section showed that the naìve extrapolation technique does not provide a good estimate. The main reason is that it is very challenging to select a good representative sample (errors are relatively rare). Furthermore, perfectly cleaning the sample might be so expensive that it outweighs the benefits of knowing how clean the data set really is. In this section, we pose the problem as a species estimation problem, for which a number of well-accepted estimation techniques exist (e.g., *Chao92* estimator [109]). The ultimate goal is to use species estimation to estimate the diminishing return and through it, the number of remaining errors in the data set.

3.3.1 Overview

In the basic species estimation problem, we are given n observations drawn with replacement from a population P. Suppose, we observe that there are c distinct values in the n samples, the species estimation problem is to estimate $|P|$. This problem mirrors the problem of predictive data quality estimates. Consider a finite population of records (or pairs in the case of entity resolution) a hypothetical infinite pool of workers and for simplicity no false positive errors (i.e., workers might miss an error but not mark a clean record as dirty). If we then assume that each worker goes over the whole data set R and marks records in that subset as clean or dirty, the votes from a worker can be seen as "discoveries" of dirty items, i.e, the "species". The species estimation problem is to estimate the total number of distinct erroneous records.

Unfortunately, it is often unrealistic to assume that every worker goes over all records. However, without loss of generality we can also present every worker with a random, even differently sized,

sub-set of the records. As more workers go over R, we will still see redundant dirty records marked as dirty by multiple workers as the *sample coverage* increases. That is, the resulting sample contributed by all the workers remains a sample with replacement.

3.3.2 Chao92 Estimator

There are several species estimation techniques, and it is well established that no single estimator performs well in all of the settings [107]. In this work, we use the popular *Chao92* estimator, which is defined as:

$$\hat{D}_{noskew} = \frac{c}{\hat{C}} \tag{3.1}$$

Here, c is the number of unique items (e.g., errors) we observed so far, C the sample coverage, and D_{noskew} our estimate for the total number of unique items. As true sample coverage is not known, it is estimated using Equation 2.4, where f_1 refer to all singletons, the errors which were exactly observed ones, whereas the f_2, refer to all doubletons, the errors which were exactly observed twice.

While it is obvious that f_1 refers to the number of times a rare error was observed, it is less clear how n should be defined in this context. One might argue that n should refer to the number of total votes by the workers. However, we are not interested in votes which declare an item as clean (also recall, that we do not consider any false positives for the moment). In fact, as we assume that all items are clean and no false positive errors by workers, a negative vote (i.e., clean) is a no-op. Thus, n should consist of positive votes $n^+ = \sum_{i=1}^{N} n_i^+$ only. This also ensures that $n = n^+ = \sum_{i=1}^{\infty} f_i$, yielding to the following estimator:

$$\hat{D}_{noskew} = \frac{c}{\hat{C}} = \frac{c}{1 - f_1/n^+} \tag{3.2}$$

We can also explicitly incorporate the skewness of the underlying data as follows:

$$\hat{D}_{Chao92} = \frac{c}{1 - f_1/n^+} + \frac{f_1 \cdot \hat{\gamma}^2}{1 - f_1/n^+} \tag{3.3}$$

where γ is *coefficient of variation* estimated using Equation 2.6.

Examples and Limitations

In the following, we show by examples how false positives and false negatives impact the prediction accuracy.

Example 3.1 (No False Positives)**.** *Suppose there are* 1000 *critical pairs with* 100 *duplicates. Each task assigned to a worker contains* 20 *randomly selected pairs. Workers have error detection rate of* 0.9*, but make no mistakes (e.g., wrongly mark a clean pair as a duplicate). We simulated this scenario and found that after* 100 *task, approximately* $c = 83$ *unique errors with* $n^+ = 180$ *positive votes and* $f_1 = 30$ *errors, which were only identified by a single worker. The basic estimate (without the correction factor* γ*) for the number of remaining errors is then:*

$$\hat{D}_{Chao92} - c = \frac{83}{1 - 30/(180)} - 83 = 16.6,$$

which is almost a perfect estimate. This is not surprising as our simulated sampled uniformly with replacement and thus, the Good-Turing estimate will, on average, be exact[1].

Example 3.2 (With False Positives)**.** *Unfortunately, it is unrealistic to assume that there will be no false positives. Even worse as errors are rare the number of false positives might be relatively high compared to the number of actual found errors. Let's assume* 1% *chance of false positive error (wrongly classify as dirty). Our simulation shows that on average that would add* 19 *wrongly marked duplicates, increases the* f_1 *count to* 46 *and* $n^+ = 208$*. With the false positives the estimate changes:*

$$\hat{D}_{Chao92} - c = \frac{83 + 19}{1 - (46)/(208)} - (83 + 19) \approx 131$$

Overestimating the number of true errors by more than 30%*.*

The Singleton-Error Entanglement

The reason why the false positives have such a profound impact on the species estimator is two-fold: First, they increase the number of unique errors c. Thus, if we estimate the number of remaining errors, we already start with a higher value. Second, and worse yet, the (number of) singletons, f_1, are the best indicator for of many errors are missing, while, at the same time, it is also the best indicator of erroneously classified items (i.e., false positives). We refer to this problem as the **singleton-error entanglement**.

3.3.3 vChao92 Estimator

In order to make the estimator more robust, we need a way to mitigate the effect of false positives on the estimate. First, we could use $c_{majority}$ instead of c to mitigate the impact on the number of found unique items. However, recall that the Chao92 estimate without the correction for the skew is defined as $c/(1 - f_1/n)$. That is the estimator is highly sensitive to the singletons f_1 as discussed before. One idea to mitigate the effect of false positives, is to **shift** the frequency statistics f by s and treat f_{1+s} as f_1, etc. For instance, with a shift of $s = 1$ we would treat doubletons f_2 (i.e., the items which were observed twice) as singletons f_1 and tripletons f_3 as doubletons f_2. Shifting f also means that we need to adjust $n^{+,s} = n^+ - \sum_{i=1}^{s} f_i$, to ensure the equality of $n = \sum_{i=1}^{\infty} f_i$. These statistics are more robust to false positives since they require more workers to mark a record as dirty, but at the cost of some predictive power. Taking the above ideas into account, we derive a new estimator:

$$\hat{D}_{vChao92} = \frac{c_{majority}}{1 - f_{1+s}/n^{+,s}} + \frac{f_{1+s} \cdot \hat{\gamma}^2}{1 - f_{1+s}/n^{+,s}} \tag{3.4}$$

vChao92 is more robust to false positives and estimate the size of the remaining errors; however, it converges more slowly than the original sample coverage-based estimators (*Chao92*) due to the singleton-error entanglement. Not only the rare errors f_1 are the best indicator for a false positive

[1]Note, that we do not have a perfect uniform sample, since every task samples the pairs without replacement; however, our results in Section 3.6 show that this is negligible if the number of tasks is large enough.

(i.e., since nobody else marked the same error, it might as well be a mistake), but also they hint on how many more errors are remaining (i.e., if there are many true errors that are hard to identify, then there might be more errors that are undetected because they are relatively more difficult to spot). Furthermore, *vChao92* requires to set s, which is hard to tune. Worse yet, the estimator violates an important (often desired) estimator property: it might not converge to the ground truth. In the next section we present another technique, which is parameter free and does not suffer from this problem.

3.4 Switch Estimation Approach

In this section, we show how we can estimate a slightly different quantity to avoid the shift parameter s and with better convergence guarantees.

3.4.1 Switch Estimation Problem

Ideally, we want to estimate how many errors are still remaining in a data set. In the previous section, we tried to estimate the **total number of errors** based on the initial "dirty" data set and the votes (i.e., $\{1, 0, \emptyset\}$) from multiple crowd-workers. Now, we ask an alternative question to estimate the **total number of switches**: After we used a cleaning technique (e.g., workers or an automatic algorithm), how many of the initially identified "clean" or "dirty" items are incorrect?

This is a different problem since we are no longer trying to just estimate the total number of errors ("dirty" items). Instead, we estimate the wrongly marked items: **At any given time, how many wrongly marked items (false positives and false negatives) does the data set still contain?** Assuming that workers are, on average, better than a random guesser, the majority consensus on each item will eventually be correct with a sufficient number of workers and their responses. This observation allows us to rephrase the original problem in terms of the consensus: **At any given time, how many of the majority vote decisions for any given record do we expect to switch?**

Problem 2 (Switch Estimation)**.** *We estimate the number of expected switches for the current majority consensus vector $\mathcal{V} \in \{0,1\}^N$ to reach the ground truth vector $\mathcal{E} \in \{0,1\}^N$. The data quality estimation problem is to estimate $|\{(v_i, e_i) : v_i \neq e_i, v_i \in \mathcal{V}, e_i \in \mathcal{E}\}|$ given $\mathcal{V}$, but not $\mathcal{E}$.*

Switches act as a proxy to actual errors and, in many cases, might actually be more informative. However, since a record can switch from clean to dirty and then again from dirty to clean, it is not the same as the amount of dirty records or remaining errors in the data set. We define the number of switches in $\mathcal{I}$ as follows:

$$switch(\mathcal{I}) = \sum_{i=1}^{N} \left(\sum_{j=2}^{K} \underbrace{\mathbb{1}[n^+_{i,1:j} = n^-_{i,1:j}]}_{(a)} + \underbrace{\mathbb{1}[n^+_{i,1} = 1]}_{(b)} \right) \tag{3.5}$$

Here, $n^+_{i,1:j}$ (or $n^-_{i,1:j}$) denotes the number of positive (or negative) votes on item i among workers 1 through j. We assume that the consensus on i flips (i.e., a switch happens) whenever there is a tie (part (a) of equation 3.5). For example, i is initially assumed to be clean, and after the first positive (dirty) vote we say i is dirty. And with another negative (clean) vote we have a tie and i is said to be clean because we flips the previous consensus on tie. Yet special attention has to be given to the beginning (i.e., the first vote). Here, we assume that all data items in the beginning are clean, and that if the first vote for a record is positive (e.g., marks it as dirty), the record switches (part (b) of equation 3.5). However, it is easy to extend the switch counting definition to consider various policies (e.g., tie-breaking).

3.4.2 Remaining Switch Estimation

While problem re-formulation is promising (Problem 2: it better reflects the number of remaining errors in the presence of human errors), it poses one significant challenge to our species estimation techniques: How should we define the f-statistics in the problem context? Surprisingly, there exists a simple and sound solution. Whenever a new switch occurs on item i, it is a *singleton*. If we receive an additional vote on i without flipping the consensus, we say the *switch is rediscovered*, and it changes to a *doubleton*. If another vote, yet again, does not flip the consensus, the same switch gets rediscovered again (e.g., changes it from a *doubleton* to a *tripleton*). On the other hand, if the consensus on i flips from, say, clean to dirty and then clean again, we do not rediscover the same switch, but a new one.

It naturally follows that we define n as the sum of the frequencies of the switch frequencies, $n = \sum_{i=1}^{\infty} f_i$. While this preserves the relationship between f_1 and n in the original species estimation technique, we found that this definition has a tendency to overestimate the number of switches. The reason is that, with every new switch, it is implicitly assumed that the sampling for the item resets and starts sampling for a new switch (although it is still for the same item) from scratch again. Therefore, we use a small modification and simply count all votes as n.

Finally, we need to, again, pay special attention to the first votes. Ideally, we would first get at least three votes for every item before doing any estimation. However, in practice we often want to start estimating based on the default labels (e.g., all records are assumed to be clean) before receiving a lot of votes for every record. The question is how votes that stay with the default labels (i.e., votes before the first switch) should be counted. Again, the solution is rather simple: The first k votes, which do not cause the default label to switch, do not re-discover a switch, and thus, they are no-ops (i.e., do not contribute to f-statistics as well as to n). Hence, we need to adjust n based on this new assumption and subtract all no-ops:

$$n^{switch} = n - \sum_{i=1}^{N} (\underset{j \in [1,K]}{\operatorname{argmin}} \{n^+_{i,1:j} \geq n^-_{i,1:j}\} - 1)$$

The last term of the equation removes all the no-ops before the first switch on each items (i.e., votes prior to the first switch do not contribute to the f-statistics, so they also should not contribute to

n).

We can now estimate the total number of switches as $K \to \infty$, using the same $Chao92$ estimation technique:

$$\hat{D}_{Switch} = \frac{c_{switch}}{1 - f_1'/n^{switch}} + \frac{f_1' \cdot \hat{\gamma}^2}{1 - f_1'/n^{switch}} \tag{3.6}$$

where c_{switch} is the number of records with any consensus flips, i.e., switches:

$$c_{switch} = \sum_{i=1}^{N} \mathbb{1}[switch(\mathcal{I}_{i,1:K}) > 0].$$

Using this estimator, then the expected number of switches needed for the current majority consensus to reach the ground truth is

$$\xi = \hat{D}_{Switch} - switch(\mathcal{I}).$$

This estimator has several desired properties. First, it is parameter-free, and there is no s to adjust. Furthermore, the estimator is guaranteed to converge to the ground truth. This is because of our main assumption, where the majority consensus will eventually become the correct labels (i.e., workers are better than a random guesser). And fewer switches are observed as the size of consensus on each item grows, and the estimator will predict the lower number of remaining switches (consider the definition of our f'-statistic). Lastly, as the estimator is more robust against false positives, it becomes less likely that, as the number of votes per item increases, a false positive would flip the consensus.

3.4.3 Switch-Based Total Error Estimation

Interestingly, although we changed the estimation problem from trying to estimate the total number of errors (Problem 1) to how many switches we expect from the current majority vote (Problem 2), we can still use the switch estimation to derive an estimate for the total number of errors (Problem 1). The idea is that we adjust $majority$ by the number of positive and negative switch estimates:

$$majority(\mathcal{I}) + \xi^+ - \xi^-$$

where positive switch ξ^+ is defined as switches from the "clean" label to the "dirty" label and negative switch ξ^- as switches from "dirty" to "clean." This only requires that we count c_{switch} and f_1' based on positive (or negative) switches only and estimate ξ^+ (or ξ^-) using the switch estimator (Equation 3.6) .

This makes sense, except that a separate estimation of positive and negative switches can cause either the positive or negative switch estimation to fail due to a lack of observed switches. To mitigate this problem, we make a key observation that $majority$ tends to improve monotonically with more task responses. Therefore, if we detect an increasing trend in $majority$, then it means that we have more positive switches and would have even more due to the monotonicity. If this is the case, then we focus on the positive switch estimates to adjust $majority$ total error estimate as:

$$majority(\mathcal{I}) + \xi^+$$

Similarly, if we observe a decreasing *majority* trend, then we can focus on the negative switch estimate:

$$majority(\mathcal{I}) - \xi^{-}$$

Our final total error estimation technique makes this decision dynamically to improve the estimates.

3.5 Estimation With Prioritization

A common design paradigm for data cleaning methods is "propose-verify". In a first step, a heuristic identifies candidate errors and proposes repairs. Then, a human verifies whether the proposed repair should be applied. This can further be extended where a human only verifies ambiguous or difficult repairs. In this section, we discuss the case where the data are not sampled uniformly.

3.5.1 Prioritization and Estimation

For some types of error, randomly sampling records to show first to the crowd will be very inefficient. Consider a crowdsourced Entity Resolution scenario, where the crowd workers are employed to verify matching pairs of records. Out of N total records, suppose k have exactly one duplicate in the dataset. This means that out of $\frac{N(N-1)}{2}$ pairs only k are duplicate pairs–meaning that even though the probability of sampling a record that is duplicated is $\frac{k}{N}$, it is roughly $\frac{k}{N^2}$ to sample a duplicated pair. It would be infeasible to show workers a sufficiently rich random sample for meaningful estimation in large datasets.

It is often the case the crowd sourced data cleaning is run in conjunction with algorithmic techniques. For example, in Entity Resolution, we may merge records that are sufficiently similar automatically, and reserve the ambiguous pairs for the crowd. To formalize, there exists a function $H : R \mapsto \mathbb{R}_+$ that is a measure of confidence that a record is erroneous. We assume that we are given a set of ambiguous records selected by the heuristic $R_H = \{\forall r \in R : \alpha \leq H(r) \leq \beta\}$, and this section describes how to utilize R_H in our estimates.

3.5.2 Simple: Perfect Heuristic

First, we consider the case where the heuristic H is perfect, that is, $\{r \in R : H(r) > \beta\} \cap R^c_{dirty} = \emptyset$ and $\{r \in R : H(r) < \alpha\} \cap R_{dirty} = \emptyset$. Note that the number of true errors in R_H is less than or equal to $|R_{dirty}|$, i.e., R_H does not contain any obvious cases, $R^c_H = \{\forall r \in R : H(r) < \alpha \text{ or } H(r) > \beta\}$. It turns out that this is the straight-forward case, and the problem is essentially as same as the original estimation problems (Problem 1, 2).

In the case of perfect heuristic, we randomly select a sample of p pairs from R_H and assign a number of workers to the sample; overall with K workers, we have $n = p \cdot K$ pairs/records marked by workers. Now, the target estimate is: $\hat{D}(R_H) = |R_{dirty} - R^c_H|$ instead of $|R_{dirty}|$. This result is easy to interpret as the perfect heuristic guarantees that $\{r \in R : H(r) > \beta\} \cap R^c_{dirty} = \emptyset$ and $\{r \in R : H(r) < \alpha\} \cap R_{dirty} = \emptyset$:

$$|R_{dirty}| = \hat{D}(R_H) + |\{r \in R : H(r) > \beta\}| \tag{3.7}$$

$\hat{D}(R_H)$ can be either $\hat{D}_{vChao92}$ or $\hat{D}_{switch}$ on R_H. Note that, depending on the qualities of workers, we still have false positive and false negative errors; the workers might make more mistakes as less obvious are asked. The obvious cases are efficiently classified by algorithm techniques.

3.5.3 Harder: Imperfect Heuristic

Next, we consider the harder case where the heuristic is imperfect: $\{r \in R : H(r) > \beta\} \cap R^c_{dirty} \neq \emptyset$ and $\{r \in R : H(r) < \alpha\} \cap R_{dirty} \neq \emptyset$. In this case, we would not only have false positives and false negatives on R_H due to the workers, but also have false negatives in R^c_H missed and false positives in R^c_H incorrectly identified by the heuristic itself.

In particular, we cannot use the simple approach (Equation 3.7) as with perfect heuristic, since $|\{r \in R : H(r) > \beta\}|$ might contain false positives by algorithm techniques. For example, if the heuristic's upper threshold α is too loose (with many false positives), we may not see any new errors in R_H. Moreover, we also need to include false negatives in $\{r \in R : H(r) > \beta\}$ to get the ground truth $|R_{dirty}|$.

To address this problem, we employ randomization. Workers see records from R_H with some probability $1 - \epsilon$ and see records from R^c_H with probability ϵ. This allows us to estimate remaining errors in R_H and R^c_H using the proposed techniques:

$$|R_{dirty}| = \hat{D}(R) \tag{3.8}$$

Although we are estimating over R, we still ask workers to look at mainly examples from R_H. The idea is that R^c_H still contains a few true errors compared to R_H, and we need fewer crowd answers to perform accurate species estimation. ϵ can be thought of as a measure of "trust" in the heuristic. As ϵ goes to zero, the model assumes the perfect case. On the other hand, as ϵ goes to $1 - \frac{|R_H|}{|R|}$ the model assumes the random sampling case[2]. Thus, ϵ defines a family of tunable data quality estimators that can leverage user-specified heuristics. In our experiments, we found that $\epsilon = 0.1$ is a good value to use.

3.6 Experiments

We evaluate our estimation technique using real world crowd-sourced data sets as well as synthetic ones. We designed our experiments to address the following questions:

- Do techniques perform on real-world data sets?
- What is the sensitivity of our estimators to false positive and false negative errors?
- Is reformulated switch estimation more robust to false positive errors?
- How useful is prioritization in error/switch estimation?

[2] $1 - \frac{|R_H|}{|R|} < \epsilon < 1$ implies that the Heuristic is negatively correlated with errors

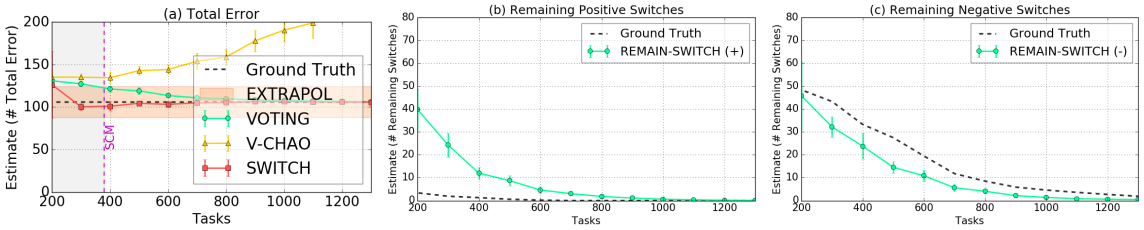


Figure 3.2: Total error estimation and positive and negative switch estimation results on Restaurant Data Set: there are more false positive errors, so *VOTING* monotonically decreases; hence, *SWITCH* provides the most accurate total error estimates based on the negative swith estimation.

3.6.1 Real-World Data Sets

We used Amazon Mechanical Turk (AMT) to crowd-source entity resolution tasks. Each worker receives a task consisting of a number of candidate records sampled from a data set R; the worker identifies which of the records are duplicates (label as *dirty*), for \$0.03 per task (e.g., \$3.00 per 100 tasks). Each task contains 10 records and a worker may take on more than a single task. In an effort to reduce noise in the collected responses, we designed a qualification test to exclude workers who are not familiar with the contexts of the data sets. When presenting results, we also randomly permute the workers and average the results over $r = 10$ such permutations. This averages out any noise in the particular order of tasks seen by workers. We consider the following approaches: **V-CHAO**, the voting and shift-based estimator with shift $s = 1$ (Section 3.3.3); **SWITCH**, a total error estimate derived from the switch estimator (Section 3.4.2); **VOTING**, the current majority vote over all items (Section 3.2); and **Ground Truth**, the true number of errors (or switches needed). We also show **EXTRAPOL** (Section 3.2) as a range of +/- 1-std around the mean of extrapolation results based on a perfectly clean 5% sample (assuming it was provided by an oracle). Lastly, we also show what we call Sample Clean Minimum (**SCM**), or the minimum number of tasks needed to clean the sample with a fixed number of workers per record:

$$\frac{3\ worker \times S\ records}{p\ records/task \times 1\ task/worker},$$

where we assume three workers to each record in the sample of size $S << N$, and each task contains p records and is assigned to a single (independent) worker. The point is to illustrate that the proposed DQM is comparable in the cost (the number of tasks) to actually cleaning the sample with a fixed number of workers, without the added redundancy of worker assignment. Of course, a perfectly clean sample does not provide reliable estimates like the proposed technique.

Restaurant Data set

We experimented on the restaurant dataset described earlier in the paper. The heuristic that we used to filter the obvious matches and nom-matching pairs was a normalized edit distance-based similarity > 0.9 and < 0.5. The remaining candidate pairs among 858×858 total pairs are 1264

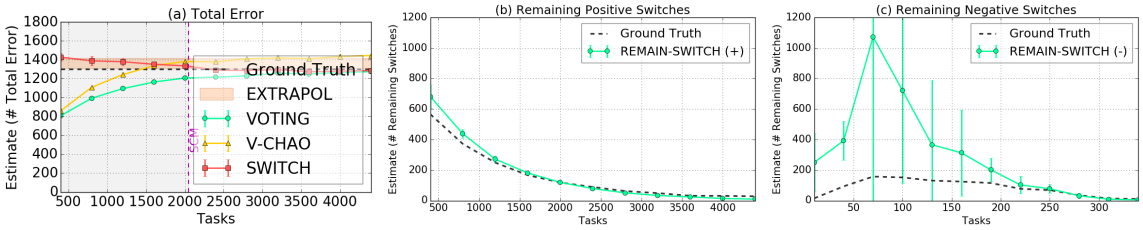


Figure 3.3: Product Data Set contains more false negative errors and *SWITCH* uses the more accurate positive switch estimation to yield the most accurate total error estimates.

pairs, with 12 pairs being true duplicates. Each task consists of ten random pairs from the 1264 critical pairs, which is given to a worker. Figure 3.2 shows how different estimation techniques work on the data set.

Given samples from the candidate pairs, the workers make a lot of false positive errors; the species estimation technique *V-CHAO* fails because the f_1 and f_2 counts are over-inflated. However, the switch estimation is more robust to false positives; thus *SWITCH* traces the ground truth very nicely and with a fewer number of tasks than the current majority vote (*VOTING*). So, even before all the pairs have been marked as clean or dirty (duplicate), we can already make a good prediction about the number of errors (duplicates). *EXTRAPOL* and its one standard deviation band illustrates that extrapolation based on a small perfectly clean sample does not provide reliable estimates due to the high variability. Unless you take many perfectly clean samples to average out the estimates, this is expensive and impractical; other estimates (e.g., *SWITCH*) are based on just a single sample, with some false positive and false negative labels. Lastly, the species estimation techniques require added redundancy due to the random assignment of workers with overlaps. But because *SWITCH* can estimate the number of remaining/total errors correctly, even before covering all the pairs within the sample, the number of tasks needed for *SWITCH* to provide good estimates is comparable to *SCM*. This is encouraging, especially because cleaning a sample perfectly and efficiently without redundancy would not guarantee good estimates.

Figure 3.2(b) and (c) show the number of positive and negative switches over time (i.e., tasks). The ground truth in switch estimation is defined as the number of switches needed (split in positive and negative) for the current consensus on all candidate pairs to reach the true labels. As it can be seen, there are more negative switches than positive switches left. This implies that we have more false positives (i.e., wrongly marked duplicate pairs) than false negatives. Another indication of the number of increased false positives is that *VOTING* in Figure 3.2(a) decreases over time. As we described in Section 3.4.3, *SWITCH* uses the number of estimated negative or positive switches to adjust the current majority consensus result. For the restaurant data set, we observe that the *SWITCH* estimates starts to exclusively use the estimate for the negative switches early on as the number of records marked as dirty is monotonically decreases. Figure 3.2(b) and (c) show that this is also the right choice. The estimation mechanism of *SWITCH* exploits the fact that the dataset error

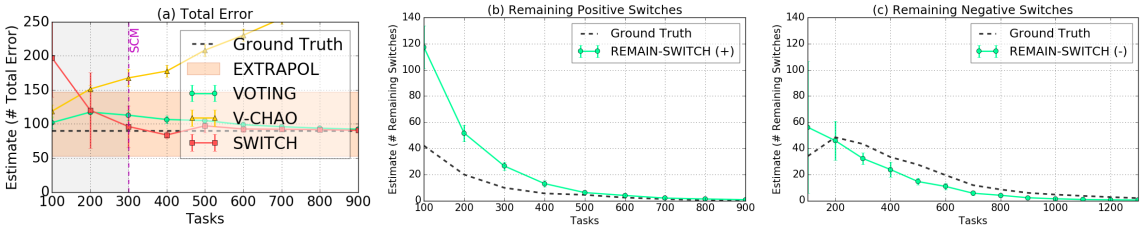


Figure 3.4: Address Data Set contains both false positive and negative errors in fair amounts. *SWITCH* uses positive switch estimates initially and overestimates further; however, once the workers slowly begin to correct false positive errors, *SWITCH* quickly starts to converge to the ground truth leveraging the accurate negative switch estimates.

rate monotonically improves (aka diminishing utility of error cleaning) over the increasing cleaning efforts; *SWITCH* uses either positive or negative switch estimates to correct *VOTING*. This results in a couple of nice properties for *SWITCH*. First, *SWITCH* makes the correction based on more reliable switch estimates. Tasks often consist of a single type of switch, which means that only positive or negative switch estimates are more reliable. Second, the monotonicity allows *SWITCH* to be at least as good as the baseline, since it uses positive switch (adding errors) only if *VOTING* is increasing and negative switch (subtracting errors) if *VOTING* is decreasing. This also means that, *SWITCH* can converge with the help of only one-sided switch estimates (negative switch estimation in this case).

Product Data Set

The product data set [50, 106] contains 2336 records by Amazon and 1363 records by Google in the form:

$$\textbf{Product}(retailer, id, name1, name2, vendor, price)$$

Each row belongs to either Amazon or Google ($retailer = \{Amazon, Google\}$), and the size of possible pairs is 1363×2336. Each product has at most one other matched product. The heuristic we used to filter the obvious matches and non-matching pairs was a normalized edit distance-based similarity > 0.7 and < 0.4. Thus, the prioritized candidate pairs contain 13022 pairs, of which 607 pairs are true duplicates. Because the matching task is more difficult than that of the restaurant data set, we observed more mistakes from workers.

Unlike the Restaurant dataset, the Product dataset contains more false negative errors; hence, *VOTING* exhibits an increasing trend, and we observe more positive switches. Therefore, *SWITCH* leverages remaining positive switch estimates. Figure 3.3 shows *SWITCH* outperforms all other estimators. Compared to the current majority votes, it provides a good estimate for the total number of errors early on. It is interesting to see that *V-CHAO* performs reasonably well in the early stage ($<$ 1200 tasks); however, its error sharply increases as we add more tasks. This can be attributed to the fact that we have a few difficult pairs on which more than just a single worker make mistakes. In this case, a fixed shift $s = 1$ cannot prevent *V-CHAO* from overestimating. Coming up

with the optimal shift parameter s a priori is a challenge, especially as it will vary across different data sets and worker responses.

Figure 3.3(b) and (c) show, as before, the separate estimates for the number of remaining positive and negative switches. This time, the positive switch estimation is more accurate; the negative switch estimation over-estimates and has rather large error bars. This indicates a low number of existing negative switches observed to perform reliable estimations. Therefore, as *VOTING* increases monotonically, *SWITCH* uses only the remaining positive switch estimates, and the heuristic yields a good total/remaining number of errors estimates as seen in Figure 3.3(a). *EXTRAPOL*, visualized through the shaded area in Figure 3.3(a), highly varies in the quality. The reason is that, again, the relatively small random sample may or may not be representative of the true error distribution. Furthermore, it should be noted that perfectly cleaning even a small 5% sample ($13022 \cdot 0.05 \approx 651$ pairs) is often already impractical (i.e., *chicken-and-egg problem*); how do we know after cleaning if the sample is perfectly clean?

Address Data Set

The address data set contains 1000 registered home addresses in Portland, OR, USA, which confirms to the following format in the given order:

$$< number\ street\ unit,\ city,\ state,\ zip >$$

The house number (*unit*) is optional, and the task is to identify any malformed address entries. The data set contains 90 errors (misformatted records). Since the task does not require pairwise comparison and the number of candidate entries is reasonable, we did not impose any prioritization rules.

This experiment is interesting for a couple of reasons. First, it deals with a different type of error (i.e., misformatted data entry). Second, the data set contains both false positives and false negatives in fair amounts, so *VOTING* does not improve much initially, for up to 300 tasks. In response, *SWITCH* uses positive switch estimates initially and overestimates further. However, once the workers slowly begin to correct false positive errors, *SWITCH* quickly starts to converge to the ground truth. Figure 3.4 illustrates this, along with both positive and negative remaining switch estimation results.

As a result, *SWITCH* estimates first based on the number of remaining positive switches and then on the remaining negative switches later. But it has a tendency to overestimate early on due to the large number of initial false positives. However, after gathering enough tasks (*SCM*), the *SWITCH* estimates converge to the true number of errors in the dataset.

3.6.2 Simulation Study

Next, we evaluate the different estimation techniques in a simulation based on the *Restaurant* data set. For all simulations, we used a subset with 1000 candidate pairs, among which 100 pairs are true duplicates. We randomly generated tasks and worker responses with the desired worker accuracy

(precision) and task sampling rate (coverage); there are three types of workers: namely, a false positive errors only-worker, false negative errors only-worker, or worker that make both types of errors. When a direct comparison of the original estimates is not appropriate (e.g., *Chao92* heavily overestimates with false positives), we used a scaled error metric, $SRMSE = \frac{1}{D}\sqrt{\frac{1}{r}\sum_r (\hat{D} - D)^2}$, to compare widely varying total error estimates of different techniques. Again, we permute the simulated data to repeat the experiments $r = 10$ times.

Sensitivity of Total Error Estimation

One of the central claims of this paper is that species estimation is not robust to false positive errors. To illustrate this point, we first explore exactly where the trade-off point would be between the two types of errors.

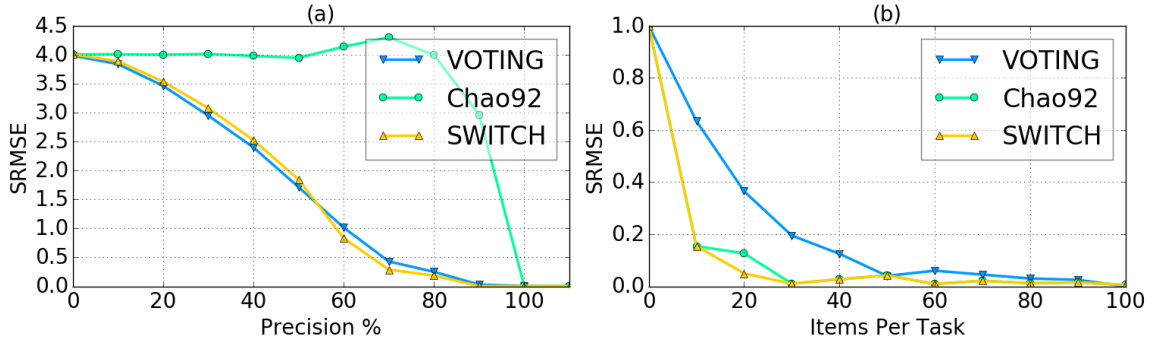


Figure 3.5: For a fixed number of tasks, we measure the scaled errors of the estimates as a function of (a) worker quality (precision) and (b) number of items per task (coverage).

In Figure 3.5(a), we vary precision (i.e., the failure rate of workers) for the given 50 tasks, each containing 15 items; in (b), we vary the number of items per task from 0 to 100. What can be seen in Figure 3.5(a) is that the *Chao92* is very sensitive to the number of false positives whereas *SWITCH* follows *VOTING* more closely. Furthermore, with a higher precision, *SWITCH* does slightly better than *VOTING* (e.g., it has the correct predictive power). But once workers becomes more fallible ($precision < 50\%$), *VOTING* becomes slightly better because there is really nothing that the *SWITCH* estimator can do (i.e., our main assumption that the majority consensus eventually converges to the ground truth is violated).

On the other hand, Figure 3.5(b) shows that if there are no false positives, *Chao92* does very well. This is not surprising as the estimation technique is forward looking (i.e., robust to false negatives). However, only *SWITCH* is capable of dealing with both false positives and false negatives.

Switch Estimation Is More Robust

We now study how the switch estimator behaves in comparison to the original species estimatiors (*Chao92*, *V-CHAO*). We consider three scenarios: 1) only with a false negative rate of 10% (e.g., a

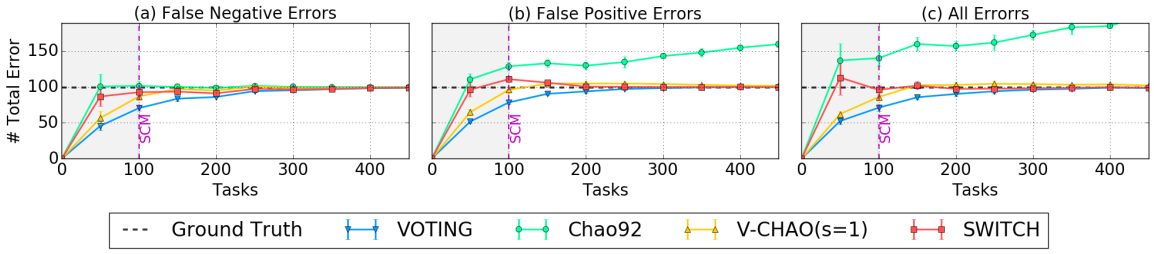


Figure 3.6: Total error estimates using the simulated datasets; *SWITCH* is the most robust estimator against all error types.

10% chance that a worker overlooks a true error), 2) only with a false positive rate of 1%, and 3) with both false negative and false positive errors at rates of 10% and 1%, respectively. Each task contains 15 items.

Figure 3.6(a) shows, without any false positives, that *Chao92* performs the best, and all other estimators converge more slowly. However, *SWITCH* still converges much faster than *V-CHAO*. The picture changes quite a bit in the case of false positive errors (b). *Chao92* now strongly overestimates, whereas *V-CHAO* and *SWITCH* provide accurate estimates. Here, *V-CHAO* actually performs surprisingly well. The reason is that, in our simulation, the error is evenly distributed across the items, making the (f-statistics) shifting more effective; an assumption that, unfortunately, rarely holds in practice as our real-world datasets experiments have demonstrated. Finally, in the case of both type of errors (c), *SWITCH* again performs well while *Chao92* again strongly overestimates and *V-CHAO* takes longer to converge and also slightly overestimates.

Prioritization

In our experiments on real data, we showed how our estimation techniques can be coupled with heuristics to prioritize what to show to the workers. However, sometimes, the heuristics may be imperfect (Section 3.5.3). To address this issue, we employ randomization. Workers see records from R_H, with some probability $1-\epsilon$, and records from R_H^c with probability ϵ. We measure the sensitivity of our best approach (*SWITCH*) to the choice of ϵ in Figure 3.7. For a fixed error rate and 50 tasks, we measure the accuracy of the estimator as a function of ϵ, the quality of the heuristic. We have a heuristic that has a 10% error rate and one that has a 50% error rate. When the heuristic is mostly accurate (10% heuristic error) small settings of ϵ suffice, and it is better to set ϵ lower. On the other hand, when the heuristic is very inaccurate (50% heuristic error), random sampling ($\epsilon \approx 0.5$) would minimize the estimation error. In our real experiments, we found that standard similarity metrics work very well as heuristics for de-duplication tasks.

3.6.3 Trust In The Results

The proposed data quality metric (DQM) is the first step to measure data quality, in terms of any undetected remaining errors, without the ground truth (e.g., prior set of rules or constrainsts to

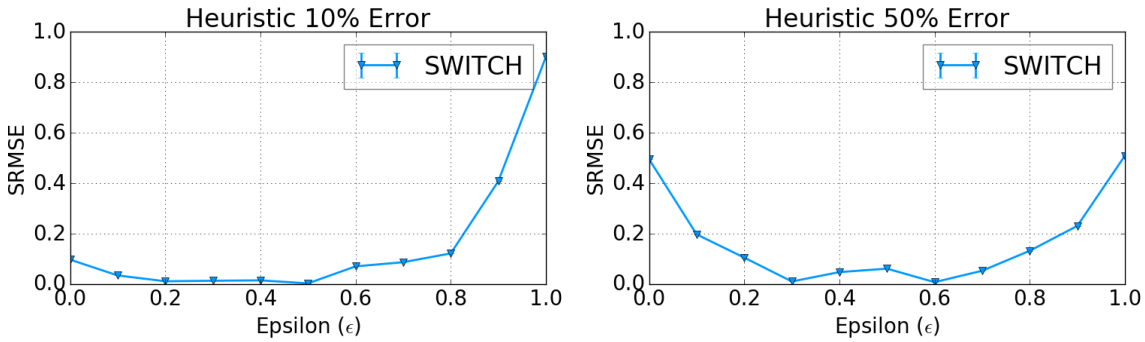


Figure 3.7: For a fixed error rate and 50 tasks, we measure the accuracy of the switch estimate as a function of the quality of the heuristic (ϵ).

define the perfect state of the perfect dataset). This is an important problem in data exploration as data scientists often lack the complete domain knowledge or such constraints to cover any types of errors. In this work, we demonstrated that DQM or *SWITCH* performs much better than any feasible baselines. Note that without the ground truth, there are not many ways to work with possibly fallible data cleaning approaches (e.g., algorithm or crowd workers). However, one question still remains: How much trust can an analyst place in our estimates?

We believe that the *SWITCH* estimator actually provides valuable information, in addition to the currently observed crowdsourced cleaning results (*VOTING*), to assess the quality of data when the ground truth is not available. However, it should be noted that *SWITCH* is not able to estimate very hard-to-detect errors (i.e., black swan events) and assumes that the workers are better than a random guesser, which holds true in practice [104, 110, 111].

3.7 Related Work

To the best of our knowledge, this is the first work to consider species estimation for data quality quantification. Species estimation has been studied in prior work for distinct count estimation and crowdsourcing [9, 51, 107]. However, the previous work only considered species estimation on clean data without false positive and false negative errors, which is inherent to the data quality estimation setting. There are also several other relevant lines of work related to this problem:

Label Estimation In Crowdsourcing: The problem of estimating true labels given noisy crowds is well-studied. For example, CrowdDB used majority votes to mitigate the effects of worker errors [110]. There have also been several proposed statistical techniques for accounting for biases between workers [104, 111]. Other approaches leverage gold-standard data to evaluate the differences between workers [112]. While the work in this area is extensive, all of the prior work studies the problem of recovering the true values in the already cleaned data. In contrast, we explore the problem of estimating the number of errors that may be missed in the uncleaned or sparsely cleaned dataset.

Progressive Data Cleaning: When data cleaning is expensive, it is desirable to apply it **progressively**, where analysts can inspect early results with only $k \ll N$ records cleaned. Progressive data cleaning is a well studied problem especially in the context of entity resolution [113–116]. Similarly, sampling is now a widely studied in the context of data cleaning [47, 117]. A progressive cleaning approach would assume a perfectly clean sample of data, and infer rules from the sample of data to apply to the rest of the dataset. This is similar to the extrapolation baseline evaluated in this paper. These techniques often assume that their underlying data cleaning primitives are infallible (or at least unbiased). When these techniques are integrated with possibly fallible crowds, then we arrive at the species estimation problems described in this paper.

Data Quality Metrics: There have also been a number of different proposals for both quantitative and qualitative data quality metrics [36–46]. Most of the techniques rely on assessing the number of violated constraints or tests designed by the user, or qualitatively measure the number of erroneous decisions made by programs using the data; the major drawback of these techniques is that they work on the basis of having the ground truth. On the other hand, our statistical approaches are designed to work without the ground truth for any/mixed types of errors that are countable. In terms of probabilistic techniques, Sessions et al. [39] learn a Bayesian Network model to represent the data and measured how well the model fits the data. This formulation is not sufficient for our problem where we consider large number of missing data. We explore a statistical formalism for data quality assessment (during progressive cleaning) when the data cleaning algorithms are fallible, and there is no available ground truth.

3.8 Summary

In this paper, we explored how to quantify the number of errors that remain in a data set after crowd-sourced data cleaning. We formalized a surprising link between this problem and the species estimation problem. However, we found that the direct application of species estimation techniques is highly sensitive to false positives during cleaning, and we proposed an alternative estimator, *SWITCH*, to address the issue. Our experimental results suggest that this new estimator, which is based on switches in worker consensus on records, is more robust to both false positives and false negatives.

We believe that variants of our approach could also apply to pure algorithmic cleaning (e.g., various machine learned error classifiers [48]). That is, instead of semi-independent workers, one could use several semi-independent automatic algorithms. One challenge will be to relax the assumption that workers are drawn from a single infinite population (i.e., workers are identical and consistent in their skill levels, with some noise). The finite-sample species estimation problem is discussed in [107], and we are interested in exploring such approaches in the future.

Chapter 4

Uncertainty as Model Quality

As machine learning (ML) systems become democratized, helping users easily debug their models becomes increasingly important. Yet current data tools are still primitive when it comes to helping users trace model performance problems all the way to the data. We focus on the particular problem of slicing data to identify subsets of the training data where the model performs poorly. Unlike general techniques (e.g., clustering) that can find arbitrary slices, our goal is to find interpretable slices (this is important because users need to take an action based on their interpretation and understanding of the problem) that are problematic and large. We propose Slice Finder, which is an interactive framework for identifying such slices using statistical techniques. The slice finder can be used for applications like diagnosing model fairness and fraud detection.

4.1 Data Slicing Problem

In this section, we provide a formal definition of the data slicing problem. To find interpretable slices that are also problematic and large, we treat each slice S and its complement as a hypothesis test. This allows Slice Finder to identify real problematic slices, where the input model significantly performs worse than on the rest of the data. For convenience, Appendix C.1 contains a symbol-table. We assume a dataset D with n examples and a model h that needs to be tested. Following common practice, we assume that each example $x_F^{(i)}$ contains features $F = \{F_1, F_2, ..., F_m\}$ where each feature F_j (e.g., country) has a list of values (e.g., {US, DE}) or discretized numeric value ranges (e.g., {[0, 50), [50, 100)}). We also have a ground truth label $y^{(i)}$ for each example, such that $D = \{(x_F^{(1)}, y^{(1)}), (x_F^{(2)}, y^{(2)}), ..., (x_F^{(n)}, y^{(n)})\}$. The test model h is an arbitrary function that maps an input example to a prediction using F, and the goal is to validate if h is working properly for different subsets of the data.

A *slice* S is a subset of examples in D with common features and can be described as a predicate that is a conjunction of literals $\bigwedge_j F_j \ op \ v_j$ where the F_j's are distinct (e.g., country = DE $\wedge$ gender = Male), and *op* can be one of $=$, $<$, $\leq$, $\geq$, or $>$. For numeric features, we can discretize their values (e.g., quantiles or equi-height bins) and generate ranges so that they are effectively

categorical features (e.g., age = [20,30)). Numeric features with large domains tend to have fewer examples per value, and hence do not appear as significant. By discretizing numeric features into a set of continuous ranges, we can effectively avoid searching through tiny slices of minimal impact on model quality and group them more sizable and meaningful slices.

We also assume a classification loss function $\psi(S, h)$ that returns a performance score for a set of examples by comparing h's predictions $h(x_F^{(i)})$ with the true labels $y^{(i)}$. A common classification loss function is logarithmic loss (*log loss*).

4.1.1 Model Validation

We consider the model validation scenario of pointing the user to "problematic" slices on which a single model performs relatively poorly. That is, we would like to find slices where the loss function returns a significantly higher loss than the rest of the examples in D. At the same time, we prefer these slices to be large as well. For example, the slice country = DE may be too large for a model to perform significantly worse than other countries. On the other hand, the slice country = DE $\wedge$ gender = Male $\wedge$ age = 30 may have a high loss, but may also be too specific and thus small to have much impact on the overall performance of the model. Finally, we would like the slices to be interpretable in the sense that they can be expressed with a few literals. For example, country = DE is more interpretable than country = DE $\wedge$ age = 20-40 $\wedge$ zip = 12345.

Finding the most problematic slices is challenging because it requires a balance between how significant the difference in loss is and how large the slice is. Simply finding a slice with many classification errors will not work because there may also be many correct classifications within the same slice (recall that a slice is always of the form $\bigwedge_j F_j \; op \; v_j$). Another solution would be to score each slice based on some weighted sum of its size and difference in average losses. However, this weighting function is hard to tune by the user because it is not clear how size relates to loss. Instead, we envision the user to either fix the significance or size.

4.1.2 Problematic Slice as Hypothesis

We now discuss what we mean by significance in more detail. For each slice S, we define its counterpart S' as $D - S$, which is the rest of the examples. We then compute the relative loss as the difference $\psi(S, h) - \psi(S', h)$. Without loss of generality, we only look for positive differences where the loss of S is higher than that of S'.

A key question is how to determine if a slice S has a significantly higher loss than S'. Our solution is to treat each slice as a hypothesis and perform two tests: determine if the difference in loss is *statistically significant* and if the *effect size* of the difference is large enough. Using both tests is a common practice [118] and necessary because statistical significance measures the existence of an effect (i.e., the slice indeed has a higher loss than its counterpart) while the effect size complements statistical significance by measuring the magnitude of the effect (i.e., how large the difference is).

To measure the statistical significance, we use hypothesis testing with the following null (H_o)

and alternative (H_a) hypotheses:

$$H_o : \psi(U, h) \leq \psi(U', h)$$

$$H_a : \psi(U, h) > \psi(U', h)$$

Here both U (or U') should be viewed as all the possible examples in the world that satisfy (or does not satisfy) the given predicate. This includes the training data and even the examples that the moel might serve in the future (i.e., $S \subset U$ and $S' \subset U'$). We then use Welch's t-test, which is designed for the case where two populations have unequal variances and sizes.

To measure the magnitude of the difference between the distributions of losses of S and S', we compute the effect size ϕ, which is defined as follows:

$$\phi = \sqrt{2} \times \frac{\psi(S, h) - \psi(S', h)}{\sqrt{\sigma_S^2 + \sigma_{S'}^2}}$$

Intuitively, if the effect size is 1.0, we know that the two distributions differ by one standard deviation. According to Cohen's rule of thumb [119], an effect size of 0.2 is considered small, 0.5 is medium, 0.8 is large, and 1.3 is very large.

4.1.3 Problem Definition

For two slices S and S', we say that $S \prec S'$ if S precedes S' when ordering the slices by increasing number of literals, decreasing slice size, and decreasing effect size. Then the goal of Slice Finder is to identify *problematic slices* as follows:

Definition 4.1. Given a positive integer k, an effect size threshold T, and a significance level α, find the top-k slices sorted by the ordering $\prec$ such that:

(a) Each slice S has an effect size at least T,

(b) The slice is statistically significant,

(c) No slice can be replaced with one that has a strict subset of literals and satisfies the above two conditions.

The top-k slices do not have to be distinct, e.g., country = DE and education = Bachelors overlap in the demographic of Germany with a Bachelors degree. In a user's point of view, setting the effect size threshold T may be challenging, so Slice Finder provides a slider for T that can be used to explore slices with different degrees of problematicness (see Section 4.2.3).

4.2 Slice Finder: An Automated Data Slicing Tool

Underlying the Slice Finder system is an extensible architecture that combines automated data slicing and interactive visualization tools. The system is implemented in Python (for a single node processing and the front-end) and C++ (to run the Slice Finder lattice search on a distributed processing framework such as Flume [120]).

Slice Finder loads the validation data set into a Pandas DataFrame [121]. The DataFrame supports indexing individual examples, and each data slice keeps a subset of indices instead of a copy of the actual data examples. Slice Finder provides basic slice operators (e.g., intersect and union) based on the example indices; Slice Finder accesses the actual examples only when evaluating the ML model. The Pandas library also provides a number of options to deal with dirty data and missing values. For example, one can drop NaN (missing values) or any values that deviate from the column types as necessary.

Once data is loaded into a DataFrame, Slice Finder processes the data to identify the problematic slices and allow the user to explore them. This process comprises three major components, summarized below.

Slice Finder searches for problematic slices either by training a CART decision tree [122] around mis-classified examples or by performing a more exhaustive search on a lattice of slices. Both search strategies progress in a top-down manner until they find top-k large problematic slices with $\phi \geq T$. The decision tree approach materializes the tree model and traverses to extract nodes for different request queries (with different k and T). In lattice searching, Slice Finder materializes all the candidate slices, even non-problematic slices. This allows Slice Finder to quickly respond to a new request with different T or continue searching with more filter clauses.

As Slice Finder searches through a large number of slices, some slices might appear problematic by chance (i.e., multiple comparisons problem [123]). Slice Finder controls such a risk, by applying a marginal false discovery rate (mFDR) controlling procedure [123]. Slice Finder compiles a final top-k recommendation list with only statistically significant problematic slices.

Lastly, even a handful of problematic slices can still be overwhelmingly large, since the user needs to take an action (e.g., deeper analyses or model debugging) on each slice. Hence, it is important to enable the user to quickly browse through the slices by their impacts (size) and scores (effect size). To this end, Slice Finder allows the user to explore the recommended slices with interactive visualization tools.

The following subsections describe the Slice Finder components in detail. Section 4.2.1 introduces the automated data slicing approaches without false discovery control, Section 4.2.2 discusses the false discovery control, and Section 4.2.3 describes the interactive visualization. Section 4.4.5 explains optimization techniques to get around the scalability issues in exploring many slices.

4.2.1 Automated Data Slicing

As mentioned earlier, the goal of this component is to automatically identify problematic slices for model validation. To motivate the development of the two techniques that we mentioned (decision trees and lattice search), let us first consider a simple baseline approach that identifies the problematic slices through clustering. And then, we discuss two automated data slicing approaches used in Slice Finder that improve on the clustering approach.

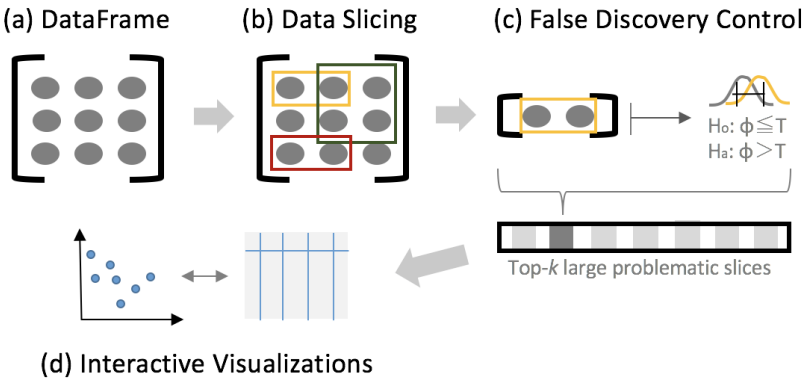


Figure 4.1: The Slice Finder **architecture: (a) Data is loaded into a Pandas DataFrame, and (b)** Slice Finder **perform automated data slicing to find top-k large problematic slices. (c)** Slice Finder **uses a false discovery control procedure to include only statistically significant problematic slices (d) for interactive visualizations.**

Clustering

The idea is to cluster similar examples together and take each cluster as an arbitrary data slice. If a test model fails on any of the slices, then the user can examine the data examples within or run a more complex analysis to fix the problem. This is an intuitive way to understand the model and its behavior (e.g., predictions) [57–59]; we can take a similar approach to the automated data slicing problem. The hope is that similar examples would behave similarly even in terms of data or model issues.

Clustering is a reasonable baseline due to its ease of use, but it has major drawbacks: first, it is hard to cluster and explain high dimensional data. We can reduce the dimensionality using principled component analysis (PCA) before clustering, but many features of clustered examples (in its original feature vector) still have high variance or high cardinality of values. Unlike an actual data slice filtered by certain features, this is hard to interpret unless the user can manually go through the examples and summarize the data in a meaningful way. Second, the user has to specify the number of clusters, which affects crucially the quality of clusters in both metrics and size. As we want slices that are problematic and large (more impact for model quality), this is a key parameter which is hard to tune.

The two techniques that we present next overcome these deficiencies of clustering. The first technique is based on decision-trees that capture the distribution of classification results. Here the effect sizes are large, but the slices may be smaller as a result. In contrast, the second technique, called lattice searching, focuses on slices that are neither too small nor large, but have large-enough effect sizes.

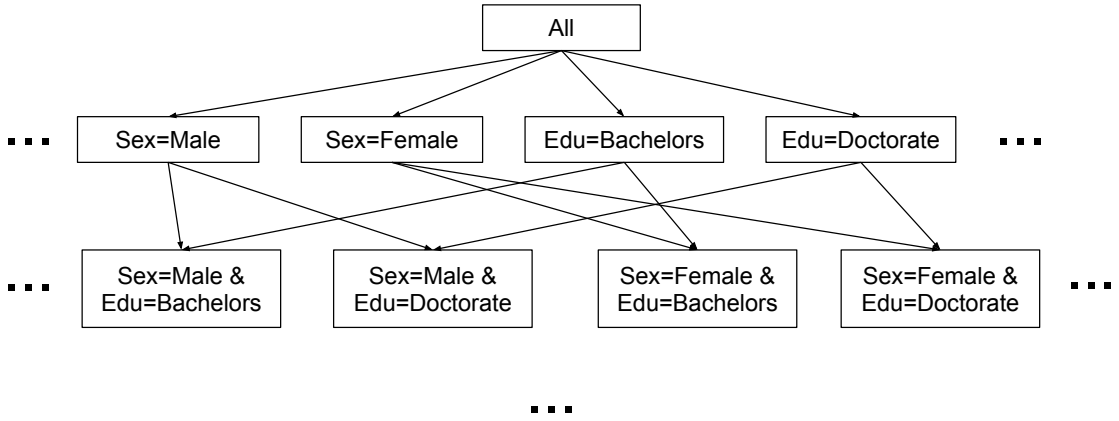


Figure 4.2: A lattice hierarchy of slices. In contrast with a decision tree, the search is more exhaustive and covers all possible feature combinations.

Decision Tree Training

To identify more interpretable problematic slices, we train a decision tree that can classify which slices are problematic. The output is a partitioning of the examples into the slices defined by the tree. For example, a decision tree could produce the slices $\{A > v,\ A \leq v\ \&\ B > w,\ A \leq v\ \&\ B \leq w\}$. For numeric features, this kind of partitioning is natural. For categorical features, a common approach is to use one-hot encoding where all possible values are mapped to columns, and the selected value results in the corresponding column to have a value 1.

To use a decision tree, we first identify the bottom-most problematic slices (leaves) with the highest effect size (i.e., highest error concentration). Then we can go up the decision tree to find larger (and more interpretable) slices that generalize the problematic slices, which still have effect size larger than a user-specified effect size threshold, T.

The advantage of decision trees is that they have a natural interpretation, since the leaves correspond directly to slices. The downside of using a tree is that it only finds non-overlapping slices that are problematic. In addition, if the decision tree gets too deep with many levels, then it starts to become uninterpretable as well [124].

The Decision Tree approach can be viewed as "greedy" because it optimizes on the classification results and is thus not designed to exhaustively find all problematic slices according to Definition 4.1. For example, if some feature is split on the root node, then it will be difficult to find single-feature slices for other features. In addition, a decision tree always partitions the data, so even if there are two problematic slices that overlap, at most one of them will be found. Hence, a more exhaustive approach is needed to ensure all possibly-overlapping problematic slices are found.

Lattice Searching

The lattice searching approach considers a larger search space where the slices form a lattice, and problematic slices can overlap with one another. We assume that slices only have equality predicates,

e.g., $\bigwedge_i F_i = v_i$. A value range for a feature can be expressed as a union of values of the feature, and we discretize any numeric (continuous) features. We search for larger problematic slices by expanding only non-problematic slices (filter predicates) with one additional literals at a time. In contrast with the decision tree training approach, lattice searching can be more expensive because it searches overlapping slices (slices can have different predicates, but still common examples).

Figure 4.2 illustrates how the slices are organized as a lattice. The key intuition is to perform a breadth-first search and efficiently identify problematic slices as shown in Algorithm 4.

ALGORITHM 4: Lattice Searching Algorithm

Input : Lattice L, max. number of slices to return k, effect size threshold T, significance level α, the set of all possible features F

Output: Problematic slices S

```
1  S = [];                                                    /* problematic slices */
2  Q = PriorityQueue();                          /* priority queue sorted by the ≺ ordering */
3  Q.push(L.root);
4  while |S| ≤ k and Q not empty do
5      s = Q.pop();                                           /* s = ⋀_{i∈I} F_i = v_i */
6      if EffectSize(s) ≥ T && IsSignificant(s, α) then
7          S.append(s);
8          UpdateWealth(α, 1);
9      end
10     else
11         Q.push({⋀_{i∈I} F_i = v_i ∧ G = v | G ∈ F − {F_1, ..., F_|I|}, v ∈ G's values});
12         UpdateWealth(α, 0);
13     end
14 end
15 return S;
```

The input is the training data, a model, an effect size threshold T, and a significance level α. As a pre-processing step, Slice Finder takes the training data and discretizes numeric features. For categorical features that contain too many values (e.g., IDs are unique for each example), Slice Finder uses a heuristic where it considers up to the N most frequent values and places the rest into an "other values" bucket. The possible slices of these features form a lattice where a slice S is a parent of every S with one more literal.

Slice Finder finds the top-k interpretable and large slices by traversing the slice lattice in a breadth-first manner using a priority queue. The priority queue contains the current slices being considered sorted by the ordering $\prec$. For each slice $\bigwedge_{i \in I} F_i = v_i$ that is popped, Slice Finder checks if it has an effect size at least T and is statistically significant. The statistical significance testing can be done using α-investing, which we discuss in Section 4.2.2. If the condition holds, the slice is added to the top-k list. Otherwise, the slice is expanded where the slices $\{\bigwedge_{i \in I} F_i = v_i \wedge G = v | G \in F - \{F_1, \ldots, F_{|I|}\}, v \in G's \text{ values}\}$ are added to the queue. Slice Finder optimizes this traversal by avoiding slices that are subsets of previously identified problematic slices. The intuition is that any subsumed (expanded) slice contains a subset of the same exact examples of its parent and is smaller with more filter predicates (less interpretable); thus, we do not expand larger and already

problematic slices. By starting from the base slices whose predicates are single literals and expanding only non-problematic slices with one additional literal at a time (i.e., top-down search from lower order slices to higher order slices), we can generate a superset of all candidate slices. This is similar to the *Apriori* fast frequent itemset mining algorithm [125], where only large $(d-1)$-itemsets are joined together to generate a superset of all large d-itemsets. Depending on whether the condition (line 6) holds or not, we either increase or decrease the α-wealth accordingly (details on the updating strategy are discussed in Section 4.2.2). This process repeats until the top-k slices have been found or there are no more slices to explore.

Example 4.1. *Suppose there are three features A, B, and C with the possible values $\{a_1\}$, $\{b_1, b_2\}$, and $\{c_1\}$, respectively. Also say $k = 2$, and the effect size threshold is T. For simplicity, we assume all slices are statistically significant and thus do not update the α-wealth. Initially, the priority queue Q contains the entire slice. This slice is popped and expanded to the slices $A = a_1$, $B = b_1$, $B = b_2$, and $C = c_1$, which are inserted back into the queue. Among them, suppose $A = a_1$ is the largest slice with an effect size at least T. Then this slice is popped from Q and added to the top-k result. Suppose that no other slice has an effect size at least T, but $B = b_1$ is the largest. This slice is then expanded to $B = b_1 \wedge C = c_1$ (notice that $B = b_1 \wedge A = a_1$ is unnecessary because it is a subset of $A = a_1$). If this slice has an effect size at least T, then the final result is $[A = a_1, B = b_1 \wedge C = c_1]$. Finally, there are no ties to break in terms of effect size.*

The following theorem formalizes the correctness of this algorithm for the slice-identification problem.

Theorem 4.2. *The* Slice Finder *slices identified by Algorithm 4 satisfy Definition 4.1.*

Proof. Since we only add slices with effect size at least T and statistically significant, given α-level, to the priority queue, the first and the second conditions are satisfied trivially. The third condition can be proven to hold using contradiction. Suppose a slice S that is popped from the queue has a large enough effect size, but there is another slice S' that has not yet been added to the result, but has the strict subset of literals (predicates) of S and should have been added to the result first. S' must have been all popped and expanded before S was popped, like any other ancestors of S. In addition, since S' has fewer features than S, it should have been placed before S in the queue (hence the contradiction). □

Scalability

Slice Finder optimizes its search by expanding the filter predicate by one additional feature/value at a time (top-down strategy). Unfortunately, this does not solve the scalability issue of the data slicing problem completely, and Slice Finder could still search through an exponential number of slices, especially for big high-dimensional data sets. To this end, Slice Finder implements two approaches that can speed up the search.

Parallelization: For lattice searching, evaluating a given model on a large number of slices one-by-one (sequentially) can be very expensive. So instead, Slice Finder distributes the slice evaluation

jobs (lines 5–10 in Algorithm 4) by keeping separate priority queues Q_d for the different numbers of filter predicates d. The idea is that workers take slices from the current Q_d in a round-robin fashion and evaluate them asynchronously; the workers push the next candidate slices $\{\bigwedge_{i \in I} F_i = v_i \wedge G = v | G \in F - \{F_1, \ldots, F_{|I|}\}, v \in G's \text{ values}\}$ with one additional filter clause G to Q_{d+1} as they finish evaluating the slices. Once done with Q_d (i.e., Q_d is empty and $|S| \leq K$), Slice Finder moves onto the next queue Q_{d+1} and continue searching until $|S| \geq K$. Keeping slices of different d in separate queues allows multiple workers to evaluate multiple slices in parallel, without having to worry about redundant discoveries because only slices with $d + 1$ predicates can be subsumed by slices with d predicates. The added memory and communication overheads are negligible, especially, with respect to the slice evaluation time.

On the other hand, for decision tree training approach (DT), our current implementation does not support parallel learning algorithms for constructing trees. But, there exist a number of highly parallelizable learning processes for decision trees [126], which Slice Finder could implement to make DT more scalable.

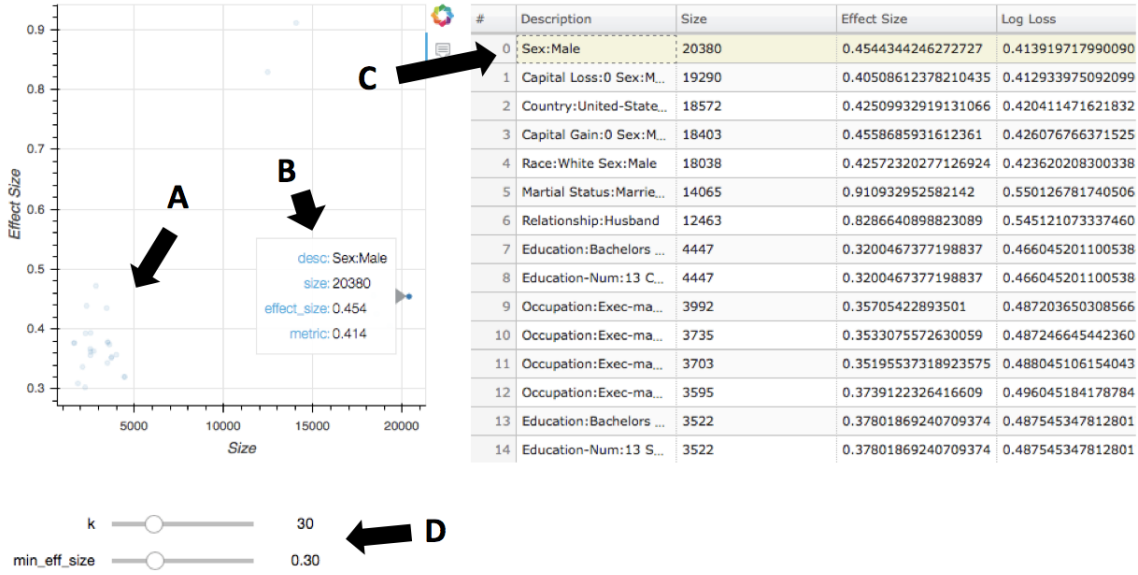
Sampling: We take a smaller sample to run Slice Finder if the original data set is too large. Note that the run time is linearly proportional to the size of sample, assuming that the run time for the test model is constant for each example. Taking a sample, however, comes with a cost. Namely, we run the risk of false positives (non-problematic slices that appear problematic) and false negatives (problematic slices that appear non-problematic or completely disappear from the sample) due to a decreased number of examples. Since we are interested in large slices that are more impactful to model quality, we can disregard false negatives that disappeared from the sample. Furthermore, we perform significance testing to filter slices that falsely appear as problematic or non-problematic (Section 4.2.2).

It is interesting to further investigate the tradeoff between the cost of sampling (e.g., increased false positive rate) and its benefit (e.g., performance gain).

4.2.2 False Discovery Control

As Slice Finder finds more slices for testing, there is also the danger of finding more "false positives," which are slices that are not statistically significant. Slice Finder controls false positives (Type-1 errors) in a principled fashion using α-investing [123]. Given an alpha-wealth (overall Type I error rate) α, α-investing spends this over multiple comparisons, while increasing the budget α towards the subsequent tests with each rejected hypothesis. This so called pay-out (increase in α) helps the procedure become less conservative and puts more weight on null hypotheses that are more likely to be faulty. More specifically, an alpha-investing rule determines the wealth for the next test in a sequence of tests. This effectively controls marginal false discovery rate at level α:

$$\frac{\mathbb{E}(V)}{\mathbb{E}(R)} \leq \alpha$$



#	Description	Size	Effect Size	Log Loss
0	Sex:Male	20380	0.4544344246272727	0.413919717990090
1	Capital Loss:0 Sex:M...	19290	0.40508612378210435	0.412933975092099
2	Country:United-State...	18572	0.42509932919131066	0.420411471621832
3	Capital Gain:0 Sex:M...	18403	0.4558685931612361	0.426076766371525
4	Race:White Sex:Male	18038	0.42572320277126924	0.423620208300338
5	Martial Status:Marrie...	14065	0.910932952582142	0.550126781740506
6	Relationship:Husband	12463	0.8286640898823089	0.545121073337460
7	Education:Bachelors ...	4447	0.3200467377198837	0.466045201100538
8	Education-Num:13 C...	4447	0.3200467377198837	0.466045201100538
9	Occupation:Exec-ma...	3992	0.35705422893501	0.487203650308566
10	Occupation:Exec-ma...	3735	0.3533075572630059	0.487246645442360
11	Occupation:Exec-ma...	3703	0.35195537318923575	0.488045106154043
12	Occupation:Exec-ma...	3595	0.3739122326416609	0.496045184178784
13	Education:Bachelors ...	3522	0.37801869240709374	0.487545347812801
14	Education-Num:13 S...	3522	0.37801869240709374	0.487545347812801

Figure 4.3: Slice Finder **visualization tools help the user quickly browse through problematic slices by effect size and by slice size on a scatter plot (A) and see slice summary by hovering over any point (B); the user can sort slices by any metrics and select on the scatter plot view or on the table view. The selections are highlighted on the linked views (C). The user can also explore top-k large problematic slices by different effect size threshold using the slider (*min_eff_size*) on the bottom left corner (D).**

Here, V is the number of false discoveries and R the number of total discoveries returned by the procedure. Slice Finder uses α-investing, mainly because it allows more interactive multiple hypothesis error control with an unspecified number of tests in any order. On the contrary, more restricted multiple hypothesis error control techniques, such as Bonferroni correction and Benjamini-Hochberg procedure [61] fall short as they require the total number of tests m in advance or become too conservative as m grows large.

There are different α-investing policies for testing a sequence of hypotheses. In particular, we use an exploration strategy called β-farsighted [127] does not assume any order of the slices and tests hypotheses believed most likely to be rejected. We test the seemingly more significant slices with more power, and continue testing the rest only if we have left over α-wealth. The successful discovery of significant slices earns extra testing power (alpha-wealth), helping us to continue testing until there is no remaining wealth.

4.2.3 Interactive Visualization Tool

Slice Finder interacts with users through the GUI in Figure 4.3. **A**: On the left side is a scatter plot that shows the (size, effect size) coordinates of all slices. This gives a nice overview of top-k problematic slices, which allows the user to quickly browse through large and also problematic slices and compare slices to each other. **B**: Whenever the user hovers a mouse over a dot, the slice

description, size, effect size, and metric (e.g., log loss) are displayed next to it. If a set of slices are selected, their details appear on the table on the right-hand side, **C**: On the table view, the user can sort slices by any metrics on the table.

On the bottom, **D**: Slice Finder provides configurable sliders for adjusting k and T. Slice Finder materializes all the problematic slices ($\phi \geq T$) as well the non-problematic slices ($\phi < T$) searched already. If T decreases, then we just need to reiterate the slices explored until now to find the top-K slices. If T increases, then the current slices may not be sufficient, depending on k, so we continue searching the slice lattice. This is possible because Slice Finder looks for top-k problematic slices in a top-down manner.

4.3 Using Slice Finder For Model Fairness

In this section, we look at model fairness as a use case of Slice Finder where identifying problematic slices can be a preprocessing step before more sophisticated analysis on fairness on the slices.

As machine learning models are increasingly used in sensitive applications, such as predicting whether individuals will default on loans [128], commit crime [129], or survive intensive hospital care [130], it is essential to make sure the model performs equally well for all demographics to avoid discrimination. However, models may fail this property for various reasons: bias in data collection, insufficient data for certain slices, limitations in the model training, to name a few cases.

Model fairness has various definitions depending on the application and is thus non-trivial to formalize (see recent tutorial [131]). While many metrics have been proposed [128, 132–134], there is no widely-accepted standard, and some definitions are even at odds. In this paper, we focus on a relatively common definition, which is to find of data where the model performs relatively worse using some of these metrics, which fits nicely into the Slice Finder framework.

Using our definition of fairness, Slice Finder can be used to quickly identify interpretable slices that have fairness issues without having to specify the sensitive features in advance. Here, we demonstrate how Slice Finder can be used to find any unfairness of the model with equalized odds [128]. Namely, we explain how our definition of problematic slice using effect size also conforms to the definition of equalized odds. Slice Finder is also generic and supports any fairness metric that can be expressed as a scoring function. Any subsequent analysis of fairness on these slices can be done afterwards.

Equalized odds requires a predictor $\hat{Y}$ (e.g., a classification model h in our case) to be independent of protected or sensitive feature values $A \in \{0, 1\}$ (e.g., gender = Male or gender = Female) conditional on the true outcome Y [128]. In binary classification ($y \in \{0, 1\}$), this is equivalent to:

$$Pr\{\hat{Y} = 1 | A = 0, Y = y\} = Pr\{\hat{Y} = 1 | A = 1, Y = y\} \tag{4.1}$$

Notice that equalized odds is essentially matching true positive rates (tpr) in case of $y = 1$ or false negative rates (fnr) otherwise.

Slice Finder can be used to identify slices where the model is potentially discriminatory; an ML practitioner can easily identify feature dimensions of the data, without having to manually

consider all feature value pair combinations, on which a deeper analysis and potential model fairness adjustments are needed. The problematic slices with $\phi > T$ suffer from higher loss (lower model accuracy in case of log loss) compared to the counterparts. If one group is enjoying a better rate of accuracy over the other, then it is a good indication that the model is biased. Namely, accuracy is a weighted sum of *tpr* and *fnr* by their proportions, and thus, a difference in accuracy means there are differences in *tpr* and false positive rate ($fpr = 1 - tpr$), assuming there are any positive examples. As equalized odds requires matching *tpr* and *fpr* between the two demographics (a slice and its counterpart), Slice Finder using log loss ψ can identify slices to show that the model is potentially discriminatory. In case of the gender = Male slice above, we flag this as a signal for discriminatory model behavior because the slice is defined over a sensitive feature and has a high effect size.

There are other standards, but equalized odds ensures that the prediction is non-discriminatory with respect to a specified protected attribute (e.g., gender), without sacrificing the target utility (i.e., maximizing model performance) too much [128].

4.4 Experiments

In this section, we compare the two Slice Finder approaches (decision tree and lattice search) with the baseline (clustering). For the clustering approach, we use the k-means algorithm. We address the following key questions:

- How accurate and efficient is Slice Finder ?
- What are the trade-offs between the slicing techniques?
- What is the impact of adjusting the effect size threshold T?
- Are the identified slices interpretable enough to understand the model's performance?
- How effective is false discovery control using α-investing?

4.4.1 Experimental Setup

We used the following two problems with different datasets and models to compare how the three different slicing techniques – lattice search (LS), decision tree (DT), and clustering (CL) – perform in terms of recommended slice quality as well as their interpretability.

- *Census Income Classification:* We trained a random forest classifier to predict whether income exceeds \$50K/yr based on the UCI census data [135]. There are 15 features and 30K examples.
- *Credit Card Fraud Detection:* We trained a random forest classifier to predict fraudulent transactions among credit card transactions [136]. This dataset contains transactions that occurred over two days, where we have 492 frauds out of 284k transactions (examples), each

with 29 features. Because the data set is heavily imbalanced, we first undersample non-fraudulent transactions to balance the data. This leaves a total of 984 transactions in the balanced dataset.

As we shall see, the two datasets – Census Income and Credit Card Fraud – have different characteristics and are thus useful for comparing the behaviors of the decision tree and lattice search algorithms. In addition, we also use a synthetic dataset when necessary. The main advantage of using synthetic data is that it gives us more insights into the operations of Slice Finder . In Sections 4.4.2–4.4.6, we assume that all slices are statistically significant for simplicity and separately evaluate statistical significance in Section 4.4.7.

Accuracy Measure: Since problematic slices may overlap, we define *precision* to be the fraction of examples in the union of the slices identified by the algorithm being evaluated that also appear in actual problematic slices. Similarly, *recall* is defined as the fraction of the examples in the union of actual problematic slices that are also in the identified slices. Finally, *accuracy* is the harmonic mean of precision and recall.

4.4.2 Problematic Slice Identification

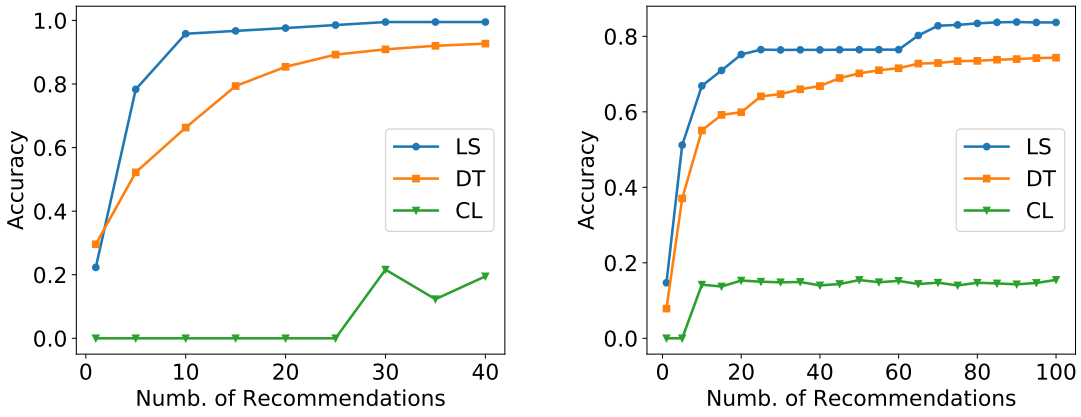
An important question to answer is whether Slice Finder can indeed find the most problematic slices, in the user's point of view. Unfortunately for the real datasets, we do not know what are the true problematic slices, which makes our evaluation challenging. Instead, we *add new problematic slices* by randomly perturbing labels and focus on finding those slices. While Slice Finder may find both new and existing problematic slices, our evaluation will only be whether Slice Finder finds the new problematic slices.

We first experiment on a synthetic dataset and compare the performances of LS, DT, and CL. We then experiment on the real datasets and show similar results.

Synthetic Data

We generate a simple synthetic dataset where the generated examples have two discretized features F_1 and F_2 and can be classified into two classes – 0 and 1 – perfectly. We make the model use this decision boundary and do not change it further. Then we add problematic slices by choosing random possibly-overlapping slices of the form $F_1 = A$, $F_2 = B$, or $F_1 = A \wedge F_2 = B$. For each slice, we flip the labels of the examples with 50% probability. Note that this perturbation results in the worst model accuracy possible.

Figure 4.4(a) shows the accuracy comparison of LS, DT, and CL on synthetic data. As the number of recommendations increases, LS consistently has a higher accuracy than DT because LS is able to better pinpoint the problematic slices including overlapping ones while DT is limited in the sense that it only searches non-overlapping slices. For CL, we only evaluated the clusters with effect sizes at least T. Even so, the accuracy is much lower than those of LS and DT.



(a) Synthetic data

(b) Census Income Data

Figure 4.4: Accuracy comparison of finding problematic slices using (a) synthetic data and (b) real data.

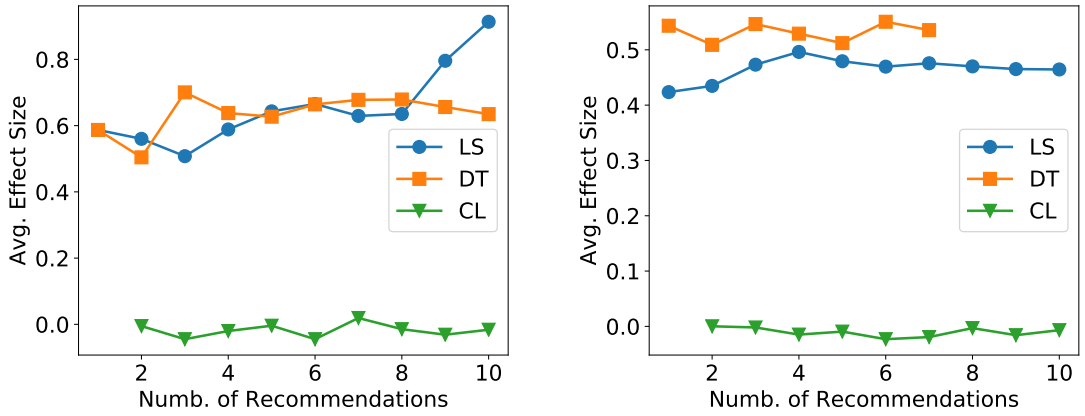
Real Data

We also perform a similar experiment using the Census Income dataset where we generate new problematic slices on top of the existing data by randomly choosing slices and flipping labels with 50% probability. Compared to the synthetic data, the existing data may also have problematic slices, which we do not evaluate because we do not know what they are. Figure 4.4(b) shows similar comparison results between LS, DT, and CL. The accuracies of LS and DT are lower than those in the synthetic data experiments because some of the identified slices may be problematic slices in the existing data, but are considered incorrect when evaluated.

4.4.3 Large Problematic Slices

Figures 4.5 and 4.6 show how LS and DT outperform CL in terms of average slice size and average effect size on the real datasets. CL starts with the entire dataset where the number of clusters (i.e., recommendations) is 1. CL produces large clusters that have very low effect sizes where the average is around 0.0 and sometimes even negative, which means some slices are not problematic. The CL results show that grouping similar examples does not necessarily guide users to problematic slices. In comparison, LS and DT find smaller slices with effect sizes above the threshold $T = 0.4$.

LS and DT show different behaviors depending on the given dataset. When running on the Census Income data, both LS and DT are able to easily find up to $k = 10$ problematic slices with similar effect sizes. Since LS generally has a larger search space than DT where it also considers overlapping slices, it is able to find larger slices as a result. When running on the Credit Card Fraud data, DT has a harder time finding enough problematic slices. The reason is that DT initially finds a large problematic slice, but then needs to generate many levels of the decision tree to find additional



(a) Census Income Data

(b) Credit Card Fraud Data

Figure 4.5: Effect size comparisons between different data slicing approaches ($T = 0.4$).

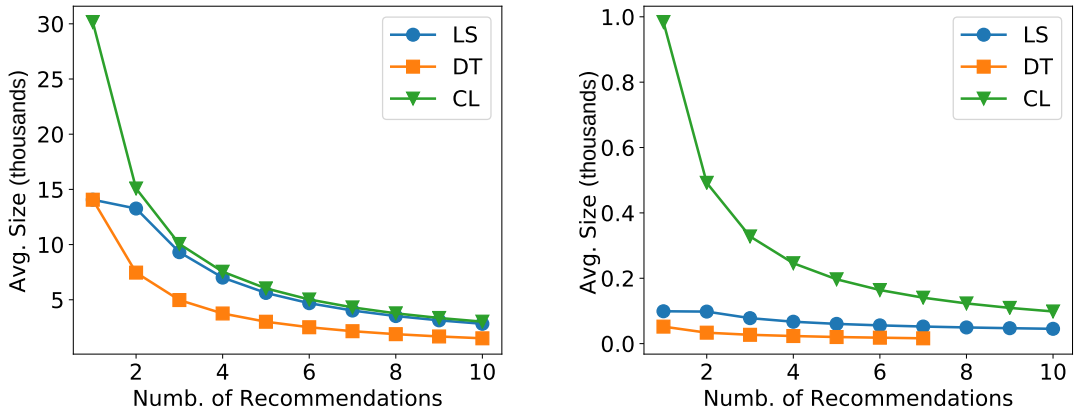
problematic slices because it only considers non-overlapping slices. Since a decision tree is designed to partition data to minimize impurity, the slices found deeper down the tree tend to be smaller and "purer," which means the problematic ones have higher effect sizes. Lastly, DT could not find more than 7 problematic slices because the leaf nodes were too small to split further. These results show that, while DT may search a level of a decision tree faster than LS searching a level of a lattice, it may have to search more levels of the tree to make the same number of recommendations.

4.4.4 Adjusting Effect Size Threshold T

Figure 4.7 shows the impact of adjusting the effect size threshold T on LS and DT. For a low T value, there are more slices that can be problematic. Looking at the Census Income data, LS indeed finds larger slices than those found by DT, although they have relatively smaller effect sizes as a result. As T increases, LS is forced to search smaller slices that have high-enough effect sizes. Since LS still has a higher search space than DT, it does find slices with higher effect sizes when T is at least 0.4. The Credit Card Fraud data shows a rather different comparison. For small T values, recall that DT initially finds a large problematic slice, which means the average size is high, and the effect size small. As T increases, DT has to search many levels of the decision tree to find additional problematic slices. These additional slices are much smaller, which is why there is an abrupt drop in average slice size. However, the slices have higher effect sizes, which is why there is also a corresponding jump in the average effect size.

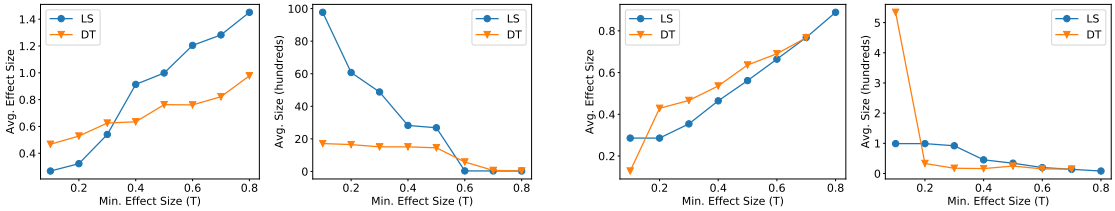
4.4.5 Scalability

We evaluate the scalabilities of LS and DT against different sample fractions, degree of parallelization, and the number of top-k slices to recommend. All experiments were performed on the Census



(a) Census Income Data (b) Credit Card Fraud Data

Figure 4.6: Average slice size (unit is 1000) comparisons between different data slicing approaches ($T = 0.4$).



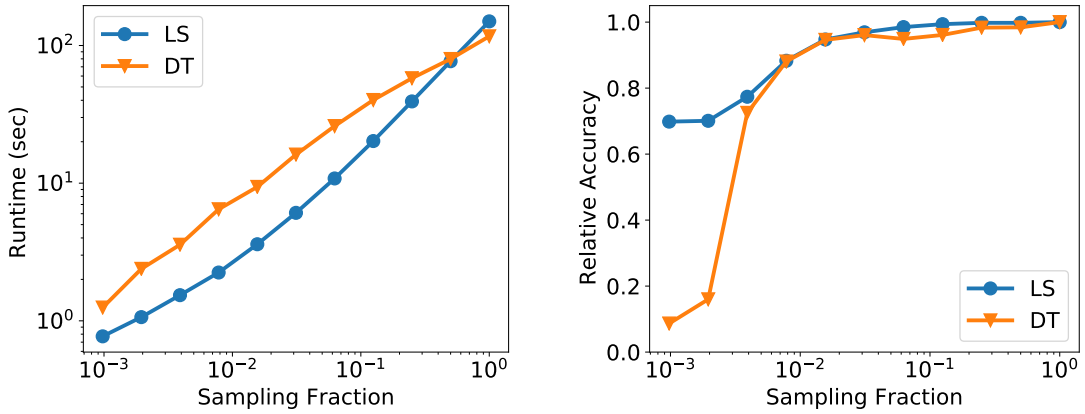
(a) Census Income Data (b) Credit Card Fraud Data

Figure 4.7: The impact of adjusting the effect size threshold T on average slice size and average effect size.

Income dataset.

Figure 4.8 shows how the runtimes of LS and DT change versus the sampling fraction. For both algorithms, the runtime increases almost linearly with the sample size. We also measure the relative accuracy of the two algorithms where we compare the slices found in a sample with the slices found in the full dataset. For a sample fraction of 1/128, both LS and DT maintain a high relative accuracy of 0.88. These results show that it is possible to find most of the problematic slices using a fraction of the data, about two orders of magnitude faster.

Figure 4.11(a) illustrates how Slice Finder can scale with parallelization. LS can distribute the evaluation (e.g., effect size computation) of the slices with the same number of filter predicates to multiple workers. As a result, for the full Census Income data, increasing the number of workers results in better runtime. Notice that the marginal runtime improvement decreases as we add more workers. The results for DT are not shown here because the current implementation does not support parallel DT model training.



(a) (b)

Figure 4.8: Slice Finder **(LS, DT) runtime (on a single node) and accuracy results using different sample sizes (Census Income data).**

Figure 4.11(b) compares the runtimes of LS and DT when the number of top-k recommendations increase. For small k values less than 5, DT is faster because it searchers fewer slices to find k problematic ones. However, as k increases, DT needs to search through many levels of a decision tree and starts to run relatively slower than LS. Meanwhile, LS only searches the next level of the lattice if k is at least 70 at which point DT is again relatively faster. Thus, whether LS or DT is faster depends on k and how frequently problematic slices occur.

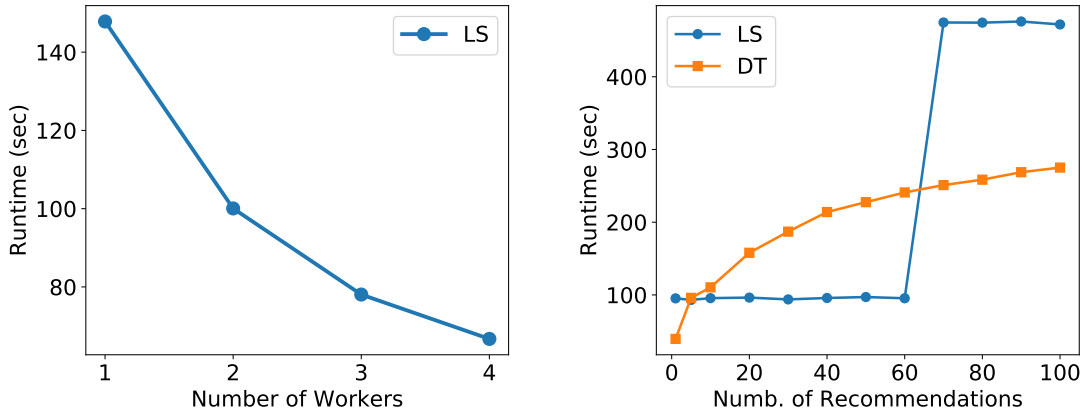
4.4.6 Interpretability

An important feature of Slice Finder is that it can find interpretable slices that can help a user understand and describe the model's behavior using a few common features. A user without Slice Finder may have to go through all the misclassified examples (or clusters of them) manually to see if the model is biased or failing.

Table 4.1 shows top-5 problematic slices from the two datasets using LS and DT. Looking at the top-5 slices found by LS from the Census Income data, the slices are easy to interpret with a few number of common features. We see that the Marital Status = Married-civ-spouse slice has the largest size as well as a large effect size, which indicates that the model can be improved for this slice. It is also interesting to see that the model fails for the people who are husbands or wives, but not for other relationships: own-child, not-in-family, other-relative, and unmarried. We also see slices with high capital gains tend to be problematic in comparison to the common case where the value is 0. In addition, the top-5 slices found by DT from the Census Income data can also be interpreted in a straightforward way, although having more literals makes the interpretation more tedious. Finally, the top-5 slices from the Credit Card Fraud data are harder (but still reasonable) to interpret because many feature names are anonymized (e.g., V14).

Slice	# Literals	Size	Effect Size
LS slices from Census Income data			
Marital Status = Married-civ-spouse	1	14065	0.58
Relationship = Husband	1	12463	0.52
Relationship = Wife	1	1406	0.46
Capital Gain = 3103	1	94	0.87
Capital Gain = 4386	1	67	0.94
DT slices from Census Income data			
Marital Status = Married-civ-spouse	1	14065	0.58
Marital Status $\neq$ Married-civ-spouse $\rightarrow$ Capital Gain $\geq$ 7298 $\rightarrow$ Capital Gain $<$ 8614 $\rightarrow$ Education-Num $<$ 13	4	7	0.58
Marital Status $\neq$ Married-civ-spouse $\rightarrow$ Capital Gain $<$ 7298 $\rightarrow$ Education-Num $\geq$ 13 $\rightarrow$ Age $\geq$ 28 ... $\rightarrow$ Hours per week $\geq$ 44	5	855	0.43
Marital Status $\neq$ Married-civ-spouse $\rightarrow$ Capital Gain $<$ 7298 $\rightarrow$ Education-Num $\geq$ 13 $\rightarrow$ Age $<$ 28 ... $\rightarrow$ Capital Loss $\geq$ 2231	5	5	1.07
Marital Status $\neq$ Married-civ-spouse $\rightarrow$ Capital Gain $<$ 7298 $\rightarrow$ Education-Num $\geq$ 13 $\rightarrow$ Age $\geq$ 28 ... $\rightarrow$ Hours per week $<$ 44 $\rightarrow$ Education-Num $\geq$ 15	6	101	0.47
LS slices from Credit Card Fraud data			
V14 = -3.69 – -1.00	2	98	0.45
V7 = 0.94 – 23.48 $\wedge$ V10 = -2.16 – -0.87	3	29	0.41
V1 = 1.13 – 1.74 $\wedge$ V25 = 0.48 – 0.71	4	28	0.54
V7 = 0.94 – 23.48 $\wedge$ Amount = 270.54 – 4248.34	4	28	0.53
V10 = -2.16 – -0.87 $\wedge$ V17 = 0.92 – 6.74	5	27	0.44
DT slices from Credit Card Fraud data			
V14 $<$ -2.17 $\rightarrow$ V10 $\geq$ -1.52	2	31	0.60
V14 $\geq$ -2.17 $\rightarrow$ V4 $\geq$ 0.76 $\rightarrow$ V12 $<$ -0.42	3	59	0.48
V14 $\geq$ -2.17 $\rightarrow$ V4 $<$ 0.76 $\rightarrow$ V14 $<$ -0.93 $\rightarrow$ V2 $<$ 1.04	4	23	0.42
V14 $\geq$ -2.17 $\rightarrow$ V4 $<$ 0.76 $\rightarrow$ V14 $\geq$ -0.93 $\rightarrow$ Amount $\geq$ 320	4	18	0.52
V14 $\geq$ -2.17 $\rightarrow$ V4 $\geq$ 0.76 $\rightarrow$ V12 $\geq$ -0.42 $\rightarrow$ Amount $\geq$ 1 $\rightarrow$ V17 $\geq$ 1.68	5	6	0.63

Table 4.1: Top-5 slices found by LS and DT from the Census Income and Credit Card Fraud datasets. When denoting a slice from a decision tree, we use the $\rightarrow$ notation to order the literals by level.



(a) Parallelization

(b) # Recommendations

Figure 4.9: (a) Slice Finder runtime results with increasing number of (a) parallel workers and (b) recommendations (Census Income data).

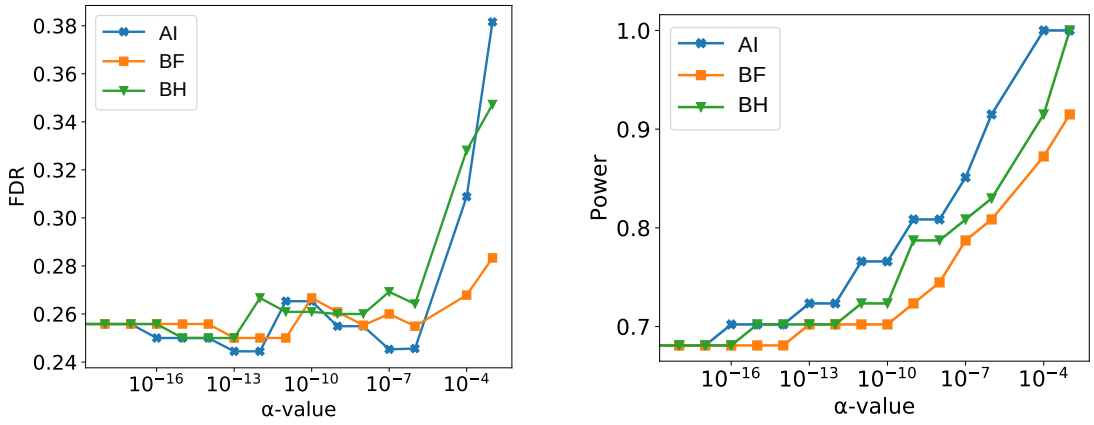
4.4.7 False Discovery Control

Even for a small data set (or sample), there can be an overwhelming number of problematic slices. The goal of Slice Finder is to bring the user's attention to a handful of large problematic slices; however, if the sample size is small, most slices would contain fewer examples, and thus, it is likely that many slices and their effect size measures are seen by chance. In such a case, it is important to prevent false discoveries (e.g., non-problematic slices appear as problematic where $\phi \geq T$ due to sampling bias). For evaluation, we use the Census Income data and compare the results of Bonferroni correction (BF), the Benjamini-Hochberg procedure (BH), and α-investing (AI) using two standard measures: *false discovery rate*, which was described in Section 4.4.7, and *power* [**?**], which is the probability that the tests correctly reject the null hypothesis.

The results in Figure 4.10 show that, as the α-value (or wealth when using AI) increases up to 0.01, AI and BH have higher FDR results than BH, but higher power results as well. When measuring the accuracy of slices, AI slightly outperforms both BH and BF because it invests its α more effectively using the Best-foot-forward policy. In comparison, BF is conservative and has a high false-discovery rate (which results in lower accuracy), and BH does not exploit the fact that the earlier slices are more likely to be problematic as AI does. The more important advantage of AI is that it is the only technique that works in an interactive setting.

4.5 Related Work

In practice, the overall performance metrics can mask the issues on a more granular-level, and it is important to validate the model accordingly on smaller subsets/sub-populations of data (slices). While a well-known problem, the existing tools are still primitive in that they rely on domain experts



(a) False Discovery Rate

(b) Power

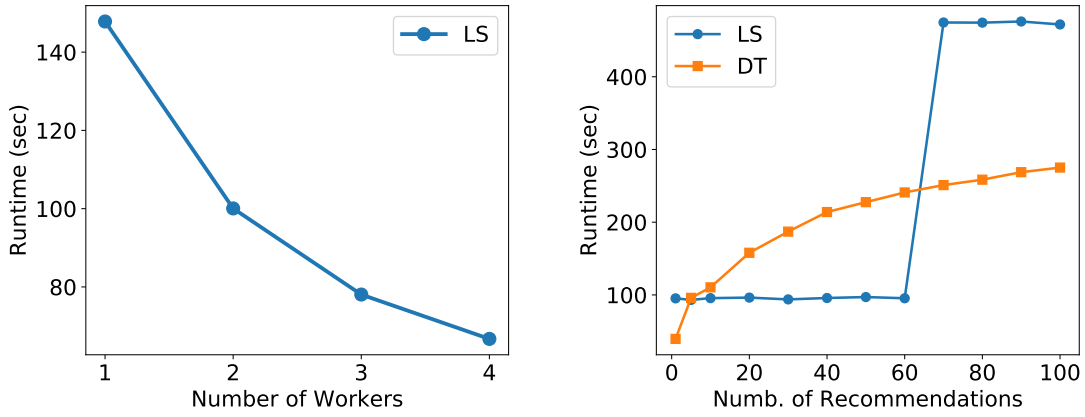
Figure 4.10: (a) False discovery rate and (b) power comparison of the Bonferroni, Benjamini Hochberg, and α-investing techniques (Census Income data).

to pre-define important slices. State-of-art tools for ML model validation include Facets [137], which can be used to discover bias in the data, TensorFlow Model Analysis (TFMA), which slices data by an input feature dimension for a more granular performance analysis [56], and MLCube [55], which provides manual exploration of slices and can both evaluate a single model or compare two models. While the above tools are manual, Slice Finder complements them by automatically finding slices useful for model validation.

There are also several other relevant lines of work related to this problem, and here we list the most relevant work to Slice Finder.

Data Exploration: Online Analytical Processing (OLAP) has been tackling the problem of slicing data for analysis, and the techniques deal with the problem of large search space (i.e., how to efficiently identify data slices with certain properties). For example, Smart Drilldown [138] proposes an OLAP drill down process that returns the top-K most "interesting" rules such that the rules cover as many records as possible while being as specific as possible. Intelligent rollups [139] goes the other direction where the goal is to find the broadest cube that share the same characteristics of a problematic record. In comparison, Slice Finder finds data slices, on which the model under-performs, without having to evaluate the model on all the possible slices. This is different from general OLAP operations based on cubes with pre-summarized aggregates, and the OLAP algorithms cannot be directly used.

Model Understanding: Understanding a model and its behavior is a broad topic that is being studied extensively [58, 59, 124, 140–142]. For example, LIME [58] trains interpretable linear models on local data and random noise to see which feature are prominent. Anchors [59] are high-precision rules that provide local and sufficient conditions for a black-box model to make predictions. In comparison, Slice Finder is a complementary tool to provide part of the data where the model is



(a) Parallelization

(b) # Recommendations

Figure 4.11: (a) Slice Finder runtime results with increasing number of (a) parallel workers and (b) recommendations (Census Income data).

performing relatively worse than other parts. As a result, there are certain applications (e.g., model fairness) that benefit more from slices. PALM [57] isolates a small set of training examples that have the greatest influence on the prediction by approximating a complex model into an interpretable meta-model that partitions the training data and a set of sub-models that approximate the patterns within each pattern. PALM expects as input the problematic example and a set of features that are explainable to the user. In comparison, Slice Finder finds slices with high effective sizes and does not require any user input. Influence functions [143] have been used to compute how each example affects model behavior. In comparison, Slice Finder identifies interpretable slices instead of individual examples. An interesting direction is to extend influence functions to slices, to quantify the impact of each slice on the overall model quality.

Feature Selection: Slice Finder is a model validation tool, which comes after model training. It is important to note that this is different from feature selection [3, 144] in model training, where the goal is often to identify and (re-)train on the most correlated features (dimensions) to the target label (i.e., finding representative features that best explain model predictions). Instead, Slice Finder identifies a few common feature values that describe subsets of data with significantly high error concentration for a given model; this, in turn, could help the user to interpret hidden model performance issues that are masked by good overall model performance metrics.

4.6 Summary

We have proposed Slice Finder as a tool for efficiently finding large, problematic, and interpretable slices. The techniques are relevant to model validation in general, but also to model fairness and fraud detection where human interpretability is critical to understand model behavior. We have proposed

two methods for automated data slicing for model validation: decision tree training, which is efficient and finds slices defined as ranges of values, and slice lattice search, which can find overlapping slices and are more effective for categorical features. We also provide an interactive visualization front-end to help user quickly browse through a handful of problematic slices.

In the future, we would like to improve Slice Finder to better discretize numeric features and support the merging of slices. We would also like to deploy SliceFinder to products and conduct a user study on how helpful the slices are for explaining and debugging models.

Chapter 5

Conclusions

In this thesis, we present several cases for the uncertainty in data exploration (QUDE). First, the uncertainty in the form of missing unknown data items is addressed in Chapter 2. The initial work was done to estimate the impact of unknown unknowns on simple aggregate queries; later, the proposed techniques were expanded to work with arbitrary feature space (i.e., more than a single attribute) to define unknown unknowns in the context of machine learning. Chapter 3 discusses the number of undetected data errors in a data set as a data quality metric. The quality of data itself is an important source of uncertainty around any data exploration pipeline; being able to quantify the quality of data is an important step forward. Chapter 4 presents an automated data slicing tool (Slice Finder) for model validation. Model validation in general is critical to minimize uncertainty around the model performance. This work focuses on a common mistake in model validation, and Slice Finder can help user quickly identify problematic slices, which can be masked by a good overall performance. In addition to what we have addressed in this thesis, there are also other types of uncertainty problems, such as multiple comparisons problem [127] and Simpson's paradox [145] in interactive data exploration, that were also addressed in the QUDE project.

Our overarching goal is to quantify all types of uncertainty in data analysis and exploration, and in turn, provide measures to correct and validate the analysis results and discoveries. As such, QUDE is part of a larger system, called "Northstar" [18]: an interactive data science system. The goal of Northstar is to advanced analytics more accessible and, thus, democratize data science. The techniques developed in QUDE can protect users without deep statistical and ML backgrounds form making some of the common mistakes in data exploration.

Appendix A

Appendix for Chapter 2

A.1 Symbol table

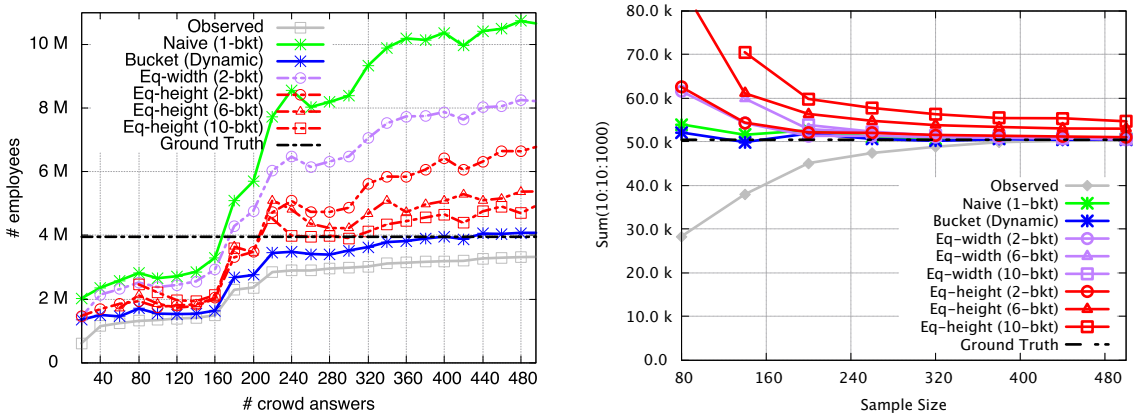
Ω	Universe of all valid entities (unknown size)
r	A valid unique entity or data item
D	Ground truth or the underlying population
S	Observed sample of size $n = \|S\|$, with duplicates
K	Integrated database with only unique entities from S
U	*Unknown unknowns* that exist in D, but not in S or K
M_0	*Unknown unknowns* distribution mass in D
c	The number of unique data items in S; $c = \|K\|$
s_j	Source j with $n_j = \|s_j\|$ data items
N	The size of the ground truth; $N = \|D\|$
ϕ	The aggregated query result: e.g., ϕ_D (over D)
Δ	*The impact of unknown unknowns*: $\Delta = \phi_D - \phi_K$
f_j	A frequency statistic, i.e., the number of data items with exactly j occurrences in S.
F	The set of frequency statistics, $\{f_1, f_2, ..., f_n\}$
ρ	The correlation between publicity and value distributions, i.e., *publicity-value correlation*
γ	Coefficient of variance (data skew measure)
C	Sample coverage, also $C = 1 - M_0$

Table A.1: Symbols used in Chapter 2

A.2 Static Bucket Based Estimator

In Section 2.2.3, we state that the optimal number of buckets depends on the underlying *publicity* distribution. Here, we elaborate on this with the two examples.

Figure A.1(a) shows the US tech-sector employment estimates by various estimators: *Naive* (1-bucket), *Bucket* (a.k.a., *Dynamic Bucket*), and *Static Bucket* (Eq-width and Eq-height). In



(a) US Tech Employment

(b) Uniform publicity, uncorrelated to values

Figure A.1: (a) The best US tech-sector employment estimation with static buckets. Splitting into more buckets improves estimation. Eq-width (6-bkt, 10-bkt) are missing due to some of the buckets are empty; (b) Sum(1:10:1000) estimation with static buckets. Splitting less (e.g., *Naive*) improves estimation. Data points are missing when some buckets contain *singletons* only (i.e., infinite estimation).

this particular example, splitting into more buckets improves estimation, as the underlying *publicity* distribution is skewed and correlated to the values (i.e., larger companies are more well known).

In contrast, in the simulated case in Figure A.1(b), splitting into less (e.g., *Naive*) improves estimation as the underlying *publicity* is uniform. Notice, that in both examples above, the *bucket* estimator yields the best estimates, dynamically resizing buckets on its own.

Also notice, that we consider two variants of static buckets: the one described in the paper, **equi-width**, which divides the observed value range into a fixed number of buckets, and another obvious variant, **equi-height**, which divides the observed sample, sorted by value, evenly into a fixed number of buckets. Both static bucket types are simple to use, but they require parameter tuning for the optimal number of buckets, which is hard to predict without knowing the true *publicity* distribution.

A.3 The Increase in Count Estimate After Bucket Split

In equation 2.14, we claimed that the count estimation ($\hat{N}_{Chao92} = nc/(n-f_1)$) of a bucket increases after splitting the bucket, if data items are evenly distributed over the *attribute* value range, and there is no *publicity-value correlation*:

$$\hat{N}_{Chao92} = \frac{c}{1-f_1/n} = \overbrace{\frac{n\cdot c}{n-f_1}}^{\text{Before split}}$$
$$\leq \underbrace{\frac{\frac{n}{2}\cdot\frac{c}{2}}{\frac{n}{2}-\alpha\cdot f_1} + \frac{\frac{n}{2}\cdot\frac{c}{2}}{\frac{n}{2}-(1-\alpha)\cdot f_1}}_{\text{After split}}$$

The α parameter governs the split of the original singleton count (f_1) into a pair of smaller buckets. We assume n and c are evenly distributed between the split buckets, as items are evenly distributed over the value range, and all values are equally likely (no *value-publicity correlation*). We now show that the above inequality holds by showing that the right hand side (after split) is minimized at $nc/(n - f_1)$. Note that $nc/(n - f_1)$ is a positive number as $n \geq f_1 \geq 0$ and $c \geq 0$.

To find the minimum, we take the first derivative of the right hand side (denoted by $\mathcal{R}$) with respect to α:

$$\mathcal{R}' = \frac{-c \cdot f_1 \cdot n}{4(-(1-\alpha) \cdot f_1 + \frac{n}{2})^2} + \frac{-c \cdot f_1 \cdot n}{4(-\alpha \cdot f_1 + \frac{n}{2})^2}$$

Solving $\mathcal{R}' = 0$, we get $\alpha = 0.5$; we have $\mathcal{R}(0.5) = nc/(n - f_1)$ as shown below:

$$\begin{aligned}\mathcal{R}(0.5) &= \frac{\frac{n}{2} \cdot \frac{c}{2}}{\frac{n}{2} - 0.5 \cdot f_1} + \frac{\frac{n}{2} \cdot \frac{c}{2}}{\frac{n}{2} - (1-0.5) \cdot f_1} \\ &= \frac{\frac{n}{2} \cdot \frac{c}{2} + \frac{n}{2} \cdot \frac{c}{2}}{\frac{n}{2} - 0.5 \cdot f_1} = \frac{n \cdot c}{n - f_1}\end{aligned}$$

Finally, we show $\mathcal{R}(0.5) = nc/(n - f_1)$ is the minimum by ensuring $\mathcal{R}''(0.5) > 0$:

$$\begin{aligned}\mathcal{R}'' &= \frac{c \cdot f_1^2 \cdot n}{2(-(1-\alpha) \cdot f_1 + \frac{n}{2})^3} + \frac{c \cdot f_1^2 \cdot n}{2(-\alpha \cdot f_1 + \frac{n}{2})^3} \\ \mathcal{R}''(0.5) &= \frac{c \cdot f_1^2 \cdot n}{2(-(1-0.5) \cdot f_1 + \frac{n}{2})^3} + \frac{c \cdot f_1^2 \cdot n}{2(-0.5 \cdot f_1 + \frac{n}{2})^3} \\ &= \frac{c \cdot f_1^2 \cdot n}{(-0.5 \cdot f_1 + \frac{n}{2})^3} = \frac{8c \cdot f_1^2 \cdot n}{(-f_1 + n)^3}\end{aligned}$$

Note that $n \geq f_1$, and this makes $\mathcal{R}'' > 0$; $\mathcal{R}$ is minimized at $nc/(n - f_1)$ and the inequality holds true:

$$\overbrace{\frac{n \cdot c}{n - f_1}}^{\text{Before split}} \leq \underbrace{\frac{\frac{n}{2} \cdot \frac{c}{2}}{\frac{n}{2} - \alpha \cdot f_1} + \frac{\frac{n}{2} \cdot \frac{c}{2}}{\frac{n}{2} - (1-\alpha) \cdot f_1}}_{\text{After split}}$$

A.4 Other Unknown Unknowns Estimators

Many proposed techniques can be combined: we can use the *frequency* estimator, instead of the *naïve* estimator, with the *bucket* (i.e., *Dynamic Bucket* approach) estimator or the *Monte-Carlo* estimator. We can also combine the *Monte-Carlo* estimator with the *bucket* estimator.

However, as the *Monte-Carlo* estimator requires large sample sizes to be accurate, combining it with *bucket* estimator often results in lower estimation quality (i.e., each bucket contains a smaller sample). Furthermore, each bucket (a smaller value range) entails a part of the underlying *publicity* distribution; hence, the *publicity* distribution per bucket appears more uniform. As a major

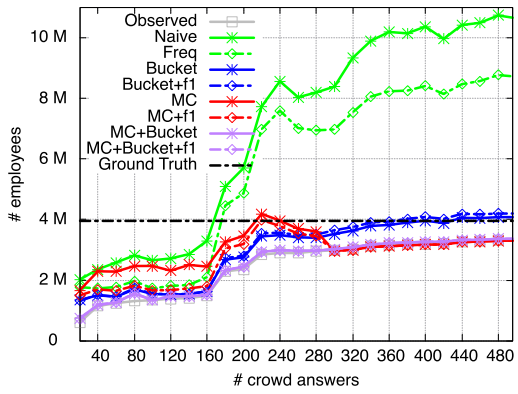
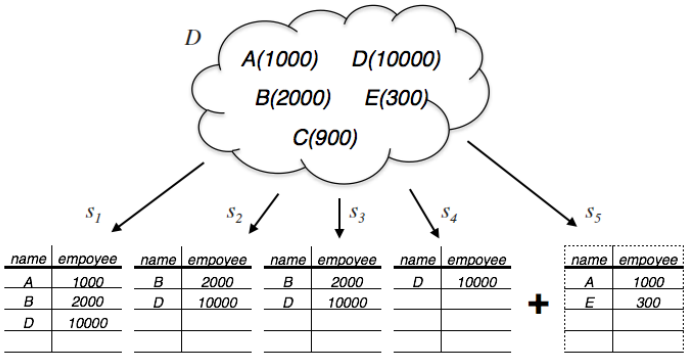


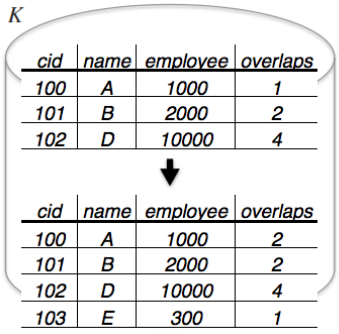
Figure A.2: The best US tech-sector employment estimation with other estimators

drawback, the *Monte-Carlo* estimator exhibits a tendency to favor its count estimate $\hat{N}_{MC} \sim c$ (see Section 2.5.1). Such tendency gets more imminent in *Monte-Carlo* with *Bucket* estimator as seen in Figure A.2. Similarly, we found that the difference between the *naïve* and *frequency* estimators is not significant for the *bucket* estimator (i.e., uniform *publicity*).

A.5 A Toy Example For Unknown Unknowns



(a) Multiple sources s_i sampled without replacement from the unknown population D. s_5 is added later to the original integrated database.



(b) Integrated Database K, before (top) and after (bottom) adding s_5

Figure A.3: A toy example for `SELECT SUM(employee) FROM K`

In this section, we walk through the different estimators step by step using a simple toy example. Again, we use the same query, `SELECT SUM(employee) FROM K`, from the introduction but over a very simplistic data set, shown in Figure A.3. It should be noted, that this toy example can not convey any statistical properties because of its small size, but we can explain the general reasoning behind the techniques using the example.

Figure A.3 shows the data integration scenario of our example. We assumes that the ground truth D consists of 5 companies $\{A, B, C, D, E\}$ (the bubble on the top), with different numbers of

	before adding s_5 ($n = 7$, $c = 3$, $f_1 = 1$, $\hat{\gamma}^2 = 0.1667$)	after adding s_5 ($n = 10$, $c = 4$, $f_1 = 1$, $\hat{\gamma}^2 = 0$)
Ground Truth	$\phi_D = 1000 + 2000 + 900 + 10000 + 300 = 14200$	
Observed	$\phi_K = 1000 + 2000 + 10000 = 13000$	$1000 + 2000 + 10000 + 300 = 13300$
$\phi_K + \Delta_{naive}$	$= \phi_K + \frac{\phi_K \cdot f_1 \cdot \left(c + \hat{\gamma}^2 n\right)}{c \cdot (n - f_1)}$ $= 13000 + \frac{13000 \cdot 1 \cdot (3 + 0.1667 \cdot 7)}{3 \cdot (7 - 1)}$ ≈ 16009	$= 13300 + \frac{13300 \cdot 1 \cdot (4 + 0 \cdot 9)}{4 \cdot (9 - 1)}$ ≈ 14962
$\phi_K + \Delta_{freq}$	$= \phi_K + \frac{\phi_{f_1} \left(c + \hat{\gamma}^2 n\right)}{n - f_1}$ $= 13000 + \frac{1000\,(3 + 0.1667 \cdot 7)}{7 - 1}$ ≈ 13694	$= 13300 + \frac{300\,(4 + 0 \cdot 9)}{9 - 1}$ $= 13450$
$\phi_K + \Delta_{bucket}$	$= \phi_K + \Delta_{b_1:\{A,B\}} + \Delta_{b_2:\{D\}}$ $= \phi_K + \{\Delta_{naive}\}_{b_1} + \{\Delta_{naive}\}_{b_2}$ $= 13000 + \frac{3000 \cdot 1 \cdot (2 + 0 \cdot 3)}{2 \cdot (3 - 1)}$ $+ \frac{10000 \cdot 0 \cdot (1 + 0 \cdot 4)}{1 \cdot (4 - 0)}$ $= 14500$	$= \phi_K + \Delta_{b_1:\{A,E\}} + \Delta_{b_2:\{B\}}$ $+ \Delta_{b_3:\{D\}}$ $= \phi_K + \{\Delta_{naive}\}_{b_1} + \{\Delta_{naive}\}_{b_2}$ $+ \{\Delta_{naive}\}_{b_3}$ $= 13300 + \frac{1300 \cdot 1 \cdot (2 + 0 \cdot 3)}{2 \cdot (3 - 1)}$ $+ \frac{2000 \cdot 0 \cdot (1 + 0 \cdot 2)}{1 \cdot (2 - 0)}$ $+ \frac{10000 \cdot 0 \cdot (1 + 0 \cdot 4)}{1 \cdot (4 - 0)}$ $= 13950$

Table A.2: `SELECT SUM(employee) FROM K` **results with different *unknown unknowns* estimators;** *bucket* **estimator gives the most accurate estimation of ϕ_D.**

employees (e.g., company A has 1000, whereas company B has 2000). In the beginning we have four data sources $\{s_1, s_2, s_3, s_4\}$ and they sample without replacement from D. For instance data source s_1 lists companies A, B, and D. In the example we also assume a *publicity-value correlation*; that is, the biggest company D appears in all data sources ($\{s_1, s_2, s_3, s_4\}$), while smaller companies appear in fewer sources. To show how the estimates improve, we assume that the data source s_5 is added later on (visualized through the plus). The tables in Figure A.3(b) show the integrated database before (top) and after (bottom) adding the fifth data source. For convenience, the last column shows the number of times each company was observed.

Table A.2 shows the estimates by different estimators before and after adding the fifth data source. We exclude *Monte-Carlo* estimator due to its simulation based nature. The top row contains the relevant statistics of K. For instance, with 4 data sources, the number of observed items / sample size is $n = 7$, the number of observed unique items is $c = 3$ (i.e., companies A, B, and D from the top table in Figure A.3(b)), the number of singletons $f_1 = 1$ (i.e., company D as it is the only company, which was observed exactly ones across the data sources). and the calculated *coefficient of variation* (CV) $\gamma = 0.1667$ calculated over the sample.

Before adding the fifth data source, the observed total sum is $\phi_K = 1000 + 2000 + 10000 = 13000$, after adding the fifth data source $\phi_K = 1000 + 2000 + 10000 = 13300$. In this example, the observed

total sum does not converge to the ground truth of 14200

Table A.2 shows the values with calculations for the different estimators. As it can be seen, the naìve estimator performs the worse; the estimator is quite far off, especially with 4 data sources. The reason is the value estimator (*mean substitution*) used. The average number of employees is $\phi_K/3 \approx 4333$. Thus all missing companies (i.e., *unknown unknowns*) are also assumed to be that big. Now knowing that bigger companies are more likely to be sampled, now the naìve estimator heavily over-estimates.

In contrast, the *frequency* estimator performs much better than the naìve estimator because it assumes that the missing companies have the average value over *singletons*, which includes A, but not the extremely big company D; the missing companies are assume to have a value of $\phi_{f_1}/1 = 1000$. Because less popular companies are more likely to be smaller (i.e., the *publicity-value correlation*), this yields to a much better estimate.

Finally, the *bucket* estimator performs the best. Before adding the fifth source, the algorithm creates two buckets: $b_1 : \{A, B\}$ and $b_2 : \{D\}$. The estimate quality of *bucket* persists even after we add s_5 (i.e., *Bucket* is the best). In this case, the *bucket* estimator generates $b_1 : \{A, E\}$, $b_2 : \{B\}$ and $b_3 : \{D\}$. The *bucket* estimator automatically groups the small companies (A and E) together and uses their average number of employees for the missing companies (all other buckets have unknown count estimation of 0); in this example, the *bucket* estimator has a smoothed value in between 300 and 1000. This is particularly more desirable compared to the case of the *frequency* estimator: E is the new one and only *singleton* and ϕ_{f_1} is now 300.

Appendix B

Appendix for Chapter 3

B.1 Symbol table

Q	A relation Q where some records refer to the same real-world entity.
R	A data set with a set of N records $\{r_1, ..., r_N\}$.
R_{dirty}	A subset $R_{dirty} \subseteq R$ of errorneous records.
W	A set of K workers $\{w_1, ..., w_K\}$
$S^{(i)}_{dirty}$	A set of items marked as dirty/error by w_i.
$S^{(i)}_{clean}$	A set of items marked as clean by w_i.
$\mathcal{I}$	a $N \times K$ matrix where every column represents the answers from a worker k, and which entries are $\{1, 0, \emptyset\}$ denoting dirty, clean, unseen respectively.
n_i	number of votes (worker responses) on item i. n_i^+ refers to the number of positive (dirty) votes on i and n_i^+ the number of clean votes.
c	The number of unique errors observed so far from R.
D	The true number of unique errors exist in R.
ξ	the expected number of switches needed for the current majority consensus to reach D. $\xi = \xi^+ + \xi^-$, where ξ^+ is the number of positive switches (clean to dirty) and ξ^- for negative switches.
f_j	A frequency statistic, i.e., the number of data items with exactly j occurrences in S.
γ	Coefficient of variance (data skew measure)
C	Sample coverage.
H	a function $H : R \mapsto \mathbb{R}_+$ that is a measure of confidence that a record is erroneous.
R_H	a set of ambiguous records selected by the heuristic $R_H = \{\forall r \in R : \alpha \leq H(r) \leq \beta\}$.

Table B.1: Symbols used in Chapter 3

Appendix C

Appendix for Chapter 4

C.1 Symbol table

D	a data set with n examples $D = \{(x_F^{(1)}, y^{(1)}), (x_F^{(2)}, y^{(2)}), ..., (x_F^{(n)}, y^{(n)})\}$.
F	features $F = \{F_1, F_2, ..., F_m\}$ where each feature F_j (e.g., country) has a list of values (e.g., {US, DE}) or discretized numeric value ranges (e.g., {[0, 50), [50, 100)}).
h	a machine learning model under testing.
S	A slice is a subset of examples in D with common features and can be described as a conjunction of the common feature-value pairs $\bigwedge_j F_j$ *op* v_j.
S'	A complment slice of S, $S' = D - S$.
ψ	A classification loss function $\psi(S, h)$ that returns a performance score for a set of examples by comparing h's prediction $h(x_F^{(i)})$ with the true label $y^{(i)}$.
H	A hypothesis in problematic slice testing.
ϕ	a standardized score ϕ of the difference by the pooled standard deviations of $\psi(S, h)$ and $\psi(S', h)$, σ_S and $\sigma_{S'}$ respectively (a.k.a., *effect size*)
T	An effect size threshold
K	Top-K large problematic slices to return.
V	The number of false discoveries (problematic slices).
R	The number of total discoveries (problematic slices).
fpr	false positive rate, which is V/R.
tpr	true positive rate, which is also $tpr = 1 - fpr$.

Table C.1: Symbols used in Chapter 4

Bibliography

[1] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of SIGMOD*, pages 2201–2206, 2016. doi: 10.1145/2882903.2912574. URL http://doi.acm.org/10.1145/2882903.2912574.

[2] Jason W Osborne. *Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data*. Sage, 2012.

[3] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[4] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1137–1145, 1995. URL http://ijcai.org/Proceedings/95-2/Papers/016.pdf.

[5] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[6] Çağatay Demiralp, Peter J Haas, Srinivasan Parthasarathy, and Tejaswini Pedapati. Foresight: Rapid data exploration through guideposts. *arXiv preprint arXiv:1709.10513*, 2017.

[7] Xin Luna Dong and Divesh Srivastava. Big data integration. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1245–1248. IEEE, 2013.

[8] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, Purnamrita Sarkar, and Venketaram Ramachandran. Crowdsourcing enumeration queries: Estimators and interfaces. In *to appear at IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2015.

[9] Yeounoh Chung, Michael L. Mortensen, Carsten Binnig, and Tim Kraska. Estimating the impact of unknown unknowns on aggregate query results. In *Proceedings of ACM SIGMOD*, pages 861–876, 2016.

[10] Yeounoh Chung, Michael L. Mortensen, Carsten Binnig, and Tim Kraska. Estimating the impact of unknown unknowns on aggregate query results. *TODS*, 43, 2018.

[11] Yeounoh Chung, Sanjay Krishnan, and Tim Kraska. A data quality metric (dqm): how to estimate the number of undetected errors in data sets. *Proceedings of the VLDB Endowment*, 10(10):1094–1105, 2017.

[12] Yeounoh Chung, Tim Kraska, Steven Euijong Whang, and Neoklis Polyzotis. Slice finder: Automated data slicing for model interpretability. *arXiv preprint arXiv:1807.06068*, 2018. URL `http://arxiv.org/abs/1807.06068`.

[13] The 6 components of open-source data science/ machine learning ecosystem; did python declare victory over r? `https://www.kdnuggets.com/2018/06/ecosystem-data-science-python-victory.html`, 2018.

[14] Lessons from next-generation data wrangling tools. `https://www.oreilly.com/ideas/lessons-from-next-generation-data-wrangling-tools`, 2015.

[15] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, George Beskales, Mitch Cherniack, Stanley B Zdonik, Alexander Pagan, and Shan Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[16] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016. URL `http://www.vldb.org/pvldb/vol9/p993-abedjan.pdf`.

[17] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Vizdom: interactive analytics through pen and touch. *Proceedings of the VLDB Endowment*, 8(12): 2024–2027, 2015.

[18] Tim Kraska. Northstar: An interactive data science system. *Proc. VLDB Endow.*, 11(12): 2150–2164, August 2018. ISSN 2150-8097. doi: 10.14778/3229863.3240493. URL `https://doi.org/10.14778/3229863.3240493`.

[19] Hector Gonzalez, Alon Y Halevy, Christian S Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *Proceedings of ACM SIGMOD*, pages 1061–1066, 2010.

[20] Google. Freebase. `https://www.freebase.com`, 2015. Accessed: 2015-07-08.

[21] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, April 2011. ISSN 0001-0782.

[22] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: Answering queries with crowdsourcing. In *Proceedings of ACM SIGMOD*, pages 61–72, 2011.

[23] Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *Proceedings of CIDR*, pages 211–214, 2011.

[24] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Demonstration of qurk: a query processor for humanoperators. In *Proceedings of ACM SIGMOD*, pages 1315–1318, 2011.

[25] Aditya Parameswaran and Neoklis Polyzotis. Answering Queries using Humans, Algorithms and Databases. In *Proceedings of CIDR*, 2011.

[26] Sihem Amer-Yahia, AnHai Doan, Jon Kleinberg, Nick Koudas, and Michael Franklin. Crowds, clouds, and algorithms: Exploring the human side of "big data" applications. In *Proceedings of ACM SIGMOD*, pages 1259–1260, 2010. ISBN 978-1-4503-0032-2.

[27] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: Exploiting crowds for accurate real-time image search on mobile phones. In *Proc. of MobiSys*, pages 77–90, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5.

[28] Daniel Haas, Matthew Greenstein, Kainar Kamalov, Adam Marcus, Marek Olszewski, and Marc Piette. Reducing error in context-sensitive crowdsourced tasks. In *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.

[29] Matteo Magnani and Danilo Montesi. A survey on uncertainty management in data integration. *Journal of Data and Information Quality*, 2(1):1–33, 2010.

[30] Paul D Allison. Handling missing data by maximum likelihood. In *SAS global forum*, pages 1–21, 2012.

[31] Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

[32] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[33] Pew Research Center. How u.s. tech-sector jobs have grown, changed in 15 years. `http://pewrsr.ch/PtqZDA`, 2014. Accessed: 2015-07-08.

[34] Beth Trushkowsky, Tim Kraska, Michael J. Franklin, and Purnamrita Sarkar. Crowdsourced enumeration queries. In *Prodeedings of ICDE*, pages 673–684, 2013.

[35] David A McAllester and Robert E Schapire. On the convergence rate of good-turing estimators. In *COLT*, pages 1–6, 2000.

[36] Leo Pipino, Yang W. Lee, and Richard Y. Wang. Data quality assessment. *Commun. ACM*, 45(4):211–218, 2002. doi: 10.1145/505248.5060010. URL `http://doi.acm.org/10.1145/505248.5060010`.

[37] You-Wei Cheah and Beth Plale. Provenance quality assessment methodology and framework. *J. Data and Information Quality*, 5(3):9:1–9:20, 2015. doi: 10.1145/2665069. URL `http://doi.acm.org/10.1145/2665069`.

[38] Adir Even and Ganesan Shankaranarayanan. Dual assessment of data quality in customer databases. *J. Data and Information Quality*, 1(3), 2009. doi: 10.1145/1659225.1659228. URL `http://doi.acm.org/10.1145/1659225.1659228`.

[39] Valerie Sessions and Marco Valtorta. Towards a method for data accuracy assessment utilizing a bayesian network learning algorithm. *J. Data and Information Quality*, 1(3), 2009. doi: 10.1145/1659225.1659227. URL `http://doi.acm.org/10.1145/1659225.1659227`.

[40] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011. doi: 10.1109/TKDE.2010.154. URL `http://dx.doi.org/10.1109/TKDE.2010.154`.

[41] Kimberly Keeton, Pankaj Mehra, and John Wilkes. Do you know your iq?: a research agenda for information quality in systems. *SIGMETRICS Performance Evaluation Review*, 37(3): 26–31, 2009. doi: 10.1145/1710115.1710121. URL `http://doi.acm.org/10.1145/1710115.1710121`.

[42] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011. URL `http://portal.acm.org/citation.cfm?id=1952378&CFID=12591584&CFTOKEN=15173685`.

[43] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of SIGMOD*, pages 143–154, 2005.

[44] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005.

[45] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *PVLDB*, pages 315–326, 2007.

[46] Andrei Lopatenko and Loreto Bravo. Efficient approximation algorithms for repairing inconsistent databases. In *Proceedings of ICDE*, pages 216–225, 2007.

[47] Jiannan Wang, Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proceedings of SIGMOD*, pages 469–480, 2014.

[48] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.

[49] Adam Marcus and Aditya Parameswaran. Crowdsourced data management industry and academic perspectives. *Foundations and Trends in Databases*, 2015.

[50] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012. URL `http://vldb.org/pvldb/vol5/p1483_jiannanwang_vldb2012.pdf`.

[51] Beth Trushkowsky, Tim Kraska, and Purnamrita Sarkar. Answering enumeration queries with the crowd. *Commun. ACM*, 59(1):118–127, 2016. doi: 10.1145/2845644. URL `http://doi.acm.org/10.1145/2845644`.

[52] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *KDD*, pages 1387–1395, 2017.

[53] F. Doshi-Velez and B. Kim. Towards A Rigorous Science of Interpretable Machine Learning. *ArXiv e-prints*, February 2017.

[54] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *KDD*, pages 1222–1230, 2013.

[55] Minsuk Kahng, Dezhi Fang, and Duen Horng Polo Chau. Visual exploration of machine learning results using data cube analysis. In *HILDA*, page 1. ACM, 2016.

[56] Introducing tensorflow model analysis. `https://medium.com/tensorflow/introducing-tensorflow-model-analysis-scaleable-sliced-and-full-pass-\\metrics-5cde7baf0b7b`, 2018.

[57] Sanjay Krishnan and Eugene Wu. Palm: Machine learning explanations for iterative debugging. In *HILDA*, pages 4:1–4:6, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5029-7. doi: 10.1145/3077257.3077271. URL `http://doi.acm.org/10.1145/3077257.3077271`.

[58] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016. doi: 10.1145/2939672.2939778. URL `http://doi.acm.org/10.1145/2939672.2939778`.

[59] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[60] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[61] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B (Methodological)*, 57(1):289–300, 1995. URL `http://dx.doi.org/10.2307/2346101`.

[62] Daniela Florescu, Daphne Koller, and Alon Y. Levy. Using probabilistic information in data integration. In *Proceedings of VLDB*, pages 216–225, 1997. ISBN 1-55860-470-7.

[63] Ulf Leser and Felix Naumann. Query planning with information quality bounds. *Flexible Query Answering Systems*, pages 85–94, 2001.

[64] Felix Naumann, Johann-Christoph Freytag, and Ulf Leser. Completeness of integrated information sources. *Inf. Syst.*, 29(7):583–615, September 2004. ISSN 0306-4379.

[65] Mat Tis Neiling and Hans-Joachim Lenz. Data integration by means of object identification in information systems. In *Proceedings of European Conference on Information Systems*, page 69, 2000.

[66] Weiyi Meng, King-Lup Liu, Clement Yu, Wensheng Wu, and N Naphtali Rishe. Estimating the usefulness of search engines. In *Proceedings of ICDE*, pages 146–153, 1999.

[67] Barna Saha and Divesh Srivastava. Data quality: The other face of big data. In *Proceedings of ICDE*, pages 1294–1297, 2014.

[68] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? In *Proceedings of the VLDB*, pages 97–108, 2012.

[69] Anne Chao and Shen-Ming Lee. Estimating the Number of Classes via Sample Coverage. *Journal of the American Statistical Association*, 87(417):210–217, 1992.

[70] John Bunge and M Fitzpatrick. Estimating the number of species: a review. *Journal of the American Statistical Association*, 88(421):364–373, 1993.

[71] Anne Chao. Species estimation and applications. In *Encyclopedia of Statistical Sciences, 2nd Edition*, pages 7907–7916. Wiley, New York, 2005.

[72] Irving J. Good. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, 40(3/4):237–264, 1953.

[73] K. P. Burnham and W. S. Overton. Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals. *Biometrika*, 65(3):625–633, 1978.

[74] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[75] Ralph B D'Agostino Jr and Donald B Rubin. Estimating and using propensity scores with partially missing data. *Journal of the American Statistical Association*, 95(451):749–759, 2000.

[76] Yang C Yuan. Multiple imputation for missing data: Concepts and new development (version 9.0). *SAS Institute Inc, Rockville, MD*, 2010.

[77] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[78] Michael Lexa. Useful facts about the kullback-leibler discrimination distance. `https://scholarship.rice.edu/bitstream/handle/1911/20061/Lex2004Dec8UsefulFact.PDF?sequence=1&isAllowed=y`, 2004. Accessed:2015-07-08.

[79] Michael Fu et al. *Handbook of simulation optimization*, volume 216. Springer, 2015.

[80] Anne Chao. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of statistics*, pages 265–270, 1984.

[81] Anne Chao. Estimating the population size for capture-recapture data with unequal catchability. *Biometrics*, pages 783–791, 1987.

[82] Wikipedia. List of u.s. states by gdp. `https://en.wikipedia.org/wiki/List_of_U.S._states_by_GDP`, 2015. Accessed: 2015-07-08.

[83] Nicholas J Gotelli and Robert K Colwell. Estimating species richness. *Biological diversity: frontiers in measurement and assessment*, 12:39–54, 2011.

[84] Anne Chao and Tsung-Jen Shen. Nonparametric estimation of shannon's index of diversity when there are unseen species in sample. *Environmental and ecological statistics*, 10(4):429–443, 2003.

[85] Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952.

[86] Anne Chao and Mark CK Yang. Stopping rules and estimation for recapture debugging with unequal failure rates. *Biometrika*, 80(1):193–201, 1993.

[87] Samuel Zahl. Jackknifing an index of diversity. *Ecology*, 58(4):907–913, 1977.

[88] Harshana Rajakaruna, D Andrew R Drake, Farrah T Chan, and Sarah A Bailey. Optimizing performance of nonparametric species richness estimators under constrained sampling. *Ecology and Evolution*, 6(20):7311–7322, 2016.

[89] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of VLDB*, pages 311–322, 1995.

[90] Anne Chao. Nonparametric Estimation of the Number of Classes in a Population. *SJS*, 11(4), 1984.

[91] Gregory Valiant and Paul Valiant. Estimating the unseen: an n/log (n)-sample estimator for entropy and support size, shown optimal via new clts. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 685–694. ACM, 2011.

[92] Jie Liang. Estimation Methods for the Size of Deep Web Textural Data Source: A Survey. cs.uwindsor.ca/richard/cs510/survey_jie_liang.pdf, 2008. Accessed: 2015-07-08.

[93] Jianguo Lu and Dingding Li. Estimating deep web data source size by capture—recapture method. *Inf. Retr.*, 13(1):70–95, February 2010. ISSN 1386-4564.

[94] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Towards estimation error guarantees for distinct values. In *Proceedings of ACM PODS*, pages 268–279, 2000.

[95] Roger Sapsford. *Survey research*. Sage, 2006.

[96] James T McClave, P George Benson, and Terry Sincich. *Statistics for business and economics*. Pearson Essex, 2014.

[97] Leslie Kish. *Survey sampling*. John Wiley and Sons, 1965.

[98] Willis Lang, Rimma V Nehme, Eric Robinson, and Jeffrey F Naughton. Partial results in database systems. In *Proceedings of ACM SIGMOD*, pages 1275–1286, 2014.

[99] Simon Razniewski, Flip Korn, Werner Nutt, and Divesh Srivastava. Identifying the extent of completeness of query answers over partially complete databases. In *Proceedings of ACM SIGMOD*, pages 561–576, 2015.

[100] Frank Olken and Doron Rotem. Simple random sampling from relational databases. In *In proceedings of VLDB*, volume 86, pages 25–28, 1986.

[101] Peter J Haas. *Hoeffding Inequalities for Join Selectivity Estimation and Online Aggregation*. IBM, 1996.

[102] John Rice. *Mathematical statistics and data analysis*. Cengage Learning, 2006.

[103] For big-data scientists, 'janitor work' is key hurdle to insights. https://nyti.ms/2jNUfHo, 2014.

[104] Yuchen Zhang, Xi Chen, Dengyong Zhou, and Michael I. Jordan. Spectral methods meet EM: A provably optimal algorithm for crowdsourcing. In *Proceedings of NIPS*, pages 1260–1268, 2014. URL http://papers.nips.cc/paper/5431-spectral-methods-meet-em-a-provably-optimal-algorithm-for\-crowdsourcing.

[105] Rense Lange and Xavier Lange. Quality control in crowdsourcing: An objective measurement approach to identifying and correcting rater effects in the social evaluation of products and services. In *AAAI Spring Symposium: Wisdom of the Crowd*, pages 32–37, 2012. URL http://www.aaai.org/ocs/index.php/SSS/SSS12/paper/view/4322.

[106] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of SIGMOD*, pages 601–612, 2014. doi: 10.1145/2588555.2588576. URL http://doi.acm.org/10.1145/2588555.2588576.

[107] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *PVLDB*, pages 311–322, 1995. URL http://www.vldb.org/conf/1995/P311.PDF.

[108] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of EuroSys*, pages 29–42, 2013.

[109] Anne Chao and Shen-Ming Lee. Estimating the number of classes via sample coverage. *J. the American statistical Association*, 87(417):210–217, 1992.

[110] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of SIGMOD*, pages 61–72, 2011. doi: 10.1145/1989323.1989331. URL http://doi.acm.org/10.1145/1989323.1989331.

[111] Qiang Liu, Jian Peng, and Alexander T. Ihler. Variational inference for crowdsourcing. In *Proceedings of NIPS*, pages 701–709, 2012. URL http://papers.nips.cc/paper/4627-variational-inference-for-crowdsourcing.

[112] David Oleson, Alexander Sorokin, Greg P Laughlin, Vaughn Hester, John Le, and Lukas Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. *Human computation*, 11(11), 2011.

[113] Yasser Altowim, Dmitri V. Kalashnikov, and Sharad Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014. URL http://www.vldb.org/pvldb/vol7/p999-altowim.pdf.

[114] Steven Euijong Whang and Hector Garcia-Molina. Incremental entity resolution on rules and data. *VLDB J.*, 23(1):77–102, 2014. doi: 10.1007/s00778-013-0315-0. URL http://dx.doi.org/10.1007/s00778-013-0315-0.

[115] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. Progressive duplicate detection. *TKDE*, 27(5):1316–1329, 2015. doi: 10.1109/TKDE.2014.2359666. URL http://dx.doi.org/10.1109/TKDE.2014.2359666.

[116] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. Incremental record linkage. *PVLDB*, 7(9), 2014. URL http://www.vldb.org/pvldb/vol7/p697-gruenheid.pdf.

[117] Arvid Heise, Gjergji Kasneci, and Felix Naumann. Estimating the number and sizes of fuzzy-duplicate clusters. In *Proceedigns of CIKM*, 2014. doi: 10.1145/2661829.2661885. URL http://doi.acm.org/10.1145/2661829.2661885.

[118] Gail M. Sullivan and Richard Feinn. Using effect size—or why the p value is not enough. *Journal of Graduate Medical Education*, 4(3):279–282, 2012.

[119] Jacob Cohen. Statistical power analysis for the behavioral sciences . hilsdale. *NJ: Lawrence Earlbaum Associates*, 2, 1988.

[120] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. Flumejava: easy, efficient data-parallel pipelines. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, pages 363–375, 2010. doi: 10.1145/1806596.1806638. URL `http://doi.acm.org/10.1145/1806596.1806638`.

[121] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, pages 1–9, 2011.

[122] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[123] Dean Foster and Bob Stine. Alpha-investing: A procedure for sequential control of expected false discoveries. *Journal of the Royal Statistical Society Series B (Methodological)*, 70(2): 429–444, 2008.

[124] Alex A. Freitas. Comprehensible classification models: A position paper. *SIGKDD Explor. Newsl.*, 15(1):1–10, March 2014. ISSN 1931-0145. doi: 10.1145/2594473.2594475. URL `http://doi.acm.org/10.1145/2594473.2594475`.

[125] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *VLDB*, volume 1215, pages 487–499, 1994.

[126] Anurag Srivastava, Eui-Hong Han, Vipin Kumar, and Vineet Singh. Parallel formulations of decision-tree classification algorithms. In *High Performance Data Mining*, pages 237–261. Springer, 1999.

[127] Zheguang Zhao, Lorenzo De Stefani, Emanuel Zgraggen, Carsten Binnig, Eli Upfal, and Tim Kraska. Controlling false discoveries during interactive data exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 527–540. ACM, 2017.

[128] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *NIPS*, pages 3315–3323, 2016. URL `http://papers.nips.cc/paper/6374-equality-of-opportunity-in-supervised-learning`.

[129] Machine bias. `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-se` 2016.

[130] Marzyeh Ghassemi, Tristan Naumann, Finale Doshi-Velez, Nicole Brimmer, Rohit Joshi, Anna Rumshisky, and Peter Szolovits. Unfolding physiological state: Mortality modelling in intensive

care units. In *KDD*, KDD '14, pages 75–84, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623742. URL http://doi.acm.org/10.1145/2623330.2623742.

[131] Solon Barocas and Moritz Hardt. Fairness in machine learning. *NIPS Tutorial*, 2017.

[132] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *ITCS*, pages 214–226, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1115-1. doi: 10.1145/2090236.2090255. URL http://doi.acm.org/10.1145/2090236.2090255.

[133] Jon M. Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. In *ITCS*, pages 43:1–43:23, 2017. doi: 10.4230/LIPIcs.ITCS.2017.43. URL https://doi.org/10.4230/LIPIcs.ITCS.2017.43.

[134] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *KDD*, pages 259–268, 2015. ISBN 978-1-4503-3664-2.

[135] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, pages 202–207, 1996.

[136] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *SSCI*, pages 159–166. IEEE, 2015.

[137] Facets overview. https://research.googleblog.com/2017/07/facets-open-source-visualization-tool.html, 2017.

[138] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. Interactive data exploration with smart drill-down. In *ICDE*, pages 906–917. IEEE, 2016.

[139] Gayatri Sathe and Sunita Sarawagi. Intelligent rollups in multidimensional olap data. In *VLDB*, pages 531–540, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4. URL http://dl.acm.org/citation.cfm?id=645927.672366.

[140] Paolo Tamagnini, Josua Krause, Aritra Dasgupta, and Enrico Bertini. Interpreting black-box classifiers using instance-level visual explanations. In *HILDA*, pages 6:1–6:6, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5029-7. doi: 10.1145/3077257.3077260. URL http://doi.acm.org/10.1145/3077257.3077260.

[141] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting blackbox models via model extraction. *CoRR*, abs/1705.08504, 2017. URL http://arxiv.org/abs/1705.08504.

[142] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Interpretable & explorable approximations of black box models. *CoRR*, abs/1707.01154, 2017. URL `http://arxiv.org/abs/1707.01154`.

[143] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, pages 1885–1894, 2017. URL `http://proceedings.mlr.press/v70/koh17a.html`.

[144] Moses Charikar, Venkatesan Guruswami, Ravi Kumar, Sridhar Rajagopalan, and Amit Sahai. Combinatorial feature selection problems. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 631–640. IEEE, 2000.

[145] Yue Guo, Carsten Binnig, and Tim Kraska. What you see is not what you get!: Detecting simpson's paradoxes during data exploration. In *Proceedings of the 2Nd Workshop on Human-In-the-Loop Data Analytics*, HILDA'17, pages 2:1–2:5, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5029-7. doi: 10.1145/3077257.3077266. URL `http://doi.acm.org/10.1145/3077257.3077266`.