Scaling Machine Learning Data Repair Systems for Sparse Datasets

by

Omar Attia

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor(s):	Ihab F. Ilyas Professor, David R. Cheriton School of Computer Science University of Waterloo.
Internal Member:	M. Tamer Özsu Professor, David R. Cheriton School of Computer Science University of Waterloo.
Internal Member:	Jimmy Lin Professor, David R. Cheriton School of Computer Science University of Waterloo.

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Machine learning data repair systems (e.g. HoloClean [37]) have achieved state-of-the-art performance for the data repair problem on many datasets. However, these systems face significant challenges with sparse datasets. In this work, the challenges presented by such datasets to machine learning data repair systems are investigated. Dataset-independent methods are presented to mitigate the effects of data sparseness. Finally, experimental results are validated on a large, sparse real-world dataset: Census. Showing that the problem size can be reduced by more than 70%, saving significant computational costs, while still getting high accuracy data repairs (94.5% accuracy).

Acknowledgements

I would like to start by sincerely thanking my advisor: *Ihab Ilyas* for his continuous guidance and the numerous amazing opportunities he provided me. Meeting Ihab after his talk in Alexandria 2017 was a turning point in my life. He believed in me and mentored me -in work and in life- which was essential to enabling this work, and much more. I would also like to thank *Theodoros Rekatsinas* for his valuable input and mentorship during the inductiv days, and now in Apple.

To my parents *Yousry* and *Hanaa*: You have sacrificed the most for my future and supported me through the darkest of times. There is nothing that I can do to repay you, and I will forever be grateful to you.

To my brothers *Abdulrahamn* and *Mahmoud*, and my little sister *Mariam*, no matter how far we are, you are always close to my heart.

I would also like to thank my dear friends: *Amine Mhedhbi*, and *Aly Tarek* for their constant encouragement and support during the writing of this thesis. *Abdelhameed Shalaby* and *Akram Mohamed* for their companionship and always being there for me when I needed it the most. *Ahmed El Bagoury* and *Abdulrahman Ghanem* for the long walks and all the food. And the inductiv team: *Mina Farid*, *Josh McGrath* and *Ryan Clancy* for their camaraderie and hard-work throughout the rough times.

Finally, I would also like to thank my readers: *M. Tamer Özsu* and *Jimmy Lin* for taking the time to read my thesis and provide invaluable feedback.

Dedication

The Earth Is Not A Cold Dead Place [1]

[1] https://www.youtube.com/watch?v=Ziw4yd5R0QI

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In an increasingly data-driven world, wrong and low-quality data can have massive adverse effects on organizations and people alike. According to Gartner [30], low data quality is the reason many organizations lose an average of $15 million per year. In other fields, like health care, low data quality for medical records of a patient or drug information could lead to bad health care decisions that would affect people's lives and well-being.

Data repair is a challenging problem that has been studied extensively in the data management literature (e.g. [8, 10, 12, 35]). The hardness of the problem in real-world datasets is due to many factors: First, error types and causes in data vary (e.g. duplicates, missing values, wrong values) and have many sources (e.g. mistakes in data entry, typos, etc.). Second, repairing these errors in a data source requires understanding a lot of context about the data. This usually requires some human involvement, which -among other causes- affects the scalability of the solutions. Finally, various data quality problems do not come piecemeal. The problems are usually interleaved and affect each other. Still, many solutions were suggested, achieving excellent results on several classes of datasets.

Given the importance of the problem, its impact on businesses and lives, and the relative success of the suggested solutions. Surprisingly, such solutions were not adopted widely in the industry. One of the main reasons behind this is that these solutions do not work well for many real-world datasets, specifically: large and sparse datasets.

This thesis explores the challenges facing state-of-the-art machine learning data repair systems when applied to real-world, large, and sparse datasets. We suggest solutions to address these problems and provide experimental results to quantify our findings on a new dataset: Census.

1.1 Machine Learning for Data Repair

Previous work on data repair included rule-based approaches (e.g. [13, 9]). Rule-based approaches encode the data quality signals and business requirements into a rich set of rules that have expressive inference systems. Errors are then expressed in terms of rules violations, and repairs can be made to make the dataset consistent with the rules (i.e. no violations). These rule-based systems were severely limited and not widely adopted in real-world scenarios. This is mainly due to their significant dependence on humans and domain knowledge in the data quality process (e.g. [50, 13]). In many rule-based systems, humans are needed to craft the rules (e.g. encoding business and data quality rules as conditional functional dependencies in [9]), maintain the rules and keep them up-to-date as data and requirements change, and design the model features and label data (e.g. [50]). That significant dependence on humans drives up the costs and slows down the process.

Machine learning approaches (e.g. [37, 19, 26]) recently showed great progress in comparison to the rule-based approaches in solving data repair problems. This is due to three main reasons:

1. **Holistic Context**: Machine learning systems can combine all signals and contexts about data quality (e.g. rules, statistics). This leads to major improvements and offers desirable features like:

    - **Increased Recall**: Rule-based systems tend to have high precision but low recall (i.e. they fix certain types of errors well but do not have high coverage for many different types of errors). Machine learning models, on the other hand, include holistic contexts and features about the data. This enables machine learning systems to have a significantly higher recall with comparable precision to rule-based systems.
    - **Generalisation**: Learning-based systems can subsume rule-based systems by including statistics and features about the rules and data interactions to the model. For example, HoloClean [37] does this by including the normalized rule violation counts as a feature for each cell. Also, learning systems tend to generalize better to unseen data. The drawback here is that modern machine learning systems usually require a lot of data to generalize well.

2. **Rule Explosion and Maintenance**: Machine learning systems avoid rule explosion. In contrast, rule-based systems tend to include maintenance overhead for a growing repository of rules. This usually requires much time, effort, and human

domain expertise that is hard to automate. Additionally, these rules often get outdated with time (e.g. policy changes or customer behaviour changes). While learning systems tend to get better with time as the system is retrained with more updated observations.

3. **Probabilistic Semantics**: Machine learning systems can communicate confidence of repairs instead of the binary or certain quality semantics that rule-based systems produce. This makes it easier to quantify the quality of cells and the confidence of the system.

These three features gave machine learning data repair systems an edge over the rule-based systems; Enabling a new class of state-of-the-art systems with a fraction of the cost and time of traditional rule-based systems. For example, HoloClean [37] cites an average F1 improvement of more than 2x on various datasets.

1.2 Challenges for Machine Learning in Data Repair

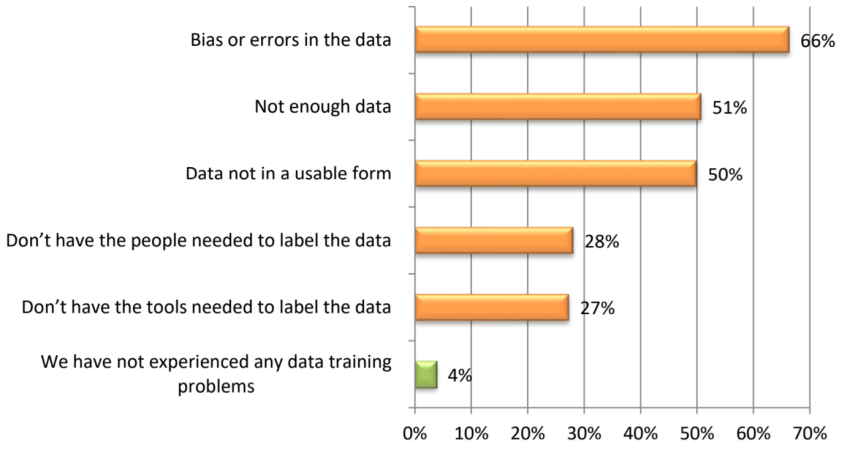


Figure 1.1: Breakdown of Training Data Problems. From [38]

Although machine learning systems have achieved state-of-the-art results for the data repair problem. They are not without drawbacks. For machine learning approaches to perform well and be widely adopted for structured data quality problems, they must solve three hard problems:

1. **Lack of Training Data for Building Supervised Models**:
   In many tasks, modern machine learning models (e.g. deep neural networks) require a lot of labelled data during training to learn useful representations and generalize well [28]. Figure 1.1 from a recent study by Dimensional Research[1] [38] shows that 51% of the surveyed industry machine learning projects run into problems related to lack of training data. The same study claims that most projects need over 100,000 data samples for training to perform well and be deployed with confidence.

   Constructing these large labelled datasets usually requires significant human effort to label the training data. Furthermore, these models tend to be very sensitive to mistakes in the labelling process [40, 15]. Additionally, in the specific case of data repair, the types of errors in the data are not known in advance, and the errors are generally rare events in the dataset; Making it hard to learn the representation of errors.

2. **Data and Feature Sparseness**:
   Data and feature sparseness is a well-known problem in machine learning literature in various settings [34, 41]. Despite this, modern machine learning techniques were still able to achieve success in many tasks on unstructured data (e.g. images [22, 20], audio [6, 33], and free text [53, 25]).

   Structured data is fundamentally different from unstructured data in that it is very sparse. Furthermore, structured data also often includes multiple modalities like categorical, numerical, and textual data. For example, consider a table about products with N categorical columns (e.g. product type, availability, manufacturer, etc.). Each column C_i has a domain size of $|dom(C_i)|$. It is easy to see that the number of possible combinations of value-assignment for each row in this table is $\Pi_{i=1}^{i=N}|dom(C_i)|$. For even the small values of $N = 100$ and $|dom(C_i)| \in [10, 100]$, this space is huge. However, we know from observation that only a small fraction of these possible combinations are valid. This means that the data space is very sparse.

3. **Scaling Learning and Inference to Large Datasets**:
   Large datasets with millions of rows are considered the norm in real-world machine learning and data repair projects. Hence, scaling the training and inference procedures of machine learning approaches to these large datasets with reasonable cost (i.e. time and computational resources) is essential for the success and wide adoption of these techniques. However, modern machine learning solutions can be costly computationally, and that cost increases as the expressiveness and the richness of

[1] https://dimensionalresearch.com

features increase [43]. This makes applying these techniques to large datasets an even more challenging task.

The Trade-off for Machine Learning Data Repair Systems

Solving the three problems at the same time is hard because they are interleaved. Consequently, current machine learning approaches try to solve the first problem (Lack of training data for building supervised models) by using some new techniques, including self-supervised training procedures [37, 49, 4], weak-labelling [36], and data augmentation methods [19]. However, the solution to the first problem using these techniques creates a trade-off between the other two problems (Data sparsity and Scale on large datasets): Solving the sparseness problem using traditional techniques makes it very hard to scale these systems to large datasets. The sparseness problem is traditionally solved by using rich features, and contextual representations [19, 49], as well as adding domain knowledge and rules (e.g. constraints) to the model [2, 29, 37]

These solutions to the sparseness problem make certain homogeneity and density assumptions on the data. This is done by utilizing schema information to encode information about the structure. For instance, in HoloClean [37], the feature weights are learned on the functional dependency level, not on the value level. Similarly, in [49], the attention weights are learned on the positional encoding of the schema columns, not the values themselves. The homogeneity assumptions enabled and encouraged constructing a single, large, and global model on the entire dataset. This is because, under these homogeneity assumptions, more data results in better model accuracy.

Although these assumptions and models worked well for many average-sized and relatively-dense datasets, they still fail for large, sparse datasets. This is because the homogeneity assumptions do not always hold for these datasets. This leads to having local statistics in data islands that are different from the dataset's global statistics—causing a global model trained for the entire dataset to fail. Additionally, self-supervised training on the entire dataset becomes very computationally expensive as the dataset becomes large. This is usually addressed using some sampling techniques (e.g. uniform sampling). However, these sampling techniques perform poorly on sparse datasets due to the data's heterogeneity and sparseness.

1.3 Scaling Machine Learning Data Repair Systems for Sparse Datasets

This thesis proposes methods to solve the outstanding problem of scaling machine learning data repair systems to large and sparse datasets. The suggested solution consists of constructing multiple models per dataset, where each model is specialized in a specific part of the data. We show that data partitioning techniques (correlation-based vertical partitioning and conditional dependence based horizontal partitioning) can build multiple sparse models and enable efficient sampling for training these models. We then experimentally show how these methods can be used to improve the performance of HoloClean on a new dataset: Census.

The rest of this thesis is organized as follows: In Chapter 2, this work's relevant background is presented. Chapter 3 presents a suite of solutions to reduce the effects of data and model sparseness on machine learning data repair systems. Chapter 4 presents the Census dataset and shows the results of implementing these solutions in HoloClean on the Census dataset.

1.4 Notes

Notes and clarifications about the terminology used throughout this thesis:

- The name "HoloClean" is used throughout this work to refer to the open-source implementation [2]. This open-source implementation is primarily based on the same principles as the original paper [37], but it has significant implementation changes. Some of these implementation changes will be discussed in chapter 2.
- Throughout this work, we use the words "data cleaning", "data repair" and "data imputation" interchangeably, since this is the main focus of the HoloClean system.

[2] https://github.com/HoloClean/HoloClean

Chapter 2

Background

2.1 Introduction

This chapter presents relevant background on the data cleaning problem and how machine learning can improve data quality. We also present some of the definitions and measures that we will use in the following chapters. A comprehensive survey of data repair and even more data quality problems can be found in the Data Cleaning book [21].

2.2 The Data Repair Problem

Concretely, the data repair problem can be presented as follows: Given a relational table instance $\mathcal{D}$ defined over a set of attributes (also referred to as columns, or variables) $\mathcal{R}$, an optional set of integrity rules Σ, and a set of potentially wrong cells Ω. The data repair problem is concerned with finding the "correct" value assignment for each cell in Ω.

Different approaches differ in how this correctness is defined. The rule-based systems usually define it as a value with no violations to any rule in Σ. Some other approaches add another constraint that the correction set is minimal in changes to the dataset. Machine learning-based systems usually define correctness as a cost function to be optimized by changing some learnable parameters.

2.3 Integrity Constraints

Integrity constraints (ICs) are an important part of the toolbox for many data management problems. They found their first use in informing the database schema designs (e.g. using functional dependencies to do schema normalization). However, more recently, different kinds of integrity constraints have been used in data quality problems, especially data repair, as they can determine potentially corrupt cells (i.e. the set Ω). They can also select which values can be used for a specific cell (e.g. only include values that do not violate any applicable rules to a specific cell). We focus on three types of constraints that have seen much use in data repair problems: Functional Dependencies (FDs), Conditional Functional Dependencies (CFDs), and Denial Constraints (DCs).

2.3.1 Functional Dependencies

A Functional Dependency (FD) is a constraint between two sets of attribute $\mathcal{Y}$ and $\mathcal{X}$ in a relational schema $\mathcal{R}$ (i.e. two subsets of columns in a table) that defines a mathematical function from $\mathcal{X}$ to $\mathcal{Y}$. In other words, that the attribute subset $\mathcal{Y}$ is a function of the attribute subset $\mathcal{X}$. Concretely, the formal definition can be given as follows:

- Given a relation $\mathcal{R}$, a set of attributes $\mathcal{X}$ in $\mathcal{R}$ is said to functionally determine another set of attributes $\mathcal{Y}$, also in $\mathcal{R}$, (written $\mathcal{X} \longrightarrow \mathcal{Y}$) if, and only if, each $\mathcal{X}$ value in $\mathcal{R}$ is associated with precisely one $\mathcal{Y}$ value in $\mathcal{R}$; $\mathcal{R}$ is then said to satisfy the functional dependency $\mathcal{X} \longrightarrow \mathcal{Y}$.
- Equivalently, the projection $\Pi_{\mathcal{X},\mathcal{Y}}\mathcal{R}$ is a function, i.e. $\mathcal{Y}$ is a function of $\mathcal{X}$, or the values of $\mathcal{Y}$ are determined by the values of $\mathcal{X}$.
- $\mathcal{X}$ is usually referred to as the determinant set and $\mathcal{Y}$ as the dependent set.
- A functional dependency $\mathcal{X} \longrightarrow \mathcal{Y}$ is called trivial if $\mathcal{Y}$ is a subset of $\mathcal{X}$.

From this definition, it can be seen how functional dependencies between two attributes can be used to make data repair decisions about these two columns. For example, for a functional dependency $X \longrightarrow Y$, where X and Y are single attributes in the same relational table $\mathcal{R}$, from definition **??**, it follows that two tuples sharing the same values of X will necessarily have the same values of Y. If the value assignment in one of the rows is $(X = x_1, Y = y_1)$ and another row has $(X = x_1, Y = y_2)$; we can immediately tell that

there is at least one violation between these four cells. If we had the additional information that only the $Y = y_2$ cell assignment is wrong, we could confidently change the assignment $Y = y_2$ to match the first $Y = y_1$. More generally, if the X attribute values are known (say they are x), then the values for the Y attributes corresponding to x can be determined by looking them up in any tuple of $\mathcal{R}$ containing x.

Functional dependencies have multiple properties, the most important ones are Armstrong's Axioms given in the following definition. Concretely, for a functional dependency $\mathcal{X} \longrightarrow \mathcal{Y}$, the following basic properties hold:

- **Reflexivity**: If $\mathcal{Y}$ is a subset of $\mathcal{X}$, then $\mathcal{X} \longrightarrow \mathcal{Y}$.
- **Augmentation**: If $\mathcal{X} \longrightarrow \mathcal{Y}$, then $\mathcal{XZ} \longrightarrow \mathcal{YZ}$.
- **Transitivity**: If $\mathcal{X} \longrightarrow \mathcal{Y}$, and $\mathcal{Y} \longrightarrow \mathcal{Z}$ then $\mathcal{X} \longrightarrow \mathcal{Z}$.

These three properties enable logical inference on a set of functional dependency rules. That is, it enables creating more rules that are logically equivalent to the original set but expressed differently. The closure of a set of functional dependencies $\mathcal{F}$ is the set of all functional dependencies that are logically implied by $\mathcal{F}$ (through the application of Armstrong's Axioms or other correct properties).

2.3.2 Conditional Functional Dependencies

Conditional Functional Dependencies (CFDs) are a generalization of the functional dependencies defined not only with two sets of attributes but also with conditional value assignments. Formally, it is defined as follows:

A conditional functional dependency on a relation $\mathcal{R}$ is a pair $(\mathcal{X} \longrightarrow \mathcal{Y}, \mathcal{T}_p)$, where:

- $\mathcal{X}$, $\mathcal{Y}$ are sets of attributes from $attr(\mathcal{R})$
- $\mathcal{X} \longrightarrow \mathcal{Y}$ is a standard functional dependency, called the embedded FD.
- $\mathcal{T}_p$ is a tableau with all attributes in $\mathcal{X} \cup \mathcal{Y}$, called the pattern tableau, where for each $A \in (\mathcal{X} \cup \mathcal{Y})$ and each tuple $t \in \mathcal{T}_p$, the following holds: $t[A] \in (dom(A) \cup \{\text{“\_”}\})$ where “_” is an unnamed variable, refers to undefined value.

While functional dependencies apply to all the tuples in the relational table, conditional functional dependencies can apply to tuples that match the provided value assignment. For example, for a relational schema of three columns: $\mathcal{R}(Country, Zip, City)$, a usual functional dependency that holds in many countries in the world would be $Zip \longrightarrow City$. However, that is not necessarily the case in some countries in the world. In this case, if we want to make this functional dependency only hold in the USA, we can write it as a conditional functional dependency of the format $[Country =' USA', Zip] \longrightarrow City$. More comprehensive examples and explanations can be found in [9]. Many of the properties that hold for functional dependencies do hold for conditional functional dependencies too. For example, Armstrong's axioms hold for conditional functional dependencies with some changes to account for the value assignment table.

Conditional functional dependencies can be used to make data repairs in a similar way to that of functional dependencies [9], but they are even more useful because they contain value assignments that can be used for inference directly.

2.3.3 Denial Constraints

Denial Constraints (DCs) [14, 5, 10] are a very general class of integrity constraints. The reason they are so general is that they are expressed as conjunctive first-order logic predicates. This makes them not only very expressive but also very easy to implement in SQL using the usual comparison or Boolean operators. Concretely [1]: A denial constraint on a relation $\mathcal{R}$ is defined as follows:

$$\forall t_a, t_b \dots t_n \in \mathcal{R} : \neg(p_1 \wedge p_2 \wedge \dots p_m)$$

$$\text{And } p_i : t_x.A \; \phi \; t_y.B \quad \text{or} \quad p_i : t_x.A \; \phi \; c$$

Where c is a value constant, and ϕ is any operator that takes in values and returns a boolean true/false (e.g. $=, \neq, \leq, \geq \dots$)

In other words, a denial constraint expresses that a set of predicates cannot be true together for any combination of tuples in a relation, where each predicate expresses a relationship between two cells or between a cell and a constant

Denial constraints also have a set of powerful inference rules like other constraint types. For example, Armstrong's Axioms hold for denial constraints with minor changes.

[1] DC definition from: https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/folien/SS17/DP_10_DCs.pdf

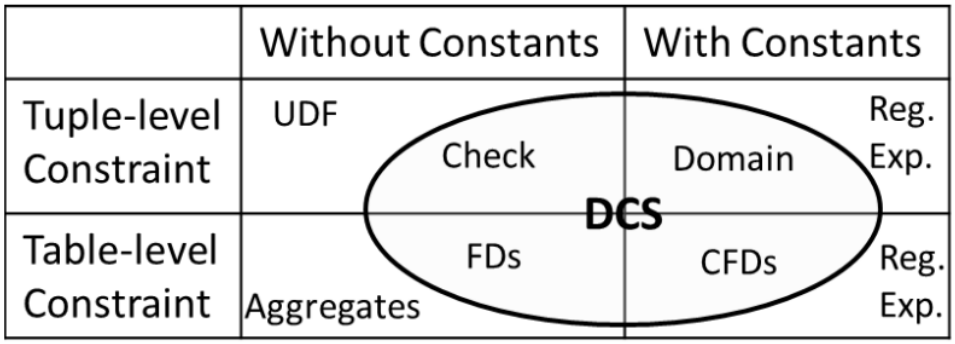


Figure 2.1: Connection between Denial Constraints and the different classes of Integrity Constraints. From [11]

Figure 2.1 (obtained from [11]) shows a classification of different integrity constraints based on two criteria: (i) single tuple level versus table level, and (ii) involvement of constants in the constraint versus only column variables. As illustrated, denial constraints can express other types of integrity constraints such as check constraints, functional dependencies, and conditional functional dependencies.

In HoloClean, the user can specify the dataset's integrity constraints (as Σ) expressed as denial constraints (whether they were: check constraints, functional dependencies, conditional functional dependencies, or any general denial constraint) to be used for error detection (specification of Ω) and as features in the model. The original HoloClean paper suggests a denial constraint relaxation procedure that relaxes one hard denial constraint to multiple softer ones. The denial constraint featurizer then uses the soft constraints to generate features for each cell. The feature value is simply the normalized count of the denial constraint violations that the possible value causes when put in that cell. We refer to the paper of the details of the relaxation procedure [37].

2.4 Information Measures As Correlations

In the context of inference systems on relational tables, it is often needed to determine the information dependencies between columns. That is, decide what columns are needed to predict or compute other columns.

In database terms, this is equivalent to discovering the existing functional dependencies between the interesting columns from the data. This is often a hard problem to solve, and

a lot of work has been done to discover functional dependencies (a survey can be found in [32]). However, these systems are usually expensive to compute, and they often overfit the discovered rules to the data, especially when the data is already dirty.

Functional dependencies define a causal relationship from the determinant set to the dependant set. However, since they can be expensive to discover, a more straightforward proxy that is often used is to study the correlations between columns. We review two information-theoretic measures used as generalized correlation measures: Mutual Information and Normalized Conditional Entropy. These measures have probabilistic interpretations and are often used to measure associations beyond the limitations of correlation coefficients, like non-linear relationships and categorical variables.

2.4.1 Mutual Information

Mutual information (MI) (Also known as information gain) can be defined as follows:

Mutual information is a measure of the mutual dependence between two random variables. It quantifies the "amount of information" obtained about one random variable by observing the state of the other random variable. Formally, the mutual information of two random variables X and Y is given by:

$$I(X;Y) = D_{KL}(p_{(X,Y)}(x,y) \| p_{(X)}(x) p_{(Y)}(y)) \tag{2.1}$$

$$= \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p_{(X,Y)}(x,y) \log\left(\frac{p_{(X,Y)}(x,y)}{p_{(X)}(x) p_{(Y)}(y)}\right) \tag{2.2}$$

Where D_{KL} is the Kullback–Leibler divergence.

In other words, mutual information is a measure of the inherent dependence expressed in the joint distribution of X and Y relative to the marginal distribution of X and Y under the assumption of independence. From definition **??**, equation 2.1 shows that the mutual information can be viewed as comparing the "distance"[2] between the joint distribution of the variables and the product of their marginal distributions. It is also to see that mutual information is non-negative and is symmetric.

Since mutual information measures how much knowing one variable reduces uncertainty about the other, it can be used in some cases to make a statement about functional dependencies between the columns. For example, if X and Y are independent, then knowing

[2]The general Kullback–Leibler divergence is not a proper distance, but is often treated as one although it is not always symmetric, in the mutual information case it acts as distance.

X does not give any information about Y and vice versa, so their mutual information is zero. At the other extreme, if X is a deterministic function of Y and Y is a deterministic function of X (i.e. $X \longrightarrow Y$ and $Y \longrightarrow X$), then all information conveyed by X is shared with Y: knowing X fully determines the value of Y and vice versa.

2.4.2 Normalized Directed Conditional Entropy

First, we define Conditional entropy, and then show how it can be normalized to be a correlation measure. Conditional entropy can be defined as follows:

The conditional entropy quantifies the amount of information needed to describe the outcome of a random variable Y given that the value of another random variable X is known. The conditional entropy of Y given X is:

$$H(Y|X) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{(X,Y)}(x,y)\log(\frac{p_{(X,Y)}(x,y)}{p_{(X)}(x)}) \tag{2.3}$$

However, if we want this quantity to behave as a normalized score, we need to change the definition as follows:

$$H_{[0,1]}(Y|X) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{(X,Y)}(x,y)\log_{|dom(Y)|}(\frac{p_{(X,Y)}(x,y)}{p_{(X)}(x)}) \tag{2.4}$$

The only difference is setting the log base to $|dom(Y)|$ (i.e. the domain size of Y). This ensures that $H_{[0,1]}(Y|X)$ is always normalized in range $[0,1]$.

Conditional entropy can also be used to make a statement about functional dependencies between the columns. For example, if Y is a deterministic function of X (i.e. $X \longrightarrow Y$), then $H(Y|X)$ is equal to zero. At the other extreme, if the column Y is independent of X, then the uncertainty in Y does not change by knowing anything about X, so it stays the same, equal to the individual entropy of the variable Y (i.e. $\mathrm{H}(Y|X) = \mathrm{H}(Y)$).

It is important to note that normalized conditional entropy on its own is not a direct association measure. However, it can easily be turned into a one by subtracting $H_{[0,1]}(Y|X)$ from 1 to give the correlation measure meaning that we are used to.

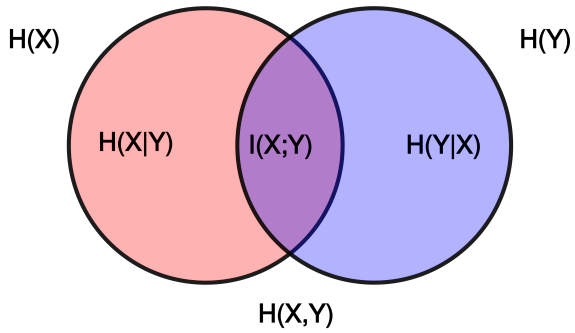


Figure 2.2: Relating information-theoretic quantities. From [44]

The normalized directed conditional entropy correlation score is formally defined as:

$$NDCE(X \longrightarrow Y) = 1.0 - H_{[0,1]}(Y|X) \tag{2.5}$$

This correlation score is in the range $[0, 1]$ and is not symmetric. This asymmetry defines the direction of the functional dependency.

2.5 HoloClean

From the HoloClean official website[3]:

"HoloClean is a statistical inference engine to impute, clean, and enrich data. As a weakly supervised machine learning system, HoloClean leverages available quality rules, value correlations, reference data, and multiple other signals to build a probabilistic model that accurately captures the data generation process and uses the model in a variety of data curation tasks. HoloClean allows data practitioners and scientists to save the enormous time they spend in building piecemeal cleaning solutions, and instead, effectively communicate their domain knowledge in a declarative way to enable accurate analytics, predictions, and insights from noisy, incomplete, and erroneous data."

One of the most important principles behind HoloClean is integrating multiple quality signals from various sources. This is done by treating all the various signals (e.g. quality

[3] http://www.holoclean.io

rules, value correlations, textual similarity etc.) as different features for a machine learning model. The final classification model can then be seen as a simple voter model that learns how to weigh these different features to get the most probable decision, guided by a training procedure that optimizes predicting the correct value out of a set of possible values.

2.5.1 Paper Vs. Open-Source Code

The HoloClean paper [37] presented the original inference problem as an inference problem over a large scale factor graph. The paper was then implemented in DeepDive[4] [52], a system that facilitated the construction of these large scale factor graphs using DDLog statements. That was complex and very expensive. The HoloClean open-source system[5] kept many of the same principles of the paper but implemented the core inference components as neural networks in PyTorch[6].

This section presents some of the important implementation details. More details are presented in the following chapters when needed.

2.5.2 Pipeline

In this part, a rundown of how HoloClean runs on a dataset is given. Figure 4.1 shows the breakdown of steps. First, the dataset is given in terms of four components:

- **Raw Data**: The data that needs to be cleaned, in ".csv" format.
- **Ground Truth**: Optional. It can be used to evaluate the repair module or to detect some of the errors.
- **Unknown Cells**: Optional, this is the list of cell ids that inference needs to run on.
- **Constraints**: Optional, this is a list of all integrity constraints, expressed as denial constraints in the HoloClean DC grammar.

After the data is fully specified, HoloClean assigns a unique cell ID for each cell in the raw data. The HoloClean pipeline consists of the following steps:

[4] http://deepdive.stanford.edu
[5] https://github.com/HoloClean/holoclean
[6] https://pytorch.org

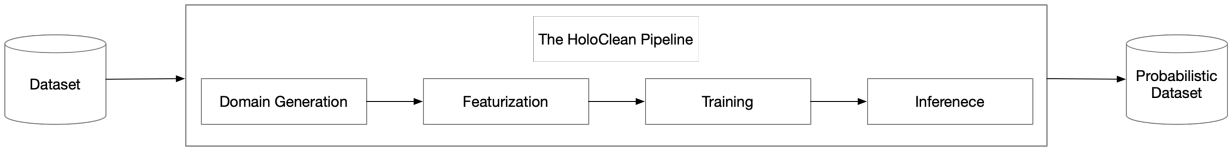


Figure 2.3: Overall HoloClean Pipeline for The Data Repair Problem

1. **Domain Generation**:
   In this step, all cells are given a domain of possible values. Domain generation serves two purposes:

   - **Negative Sampling**: For training data, the cells are expected to be mostly clean. Domain generation adds values that are known to be wrong to the domain. These values are not randomly selected as in other applications of negative sampling (e.g. Word2Vec). However, they are selected considering correlations between columns and co-occurrences of values. This ensures that values selected in a cell's domain are, in some sense, the most likely (or confusing) values for this cell. This increases the ability of the model to distinguish between confusing values better.
   - **Domain Pruning**: In test data (i.e. unknown cells), domain generation serves as a way to limit the possible value for predictions of the model. This makes the softmax classification step better at assigning probabilities to each of the possible values because the softmax normalization is not done across a large number of values.

   Algorithm 2 shows a rough outline of the domain generation algorithm used in the original HoloClean [37] paper (without most of the performance optimizations included in the open-source implementation, except for excluding low-correlated columns).

2. **Featurization**:
   One of the most important principles of HoloClean is the ability to integrate various quality signals like rules and value co-occurrence. This is done by encoding these quality signals as features. Features are extracted for each possible value in the domain of each cell. There are two main features that we need to focus on: co-occurrence and denial constraint features.

- **Co-occurrence feature**: The co-occurrence feature captures relationships between values. It is defined between all column pairs. For example, if we have dataset with three columns $\mathcal{D}(A, B, C)$ the co-occurrence feature vector for each possible value in A would be: $< cooc(A, A), cooc(A, B), cooc(A, C), cooc(B, B), cooc(B, C), cooc(C, C) >$ (where $cooc$ is defined in equation 2.6). Concretely, within the dataset D, if we have the discrete columns $C1$ and $C2$, the cooccurrence feature of between any two values $v1 \in C1$ and $v2 \in C2$ is defined as:

$$cooc(v1, v2) = Pr[v2|v1] = \frac{\#(v1, v2) \text{ appear together in } D}{\#v2 \text{ appears in } D} \tag{2.6}$$

Algorithm 1 computes the unnormalized co-occurrence statistic for the full dataset.

Algorithm 1: Co-occurrence statistic computation

Input: ds: Dataset object that represents a relational table
Result: Co-occurrence count table of the schema: (column-1-name, value-1, column-2-name, value-2, cooc-count)

```
1  for c1 in ds.columns do
2      for c2 in ds.columns do
           /* Insert the co-occurrence counts for the two columns */
3          Execute-Update(
4          INSERT INTO cooc − table
5          SELECT 'c1.name' as column-1-name, c1 as value-1, "c2.name", c2 as
             value-2, count(*) as cooc-count
6          FROM ds.data-table
7          GROUP BY c1, c2
8          )
9      end
10 end
```

Normalization of the output of algorithm 1 to the features as expressed in equation 2.6 is done by dividing the resulting count by the absolute count for each value. As we can see from algorithm 1, featurizing the whole dataset D for all values is quadratic in the number of columns.

- **Denial constraint feature**: The feature is constructed as the normalized count of rule violations caused by each possible value in that cell. One important note here is: The original HoloClean paper [37], a method is suggested in section

5.2 to relax a single hard integrity constraint (expressed as a denial constraint) to multiple soft integrity constraints such that each of the new relaxed rules contains one random variable. Each of the relaxed rules becomes a feature in the model that has additional learnable weight. This makes it even more critical to control the number of rules going into the system as this could potentially increase the dimensionality of the feature space and, hence, increase sparsity.

Featurization is implemented under a unified interface. This makes adding more features to HoloClean very easy.

3. **Training**:
   Training is the step in which model parameters and weights are tuned to fit the data. The model in the HoloClean open-source code is a simple linear layer with weight tying across values. The cross-entropy loss function is used with the softmax output to optimize the weights of the linear layer. The weights are then used to combine the features and can be used to do feature importance analysis according to the assigned weight by the model. For example, section 3.3.2 describes a method where rules can be ranked according to the model weight. The model is implemented in PyTorch.

4. **Inference**:
   After the model is trained, the same forward step of the model (with the softmax classification) can run inference on the unknown cells with their features and possible value domain as input. Resulting in a probability density function (PDF) over all the domain values (i.e. each domain value receives a probability, and all domain values would sum up to one. The Maximum A Posteriori (MAP) prediction can then be applied simply by choosing the domain value with the highest probability as the correct value. However, in many cases, especially for cells with a huge domain, the highest probability can be low (e.g. < 0.001). In this case, we can use other measures (like the entropy of the generated PDF) to assess the confidence in the predictions.

The output of this pipeline can be seen as a probabilistic dataset where each cell has a domain and a probability density function over the possible values.

Algorithm 2: High-Correlation Domain Generation Algorithm. From [37]

Input: ds: Dataset object that represents a relational table, μ correlation threshold, τ co-occurrence threshold

Result: domain values (i.e. repair candidates) of cells

```
for each cell c in ds do
    /* Initialize domain for cell c                                    */
    R_c ⟵ φ for each cell c in ds do
        A_c ⟵ Attribute of cell c
        for each cell ĉ ≠ c in c's tuple do
            A_ĉ ⟵ Attribute of cell ĉ
            /* Iterate only on highly correlated attributes            */
            if directed-correlation(from=A_ĉ, to=A_c)[a] ≥ μ then
                continue
            end
            U_{A_c} ⟵ The domain of attribute A_c
            v_ĉ ⟵ the value of cell ĉ
            for each value v ∈ U_{A_c} ĉ ≠ c in c's tuple do
                if normalized-co-occurrence(v, v_ĉ)[b] ≥ τ then
                    R_c ⟵ R_c ∪ {v}
                end
            end
        end
    end
end
return repair candidates R_c for each c ∈ ds;
```

[a] The "directed-correlation" is the normalized directed cross entropy correlation from equation 2.5

[b] The "normalized-co-occurrence" function is defined in equation 2.6

Chapter 3

Scaling Machine Learning Data Repair for Sparse Datasets

3.1 Introduction

Data sparseness is a frequent problem in real-life datasets that has multiple types and causes. To show some of the sparseness types, consider the following example schema in table 3. The table represents an example of data about individuals' work and education status in different parts of the world. The schema specifies a person in terms of education enrollment status (EduStat), the last obtained degree (LastDeg), the current degree being pursued (CurDeg), the employment status (EmpStat), the work or job being performed by the person (Work), the hourly wage in a US dollar (HrWge), and location information represented as country code (CC), zip code (ZC), and city (CT). The symbol $\perp$ represents a null value.

ID	EduStat	LastDeg	CurDeg	EmpStat	Work	HrWge	CC	ZC	CT
$t1$	$\perp$	College	$\perp$	Full Time	Technician	30	A	12	X
$t2$	Full Time	High School	B.Sc.	$\perp$	$\perp$	$\perp$	A	12	X
$t3$	Part Time	B.Sc.	M.Sc.	Part Time	Barista	15	A	90	W
$t4$	$\perp$	High School	$\perp$	$\perp$	Houswife	$\perp$	B	54	Y
$t5$	Full Time	Primary School	High School	$\perp$	$\perp$	$\perp$	B	54	Z

Table 3.1: Example of workforce and education data table.

We focus on two types of sparsity because they are crucial for the assumptions of machine learning data cleaning models:

1. **Column Non-Applicability**:
   In this case, not all columns apply to all rows. Resulting in cases where a null is filled for the inapplicable cell. These non-applicable columns increase the sparsity of the data and add computational costs in different parts of the pipeline. For example, in table 3, the column "CurDeg" does not apply to $t1$ because the person is working full time. A similar argument applies to the column "HrWge" with all non-working records ($t2$, $t4$, and $t5$).

2. **The Locality of Dependence Structure**:
   This is when the dependence structure between columns of the table does not hold for all rows. These dependence structures between columns are usually encoded as schema-level integrity constraints (e.g. functional dependencies). For example, in table 3, we can spot this problem by inspecting the relationship between the three location columns [CC, ZC, CT]. A traditional functional dependency that usually exists for these types of columns is $ZC \longrightarrow CT$. In this table, we see that this is true, but only for the case that $CC = A$. Although this type of constraint is easily captured using conditional functional dependencies [9], the problem remains for correlations and other statistics. These statistics are often used to define an approximate version of these constraints (i.e. dependencies that hold with high probability in a given dataset) [17]. For example, in the Census dataset, we can observe that it is unlikely to have homogeneous correlations between columns across all horizontal slices (e.g. cities or provinces). This is due to these provinces' local properties (e.g. some provinces are more agrarian versus industrial, which will affect the dependence between workforce variables).

Current machine learning solutions for data repair make certain homogeneity and density assumptions on the data. This is done by utilizing schema information to encode information about the structure of the attributes. For instance, in HoloClean [37], the feature weights are learned on the functional dependency level, not on the value level. Similarly, in [49], the attention weights are learned on the positional encoding of the schema columns, not the values themselves. These assumptions and models work well for many average-sized and relatively-dense datasets. However, as shown in the previous example, homogeneity assumptions do not hold for sparse datasets. This leads to the performance degradation of these systems on sparse datasets.

Furthermore, under these assumptions, the representation of machine learning models becomes very sparse. For example, a feature tensor built with the assumption that all features apply to all cells (i.e. first problem) will have many zeros for the non-applicable features. This leads to computational problems that make the sparsity effect even worse on these systems.

The rest of this chapter is organized as follows: section 3.2 presents solutions to column non-applicability and locality of dependence structures, respectively. Section 3.3 will suggest some solutions to the computational problems resulting from data sparsity in machine learning models. Finally, section 3.4 presents general implementation improvements for data pipelines that proved to be particularly useful during the experiments.

3.2 Slicing to Solve Data Sparseness

We can reduce data sparseness by conditionally partitioning the table horizontally and vertically. The goal is to produce a different dense table for each repair target column, where the assumptions of current machine learning models hold. We produce these dense tables by partitioning the data. Data can be partitioned in two ways: Vertically and horizontally. We present methods for each way and explain why these methods work to solve the two data sparseness problems illustrated in the introduction.

3.2.1 Correlation Based Vertical Partitioning

Including more context columns for a target prediction task often has diminishing returns. That is, not all columns are equally important for all data cleaning tasks. Vertical partitioning refers to using only the relevant set of context columns C when running an inference task for a target column t. In a standard machine learning setting, this is equivalent to the process of feature selection. In general, feature selection is known to offer several benefits to machine learning algorithms, including [45]:

1. Decrease model complexity, which leads to easier interpretability.
2. Decrease the training time of the model.
3. Decreasing feature-space sparsity (i.e. avoiding the curse of dimensionality).
4. Improving generalization by reducing variance (overfitting).

A feature selection method selects the most important features from a more extensive set of features according to some measure of importance. A simple exhaustive search algorithm would test each possible subset of features finding the one which minimizes the final model error rate. This algorithm is $O(2^N)$, where N is the number of all possible features to choose from, and hence, intractable for large numbers of possible features. We refer to [45, 18] for a survey of other feature selection methods.

In this work, multiple correlation measures with the target variable are used to make the feature selection. The correlation-based feature selection measures evaluate subsets of features based on the following hypothesis: "Good feature subsets contain features highly correlated with the classification, yet uncorrelated to each other" [45]. We can then sort the attributes in descending order according to the correlation score, then use the top-k attributes, or look for an obvious distinctive change in the observed correlation score (i.e. The elbow method). Algorithm 3 illustrates the steps of correlation-based vertical partitioning.

Algorithm 3: Correlation-based vertical partitioning for a target column.

Input: ds: Dataset object that represents a relational table
t: target column for cleaning
Ω: a set of correlation measures between the target column and context columns
k: number of top correlated columns to keep from each correlation measure
Result: C: set of most correlated columns, at most of size k, to be used as the context for cleaning the target column t

```
/* Initialize C to be all attributes in ds except the target                    */
C ⟵ All attributes in ds except t
for ω in Ω do
    /* Initialize list to store correlations                                    */
    corr_ωt ⟵ []
    /* Iterate on all potential context columns compute the correlations        */
    for each column c in C do
        corr_ωt.append((c, ω(t, c)))
    end
    /* Apply any top-k algorithm to get the highest correlated context columns */
    SelectedContext ⟵ TOP − k(corr_ωt, k)
    /* Update C by taking the correlations                                      */
    C ⟵ C ∩ SelectedContext
end
return C
```

For categorical variables, mutual information, and normalized conditional entropy correlation (as defined in chapter 2) have been used as measures of discrete correlations (e.g. [18, 51, 31]). These metrics are already being used in data repair machine learning systems to do similar tasks. For example, normalized conditional entropy correlation is already being used in HoloClean [37] to determine the columns with the highest correlations to include in the domain generation step.

This correlation-based vertical partitioning can significantly help with the problem of column non-applicability for a specific target column. The assumption here is that columns with high correlations do apply and provide useful information to predict the target column. It is important to note that algorithm 3 makes a critical assumption about the correlations measures used: Null values can not be used to infer any correlations from the data. That is, when the values of a column pair are considered for a correlation computation, the only records considered are the ones where values from both columns are non-null.

3.2.2 Conditional Dependence Based Horizontal Partitioning

The data homogeneity assumptions made by machine learning data repair systems results in smaller and simpler models. For instance, in HoloClean [37], the feature weights are learned on the functional dependency level, not on the value level. Similarly, in [49], the attention weights are learned on the positional encoding of the schema columns, not the values themselves. These simple models are easier to train in a self-supervised way with little or no labelled data. However, in systems like HoloClean, these self-supervised models were still trained to overfit the entire dataset to capture all possible correlations. Self-supervised training on the entire dataset becomes very expensive computationally as the dataset becomes large. This problem can be addressed in other machine learning applications using sampling techniques (e.g. stratified or uniform sampling). However, these sampling techniques perform poorly on sparse datasets due to the data's heterogeneity and sparseness. For example, uniform sampling from a large sparse dataset is not guaranteed to capture all islands' different dependence information.

Datasets can often be split into logical partitions easily. The split referred to here is where the values of one or more columns are projected out to create multiple datasets, one for each value. For example, given a dataset with three attributes: $\mathcal{D}(A, B, C)$, if column A has three values $A \in \{a_1, a_2, a_3\}$, then we can generate three different datasets: $\mathcal{D}_{a_1}(A = a_1, B, C)$, $\mathcal{D}_{a_2}(A = a_2, B, C)$ and $\mathcal{D}_{a_3}(A = a_3, B, C)$ and then the imputation

pipeline is applied to each dataset separately. Algorithm 4 illustrates the steps.

Algorithm 4: Conditional dependence based horizontal partitioning.

Input:
ds: Dataset object that represents a relational table
c: column to use for horizontal partitioning
Result:
ds_c: multiple datasets, each of which is projected on one of the values in column c

```
/* Initialize V to be the domain of column c (i.e. set of unique values in
   column c)                                                              */
```
$V \longleftarrow \{v : v \in c\}$
```
/* Initialize ds_c to the empty list to store all the horizontal partitions */
```
$ds_c \longleftarrow []$
for v *in* V **do**
  $ds_v \longleftarrow PROJECT(ds, c = v)$
  ds_c.append(ds_v)
end
return ds_c

The $PROJECT$ function is equivalent to applying the SQL query: "SELECT * FROM ds WHERE $c = v$ ". Where c is the column used for horizontal partitioning and v is the value for that column that is being projected.

This horizontal partitioning can boost the statistical signal in each partition because it will not be mixed with signals from other irrelevant partitions. The horizontal partitioning can also be viewed as partitioning to splits for a single tableau value of a conditional functional dependency. For example, if we have a conditional functional dependency as defined in chapter 2: $(\mathcal{X} \longrightarrow \mathcal{Y}, \mathcal{T}_p)$. If the tableau $\mathcal{T}_p$ only includes assignments for a few domain values of $\mathcal{X}$, then the rest of the domain of $\mathcal{X}$ does not have the $\mathcal{X} \longrightarrow \mathcal{Y}$ rule. So, by horizontally partitioning the dataset using different columns, the resulting datasets should have a simpler, more local dependence structure, contributing to solving the second problem mentioned above.

The combination of vertical and horizontal partitioning for a large and sparse dataset effectively creates dense and homogeneous data islands where the models' assumptions hold. Furthermore, this combination enables efficient uniform sampling for training within each island. This is because if the sampling is done per homogeneous dense island, a small uniform sample from each island is enough to capture the necessary dependence information, unlike the case where sampling is done over the entire sparse dataset, where

the sample will not include all information needed to capture the heterogeneity of the dataset.

3.3 Improvements to Machine Learning Model and Feature Sparsity

Data sparseness does not only violate the fundamental assumptions of machine learning data repair systems. It also poses significant computational problems (memory and runtime) to these machine learning systems. This is because these systems will often try to construct high dimensional feature tensors with sizes directly affected by the dataset's sparseness. Resulting in extremely sparse feature tensors (i.e. the number of zero values $\gg$ the number of non-zero values). This section addresses the problem of feature and model sparseness and suggests solutions to mitigate their effects. Some of these suggestions are specific to HoloClean, while others are general and can be applied to other machine learning data repair systems.

3.3.1 Removing Pseudo-key Columns

An attribute A of a dataset D is a pseudo-key if it has a relatively large number of unique values. More concretely, if the following condition holds:

$$\frac{\#\text{unique values of } A}{|D|} \geqslant 1.0 - \epsilon \tag{3.1}$$

Where $|D|$ is the number of rows of the dataset D. We call the left-hand side of this inequality the **Pseudo-Key Ratio**. For proper table keys, this ratio is equal to 1.0 since the key is unique across all records in a table.

Pseudo-keys often cause many problems for machine learning data repair systems like HoloClean [37] for three main reasons:

1. **Pseudo-key attributes are not suitable to be a target for imputation**: This is because -by definition- pseudo-keys are almost unique per row, that is, no other non-key attribute can predict them well. In other words, trying to predict a pseudo-key attribute will result in a model with a high loss and hence, inaccurate imputation.

2. **Pseudo-key attributes are not suitable to be a context column**:
   Using the same reasoning from point 1, pseudo-key attributes do not have enough statistical signal to be used as an accurate predictor for any other attribute. Furthermore, keeping pseudo-keys in the context for predicting any column could lead to adverse effects because machine learning models could easily overfit to pseudo-key features, ignoring other useful features, leading to the model's inability to generalize to unseen examples.

3. **Pseudo-key attributes lead to computational performance problems**:

   In particular, for operations that are quadratic in the number of the columns. For example, computing the co-occurrence statistic as defined by equation 2.6 for all columns is quadratic in the number of columns.

Removing pseudo-keys can be very helpful with many of the problems arising from having sparse datasets. It is also important to note that computing the statistic in equation 3.1 is efficient and can be implemented to run concurrently for each column as the cost of computing it per column is $\mathcal{O}(M \log M)$, where M is the number of rows in the dataset (assuming an implementation that is similar to that of a sort and group-by). So, for operations that are quadratic in the number of columns, removing one pseudo-key column can result in savings that are $\mathcal{O}(N)$ bigger than the cost of computing it, where N is the number of columns in the dataset. This will also make the cost for constructing the feature tensors less by removing columns that will not add additional signals.

3.3.2 Removing Superfluous Rules

We can apply similar reasoning from subsection 3.2.1 to the rules and integrity constraints: Including more rules for a prediction task often has diminishing returns. That is, not all rules are equally important for all data cleaning tasks. Hence, We need methods to prune unimportant rules for the given data cleaning task with minimal effect on the system's cleaning performance.

There are multiple ways rules can be pruned with minimal effect on the data quality system, and the method used will depend on the dataset and the other transformations applied to it (e.g. vertical partitioning, sampling ... etc.). For example: If we apply vertical partitioning from subsection 3.2.1, we only need to include the relevant rules for the target vertical slice (which includes one target column and its selected features). Pruning

methods could also depend on the semantic meaning and nature of the rules themselves. For example, if we limit the set of possible rules to first-order logic statements, presented as denial constraints, we can use logical inference rules and systems to get a minimal set of logically equivalent rules. However, this new minimal set might be smaller in number but might be more complicated for humans to understand and maintain.

In this work, it is assumed that the set of rules (denoted hereafter as Σ) is minimal for the dataset at hand. That is, all rules include only columns from the slice of data of interest, and that the logical minimization is done or is ignored. Similar to subsection 3.2.1, the goal is to produce some measure of importance for each rule. We can then sort the rules in descending order according to the importance score and use only the top-k rules or look for an obvious distinctive change in the observed importance score (i.e. The elbow method). This is very similar to algorithm 3.

Two heuristics are provided to rank the importance of a rule: The number of violations and the model's weight of the rule.

1. **Ranking Rules By The Number of Violations**:
   In this simple heuristic, rules are ranked by the number of violating cells of that rule. Checking a two tuple denial constraint violations is usually as simple as a self-join and a few conditions (that correspond to first order logic predicates in the constraint) and hence, it can be efficiently implemented. Concretely, for a rule $r \in \Sigma$, the violations importance score is defined as:

   $$\text{Importance Score}_{Violations}(r) = \text{Absolute Number of Violating Cells of } r \tag{3.2}$$

   The assumption behind this importance score is that some rules would have very few violations, and so, they are safer to ignore because they do not have a strong quality signal as other rules with more violations. One possible explanation is the following: In a traditional rule-based data quality system, rules (not features) are the primary way to communicate data quality signals to the system. So, human operators might find themselves in a position where they have to add some rules to handle some rare cases that are not representative of the dataset's underlying error distribution. These rules are expected to have very few violations and should be captured by this importance score.

2. **Ranking Rules By The Model Weight**:
   As mentioned in chapter 2, the HoloClean system [37] uses the normalized count of

violations of the rules as one of the features for the model to provide a signal about the correctness of each data cell. Furthermore, each hard rule can be relaxed to multiple soft rules, which are then used as features with a learnable weight.

Weights of the features in the model can be positive or negative, but in both cases, the absolute value of the weight (when the features are normalized, not just counts) can be used to indicate the level of importance the feature has, and hence the importance of the rule that generated the feature.

By running HoloClean [37] using only the relaxed rules as features and then inspecting the learnable weights assigned by the model, we can produce a relative importance score for each rule. Assuming each rule $r \in \Sigma$ produces a set of real normalized features $F = \{f_1, f_2, \dots f_K\}, f_i \in R$, which are then assigned a corresponding set of weights $W = \{w_1, w_2 \dots w_K\}$, the model importance score is defined as:

$$\text{Importance Score}_{ModelWeight}(r) = AGG(\forall_{f_i \in F} |w_i|) \tag{3.3}$$

Where $|.|$ is the absolute value, and AGG is an aggregate function for a set of real positive numbers that also produces a positive real number. For example, max, min, or $mean$ could all be applied.

3.3.3 Sparse Tensors

The representation of the machine learning models and features for data repair systems is often very sparse (i.e. the number of zero values $\gg$ the number of non-zero values). For example, a feature tensor built with the assumption that all columns apply to all cells will have many zeros for all the non-applicable cells. This leads to computational problems that make the sparsity effect even worse on these systems.

Although this implementation detail works for many machine learning models and systems, it was essential to improving the memory consumption of HoloClean. This is because the feature tensor built for the HoloClean model is usually extremely sparse. There are two additional reasons why the HoloClean tensors are very sparse:

- **Tensor Padding**: HoloClean uses a three dimensional feature tensor as input: (#cells, #values, #features). That is, each cell has multiple possible values, and each value has a feature vector describing it with respect to its context. The number of possible values generated per cell is not constant, and this means that padding to

the maximum number of possible values must be used to make the tensor of consistent shape; cells with huge domains will usually lead to out-of-memory tensors even for the smaller cells included with it.

- **Features Sparsity**: One of the most important features in HoloClean is the co-occurrence feature, and it is defined between all column pairs. For example, for a dataset with three columns $\mathcal{D}(A, B, C)$ and a chosen target column A, the co-occurrence feature vector for each possible value in A would be: $< cooc(A, A)$, $cooc(A, B)$, $cooc(A, C) >$ (where $cooc$ is defined in equation 2.6), this way of featurization could lead to many zeros because not all values for A will have non-zero co-occurrences with all values from B and C. A similar argument can be made for the denial constraint features. If the denial constraint does not apply to a certain cell, the feature value is added with zero value. This creates many zero values that are not informative.

Using sparse tensors incurs a computational performance cost, making the trade-off between time and memory more pronounced. In many cases, it is acceptable to take more time to run the task if the available memory is low. For example, savings in memory consumption by implementing sparse tensors for HoloClean was 97% for the test dataset presented in chapter 4, and it only caused a 30% slower model inference due to the implementation used in the sparse python library. [1]

3.4 General Implementation Improvements

In this section, we provide some implementation details that made significant improvements in the computational performance of HoloClean. These changes can also be seen as general principles of implementing data pipelines since their benefits extend beyond just the case of HoloClean implementation.

3.4.1 Concurrency

Concurrency refers to executing multiple, independent steps of an algorithm or a program at the same time, instead of executing them sequentially. Modern commodity hardware has multiple CPU cores and enough memory to enable this mode of execution. In simple

[1] https://github.com/pydata/sparse

cases, the speed up from concurrent execution can be nearly linear with the number of jobs. When implemented concurrently, I/O intensive operations like reading or writing to disk (e.g. operations with many reads and writes to and from the database) tend to do especially well because it increases the utilization of CPU cores when it is blocked on the I/O.

For example, the process of computing the co-occurrence statistics for the entire dataset is quadratic in the number of columns (as seen in chapter 2, algorithm 1). However, the implementation of this algorithm can be optimized by observing that the co-occurrence computation for each pair of columns is independent of all other column pairs. Hence, all queries can be generated first and then executed concurrently. This simple change led to a decrease in execution time for this algorithm by almost six folds (i.e. x6 faster code) in some cases. Python's multiprocessing module [2] was used to do this, and it is offered as a part of the Python standard library.

3.4.2 Indexing

Since many operations in HoloClean are implemented as database operations (e.g. joins, group by etc.). Indexing becomes a vital tool to make these database operations much faster for larger datasets. For example, having indexes enables the database optimizer to make smarter decisions about the algorithm it should use for a join query. Indexes incur additional costs in the case of inserting new data since the new data must be inserted in the data table and the built index to maintain the consistency between the table and the index. So, it is more efficient if indexes are built after filling the tables with data; this way, indexes can be built more efficiently using bulk-insert operations, which are much more I/O efficient.

3.4.3 Batching

Modern machine learning models, especially deep learning models, are usually defined in terms of tensor operations, and so they are usually implemented in one of the popular tensor computation libraries (e.g. PyTorch [3], Tensorflow [4]). These libraries provide high-level abstractions and efficient low-level implementations of the most frequent operations that take advantage of hardware acceleration, either by using graphics processing units GPUs

[2] https://docs.python.org/3/library/multiprocessing.html
[3] https://pytorch.org
[4] https://www.tensorflow.org

or using special vector instructions on modern CPUs. In all cases, using these libraries increases the development speed significantly and offers a homogeneous interface for all tensor operations that could be used to define a machine learning model.

When using operations tensor processing libraries (like dot product or matrix multiplication), it is important to know that the efficiency of execution heavily depends on whether or not the input is "vectorized". That simply means that the input is appropriately batched instead of applying the operation on each input individually. Concretely, vectorization is a word that refers to using Single Instruction Multiple Data (SIMD) operations. This means that the same instruction is applied to multiple operands at the same time.

For example, if we want to compute the dot product between each pair of a set of embedding row vectors $V = \{v_1, v_2 \dots v_N\}$. Instead of computing each dot product for each pair individually (i.e. $v_i, v_j \in V \longrightarrow < v_i, v_j >$), we can express the same operation as a matrix multiplication as follows: $A = [v_1; v_2; \dots v_N]$ (i.e. vertically stack all the row vectors as rows of a matrix A), then the computation can be expressed as $S =< A, A^T >$. In many cases, this could lead to a code that is tens of times faster, depending on the matrix's size and the hardware being used.

Chapter 4

Dataset and Results

4.1 Introduction

This chapter presents a large, sparse dataset: Census and experimental results for the suggested solutions in chapter 3, implemented using HoloClean [37]. The experiments show that the suggested solutions resulted in a faster pipeline without sacrificing the data cleaning performance.

4.2 Dataset

In this section, we introduce the Census dataset and some of the pre-processing steps necessary to make running HoloClean [37] on it possible.

4.2.1 Description

The data were acquired in collaboration with a governmental agency in a developing country. It was the result of surveying more than 85,000 families for their income and work conditions. A record is generated for each family member with their information, yielding a total of 350669 records. The governmental agency conducted the survey as a part of the 2013 census and was then processed and manually checked and corrected against predetermined domain-specific rules. It is interesting to note that many columns are entirely error-free. Out of the total 120 attributes, 66 attributes have mistakes when compared to

the provided ground truth. For example, the column 'reltohd' (relation of family member to the family head) has 1138 errors.

The dataset has the following components:

- **Raw data** :
  A raw data file of 350,669 records in 120 columns.
- **Ground truth** :
  The 350,669 records but manually verified in 120 columns.
- **Constraints** :
  A set of 131 integrity constraints was provided with the dataset; they were used with manual labelling to correct the mistakes in the raw data.

A complete enumeration of the dataset variables, their description, and the integrity constraints (defined as denial constraints in the HoloClean DC grammar) are provided in appendix A.

4.2.2 Pre-processing

All three components needed some pre-processing before they were ready for input to HoloClean. The pre-processing steps are listed below:

1. **Transform data format to CSV** :
   The provided datasets (raw and ground truth) were both provided in SPSS format [1]. Extraction and transformation to Comma-Separated Values (CSV) format were done before any other step. This was done using the python pandas library SPSS function[2].
2. **Unify the raw data and the ground truth** :
   The raw data values were given in their textual representation, but the values in ground truth were given in a categorical enumeration, and dictionaries were provided to do the mapping. Unifying the data types and the value representation is essential for HoloClean and enables easier debugging.

[1] https://www.ibm.com/products/spss-statistics
[2] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_spss.html

3. **Link the raw data and ground truth records** :
   The key given for the raw data table was different from the key given for the ground truth table, making linking them difficult. Luckily, the raw data key is a computed column from other clean id columns in the dataset. The keys were unified by applying the same computation on the ground truth table, and the two datasets were linked.

4. **Translate the constraints to HoloClean denial constraints grammar**:
   Since data quality rules were given in plain English, they needed translation to the HoloClean denial constraint grammar. This enables HoloClean to use these rules as features in the model.

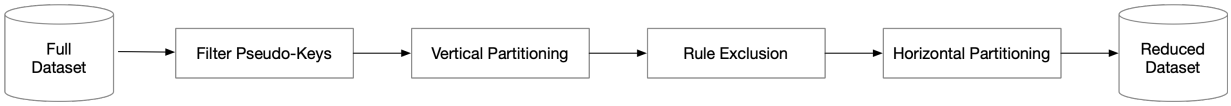
4.3 Experiments

The goal of this section is to show how some of the suggestions that we provided in chapter 3 can be applied to the test dataset described in section 4.2 to run HoloClean's data repair pipelines in a much faster time with much less memory. It is important to note here that, although we choose only one target column ("reltohd") to focus our investigation on, the same steps can be applied to any other target column since none of them is specific to this column. Subsection 4.3.1 explains the other reasons this column was selected in particular.

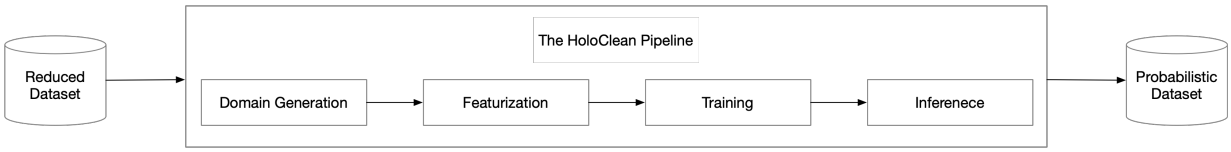
4.3.1 The Target Column: "reltohd"

The "reltohd" column encodes the record's relationship to the head of the family. The column was selected for multiple reasons. First, it is the column with the highest number of errors (1138 errors). Second, it has an important semantic meaning for this dataset about workforce and income; the head of the family in that country is often the main worker and breadwinner. Finally, it is involved in many rules with other columns like marriage, age, and education (25 rules).

The column has a domain of eleven values, illustrated with their absolute and normalized counts in table 4.1, ordered by counts in descending order. The low number of unique values makes it easier to test many of the ideas presented in chapter 3 on regular hardware like a laptop with small memory due to features tensor size because we can use the entire set of eleven values as the cell domain for HoloClean.



(a) Steps 1-4: Data Reduction Pipeline



(b) Step 5: HoloClean Pipeline

Figure 4.1: Overall Pipeline

	Domain Value	Absolute Count	Normalized Count
1	son/daugther	182550	52.05%
2	head	85392	24.35%
3	spouse	66812	19.05%
4	niece/nephew	6879	1.96%
5	father/mother	2880	0.82%
6	daughter/son-in-law	2864	0.81%
7	brother/sister	1810	0.51%
8	other relatives	1208	0.34%
9	brother wife	157	0.044%
10	no relation	113	0.032%
11	baby sitter	4	0.0011%

Table 4.1: Column “reltohd” domain value distribution

4.3.2 Improved HoloClean Pipeline

Initially, when trying to run HoloClean out of the box on the full dataset to repair the column “reltohd”, the code fails (after 4+ hours) because of memory problems during the co-occurrence statistic computation step. The runs were made on a laptop with eight cores and 16 GB of RAM. Using some of the suggestions from the previous chapters, we present a simple pipeline, which was able to run the HoloClean repair pipeline successfully on the test dataset. The pipeline is illustrated in figure 4.1.

1. **Eliminate pseudo-key columns**:
   Pseudo-key attributes are defined in section 3.3.1. The statistic of equation 3.1 was computed for all columns. The highest two columns ('caseser', and 'fam_unique_number') had a score of 25% unique. That means they have approximately 88,000 unique values. Plus, both columns are obvious to be some identifier that can not be used for imputation purposes. So, both of these columns were excluded from the dataset.

2. **Partition the data vertically**:
   Next, we consider only context columns that are needed for the task at hand, that is, imputing the column "reltohd". To do that, we use correlation-based feature selection, since all our variables are categorical, we use two discrete measures of correlation: Mutual Information, and Normalized Conditional Entropy Correlation (as defined in chapter 2). Normalized Conditional Entropy Correlation is already being used in HoloClean to determine the columns with the highest correlations to include in the co-occurrences for the domain generation step. Mutual Information was used as a secondary measure of correlation to exclude further columns. We show columns with highest correlations (expressed as both normalized conditional entropy and mutual information) in figure 4.2. We can see that there is a sharp drop in both correlation measures after including 60 columns. For this experiment, we took the top 60 out of 120 columns in both correlation measures and got the intersection of these two sets. This resulted in only 35 context columns to be included for predicting "reltohd". For reference, the 35 columns are: ("bir_yr", "contr_type", "crecact", "crempst", "crestabl", "crhlthins", "croccup", "crsectot", "crsocins", "crstabl", "crwrkdur", "crwrkhrs_wek", "crwrkm", "dsready", "ecact_2d", "edu_enrol", "educ_class_comp", "educ_st", "educ_stage_comp", "employees_no", "expan", "fam_ser_gov", "firstentry_yr", "mar_st", "person_number", "proccup", "prwrk", "respondant", "rsn_ntdsr_wrk", "scoccup", "sex", "wg_basic", "wg_basic_prd", "work", "wrkchang").

3. **Filter out the unnecessary rules after vertical partitioning**:
   Once vertical partitioning is done, it is straightforward to see that only constraints that include "reltohd" and any of the other columns from above should be included, while all other denial constraints should be excluded. This resulted in retaining only 25 denial constraints out of the 131 rules initially given.

4. **Partition the data horizontally**:
   The simplest solution for horizontal partitioning is random sampling. For this experiment, we randomly selected only 100,000 "reltohd" cells to be used for training while retaining the rest for testing. Another possible way would be to break down the dataset with the "gov" column, obtaining 27 smaller datasets.

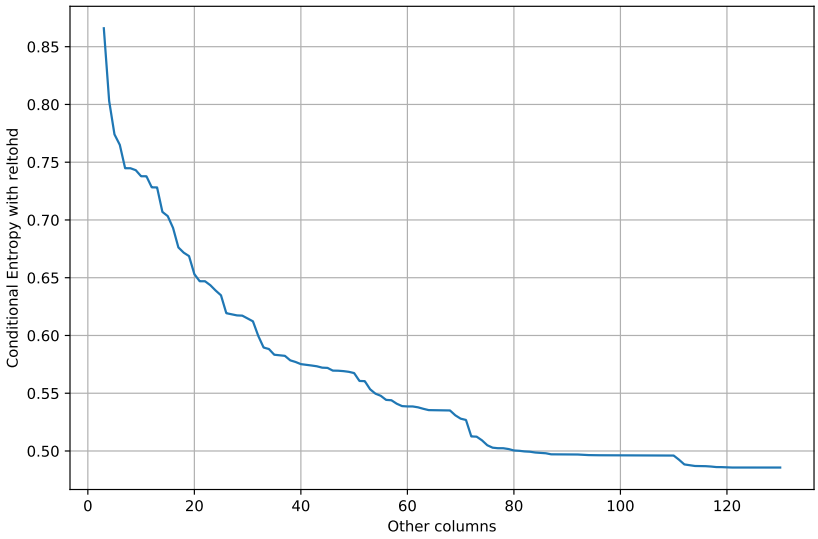
5. **Run the standard HoloClean pipeline**:
   After these changes to the dataset, it becomes much easier to run the standard HoloClean pipeline of the following stages: (Domain Generation, Featurization [with co-occurrence and denial constraints], Training, Inference). The overall run-time for this pipeline was less than 2 hours with the reduced dataset and constraints.

6. **Evaluate the model using the provided ground truth data**:
   The HoloClean model was able to suggest repairs for the "reltohd" column with **94.5%** accuracy (out of 1138 errors in the column, the model was able to suggest changes that agree with the ground truth for 1076 cells, and suggested changes that disagree with the ground truth for 62 cells).

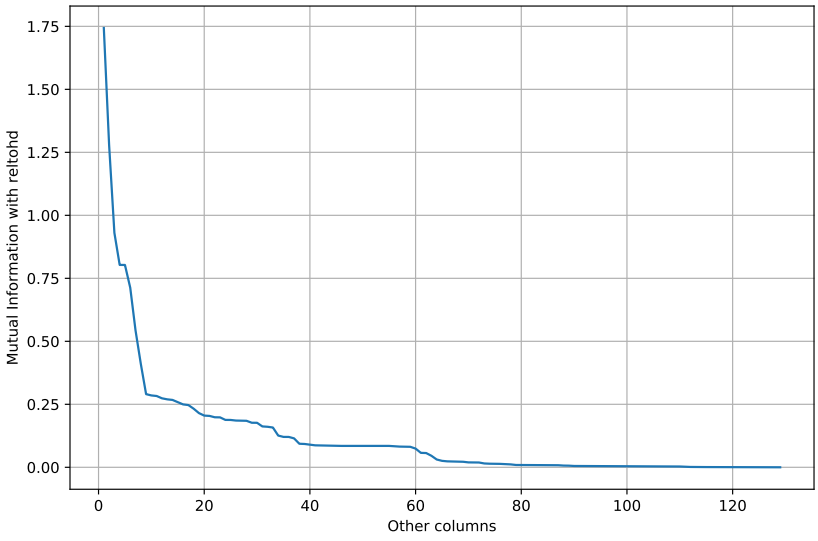
A summary of the data reduction results is available in table 4.3.2.

	#Columns	#Training Cells	#Rules
1	70.83%(120 $\longrightarrow$ 35)	71.46%(350k $\longrightarrow$ 100k)	80.9%(131 $\longrightarrow$ 25)

Table 4.2: Summary for data reduction output on the "reltohd" column



(a) Normalized Conditional Entropy Correlation



(b) Mutual Information

Figure 4.2: Discrete correlation measures between other columns and reltohd

Chapter 5

Conclusion

Despite machine learning data repair systems achieving state-of-the-art performance for the data repair problem on many datasets, these systems face significant challenges with sparse datasets. The major challenge is the data-homogeneity assumptions made by these models, which do not hold for sparse datasets. Previous solutions suggested training large global models with rich features on the full data. However, large, sparse datasets suffer from two main problems that violate these data-homogeneity assumptions: *Column Non-Applicability* and *Locality of Dependence Structure*—rendering the previous solutions less useful.

This work presented a solution that consists of constructing multiple different models per dataset, where each model is trained and used only on a specific part of the data. Data partitioning techniques can be used to construct islands of homogeneous data, where local models can be trained efficiently using standard sampling techniques. The correlation-based vertical partitioning (using mutual information and normalized conditional entropy) can solve the Column Non-Applicability problem. Conditional dependence based horizontal partitioning is suggested to solve the Locality of Dependence Structure problem. Furthermore, multiple implementation improvements (e.g. Removing Pseudo-key Columns, Removing Pseudo-key Columns, and using sparse tensors) were suggested to mitigate the effects of data and feature sparseness. For example, sparse tensors were suggested to reduce the memory requirement of HoloClean for the Census dataset by 97%.

Using HoloClean [37] as an instance of machine learning data repair systems, experimental results showed how the suggested methods could improve the performance on a new dataset: Census. Census is a sparse dataset with more than 120 columns about income and work data. While the vanilla HoloClean implementation fails on this dataset

(running out of memory after 4+ hours of run time), the suggested methods show that we can reduce the problem size by more than 70%, saving expensive computations, while still getting high accuracy data repairs (94.5% accuracy).

References

[1] Constantin F Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D Koutsoukos. Local causal and markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation. *Journal of Machine Learning Research*, 11(1), 2010.

[2] Eric E Altendorf, Angelo C Restificar, and Thomas G Dietterich. Learning from sparse data by exploiting monotonicity constraints. *arXiv preprint arXiv:1207.1364*, 2012.

[3] Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, and Ben Taskar. *Predicting structured data*. MIT press, 2007.

[4] Joshua Batson and Loic Royer. Noise2self: Blind denoising by self-supervision. *arXiv preprint arXiv:1901.11365*, 2019.

[5] Leopoldo Bertossi. Database repairing and consistent query answering. *Synthesis Lectures on Data Management*, 3(5):1–121, 2011.

[6] Michael J Bianco, Peter Gerstoft, James Traer, Emma Ozanich, Marie A Roch, Sharon Gannot, and Charles-Alban Deledalle. Machine learning in acoustics: Theory and applications. *The Journal of the Acoustical Society of America*, 146(5):3590–3628, 2019.

[7] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. Efficient denial constraint discovery with hydra. *Proceedings of the VLDB Endowment*, 11(3):311–323, 2017.

[8] 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 the 2005 ACM SIGMOD international conference on Management of data*, pages 143–154, 2005.

[9] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *2007 IEEE 23rd international conference on data engineering*, pages 746–755. IEEE, 2007.

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

[11] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.

[12] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *VLDB*, volume 7, pages 315–326, 2007.

[13] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 541–552, 2013.

[14] Wenfei Fan and Floris Geerts. Foundations of data quality management. *Synthesis Lectures on Data Management*, 4(5):1–217, 2012.

[15] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2013.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[17] Zhihan Guo and Theodoros Rekatsinas. Learning functional dependencies with sparse regression. *arXiv preprint arXiv:1905.01425*, 2019.

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

[19] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*, pages 829–846, 2019.

[20] MD. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *ACM Comput. Surv.*, 51(6), February 2019.

[21] Ihab F. Ilyas and Xu Chu. *Data Cleaning*. Association for Computing Machinery, New York, NY, USA, 2019.

[22] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019.

[23] Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. Metric functional dependencies. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1275–1278. IEEE, 2009.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[25] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[26] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*, pages 865–882, 2019.

[27] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. Discovering dependencies with reliable mutual information. *Knowledge and Information Systems*, pages 1–31, 2020.

[28] G Marcus. Deep learning: A critical appraisal. arxiv 2018. *arXiv preprint arXiv:1801.00631*, 2019.

[29] Renqiang Min and Sanjay Purushotham. Knowledge based factorized high order sparse learning models, September 8 2016. US Patent App. 15/049,983.

[30] Susan Moore. How to create a business case for data quality improvement, 2018. Follow these 5 steps to effectively design a compelling data quality improvement business case.

[31] Xuan Vinh Nguyen, Jeffrey Chan, Simone Romano, and James Bailey. Effective global approaches for mutual information based feature selection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 512–521, 2014.

[32] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.

[33] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[34] Lin-bo Qiao, Bo-feng Zhang, Jin-shu Su, and Xi-cheng Lu. A systematic review of structured sparse learning. *Frontiers of Information Technology & Electronic Engineering*, 18(4):445–463, Apr 2017.

[35] Vijayshankar Raman and Joseph M Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[36] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 11, page 269. NIH Public Access, 2017.

[37] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820*, 2017.

[38] Dimensional Research. Artificial intelligence and machine learning projects are obstructed by data issues. global survey of data scientists, ai experts and stakeholders, 2019.

[39] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[40] V. Sheng, Rahul Tada, and Abhinav Atla. An empirical study of class noise impacts on supervised learning algorithms and measures.

[41] Nino Shervashidze and Francis Bach. Learning the structure for structured sparsity. *IEEE Transactions on Signal Processing*, 63(18):4894–4902, 2015.

[42] Alexander Statnikov, Nikita I Lytkin, Jan Lemeire, and Constantin F Aliferis. Algorithms for discovery of multiple markov boundaries. *Journal of Machine Learning Research*, 14(Feb):499–566, 2013.

[43] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

[44] Wikipedia contributors. Conditional entropy — Wikipedia, the free encyclopedia, 2020. [Online; accessed 4-October-2020].

[45] Wikipedia contributors. Feature selection — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Feature_selection&oldid=976256335, 2020. [Online; accessed 8-September-2020].

[46] Wikipedia contributors. Functional dependency — Wikipedia, the free encyclopedia, 2020. [Online; accessed 4-October-2020].

[47] Wikipedia contributors. Mutual information — Wikipedia, the free encyclopedia, 2020. [Online; accessed 4-October-2020].

[48] Wikipedia contributors. Structured prediction — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Structured_prediction&oldid=965415307, 2020. [Online; accessed 9-September-2020].

[49] Richard Wu, Aoqian Zhang, Ihab Ilyas, and Theodoros Rekatsinas. Attention-based learning for missing data imputation in holoclean. *Proceedings of Machine Learning and Systems*, pages 307–325, 2020.

[50] Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. Guided data repair. *arXiv preprint arXiv:1103.3103*, 2011.

[51] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, page 35. Nashville, TN, USA, 1997.

[52] Ce Zhang. Deepdive: A data management system for automatic knowledge base construction. 2015.

[53] Lei Zhang, Shuai Wang, and Bing Liu. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1253, 2018.

[54] Juan Zhao, Yiwei Zhou, Xiujun Zhang, and Luonan Chen. Part mutual information for quantifying direct associations in networks. *Proceedings of the National Academy of Sciences*, 113(18):5130–5135, 2016.

APPENDICES

Appendix A

Dataset Description

This appendix has a detailed description of the two components of the Census dataset: The schema, and the constraints.

A.1 Schema

This section presents the list of 120 columns used in the original dataset with their descriptions. The variable definitions used below were given with the dataset and sometimes are not very clear.

	Variable name	Variable definition
1	year	year2013
2	period	round
3	rakam_tattb3ya	vist number
4	group_no	group number
5	urban	urban/rural
6	psu_no	number of the sampling unit
7	fam_ser_fram	serial number of the family in the frame
8	fam1_ser_psu	serial number of the family in primary sampling unit
9	fam_unique_number	serial number of the family in the sample
10	gov	gov

Continued on next page

	Variable name	Variable definition
11	qism	qism
12	shiakha	shiakha
13	fam_ser_sh	serial number of the family in shiakha
14	fam_ser_gov	serial number of the family in governorates
15	person_number	person_number
16	reltohd	Relation to HHH
17	sex	sex
18	birth_mth	Month of birth
19	bir_yr	Year of birth
20	age	age
21	edu_enrol	Education Enrollment
22	educ_st	Education level
23	educ_lastcetif	last Certificate and specialization
24	mar_st	Maritalstatus
25	crwrkm	Did you work during last week at least 1hr(market def.)
26	crwrkm_supp	engage in any work
27	crwrkm_agr	activities to earn money_Agriculture
28	crwrkm_cattle	activities to earn money_cattle
29	crwrkm_dairy	activities to earn money_Dairy products
30	crwrkm_fuel	activities to earn money_fuel Collection
31	crwrkm_vegi	activities to earn money_Vegetables
32	crwrkm_sewing	activities to earn money_Sewing
33	crwrkm_Handcraft	activities to earn money_Handicraft
34	crwrkm_housek	activities to earn money_House keeping
35	crwrkm_paidcraft	activities to earn money_Paid craft
36	crwrkm_selling	activities to earn money_Selling
37	crwrkm_maketing	activities to earn money_marketing
38	crwrkm_constr	activities to earn money_constructing
39	crwrkm_fishing	activities to earn money_fishing
40	crwrkm_learningting	activities to earn money_learning craft
41	everwork	ever worked before
42	crwrks_agri	family consumption _Agriculture(subsistence
43	crwrks_cattle	family consumption _cattle
44	crwrks_dairy	family consumption _Dairy products

Continued on next page

	Variable name	Variable definition
45	crwrks_fuel	family consumption _fuel Collection
46	dsr_to_wrk	Want to work
47	ready_to_wrk	ready to work
48	rsn_ntdsr_wrk	reason of unwillingness to work
49	srch	seek for work
50	srch_gov	Attempts to find work_governmental office
51	srch_pri	Attempts to find work_private office
52	srch_comp	Attempts to find work_recruitment competition
53	srch_direct	Attempts to find work_employer directly
54	srch_wrkplace	Attempts to find work_work place
55	srch_adv1	Attempts to find work_Publish advertisement
56	srch_adv2	Attempts to find work_Submit at advertisement
57	srch_friends	Attempts to find work_Ask friends
58	srch_employer	Attempts to find work_Contact with employer
59	srch_contractor	Attempts to find work_Contact with contractor
60	srch_wait	Attempts to find work_waiting in a gathering place
61	srch_bus	Attempts to find work_own business
62	srch_credit	Attempts to find work_financial resources
63	srch_other	Attempts to find work_other
64	rsn_ntsrch	Why not search
65	expan	Expansion factor
66	responce	Response yes/no
67	NO1	male 6-14
68	NO3	male 15-64
69	NO4	male 65+
70	NO5	male unemp worked bef.
71	NO6	male unemp.
72	NO7	male out LF
73	NO8	male out manpower
74	NO9	male total
75	NO10	female 6-14
76	NO12	female 15-64
77	NO13	female 65+
78	NO14	female unemp worked bef.

Continued on next page

	Variable name	Variable definition
79	NO15	female unemp.
80	NO16	female out LF
81	NO17	female out manpower
82	NO18	female total
83	NO19	total 6-14
84	NO21	total 15-64
85	NO22	total 65+
86	NO23	total unemp worked bef.
87	NO24	total unemp.
88	NO25	total out LF
89	NO26	total out manpower
90	NO27	total total
91	respondant	Respondant
92	firstentry_yr	year of start working
93	crempst	employment status
94	crestabl	working in an establishment
95	employees_no	Number of Employees
96	crecact	economic activity
97	crsectot	working sector
98	croccup	Current job
99	crwrkdur	job duration in years,0 less than 1 yr
100	prwrk	Yes/No Previous job
101	proccup	describe Previous job
102	scoccup	Yes/No second job
103	crwrkhrs_wek	working Hours in week
104	crwrkhrs	working Hours in week (second job)
105	rsnlt35	reason work hr¡35hr
106	wrkchang	Yes/No change your work
107	crstabl	work Stability
108	crsocins	social insurance coverage
109	crhlthins	health insurance coverage
110	contr_type	legal contract type
111	wg_basic_prd	Basic wage period
112	wg_basic	Basic wage amount

Continued on next page

	Variable name	Variable definition
113	wg_avdaily	average daily wage_irregular emp
114	ecact_2d	2_digit level Economic Act code
115	unemp_typ	type of unemployment
116	unemp_lst_occ	last occupation
117	unemp_lseco	last economic activity
118	unemp_prd	unemployment period
119	probleme	problem
120	ecact_str2	level Economic Act code

Table A.1: Breakdown of the variables (columns) in the Census dataset as received from the source.

A.2 Rules

The rules were provided as a part of the dataset and were translated to the Denial Constraints notation below. The notation defines six first order predicates ('EQ' for equality check, 'IQ' for inequality check, 'LT' for numerical less than check, 'GT' for numerical greater than check, 'LTE' for numerical less than or equal check, 'GTE' for numerical greater than or equal check). In addition, a special value of 'NULL' represents the lack of value in a certain cell.

For example, the first rule: t1&EQ(t1.reltohd,'head')&LT(t1.age,17) is a check constraint that involves one tuple, and can be read as: "For any tuple, t1, if it has a 'reltohd' equal to 'head', its 'age' can not be less than 17".

	Rule Denial Constraint
1	t1&EQ(t1.reltohd,'head')&LT(t1.age,17)
2	t1&EQ(t1.reltohd,'spouse')&LT(t1.age,18)
3	t1&EQ(t1.reltohd,'son/daugther')&GT(t1.age,60)
4	t1&EQ(t1.reltohd,'daughter/son-in-law')&LT(t1.age,18)
5	t1&EQ(t1.reltohd,'daughter/son-in-law')&GT(t1.age,60)
6	t1&EQ(t1.reltohd,'brother/sister')&LTE(t1.age,10)
7	t1&EQ(t1.reltohd,'brother/sister')&GTE(t1.age,80)
8	t1&EQ(t1.reltohd,'brother wife')&LT(t1.age,8)
9	t1&EQ(t1.reltohd,'brother wife')&GT(t1.age,63)
10	t1&EQ(t1.reltohd,'ather/mother')&LT(t1.age,43)
11	t1&EQ(t1.mar_st,'married')&LT(t1.age,18)
12	t1&EQ(t1.edu_enrol,'enrolled now')&GTE(t1.age,50)
13	t1&EQ(t1.educ_st,'preparatory')&LTE(t1.age,13)
14	t1&EQ(t1.educ_st,'general secondary')&LTE(t1.age,16)
15	t1&EQ(t1.educ_st,'tech. sec.')&LTE(t1.age,16)
16	t1&EQ(t1.educ_st,'above intermaediate')&LTE(t1.age,18)
17	t1&EQ(t1.educ_st,'university')&LTE(t1.age,20)
18	t1&LT(t1.age,6)&IQ(t1.dsr_to_wrk,'NULL')
19	t1&LT(t1.age,6)&IQ(t1.ready_to_wrk,'NULL')
20	t1&LT(t1.age,6)&IQ(t1.rsn_ntdsr_wrk,'NULL')
21	t1&LT(t1.age,6)&IQ(t1.srch,'NULL')
22	t1&LT(t1.age,6)&IQ(t1.respondant,'NULL')

Continued on next page

	Rule Denial Constraint
23	t1&LT(t1.age,6)&IQ(t1.crempst,'NULL')
24	t1&LT(t1.age,6)&IQ(t1.crestabl,'NULL')
25	t1&LT(t1.age,6)&IQ(t1.employees_no,'NULL')
26	t1&LT(t1.age,6)&IQ(t1.crecact,'NULL')
27	t1&LT(t1.age,6)&IQ(t1.crsectot,'NULL')
28	t1&LT(t1.age,6)&IQ(t1.croccup,'NULL')
29	t1&LT(t1.age,6)&IQ(t1.crwrkdur,'NULL')
30	t1&LT(t1.age,6)&IQ(t1.crwrkhrs_wek,'NULL')
31	t1&LT(t1.age,6)&IQ(t1.rsnlt35,'NULL')
32	t1&LT(t1.age,6)&IQ(t1.wrkchang,'NULL')
33	t1&LT(t1.age,6)&IQ(t1.crstabl,'NULL')
34	t1&LT(t1.age,6)&IQ(t1.crsocins,'NULL')
35	t1&LT(t1.age,6)&IQ(t1.crhlthins,'NULL')
36	t1&LT(t1.age,6)&IQ(t1.contr_type,'NULL')
37	t1&LT(t1.age,6)&IQ(t1.wg_basic_prd,'NULL')
38	t1&LT(t1.age,6)&IQ(t1.wg_basic,'NULL')
39	t1&LT(t1.age,6)&IQ(t1.wg_avdaily,'NULL')
40	t1&LT(t1.age,6)&IQ(t1.scoccup,'NULL')
41	t1&LT(t1.age,6)&IQ(t1.prwrk,'NULL')
42	t1&LT(t1.age,6)&IQ(t1.proccup,'NULL')
43	t1&LT(t1.age,6)&IQ(t1.unemp_typ,'NULL')
44	t1&LT(t1.age,6)&IQ(t1.unemp_lst_occ,'NULL')
45	t1&LT(t1.age,6)&IQ(t1.unemp_lseco,'NULL')
46	t1&LT(t1.age,6)&IQ(t1.unemp_prd,'NULL')
47	t1&EQ(t1.reltohd,'spouse')&EQ(t1.mar_st,'never married')
48	t1&EQ(t1.reltohd,'spouse')&EQ(t1.mar_st,'contract married')
49	t1&EQ(t1.reltohd,'spouse')&EQ(t1.mar_st,'divorced')
50	t1&EQ(t1.reltohd,'spouse')&EQ(t1.mar_st,'widowed')
51	t1&EQ(t1.reltohd,'daughter/son-in-law')&EQ(t1.mar_st,'never married')
52	t1&EQ(t1.reltohd,'daughter/son-in-law')&EQ(t1.mar_st,'contract married')
53	t1&EQ(t1.reltohd,'daughter/son-in-law')&EQ(t1.mar_st,'divorced')
54	t1&EQ(t1.reltohd,'brother wife')&EQ(t1.mar_st,'never married')
55	t1&EQ(t1.reltohd,'brother wife')&EQ(t1.mar_st,'divorced')
56	t1&EQ(t1.reltohd,'ather/mother')&EQ(t1.mar_st,'never married')

Continued on next page

	Rule Denial Constraint
57	t1&EQ(t1.reltohd,'ather/mother')&EQ(t1.mar_st,'contract married')
58	t1&EQ(t1.reltohd,'ather/mother')&EQ(t1.mar_st,'divorced')
59	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'primary')
60	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'preparatory')
61	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'general secondary')
62	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'azhar secondary')
63	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'tech. sec.')
64	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'above intermaediate')
65	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'university')
66	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.educ_st,'above university')
67	t1&EQ(t1.edu_enrol,'less than 6 years')&EQ(t1.rsn_ntdsr_wrk,'student')
68	t1&EQ(t1.edu_enrol,'no enrolled')&EQ(t1.rsn_ntdsr_wrk,'student')
69	t1&EQ(t1.edu_enrol,'completed')&EQ(t1.rsn_ntdsr_wrk,'student')
70	t1&EQ(t1.edu_enrol,'enrolled but not finished')&EQ(t1.rsn_ntdsr_wrk,'student')
71	t1&EQ(t1.dsr_to_wrk,'no')&IQ(t1.ready_to_wrk,'NULL')
72	t1&EQ(t1.dsready,'1')&EQ(t1.rsn_ntdsr_wrk,'NULL')
73	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch,'NULL')
74	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_gov,'NULL')
75	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_pri,'NULL')
76	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_comp,'NULL')
77	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_direct,'NULL')
78	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_wrkplace,'NULL')
79	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_adv1,'NULL')
80	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_adv2,'NULL')
81	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_friends,'NULL')
82	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_employer,'NULL')
83	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_contractor,'NULL')
84	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_wait,'NULL')
85	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_bus,'NULL')
86	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_credit,'NULL')
87	t1&EQ(t1.ready_to_wrk,'NULL')&IQ(t1.srch_other,'NULL')
88	t1&IQ(t1.notsrch,'1')&IQ(t1.rsn_ntsrch,'NULL')
89	t1&EQ(t1.work,'2')&IQ(t1.crempst,'NULL')
90	t1&EQ(t1.rsnlt35,'NULL')&LT(t1.crwrkhrs_wek,35)

Continued on next page

	Rule Denial Constraint
91	t1&EQ(t1.crsectot,'government')&EQ(t1.crempst,'self-employed')
92	t1&EQ(t1.crempst,'NULL')&IQ(t1.wg_basic_prd,'NULL')
93	t1&EQ(t1.crempst,'NULL')&IQ(t1.wg_basic,'NULL')
94	t1&EQ(t1.crempst,'NULL')&IQ(t1.wg_avdaily,'NULL')
95	t1&EQ(t1.crstabl,'seasonal')&IQ(t1.wg_basic,'NULL')
96	t1&EQ(t1.crstabl,'casual')&IQ(t1.wg_basic,'NULL')
97	t1&EQ(t1.crstabl,'NULL')&IQ(t1.wg_basic,'NULL')
98	t1&EQ(t1.crstabl,'seasonal')&IQ(t1.wg_basic_prd,'NULL')
99	t1&EQ(t1.crstabl,'casual')&IQ(t1.wg_basic_prd,'NULL')
100	t1&EQ(t1.crstabl,'NULL')&IQ(t1.wg_basic_prd,'NULL')
101	t1&EQ(t1.work,'2')&IQ(t1.prwrk,'NULL')
102	t1&IQ(t1.proccup,'NULL')&EQ(t1.prwrk,'NULL')
103	t1&EQ(t1.srchwrk,'0')&IQ(t1.unemp_typ,'NULL')
104	t1&EQ(t1.srchwrk,'0')&IQ(t1.unemp_lst_occ1,'NULL')
105	t1&EQ(t1.srchwrk,'0')&IQ(t1.unemp_prd,'NULL')
106	t1&EQ(t1.srchwrk,'0')&IQ(t1.unemp_lseco1,'NULL')
107	t1&LT(t1.star_unemp,6)&EQ(t1.unemp_typ,'never worked')
108	t1&EQ(t1.person_number,'1')&IQ(t1.reltohd,'head')
109	t1&EQ(t1.reltohd,'spouse')&EQ(t1.sex,'male')
110	t1&EQ(t1.reltohd,'spouse')&IQ(t1.mar_st,'married')
111	t1&IQ(t1.educ_stage_comp,'NULL')&LT(t1.age,6)
112	t1&EQ(t1.educ_stage_comp,'preb')&LT(t1.age,12)
113	t1&EQ(t1.educ_stage_comp,'general secondary')&LT(t1.age,15)
114	t1&EQ(t1.educ_stage_comp,'azhar secondary')&LT(t1.age,15)
115	t1&EQ(t1.educ_stage_comp,'tech. sec.')&LT(t1.age,15)
116	t1&EQ(t1.educ_stage_comp,'above intermaediate')&LT(t1.age,18)
117	t1&EQ(t1.educ_stage_comp,'university')&LT(t1.age,18)
118	t1&EQ(t1.educ_stage_comp,'above university')&LT(t1.age,22)
119	t1&EQ(t1.educ_stage_comp,'primary')&GT(t1.educ_class_comp,6)
120	t1&EQ(t1.educ_stage_comp,'preb')&GT(t1.educ_class_comp,3)
121	t1&EQ(t1.educ_stage_comp,'general secondary')&GT(t1.educ_class_comp,3)
122	t1&EQ(t1.educ_stage_comp,'azhar secondary')&GT(t1.educ_class_comp,3)
123	t1&EQ(t1.educ_stage_comp,'tech. sec.')&GT(t1.educ_class_comp,5)
124	t1&EQ(t1.educ_stage_comp,'above intermaediate')&GT(t1.educ_class_comp,4)

Continued on next page

	Rule Denial Constraint
125	t1&EQ(t1.educ_stage_comp,'university')&GT(t1.educ_class_comp,6)
126	t1&EQ(t1.edu_enrol,'less than 6 years')&IQ(t1.educ_stage_comp,'NULL')
127	t1&EQ(t1.edu_enrol,'no enrolled')&IQ(t1.educ_stage_comp,'NULL')
128	t1&EQ(t1.edu_enrol,'completed')&IQ(t1.educ_stage_comp,'NULL')
129	t1&EQ(t1.edu_enrol,'less than 6 years')&IQ(t1.educ_class_comp,'NULL')
130	t1&EQ(t1.edu_enrol,'no enrolled')&IQ(t1.educ_class_comp,'NULL')
131	t1&EQ(t1.edu_enrol,'completed')&IQ(t1.educ_class_comp,'NULL')

Table A.2: Breakdown of the rules (defined as denial constraints) in the Census dataset.