Structured Prediction on Dirty Datasets

by

Alireza Heidarikhazaei

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
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.

External Examiner: **Yeye He**
Principal Researcher, Microsoft Research

Supervisor(s): **Ihab F. Ilyas**
Professor, School of Computer Science, University of Waterloo

Internal Members: **M. Tamer Özsu**
Professor, School of Computer Science, University of Waterloo
Yaoliang Yu
Assistant Professor, School of Computer Science, University of Waterloo

Internal-External Member: **Lukasz Golab**
Professor, Department of Management Sciences, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the 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.

Statement of Contributions

This thesis consists of material all of which I authored or co-authored. Chapter 2 is based on a joint work with Joshua McGrath, Ihab F. Ilyas and Theodoros Rekatsinas [96]. Chapter 3 is based on the joint work with Ihab F. Ilyas and Theodoros Rekatsinas [94]. Chapter 4 is a joint work with Ester Livshits, Ihab F. Ilyas and Benny Kimelfeld [136]. Chapter 5 is a joint work with Shrinu Kushagra, Ihab F Ilyas [95]. Chapter 6 is a joint work with George Michalopoulos, Shrinu Kushagra, Ihab F Ilyas and Theodoros Rekatsinas [97].

Abstract

Real-world datasets contain errors, missing values, and poor representations. Data cleaning is a critically important step in any data-related task. Over the past few decades, many algorithms and systems have been presented to clean data. However, most of these solutions are either pure statistical or machine learning models that do not consider the underlying structure of the data or they are rule-based logical methods. In both cases, they fall short in effectively cleaning and finding errors in structured data sets.

Many errors cannot be detected or repaired without taking into account the underlying structure and dependencies in the dataset. One way of modeling the structure of the data is graphical models. Graphical models combine probability theory and graph theory in order to address one of the key objectives in designing and fitting probabilistic models, which is to capture dependencies among relevant random variables. Structure representation helps to understand the side effect of the errors or it reveals correct interrelationships between data points. Hence, principled representation of structure in prediction and cleaning tasks of dirty data is essential for the quality of downstream analytical results. Existing structured prediction research considers limited structures and configurations, with little attention to the performance limitations and how well the problem can be solved in more general settings where the structure is complex and rich.

In this dissertation, I present the following thesis: By leveraging the underlying dependency and structure in machine learning models, we can effectively detect and clean errors via pragmatic structured predictions techniques. To highlight the main contributions: I investigate prediction algorithms and systems on dirty data with a more realistic structure and dependencies to help deploy this type of learning in more pragmatic settings. Specifically, We introduce a few-shot learning framework for error detection that uses structure-based features of data such as denial constraints violations and Bayesian network as co-occurrence feature. I have studied the problem of recovering the latent ground truth labeling of a structured instance. Then, I consider the problem of mining integrity constraints from data and specifically using the sampling methods for extracting approximate denial constraints. Finally, I have introduced an ML framework that uses solitary and structured data features to solve the problem of record fusion.

Acknowledgements

Over the past five years, I have received a great deal of support from a number of remarkable individuals. Despite the impossibility of distilling these important relationships into just a few lines, I would like to mention some of the people who have been particularly meaningful throughout my journey.

First of all, I want to thank my advisor *Ihab F. Ilyas*. I am grateful for every bit of wisdom he generously shared with me. He shaped my way of thinking about data science in his unique style that mixes mathematical rigor with an entertaining narrative. While I do not know what kind of systems I will be dealing with in twenty years, I know that part of my job will be to persify it.

I would like to express my sincere thanks to *Theodoros Rekatsinas*, for his incredible patience and kindness. I have been very lucky to work with someone so intellectually supportive.

Next, I would like to thank my committee members. I would like to thank Prof. Tamer Öszu, Prof. Yaoliang Yu, and Prof. Lukasz Golab for spending time reading and providing valuable comments on my dissertation. Finally, I would like to thank my external committee member *Dr. Yeye He*, for his valuable suggestions, which helped me address the practical aspects of my work.

Finally, I am grateful to my family for their unconditional love and encouragement throughout these years. Special thanks to Banoo, my mother, for always believing in me. I would like to dedicate this work to them.

Dedication

To my mother, Farhad, and Fereshte and in memory of my father

Table of Contents

List of Figures

2figure.caption.5

List of Tables

Chapter 1

Introduction

Businesses have been generating and collecting more data than ever in order to build significant data sources to enable powerful models and analytics. Using significant data enriches the accuracy of downstream processes. Big data is a term for any collection of large or complex datasets that is difficult to process using traditional data management techniques. Data collection and acquisition often introduce errors in data, e.g., missing values, typos, mixed formats, replicated entries for the same real-world entity, and violations of business and data integrity rules. Also, such errors are expected to grow as the size of data increases. Since data collection is often error-prone, establishing trust in data is a challenge.

With the popularity of data science, it has become increasingly evident that data curation, unification, preparation, and cleaning are critical enablers in unleashing the value of data [108]. Gartner predicted that more than 25% of critical data in the world's top companies is flawed [191]. Poor data across businesses and the government costs the U.S. economy $3.1 trillion a year [43]. In general, data cleaning is a process or a chain of processes that detect data errors and repair them by predicting the correct values.

Developing effective and efficient data cleaning solutions is challenging and requires the solution of deep theoretical and engineering problems. The purpose of data cleaning is to increase the quality level of data such that it can reliably be used for the production of statistical models or statements. Regardless of the type of data errors, data cleaning activities usually consist of four phases:

- Data Preparation and acquisition, where change data representation to be used for analytical tasks; domain experts' suggestions help to select the data curation approach.

- Error detection, where various errors and violations are identified and possibly validated by experts.
- Error repair, where updates to the database are applied (or suggested to human experts) to bring the data to a cleaner state suitable for downstream applications and analytics.
- Feedback, where a result-based module suggests changes on any previous modules to improve the performance of the data quality system.

1.1 Structured Prediction and Data Cleaning

Regardless of the data model (relational, graph, RDF, etc.), the values in a database are not independent of each other. Consider that each attribute/entity pair (e.g., a cell in a relational table) is a node in a graph. The edge between two nodes represents a dependency that governs how their values are assigned—these graph structure models include constraints such as keys, and functional dependencies.

Naïve prediction techniques ignore this underlying structure or assume that the structure is simple. We leverage the underlying rich structure in this thesis. Expressive structures diminish the approximation errors, so they enrich the prediction processes (Figure 1.1).

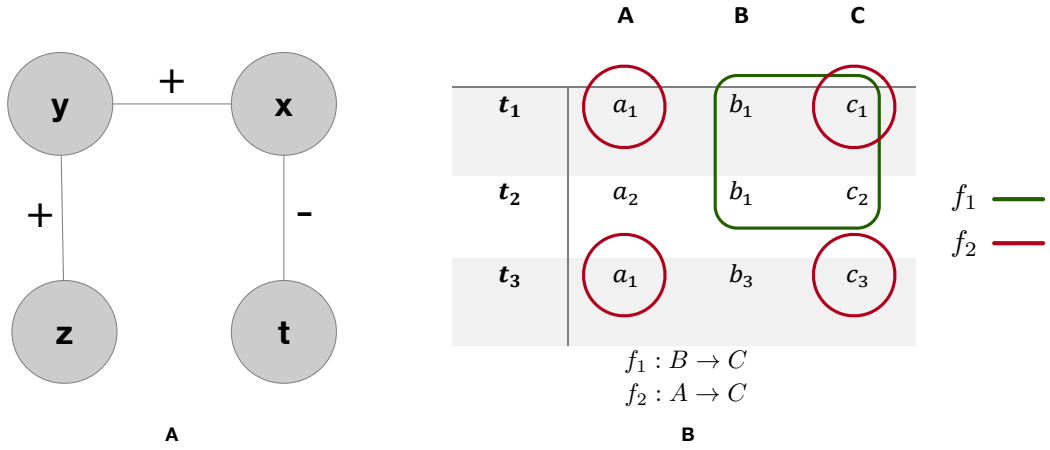


Figure 1.1: Underlying data structure examples. Data constraints specify hyper-edges between cell values indicating more complex dependency between cell values. (Edges with "+" indicate variables here to take the same value, while "−" indicate different values.)

In this thesis "structure" refers to the "dependency" among different units of data (e.g., cells, properties,...) as the underlying structure. This is in contrast to the modeling of data structure (e.g., relations, graphs, JSON records) that is used to represent the data.

1.2 Approximation on Data Cleaning

Data approximation is a recent, fast-growing research area. It deals with the problem of reconstructing an unknown function from given complex data. Naturally, it has many applications, such as surface reconstruction [114], database operations [111, 143] , the numerical solution of partial differential equations [74, 181], kernel learning [188], and parameter estimation [10], to name a few. These applications come from different fields such as applied mathematics, computer science, biology, etc. Especially in the last decades, with appearance on large datasets, data approximation becomes a more challenging topic. In practical applications over a wide field of study one often faces the problem of reconstructing an unknown function f from a finite set of discrete data. These data consist of data sites $X = \{x_1, \ldots, x_N\}$ and data values $f_j = f(x_j), 1 \leq j \leq N$, and the reconstruction has to approximate the data values at the data sites. In other words, a function s is sought that either interpolates the data, i.e. that satisfies $s(x_j) = f_j, 1 \leq j \leq N$, or at least approximates the data, $s(x_j) \approx f_j$.

An end-to-end data cleaning process starts with an optional discovery and profiling step, an error detection step, and an error repair step. When the data is large, approximation can be used in each of these steps. In data profiling, there is a process of discovering denial constraints (DC), a universally quantified first-order logic formalism. Pure schema-driven approaches for DC discovery are less applicable since DCs involve non-symmetric predicates and checking the validity of the space of all DCs systematically is difficult. FASTDC [39] is an instance-driven algorithm. It follows a depth-first search procedure to exhaustively search for all minimal set covers and includes multiple optimizations and pruning opportunities based on the properties of DCs to speed up the search procedure. FASTDC is able to prune the search space, that the number of returned DCs can still be too large. Thus, it uses a scoring function to approximate the value of the DCs based on their size and their support from the data. In data repair, the HoloClean [177] system uses the observed dataset to build a probabilistic model capable of predicting the most likely value for cells identified as (possibly) noisy. It uses factor graphs to encode various kinds of features and inputs to the repair process, such as denial constraints, statistical properties, minimality, and external master data. However, the factor graph on denial constraints represents hyper-dimensional functions, for which it is a hard to find the optimal point. It also needs more data to learn. Holoclean approximates these factor graphs by relaxing denial constraints. SampleClean [123] targets the problem of answering aggregate queries when

the input data is dirty. Since cleaning a large dirty dataset is usually time-consuming and requires human expertise, SampleClean aims at answering a query only by cleaning a sample of the dirty dataset while providing confidence interval guarantees for the query results. So, it approximates the answer of the query by choosing a subset of the data. It assumes that there are two types of errors in the input: attribute error and duplication error. A row in a relational table is said to have an attribute error if one of the attributes has a missing or incorrect value. A relational table is said to have an duplication error if there exist a row with a duplication. The challenge is to remove the effect of the duplication from sampled data.

1.3 Structured Prediction

Structured prediction refers to machine learning models that predict multiple interrelated and dependent quantities. These models are commonly used in computer science to accurately reflect prior knowledge, task-specific relations, and constraints [153]. They are expressive and powerful, but exact computation in these models is often intractable. This difficulty, paired with the practical significance, has resulted in a broad research effort in recent years to design structured prediction models and approximate inference and learning procedures that are computationally efficient.

Loss Functions and Decision Rules A generally accepted standard for evaluating the quality of a given model is that of the expected loss of the model as a function of the true generating probability distribution $q(x, y)$ and a loss function1 $l : \mathcal{Y}(x) \times \mathcal{Y}(x) \to \mathbb{R}$, where $\mathcal{Y}$ refers to the finite but exponentially large output space of the models. The distribution $q(x, y)$ is distribution for sampling we encounter when using the model $y = f(x)$, where $y \in \mathcal{Y}(x)$ is a our model prediction. q is unknown, but a convention assumption is that we can obtain independent and identically distributed (iid) samples from it. The loss function $l(z, y)$ quantifies—on an arbitrary but fixed scale—the loss suffered if z happens to be the truth, and we decide y. The characteristic of a structured prediction model can now be quantified as the risk,

$$\mathcal{R}(f, q, l) = \mathbb{E}_{(x,y)\sim q}[l(y, f(x))]$$

Because $\mathcal{R}$ depends on the unknown distribution q, the expectation is approximated using a dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1,\dots,N}$ sampled iid from q, yielding the empirical risk,

$$\mathcal{R}_{emp}(f, q, l) = \frac{1}{N} \sum_{i=1}^{N} l(y^{(i)}, f(x)^{(i)}) \tag{1.1}$$

While there are different opinions concerning how to build structured prediction models and which loss function to apply for a given application, equation 1.1 is widely adopted. The best possible risk, which is the lowest possible, is recognized as the Bayes risk. It is defined by making the optimal decisions with the information of q, that is, $\mathcal{R}_{Bayes}(q, l) = \mathcal{R}(f_{Bayes}, q, l)$, where f_{Bayes} is the Bayes-optimal predictor,

$$f_{Bayes}(f, q, l) = \arg \min_{y \in \mathcal{Y}(x)} \mathbb{E}_{z \sim q(z|x)}[l(z, y)]. \tag{1.2}$$

Evaluation For expressing function f, different options exist. One approach is to represents $f(x)$ as the maximizer of an auxiliary optimization problem (which can be considered as a potential function),

$$f(x) = \arg \max_{y \in \mathcal{Y}(x)} F(x, y, \theta), \tag{1.3}$$

where $\theta \in \Theta$ are model parameters. In many utilization, solving equation 1.3 corresponds to solving a combinatorial optimization problem. The function $F(x, y, \theta)$ to be maximized is commonly parameterized as a linear form,

$$F(x, y, \theta) = \langle \varphi(x, y), \theta \rangle \tag{1.4}$$

where $\Theta = \mathbb{R}^d$ and $\varphi(x, y)$ is a joint feature map, transforming x and y into a large but fixed-size feature vector. The class of functions is now indexed by θ, and we have

$$\mathcal{F} = \{F(., ., \theta) | \theta \in \mathbb{R}^d\} \tag{1.5}$$

Another strategy to constructing structured prediction functions is starting with a probabilistic model and applying Bayesian decision theory [13]. For this, we assume that we have a model for the conditional distribution $p(y|x; \theta)$ over $\mathcal{Y}(x)$. Given a loss function l, we can then use the Bayes decision rule,

$$f(x) = \arg \min_{y \in \mathcal{Y}(x)} \mathbb{E}_{z \sim q(z|x)}[l(z, y)] \tag{1.6}$$

This rule is the same as equation 1.2, except the unknown distribution q is replaced with model p. Intuitively equation 1.6 chooses our prediction so that we minimize our expected loss under every feasible z, weighted by our beliefs about the state of the world as encoded in $p(z|x)$. The

association between equations 1.6 and 1.2 implies that if p equals the correct distribution q, then our decisions conceived using the Bayes decision rule will be optimal; that is, they will obtain the Bayes risk.

Using either equation 1.2 or 1.3, we can define the structured prediction. In the following, we define it using equation 1.2.

Definition 1.3.1 (Structured Prediction)**.** *Given an observation $x \in X$, make a prediction $y \in \mathcal{Y}(x)$ as $y = f(x)$. $\mathcal{Y}(x)$ is typically finite but exponentially large. For representing the function f, defines it as the maximizer of an auxiliary optimization problem in equation 1.3, where $\theta \in \Theta$ are model parameters. Depending on the structure of F and $\mathcal{Y}(x)$, we need to develop approximate inference methods. The models can be learned with* regularized risk minimization *from a given dataset of i.i.d. samples by minimizing regularized empirical risk,*

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \Omega(f) + \frac{1}{N} \sum_{i=1}^{N} l(y^{(i)}, f(x^{(i)})). \tag{1.7}$$

Here $\Omega(f)$ is a regularizer that controls the capacity of the learned model $\hat{f}$.

Globerson [82] considers the statistical problem of recovering a hidden "ground truth" binary labeling for the vertices V of a graph up to low Hamming error from noisy edge and vertex measurements. Hamming error for $f \in \mathcal{F}$ defines as follow,

$$L = \sum_{v \in V} \mathbb{1}(y^{(v)} \neq f(x^{(v)})) \tag{1.8}$$

The structure complexity in this research is limited to square grid lattices. It has been proven that the statistical complexity of the problem is essentially determined by the number of cuts with a cutset of size k, where k ranges over non-negative integers. This observation, together with clever use of planar duality, enables the determination of the optimal Hamming error for the square grid. Chen et al. [32] have recently considered exact recovery for edges in this setting for sparse graphs such as grid and rings. They consider the case where there are multiple i.i.d observations of edge labels. In contrast, we focus on the case where there is a single (noisy) observation for each edge, on side information, and on partial recovery. More recently, Foster et al. [76] introduced an approximate inference algorithm based on tree decompositions that achieve low expected Hamming error for general graphs with bounded tree-width.

1.4 Challenges for Structured Prediction on Dirty Data

In general, the task of structured prediction on dirty datasets introduces multiple challenges involving *scalability*, *cleaning accuracy*, and *the supported complexity of structure patterns*. To achieve scalability, a common practice is to consider simple or small graph structures. Considering simple structures decreases accuracy by eliminating essential information that can save as important context in recovering wrong and missing values. Statistical methods cannot be used for small structures since there is insufficient data to support the learning models and learning accuracy decreases. On the other hand, Statistical models can achieve an accurate prediction on complex structures that can only apply to small datasets. For each specific task, it is unknown how to balance accuracy and scalability, or even how to develop data cleaning models for simple structures like trees [82]. Despite the high accuracy of the traditional methods on relational data, converting structure information to tables is not promising because of unbounded dimensional reduction [54].

1.5 Dissertation's Hypothesis

In this dissertation, we consider structured prediction as the source of revealing hidden properties of data that describe interrelationship among data records (what we refer to as structure); and propose that data cleaning tasks can achieve high-quality results by considering these features. The solutions presented in this dissertation adopt the following principles:

- *Using noise-free training data allows the learning algorithm to be trained from the correct version of the data generator.* In agnostic learning, the training data might have some errors due to the effects of noise, the total error is impacted. In this work, we have assumed that the domain experts that generate training data do not introduce any data annotation errors.
- *Domain experts provide correct data integrity rules.* Some algorithms use a set of data rules that help models to understand the underlying structure of the records. Errors in these rules leads to structural errors, which is out of the scope of this dissertation.
- *The data distribution is fixed during models training and prediction and the given data is obtained by uniform sampling from the support of this distribution.* Data generators create data with a distribution that can be fixed or change over time. In this dissertation, the assumption is that the distribution is fixed.

1.6 Contributions and Outline

This thesis proposes frameworks for data cleaning on structured data, with guarantees for cleaning accuracy and scalability, while preserving the underlying structure. The contributions are the following:

1. **Error detection using structured featurization** Data errors are common in all real-world datasets. Errors are generated in different ways. Heterogeneity of errors makes error detection a difficult task. Previous solutions focus on a subset of the side effects and properties of the data. A more effective approach is to leverage data-dependency and structure-based properties. We develop practical, scalable error detection models in the presence of pragmatic challenges such as the lack of sufficient training data to learn the structure and the prediction model [96]. (Chapter 2)

2. **Correctness of inference using structured information** Using graphical models on a dataset with categorical attributes has been popular in the last few years. However, there is no good theoretical work to support non-binary attributes. We address this problem and fulfill the missing theoretical piece. For a given structure complexity, the effect of statistical inference and approximation on scalability and data cleaning accuracy are investigated [94]. (Chapter 3)

3. **Sampling from data with structured rules** Integrity constraints represent graphical structures over the dataset. Often, however, these rules are not explicitly provided and have to be discovered from the given data instance [39]. We propose a method to mine approximate denial constraints (representing the underlying structure) to be fed explicitly into data cleaning models that can leverage this structure [136]. We also provide a theoretical analysis on the proposed algorithm. (Chapter 4)

4. **Sampling method for data with structured noise** It is common to sample large datasets and use an unbiased estimator to measure estimation. When the dataset has noise that can influence the structure of the data, simple sampling schemes do not work. Data duplication is an example of errors that changes the structure of the data. The problem of sampling from data with duplicate records defines a structure over data points where each pair representing the same real-world entity has an edge connecting them. Therefore, each connected component corresponds to one real-world entity, and its size directly affects the probability of being selected in the sample. This problem has theoretical and experimental aspects. We develop practical sampling methods on dirty data based on theoretical analysis that avoid the expensive data cleaning task. (Chapter 5)

5. **Framework for fusing datasets using structured featurization** When multiple datasets representing a set of real-world entities are merged, conflicting data require a *decision process* to select the correct value(s) [59, 60]. Many of the previous *data fusion* algorithms assume a set of homogeneous features and use the concept of *source trustworthiness*. We propose a new model for the record fusion problem that uses structured prediction to predict the canonical representation of entities (cluster of records). (Chapter 7)

Chapter 2

HoloDetect: Few-Shot Learning for Error Detection

Error detection is a natural first step in every data analysis pipeline [107, 157]. Data inconsistencies due to incorrect or missing data values can have a severe negative impact on the quality of downstream analytical results. However, identifying errors in a noisy dataset can be a challenging problem. Errors are often heterogeneous and exist due to a diverse set of reasons (e.g., typos, integration of stale data values, or misalignment), and in many cases can be rare. This makes manual error detection prohibitively time consuming.

Several error detection methods have been proposed in the literature to automate error detection [56, 68, 107, 166]. Most of the prior works leverage the side effects of data errors to solve error detection. For instance, many of the proposed methods rely on violations of integrity constraints [107] or value-patterns [113] or duplicate detection [66, 149] and outlier detection [47,165,201] methods to identify erroneous records. While effective in many cases, these methods are tailored to specific types of side effects of erroneous data. As a result, their recall for identifying errors is limited to errors corresponding to specific side effects (e.g., constraint violations, duplicates, or attribute/tuple distributional shifts) [2].

One approach to address the heterogeneity of errors and their side effects is to combine different detection methods in an ensemble [2]. For example, given access to different error detection methods, one can apply them sequentially or can use voting-based ensembles to combine the outputs of different methods. Despite the simplicity of ensemble methods, their performance can be sensitive to how different error detectors are combined [2]. This can be either with respect to the order in which different methods are used or the confidence-level associated with each method. Unfortunately, appropriate tools for tuning such ensembles are limited, and the burden

of tuning these tools is on the end-user.

A different way to address heterogeneity is to cast error detection as a machine learning (ML) problem, i.e., a binary classification problem: given a dataset, classify its entries as erroneous or correct. One can then train an ML model to discriminate between erroneous and correct data. Beyond automation, a suitably expressive ML model should be able to capture the inherent heterogeneity of errors and their side effects and will not be limited to low recall. However, the end-user is now burdened with the collection of enough labeled examples to train such an expressive ML model.

2.1 Challenges Overview

We propose a few-shot learning framework for error detection based on weak supervision [172, 176], which exploits noisier or higher-level signals to supervise ML systems. We start from this premise and show that *data augmentation* [162, 206], a form of weak supervision, enables us to train high-quality ML-based error detection models with minimal human involvement.

Our approach exhibits significant improvements over a comprehensive collection of error detection methods: we show that our approach is able to detect errors with an average precision of $\sim$94% and an average recall of $\sim$93%, obtaining an average improvement of 20 F_1 points against competing error detection methods. At the same time, our weakly supervised methods require access to $3\times$ fewer labeled examples compared to other ML approaches. Our ML-approach also needs to address multiple technical challenges:

- **[Model]** The heterogeneity of errors and their side effects makes it challenging to identify the appropriate statistical and integrity properties of the data that should be captured by a model in order to discriminate between erroneous and correct cells. These properties correspond to attribute-level, tuple-level, and dataset-level features that describe the distribution governing a dataset. Hence, we need an appropriately expressive model for error detection that captures all these properties (features) to maximize recall. We introduce a template ML-model to learn a representation that captures attribute-, tuple-, and dataset-level features that describe a dataset. We demonstrate that representation learning obviates the need for feature engineering. Finally, we show via ablation studies that all granularities need to be captured by error detection models to obtain high-quality results.

- **[Imbalance]** Often, errors in a dataset are limited. ML algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets. The features of the minority

class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class. To deal with imbalance, one needs to develop strategies to balance classes in the training data. Standard methods to deal with the imbalance problem such as resampling can be ineffective due to error heterogeneity as we empirically show in our experimental evaluation. We show how to use data augmentation to address data imbalance. Data augmentation proceeds as follows: Given a small set of labeled data, it allows us to generate synthetic examples or errors by transforming correct examples in the available training data. This approach minimizes the amount of manually labeled examples required. We show that in most cases a small number of labeled examples are enough to train high-quality error detection models.

- **[Heterogeneity]** Heterogeneity amplifies the imbalance problem as certain errors and their side effects can be underrepresented in the training data. Resampling the training data does not ensure that errors with different properties are revealed to the ML model during training. While active learning can help counteract this problem in cases of moderate imbalance [30, 67], it tends to fail in the case of extreme imbalance [92] (as in the case of error detection). This is because the lack of labels prevents the selection scheme of active learning from identifying informative instances for labeling [92]. Different methods that are robust to extreme imbalance are needed. We present a weakly supervised method to learn data transformations and data augmentation policies (i.e., the distribution over those data transformation) directly from the noisy input dataset. The use of different transformations during augmentation provides us with examples that correspond to different types of errors, which enables us to address the aforementioned heterogeneity challenge.

A solution that addresses the aforementioned challenges needs to: (1) introduce an expressive model for error detection, while avoiding explicit feature engineering; and (2) propose novel ways to handle the extreme imbalance and heterogeneity of data in a unified manner.

2.2 Preliminaries

We review basic background material for the problems and techniques discussed in this chapter.

2.2.1 Error Detection

The goal of error detection is to identify incorrect entries in a dataset. Existing error detection methods can be categorized in three main groups: (1) Rule-based methods [37, 46] rely on

integrity constraints such as functional dependencies and denial constraints, and suggest errors based on the violations of these rules. Denial Constraints (DCs) are first order logic formulas that subsume several types of integrity constraints [35]. Given a set of operators $B = \{=, <, >, \neq, \leq, \geq, \approx\}$, with $\approx$ denoting similarity, DCs take the form $\forall t_i, t_j \in D : \neg(P_1 \wedge \cdots \wedge P_k \wedge \cdots \wedge P_K)$ where D is a dataset with attributes $A = \{A_1, A_2, \ldots, A_N\}$, t_i and t_j are tuples, and each predicate P_k is of the form $(t_i[A_n]\ \mathrm{op}\ \mathrm{t_j[A_m]})$ or $(t_i[A_n]\ \mathrm{op}\ \alpha)$ where $A_n, A_m \in A$, α is a constant and $\mathrm{op} \in \mathrm{B}$. (2) Pattern-driven methods leverage normative syntactic patterns and identify erroneous entries such as those that do not conform with these patterns [113]. (3) Quantitative error detection focuses on outliers in the data and declares those to be errors [98]. A problem related to error detection is record linkage [56, 66, 149], which tackles the problem of identifying if multiple records refer to the same real-world entity. While it can also be viewed as a classification problem, it does not detect errors in the data and is not the focus of this work.

2.2.2 Data Augmentation

Data augmentation is a form of weak supervision [176] and refers to a family of techniques that aim to extend a dataset with additional data points. Data augmentation is typically applied to training data as a way to reduce overfitting of models [206]. Data augmentation methods typically consist of two components: (1) a set of *data transformations* that take a data point as input and generate an altered version of it, and (2) an *augmentation policy* that determines how different transformations should be applied, i.e., a distribution over different transformations. Transformations are typically specified by domain experts while policies can be either pre-specified [162] or learned via reinforcement learning or random search methods [45, 174]. In contrast to prior work, we show that for error detection both transformations and policies can be learned directly from the data.

2.2.3 Representation Learning

The goal of representation learning is to find an appropriate representation of data (i.e., a set of features) to perform a machine learning task [11]. In our error detection model we build upon three standard representation learning techniques:

Neural Networks Representation learning is closely related to neural networks [86]. The most basic neural network takes as input a vector $\mathbf{x}$ and performs an affine transformation of the input $\mathbf{w}\mathbf{x} + b$. It also applies a non-linear activation function σ (e.g., a sigmoid) to produce the output $\sigma(\mathbf{w}\mathbf{x}+b)$. Multiple layers can be stacked together to create more complex networks. In a neural

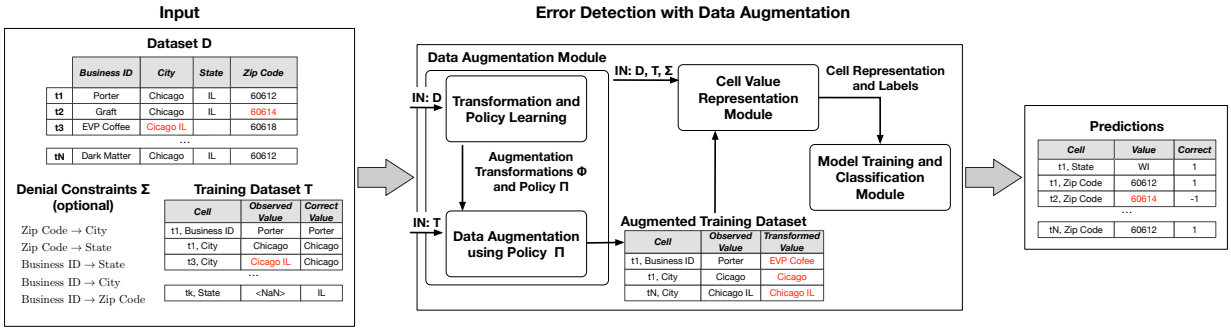


Figure 2.1: Overview of Error Detection with Augmentation.

network, each hidden layer maps its input data to an internal representation that tends to capture a higher level of abstraction.

Highway Neural Networks Highway Networks, adapt the idea of having "shortcut" gates that allow unimpeded information to flow across non-consecutive layers [189]. Highway Networks are used to improve performance in many domains such as speech recognition [207] and language modeling [117], and their variants called Residual networks have been useful for many computer vision problems [93]

Distributed Representations Distributed representations of symbolic data [100] were first used in the context of statistical language model [12]. The goal here is to learn a mapping of a token (e.g., a word) to a vector of real numbers, called a *word embedding*. Methods to generate these mappings include neural networks [145], dimensionality reduction techniques such as PCA [130], and other probabilistic techniques [81].

2.3 Framework Overview

We formalize the problem of error detection and provide an overview of our solution to error detection.

2.3.1 Problem Statement

The goal of our framework is to identify erroneous entries in a relational dataset D. We denote $A = \{A_1, A_2, \ldots, A_N\}$ the attributes of D. We follow set semantics and consider D to be a set of tuples. Each tuple $t \in D$ is a collection of cells $C_t = \{t[A_1], t[A_2], \ldots, t[A_N]\}$ where $t[A_i]$

denotes the value of attribute A_i for tuple t. We use C_D to denote the set of cells contained in D. The input dataset D can also be accompanied by a set of integrity constraints Σ, such as Denial Constraints as described in Section 2.2.1.

We assume that errors in D appear due to inaccurate cell assignments. More formally, for a cell c in C_D we denote by v_c^* its unknown true value and v_c its observed value. We define an *error* in D to be each cell c with $v_c \neq v_c^*$. We define a *training dataset* T to be a set of tuples $T = \{(c, v_c, v_c^*)\}_{c \in C_T}$ where $C_T \subset C_D$. T provides labels (i.e., correct or erroneous) for a subset of cells in D. We also define a variable E_c for each cell $c \in C_D$ with $E_c = -1$ indicating that the cell is erroneous and with $E_c = 1$ indicating that the cell is correct. For each E_c we denote e_c^* its unknown true assignment.

Our goal is stated as follows: given a dataset D and a training dataset T find the most probable assignment $\hat{e}_c$ to each variable E_c with $c \in C_D \setminus C_T$. We say that a cell is correctly classified as erroneous or correct when $\hat{e}_c = e_c^*$.

2.3.2 Model Overview

Prior models for error detection focus on specific side effects of data errors. For example, they aim to detect errors by using only the violations of integrity constraints or aim to identify outliers with respect to the data distribution that are introduced due to errors. Error detectors that focus on specific side effects, such as the aforementioned ones, are not enough to detect errors with a high recall in heterogeneous datasets [3]. This is because many errors may not lead to violations of integrity constraints, nor appear as outliers in the data. We propose a different approach: we model the process by which the entries in a dataset are generated, i.e., we model the distribution of both correct and erroneous data. This approach enables us to discriminate better between these two types of data.

We build upon our recent Probabilistic Unclean Databases (PUDs) framework that introduces a probabilistic framework for managing noisy relational data [182]. We follow the abstract generative model for noisy data from that work, and introduce an instantiation of that model to represent the distribution of correct and erroneous cells in a dataset.

We consider a noisy channel model for databases that proceeds in two steps: First, a clean database is sampled from a probability distribution I^*. Distribution I^* captures how values within an attribute and across attributes are distributed and also captures the compatibility of different tuples (i.e., it ensures that integrity constraints are satisfied). To this end, distribution I^* is defined over attribute-, tuple-, and dataset-level features of a dataset. Second, given a clean database sampled by I^*, errors are introduced via a noisy channel that is described by a

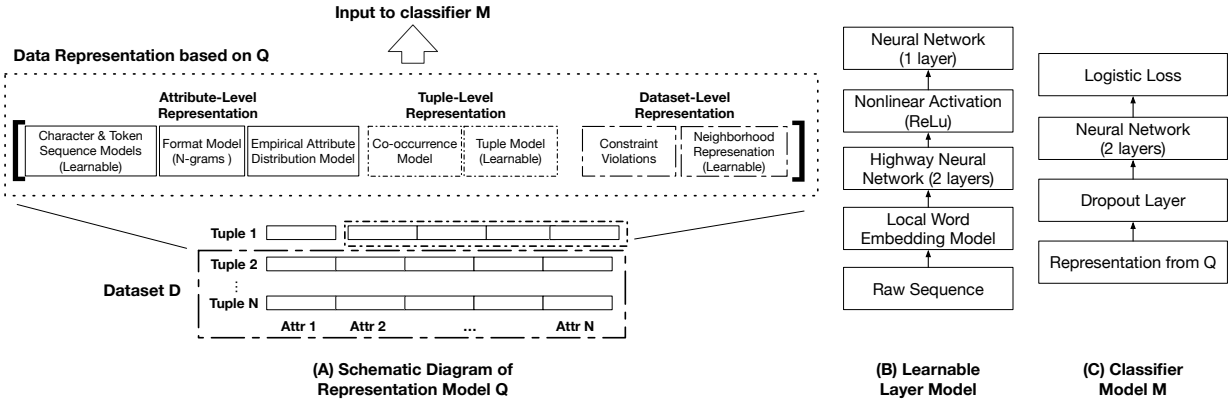


Figure 2.2: (A) A diagram of the representation model Q. Models associated learnable layers that are jointly trained with classifier M. (B) Architecture diagram of the learnable layers in Q. (C) The architecture of classifier M.

conditional probability distribution R^*. Given this model, I^* characterizes the probability of the unknown true value $P(v_c^*)$ of a cell c and R^* characterizes the conditional probability $P(v_c|v_c^*)$ of its observed value. Distribution I^* is such that errors in dataset D lead to low probability instances. For example, I^* assigns zero probability to datasets with entries that lead to constraint violations.

The goal is to learn a representation that captures the distribution of the correct cells (I^*) and how errors are introduced (R^*). Our approach relies on learning two models:

(1) *Representation Model* We learn a representation model Q that approximates distribution I^* on the attribute, record, and dataset level. We require that Q is such that the likelihood of correct cells given Q will be high, while the likelihood of erroneous cells given Q is low. This property is necessary for a classifier M to discriminate between correct and erroneous cells when using representation Q. We rely on representation learning techniques to learn Q jointly with M.

(2) *Noisy Channel* We learn a generative model H that approximates distribution R^*. This model consists of a set of transformations Φ and a policy Π. Each transformation $\varphi \in \Phi$ corresponds to a function that takes as input a cell c and transforms its original value v_c to a new value v'_c, i.e., $\varphi(v_c) = v'_c$. Policy Π is defined as a conditional distribution $P(\Phi|v_c)$. As we describe next, we use this model to generate training data—via data augmentation—for learning Q and M.

We now present the architecture of our framework. The modules described next are used to learn the noisy channel H, perform data augmentation by using H, and learn the representation model Q jointly with a classifier M that is used to detect errors in the input dataset.

2.3.3 Framework Overview

Our framework takes as input a noisy dataset D, a training dataset T, and (optionally) a set of denial constraints Σ. To learn H, Q, and M from this input we use three core modules:

Module 1: Data Augmentation This module learns the noisy channel H and uses it to generate additional training examples by transforming some of the labeled examples in T. The output of this module is a set of additional examples T_H. The operations performed by this module are:

(1) *Transformation and Policy Learning:* The goal here is to learn the set of transformations Φ and the policy Π that follow the data distribution in D. We introduce a weakly supervised algorithm to learn Φ and Π. This algorithm is presented in Section 2.5.

(2) *Example Generation:* Given transformations Φ and policy Π, we generate a set of new training examples T_H that is combined with T to train the error detection model. To ensure high-quality training data, this part augments only cells that are marked correct in T. Using this approach, we obtain a balanced training set where examples of errors follow the distribution of errors in D. This is because transformations are chosen with respect to policy Π which is learned from D.

Module 2: Representation This module combines different representation models to form model Q. Representation Q maps a cell values v_c to to a fixed-dimension real-valued vector $f_c \in R^d$. To obtain f_c we concatenate the output of different representation models, each of which targets a specific context (i.e., attribute, tuple, or dataset context).

We allow a representation model to be learned during training, and thus, the output of a representation model can correspond to a vector of variables (see Section 2.4). For example, the output of a representation model can be an embedding u_c obtained by a neural network that is learned during training or may be fixed to the number of constraint violations value v_c participates in.

Module 3: Model Training and Classification This module is responsible for training a classifier M that given the representation of a cell value determines if it is correct or erroneous, i.e., $M : R^d \rightarrow \{\text{“correct” (+1)}, \text{“error (-1)”}\}$. During training, the classifier is learned by using both the initial training data T and the augmentation data T_A. At prediction time, the classifier M takes as input the cell value representation for all cells in $D \setminus T$ and assigns them a label from $\{\text{“correct”}, \text{“error”}\}$ (see Section 2.4).

An overview of how the different modules are connected is shown in Figure 6.3. First, Module 1 learns transformations Φ and policy Π. Then, Module 2 grounds the representation model Q of our error detection model. Subsequently, Q is connected with the classifier model M in Module 3 and trained jointly. The combined model is used for error detection.

2.4 Representations of Dirty Data

We describe how to construct the representation model Q (see Section 2.3.2). We also introduce the classifier model M, and describe how we train Q and M.

2.4.1 Representation Models

To approximate the data generating distribution I^*, the model Q needs to capture statistical characteristics of cells with respect to attribute-level, tuple-level, and dataset-level contexts. An overview of model Q is shown in Figure 2.2(A). As shown, Q is formed by concatenating the outputs of different models. Next, we review the representation models we use for each of the three contexts. The models introduced next correspond to a bare-bone set that captures all aforementioned contexts, and is currently implemented in our prototype. More details on our implementation are provided in Appendix 2.8.1. Our architecture can trivially accommodate additional models or more complex variants of the current models.

Attribute-level Representation: Models for this context capture the distributions governing the values and format for an attribute. Separate models are used for each attribute A_i in dataset D. We consider three types of models: (1) *Character and token sequence models* that capture the probability distribution over sequences of characters and tokens in cell values. These models correspond to learnable representation layers. Figure 2.2(B) shows the deep learning architecture we used for learnable layers. (2) *Format models* that capture the probability distribution governing the format of the attribute. In our implementation, we consider an n-gram model that captures the format sequence over the cell value. Each n-gram is associated with a probability that is learned directly from dataset D. The probabilities are aggregated to a fixed-dimension representation by taking the probabilities associated with the least-k probable n-grams. (3) *Empirical distribution models* that capture the empirical distribution of the attribute associated with a cell. These can be learned directly from the input dataset D. The representation here is a scalar that is the empirical probability of the cell value.

Tuple-level Representation: Models for this context capture the joint distribution of different attributes. We consider two types of models: (1) *Co-occurrence models* that capture the empirical joint distribution over pairs of attributes. (2) A learnable *tuple representation*, which captures the joint distribution across attributes given the observed cell value. Here, we first obtain an embedding of the tuple by following standard techniques based on word-embedding models [18]. These embeddings are passed through a learnable representation layer (i.e., a deep network) that corresponds to an additional non-linear transform (see Figure 2.2(B)). For co-occurrence, we

learn a single representation for all attributes. For tuple embeddings, we learn a separate model per attribute.

Dataset-level Representation: Models for this context capture a distribution that governs the compatibility of tuples and values in the dataset D. We consider two types of models: (1) *Constraint-based models* that leverage the integrity constraints in Σ (if given) to construct a representation model for this context. Specifically, for each constraint $\sigma \in \Sigma$ we compute the number of violations associated with the tuple of the input cell. (2) A *neighborhood-based representation* of each cell value that is informed by a dataset-level embedding of D transformed via a learnable layer. Here, we train a standard word-embedding model where each tuple in D is considered to be a document. To ensure that the embeddings are not affected by the sequence of values across attributes we extend the context considered by word-embeddings to be the entire tuple and treat the tuple as a bag-of-words. These embeddings are given as input to a learnable representation layer that follows the architecture in Figure 2.2(B).

The outputs of all models are concatenated into a single vector that is given as input to Classifier M. Learnable layers are trained jointly with M. To achieve high-quality error detection, features from all contexts need to be combined to form model Q. In Section 4.9, we present an ablation study which demonstrates that all features from all types of contexts are necessary to achieve high-quality results.

2.4.2 Error Classification

The classifier M of our framework corresponds to a two-layer fully-connected neural network, with a ReLU activation layer, and followed by a *Softmax* layer. The architecture of M is shown in Figure 2.2(C). Given the modular design of our architecture, Classifier M can be easily replaced with other models. Classifier M is jointly trained with the representation model Q by using the training data in T and the data augmentation output T_H. We use ADAM [118] to train our end-to-end model.

More importantly, we calibrate the confidence of the predictions of M using *Platt Scaling* [90, 163] on a holdout-set from the training data T (i.e., we keep a subset of T for calibration). Platt Scaling proceeds as follows: Let z_i be the score for class i output by M. This score corresponds to non-probabilistic prediction. To convert it to a calibrated probability, Platt Scaling learns scalar parameters $a, b \in R$ and outputs $\hat{q}_i = \sigma(az_i + b)$ as the calibrated probability for prediction z_i. Here, σ denotes the sigmoid function. Parameters a and b are learned by optimizing the negative log-likelihood loss over the holdout-set. It is important to note that the parameters of M and Q are fixed at this stage.

2.5 Data Augmentation Learning

Having established a representation model Q for the data generating distribution I^*, we now move to modeling the noisy channel distribution R^*. We assume the noisy channel can be specified by a set of transformation functions Φ and a policy Π (i.e., a conditional distribution over Φ given a cell value). Our goal is to learn Φ and Π from few example errors and use it to generate training examples to learn model Q.

2.5.1 Noisy Channel Model

We aim to limit the number of manually labeled data required for error detection. Hence, we consider a simple noisy channel model that can be learned from few and potentially noisy training data. Our noisy channel model treats cell values as strings and introduces errors to a clean cell value v^* by applying a transformation φ to obtain a new value $v = \varphi(v^*)$. We consider that each function $\varphi \in \Phi$ belongs to one of the following three templates:

- Add characters: $\varnothing \longmapsto [a-z]^+$
- Remove characters: $[a-z]^+ \longmapsto \varnothing$
- Exchange characters: $[a-z]^+ \longmapsto [a-z]^+$ (the left side and right side are different)

Given these templates, we assume that the noisy channel model introduces errors via the following generative process: Given a clean input value v^*, the channel samples a transformation φ from a conditional distribution $\Pi(v^*) = P(\Phi|v^*)$, i.e., $\varphi \sim \Pi(v^*)$ and applies φ once to a substring or position of the input cell value. We refer to Π as a *policy*. If the transformation φ can be applied to multiple positions or multiple substrings of v^* one of those positions or strings is selected uniformly at random.

For example, to transform Zip Code "60612" to "606152", the noisy channel model we consider can apply the exchange character function $T : 60612 \longmapsto 606152$, i.e., exchange the entire string. Applying the exchange function on the entire cell value can capture misaligned attributes or errors due to completely erroneous values. However, the same transformed string can also be obtained by applying either the exchange character function $T : 12 \longmapsto 152$ on the '12' substring of "60612" or the add character function $T : \varnothing \longmapsto 5$, where the position between '1' and '2' in "60612" was chosen at random. The distribution that corresponds to the aforementioned generative process dictates the likelihood of each of the above three cases.

Given Φ and Π, we can use this noisy channel on training examples that correspond to clean tuples to augment the available training data. However, both Φ and Π have to be learned from the limited number of training data. This is why we adopt the above simple generative process. Despite its simplicity, we find our approach to be effective during data augmentation (see Section 4.9). Next, we introduce algorithms to learn Φ and Π assuming access to labeled pairs of correct and erroneous values $L = \{(v^*, v)\}$ with $v \neq v^*$. We then discuss how to construct L either by taking a subset of the input training data T or, in the case of limited training data, via an unsupervised approach over dataset D. Finally, we describe how to use Φ and Π to perform data augmentation.

2.5.2 Learning Transformations

We use a pattern matching approach to learn the transformations Φ. We follow a hierarchical pattern matching approach to identify all different transformations that are valid for each example in L. For example, for $(60612, 6061x2)$ we want to extract the transformations $\{60612 \longmapsto 6061x2, 12 \longmapsto 1x2, \varnothing \longmapsto x\}$. The approach we follow is similar to the Ratcliff-Obershelp pattern recognition algorithm [171]. Due to the generative model we described above, we are agnostic to the position of each transformation.

The procedure is outlined in Algorithm 1. Given an example (v^*, v) from L, it returns a list of valid transformations Φ_e extracted from the example. The algorithm first extracts the string level transformation $T : v^* \longmapsto v$, and then proceeds recursively to extract additional transformations from the substrings of v^* and v. To form the recursion, we identify the longest common substring of v^* and v, and use that to split each string into its prefix (denoted by lv^*) and its postfix (denoted by rv^*). Given the prefix and the postfix substrings, we recurse on the combination of substrings that have the maximum similarity (i.e., overlap). We compute the overlap of two strings as $2 * C/S$, where C is the number of common characters in the two strings, and S is the sum of their lengths. Finally, we remove all identity (i.e., trivial) transformations from the output Φ_e. To construct the set of transformations Φ, we take the set-union of all lists Φ_e generated by applying Algorithm 1 to each entry $e \in L$.

2.5.3 Policy Learning

The set of transformations Φ extracted by Algorithm 1 correspond to all possible alterations our noisy channel model can perform on a clean dataset. Transformations in Φ range from specialized transformations for specific entries (e.g., $60612 \longmapsto 6061x2$) to generic transformations, such as $\varnothing \longmapsto x$, that can be applied to any position of any input. Given Φ, the next step is to

Algorithm 1: Transformation Learning (TL)

Input: Example $e = (v^*, v)$ of a correct string and its corresponding erroneous string
Output: A list of valid transformations Φ_e for example e

if $v^* = \varnothing$ and $v = \varnothing$ **return** $\emptyset$;
$\Phi_e \leftarrow [v^* \longmapsto v]$;
$l \leftarrow$ Longest Common Substring(v^*, v);
$lv^*, rv^* \leftarrow v^* \setminus l$ /* Generate left and right substrings */;
$lv, rv \leftarrow v \setminus l$;
if *similarity*(lv^*, lv) + *similarity*$(rv^*, rv) >$ *similarity*(lv^*, rv) + *similarity*(rv^*, lv) **then**
    Add $[lv^* \longmapsto lv,\ rv^* \longmapsto rv]$ in Φ_e;
    Add $[\text{TL}(lv^*, lv),\ \text{TL}(rv^*, rv)]$ in Φ_e;
else
    Add $[lv^* \longmapsto rv,\ rv^* \longmapsto lv]$ in Φ_e;
    Add $[\text{TL}(lv^*, rv),\ \text{TL}(rv^*, lv)]$ in Φ_e;
end
Remove all identity transformations from Φ_e;
return Φ_e

learn the transformation policy Π, i.e., the conditional probability distribution $\Pi(v) = P(\Phi|v)$ for any input value v. We next introduce an algorithm to learn Π.

Algorithm 2: Empirical Transformation Distribution

Input: A set of identified transformation lists $\{\Phi_e\}_{e \in L}$
Output: Empirical Distribution $\hat{\Pi}$

$\Phi \leftarrow$ Set of unique transformations in $\{\Phi_e\}_{e \in L}$;
$c \leftarrow \sum_e$ (element count of Φ_e);
for $\varphi \in \Phi$ **do**
    $c_\varphi \leftarrow$ number of times φ appears in $\{\Phi_e\}_{e \in L}$;
    $p(\varphi) \leftarrow \frac{c_\varphi}{c}$
end
return $\{p(\varphi)\}_{\varphi \in \Phi}$

We approximate Π via a two-step process: First, we compute the empirical distribution of transformations informed by the transformation lists output by Algorithm 1. This process is described in Algorithm 2. Second, given an input string v, we find all transformations $str \longmapsto str'$ in Φ such that str is a subset of v. Let $\Phi_v \subseteq \Phi$ be the set of such transformations. We obtain a distribution $P(\Phi_v|v)$ by re-normalizing the empirical probabilities from the first step.

This process is outlined in Algorithm 3. Recall that we choose this simple model for Π as the number of data points in L can be limited.

Algorithm 3: Approximate Noisy Channel Policy

Input: An empirical transformation $\hat{\Pi}$ over transformations Φ; A string v
Output: Conditional Distribution $\hat{\Pi}(v) = P(\Phi|v)$

$\hat{\Pi}(v) \leftarrow \varnothing$;
$\Phi_v \leftarrow$ Subset of transformations $str \longmapsto str'$ in Φ such that str is a substring of v;
total mass $\leftarrow \sum_{\varphi \in \Phi_v} \hat{\Pi}(\varphi)$;
for $\varphi \in \Phi_v$ **do**
    $\hat{\Pi}(v)[\varphi] \leftarrow \frac{\hat{\Pi}(\varphi)}{\text{total mass}}$;
end
return $\hat{\Pi}(v)$

2.5.4 Generating Transformation Examples

We describe how to obtain examples (v^*, v) to form the set L, which we use in learning the transformations Φ (Section 2.5.2) and the policy $\hat{\Pi}$ (Section 2.5.3). First, any example in the training data T that corresponds to an error can be used. However, given the scarcity of errors in some datasets, examples of errors can be limited. We introduce a methodology based on weak-supervision to address this challenge.

We propose a simple unsupervised data repairing model M_R over dataset D and use its predictions to obtain transformation examples (v^*, v). We form examples $(v^*, v) = (\hat{v}, v)$ with $\hat{v} \neq v$ by taking an original cell value v and the repair $\hat{v}$ suggested by M_R. We only require that this model has relatively high-precision. High-precision implies that the repairs performed by M_R are accurate, and thus, the predictions correspond to true errors. This approach enables us to obtain noisy training data that correspond to *good samples* from the distribution of errors in D. We do not require this simple prediction model to have high recall, since we are only after producing example errors, not repairing the whole data set.

We obtain a simple high-precision data repairing model by training a Naïve Bayes model over Dataset D. Specifically, we iterate over each cell in D, pretend that its value is missing and leverage the values of other attributes in the tuple to form a Naïve Bays model that we use to impute the value of the cell. The predicted value corresponds to the suggested repair for this cell. Effectively, this model takes into account value co-occurrence across attributes. Similar models have been proposed in the literature to form sets of potential repairs for noisy cells [178].

To ensure high precision, we only accept only repairs with a likelihood more than 90%. In Section 4.9, we evaluate our Naïve Bayes-based model and show that it achieves reasonable precision (i.e., above 70%).

2.5.5 Data Augmentation

To perform data augmentation, we leverage the learned Φ and $\hat{\Pi}$ and use the generative model described in Section 2.5.1. Our approach is outlined in Algorithm 4: First, we sample a correct example with cell value v from the training data T. Second, we sample a transformation φ from distribution $\hat{\Pi}[v]$. If φ can be applied in multiple positions or substrings of input v we choose one uniformly at random, and finally, compute the transformed value $v' = \varphi(v)$. Value v' corresponds to an error as we do not consider the identity transformation. Finally, we add (v, v') in the set of augmented examples with probability α. Probability α is a hyper-parameter of our algorithm, which intuitively corresponds to the required balance in the overall training data. We set α via cross-validation over a holdout-set that corresponds to a subset of T. This is the same holdout-set used to perform Platt scaling during error classification (see Section 6.3.2).

Algorithm 4: Data Augmentation

Input: Training set T; Transformations Φ; Approximate Policy $\hat{\Pi}$; Probability α (hyper-parameter)
Output: Set T_H of augmented examples

$T_H \leftarrow \emptyset$;
$T_c \leftarrow$ set of correct examples in T;
$p \leftarrow$ number of correct examples in T;
$n \leftarrow$ number of erroneous examples in T;
/* we assume that p >> n due to imbalance */ ;
while $|T_H| < p - n$ **do**
    Draw a correct example $v \sim Uniform(T_c)$;
    $C \leftarrow$ Flip a coin with probability α;
    if $C =$ `True` *and* $\hat{\Pi}(v) \neq \emptyset$ **then**
        Draw a transformation $\varphi \sim \hat{\Pi}(v)$;
        $v' \leftarrow \varphi(v)$;
        $T_H \leftarrow T_H \cup \{(v, v')\}$
    end
end

Table 2.1: Datasets used in our experiments.

Dataset	Size	Attributes	Labeled Data	Errors (# of cells)
Hospital	1,000	19	1,000	504
Food	170,945	15	3,000	1,208
Soccer	200,000	10	200,000	31,296
Adult	97,684	11	97,684	1,062
Animal	60,575	14	60,575	8,077

2.6 Experiments

We compare our approach against a wide-variety of error detection methods on diverse datasets. The main points we seek to validate are: (1) is weak supervision the key to high-quality (i.e., high-precision and high-recall) error detection models, (2) what is the impact of different representation contexts on error detection, (3) is data augmentation the right approach to minimizing human exhaust. We also perform extensive micro-benchmark experiments to examine the effectiveness and sensitivity of data augmentation.

2.6.1 Experimental Setup

We describe the dataset, metrics, and settings we use.

Datasets: We use five datasets from a diverse array of domains. Table 6.1 provides information for these datasets. As shown the datasets span different sizes and exhibit various amounts of errors: (1) The Hospital dataset is a benchmark dataset used in several data cleaning papers [37, 178]. Errors are artificially introduced by injecting typos. This is an easy benchmark dataset; (2) The Food dataset contains information on food establishments in Chicago. Errors correspond to conflicting zip codes for the same establishment, conflicting inspection results for the same establishment on the same day, conflicting facility types for the same establishment and many more. Ground truth was obtained by manually labeling 3,000 tuples; (3) The Soccer dataset provides information about soccer players and their teams. The dataset and its ground truth are provided by Rammerlaere and Geerts [167]; (4) Adult contains census data is a typical dataset from the UCI repository. Adult is also provided by Rammerlaere and Geerts [167]; (5) Animal was provided by scientists at UC Berkeley and has been used by Abedjan et al. [2] as a testbed for error detection. It provides information about the capture of animals, including the time and location of the capture and other information for each captured animal. The dataset comes with manually curated ground truth. The datasets used in our experiments exhibit different error distributions. Hospital contains only typos, Soccer [167] and Adult [167] have errors that were introduced with BART [7]: Adult has 70% typos and 30% value swaps, and Soccer has 76%

Table 2.2: Precision, Recall and F_1-score of different methods for different datasets. AL results correspond to $k = 100$.

Dataset (T size)	M	AUG	CV	HC	OD	FBI	LR	SuperL	SemiL	ActiveL
Hospital (10%)	P	0.903	0.030	0.947	0.640	0.008	0.0	0.0	0.0	0.960
	R	0.989	0.372	0.353	0.667	0.001	0.0	0.0	0.0	0.613
	F_1	**0.944**	0.055	0.514	0.653	0.003	0.0	0.0	0.0	0.748
Food (5%)	P	0.972	0.0	0.0	0.240	0.0	0.0	0.985	0.813	0.990
	R	0.939	0.0	0.0	0.99	0.0	0.0	0.95	0.66	0.91
	F_1	**0.955**	0.0	0.0	0.387	0.0	0.0	0.948	0.657	0.948
Soccer (5%)	P	0.922	0.039	0.032	0.999	0.0	0.721	0.802	n/a[#]	0.843
	R	1.0	0.846	0.632	0.051	0.00	0.084	0.450	n/a	0.683
	F_1	**0.959**	0.074	0.061	0.097	0.00	0.152	0.577	n/a	0.755
Adult (5%)	P	0.994	0.497	0.893	0.999	0.990	0.051	0.999	n/a	0.994
	R	0.987	0.998	0.392	0.001	0.254	0.072	0.350	n/a	0.982
	F_1	**0.991**	0.664	0.545	0.002	0.405	0.059	0.519	n/a	0.988
Animal (5%)	P	0.832	0.0	0.0	0.85	0.0	0.185	0.919	n/a	0.832
	R	0.913	0.0	0.0	6×10^{-5}	0.0	0.028	0.231	n/a	0.740
	F_1	**0.871**	0.0	0.0	1×10^{-4}	0.0	0.048	0.369	n/a	0.783

[#] n/a = Semi-supervised learning did not terminate after two days.

typos and 24% swaps. Finally, the two datasets with real-world errors have the following error distributions: Food has 24% typos and 76% value swaps (based on the sampled ground truth); Animal has 51% typos and 49% swaps.

Methods: We compare our approach, referred to as `AUG`, against several competing error detection methods. First, we consider three baseline error detection models:

- **Constraint Violations (CV)**: This method identifies errors by leveraging violations of denial constraints. It is a proxy for rule-based errors detection methods [37].
- **HoloClean (HC)**: This method combines CV with HoloClean [178], a state-of-the-art data repairing engine. This method aims to improve the precision of the CV detector by considering as errors not all cells in tuples that participate in constraint violations but only those cells whose value was repaired (i.e., their initial value is changed to a different value).
- **Outlier Detection (OD)**: This method follows a correlation based outlier detection approach. Given a cell that corresponds to an attribute A_i, the method considers all correlated attributes in $A \setminus A_i$ with A_i rely on the pair-wise conditional distributions to detect if the value of a cell corresponds to an outlier.
- **Forbidden Item Sets (FBI)**: This method captures unlikely value co-occurrences in noisy data [169]. At its core, this method leverages the *lift* measure from association rule min-

ing to identify how probably a value co-occurrence is, and uses this measure to identify erroneous cell values.

- **Logistic Regression (LR)**: This method corresponds to a supervised logistic regression model that classifies cells are erroneous or correct. The features of this model correspond to pairwise co-occurrence statistics of attribute values and constraint violations. This model corresponds to a simple supervised ensemble over the previous two models.

We also consider three variants of our model where we use different training paradigms. The goal is to compare data augmentation against other types of training. For all variations, we use the representation Q and the classifier M introduced in Section 2.3. We consider the following variants:

- **Supervised Learning (SuperL)**: We train our model using only the training examples in T.
- **Semi-supervised Learning (SemiL)**: We train our model using self-training [210]. First supervised learning used to train the model on the labeled data only. The learned model is then applied to the entire dataset to generate more labeled examples as input for a subsequent round of supervised learning. Only labels with high confidence are added at each step.
- **Active Learning (ActiveL)**: We train our model using an active learning method based on uncertainty sampling [184]. First, supervised learning is used to train the model. At each subsequent round, we use an uncertainty-based selection scheme to obtain additional training examples and re-train the model. We use k to denote the number of iterations. In our implementation, we set the upper limit of labeled examples obtained per iteration to be 50 cells.

Evaluation Setup: To measure accuracy, we use Precision (P) defined as the fraction of error predictions that are correct; Recall (R) defined as the fraction of true error being predicted as errors by the different methods; and F_1 defined as $2PR/(P + R)$. For training, we split the available ground truth into three disjoint sets: (1) a training set T, from which 10% is always kept as a hold-out set used for hyper parameter tuning; (2) a sampling set, which is used to obtain additional labels for active learning; and (3) a test set, which is used for evaluation. To evaluate different dataset splits, *we perform* 10 *runs with different random seeds for each experiment*. To ensure that we maintain the coupling amongst Precision, Recall, and F_1, we report the median performance. The mean performance along with standard error measurements are reported in the

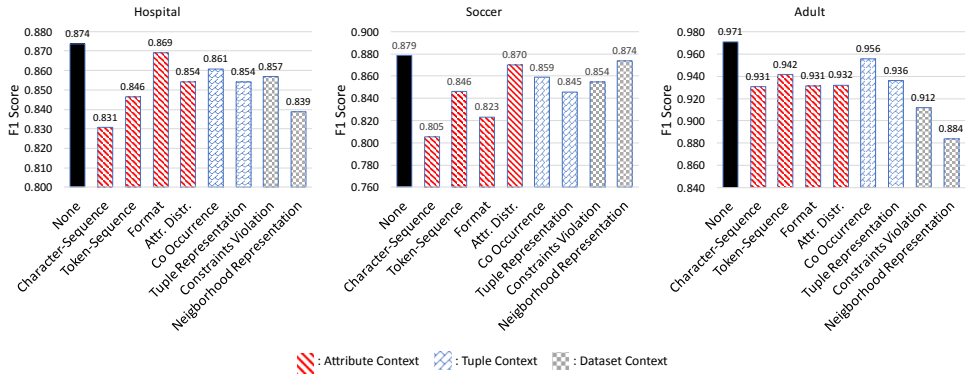


Figure 2.3: Ablation studies to evaluate the effect of different representation models.

Appendix. *Seeds are sampled at the beginning of each experiment, and hence, a different set of random seeds can be used for different experiments.* We use *ADAM* [118] as the optimization algorithm for all learning-based model and train all models for 500 epochs with a batch-size of five examples. We run Platt Scaling for 100 epochs. All experiments were executed on a 12-core Intel(R) Xeon(R) CPU E5-2603 v3 @ 1.60GHz with 64GB of RAM running Ubuntu 14.04.3 LTS.

2.6.2 End-to-end Performance

We evaluate the performance of our approach and competing approaches on detecting errors in all five datasets. Table 6.3 summarizes the precision, recall, and F_1-score obtained by different methods. For Food, Soccer, Adult, and Animal we set the amount of training data to be 5% of the total dataset. For Hospital we set the percentage of training data to be 10% (corresponding to 100 tuples) since Hospital is small. For Active Learning we set the number of active learning loops to $k = 100$ to maximize performance.

As Table 6.3 shows, our method consistently outperforms all methods, and in some cases, like Hospital and Soccer, we see improvements of 20 F_1 points. More importantly, we find that our method is able to achieve both high recall and high precision in all datasets despite the different error distribution in each dataset. This is something that has been particularly challenging for prior error detection methods. We see that for Food and Animal, despite the fact that most errors do not correspond to constraint violations (as implied by the performance of CV), AUG can obtain high precision and recall. This is because AUG models the actual data distribution and not the side-effects of errors. For instance, for Food we see that OD can detect many of the errors—it has high recall—indicating that most errors correspond to statistical outliers. We see

that AUG can successfully solve error detection for this dataset. Overall, our method achieves an average precision of 92% and an average recall of 96% across these diverse datasets. At the same time, we see that the performance of competing methods varies significantly across datasets. This validates the findings of prior work [2] that depending on the side effects of errors different error detection methods are more suitable for different datasets.

We now discuss the performance of individual competing methods. For CV, we see that it achieves higher recall than precision. This performance is due to the fact that CV marks as erroneous all cells in a group of cells that participate in a violation. More emphasis should be put on the recall-related results of CV. As shown its recall varies dramatically from 0.0 for Food and Animal to 0.998 for Adult. For OD, we see that it achieves relatively high-precision results, but its recall is low. Similar performance is exhibited by FBI that leverages a different measure for outlier detection. We see that FBI achieves high precision when the forbidden item sets have significant support (i.e., occur relatively often). However, FBI cannot detect errors that lead to outlier values which occur a limited number of times. This is why we find OD to outperform FBI in several cases.

Using HC as a detection tool is limited to these cells violating integrity constraints. Hence, using HC leads to improved precision over CV (see Hospital and Adult). This result is expected as data repairing limits the number of cells detected as erroneous to only those whose values are altered. Our results also validate the fact that HC depends heavily on the quality of the error detection used [178]. As shown in Food and Animal, the performance of HC is limited by the recall of CV, i.e., since CV did not detect errors accurately, HC does not have the necessary training data to learn how to repair cells. At the same time, Soccer reveals that training HC on few clean cells—the recall of CV is very high while the precision is very low indicating that most cells were marked as erroneous—leads to low precision (HC achieves a precision of 0.032 for Soccer). This validates our approach of solving error detection separately from data repairing.

We also see that LR has consistently poor performance. This result reveals that combining co-occurrence features and violations features in a linear way (i.e., via a weighted linear combination such as in LR) is not enough to capture the complex statistics of the dataset. *This validates our choice of using representation learning and not engineered features*.

Finally, we see that approaches that rely on representation learning model achieve consistently high precision across all datasets. This validates our hypothesis that modeling the distribution of both correct and erroneous data allows us to discriminate better. However, we see that when we rely only on the training dataset T the recall is limited (see the recall for SuperL). The limited labeled examples in T is not sufficient to capture the heterogeneity of errors. Given additional training examples either via Active Learning or via Data Augmentation helps improve the recall. However, Data Augmentation is more effective than Active Learning at capturing the

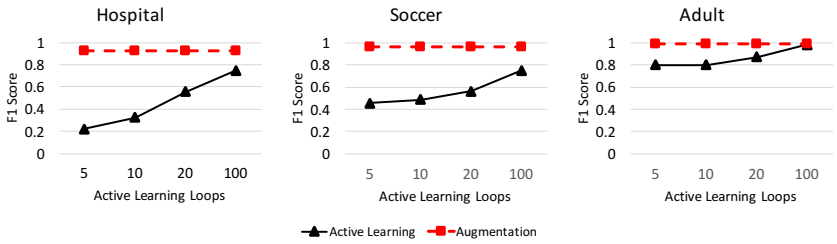


Figure 2.4: Data augmentation versus active learning as the number of active learning loops increases.

heterogeneity of errors in each dataset, and hence, achieves superior recall to Active Learning in all cases.

Takeaway: The combination of representation learning techniques with data augmentation is key to obtaining high-quality error detection models.

2.6.3 Representation Ablation Study

We perform an ablation study to evaluate the effect of different representation models on the quality of our model. Specifically, we compare the performance of AUG when all representation models are used in Q versus variants of AUG where one model is removed at a time. We report the F_1-score of the different variants as well as the original AUG in Figure 2.3. Representation models that correspond to different contexts are grouped together.

Removing any feature has an impact on the quality of predictions of our model. We find that removing a single representation model results in drops of up to 9 F_1 points across datasets. More importantly, we find that different representation models have different impact on different datasets. For instance, the biggest drop for Hospital and Soccer is achieved when the character-sequence model is removed while for Adult the highest drop is achieved when the Neighborhood representation is removed. This validates our design of considering representation models from different contexts. **Takeaway:** It is necessary to leverage cell representations that are informed by different contexts to provide robust and high-quality error detection solutions.

2.6.4 Augmentation versus Active Learning

We validate the hypothesis that data augmentation is more effective than active learning in minimizing human effort in training error detection models. In Table 6.3, we showed that data

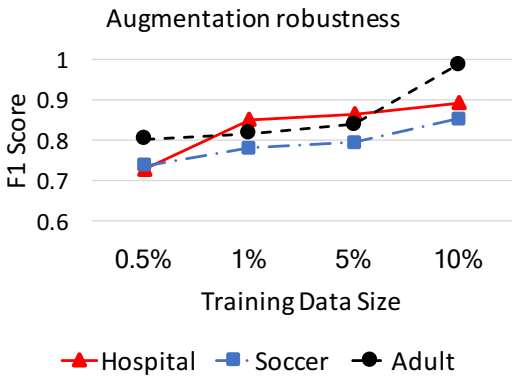


Figure 2.5: Data augmentation performance for various amounts of training data.

augmentation outperforms active learning. Furthermore, active learning needs to obtain more labeled examples to achieve comparable performance to data augmentation. In the next two experiments, we examine the performance of the two approach as we limit their access to training data.

In the first experiment, we evaluate active learning for different values of loops (k) over Hospital, Soccer, and Adult. We vary k in $\{5, 10, 20, 100\}$. We fix the amount of available training data to 5%. Each time we measure the F_1 score of the two algorithms. We report our results in Figure 2.4. Reported results correspond to median performance over ten runs.

We see that when a small number of loops is used (k=5), there is a significant gap between the two algorithms that ranges between 10 and 70 F_1 points. Active learning achieves comparable performance with data augmentation only after 100 loops. This corresponds to an additional $5,000$ ($k \times 50$) labeled examples (labeled cells). This behavior is consistent across all three datasets. In the second experiment, we seek to push data augmentation to the limits. Specifically, we seek to answer the question, can data augmentation be effective when the number of labeled examples in T is extremely small. To this end, we evaluate the performance of our system on Hospital, Soccer, and Adult as we vary the size of the training data in $\{0.5\%, 1\%, 5\%, 10\%\}$. The results are shown in Figure 2.5. As expected the performance of data augmentation is improving as more training data become available. However, we see that data augmentation can achieve good performance—F_1 score does not drop below 70%—even in cases where labeled examples T are limited. These results provide positive evidence that data augmentation is a viable approach for minimizing user exhaust.

Takeaway: Our data augmentation approach is preferable to active learning for minimizing human exhaust.

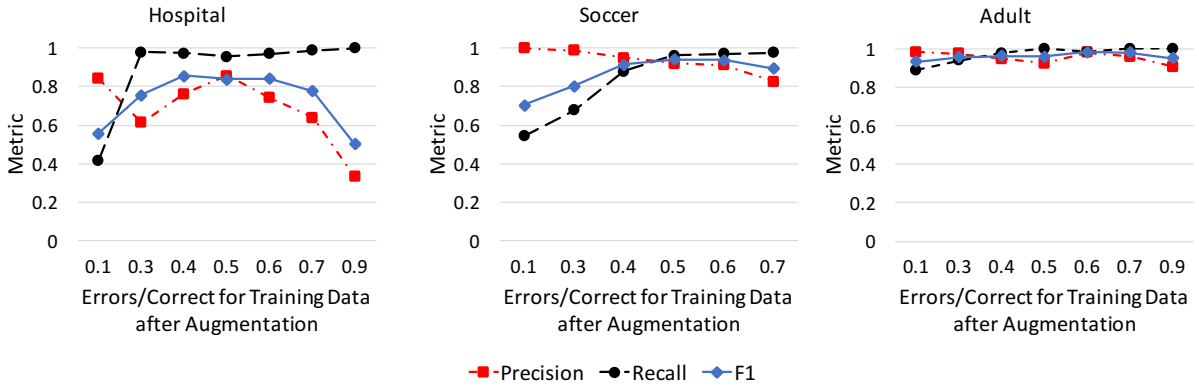


Figure 2.6: The effect of increasing the number of examples that correspond to errors via data augmentation.

2.6.5 Augmentation and Data Imbalance

We evaluate the effectiveness of data augmentation to counteract imbalance. Table 6.3 shows that using data augmentation yields high-quality error detection models for datasets with varying percentages of errors. Hence, data augmentation is robust to different levels of imbalance; each dataset in Table 6.3 has a different ratio of true errors to correct cells.

In Table 2.3, we compare data augmentation with traditional methods used to solve the imbalance problem, namely, resampling. In all the datasets, resampling had low precision and recall confirming our hypothesis discussed before: due to the heterogeneity of the errors, resampling from the limited number of negative examples was not enough to cover all types of errors. The best result for resampling was obtained in the Hospital data set (F_1 about 47%), since errors are more homogeneous than other data sets.

We also evaluate the effect of excessive data augmentation: In Algorithm 4 we do not use hyper-parameter α to control how many artificial examples should be generated via data augmentation. We manually set the ratio between positive and negative examples in the final training examples and use augmentation to materialize this ratio.

Our results are reported in Figure 2.6. We show that increasing the number of generated negative examples (errors) results in a lower accuracy as the balance between errors and correct example goes greater than 50%, as the model suffers from the imbalance problem again, this time as too few correct examples. We see that peak performance is achieved when the training data is almost balanced for all datasets. This reveals the robustness of our approach. Nonetheless, peak performance is not achieved exactly at a 50-50 balance (peak performance for Adult is at

Table 2.3: A comparison between data augmentation and resampling. We report the F_1-score as we increase the size of the training data T. We also include supervised learning as a baseline.

Dataset	Size of T	AUG	Resampling	SuperL
Hospital	1%	**0.840**	0.041	0.0
	5%	**0.873**	0.278	0.0
	10%	**0.925**	0.476	0.079
Soccer	1%	**0.927**	0.125	0.577
	5%	**0.935**	0.208	0.654
	10%	**0.953**	0.361	0.675
Adult	1%	**0.844**	0.063	0.0
	5%	**0.953**	0.068	0.294
	10%	**0.975**	0.132	0.519

60%). This justifies our model for data augmentation presented in Algorithm 4 and the use of hyper-parameter α.

Takeaway: Data augmentation is an effective way to counteract imbalance in error detection.

2.6.6 Analysis of Augmentation Learning

In this experiment, we validate the importance of learning the augmentation model (the transformations Φ, and the policy $\hat{\Pi}$). We compare three augmentation strategies: (1) Random transformations *Rand. Trans.*, where we randomly choose from a set of errors (e.g., typos, attribute value changes, attribute shifts, etc.). Here, we augment the data by using completely random transformations not inspired by the erroneous examples or the data; and (2) learned transformation Φ, but without learning the distribution policy(*Aug w/o Policy*). Given an input, we find all valid transformations in Φ and pick one uniformly at random. Table 2.4 shows the results for the three approaches. AUG outperforms the other two strategies. Rand. Trans. fails to capture the errors that exist in the dataset. For instance, it obtains a recall of 16.6% for Soccer. Even though the transformations are learned from the data, it is the results show that using these transformations in a way that conform with the distribution of the data is crucial in learning an accurate classifier.

Takeaway: Learning a noisy channel model from the data, i.e., a set of transformations Φ and a policy Π is key to obtaining high-quality predictions.

Table 2.4: A comparison between different data augmentation approaches. We report the F_1-score as we increase the size of the training data T.

Dataset	T	AUG	Rand. Trans.	AUG w/o Policy
Hospital	5%	**0.911**	0.873	0.866
	10%	**0.943**	0.884	0.870
Soccer	5%	**0.946**	0.212	0.517
	10%	**0.953**	0.166	0.522
Adult	5%	**0.977**	0.789	0.754
	10%	**0.984**	0.817	0.747

Table 2.5: Runtimes in seconds. Value n/a means that the method did not terminate after running for two days.

Approach	**Hospital**	**Soccer**	**Adult**
AUG	749.17	7684.72	6332.13
CV	204.62	1610.02	1359.46
OD	212.7	1588.06	1423.69
LR	347.95	3505.60	4408.27
SuperL	648.34	3928.46	3310.71
SemiL	14985.15	n/a	n/a
ActiveL	3836.15	56535.19	128132.56

2.6.7 Other Experiments

Finally, we report several benchmarking results: (1) we measure the runtime of different methods, (2) validate the performance of our unsupervised Naïve Bayes model for generating labeled example to learn transformations Φ and Π (see Section 6.4), and (3) validate the robustness of AUG to misspecified denial constraints.

The median runtime of different methods is reported in Table 2.5. These runtimes correspond to prototype implementations of the different methods in Python. Also recall, that training corresponds to 500 epochs with low batch-size as reported in Section 2.6.1. As expected iterative methods such as SemiL and ActiveL are significantly slower than non-iterative ones. Overall, we see that AUG exhibits runtimes that are of the same order of magnitude as supervised methods.

The performance of our Naïve Bayes-based weak supervision method on Hospital, Soccer, and Adult is reported in Table 2.6. Specifically, we seek to validate that the precision of our weak supervision method is reasonable, and thus, by using it we obtain good examples that correspond to good examples from the true error distribution. We see that our weak supervision method

Table 2.6: Performance of our weak supervision method for generating training examples for AUG.

Dataset	Precision	Recall
Hospital	0.895	0.636
Soccer	0.999	0.053
Adult	0.714	0.973

achieves a precision of more than 70% in all cases. As expected its recall can be some times low (e.g., for Soccer it is 5.3%) as emphasis is put on precision.

Finally, we evaluate AUG against missing and noisy constraints. The detailed results are presented in Appendix 2.8.2 due to space restrictions. In summary, we find AUG to exhibit a drop of at most 6 F_1 points when only 20% of the original constraints are used to missing constraints and at most 8 F_1 points when noisy constraints are used.

2.7 Related Work

Many algorithms and prototypes have been proposed for developing data cleaning tools [56, 68, 107, 166]. Outlier detection and quantitative data cleaning algorithms are after data values that looks "abnormal" with respect to the data distribution [47, 165, 201]. Entity resolution and record de-duplication focus on identifying clusters of records that represent the same real-world entity [66, 149]. Example de-duplication tools include the `Data Tamer` system [190], which is commercialized as `Tamr`. Rule-based detection proposals [1, 37, 70, 121, 198] use integrity constraints (e.g., denial constraints) to identify violations, and use the overlap among these violations to detect data errors. Prototypes such as such as `Nadeef` [46], and `BigDansing` [115] are example extensible rule-based cleaning systems. There have been also multiple proposals that identify data cells that don not follow a data "pattern". Example tools include `OpenRefine`, `Data Wrangler` [113] and its commercial descendant `Trifacta`, `Katara` [41], and `DataXFormer` [3]. An overview of these tools and how they can be combined for error detection is discussed in [2], where the authors show that even when all are used, these tools often achieve low recall in capturing data errors in real data sets.

Data Augmentation has also been used extensively in machine learning problems. Most state-of-the-art image classification pipelines use some limited for of data augmentation [162]. This consists of applying crops, flips, or small affine transformations in fixed order or at random. Other studies have applied heuristic data augmentation to modalities such as audio [147] and text [142]. To our knowledge, we are the first to apply data augmentation in relational data.

Recently, several lines of work have explored the use of reinforcement learning or random search to learn more principled data augmentation policies [45, 174]. Our work here is different as we do not rely on expensive procedures to learn the augmentation policies. This is because we limit our policies to applying a single transformation at a time. Finally, recent work has explored techniques based on Generative Adversarial Networks [85] to learn data generation models used for data augmentation from unlabeled data [146]. This work focuses mostly on image data. Exploring this direction for relational data is an exciting future direction.

2.8 Background Detail

We provide additional details for the representation models in our framework and present additional micro-benchmark experimental results on the robustness of our error detection approach to noisy denial constraints.

2.8.1 Details on Representation Models

Our model follows the wide and deep architecture of Cheng et al. [33]. Thus the model can be thought of as a representation stage, where each feature is being operated on in isolation, and an inference step in which each feature has been concatenated to make a joint representation. The joint representation is then fed through a two-layer neural network. At training time, we backpropogate through the entire network jointly, rather than training specific representations. Figure 2.7 illustrates this model's topology.

A summary of representation models used in our approach along with their dimensions is provided in Table 2.7. As shown we use a variety of models that capture all three attribute-level, tuple-level, and dataset-level contexts. We next discuss the embedding-based models and format models we use.

Table 2.7: A summary of representation models used in our approach along with their dimension.

Context	Representation Type	Description	Dimension
Attribute-Level	Character Embedding	FastText Embedding where tokens are characters	1
	Word Embedding	FastText Embedding where tokens are words in the cell	1
	Format models	3-Gram: Frequency of the least frequent 3-gram in the cell	1
	Format models	Symbolic 3-Gram; each character is replaced by a token $\{Char, Num, Sym\}$	1
	Empirical distribution model	Frequency of cell value	1
	Empirical distribution model	One Hot Column ID; Captures per-column bias	1
Tuple-Level	Co-occurrence model	Co-occurrence statistics for a cell's value	#attributes -1
	Tuple representation	FasText-based embedding of the union of tokens after tokenizing each attribute value	1
Dataset-Level	Constraint violations	Number of violations per denial constraint	#constraints
	Neighborhood representation	Distance to top-1 similar word using a FastText tuple embedding over the non-tokenized attribute values	1

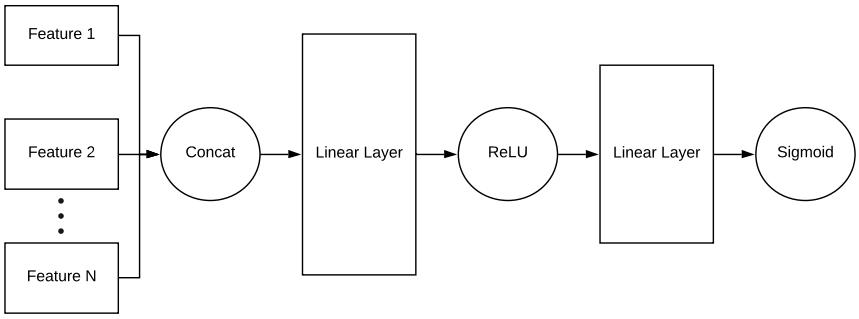


Figure 2.7: The architecture of our representation learning model following a wide and deep architecture.

Embedding-based Models: We treat different views of the data as expressing different language models, and so embed each to capture their semantics. The embeddings are taken at a character, cell and tuple level tokens, and each uses a FastText Embedding in 50 dimensions [18, 112]. Rather than doing inference directly on the embeddings, we employ a two-step process of a non-linear transformation and dimensionality reduction. At the non-linear transformation stage, we use a two-layer Highway Network [189] to extract useful representations of the data. Then, a dense layer is used to reduce the dimensionality to a single dimension. In this way, the embeddings do not dominate the joint representation. Figure 2.2(B) shows this module more explicitly.

In addition to using these singular embeddings, we also use a distance metric on the learned corpus as a signal to be fed into the model (see Neighborhood representation). The intuition behind this representation is that in the presence of other signals that would imply a cell is erroneous, there may be some similar cell in the dataset with the correct value; hence, the distance to it will be low. For this, we simply take the minimum distance to another embedding in our corpus, and this distance is fed to the joint representation.

Forma Models (3-Grams): We follow a similar approach to that of Huang and He [105]. This work introduces custom language models to do outlier detection. We follow a simplified variation of this approach and use two fixed length language models. They correspond to the 3-Gram models shown in Table 2.7. To build these representation models, we build a distribution of 3-Grams present in each column, this is done using the empirical distribution of the data and Laplace smoothing. For 3-Gram, the distribution is based on all possible ASCII 3-Grams. The difference in the symbol based variation of 3-Gram is that the distribution is based off the alphabet $\{Charcater, Number, Symbol\}$. The value returned for each model is the least frequency of all 3-grams present in the cell value.

2.8.2 Effect of Misspecified Constraints

We conduct a series of micro-benchmark experiments to evaluate the robustness of AUG against misspecified denial constraints. First, we evaluate AUG's performance as only a subset of constraints is given as input, and second, we evaluate AUG's performance as constraints become noisy.

Limiting the number of Constraints

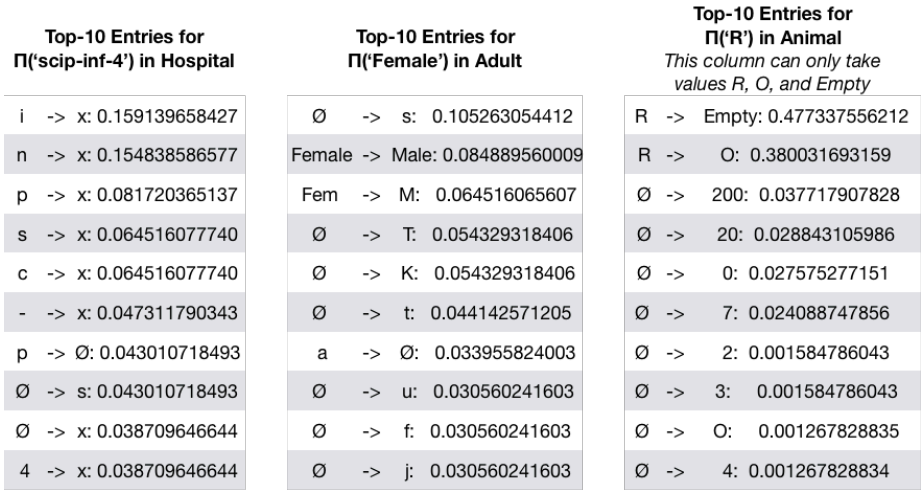
We consider Hospital, Adult, and Soccer with the denial constraints used for our experiments in Section 4.9 and perform the following experiment: For each dataset, we define a vary the number of constraints given as input to AUG by taking only a proportion ρ of the initial constraints. We vary ρ in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ where 0.2 indicates that a random subset of 20% of the constraints is used while 1.0 indicates that all constraints are used. For each configuration for ρ we obtain 21 samples of the constraints and evaluate AUG for these random subsets. We report the median F_1, precision, and recall in Table 2.9. As shown, AUGs performance gradually decreases as the number of denial constraints is reduced and converges to the performance reported in the study in Section 2.6.3 when no constraints are used in AUG. The results in Table 2.9 also show that AUG is robust to small variations in the number of constraints provided as input. We see that when $\rho > 0.4$ the F_1 score of AUG does not reduce more than two points.

Table 2.9: Median performance of AUG over 21 runs as we randomly limit the input constraints to $\rho \times |\text{initial constraints}|$.

Dataset	M	$\rho = 0.2$	$\rho = 0.4$	$\rho = 0.6$	$\rho = 0.8$	$\rho = 1$
Hospital	P	0.857	0.829	0.927	0.925	0.936
	R	0.848	0.877	0.857	0.896	0.901
	F_1	0.852	0.852	0.891	0.910	0.918
Adult	P	0.860	0.890	0.897	0.917	0.934
	R	0.994	0.992	0.999	0.999	0.999
	F_1	0.922	0.938	0.945	0.956	0.965
Soccer	P	0.836	0.855	0.864	0.860	0.863
	R	0.868	0.879	0.872	0.887	0.894
	F_1	0.852	0.867	0.868	0.873	0.878

Noisy Denial Constraints

We now turn our attention to noisy constraints. We use the following definition of noisy constraints:



Top-10 Entries for Π('scip-inf-4') in Hospital

i	->	x: 0.159139658427
n	->	x: 0.154838586577
p	->	x: 0.081720365137
s	->	x: 0.064516077740
c	->	x: 0.064516077740
-	->	x: 0.047311790343
p	->	Ø: 0.043010718493
Ø	->	s: 0.043010718493
Ø	->	x: 0.038709646644
4	->	x: 0.038709646644

Top-10 Entries for Π('Female') in Adult

Ø	->	s: 0.105263054412
Female	->	Male: 0.084889560009
Fem	->	M: 0.064516065607
Ø	->	T: 0.054329318406
Ø	->	K: 0.054329318406
Ø	->	t: 0.044142571205
a	->	Ø: 0.033955824003
Ø	->	u: 0.030560241603
Ø	->	f: 0.030560241603
Ø	->	j: 0.030560241603

Top-10 Entries for Π('R') in Animal
This column can only take values R, O, and Empty

R	->	Empty: 0.477337556212
R	->	O: 0.380031693159
Ø	->	200: 0.037717907828
Ø	->	20: 0.028843105986
Ø	->	0: 0.027575277151
Ø	->	7: 0.024088747856
Ø	->	2: 0.001584786043
Ø	->	3: 0.001584786043
Ø	->	O: 0.001267828835
Ø	->	4: 0.001267828834

Figure 2.8: Examples of learned augmentation policies for clean entries in Hospital and Adult.

Definition 2.8.1. *The denial constraint dc is α-noisy on the dataset D if it satisfies α percent of all tuple pairs in D.*

We want to see the effect of noisy denial constraints on the performance of AUG. We use the following strategy to identify noisy denial constraints for each dataset: We use the denial constraint discovery method of Chu et al. [39] and group the discovered constraints in four ranges with respect to the noise level α. Constraints with $\alpha \in (0.55, 0.65]$, constraints with $\alpha \in (0.65, 0.75]$, constraints with $\alpha \in (0.75, 0.85]$, and constraints with $\alpha \in (0.85, 0.95]$. For each range, we obtain 21 constraint-set samples, such that each sampled constraint set has the same cardinality as the original clean constraints associated with each of the Hospital, Adult, and Soccer datasets. We report the median performance of AUG in Table 2.10. As shown, the impact of noisy denial constraints on AUG's performance is not significant. The reason is that during training AUG can identify that the representation associated with denial constraints corresponds to a noisy feature and thus reduce its weight in the final classifier.

2.8.3 Learned Augmentation Policies

We provide examples of learned policies for clean entries in Hospital, Adult, and Animal. For Hospital and Adult, we know how errors were introduced, and hence, can evaluate the performance of our methods for learning augmentation policies. Errors in Hospital correspond to typos introduced artificially by swapping a character in the clean cell values with the character 'x'. On

Table 2.10: Median performance of AUG over 21 runs with noisy constraints that correspond to different noise levels α.

Dataset	M	$\alpha \in (0.55, 0.65]$	$(0.65, 0.75]$	$(0.75, 0.85]$	$(0.85, 0.95]$
Hospital	P	0.859	0.876	0.912	0.925
	R	0.822	0.869	0.899	0.914
	F_1	0.840	0.873	0.906	0.920
Adult	P	0.911	0.949	0.961	0.984
	R	0.875	0.930	0.952	0.961
	F_1	0.893	0.939	0.956	0.972
Soccer	P	0.821	0.849	0.867	0.863
	R	0.864	0.862	0.880	0.891
	F_1	0.842	0.855	0.873	0.877

the other hand, errors in the gender attribute of Adult are introduced either by swapping the two gender values 'Female' and 'Male' or by introducing typos via injection of characters. For Animal, we do not know how errors are introduced. However, we focus on an attribute that can only take values in {R, O, Empty} to evaluate the performance of our methods.

Figure 2.8 depicts the top-10 entries in the conditional distribution corresponding to entry 'scip-inf-4' for Hospital and entry 'Female' for Adult. As shown, for Hospital, almost all transformations learned by our method correspond to either swapping a character a character with the character 'x' or injecting 'x' in the original string. The performance of our approach is similar for Adult. We observe that a mix of value swaps, e.g., 'Female' $\longmapsto$ 'Male', and character injection transformations are learned. Finally, for Animal, we see that most of the mass of the conditional distribution (almost 86%) is concentrated in the value swap transformations 'R' $\longmapsto$ 'Empty' and 'R' $\longmapsto$ 'O' while all other transformations have negligible probabilities. These results demonstrate that our methods can effectively learn how errors are introduced and distributed in noisy relational datasets.

Chapter 3

Approximate inference in structured instances with noisy categorical observations

Statistical inference over structured instances of dependent variables (e.g., labeled sequences, trees, or general graphs) is a fundamental problem in many areas. Examples include computer vision [31, 57, 154], natural language processing [103, 104], and computational biology [131]. In many practical setups [96, 179, 182, 186], inference problems involve noisy observations of discrete labels assigned to the nodes and edges of a given structured instance and the goal is to infer a labeling of the vertices that achieves low disagreement rate between the correct ground truth labels Y and the predicted labels $\hat{Y}$, i.e., low *Hamming error*. We refer to this problem as *statistical recovery*.

Our motivation to study the problem of statistical recovery stems from our recent work on data cleaning [96, 179, 182]. This work introduces HoloClean, a state-of-the-art inference engine for data curation that casts data cleaning as a structured prediction problem [182]: Given a dataset as input, it associates each of its cells with a random variable, and uses logical integrity constraints over this dataset (e.g., key constraints or functional dependencies) to introduce dependencies over these random variables. The labels that each random variable can take are determined by the domain of the attribute associated with the corresponding cell. Since we focus on data cleaning, the input dataset corresponds to a noisy version of the latent, clean dataset. Our goal is to recover the latter. Hence, the initial value of each cell corresponds to a noisy observation of our target random variables. HoloClean employs approximate inference methods to solve this structured prediction problem. While its inference procedure comes with no rigorous

guarantees, HoloClean achieves state-of-the-art results in practice. Our goal in this work is to understand this phenomenon.

Recent works have also studied the problem of approximate inference in the presence of noisy vertex and edge observations. However, they are limited to the case of binary labeled variables: Globerson et al. focused on two-dimensional grid graphs and show that a polynomial time algorithm based on MaxCut can achieve optimal Hamming error for planar graphs for which a weak expansion property holds [83]. More recently, Foster et al. introduced an approximate inference algorithm based on tree decompositions that achieves low expected Hamming error for general graphs with bounded tree-width [77]. In this work, we generalize these results to the case of categorical labels.

Problem and Challenges We study the problem of statistical recovery over categorical data. We consider structured instances where each variable u takes a ground truth label Y_u in the discrete set $\{1, 2, \ldots, k\}$. We assume that for all variables u, we observe a noisy version Z_u of its ground truth labeling such that $Z_u = Y_u$ with probability $1 - q$. We also assume that for all variable pairs (u, v), we observe noisy measurements $X_{u,v}$ of the indicator $M_{u,v} = 2 \cdot \mathbb{1}(Y_u = Y_v) - 1$ such that $X_{u,v} = M_{u,v}$ with probability $1 - p$. Given these noisy measurements, our goal is to obtain a labeling $\hat{Y}$ of the variables such that the expected Hamming error between Y and $\hat{Y}$ is minimized. We now provide some intuition on the challenges that categorical variables pose and why current approximate inference methods not applicable:

First, in contrast to the binary case, negative edge measurements do not carry the same amount of information: Consider a simple uniform noise model. In the case of binary labels, observing an edge measurement $X_{u,v} = -1$ and a binary label Z_u allows us to estimate that $\hat{Y}_v = -Z_u$ is correct with probability $(1 - q)(1 - p) + qp$ when p and q are bounded away from 1/2. However, in the categorical setup, $\hat{Y}_v$ can take any of the $\{1, 2, \ldots, k\} \setminus \{Z_u\}$ labels, hence the probability of estimate $\hat{Y}_v$ being correct is up to a factor of $\frac{1}{k}$ smaller than the binary case. Our main insight is that while the binary case leverages edge labels for inference, approximate inference methods for categorical instances need to rely on the noisy node measurements and the positive edge measurements.

Second, existing approximate inference methods for statistical recovery [77, 83] rely on a "Flipping Argument" that is limited to binary variables to obtain low Hamming error: for binary node and edge observations, if all nodes in a maximal connected subgraph S are labeled incorrectly with respect to the ground truth, then at least half of the edge observations on the boundary of S are incorrect, or else the inference method would have flipped all node labels in S to obtain a better solution with respect to the total Hamming error. As we discuss later, in the categorical

case a naive extension implies that one needs to reason about all possible label permutations over the k labels.

Contributions We present a new approximate inference algorithm for statistical recovery with categorical variables. Our approach is inspired by that of [77] but generalizes it to categorical variables.

First, we show that, when a variable u is assigned one of the $k-1$ erroneous labels with uniform probability $q/(k-1)$, the optimal Hamming error for trees with n nodes is $\tilde{O}(\log(k) \cdot p \cdot n)$, when $q < 1/2$. This is obtained by solving a linear program using dynamic programming. Here, we derive a tight upper bound on the number of erroneous edge measurements, which we use to restrict the space of solutions explored by the linear program.

Second, we extend our method to general graphs using a tree decomposition of the structured input. We show how to combine our tree-based algorithm with correlation clustering over a fixed number of clusters [80] to obtain a non-trivial error rate for graphs with bounded treewidth and a specified number of k classes. Our method achieves an expected Hamming error of $\tilde{O}\big(k \cdot \log(k) \cdot p^{\lceil \frac{\Delta(G)}{2} \rceil} \cdot n\big)$ where $\Delta(G)$ is the maximum degree of graph G. We show that local pairwise label swaps are enough to obtain a globally consistent labeling with low expected Hamming error.

Finally, we validate our theoretical bounds via experiments on tree graphs and image data. Our empirical study demonstrates that our approximate inference algorithm achieve low Hamming error in practical scenarios.

3.1 Preliminaries

We introduce the problem of statistical recovery, and describe concepts, definitions, and notation used in the chapter. We consider a structured instance represented by a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. Each vertex $u \in V$ represents a random variable with ground truth label Y_u in the discrete set $L = \{1, 2, \ldots, k\}$. Edges in E represent dependencies between random variables and each edge $(u, v) \in E$ has a ground truth measurement $M_{u,v} = \varphi(Y_u, Y_v)$ where $\varphi(Y_u, Y_v) = 1$ if $\mathbb{1}(Y_u = Y_v) = 1$ and $\varphi(Y_u, Y_v) = -1$ otherwise.

Uniform Noise Model and Hamming Error We assume access to noisy observations over the nodes and edges of G. For each variable $u \in V$, we are given a noisy label observation Z_u, and for each edge $(u, v) \in E$ we are given a noisy edge observation $X_{u,v}$. These noisy observations are assumed to be generated from G, Y and M by the following process: We are

given $G = (V, E)$ and two parameters, edge noise p and node noise $q < 1/2$ with $p < q$. For each edge $(u, v) \in E$, the observation $X_{u,v}$ is independently sampled to be $X_{u,v} = M_{u,v}$ with probability $1 - p$ (a *good* edge) and $X_{u,v} = -M_{u,v}$ with probability p (a *bad* edge). For each node $u \in V$, the node observation Z_u is independently sampled to be $Z_u = Y_u$ with probability $1 - q$ (a *good* node) and can take any other label in $L \setminus Y_u$ with a uniform probability $\frac{q}{k-1}$. The uniform noise model is a direct extension of that considered by prior work [77, 83], and a first natural step towards studying statistical recovery for categorical variables.

Given the noisy measurements X and Z over graph $G = (V, E)$, a labeling algorithm is a function A: $\{-1, +1\}^E \times \{1, 2, \ldots, k\}^V \to \{1, 2, \ldots, k\}^V$. We follow the setup of [83] to measure the performance of A. We consider the expectation of the Hamming error (i.e., the number of mispredicted labels) over the observation distribution induced by Y. We consider as error the worst-case (over the draw of Y) expected Hamming error, where the expectation is taken over the process generating the observations X from Y. Our goal is to find an algorithm A such that with high probability it yields bounded worst-case expected Hamming error. In the remainder of the chapter, we will refer to the worst-case expected Hamming error as simply Hamming error.

Categorical Labels and Edge Measurements When q is close to 0.5, one needs to leverage the edge measurements to predict the node labels correctly. For binary labels, the structure of the graph G alone determines if one can obtain algorithms with a small error for low constant edge noise p [77, 83]. We argue that this is not the case for categorical labels. Beyond the structure of the graph G, the number of labels k also determines when we can obtain labeling algorithms with non-trivial error bounds.

We use the next example to provide some intuition on how k affects the amount of information in the edge measurements of G: Let nodes take labels in $L = \{1, 2, \ldots, k\}$. We fix a vertex v, and for each vertex u in its neighborhood set the estimate label $\hat{Y}_u$ to Z_u if $M_{u,v} = 1$ and to one of $L \setminus \{Z_u\}$ uniformly at random if $M_{u,v} = -1$. For a correct negative edge measurement and a correct label assignment to v, we are not guaranteed to obtain the correct label for v as we would be able in the binary case.

Given the above setup, the probability that node u is labeled correctly is $P(\hat{Y}_u = Y_u) = (1 - b(1 - \frac{1}{k-1})) \cdot ((1 - p)(1 - q) + pq))$ where b is the probability of an edge being negative in the ground truth labeling of G. Two observations emerge from this expression: (1) As the number of colors k increases, the probability $P(\hat{Y}_u = Y_u)$ decreases, hence, for a fixed graph G as k increases, statistical recovery becomes harder; (2) For a fixed graph G, as k increases the probability b of obtaining a negative edge in the ground truth labeling of G increases— this holds for a fixed graph G and under the assumption that each label should appear at least once in the

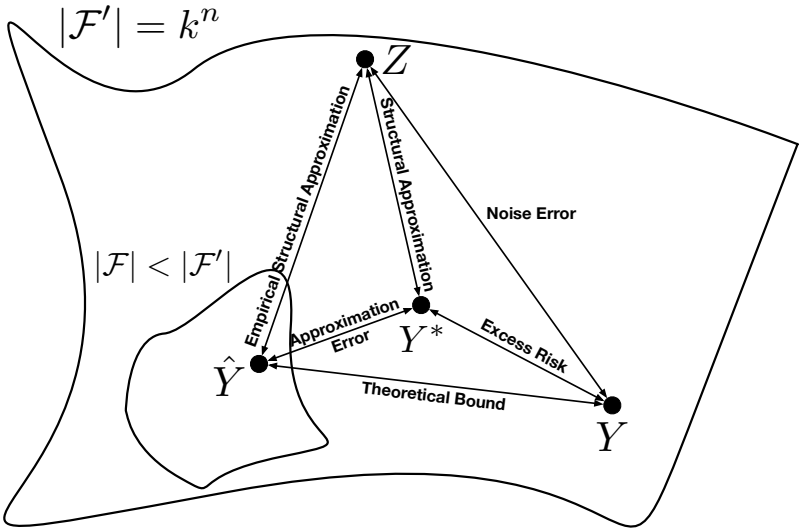


Figure 3.1: A schematic overview of our approach. Given the noise node labeling Z of a graph with ground truth labeling Y, we leverage the noisy side information to obtain an approximate labeling $\hat{Y}$. Labeling $\hat{Y}$ is an approximate solution to the information theoretic optimal solution Y^*. The goal of our analysis is to find a theoretical bound on the Hamming error between $\hat{Y}$ and Y.

ground truth—and the term $(1 - b(1 - \frac{1}{k-1}))$ approaches zero. This implies that for $P(\hat{Y}_u = Y_u)$ to be meaningful the term $((1-p)(1-q) + pq)$ should be maximized for fixed q, and hence, the edge noise p should approach zero as a function of $(1 - b(1 - \frac{1}{k-1}))$. In other words, p should be upper bounded by a function $\varphi(k)$ such that as k increases $\varphi(k)$ goes to zero. We leverage these two observations to specify when statistical recovery is possible.

Statistical Recovery Statistical recovery is possible for the family $\mathcal{G}$ of structured instances with k categories, if there exists a function $f(p, k) : [0, 1] \to [0, 1]$ with $\lim_{p\to 0} f(p, k) = 0$ such that for every p that is upper bounded by a function $\varphi(k)$ with $\lim_{k\to V} \varphi(k) = 0$, the Hamming error of a labeling algorithm on graph $G \in \mathcal{G}$ with $V = n$ vertices is at most $f(p, k) \cdot n$.

3.2 Overview

We consider a graph $G = (V, E)$ with node labels in $L = \{1, 2, \ldots, k\}$. The space of all possible labelings of V defines a *hypothesis space* $\mathcal{F}'$. In this space, we denote Y the latent, ground truth

labeling of G. In the absence of any information the size of this space is $|\mathcal{F}'| = k^n$. Access to any side information allows us to identify a subspace of $\mathcal{F}'$ that is close to Y.

First, we consider access only to noisy node labels of G and denote Z the point in $\mathcal{F}'$ for this labeling. If we have no side information on the edges of G, the information theoretic optimal solution to statistical recovery is Z (because we assume $q < 1/2$). Second, we assume access only to edge measurements for G. We denote X the observed edge measurements. If the edge measurements are accurate (i.e., $p = 0$) the size of $\mathcal{F}'$ reduces to $k!$. We assume that k is such that one can obtain a labeling for G that is edge-compatible with X by traversing G. Under this assumption, the number of edge-compatible labelings is equal to all possible label permutations, i.e., $|\mathcal{F}'| = k!$. Finally, in the presence of both node and edge observations the information theoretic optimal solution to statistical recovery corresponds to a point Y^* that is obtained by running exact marginal inference [83]. However, exact inference can be intractable, and even when it is efficient, it is not clear what is the optimal Hamming error that Y^* yields with respect to Y.

To address these issues, we propose an approximate inference scheme and obtain a bound on the worst-case expected Hamming error that it obtains. We start with the noisy edge observations X and use them to find a subspace $\mathcal{F} \subset \mathcal{F}'$ that contains node labelings which induce edge labelings that are close to X (in terms of Hamming distance). We formalize this in the next two sections. Intuitively, we have that noisy edge measurements partition the space $\mathcal{F}$ in a collection of *edge classes*.

Definition 3.2.1. *The edge class of a point $Y \in \mathcal{F}$ is a set $\mathbb{I} \in 2^{\{1,2,\ldots,k\}^{|V|}}$ such that for all $Y_i \in \mathbb{I}$, Y_i induces the same edge measurements as Y. All points in $\mathbb{I}$ can be derived via a label permutation of Y. In general, for any labeling Y', set $\mathbb{I}_{Y'}$ is the set of all labelings that can be generated by a label permutation of Y'.*

The restricted subspace $\mathcal{F}$ contains those edge classes that are close to the noisy edge observations X.

Figure 3.2 shows an example subspace $\mathcal{F}$ and the edge class-based partitions it contains.

Given the restricted subspace $\mathcal{F}$, we design an algorithm to find a point $\hat{Y} \in \mathcal{F}$ such that the Hamming error between $\hat{Y}$ and Y^* is minimized. We define the Hamming error with respect to an edge class $\mathbb{I}$ as:

Definition 3.2.2. *The Hamming error of a vector $\mathcal{Q} \in \{1, 2, \ldots, k\}^{|V|}$ to the edge class $\mathbb{I}_{Y'} \in 2^{\{1,2,\ldots,k\}^{|V|}}$ is $Hd(\mathcal{Q}, \mathbb{I}_{Y'}) = \min_{\mathcal{Y} \in \mathbb{I}_{Y'}} Hd(\mathcal{Y}, \mathcal{Q})$.*

Point Y^* might not be in $\mathcal{F}$ and the distance between $\hat{Y}$ and Y^* is the approximation error we have due to approximate inference. Finally, we prove that the expected Hamming error between

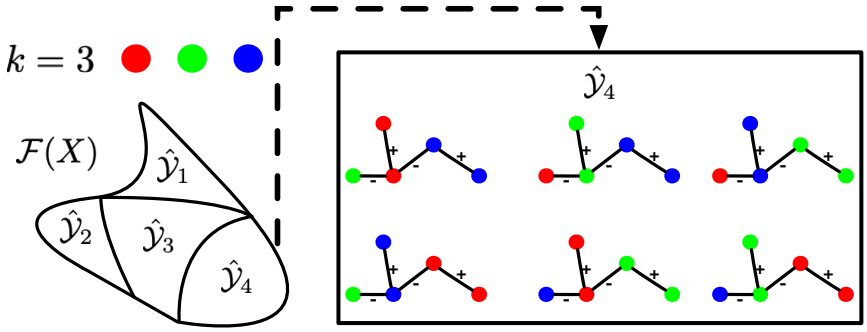


Figure 3.2: $\mathcal{F}(X)$ is a hypothesis space that is partitioned in different edge classes. We leverage the noisy edge measurements X to identify the edge class partitions that are close to X. Each partition contains $k!$ elements.

$\hat{Y}$ and Z is bounded. A schematic diagram of our approximate inference method is shown in Figure 6.3. In the following sections, we study statistical recovery for trees (in Section 3.3) and general graphs (in Section 3.4).

3.3 Recovery In Trees

We focus on trees and introduce a linear program for statistical recovery over k-categorical random variables. We prove that under a uniform noise model the optimal Hamming error is $\tilde{O}(\log(k) \cdot p \cdot n)$.

3.3.1 A Linear Program for Statistical Recovery

We follow the steps described in Section 3.2. First, we use the noisy edge observations to restrict the search for $\hat{Y}$ to a subspace $\mathcal{F}$. We describe $\mathcal{F}$ via a constraint on the number of edge disagreements between the edge labeling implied by $\hat{Y}$ and the noisy edge observations X. Second, we form an optimization problem to find a point $\hat{Y}$ with minimum Hamming distance from Z that satisfies the aforementioned constraint.

The ground truth edge labeling M (corresponding to the ground truth node labeling Y) has bounded Hamming distance from the observed noisy labeling X. Hence, we can restrict the space of considered solutions to node labelings that induce an edge labeling with a bounded Hamming distance from the observed noisy labeling X. We have: Under the uniform noise

model, edge measurements are flipped independently. Thus, the total number of bad edges is a sum over independent and identically distributed (iid) random variables. The expected number of flipped edges is $p \cdot |E| = p(n-1)$. Using the Bernstein inequality, we have:

Lemma 3.3.1. *Let G be a graph with noisy edge observations with noise parameter p. With probability at least $1-\delta$ over the draw of X:*

$$\sum_{(u,v)\in E} \mathbb{1}\{\varphi(Y_u, Y_v) \neq X_{u,v}\} \leq t \quad \textit{where}$$

$$t = (n-1)p + \frac{2}{3}\ln(\frac{2}{\delta})(1-p) + \sqrt{2(n-1)p(1-p)\ln(\frac{2}{\delta})}$$

Proof. In $G = (V, E)$, for each edge $(u, v) \in E$, we have a random variable $L_{u,v} = \mathbb{1}(\varphi(Z_u, Z_v) \neq X_{u,v})$ with distribution:

$$L_{u,v} = \begin{cases} 1, & p \\ 0, & 1-p \end{cases}$$

To apply the Bernstein inequality, we must consider $L_{u,v} - p$. We have $E[L_{u,v} - p] = 0$ and $\sigma^2(L_{u,v} - p) = p(1-p)$. We must also have that the random variables are constrained. We know that $|L_{u,v} - p| \leq \max\{1-p, p\}$ and $p < 1/2$ so $|L_{u,v} - p| \leq 1 - p$. Now, we apply the Bernstein inequality:

$$P\left(\sum_{(u,v)\in E} L_{u,v} - p \leq t\right) \geq 1 - \exp\left(-\frac{t^2}{2|E|\sigma^2 + \frac{2}{3}(1-p)t}\right)$$

Let $u \triangleq -\frac{t^2}{2|E|\sigma^2 + \frac{2}{3}t(1-p)}$. Solving for t we obtain:

$$t = \frac{1}{3}u(1-p) + \sqrt{\frac{(1-p)^2u^2}{9} + 2|E|\sigma^2 u}$$

Now we have that:

$$P\left(\sum L_{u,v} - p \leq \frac{1}{3}u(1-p) + \sqrt{\frac{(1-p)^2u^2}{9} + 2|E|\sigma^2 u}\right) \geq 1 - e^{-u}$$

We choose $u = \ln\left(\frac{2}{\delta}\right)$, and substituting $|E| = n - 1$ for trees and $\sigma^2 = p(1-p)$, we have that with probability $1 - \delta$:

$$\sum_{(u,v)\in E} \mathbb{1}(\varphi(u,v) \neq X_{u,v}) \leq$$
$$\frac{1}{3}\ln(\frac{2}{\delta})(1-p) + \sqrt{\frac{(1-p)^2\ln(\frac{2}{\delta})^2}{9} + 2(n-1)p(1-p)\ln(\frac{2}{\delta})} + (n-1)p$$

Simplifying by noting that $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$, we have proven the lemma.

□

This lemma states that under the uniform noise model the ground truth edge labeling M for Graph G is in the neighborhood of X with high probability. Given this bound, we use the following linear program to find $\hat{Y}$:

$$\begin{aligned} &\underset{\hat{Y} \in [k]^{|V|}}{\text{minimize}} \quad \sum_{v\in V} \mathbb{1}\{\hat{Y}_v \neq Z_v\} \\ &\text{subject to} \\ &\sum_{(u,v)\in E} \mathbb{1}\{\varphi(\hat{Y}_u, \hat{Y}_v) \neq X_{u,v}\} \leq t \end{aligned} \tag{3.1}$$

where t is defined as in Lemma 3.3.1. This problem can be solved via a dynamic programming algorithm with cost $O(k \cdot n^3 \cdot p)$. We describe this algorithm in Section 3.3.2.

Discussion Our approach is similar to that of [77] for binary random variables. However, we use the Bernstein inequality to obtain a tighter concentration bound on the number of flipped edge measurements. In the case of categorical random variables, it is critical to obtain a tight description of the space $\mathcal{F}$ of the possible labeling solutions as we have a larger hypothesis space.

Let $\mathcal{S}(n,k)$ be the size of hypothesis space with k labels and n nodes. If we increase n by one, the rate of change for the hypothesis space is $r_{k,n} = \Delta\mathcal{S}/\Delta n = k^n(k-1)$, which is multiplicative with respect to k. Similarly, as we increase k to $k+1$ the size of the hypothesis space changes by $s_{k,n} = \Delta\mathcal{S}/\Delta k = \sum_{i+j=n-1}(k+1)^i k^j \geq k^{n-1}$, which is exponential in the size of our input. We need a tight bound to obtain an efficient dynamic programming algorithm with respect to n and k.

3.3.2 Solving the Optimization Problem on Trees with Dynamic Programming

Because G is assumed to be a tree, we can compute optimal solutions to subproblems. Specifically, we can turn any undirected tree into a controlled one by a breadth-first search.

Then we can define a table $OPT(u, B|\ell)$ which stores optimal values to the subtree rooted at u, constrained to budget B and with the parent of u constrained to class ℓ. Given the values of OPT for all descendants of a node u, it is not difficult to find values for the table at u. We formalize this in the following theorem.

Theorem 3.3.1. *The optimization problem 1 can be solved in time $O(kn^3p)$.*

Proof. Given a tree $T = (V, E)$, a budget t, observations $X = \{X_{u,v}\}_{(u,v)\in E}$ and $Z = \{Z_v\}_{v\in V}$, we would like to compute a solution to

$$\sum_{(u,v)\in E} \mathbb{1}(\varphi(Z_u, Z_v) \neq X_{u,v}) \leq \frac{2}{3}\ln(2/\delta)(1-p) + \sqrt{2(n-1)p(1-p)\ln(2/\delta)} + (n-1)p$$

First, we turn T into a tree rooted at some node r by running a breadth-first search from r and directing nodes according to their time of discovery. Call this directed tree rooted at r $\overrightarrow{T}_r$. We specify a table OPT which will collect values of optimal subproblems.

Specifically, denote $\overrightarrow{T}_u$ as the subtree of $\overrightarrow{T}_r$ rooted at a node u. Then OPT will be a matrix parameterized by $OPT(u, B|\ell)$ where $u \in V$, $0 \leq B \leq |\overrightarrow{T}_u|$ (no tree can violate the observations more times than the number of nodes in the tree) and $\ell \in [k]$. Let $Pa(u)$ be the singular parent of the node u. Then OPT values represent the optimal value of the subtree rooted at u with a budget B and $Pa(u)$ restricted to the value ℓ. Our recursive equation for OPT is then

$$OPT(u, B|i) = \min_{\ell\in[k]} \min_{\substack{\sum_{v\in N(u)} B_v \\ = B - \mathbb{1}\{X_{u,v}\neq\varphi(i,\ell)\}}} \sum_{v\in N(u)} OPT(v, B_v|\ell) + \mathbb{1}\{\ell \neq Z_u\} \tag{3.2}$$

If we have the value of $OPT(u, B|\ell)$ for all nodes $u \neq r$, values ℓ and valid budgets $B \leq t$, we can calculate the optimum value of the tree by the following: We attach a node r' to r by an edge $r' \to r$ and set the information on the node to $X_{r',r} = 1$ then solve $OPT(r, t|1)$, then repeat the process but with $X_{r',r} = -1$, return the smaller of these two values.

For a leaf node w, the value of $OPT(w, B'|\ell)$ is simply $\min_i \mathbb{1}\{i \neq Z_w\}$ for $B' = 1$. If $B' = 0$ then we must choose i such that it does not violate the side information, i.e. we must have $\varphi(i, \ell) = X_{w,Pa(w)}$

Finally we show how to compute the summation in (3.2) efficiently. For each value $\ell \in [k]$ we must optimize the summation

$$\min_{\substack{\sum_{v \in N(u)} B_v \\ = B - \mathbb{1}\{X_{u,v} \neq \varphi(i,\ell)\}}} \left(\sum_{v \in N(u)} OPT(v, B_v|\ell) + \mathbb{1}\{\ell \neq Z_u\} \right)$$

Because each node's optimal value is independent, we can rewrite this sum by submitting an optional order on $N(u)$ of $1, 2, \ldots, m = |N(u)|$ and reforming this sum to

$$\min_{B_1 \in [0, K - \mathbb{1}\{\varphi(\ell,s) \neq X_{u,Pa(u)}\}]} OPT(1, B_1|\ell) + \min_{\sum_{j \in [2,m]} B_j = B - B_1 - \mathbb{1}\{\varphi(\ell,s) \neq X_{u,Pa(u)}\}} \sum_{j \in 2,m} OPT(j, B_j|\ell)$$

The minimization for the first two vertices whose number of constraints violated are at most B can be solved in $O(B^2)$ time. The calculation for the first three vertices can then be done in $O(B^2)$ time by reusing the information from the first two. We can repeat this until we have considered all children of u. Hence because we must calculate this value for all k possible classes, we get an algorithm which takes time $k \sum_{v \in V} |N(v)| B^2 = O(nkB^2)$. The statistical analysis below shows that B is $poly(n, p)$. □

3.3.3 Upper Bound on the Hamming Error for Trees

The Hamming error of $\hat{Y}$ obtained by Linear Program 3.1 is bounded by $\tilde{O}(\log(k) \cdot p \cdot n)$ with high probability. For our analysis, we draw connections to statistical learning.

We define a hypothesis class $\mathcal{F}$ that contains all points that satisfy the bound in Lemma 3.3.1:

$$\mathcal{F} = \{Y' \in [k]^{|V|} : \sum_{(u,v) \in E} \mathbb{1}\{\varphi(Y'_u, Y'_v) \neq X_{u,v}\} \leq t\}$$

From Lemma 3.3.1, we have that the edge class that corresponds to the ground truth labeling Y is contained in $\mathcal{F}$ with high probability over the draw of X. Moreover, since the node noise q is bounded away from $1/2$, we can use the noisy node measurements Z to find a labeling

$\hat{Y}$ that is in the same edge class as Y and close to Y. Such a labeling is obtained by solving Linear Program 3.1. From a statistical learning perspective, $\hat{Y}$ corresponds to the *empirical risk minimizer* (ERM) over $\mathcal{F}$ given Z. Thus, the Hamming error between $\hat{Y}$ and Y is associated with the *excess risk* over Z for Class $\mathcal{F}$. We have:

Lemma 3.3.2. *[77] Let $\hat{Y}$ be the empirical risk minimizer over $\mathcal{F}$ given Z and let $Y^* = \arg\min_{Y' \in \mathcal{F}} \sum_{v \in V} \mathbb{P}(Y'_v \neq Y_v)$ and $c > 0$ a constant number, then with probability $1 - \delta$ over the draw of Z,*

$$\sum_{v \in V} \mathbb{P}\big(\hat{Y}_v \neq Z_v\big) - \min_{Y' \in \mathcal{F}} \sum_{v \in V} \mathbb{P}\big(Y'_v \neq Z_v\big) \leq$$
$$\left(\frac{2}{3} + \frac{c}{2}\right) \log\left(\frac{|\mathcal{F}|}{\delta}\right) + \frac{1}{c} \sum_{v \in V} \mathbb{1}\big\{\hat{Y}_v \neq Y^*_v\big\}$$

We now analyze how the Hamming error relates to excess risk for categorical random variables. We have:

Lemma 3.3.3. *The Hamming error is proportional to the excess risk: For fixed $\hat{Y}, Y \sim \mathcal{F}'$ and Z distributed according to the uniform noise model we have that:*

$$\mathbb{1}\{\hat{Y}_v \neq Y_v\} = \frac{1}{c}\Big[P_Z(\hat{Y}_v \neq Z_v) - P_Z(Y_v \neq Z_v)\Big]$$
$$\textit{where} \quad c = 1 - k/(k-1)q$$

Proof. For $\hat{Y}_v = Y_v$, we have that $P_Z(Y_v \neq Z_v) - P_Z(Y_v \neq Z_v) = 0$ and so we are done. When $\hat{Y}_v \neq Y_v$, we have that $P_Z(Y_v \neq Z_v) = q$ and get the following for the first term:

$$\begin{aligned} P_Z(\hat{Y}_v \neq Z_v) &= P_Z(\hat{Y}_v \neq Z_v \wedge Z_v = Y_v) + P_Z(\hat{Y}_v \neq Z_v \wedge Z_v \neq Y_v) \\ &= P_Z(\hat{Y}_v \neq Z_v \wedge Z_v = Y_v) + \sum_{i \in [k] \wedge i \neq \hat{Y}_v \wedge i \neq Y_v} P_Z(\hat{Y}_v \neq Z_v \wedge Z_v \neq Y_v \wedge Z_v = i) \end{aligned}$$

We know that $P_Z(\hat{Y}_v \neq Z_v \wedge Z_v = Y_v) = 1 - q$. For each i we have $P_Z(\hat{Y}_v \neq Z_v \wedge Z_v \neq Y_v \wedge Z_v = i) = \frac{q}{k-1}$. So we have:

$$\sum_{i \in [k] \wedge i \neq \hat{Y}_v \wedge i \neq Y_v} P_Z(\hat{Y}_v \neq Z_v \wedge Z_v \neq Y_v \wedge Z_v = i) = \frac{q}{k-1} \sum_{i \in [k] \wedge i \neq \hat{Y}_v \wedge i \neq Y_v} 1 = \frac{q(k-2)}{k-1}$$

Given this we have:

$$P_Z(\hat{Y}_v \neq Z_v) = P_Z(\hat{Y}_v \neq Z_v \wedge Z_v = Y_v) + \sum_{i \in [k] \wedge i \neq \hat{Y}_v \wedge i \neq Y_v} P_Z(\hat{Y}_v \neq Z_v \wedge Z_v \neq Y_v \wedge Z_v = i)$$
$$= (1-q) + \frac{q(k-2)}{k-1} = 1 - \frac{q}{k-1}$$

Finally, consolidating these, we get:

$$P(\hat{Y}_v \neq Z_v) - P(Y_v \neq Z_v) = 1 - \frac{q}{k-1} - q = 1 - \frac{k}{k-1}q$$

This is exactly c, and so the hamming error and excess risk are proportional. Furthermore, we can set c to $1 - \frac{k}{k-1}q$. □

With $k = 2$ we have that $c = 1 - 2q$, which recovers the result of [77] for binary random variables.

Using Lemma 3.3.2, we can bound the excess risk in terms of the size of the hypothesis class. We have:

Corollary 3.3.1. *When $Y \in \mathcal{F}$ and $\hat{Y} = \arg\min_{\mathcal{Y} \in \mathcal{F}} \sum_{v \in V} \mathbb{1}\{\mathcal{Y}_v \neq Z_v\}$, we have that with probability at least $1 - \delta$ over the draw of Z:*

$$\sum_{v \in V} P(\hat{Y}_v \neq Z_v) - \min_{Y' \in \mathcal{F}} \sum_{v \in V} P(Y'_v \neq Z_v) \leq \left(\frac{4}{3} + \frac{2}{\frac{1}{4} + \left(\frac{1}{4} - \epsilon\right)\left(1 - \frac{k}{k-1}\right)} \right) \log\left(\frac{|\mathcal{F}|}{\delta} \right)$$

We now combine these results with the complexity of class $\mathcal{F}$ to obtain a bound for the Hamming error:

Theorem 3.3.2. *Let $\hat{Y}$ be the solution to Problem 3.1. Then with probability at least $1 - \delta$ over the draw of X and Z*

$$\sum_{v \in V} \mathbb{1}\{\hat{Y}_v \neq Y_v\} \leq \frac{[t \log(2k) - \log(\delta)]}{\left(1 - \frac{k}{k-1}q\right)} \left(\frac{4}{3} + \frac{2}{\frac{1}{4} + \left(\frac{1}{4} - \epsilon\right)\left(1 - \frac{k}{k-1}\right)} \right)$$
$$= \tilde{O}(\log(k) np)$$

Proof. By Lemma 1, we have with probability at least $1 - \frac{\delta}{2}$

$$\sum_{(u,v) \in E} \mathbb{1}(\varphi(u,v) \neq X_{u,v}) \leq t$$

which we will use it to define a hypothesis class $\mathcal{F}$ as

$$\mathcal{F} = \left\{ \hat{Y} : \sum_{(u,v)\in E} \mathbb{1}(\varphi(\hat{Y}_u, \hat{Y}_v) \neq X_{u,v}) \leq t \right\}$$

with

$$t = (n-1)p + \frac{2}{3}\ln(\frac{2}{\delta})(1-p) + \sqrt{2(n-1)p(1-p)\ln(\frac{2}{\delta})}$$

Which suggests that $Y \in \mathcal{F}$ with high probability. By Corollary 1, we have that $\hat{Y}$ being the ERM over $\mathcal{F}$ implies that

$$\sum_{v\in V} P(\hat{Y}_v \neq Z_v) - \min_{Y\in\mathcal{F}} \sum_{v\in V} P(Y_v \neq Z_v) \leq \left(\frac{4}{3} + \frac{2}{\frac{1}{4} + \left(\frac{1}{4} - \epsilon\right)\left(1 - \frac{k}{k-1}\right)} \right) \log\left(\frac{|\mathcal{F}|}{\delta}\right)$$

Combining this with Lemma 3, we conclude that $\sum_{v\in V} \mathbb{1}\{\hat{Y}_v \neq Y_v\}$ is bounded form above by

$$\frac{1}{1 - \frac{k}{k-1}q} \left(\frac{4}{3} + \frac{2}{\frac{1}{4} + \left(\frac{1}{4} - \epsilon\right)\left(1 - \frac{k}{k-1}\right)} \right) \log\left(\frac{|\mathcal{F}|}{\delta}\right)$$

Now, we approximate the size of the class $\mathcal{F}$. We can do so by upper-bounding the number of ways to violate the observed measurements. Pessimistically, of the possible $l = 0, 1, \ldots t$ violations, there are at most l nodes which are involved in this violation. Furthermore, there are at most $k-1$ ways for each of these nodes to be involved in such a violation. Therefore, we have,setting $t = \frac{2}{3}\ln(2/\delta)(1-p) + \sqrt{2(n-1)p(1-p)\ln(2/\delta)} + (n-1)p$

$$|\mathcal{F}| \leq \sum_{l=0}^{t} \binom{n}{l} k^l \leq k^t \sum_{l=0}^{t} \binom{n}{l} \leq k^t 2^t$$

Using this bound for $|\mathcal{F}|$, and assuming that the noise and sampling distribution is constant, we obtain that the hamming error is bounded by $\tilde{O}(\log(k)np)$ □

Here, t is the same as in Lemma 3.3.1. We see that k has a lower impact on the Hamming error than n and p. Also, when $k = 2$ we recover the result of [77]. Due to the tools we use to prove this result, this is a tight bound. We validate this bound empirically in Section 3.6.

3.4 Recovery In General Graphs

We now show how our tree-based algorithm can be combined with correlation clustering to obtain a non-trivial error rate for graphs with bounded treewidth and k-categorical random variables. We first describe our approximate inference algorithm and then show that our algorithm achieves an expected Hamming error of $\tilde{O}\big(k \cdot \log(k) \cdot p^{\lceil \frac{\Delta(G)}{2} \rceil} \cdot n\big)$ where $\Delta(G)$ is the maximum degree of the structured instance G.

3.4.1 Approximate Statistical Recovery

We build upon the concept of *tree decompositions* [55]. Let G be a graph, T be a tree, and $\mathcal{W} = (V_t)_{t \in T}$ be a family of vertex sets $V_t \subseteq V(G)$ indexed by the nodes t of T. We denote a tree-decomposition with $(T, \mathcal{W})$. The width of $(T, \mathcal{W})$ is defined as $\max\{|V_t| - 1 : t \in T\}$ and the treewidth $tw(G)$ of G is the minimum width among all possible decompositions of G. We also denote with F the $|\mathcal{W}| - 1$ edges connecting the bags in $\mathcal{W}$ in $(T, \mathcal{W})$ and represent T as $T = (\mathcal{W}, F)$.

Given a graph G, a tree decomposition of T defines a series of local subproblems whose solutions can be combined via dynamic programming to obtain a global solution for the original problem on G. For graphs of bounded treewidth, this approach allows us to obtain efficient algorithms [17]. Our solution proceeds as follows: Let $(T, \mathcal{W})$ be a tree decomposition of G. We first find a *local* labeling $\tilde{Y}^W$ for each $W \in \mathcal{W}$. Then, we design a dynamic programming algorithm that combines all local labelings to obtain a global labeling $\hat{Y}$.

Finding Local Labelings

We recover the labeling of the nodes in a bag W as follows: (1) Given W, we consider a superset of W, defined as $W^* = EXT(W) = W \cup \big(\bigcup_{v \in G} N(v)\big)$ where $N(v)$ is the one-hop neighborhood of node v; (2) Given W^*, we use the edge observations in the edge subset $E' \subseteq E$ induced by W^* to find a restricted hypothesis space $\mathcal{F}_{W^*}$. We then find a labeling $\tilde{Y}^{W^*} \in \mathcal{F}_{W^*}$ that has the minimum Hamming error with respect to Z for the nodes in W^*. Let Z_{W^*} denote this subset of Z; (3) For W, we assign $\tilde{Y}^W$ to be the restriction of $\tilde{Y}^{W^*}$ on W.

We consider two cases for Step 2 from above: (1) If $|W^*| = O(\log(n))$, we can enumerate all $k^{O(\log(n))}$ labelings for W^* and choose the one with minimum Hamming distance from Z. The complexity of this brute-force algorithm is $k^{O(\log(n))} = \texttt{poly}(n)$; (2) If $|W^*| = \Omega(\log(n))$, we use the MAXAGREE[k] algorithm of [80] over the noisy edge measurements X to restring

the subspace $\mathcal{F}$ in the neighborhood of X. MAXAGREE$[k]$ is a polynomial-time approximation scheme (PTAS) for solving the Max-Agreement version of correlation clustering for a fixed number of k labels. In the worst case, MAXAGREE$[k]$ obtains an approximation of $0.7666\text{OPT}[k]$. In our analysis, we account for the approximation factor 0.7666 by changing the probability p to $p' = 0.7666p + 0.2334$. Given the output of MAXAGREE$[k]$, let $\mathcal{F}_{CC}$ be the restricted subspace of solutions for W^*. We pick an arbitrary labeling $\bar{Y}^{W^*} \in \mathcal{F}_{CC}$ and use Algorithm 12 to get a permutation that transforms $\bar{Y}^{W^*}$ to point $\tilde{Y}^{W^*}$ that has minimum Hamming distance to Z^{W^*}.

Algorithm 5: Local Label Permutation

Input: A labeling $\bar{Y}^{W^*}$ in the subspace $\mathcal{F}_{CC}$ identified by MAXAGREE$[k]$ on W^*;
Node observations Z^{W^*}

$\bar{Y}_1^{W^*}, \bar{Y}_2^{W^*}, \dots \bar{Y}_k^{W^*} \leftarrow$ Group $\bar{Y}^{W^*}$ By Label
$Z_1^{W^*}, Z_2^{W^*}, \dots Z_k^{W^*} \leftarrow$ Group Z^{W^*} By Label
for $i, j \in [k] \times [k]$ **do**
  $I_{i,j} \leftarrow |\bar{Y}_i^{W^*} \cup Z_j^{W^*}|$
end
$Q \leftarrow$ A queue that sorts $I = \{I_{i,j}\}_{(i,j)\in[k]\times[k]}$ in decreasing order with respect to values $I_{i,j}$
while $Q \neq \emptyset$ **do**
  $I_{i,j} \leftarrow$ Pop(Q)
  $\pi(i) \leftarrow j$
  Remove all $I_{t,j}$ and $I_{i,t}$ for all $t \in [k]$ from Q
end
Return: π

Algorithm 12 greedily permutes the labels in $\bar{Y}^w$ to obtain a labeling with minimum Hamming distance to Z^W. The complexity of this algorithm is $O(n + k \log k)$.

Lemma 3.4.1. *Algorithm 12 finds a permutation π such that:*

$$\tilde{Y}^W = \pi(\bar{Y}^W) = \min_{\pi \in \Gamma_k} \sum_{v \in W} \mathbb{1}\{\pi(\bar{Y}^W) \neq Z^W\}$$

where Γ_k is the set of all permutations of the k labels.

Proof. We mix two partitions into one notation and each data point in D shows as $v_i = (\bar{Y}_i, Z_i)$, for each $i \in D$, and $\bar{Y}_i \in \bar{Y}$ and $Z_i \in Z$. We define $\forall l \in [k]$

$$X_l = \{v_i | \bar{Y}_i = l\}$$
$$T_l = \{v_i | Z_i = l\}$$

and the error is $E = \sum_{v_i \in D} \mathbb{1}\{Z_i \neq \bar{Y}_i\}$. The only thing that we allowed to change is the label of X_ls. We can represent the partition X and T as,

$$X = \{X_1, X_2, \ldots, X_k\}$$
$$T = \{T_1, T_2, \ldots, T_k\}$$

We claim that with Algorithm 2, we can find the permutation π on X, such that E minimize. Let π^* be the permutation that makes minimum E. We prove this theorem with reductio ad absurdum. Therefore

$$E_{\pi^*} \leq E_\pi \tag{3.3}$$

Let N be the set of all $v_i \in D$ such that $\pi(\bar{Y}_i) \neq \pi^*(\bar{Y}_i)$,

$$N = \{v_i \in D | \pi(\bar{Y}_i) \neq \pi^*(\bar{Y}_i)\}$$

We can write E for π,

$$\begin{aligned} E_\pi &= \sum_{v_i \in D} \mathbb{1}\{\pi(\bar{Y}_i) \neq Z_i\} \\ &= \sum_{v_i \in N} \mathbb{1}\{\pi(\bar{Y}_i) \neq Z_i\} + \sum_{v_i \notin N} \mathbb{1}\{\pi(\bar{Y}_i) \neq Z_i\} \end{aligned}$$

Similarly we can define E_{π^*},

$$\begin{aligned} E_{\pi^*} &= \sum_{v_i \in D} \mathbb{1}\{\pi^*(\bar{Y}_i) \neq Z_i\} \\ &= \sum_{v_i \in N} \mathbb{1}\{\pi^*(\bar{Y}_i) \neq Z_i\} + \sum_{v_i \notin N} \mathbb{1}\{\pi^*(\bar{Y}_i) \neq Z_i\} \end{aligned}$$

Second term in E_{π^*} and E_π are equal, using Inequality 3.3, and we define $E_\pi(N) = \sum_{v_i \in N} \mathbb{1}\{\pi(\bar{Y}_i) \neq Z_i\}$ and similarly $E_{\pi^*}(N)$ for π^*, so we have,

$$E_{\pi^*}(N) \leq E_\pi(N) \tag{3.4}$$

We know $N \subseteq D$, so the partition X on D present a sub-partition $\hat{X}$ on N. $\hat{X}$ defines like X, so $\hat{X} = \{\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k\}$. This sub-partition notion can be defined for both permutations π and π^*,

$$\hat{X}_\pi = \{\hat{X}_{\pi(1)}, \hat{X}_{\pi(2)}, \ldots, \hat{X}_{\pi(k)}\}$$
$$\hat{X}_{\pi^*} = \{\hat{X}_{\pi^*(1)}, \hat{X}_{\pi^*(2)}, \ldots, \hat{X}_{\pi^*(k)}\}$$

In the greedy algorithm, we sort the intersections of X_is and T_is and select the biggest one each time, because $\hat{X}$ is sub-partition of X, so we have,

$$\forall v_i, v_j \in \hat{X} \;\; \pi(\bar{Y}_i) = \pi(\bar{Y}_j) \longleftrightarrow \pi^*(\bar{Y}_i) = \pi^*(\bar{Y}_j) \tag{3.5}$$

Based on Equation 3.5, we can define a isomorphism on N,

$$\forall v_i \in N \;\; \varphi : \pi^*(\bar{Y}_i) \to \pi(\bar{Y}_i)$$

we define $\dot{\max}()$ as selecting the set with maximum size among all feasible sets, then we have,

$$\hat{X}_{\pi(\bar{Y}_i)} = \left\{ v_j \in D | \pi(\bar{Y}_i) = \pi(\bar{Y}_j), \pi(\bar{Y}_j) \neq Z_j, \dot{\max} |X_{\bar{Y}_j} \cap T_{\pi(\bar{Y}_j)}| \right\}$$

and also we can obtain,

$$\begin{aligned} E_{\pi^*}(N) &= \sum_{v_i \in N} \mathbb{1}\{\pi^*(\bar{Y}_i) \neq Z_i\} \\ &= \sum_{\hat{X}_i \in \hat{X}_{\pi^*}} \sum_{v=(\bar{Y},Z) \in \hat{X}_i} \mathbb{1}\{\pi^*(\bar{Y}) \neq Z\} \\ &= \sum_{\hat{X}_i \in \hat{X}_{\pi^*}} \sum_{v=(\bar{Y},Z) \in \hat{X}_i} \mathbb{1}\{\varphi^{-1}(\pi(\bar{Y})) \neq Z\} \end{aligned} \tag{3.6}$$

Also from Equation 3.6, we know

$$\dot{\max} |X_{\bar{Y}_j} \cap T_{\pi(\bar{Y}_i)}| = \hat{X}_{\pi(\bar{Y}_i)} \cup \underline{X}_{\pi(\bar{Y}_i)}$$

because $T_{\pi(\bar{Y}_i)}$ might already given to bigger intersection so we used $\dot{\max}$, and $\underline{X}_{\pi(\bar{Y}_i)}$ define as,

$$\underline{X}_{\pi(\bar{Y}_i)} = \left\{ v_j \in D | \pi(\bar{Y}_i) = \pi(\bar{Y}_j), \pi(\bar{Y}_j) = Z_j, \dot{\max} |X_{\bar{Y}_j} \cap T_{\pi(\bar{Y}_j)}| \right\}$$

Based on greedy $\underline{X}_{\pi(\bar{Y}_i)}$ is maximized on other hands from Equation 3.6, we know that $E_{\pi^*} \leq E_\pi$, so there exist equivalence C partition based on the φ, we have such that using Inequality 3.4,

$$\sum_{v \in C} \mathbb{1}\{\varphi^{-1}(\pi(\bar{Y})) \neq Z\} \leq \sum_{v \in C} \mathbb{1}\{\pi(\bar{Y}) \neq Z\} \tag{3.7}$$

moreover, this should be true for all $z \in C$. But if $\pi^*(\bar{Y})$ is not $\pi(\bar{Y})$ then,

$$\left| \left\{ v_i \in D; \mathbb{1}\{\pi^*(\bar{Y}_i) = y_i\} \right\} \right| < \underline{X}_{\pi(\bar{Y}_i)}$$

so this contradicting with Inequality 3.7 so for equivalence class C, we have

$$\sum_{v \in C} \mathbb{1}\{\pi^*(\bar{Y}) \neq Z\} = \sum_{v \in C} \mathbb{1}\{\pi(\bar{Y}) \neq Z\}$$

and because $\hat{X}_\pi$ and $\hat{X}_{\pi^*}$ is finite, this mean φ is identity function $\varphi(x) = x$ so $\pi = \pi^*$. That mean greedy algorithm finds the best permutation transformations that satisfies Z. □

We combine all steps in Algorithm 10. The output of this algorithm is a collection of labelings $\tilde{Y}$ for the local problems. Lemma 3.4.1 states that $\tilde{Y}^{W^*}$ minimizes the Hamming distance to Z. We also show that $\tilde{Y}^{W^*}$ remains a minimizer with respect to $\min_y \sum_{(u,v)} \mathbb{1}(\varphi(y_u, y_v) \neq X_{uv})$ after the swaps due to π.

Definition 3.4.1. *Given a graph $G = (V, E)$, the $swap(V, c_1, c_2)$ function changes all node labels c_1 to c_2, and all node labels c_2 to c_1.*

The swap operation enables us to switch between elements within an edge class. We show that a $swap(V, c_1, c_2)$ does not affect the disagreements between the node labeling and edge labeling of a graph.

Lemma 3.4.2. *Let L be a set of labels $L = \{1, 2, \ldots, k\}$. Consider a graph $G = (V, E)$ for which we are given a node labeling Y and an edge labeling X. For any pair $(c, c') \in L \times L$, let $Y' = swap(V, c, c')$ be the node labeling of G after swapping label c with c'. We have that: $\sum_{(u,v)\in E} \mathbb{1}\{\varphi(Y_u, Y_v) \neq X_{u,v}\} = \sum_{(u,v)\in E} \mathbb{1}\{\varphi(Y'_u, Y'_v) \neq X_{u,v}\}$.*

Proof. Let $G = (V, E)$, and set Y is the node labels from L assigned to V. Let $C \subseteq L$ be the set of all labels that used in Y. The easy case is when we want to change a color $c \in Y$ to $c' \notin Y$, this is like renaming. To proof this lemma, we use induction. For showing an edge, we use $i + j$

Algorithm 6: Find Local Labelings

Input: A tree decomposition $T = (\mathcal{W}, F)$ of G; Noisy node observations Z; Noisy edge measurements X

$\tilde{Y} \to \emptyset$
for $W \in \mathcal{W}$ **do**
  $W^* = EXT(W)$
  $\setminus*$ The next optimization problem can be solved either via enumeration or correlation clustering. $E(W^*)$ denotes the set of edges in W^*.$*\setminus$
  $\bar{Y}^{W^*} = \arg\min_{y} \sum_{(u,v)\in E(W^*)} \mathbb{1}\{\varphi(y_u, y_v) \neq X_{uv}\}$
  $\tilde{Y}^{W^*} \leftarrow$ Local Label Permutation $(\bar{Y}^{W^*}, Z^{W^*})$
  Let $\tilde{Y}^W$ be the restriction of $\tilde{Y}^{W^*}$ to W
  $\tilde{Y} \to \tilde{Y} \cup \{\tilde{Y}^W\}$
end
Return: $\tilde{Y}$

means that two end point of nodes have label i and j and the edge label is $+1$. Let $C = \{c, c'\}$, we have multiple scenarios that generate violation $Vi = \{c' + c, c + c', c - c, c' - c', \}$ and also the set of non-violation scenarios is $nVi = \{c' - c, c - c', c + c, c' + c'\}$ as you can see nVi and Vi closed under swap operation.

We assume the theorem is true for $|C| = k - 1$, let Y used for k colors to color them. We know $k - 1$ colors can swap, only color k is matter now, consider swap $i \in [k - 1]$ and k. All edges involve in this swap is $\{i+k, i-k, k+i, k-i, i+i, i-i, k-k, k+k\}$ and errors involved with these two labels are $\{i + k, k + i, i - i, k - k\}$, and this set size does not change after the swap.

Based on the statement at the beginning of the proof, we are sure about k appear to $[k - 1]$ colors, because it is like renaming, the only thing is changing k to i. Let j be a label such that $e = (v_i, v_j) \in E : label(v_l) = j \bigwedge label(v_m) = k$, the number of error are $\{j+k, k+j\}$ and after swap we have same number of edge in this set. So Y and its version after swap, Y' have same number of edge violations on the label set L, In other word, for any L, we have the following statement. $\sum_{(u,v)\in E} \mathbb{1}\{\varphi(Y_u, Y_v) \neq X_{u,v}\} =$
$\sum_{(u,v)\in E} \mathbb{1}\{\varphi(Y'_u, Y'_v) \neq X_{u,v}\}$ □

This lemma implies that $\tilde{Y}^{W^*}$ is a minimizer of $\min_y \sum_{(u,v)} \mathbb{1}\{\varphi(y_u, y_v) \neq X_{uv}\}$ since $\bar{Y}^{W^*}$ minimizes this quantity, and $\tilde{Y}^{W^*}$ is a permutation of $\bar{Y}^{W^*}$.

From Local Labelings to a Global Labeling

We now describe how to combine labelings $\{\tilde{Y}^W\}_{W \in \mathcal{W}}$ into a global labeling $\hat{Y}$. For binary random variables, the following procedure plays a central role in enforcing agreement across local labelings [77]: Given a bag W_1 and a neighbor W_2 with conflicting node labels with respect to W_1, we can maximize the agreement between W_1 and W_2 by flipping labeling $\tilde{Y}^{W_1}$ to its mirror labeling. This operation leads to consistent solutions since for binary random variables there is only one mirror labeling. However, for categorical random variables we have $k!$ possible mirror labelings for $\tilde{Y}^{W_1}$. *We show that it suffices to consider only one label swap per bag instead of* $k!$ *labelings.*

We consider the swap operation (see Section 3.4.1) and two bags W_1 and W_2 with labelings $\tilde{Y}^{W_1}$ and $\tilde{Y}^{W_2}$. We resolve conflicts in $W_1 \cap W_2$ as follows: Let $\Pi_k \subset \Gamma_k$ be the set of all permutations restricted to one pairwise color swap. Given a bag $W \in \mathcal{W}$ with labeling Y^W, we define a swap $\pi = swap(W, c_i, c_j)$ to be valid if color c_i is present in Y_W. Given a valid swap π for W, we define $\pi(Y^W)$ to be the label assignment for all nodes in W after applying π to Y^W. Also, let $\pi(Y_v^W)$ be the labeling for a node $v \in W$ after π. Finally, we define $\Pi_k(Y^W)$ as the set of all labelings for W that can be obtained if we apply any valid pairwise label swap on Y^W. To resolve inconsistencies between $\tilde{Y}^{W_1}$ and $\tilde{Y}^{W_2}$, we consider pairs in $\Pi_k(Y^{W_1}) \times \Pi_k(Y^{W_2})$ such that the labeling in the intersection of W_1 and W_2 is consistent and the number of nodes whose label is swapped is minimum.

The procedure we use is shown in Algorithm 19. The algorithm takes as input a tree decomposition $T = (\mathcal{W}, F)$ of G and the local labelings $\tilde{Y}$. For each W with labeling $\tilde{Y}^W$, we compute the cost of swapping label c_i with label c_j for each $(i, j) \in [k] \times [k]$. Then, we iterate over edges in F to identify incompatibilities between local node labelings. Finally, we use all the computed costs to find the single swap π_W to be applied locally to each bag $W \in \mathcal{W}$ such that global agreement is maximized. To this end, we solve a linear program similar to program 3.1. This program is shown in Algorithm 8.

In Algorithm 8, function $\psi(\cdot)$ is defined as:

$$\psi(\pi_W, \pi_{W'}) = \begin{cases} 1, \text{ if } \pi_W(\tilde{Y}_v^W) = \pi_{W'}(\tilde{Y}_v^{W'}) : \forall v \in W \cap W' \\ -1, \text{ if } \pi_W(\tilde{Y}_v^W) \neq \pi_{W'}(\tilde{Y}_v^{W'}) : \exists v \in W \cap W' \end{cases}$$

Constant L_n is used to restrict the space of solutions considered. A discussion on L_n is deferred to Section 3.4.2.

Algorithm 7: From Local Labelings to a Global Labeling

Input: A tree decomposition $T = (\mathcal{W}, F)$ of G; Noisy node observations Z; Noisy edge measurements X; Local labelings $\{\tilde{Y}^W\}_{W \in \mathcal{W}}$

$\hat{Y} \to \emptyset$
for $W \in \mathcal{W}$ **do**
  $\Pi_k^W \leftarrow$ the set of valid pairwise color swaps for W
  for $\pi \in \Pi_k^W$ **do**
    $\backslash * \ \pi$ is associated with a label swap (c_i, c_j) $* \backslash$
    $Cost_W[\pi] = \sum_{v \in W} \mathbb{1}(\pi(\tilde{Y}^W) \neq Z^W)$
  end
end
for $(W_1, W_2) \in F$ **do**
  Select one node v from $W_1 \cap W_2$ randomly
  $S(W_1, W_2) = 2 \cdot \mathbb{1}\{\tilde{Y}_v^{W_1} = \tilde{Y}_v^{W_2}\} - 1$
end
Compute constant L_n; $\backslash *$ See Section 3.4.2 $* \backslash$
$\{\pi_W\}_{W \in \mathcal{W}} =$ Cat. Tree Decoder$(T, Cost, S, L_n)$
for $v \in V$ **do**
  Choose arbitrary W s.t. $v \in W$ randomly
  $\hat{Y}_v = \pi_W(\tilde{Y}_v^W)$
end
Return: $\hat{Y}$

Algorithm 8: Categorical Tree Decoder

Input: A tree $T = (\mathcal{W}, F)$; Matrices $\{Cost_W\}_{W \in \mathcal{W}}$, $\{S(W, W')\}_{(W,W') \in F}$, $L_n \in \mathbb{N}$
Output: Optimal swaps $\{\pi_W\}_{W \in \mathcal{W}}$ for each $W \in \mathcal{W}$

Solve the linear program:

$$\hat{\Pi} = \arg \min_{\{\pi_W\}_{W \in \mathcal{W}} \in \Pi_k^{|\mathcal{W}|}} \sum_{W \in \mathcal{W}} Cost_W[\pi_W]$$

$$\text{s.t.} \sum_{(W,W') \in F} \mathbb{1}\{\psi(\pi_W, \pi_{W'}\} \neq S(W, W') \leq L_n$$

Return: $\hat{\Pi}$

Discussion on Correlation Clustering

We use correlation clustering in our algorithm for practical reasons. If the cardinality of the bags $T = (\mathcal{W}, F)$ is bounded by $O(\log(n))$, we can find a local labeling for each W that has minimum Hamming distance to Z efficiently. Obtaining such a decomposition T is an NP-complete problem. This challenge is also highlighted by [77]. To address this issue they assume a sampling procedure for removing edges from G to obtain a subgraph for which a low-width tree decomposition is easy to find. This procedure is a graph-specific exercise and not easily generalizable to arbitrary graphs. We follow a different approach. Instead of using specialized procedures, we rely on heuristics to obtain a low-width decompositions [49, 52] and use correlation clustering for large bags. This scheme allows us to use our algorithm with arbitrary graphs.

3.4.2 A Bound for Low Treewidth Graphs

We state our main theorem for statistical recovery over general graphs. We also provide a proof sketch.

Theorem 3.4.1. (**Main Theorem**) *Consider graph G with $T = (\mathcal{W}, F)$, noisy node observations Y, and noisy edge observations X. Let $\hat{Y}$ be the statistical recovery solution obtained by combining Algorithms 10 and 19. With high probability over the draw of Z and X:*

$$\sum_{v \in V} \mathbb{1}\{\hat{Y}_v \neq Y_v\} \leq \tilde{O}\big(k \cdot \log k \cdot p^{\lceil \frac{\texttt{mincut}^*(G)}{2} \rceil} \cdot n\big)$$

$$\leq \tilde{O}\big(k \cdot \log k \cdot p^{\lceil \frac{\Delta}{2} \rceil} \cdot n\big)$$

where $mincut^*(G)$ is the min. mincut over all extended bags in $\mathcal{W}$ and $\Delta(G)$ is the max. degree in G.

We see that the Hamming error obtained by our approach goes to zero as $p \to 0$. Theorem 3.4.1 allows us to understand when statistical recovery over a graph with categorical random variables is possible (i.e., when we can rely on edge observations to solve statistical recovery more accurately than the trivial solution of keeping the initially assigned node labels). Theorem 3.4.1 connects the level of edge-noise with the degree Δ of the input graph, the number of labels k, and the noise q on node labels. We have that for the edge noise p it should be $p \leq \sqrt[\lceil \frac{\Delta}{2} \rceil]{\frac{q}{k \log k}}$, where q is the node noise parameter, for the side information in X to be useful for statistical recovery. Otherwise, one should just use the initially observed node labels.

Proof Sketch Let S denote a maximal connected subgraph of G. Let $\delta(S)$ be the boundary of S, i.e., the set of edges with exactly one endpoint in S. Let $\tilde{Y}^S$ be the local labeling for nodes in S. We say that S is incorrectly labeled if for all $v \in S$ we have $\tilde{Y}_v^S \neq Y_v$. We have:

Lemma 3.4.3. *(Swapping lemma) Let S be a maximal connected subgraph of G with every node incorrectly labelled by $\tilde{Y}$. Then at least half the edges of $\delta(S)$ are bad.*

Proof. Let $\delta(S)^+$ and $\delta(S)^-$ show the positive and negative edges in $\delta(S)$. We define the external boundary nodes as follow,

$$V^S = \{v \in G : (v, e) \in \delta(S) \wedge v \notin S\}$$

and internal boundary nodes as

$$V_S = \{v \in G : (v, e) \in \delta(S) \wedge v \in S\}$$

It is simple to verify that for each $v \in V^S$ there exist $u \in V_S$ such that $(u, v) \in \delta(S)$ and vice versa. We know that $\tilde{Y}_v^W = Y_v$ for $v \in V^S$. If $\delta(S)^- = \emptyset$ and all edges in $\delta(S)$ be correct, we can follow the labels node in V^S, so for each $v \in V^S$ we select the edges (v, u) in $\delta(S)$ and we define $swap(S, v, u)$ so we have set of mapping $\Phi^+(S) = \{swap(S, v, u) : v \in V^S \wedge u \in V_S \wedge (u, v) \in \delta(S)\}$, from Lemma 5, we know the that the number of violations in S is same, so we resolved some violations in $\delta(S)$ which has contradiction with $\tilde{Y}^W \in \mathbb{I}_{min}$, so when $\delta(S)^- = \emptyset$, at least half of nodes are incorrect and we actually can derive the labeling.

Let $\Gamma_k(S)$ be all label permutation in S such that each permutation can be represented with a sequence of swaps. We can easily show that any sequence of swap is also does not change the edge violation, so we know for all $\pi \in \Gamma_k(S)$ the number of edge violations in S is constant. Because V^S is correct labeled so at least $\lceil \frac{\delta(S)}{2} \rceil$ of edges in $\delta(S)$ are incorrect, otherwise there exist a labeling permutation that contradict with minimization of edge violation because the edges inside S does not add violation but we resolve more than half of $\delta(S)$, In this case we

know the existential of such a this permutation but in binary and $\delta(S)^- = \emptyset$ cases, we can actually build the better permutation.

□

For a bag W, let set S be the largest connected component in W such that for all nodes v in it $\tilde{Y}_v^W \neq Y_v$. It must be the case that at least half of the $\delta(S)$ edges are incorrect or else there exists a different labeling that agrees with X better than $\tilde{Y}^W$. This contradicts the fact that $\tilde{Y}^W$ is a minimizer of $\min_y \sum_{(u,v)} \mathbb{1}\{\varphi(y_u, y_v) \neq X_{uv}\}$. This result extends the Flipping Lemma of [83] from the binary to the categorical case.

We use this result to bound the probability that a local labeling $\tilde{Y}^W$ (see Lemma 3.4.1) will fail to recover the ground truth node label for W. The probability of local labelings having large Hamming error is upper bounded:

Lemma 3.4.4. *Let Γ_k be the all label permutations on the set $L = \{1, 2, \ldots, k\}$. We have for W:*

$$\mathbb{P}\left(\min_{\pi \in \Gamma_k} \mathbb{1}\{\pi(\bar{Y}^W) \neq Y^W\} > 0 \right) \leq 2^{|W^*|} p^{\lceil \frac{mincut^*(W)}{2} \rceil}$$

with $mincut^*(W) = \min_{S \subset W^*, S \cap W \neq \emptyset, \bar{S} \cap W \neq \emptyset} |\delta_{G(W)}(S)|$.

Proof. From Lemma 6, we know at least half of $\delta(S)$ for any $S \subset W^*$ are incorrect, so we used this to find an upper bound for this probability, so the best permutation of labels also should satisfy Lemma 6 so we have

$$\begin{aligned}
&\mathbb{P}\left(\min_{\pi \in \Gamma_k(W^*)} \mathbb{1}\{\pi(\bar{Y}^{W^*}) \neq Y^W\} > 0 \right) \leq \sum_{S \subset W^*, S \cap W \neq \emptyset, \bar{S} \cap W \neq \emptyset} p^{\lceil \frac{\delta(S)}{2} \rceil} \\
&\leq \sum_{S \subseteq W^*} p^{\lceil \frac{mincut^*(W)}{2} \rceil} \textit{(because } |\delta(S)| \leq mincut^*(W) \textit{ for all } S \subseteq W^*) \\
&\leq 2^{|W^*|} p^{\lceil \frac{mincut^*(W)}{2} \rceil} \ \textit{(there are } 2^{|W^*|} \textit{ subsets)}
\end{aligned}$$

where $mincut^*(W) = min_{S \subset W^*, S \cap W \neq \emptyset, \bar{S} \cap W^* \neq \emptyset} |\delta_{G(W)}(S)|$ □

We now build upon Lemma 3.4.4 and leverage the result introduced by [19] to obtain an upper bound on the total number of mislabeled nodes across all bags in $\mathcal{W}$ for any labeling permutation $\pi \in \Gamma_k$ over the local labeling $\tilde{Y}^W$:

Lemma 3.4.5. *Let Γ_k be the all label permutations on the set $L = \{1, 2, \ldots, k\}$. For all $\delta > 0$, with probability at least $1 - \frac{\delta}{2}$ over the draw of X we have that:*

$$\min_{\pi \in \Gamma_k} \sum_{W \in \mathcal{W}} \mathbb{1}\{\pi(\tilde{Y}^W) \neq Y^W\} \leq 2^{|W|+1} p^{\lceil \frac{\texttt{mincut}(W)}{2} \rceil} + 6 \max_{e \in E} |\mathcal{W}(e)| \max_{W \in \mathcal{W}} |E(W)| \log(\frac{2}{\delta})$$

where $\mathcal{W}(e)$ denotes the set of bags in $\mathcal{W}$ that contain edge e and $E(W)$ denotes the set of edges in bag W.

We have following theorem from [19]

Theorem 3.4.2. *If there exists a constant $c > 0$ such that $V_+ \leq cS$ then*

$$\mathbb{P}\{S \geq \mathbb{E}[S] + t\} \leq exp\left(\frac{-t^2}{4c\mathbb{E}[S] + 2ct}\right)$$

Subsequently, with probability at least $1 - \delta$,

$$\begin{aligned} S &\leq \mathbb{E}(S) + \max\left\{4c\log(\frac{1}{\delta}), 2\sqrt{2c\mathbb{E}(S)\log(\frac{1}{\delta})}\right\} \\ &\leq 2\mathbb{E}(S) + 6c\log(\frac{1}{\delta}). \end{aligned}$$

Now we can prove Lemma 3.4.5,

Proof. We define a random variable that shows the number of the component that has an error concerning the real labels of each component. This random variable is a function of given edges X.

$$S(X) = \sum_{W \in \mathcal{W}} \min_{\pi \in \Gamma_k(W)} \mathbb{1}\{\pi(\tilde{Y}^W(X)) \neq Y^W\} \tag{3.8}$$

We know $S(X) = 0$ means perfect matching with a given X and in maximum $S(X) = |\mathcal{W}|$, and also $\tilde{Y}^W(X)$ is the component-wise estimator with given edge labels observation X. We know that $S : [k]^{|E|} \to \mathbb{R}$ so we can use Theorem 3.4.2 if we can prove that $S(X)$ satisfies the assumption.

$$S(X) - S(X^{(e)}) = \sum_{W \in \mathcal{W}} \left(\min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] - \min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X^{(e)})) \neq Y^W\right] \right)$$

The right-hand side of the equation is zero for hypernodes that e is not in them so we can reduce the equation to the hypernodes that have e, so we show it with $\mathcal{W}(e)$. Formally $\mathcal{W}(e) = \{W \in \mathcal{W} | e \in E(W)\}$

$$S(X) - S(X^{(e)}) = \sum_{W \in \mathcal{W}(e)} \left(\min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] - \min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X^{(e)})) \neq Y^W\right] \right)$$

For evaluate Theorem 3.4.2, in next proposition we showed V_+ is bounded.

Proposition 3.4.1. *The variation of V_+ of $S(X)$ in Equation 3.8 is bounded, $V_+ \leq cS(X)$.*

Proof.

$$(S(X) - S(X^{(e)}))^2 . \mathbb{1}\left(S(X) > S(X)^{(e)}\right) =$$

$$\mathbb{1}\left(S(X) > S(X)^{(e)}\right) \times \sum_{W \in \mathcal{W}(e)} \left(\min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] - \min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X^{(e)})) \neq Y^W\right] \right)^2$$

$$\leq \sum_{W \in \mathcal{W}(e)} \left(\min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] \right)^2$$

//second part removed and square of minus part added

$$\leq |\mathcal{W}(e)| \sum_{W \in \mathcal{W}(e)} \min_{\pi \in \Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right]$$

Now we can use this for calculating the expectation.

We directly start with V_+ to find its bound.

$$
\begin{aligned}
V_+ &= \sum_{e\in E} \mathbb{E}\left[(S(X)-S(X^{(e)}))^2 \cdot \mathbb{1}\left(S(X) > S(X)^{(e)}\right)\middle| X_1, X_2, \ldots, X_n\right] \\
&= \sum_{e\in E} (S(X)-S(X^{(e)}))^2 \cdot \mathbb{1}\left(S(X) > S(X)^{(e)}\right) \times \\
&\mathbb{P}\left[(S(X)-S(X^{(e)}))^2 \cdot \mathbb{1}\left(S(X) > S(X)^{(e)}\right)\middle| X_1, X_2, \ldots, X_n\right] \\
&\text{(we assume all probabilities are 1)} \\
&\leq \sum_{e\in E} (S(X)-S(X^{(e)}))^2 . \mathbb{1}\left(S(X) > S(X)^{(e)}\right) \\
&\text{//from last result} \\
&\leq \sum_{e\in E} |\mathcal{W}(e)| \sum_{W\in\mathcal{W}(e)} \min_{\pi\in\Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] \\
&\leq \max_{e\in E} |\mathcal{W}(e)| \sum_{e\in E} \sum_{W\in\mathcal{W}(e)} \min_{\pi\in\Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] \\
&= \max_{e\in E} |\mathcal{W}(e)| \sum_{W\in\mathcal{W}(e)} \sum_{e\in E} \min_{\pi\in\Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] \\
&\leq \max_{e\in E} |\mathcal{W}(e)| \max_{W\in\mathcal{W}} |E(W)| \sum_{W\in\mathcal{W}(e)} \min_{\pi\in\Gamma_k(W)} \mathbb{1}\left[\pi(\tilde{Y}^W(X)) \neq Y^W\right] \\
&= \max_{e\in E} |\mathcal{W}(e)| \max_{W\in\mathcal{W}} |E(W)| S(X)
\end{aligned}
$$

Therefore, there is $c = \max_{e\in E} |\mathcal{W}(e)| \max_{W\in\mathcal{W}} |E(W)|$ such that $V_+ \leq cS(X)$.

□

So with $c = \max_{e\in E} |\mathcal{W}(e)| \max_{W\in\mathcal{W}} |E(W)|$, the Theorem 3.4.2 with probability at least $1-\frac{\delta}{2}$ is valid,

$$
S \leq 2\mathbb{E}(S) + 6 \max_{e\in E} |\mathcal{W}(e)| \max_{W\in\mathcal{W}} |E(W)| \log(\frac{2}{\delta})
$$

We only need to derive $\mathbb{E}(S)$ using Lemma 3.4.4, because $\tilde{Y} \in \mathbb{I}_{\bar{Y}}$, so Lemma 3.4.4 is also valid for $\tilde{Y}$,

$$\begin{aligned}
\mathbb{E}(S) &= \sum_{W \in \mathcal{W}} \mathbb{P}\Bigg(\min_{\pi \in \Gamma_k(W)} \mathbb{1}\Big\{ \pi(\tilde{Y}^W(X)) \neq Y^W \Big\} \Bigg) \times \min_{\pi \in \Gamma_k(W)} \mathbb{1}\Big\{ \pi(\tilde{Y}^W(X)) \neq Y^W \Big\} \\
&= \sum_{W \in \mathcal{W}} \mathbb{P}\Bigg(\min_{\pi \in \Gamma_k(W)} \mathbb{1}\Big\{ \pi(\tilde{Y}^W(X)) \neq Y^W \Big\} = 1 \Bigg) \\
&= \sum_{W \in \mathcal{W}} \mathbb{P}\Bigg(\min_{\pi \in \Gamma_k(W)} \mathbb{1}\Big\{ \pi(\tilde{Y}^W(X)) \neq Y^W \Big\} > 0 \Bigg) \\
&\leq \sum_{W \in \mathcal{W}} 2^{|W|} p^{\lceil \frac{mincut(W)}{2} \rceil} \text{ // from Lemma 3.4.4}
\end{aligned}$$

so finally we have,

$$\min_{\pi \in [\Gamma_k]^{\mathcal{W}}} \sum_{W \in \mathcal{W}} \mathbb{1}\{\pi(\tilde{Y}^W) \neq Y^W\} \leq \sum_{W \in \mathcal{W}} 2^{|W|+1} p^{\lceil \frac{mincut(W)}{2} \rceil} + 6 \max_{e \in E} |\mathcal{W}(e)| \max_{W \in \mathcal{W}} |E(W)| \log(\frac{2}{\delta})$$

□

This lemma can be extended to W^* as well. This lemma combined with Lemma 3.4.3 implies that the labeling disagreement across bags in the tree decomposition are bounded. The analysis continues in a way similar to that for trees (see Section 3.3). Given the local bag labelings, we seek to find the labeling swaps across bags such that the global labeling has minimum Hamming error with respect to Y. We use the inequality from Lemma 3.4.5 to restrict the space $([k] \times [k])^{\mathcal{W}}$ of all possible pairwise label swaps over the local bag labelings. Let s^* be the optimal point in $([k] \times [k])^{\mathcal{W}}$ such that the global labeling has minimum Hamming error with respect to Y. Given the tree decomposition $T = (\mathcal{W}, F)$ of G. We define the hypothesis space:

$$\begin{aligned}
\mathcal{F} \triangleq\ & ([k] \times [k])^{\mathcal{W}} \\
& \text{s.t.} \sum_{(W,W') \in F} \mathbb{1}\{\psi(\pi_W, \pi_{W'}) \neq S(W, W')\} \leq L_n \Big\}
\end{aligned}$$

with

$$L_n = deg(T) \big[2^{wid^*(W)+2} \sum_{W \in \mathcal{W}} p^{\lceil \frac{mincut^*(W)}{2} \rceil} + 6 deg^*_E(T) \max_{W \in \mathcal{W}} |E(W^*)| \log(\frac{2}{\delta}) \big]$$

,

$$deg_E^*(T) = \max_{e \in E} |\mathcal{W}(e)|$$

and π_W and $S(W, W')$ denote the pairwise swaps and labeling disagreements between bags from Algorithm 19. We show that the optimal permutation Π^* is a member of $\mathcal{F}$ with high probability and also have that $|\mathcal{F}(X)| \leq \left(\frac{e \cdot n \cdot k!}{L_n}\right)^{L_n}$. Combining this with Lemma 3.3.2, we take $\hat{\Pi}$ is most correlated with Z, i.e., it is a minimizer for $\sum_{W \in \mathcal{W}} \sum_{v \in W} \mathbb{1}\{\pi_W(\tilde{Y}_v^W) \neq Z_v\}$. Directly from statistical learning theory we have that the Hamming error of this estimator $\hat{Y}$ is $\tilde{O}(\log(\mathcal{F})) = \tilde{O}\big(k \cdot \log k \cdot p^{\lceil \frac{\Delta}{2} \rceil} \cdot n\big)$ which establishes our main theorem.

3.4.3 Proof of Theorem 3.4.1

From Lemma 3.4.5, we can directly proof same result for extend of tree components.

Corollary 3.4.2. *There is straightforward deduction to derive the result for $W^* = EXT(W)$ on $T = (\mathcal{W}, F)$ with probability $1 - \frac{\delta}{2}$,*

$$\min_{\pi \in [\Gamma_k]^{\mathcal{W}}} \sum_{W \in \mathcal{W}} \mathbb{1}\{\pi(\bar{Y}^{W^*}) \neq Y^{W^*}\} \leq \sum_{W \in \mathcal{W}} 2^{|W^*|+1} p^{\lceil \frac{mincut^*(W)}{2} \rceil} + 6 \max_{e \in E} |\mathcal{W}^*(e)| \max_{W \in \mathcal{W}} |E(W^*)| \log(\frac{2}{\delta})$$

we define the maximum size of a hyper-graph as its degree $deg_E^*(T) = \max_{e \in E} |\mathcal{W}^*(e)|$ which $\mathcal{W}^*(e) = \{W \in \mathcal{W} | e \in E(W^*)\}$ and $E(W^*)$ is the set of all edged in E that are in W^*, so we have

$$\min_{\pi \in [\Gamma_k]^{\mathcal{W}}} \sum_{W \in \mathcal{W}} \mathbb{1}\{\pi(\bar{Y}^{W^*}) \neq Y^{W^*}\} \leq 2^{wid^*(W)+2} \sum_{W \in \mathcal{W}} p^{\lceil \frac{mincut^*(W)}{2} \rceil} + 6 deg_E^*(T) \max_{W \in \mathcal{W}} |E(W^*)| \log(\frac{2}{\delta})$$

Where $wid^*(W) \triangleq \max_{W \in \mathcal{W}} |W^*| - 1$.

Now we can start to Theorem 3.4.1,

Proof. To prove this theorem, we need to define a hypothesis class and find information bound for the optimal solution in there, next, we can find a bound for the distance of the real answer of the problem and best answer in the hypothesis class.

Consider the following permutation finding of the components in T:

$$\Pi^\star = \arg\min_{\Pi \in \Gamma_k^{|\mathcal{W}|}} \sum_{W \in \mathcal{W}} \mathbb{1}\big\{\Pi(\tilde{Y}^W) \neq Y^W\big\}$$

from Corollary 3.4.2, we know that

$$\min_{\pi \in [\Gamma_k]^{\mathcal{W}}} \sum_{W \in \mathcal{W}} \mathbb{1}\{\pi(\tilde{Y}^W) \neq Y^W\} \leq K_n$$

Because $\tilde{Y}^{W^*}$ and $\bar{Y}^{W^*}$ both are in $\mathbb{I}_{\bar{Y}^{W^*}}$ and also $\tilde{Y}^W$ is $\tilde{Y}^{W^*}$ restricted to W and K_n is

$$\begin{aligned} K_n \triangleq & 2^{wid^*(W)+2} \sum_{W \in \mathcal{W}} p^{\lceil \frac{mincut^*(W)}{2} \rceil} + \\ & 6 deg_E^*(T) \max_{W \in \mathcal{W}} |E(W^*)| \log(\frac{2}{\delta}) \end{aligned}$$

So if we have $\Pi^\star$, we can produce a vertex prediction with at most K_n mistakes with probability $1 - \delta$. However, computing $\Pi^\star$ is impossible because we do not have access to Y, so we need to see using Z as a noisy version of Y, how much approximation error will add to the theoretical bound of prediction.

We define the following hypothesis class, which is defined with K_n so we make even bigger to include an even better possible solution.

$$\begin{aligned} \mathcal{F} \triangleq & ([k] \times [k])^{\mathcal{W}} \\ & \text{s.t.} \sum_{(W,W') \in F} \mathbb{1}\{\psi(\pi_W, \pi_{W'}) \neq S(W, W')\} \leq L_n\big\} \end{aligned}$$

In this context, each element of $([k] \times [k])^{\mathcal{W}}$ is a vector of size $\mathcal{W}$ element which each sown as π. Our goal is to show that best permutation is in $\mathcal{F}$ with high probability.

Such that $L_n = deg(T).K_n$ which enrich the hypothesis class with make it bigger than using K_n. We know that if $\min_{\pi \in [\Gamma_k(W)]} \mathbb{1}\{\pi(\tilde{Y}^W) \neq Y^W\} = 0$ for a component W then we can find a $\bar{\pi}_W \in \Gamma_k$ such that we can effect on Y^W to get $\tilde{Y}^W$ so $\bar{\pi}_W(Y^W) = \tilde{Y}^W$.

We also have

$$\sum_{(W,W') \in F} \mathbb{1}\{\psi(\pi_W, \pi_{W'}) \neq S(W, W')\} = \sum_{(W,W') \in F} \mathbb{1}\{\psi(\pi_W, \pi_{W'}) \neq [2.\mathbb{1}(\tilde{Y}_v^W, \tilde{Y}_v^{W'}) - 1]\}$$

and we know $v \in W \cap W'$, so if for each $W \in \mathcal{W}$ we have $\bar{\pi}_W$, if the range of π_W and $\pi_{W'}$ be same they get 1 and their range is Y, the right hand side also is 1 because the range of two permutations are Y^W and $v \in W \cap W'$, so $\mathbb{1}\{\psi(\pi_W, \pi_{W'}) \neq [2.\mathbb{1}(\tilde{Y}_v^W, \tilde{Y}_v^{W'}) - 1]\} = 0$ when ever W and W' have no errors. Therefore $\Pi^\star \in \mathcal{F}$ with probability $1 - \delta$. The complexity of hypothesis class can parametrized with the size of $\mathcal{F}(X)$ so we have

$$\begin{aligned}
|\mathcal{F}(X)| &= \sum_{m=0}^{L_n} \binom{|\mathcal{W}|}{m} k!^m \\
&\leq \sum_{m=0}^{L_n} \binom{|\mathcal{W}|}{m} k!^{L_n} = k!^{L_n} \sum_{m=0}^{L_n} \binom{|\mathcal{W}|}{m} \\
&\leq k!^{L_n} \left(\frac{e|\mathcal{W}|}{L_n}\right)^{L_n} \leq \left(\frac{e.n.k!}{L_n}\right)^{L_n}
\end{aligned}$$

We consider non-redundant decomposed trees which means for $(W_i, W_j) \in F$ we have $W_i \backslash (W_i \cap W_j) \neq \emptyset$. In Algorithm 3, we use Z instead of Y. So we have

$$\hat{\pi} = \min_{\pi \in \mathcal{F}(X)} \sum_{W \in \mathcal{W}} \sum_{v \in W} \mathbb{1}\big\{\pi(\tilde{Y}_v^W) \neq Z_v\big\}.$$

We have following lemma to continue the proof

Lemma 3.4.14. *For* $\sum_{v \in W} \mathbb{1}\big\{\hat{\pi}(\tilde{Y}_v^W) \neq \pi^\star(\tilde{Y}_v^W)\big\}$ *we have following approximation,*

$$
\begin{aligned}
&\sum_{v\in W} \mathbb{1}\big\{\hat{\pi}(\tilde{Y}_v^W) \neq \pi^\star(\tilde{Y}_v^W)\big\} \\
&= \frac{1}{c} \sum_{v\in W\wedge \pi^\star(\tilde{Y}_v^W)=Y_v} \Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v\big\} - \mathbb{P}_Z\big\{\pi^*_W(\tilde{Y}_v^W) \neq Z_v\big\}\Big\}_= \\
&+ \frac{1}{c'} \sum_{v\in W\wedge \pi^\star(\tilde{Y}_v^W)\neq Y_v} \Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v\big\} - \mathbb{P}_Z\big\{\pi^*_W(\tilde{Y}_v^W) \neq Z_v\big\}\Big\}_\neq
\end{aligned}
$$

such that $c = -\big(1 - \frac{k}{k-1}q\big)$ *and* $c' = 1 - \frac{k}{k-1}q$.

Proof. We prove this equation step by step

$$
\begin{aligned}
&\sum_{v\in W} \mathbb{1}\big\{\hat{\pi}(\tilde{Y}_v^W) \neq \pi^\star(\tilde{Y}_v^W)\big\} = \\
&\sum_{v\in W\wedge \pi^\star(\tilde{Y}_v^W)=Y_v} \mathbb{1}\big\{\hat{\pi}(\tilde{Y}_v^W) \neq \pi^\star(\tilde{Y}_v^W)\big\}_= + \sum_{v\in W\wedge \pi^\star(\tilde{Y}_v^W)\neq Y_v} \mathbb{1}\big\{\hat{\pi}(\tilde{Y}_v^W) \neq \pi^\star(\tilde{Y}_v^W)\big\}_\neq \\
&= \frac{1}{c} \sum_{v\in W\wedge \pi^\star(\tilde{Y}_v^W)=Y_v} \Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v\big\} - \mathbb{P}_Z\big\{\pi^*_W(\tilde{Y}_v^W) \neq Z_v\big\}\Big\}_= \\
&+ \frac{1}{c'} \sum_{v\in W\wedge \pi^\star(\tilde{Y}_v^W)\neq Y_v} \Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v\big\} - \mathbb{P}_Z\big\{\pi^*_W(\tilde{Y}_v^W) \neq Z_v\big\}\Big\}_\neq
\end{aligned}
$$

We have to derive each part of the relation separately, for both sigma if $\hat{\pi}(\tilde{Y}_v^W) = \pi^\star(\tilde{Y}_v^W)$ the above is true for any c and c'.

We need to calculate c and c', for c which is $\pi^\star(\tilde{Y}_v^W) = Y_v$, we have

$$
\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v \wedge \pi^\star(\tilde{Y}_v^W) = Y_v\big\} = \frac{k-2}{k-1}q
$$

and $\mathbb{P}_Z\big\{\pi^*_W(\tilde{Y}_v^W) \neq Z_v \big| \pi^\star(\tilde{Y}_v^W) = Y_v\big\} = 1 - q$ so we can calculate c.

$$
\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v\big\} - \mathbb{P}_Z\big\{\pi^*_W(\tilde{Y}_v^W) \neq Z_v\big\} = \frac{k-2}{k-1}q - (1-q) = -\big(1 - \frac{k}{k-1}q\big)
$$

so $c = -\big(1 - \frac{k}{k-1}q\big)$. Next, we calculate c' which is $\pi^\star(\tilde{Y}_v^W) \neq Y_v$, therefore we have

$$\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v \wedge \pi^\star(\tilde{Y}_v^W) \neq Y_v\big\} =$$
$$\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v \wedge \pi^\star(\tilde{Y}_v^W) \neq Y_v \wedge Y_v = Z_v\big\} + \mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v \wedge \pi^\star(\tilde{Y}_v^W) \neq Y_v \wedge Y_v \neq Z_v\big\} =$$
$$\frac{k-2}{k-1}q + 1 - \frac{1}{k-1}q = 1 - q$$

and for second part we have,

$$\mathbb{P}_Z\big\{\pi^\star(\tilde{Y}_v^W) \neq Z_v \wedge \pi^\star(\tilde{Y}_v^W) \neq Y_v\big\} =$$
$$\mathbb{P}_Z\big\{\pi^\star(\tilde{Y}_v^W) \neq Z_v \wedge \pi^\star(\tilde{Y}_v^W) \neq Y_v \wedge Y_v = Z_v\big\} + \mathbb{P}_Z\big\{\pi^\star(\tilde{Y}_v^W) \neq Z_v \wedge \pi^\star(\tilde{Y}_v^W) \neq Y_v \wedge Y_v \neq Z_v\big\}$$
$$= q + \frac{k-2}{k-1}q$$

so we can calculate c'

$$\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W) \neq Z_v\big\} - \mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W) \neq Z_v\big\} == (1-q) - \big[q + \frac{k-2}{k-1}q\big] = 1 - \frac{k}{k-1}q$$

therefore that $c' = 1 - \frac{k}{k-1}q$.

□

Fix $\hat{\pi} \in \mathcal{F}(X)$ for each component $W \in \mathcal{W}$ we have

$$
\begin{aligned}
&\sum_{v\in W}\mathbb{1}\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\}\\
&\quad\leq \sum_{v\in W}\mathbb{1}\{\hat{\pi}(\tilde{Y}_v^W)\neq \pi^\star(\tilde{Y}_v^W)\}+\sum_{v\in W}\mathbb{1}\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\} \quad \text{//Triangle inequality}\\
&\quad\leq \sum_{v\in W}\mathbb{1}\{\hat{\pi}(\tilde{Y}_v^W)\neq \pi^\star(\tilde{Y}_v^W)\}+|W|\mathbb{1}\{\pi^\star(\tilde{Y}_v^{W^*})\neq Y_v\} \quad \text{//Maximize component error}\\
&\quad= -\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)=Y_v}\left\{\mathbb{P}_Z\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\}-\mathbb{P}_Z\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\}\right\}+\\
&\qquad\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)\neq Y_v}\left\{\mathbb{P}_Z\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\}-\mathbb{P}_Z\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\}\right\}+\\
&\qquad|W|\mathbb{1}\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\} \quad \text{//From Lemma 3.4.14}
\end{aligned}
$$

For the first part, we can the following approximation:

$$
\begin{aligned}
&-\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)=Y_v}\left\{\mathbb{P}_Z\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\}-\mathbb{P}_Z\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\}\right\}\\
&\leq 2\sum_{v\in W}\mathbb{1}\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\}+\\
&\qquad\qquad\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)=Y_v}\left\{\mathbb{P}_Z\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\}-\mathbb{P}_Z\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\}\right\}\\
&\leq 2|W|\mathbb{1}\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\}+\\
&\qquad\qquad\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)=Y_v}\left\{\mathbb{P}_Z\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\}-\mathbb{P}_Z\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\}\right\}
\end{aligned}
$$

We conclude that:

$$
\begin{aligned}
&\sum_{v\in W}\mathbb{1}\big\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\big\}\leq 3|W|\mathbb{1}\big\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\big\}+\\
&\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)\neq Y_v}\Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\big\}-\mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\big\}\Big\}+\\
&\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)=Y_v}\Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\big\}-\mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\big\}\Big\}\\
&\leq 3|W|\mathbb{1}\big\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\big\}+\\
&\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)\neq Y_v}\Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\big\}-\mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\big\}\Big\}+\\
&\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W\wedge\pi^\star(\tilde{Y}_v^W)=Y_v}\Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\big\}-\mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\big\}\Big\}\\
&\leq 3|W|\mathbb{1}\big\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\big\}+\frac{1}{1-\frac{k}{k-1}q}\sum_{v\in W}\Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\big\}-\mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\big\}\Big\}
\end{aligned}
$$

We apply this formula for all components $W\in\mathcal{W}$ we have

$$
\begin{aligned}
&\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\big\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\big\}\\
&\leq 3\Big(\max_{W\in\mathcal{W}}|W|\Big)\sum_{W\in\mathcal{W}}\mathbb{1}\big\{\pi^\star(\tilde{Y}_v^W)\neq Y_v\big\}+\\
&\qquad\frac{1}{1-\frac{k}{k-1}q}\sum_{W\in\mathcal{W}}\sum_{v\in W}\Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\big\}-\mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\big\}\Big\}\\
&\leq 3\Big(\max_{W\in\mathcal{W}}|W|\Big)K_n+\frac{1}{1-\frac{k}{k-1}q}\sum_{W\in\mathcal{W}}\sum_{v\in W}\Big\{\mathbb{P}_Z\big\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\big\}-\mathbb{P}_Z\big\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\big\}\Big\}
\end{aligned}
$$

using Lemma 3.3.2 for right hand side of the equation, we have excess risk bound with probability $1-\frac{\delta}{2}$,

$$\sum_{W\in\mathcal{W}}\sum_{v\in W}\left\{\mathbb{P}_Z\{\hat{\pi}(\tilde{Y}_v^W)\neq Z_v\}-\mathbb{P}_Z\{\pi_W^*(\tilde{Y}_v^W)\neq Z_v\}\right\}$$
$$\leq\left(\frac{2}{3}+\frac{c}{2}\right)\log(\frac{2|\mathcal{F}(X)|}{\delta})+\frac{1}{c}\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\}$$

so we can mix these inequalities,

$$\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\}$$
$$\leq 3\left(\max_{W\in\mathcal{W}}|W|\right)K_n+\frac{1}{1-\frac{k}{k-1}q}\left(\frac{2}{3}+\frac{c}{2}\right)\log(\frac{2|\mathcal{F}(X)|}{\delta})+\frac{1}{c}\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\}$$

so we have

$$\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\}\leq\frac{1}{1-\frac{1}{c}}\left[\left(3\max_{W\in\mathcal{W}}|W|\right)K_n+\frac{1}{1-\frac{k}{k-1}q}\left(\frac{2}{3}+\frac{c}{2}\right)\log(\frac{2|\mathcal{F}(X)|}{\delta})\right]$$

We put $c=\frac{1}{1-\epsilon}$ and rearrange then with probability $1-\delta$ we have

$$\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\}$$
$$\leq\frac{1}{1-\frac{1}{\frac{1}{1-\epsilon}}}\left[\left(3\max_{W\in\mathcal{W}}|W|\right)K_n+\frac{1}{1-\frac{k}{k-1}q}\left(\frac{2}{3}+\frac{\frac{1}{1-\epsilon}}{2}\right)\log(\frac{2|\mathcal{F}(X)|}{\delta})\right]$$
$$=\frac{1}{\epsilon}\left[\left(3\max_{W\in\mathcal{W}}|W|\right)K_n+\frac{1}{1-\frac{k}{k-1}q}\left(\frac{2}{3}+\frac{1}{2(1-\epsilon)}\right)\log(\frac{2|\mathcal{F}(X)|}{\delta})\right]$$

From before, we have $|\mathcal{F}(X)|\leq\left(\frac{en.k!}{L_n}\right)^{L_n}$, $wid(T)=\max_{W\in\mathcal{W}}|W|$, K_n, and Lemma 3.3.2 so we can conclude

$$\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\big\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\big\}=$$
$$=\frac{1}{\epsilon}\left(3\max_{W\in\mathcal{W}}|W|\right)K_n+\frac{1}{\epsilon.\big(1-\frac{k}{k-1}q\big)}\left(\frac{2}{3}+\frac{1}{2(1-\epsilon)}\right)\log(\frac{2|\mathcal{F}(X)|}{\delta})$$
$$=\frac{3}{\epsilon}.wid(T).K_n+\frac{1}{\epsilon.\big(1-\frac{k}{k-1}q\big)}\left(\frac{2}{3}+\frac{1}{2(1-\epsilon)}\right)\times\big(\log(\frac{2}{\delta})+L_n.\log(\frac{en.k!}{L_n})\big))$$
$$\leq\frac{3}{\epsilon}.wid(T).K_n+\frac{1}{\epsilon.\big(1-\frac{k}{k-1}q\big)}\left(\frac{2}{3}+\frac{1}{2(1-\epsilon)}\right)\times\big[\log(\frac{2}{\delta})+K_n.deg(T).k.\log(n.k)\big]$$
$$=\frac{1}{\epsilon}.K_n\times\big[3.wid(T)+deg(T).k.\log(n.k).\frac{1}{1-\frac{k}{k-1}q}.(\frac{2}{3}+\frac{1}{2(1-\epsilon)})\big]+$$
$$\frac{1}{\epsilon.\big(1-\frac{k}{k-1}q\big)}\left(\frac{2}{3}+\frac{1}{2(1-\epsilon)}\right)\log(\frac{2}{\delta})$$
$$\leq\frac{1}{\epsilon}.K_n\times\big[3.wid(T)+deg(T).k.\log(n.k).\frac{1}{1-\frac{k}{k-1}q}.(\frac{2}{3}+\frac{1}{2(1-\epsilon)})\big]+$$
$$\frac{1}{\epsilon.\big(1-\frac{k}{k-1}q\big)}\left(\frac{2}{3}+\frac{1}{2(1-\epsilon)}\right)\log(\frac{2}{\delta})$$
$$\leq\frac{1}{\epsilon}.\left[2^{wid^*(W)+2}\sum_{W\in\mathcal{W}}p^{\lceil\frac{mincut^*(W)}{2}\rceil}+6deg_E^*(T)\max_{W\in\mathcal{W}}|E(W^*)|\log(\frac{2}{\delta})\right]$$
$$\times\big[3.wid(T)+deg(T).k.\log(n.k).\frac{1}{1-\frac{k}{k-1}q}.(\frac{2}{3}+\frac{1}{2(1-\epsilon)})\big]+$$
$$\frac{1}{\epsilon.\big(1-\frac{k}{k-1}q\big)}\left(\frac{2}{3}+\frac{1}{2(1-\epsilon)}\right)\log(\frac{2}{\delta})$$

so we have

$$\sum_{W\in\mathcal{W}}\sum_{v\in W}\mathbb{1}\big\{\hat{\pi}_W(\tilde{Y}_v^W)\neq Y_v\big\}$$
$$\leq O\bigg(\frac{1}{\epsilon^2}.\bigg[2^{wid^*(W)+2}\sum_{W\in\mathcal{W}}p^{\lceil\frac{mincut^*(W)}{2}\rceil}+6deg_E^*(T)\max_{W\in\mathcal{W}}|E(W^*)|\log(\frac{2}{\delta})\bigg]\times$$
$$\big[3.wid(T)+deg(T).k.\log(n.k)\big]\bigg)$$

because mincut $\geq$ maximum degree

$$\leq \tilde{O}\big(k.\log k.p^{\lceil\frac{\Delta}{2}\rceil}.n\big)$$

As $\hat{\pi}_W(\tilde{Y}_v)=\hat{Y}_v$, so the algorithm ensures Hamming error has driven upper bound. □

3.4.4 Classical Result on Correlation Clustering Approximation

On correlation clustering approximation, we have following Theorem,

Theorem 3.4.3. *[80] There is a polynomial time factor* 0.878 *approximation algorithm for* MAXAGREE[2] *on general graphs. For every* $k\geq 3$*, there is a polynomial time factor* 0.7666 *approximation algorithm for* MAXAGREE[K] *on general graphs.*

With this assumption in the worse case, we have labeling with 0.7666OPT[K]. If OPT $=|E|-b$ which b is the number of bad edges that the optimal does not cover. We know the original graph is a k cluster with no bad-cycle (a cycle with one negative edge), so whatever bad edges that we see are the result of the noise process on the edges, so $b\leq|E|p$ because part of them do not generate bad-cycles. We can consider the approximate process as an extra source to generate more bad edges so we have $|E|-b'\geq$ APPROX$[k]=0.7666$OPT$[k]$. Also, by our assumption we have $p\leq p'$ so $b\leq b'$

$$|E|-b'\geq\text{APPROX}[k]=0.7666\text{OPT}[k]=0.7666\big(|E|-b\big)\rightarrow b'\leq 0.2334|E|+0.7666b$$

So we have

$$b\leq b'\leq 0.2334|E|+0.7666b$$

We have upper bound for the error introduced by our approximation and we assume all that noise come from edge noise process and the correlation clustering could not correct it, we can assume a noise process with p' such that $b' = |E|p'$ so :

$$|E|p' = b' \leq 0.2334|E| + 0.7666b \leq 0.2334|E| + 0.7666|E|p \rightarrow p' \leq 0.2334 + 0.7666p$$

So we consider exact correlation clustering result in our analyses and if we interested to see the effect of approximation algorithm on the result and get an error bound, we update p to $0.2334 + 0.7666p$ as worst case analysis which means we directly inject the approximation noise error to the results. This assumption is weak because part of b' can be captured by the local and global optimizer which we neglect it.

3.5 Mixture of Edges and Nodes Information

In all previous works [77, 83, 155], the algorithms consider the information of edge and node labels in different stages. For instance in [83], first solves the problem based on the edge because $p < q$, then it uses the nodes information. The information value of positive and negative edges in binary cases are same, but this courtesy breaks under categorical labels, on the other hand, we can use some properties in the graph to trust more on some information. We can calculate the probability of correctness of graph nodes and edges label using p and q. In categorical labeling, the space of noise has some variations from the binary case, so we have the following facts in the categorical case:

- Flipping an edge makes an error.
- Switching the label of a node might not make an error.

Using Bayes rule and the property of nodes, we have $Pr(v = i|v' = j) = Pr(v' = j|v = i))$, the prim for a vertex shows the vertex after effecting noise.

We have following theorem,

Theorem 3.5.1. *The likelihood of correctness of an edge $e = (v_i, v_j) \in E$ with label with L are*

as follow,

$$Pr(L \textit{ is untouched } |e, L) = \\ c_L \times \begin{cases} 2(1-q)q + (\frac{q}{k-1})^2.\frac{k-2}{k.(k-1)} & L = 1, \textit{vio} \\ (1-q)^2 + (\frac{q}{k-1})^2.\frac{1}{k.(k-1)} & L = 1, \textit{nvio} \\ 2(1-q).\frac{q}{k-1} + (\frac{q}{k-1})^2.\frac{k-2}{k(k-1)} & L = -1, \textit{vio} \\ (1-q)^2 + (\frac{q}{k-1})^2.\frac{k-2}{k} & L = -1, \textit{nvio} \end{cases}$$

which $c_L = \frac{(1-p)|E|}{\#L \textit{ in graph}}$*, vio means* $\varphi(X_i, X_j) \neq X_{ij}$*, and nvio means* $\varphi(X_i, X_j) = X_{ij}$*.*

Proof. In all cases, two head nodes of a given edge are v_i and v_j, and L shows the label of the edge. We first calculate the probability $Pr(v_i, v_j, L|L \text{ is untouched})$ the using Bayes theorem, we derive the likelihood.

- The first case is e generates a violation $\varphi(Z_i, Z_j) \neq X_{ij}$, and the edge label $L = 1$, in this case, the probability of the event is only one of the node labels are changed or both node labels have been changed but to the different labels.

$$\begin{aligned} &Pr(\text{only one of the node labels are changed}) = \\ &2Pr(v_i \text{ is changed}) = \\ &2(1-q). \sum_{v_i.label = j \wedge j \neq X_i} Pr(v_i.label = j | v_i.label = i) \\ &= 2(1-q). \sum_{v_i.label = j \wedge j \neq X_i} \frac{q}{k-1} = 2.(1-q)q \end{aligned}$$

and also we have, (v'_i and v'_j are the label of given nodes after noise effect)

$$\begin{aligned} &Pr(v'_i \neq v'_j \wedge v_i = v_j \wedge v'_j \neq v_j \wedge v'_i \neq v_i) \\ &= Pr(v_i \neq v'_i).Pr(v_j \neq v'_j).Pr(v_i = v_j) \times Pr(v'_i \neq v'_j | v_i = v_j \wedge v'_j \neq v_j \wedge v'_i \neq v_i) \\ &= \frac{q}{k-1}.\frac{q}{k-1}.\frac{1}{k}.\frac{(k-1)(k-2)}{(k-1).(k-1)} \\ &= (\frac{q}{k-1})^2.\frac{k-2}{k.(k-1)} \end{aligned}$$

Because $Pr(v_i \neq v'_i)$, $Pr(v_j \neq v'_j)$, and $Pr(v_i = v_j)$ are independent, so the whole probability would be $2.(1-q)q + (\frac{q}{k-1})^2.\frac{k-2}{k.(k-1)}$.

- The second case is e does not generate any violation, $\varphi(Z_i, Z_j) = X_{ij}$, and the edge label $L = 1$, in this case, either both node labels are untouched or they changed but to the same label.

$$Pr(\text{both node labels are untouched}) =$$
$$Pr(v_i = v_i').Pr(v_j = v_j') = (1-q)(1-q) = (1-q)^2$$

and also we have,

$$\begin{aligned}
&Pr(v_i' = v_j' \wedge v_i \neq v_i' \wedge v_j \neq v_j' \wedge v_i = v_j) \\
&= Pr(v_i \neq v_i').Pr(v_j \neq v_j').Pr(v_i = v_j) \times Pr(v_i' = v_j' | v_i \neq v_i' \wedge v_j \neq v_j' \wedge v_i = v_j) \\
&= \frac{q}{k-1}.\frac{q}{k-1}.\frac{1}{k}.\frac{(k-1)(1)}{(k-1).(k-1)} \\
&= (\frac{q}{k-1})^2.\frac{1}{k.(k-1)}
\end{aligned}$$

so the whole probability would be $(1-q)^2 + (\frac{q}{k-1})^2.\frac{1}{k.(k-1)}$.

- The third case is e generates a violation $\varphi(Z_i, Z_j) \neq X_{ij}$, and the edge label $L = -1$, in this case, the probability of the event is either one label change to the same label of other head or both change to the same label

$$\begin{aligned}
&Pr(\text{a label change to the same of other head}) \\
&= 2Pr(v_i \text{ is changed to } X_j) = 2(1-q).\frac{q}{k-1}
\end{aligned}$$

and also we have,

$$\begin{aligned}
&Pr(v_i' = v_j' \wedge v_i \neq v_i' \wedge v_j \neq v_j' \wedge v_i \neq v_j) \\
&= Pr(v_i \neq v_i').Pr(v_j \neq v_j').Pr(v_i \neq v_j) \times Pr(v_i' = v_j' | v_i \neq v_i' \wedge v_j \neq v_j' \wedge v_i \neq v_j) \\
&= \frac{q}{k-1}.\frac{q}{k-1}.\frac{k-1}{k}.\frac{(k-2)(1)}{(k-1).(k-1)} \\
&= (\frac{q}{k-1})^2.\frac{k-2}{k(k-1)}
\end{aligned}$$

so the whole probability would be $2(1-q).\frac{q}{k-1} + (\frac{q}{k-1})^2.\frac{k-2}{k(k-1)}$.

- The fourth case is e does not generate any violation, $\varphi(Z_i, Z_j) = X_{ij}$, and the edge label $L = -1$, in this case, either both node labels are untouched or they changed but to different

labels.

$$\begin{aligned}&Pr(\text{both node labels are untouched})\\&= Pr(v_i = v_i').Pr(v_j = v_j')\\&= (1-q)(1-q) = (1-q)^2\end{aligned}$$

and also we have,

$$\begin{aligned}&Pr(v_i' \neq v_j' \wedge v_i \neq v_i' \wedge v_j \neq v_j' \wedge v_i \neq v_j)\\&= Pr(v_i \neq v_i').Pr(v_j \neq v_j').Pr(v_i \neq v_j) \times Pr(v_i' \neq v_j' | v_i \neq v_i' \wedge v_j \neq v_j' \wedge v_i \neq v_j)\\&= \frac{q}{k-1}.\frac{q}{k-1}.\frac{k-1}{k}.\frac{(k-1)(k-2)}{(k-1).(k-1)}\\&= (\frac{q}{k-1})^2.\frac{k-2}{k}\end{aligned}$$

so the whole probability would be $(1-q)^2 + (\frac{q}{k-1})^2.\frac{k-2}{k}$.

Based on the Bayes theorem we have,

$$Pr(L \text{ is untouched} | v_i, v_j, L) = \frac{Pr(v_i, v_j, L | L \text{ is untouched}).Pr(L \text{ is untouched})}{Pr(v_i, v_j, L)}$$

We have $Pr(v_i, v_j, L) = \frac{\#L \text{ in graph}}{|E|}$, and $Pr(L \text{ is untouched}) = 1 - p$, so we can derive the result. □

As it can be seen with $k = 2$, the trust score for positive and negative are only depend to their frequencies, and if their frequencies are equal we can trust them equally.

Example 3.5.1. *(Uniform Frequencies) Let $\#\{L = +1\} \simeq \#\{L = -1\}$ and $k \geq 3$, then the second part of is negligible because of $(\frac{q}{k-1})^2$ parameter, then if $2(1-q)q \leq (1-q)^2$ and $2(1-q).\frac{q}{k-1} \leq (1-q)^2$ which is $q < \min\{\frac{1}{3}, \frac{k-1}{k+1}\} = \frac{1}{3}$ then the non-violating edges are more reliable.*

The following example is more related to the grid graphs that considered in [83].

Example 3.5.2. *(Image Segmentation) The case $k \geq 3$ and $\#\{L = +1\} \geq \#\{L = -1\}$, which we usually see in the images, because the negative edges are on the boundary of regions. If $q < 1/3$, We have can trust more on the non-violating negative edges than non-violating positive edges.*

To the best of our knowledge, no algorithm considers the mixture of edges and nodes information on the categorical data. Therefore, Theorem 3.5.1 can be a guide to design such an algorithm.

3.6 Experiments

Experimental Setup We evaluate our approach on trees and grid graphs. For trees, we use Erdős–Rényi random trees to obtain ground truth instances. For grids, we use real images to obtain the ground truth. We create noisy observations via a uniform noise model. We compare our approach with two approximate inference baselines: (1) a Majority Vote algorithm, where we leverage the neighborhood of a node to predict its label, and (2) (Loopy) Belief Propagation. To evaluate performance we use the normalized Hamming distance $\sum_{v \in V} \mathbb{1}(Y_v \neq \hat{Y}_v)/|V|$.

Hamming Error of Random Trees Our analysis suggests that Linear Program 3.1 yields a solution with Hamming error $\tilde{O}(\log(k)np)$. We evaluate experimentally that the Hamming error increases at a logarithmic rate with respect to k. Figure 3.3 shows the Hamming error for a fixed tree generative model with $p = 0.1$ and $q = 0.2$ as we increase the number of labels k. We fix q away from 0.5 and generate $10,000$ trees for each k. We report the average error. As shown, we observe the expected logarithmic behavior that we proved theoretically. The graph size is chosen randomly $n \in [10^3, 1.5 \times 10^3]$.

Trees Generation Process: We generate random trees, and we apply the noise to the generated graph. We need to have at least one example of each k labels, so the generation process starts by creating k nodes, one example for each category. Then, it generates k random numbers $n_1, \ldots, n_k$ such that $\sum_{i=1}^{k} n_i = n - k$. Next, it creates tree edges for the set of nodes V. Let S and E be empty sets. We select two nodes v and u randomly from V and add (u, v) to E such that the label of the edge satisfies the label of u and v and set $S = S \cup \{u, v\}$, and $V = V \backslash \{u, v\}$. Now, we select one node $v \in S$ and one node $u \in V$ randomly and add (u, v) to E such that the edge label satisfies the endpoints and remove u from V and add it to S. We repeat until V is empty. This process follows the Brooks theorem [21]. Finally, we apply uniform noise model with probabilities of p and q. We select this simple generative process because it covers an extensive range of random trees.

Grids Graph Generation: We use gray scale images as the source of grid graphs. The range of pixel values in gray scale images is $r = [0, 255]$, so we have that $0 \leq k \leq 255$. We divide r to k equal ranges $\{r_1, r_2, \ldots, r_k\}$. We map all pixels whose values are in r_i to $median(r_i)$. For edges, we only consider horizontal and vertical pixels and assign the ground truth edge labels based on the end points. We generate noisy node and edge observations using the uniform noise model. We use [89] dataset to select gray-scale images.

Baseline Method: A Majority Vote Algorithm: For each node $v \in G$ assign $f_v = [s_1, s_2, \ldots, s_k]$ with $s_i = 0 : \forall i \in [k]$. Let $label(.)$ shows the label of the passed node. Then, for nodes in neighbourhood of v, $u \in N(v)$, we update f_v with $s_{label(u)} = s_{label(u)} + X_{uv}$. At the end, for each node v, $\hat{Y}_v = \arg\max_{i \in [k]}(f_v)$ if $|\max(f_v)| = 1$ otherwise if $Z_v \in \arg\max(f_v)$, then $\hat{Y}_v = Z_v$ else $\hat{Y}_v = random(\arg\max(f_v))$. This is a simple baseline. We use it as we want to validate that our methods considerably outperform simple baselines.

Evaluation Metric: We use the normalized Hamming distance $\sum_{v \in V} \mathbb{1}(Y_v \neq \hat{Y}_v)/|V|$. between an estimated labeling $\hat{Y}$ and the ground truth labeling Y.

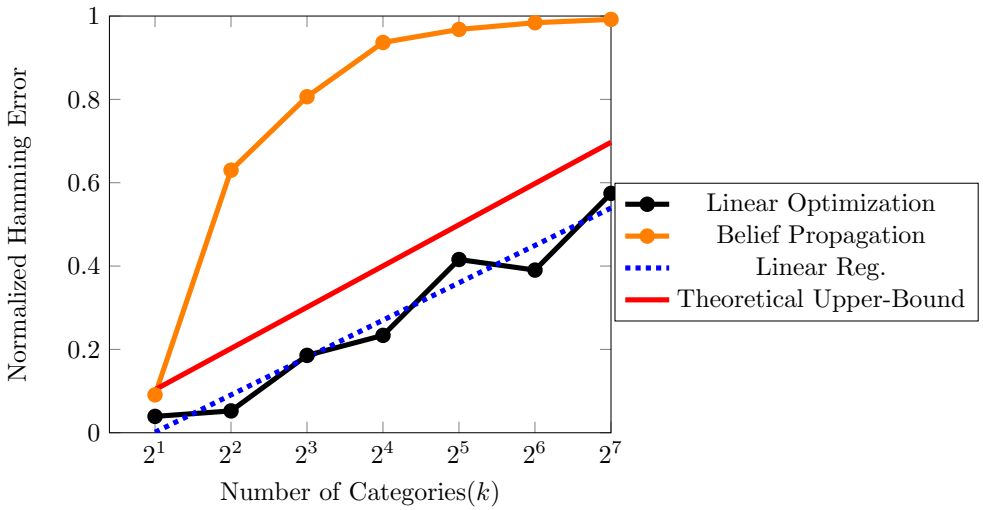


Figure 3.3: Experimental validation that Hamming error for trees increases with a logarithmic rate w.r.t. k.

Hamming Error of Grids We have two experiments on grids. In the first experiment, we select $1,000$ grayscale images and compute the Hamming error obtained by our algorithm. We consider a uniform noise model with $p = 0.05$ and $q = 0.1$. Figure 3.4 shows the Hamming error as k increases. As expected we see that the Hamming error increases. This is because as k increases negative edges carry lower information, and with non-zero edge error (p), the positive edges also provide low information observations (i.e., a wrong measurement).

In the second experiment, we evaluate the effect of edge noise p on the quality of solution obtained by our methods for a fixed number of labels k and fixed node noise q. In Figure 3.5,

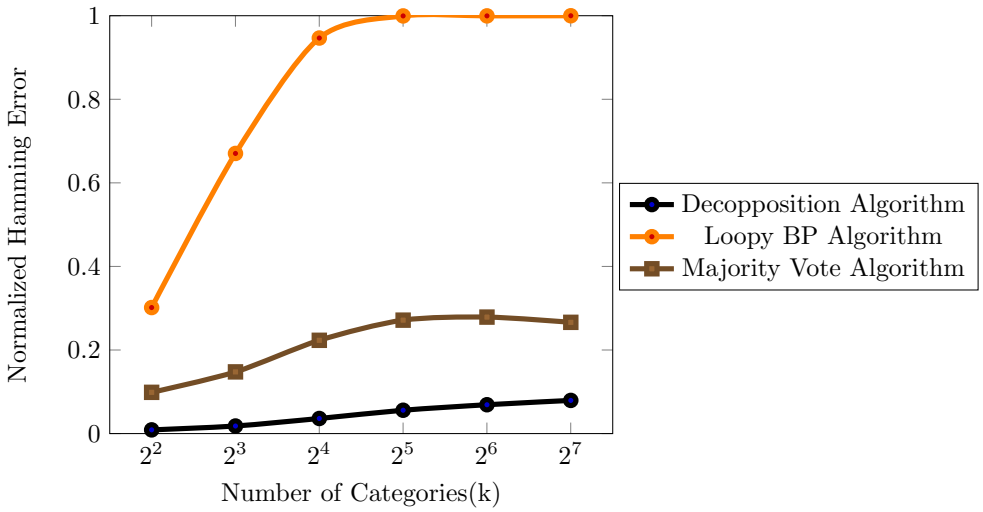


Figure 3.4: The Hamming error for different methods on grids. We show mean the mean error of $1,000$ repetitions.

we show the effect of p on the average of Hamming error when other parameters are fixed ($n = 6 \times 10^4, k = 128, q = 0.1$). We vary p from zero to 0.5. We repeat each experiment 100 times. We find that our approximate inference algorithm is robust to small amounts of noise.

This experiment also validates Theorem 3.4.1 which states when the side information from edges X helps with statistical recovery. For the setups we consider in this experiment, we have k = 128 and vary q in 0.1, 0.15, 0.2. If we keep the initial node labels the expected normalized Hamming error will be 0.1, 0.15, and 0.2 respectively. Theorem 3.4.1 states that to obtain a better Hamming error than the above one, the edge noise p has to be less than $\sqrt{0.1/(128 \log 128)} \sim 0.04$, $\sqrt{0.15/(128 \log 128)} \sim 0.05$, $\sqrt{0.2/(128 \log 128)} \sim 0.06$ respectively. Figure 3.5 shows that the normalized Hamming error obtained by our algorithm reaches the Hamming error of the trivial algorithm (and plateaus around it) at the expected edge-noise levels of 0.04, 0.05, and 0.06.

Our approximate inference algorithm is robust to small amounts of noise. As expected, when the noise increases the Hamming error increases.

3.6.1 Experiments on Grids

Figure 3.6 presents a qualitative view of the results obtained by our method (and the majority vote baseline) as k increases on the grey scale images. We see that using only the edge information (edge-based prediction) becomes more chaotic for larger values of k. This is because

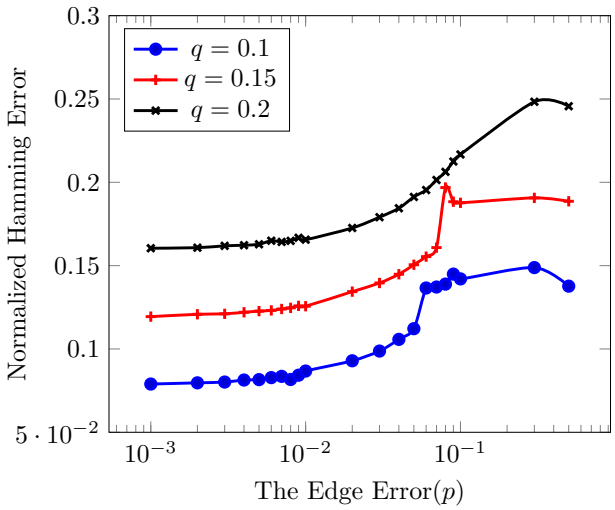


Figure 3.5: The effect of varying p on the average of normalized Hamming error(Hd) with fixed q.

the information that edges carry decreases. However, we see that combining the information provided by both node and edge observations allows us to recover the noisy image. As expected, the simple Majority vote baseline yields worse results than our method.

Figure 3.6: At each column, different stages of the inference process on the image that generates median error can be seen. It starts with generating k value image, adding noise following the model, generates best edge based prediction, and minimize it with noisy ground truth; we also report its corresponding error, you can also see the result and its error from majority algorithm.

Chapter 4

Mining Approximate Denial Constraints

4.1 Introduction

Integrity constraints are used for stating semantic conditions that the data in the database must comply with. Enforcing the constraints helps to make the database a more accurate model of the real world. Integrity constraints may be obtained by domain experts; however, this is often an expensive task that requires expertise not only in the domain but also in the constraint language. In the past two decades, extensive effort has been invested in exploring the challenge of automatically discovering constraints from the data itself, for different types of constraints, including the classic Functional Dependencies (FDs) [73, 96, 106, 135, 140, 152, 158, 203], the more general Conditional FDs (CFDs) [34, 69, 168], and the more general Denial Constraints (DCs) [15, 40, 160, 161].

In practice, databases nowadays are often inconsistent and violate the integrity constraints that are supposed to hold. In most large enterprises, information is obtained from imprecise and sometimes contradicting sources (e.g., social networks, news feeds, and user behavior data) via imprecise procedures (e.g., natural-language processing and image processing) In such cases, mining constraints that are satisfied by the entire database will be inadequate, as they rely on the assumption that all data values are correct. Hence, in this work, we consider the problem of mining *approximate constraints*, that is, constraints that are "almost" satisfied. Approximate constraints are useful even for accurate datasets, since they avoid overfitting to the current observations, and allow us to detect more general and less contrived rules, as well as rules that are generally correct but may have a few exceptions (which is useful, for example, for the task of detecting outliers).

Example 4.1.1. *Consider the database of Table 4.1 storing information about the yearly income and tax payments of people from different states in the US. We assume that as a general rule, for a given state, it holds that a higher yearly income implies higher tax payments. However, the database does not satisfy this constraint (e.g., tuples t_6 and t_7 jointly violate the constraint, and the same holds for tuples t_{14} and t_{15}). If we consider constraints that are satisfied by the entire database, these violations require us to add additional conditions to the constraint, such as "the constraint holds only for two people who have the same name" or "the constraint holds only if none of the people is called Julia and none of them lives in Illinois", which results in very specific and complicated rules. However, we will be able to find the correct constraint if we allow for exceptions, and consider approximate constraints.*

Most of the work to date on approximate constraint discovery has focused on approximate FDs [44,106,133] or CFDs [34,69,168]. Chu et al. [40] and later Pena et al. [160,161] considered approximate DCs. As the expressive power of (C)FDs is rather restricted, in this work, we consider the problem of mining approximate DCs (ADCs for short) from data. This problem has not received much attention and the currently existing algorithms are AFASTDC [40] and its improved versions BFASTDC [161] and DCFinder [160], that we will discuss in more details in the next section.

A common shortcoming of many works on approximate constraints (including the existing works on ADCs) is that the algorithms proposed for this task are often an after-thought of detecting valid exact constraints, and are usually obtained by relaxing some of the parameters of the original algorithm. Hence, existing algorithms miss opportunities to use techniques that are designed specifically for mining approximate constraints. These existing algorithms are often inefficient, since they need to examine "all" combinations of records necessary to validate the discovered DCs. Another drawback of existing algorithms is the fact that the approximation function is hard-wired into the algorithm. However, there are many possible definitions of approximate constraints, and different works indeed consider different definitions that produce very different results. The most common definition of approximate (C)FDs, for example, is based on the minimal number of tuples that should be removed for the (C)FD to hold [34, 44, 106, 133], while the definition used for approximate DCs is based on the number of tuple pairs violating the DC [40, 160, 161]. It is not clear whether one of the definitions is the "best" one, and it may be the case that different definitions produce better results in different cases.

Example 4.1.2. *Consider again the database of Table 4.1 and the DC of Example 4.1.1 (i.e., $\varphi_1 = \forall t, t' \neg(t[\mathsf{State}] = t'[\mathsf{State}] \wedge t[\mathsf{Income}] > t'[\mathsf{Income}] \wedge t[\mathsf{Tax}] \leq t'[\mathsf{Tax}])$). Two out of two hundred and ten pairs of tuples (i.e., 0.95%) violate this DC (note that $\langle t, t' \rangle$ and $\langle t', t \rangle$ are considered separately). The minimal number of tuples that should be removed from the database for the DC to hold is two (one of t_6, t_7 and one of t_{14}, t_{15}); that is, 13.3%. Therefore, if we allow,*

	Name	State	Zip	Income	Tax
t_1	Alice	NY	11803	28K	2.4K
t_2	Mark	NY	10102	42K	4.7K
t_3	Bob	NY	13914	93K	11.8K
t_4	Mary	NY	10437	58K	6.7K
t_5	Alice	NY	10437	26K	2.1K
t_6	Julia	WA	98112	27K	1.4K
t_7	Jimmy	WA	98112	24K	1.6K
t_8	Sam	WA	98112	49K	6.8K
t_9	Jeff	WA	98112	56K	7.8K
t_{10}	Gary	WA	98112	50K	7.2K
t_{11}	Ron	WA	98112	58K	8K
t_{12}	Jennifer	WA	98112	61K	8.5K
t_{13}	Adam	WA	98112	20K	1K
t_{14}	Tim	IL	62078	39K	5K
t_{15}	Sarah	IL	98112	54K	5K

Table 4.1: Running example.

for example, an exception rate of 5%, then φ will be an approximate DC according to the first definition, but it will not be an approximate DC according to the second one.

Now, consider the DC $\varphi_2 = \forall t, t' \neg(t[\mathsf{Zip}] = t'[\mathsf{Zip}] \wedge t[\mathsf{State}] \neq t'[\mathsf{State}])$ (i.e., it cannot be the case that the same zip code appears for two different states). Sixteen out of two hundred and ten pairs of tuples (i.e., 7.62%) violate the DC (every pair of tuples that includes t_{15} and one of $t_6, \ldots, t_{13}$). The only tuple that needs to be removed from the database for the DC to be satisfied is t_{15}; thus, it is possible to remove at most 6.67% of the tuples. In this case, if the allowed exception rate is 7%, then φ_2 is an approximate DC according to the second definition, but it is not an approximate DC according to the first one. Note that while the difference in the exception rate for these two definitions is very small here, this difference can be very significant in larger datasets.

The main objective of this work is to gain a deeper understanding of ADCs and introduce a general framework for mining ADCs that takes the semantics (i.e., the approximation function) as an input. We introduce the algorithm ADCMiner for mining ADCs from data. The algorithm consists of four main components – a predicate space generator, an evidence set constructor, an enumeration algorithm and a sampler. In summary, our main contributions in this work are as follows:

- We formally define the problem of approximate DC mining (Section 4.4), and we give a formal definition of a valid approximation function (Section 4.6) that is used to define

ADCs. To the best of our knowledge, we are the first to consider approximate constraint discovery that is not tied to a specific approximation function, but rather to a general family of approximation functions, that captures, but is not limited to, commonly used approximation functions.

- We introduce an algorithm for enumerating ADCs that takes the approximation function as input (Section 4.7). Our algorithm is a general algorithm for enumerating *minimal approximate hitting sets* that can even be used outside the scope of constraint discovery.
- For efficiency, we propose a *sampling* scheme (Section 4.8), and we address two fundamental problems: *(1)* how to estimate the number of violations of φ in D from a sample; and *(2)* how to use this estimate to deduce the right threshold (or approximation function) to be used when enumerating the ADCs from the sample. Sampling, while cannot be used to mine exact DCs, allows us to efficiently return highly accurate results (w.r.t. the approximation metric) by leveraging the nature of ADCs and avoiding the space explosion, which algorithms designed for exact valid DCs suffer from.

We experimentally evaluate our proposal (Section 4.9) and show that although it subsumes previously proposed approximation frameworks, we manage to achieve better efficiency. Our experiments also show that we can achieve high precision and recall from a relatively small sample, while reducing the time by as much as 90%.

4.2 Related Work

We now discuss the relationship between our work and past work on mining DCs from data. Chu et al. [40] have introduced the first algorithms for mining DCs and ADCs from data (FASTDC and AFASTDC, respectively). Their definition of an ADC is based on the fraction of tuple pairs violating the DC. The algorithm AFASTDC is obtained from FASTDC by modifying the base case of the algorithm; that is, they return a constraint if the fraction of tuple pairs violating it is smaller than some predefined threshold ϵ, rather than when it is zero. Their solution consists of two main parts. First, they generate a certain data structure, namely the evidence set, that we will formally define later on, and then they use the evidence set to generate all the (A)DCs. The first part has a very high computational cost, as it requires going over all tuple pairs in the database; hence, this algorithm may run for days on a database that consists of one million tuples [40].

Pena et al. [160, 161] significantly improved the running times of this part using bit-level operations, and Position List Indexes (PLIs) that minimize the number of required tuple comparisons. Their focus was on improving the efficiency of the evidence set construction, and they did

Notation	Meaning
S_φ	The set of predicates in the DC φ
$\mathcal{P}_R$	The predicate space over the relation R
$\mathbf{Sat}(t, t')$	The set of predicates satisfied by $\langle t, t' \rangle$
$\mathbf{Evi}(D)$	The evidence set of the database D

Table 4.2: Notation table.

not modify the second part of the solution (that generates the ADCs) and adopted the definition of ADCs used by Chu et al. [40]. Our work is complementary to that of Pena et al. [160, 161] as we focus on other aspects of ADC discovery. In particular, we do not propose a new method to construct the evidence set, but rather use the algorithm of Pena et al. [160] for this purpose.

Another related work is that of Bleifuß et al. [15], who introduced Hydra—an algorithm that significantly improves the running times of DC discovery by incorporating sampling to invalidate candidates. However, their algorithm only works for valid exact DCs, and, as stated by the authors, it is not clear whether and how their approach can be generalized to ADCs.

4.3 Preliminaries

We first present some basic terminology and notation that we use throughout the chapter.

By $R(A_1, \ldots, A_k)$ we denote a relation symbol R with the attributes $A_1, \ldots, A_k$. A *database* D over a relation $R(A_1, \ldots, A_k)$ is a finite set of *tuples* $(c_1, \ldots, c_k)$ where each c_i is a constant. We denote by $t[A_i]$ the value of tuple t in attribute A_i.

A *denial constraint* (DC for short) is an expression of the form $\forall x \neg(\omega(x) \wedge \psi(x))$, where x is a sequence of variables, $\omega(x)$ is a conjunction of atomic formulas and $\psi(x)$ is a conjunction of comparisons between two variables in x. Following previous works on the problem of mining DCs [15, 40, 160, 161], we limit ourselves to DCs where $\omega(x)$ is a conjunction of precisely two atomic formulas over the same relation and the comparison operators are $\mathbb{B} = \{=, \neq, >, <, \geq, \leq\}$.

Let R be a relation and let D be database over R. The *predicate space* $\mathcal{P}_R$ from which DCs can be formed consists of predicates of the form $t[A]\ \rho\ t'[B]$, where A and B are attributes of R, and ρ is a comparison operator from $\mathbb{B}$. Throughout the chapter, we will use the following notation for DCs: $\forall t, t' \neg(P_1, \ldots, P_m)$, where each P_i is a predicate from $\mathcal{P}_R$. The *complement* of a predicate $t[A]\ \rho\ t'[B]$ is the predicate $\widehat{P} = t[A]\ \widehat{\rho}\ t'[B]$, where $\widehat{\rho}$ is the complement operator of ρ (e.g., the complement operator of $>$ is $\leq$). The complement of a set $S = \{P_1, \ldots, P_m\}$ of predicates is the set of predicates $\{\widehat{P_1}, \ldots, \widehat{P_m}\}$. We denote this set by $\widehat{S}$.

For a pair $\langle t, t' \rangle$ of tuples in a database D over R, we denote by $\mathbf{Sat}(t, t')$ the set of all predicates in $\mathcal{P}_R$ satisfied by $\langle t, t' \rangle$. We denote by $\mathbf{Evi}(D)$ the set $\{\mathbf{Sat}(t, t') \mid t, t' \in D\}$, which we refer to as the *evidence set* [40]. Throughout the chapter we assume the bag semantics for $\mathbf{Evi}(D)$, as the number of occurrences of each set in $\mathbf{Evi}(D)$ is important. In practice, we store every set in $\mathbf{Evi}(D)$ once, along with its number of occurrences. We identify a DC φ with the set S_φ of its predicates. A DC states that its predicates cannot be satisfied all at the same time. That is, a DC φ is satisfied by a tuple pair $\langle t, t' \rangle$ if at least one of the predicates $P \in S_\varphi$ does not hold for $\langle t, t' \rangle$, or, equivalently, $\widehat{P} \in \mathbf{Sat}(t, t')$. A DC φ is *satisfied* by a database D (denoted by $D \models \varphi$) if it is satisfied by all pairs of tuples , and *violated* otherwise. If a DC φ is satisfied by a database D, we say that it is a *valid* DC w.r.t. D.

Example 4.3.1. *Table 4.3 contains a subset of the predicate space $\mathcal{P}_R$ over the relation of our running example. We use the operations in $\{<, \leq, >, \geq\}$ only for numeric attributes, and we only allow comparisons among attributes of the same type (i.e., two numeric or string attributes). For example, the predicate $t[\mathsf{Name}] = t[\mathsf{Income}]$ will not appear in $\mathcal{P}_R$. Among the predicates of Table 4.3, the predicate set $\mathbf{Sat}(t_2, t_5)$ of the tuples t_2 and t_5 of our running example will contain the predicates $t[\mathsf{Name}] \neq t'[\mathsf{Name}]$, $t[\mathsf{Income}] > t'[\mathsf{Income}]$, $t[\mathsf{Income}] \geq t'[\mathsf{Income}]$, and $t[\mathsf{Income}] > t'[\mathsf{Tax}]$. The set $\mathbf{Sat}(t_5, t_2)$ will also contain the first two predicates, but it will not contain the other two predicates; instead, $t[\mathsf{Income}] < t'[\mathsf{Income}]$ and $t[\mathsf{Income}] \leq t'[\mathsf{Income}]$ will appear in the set.*

In principle, our solution could be extended to more general DCs. For example, we could relax the limitation on the number of atomic formulas, which will affect mainly the size of $\mathbf{Evi}(D)$ (i.e., if we allow for k atomic formulas, then $\mathbf{Evi}(D)$ will contain a set $\mathbf{Sat}(t_1, \ldots, t_k)$ for each sequence $t_1, \ldots, t_k$ of tuples in D, and each such set will consist of more predicates, as $t_1[A] = t_2[A]$ is different than $t_2[A] = t_3[A]$). We could also consider other types of predicates, such as $t[A]\ \rho\ (k \times t'[B])$, which will increase the size of the predicate space. However, such extensions will have a significant impact on the running times, and the trade-off between more general constraints and lower running times has to be taken into account. When we focus on the DCs considered in this work, we are already able to discover many constraints that cannot be discovered using FD discovery methods. In our experiments, about 70% of the discovered constraints cannot be expressed as FDs.

4.4 Problem Definition

We start by defining a *valid approximation function*. Let D be a database, and let φ be a DC. Let f be a function $f : (D, S_\varphi) \to [0, 1]$. We now define two properties of such a function f, namely,

$t[\mathsf{Name}] = t'[\mathsf{Name}]$	$t[\mathsf{Name}] \neq t'[\mathsf{Name}]$
$t[\mathsf{Income}] = t'[\mathsf{Income}]$	$t[\mathsf{Income}] \neq t'[\mathsf{Income}]$
$t[\mathsf{Income}] > t'[\mathsf{Income}]$	$t[\mathsf{Income}] \geq t'[\mathsf{Income}]$
$t[\mathsf{Income}] < t'[\mathsf{Income}]$	$t[\mathsf{Income}] \leq t'[\mathsf{Income}]$
$t[\mathsf{Income}] > t'[\mathsf{Tax}]$	$t[\mathsf{Income}] \geq t'[\mathsf{Tax}]$
$t[\mathsf{Income}] < t'[\mathsf{Tax}]$	$t[\mathsf{Income}] \leq t'[\mathsf{Tax}]$

Table 4.3: A sample of the predicate space of our example.

Monotonicity and *Indifference to Redundancy*.

Definition 4.4.1 (Monotonicity)**.** *A function* $f : (D, S_\varphi) \rightarrow [0, 1]$ *is monotonic if it holds that* $f(D, S_\varphi) \leq f(D, S_{\varphi'})$ *whenever* $S_\varphi \subset S_{\varphi'}$. □

Intuitively, monotonicity ensures that the more predicates a DC contains, the higher its score is, as the number of tuple pairs that satisfy the DC can only increase. Monotonicity allows us to consider only minimal ADCs (i.e., ADCs that do not strictly contain any ADC), as it assures that whenever φ is an ADC, every φ' such that $S_\varphi \subset S_{\varphi'}$ is also an ADC. Hence, when returning only minimal ADCs φ, we also implicitly provide the user with information on any φ' that can be obtained from φ by adding more predicates. For non-monotonic functions, on the other hand, it may be the case, for example, that for φ, φ' and φ'' such that $S_\varphi \subset S'_\varphi \subset S''_\varphi$, the DCs φ and φ'' are ADCs, while φ' is not. Thus, returning only φ will result is the loss of valuable information (that is, the fact that φ' is not an ADC), and it will be necessary to go over the entire space of possible ADCs to make sure that we return all of them.

Definition 4.4.2 (Indifference to Redundancy)**.** *A function* $f : (D, S_\varphi) \rightarrow [0, 1]$ *is indifferent to redundancy if we have that* $f(D, S_\varphi) = f(D, S'_\varphi)$ *whenever* $S_\varphi \subset S_{\varphi'}$ *and* $\{\langle t, t' \rangle \mid t, t' \in D, \{t, t'\} \models \varphi\} = \{\langle t, t' \rangle \mid t, t' \in D, \{t, t'\} \models \varphi'\}$. □

A function f is indifferent to redundancy if adding more predicates to a DC φ without affecting the coverage, does not affect the score; that is, if two DCs φ and φ' such that $S_\varphi \subset S_{\varphi'}$ are satisfied by the exact same tuple pairs, then f gives them the same score. While our algorithm for enumerating minimal ADCs could work for functions that do not satisfy indifference to redundancy, having this property allows us to significantly increase the algorithm efficiency by pruning the search tree early, as we explain in Section 4.7.

We now define valid approximation functions.

Definition 4.4.3 (Valid Approximation Function)**.** *A function* $f : (D, S_\varphi) \rightarrow [0, 1]$ *is a valid approximation function if it satisfies monotonicity and indifference to redundancy.* □

In the next section, we will show that this definition is quite general and captures commonly used approximation functions. Next, we give the formal definition of a minimal ADC.

Definition 4.4.4 (Approximate Denial Constraint)**.** *Let D be a database, let f be a valid approximation function, and let $\epsilon \geq 0$. Then, a DC φ is a minimal ADC if:*

1. $1 - f(D, S_\varphi) \leq \epsilon$, and
2. *no DC φ' s.t. $S_{\varphi'} \subset S_\varphi$ satisfies $1 - f(D, S_{\varphi'}) \leq \epsilon$.* □

The intuition behind using valid approximation functions (i.e., combining the two properties) when considering ADCs is illustrated in the following example.

Example 4.4.1. *Consider the following DCs:*

$$\varphi = \forall t, t' \neg(t[A] < t'[A] \wedge t[A] \leq t'[A])$$
$$\varphi' = \forall t, t' \neg(t[A] < t'[A])$$

The DC φ' is satisfied by the exact same pairs of tuples from D as φ, since whenever a tuple pair satisfies the predicate $t[A] < t'[A]$ it also satisfies $t[A] \leq t'[A]$. Intuitively, the DC φ' is minimal, while φ is not minimal, as there is no benefit in adding the predicate $t[A] \leq t'[A]$ to the DC. For a monotonic function f, it will hold that $f(D, S_{\varphi'}) \leq f(D, S_\varphi)$; however, it may be the case that $1 - f(D, S_\varphi) \leq \epsilon$, while $1 - f(D, S'_\varphi) > \epsilon$, in which case we will return φ and not φ'. The existence of the second property (i.e., indifference to redundancy) resolves this problem since, as aforementioned, the same pairs of tuples satisfy both DCs; thus, we have that $f(D, S_\varphi) = f(D, S_{\varphi'})$ and we will either return φ' (if $1 - f(D, S_{\varphi'}) \leq \epsilon$) or none of the DCs.

Finally, we define the problem that we study in this work.

Problem 4.4.1 (ADC Mining Problem)**.** *For a database D, an approximation function f, and a threshold $\epsilon \geq 0$, generate all the nontrivial minimal ADCs for D w.r.t. f and ϵ.*

Since generating ADCs from the entire database may be very time consuming for large databases, we also consider the problem of discovering ADCs from a sample.

4.5 Overview

Alg. 4 depicted Our algorithm, ADCMiner. The input to the algorithm consists of a database D over a relation R, a valid approximation function f, and an approximation threshold $\epsilon \geq 0$. The following are the four main components of the algorithm.

1. A *predicate space generator*, which builds the predicate space $\mathcal{P}_R$ for the given relation R. We use the algorithm of Chu et al. [40] for this task. The predicates in $\mathcal{P}_R$ may compare the same attribute in two different tuples (i.e., $t[A]\,\rho\,t'[A]$), two different attributes in the same tuple (i.e., $t[A]\ \rho\ t[B]$), or two different attributes in two tuples (i.e., $t[A]\ \rho\ t'[B]$). We allow comparing two attributes only if they have at least 30% common values as in [40,160]. In principle, it is possible to compare attributes with less than 30% common values; however, relaxing this requirement may also significantly increase the number of unuseful predicates (like $t_1[Age] \neq t_2[Zip]$). The experiments conducted by Chu et al. [40] have shown that requiring at least 30% common values allows us to identify many of the comparable attributes, while avoiding a significant increase in the number of meaningless predicates.
2. A *sampler*, which draws a random sample J of tuples from D. We provide a theoretical analysis of mining ADCs from a sample in Section 4.8 and experimentally evaluate the accuracy of the results obtained from a sample in Section 4.9.
3. An *evidence set generator*, which builds the evidence set from the sample J. In this work, we use an existing algorithm for constructing the evidence set [160].
4. An *enumeration algorithm*, which takes as input the sample J, the evidence set $\mathbf{Evi}(J)$, the approximation function f and the approximation threshold ϵ and enumerates all the minimal ADCs of J w.r.t. f and ϵ (cf. Section 4.7).

Note that ADCs allow exceptions by definition, and can be seen as DCs obtained from a sample, where the sample consists of the subset of tuples that jointly satisfy the DC. Hence, we are able to obtain good results from a sample, instead of using the whole database D. Our experimental evaluation shows that using a sample of $30\% - 40\%$ of the tuples, we consistently obtain results with a high F_1 score (compared to mining the whole database), while reducing the running time by as much as 90%.

Algorithm 9: An algorithm for discovering ADCs. $ADCMiner(R, D, f, \epsilon)$

$\mathcal{P}_R = \mathsf{GeneratePSpace(R)}$
$J = \mathsf{Sample}(D)$
$\mathbf{Evi}(J) = \mathsf{ConstructEvidence}(J)$
$\mathsf{ADCEnum}(J, \mathbf{Evi}(J), \mathcal{P}_R, f, \epsilon)$

4.6 Approximation Functions

In this section, we discuss three specific valid approximation functions. Kivinen et al. [120] introduced three definitions of approximate FDs, based on three different measures, which can be easily generalized to DCs. We start by discussing each one of these measures and the corresponding approximation functions.

Let D be a database and let φ be a DC. The first measure proposed by Kivinen et al. [120] (denoted by g_1) is based on the proportion of tuple pairs violating the constraint. Formally, we define the following approximation function based on this measure:

$$f_1(D, S_\varphi) = |\{\langle t, t' \rangle \mid t, t' \in D, \{t, t'\} \models \varphi\}| \, / |D|^2$$

Note that in our definition we count the pairs satisfying the constraint; hence, we have that $g_1(D, \varphi) = 1 - f(D, S_\varphi)$. Intuitively, $f_1(D, S_\varphi)$ is the probability to select a satisfying tuple pair among all pairs, assuming a uniform distribution of the violations. This measure has been used in [40] and [160, 161] to define ADCs.

The second measure in [120], denoted by g_2, is based on the proportion of "problematic" tuples (i.e., tuples that are involved in a violation of the constraint). Here, we define the following approximation function:

$$f_2(D, S_\varphi) = |\{t \mid t \in D, \nexists t' \in D, \{t, t'\} \not\models \varphi\}| \, / |D|$$

Again, we have that $g_2(D, \varphi) = 1 - f_2(D, S_\varphi)$. If we consider an inconsistent database D, it may be the case that only one tuple contains errors, but every pair of tuples that includes this tuple violates the DC φ. In this case, it holds that $f_2(D, S_\varphi) = 0$, as all the tuples appear in one violating pair. However, if we just remove this one tuple, the DC will hold. Thus, this measure may be too sensitive, and the last measure (g_3) proposed by Kivinen et al. [120], that is based on the minimal number of tuples to remove from the database for the constraint to hold,

seems to be a better fit in this case. Hence, we introduce the following approximation function.

$$f_3(D, S_\varphi) = \max_{D'} \{|D'| \mid D' \subseteq D, D' \models \varphi\} / |D|$$

That is, the value $f_3(D, S_\varphi)$ (or, equivalently, $1 - g_3(D, \varphi)$) is the size of a *cardinality repair* [139] of D (i.e., the largest subinstance of D among all those satisfying the DC). The subinstance D' considered in this function can also be seen as a Most Probable Database [88] in the framework

of tuple independent probabilistic databases. This approximation function has been used in many works on approximate (C)FDs [34, 44, 106, 133].

We now prove that the functions f_1, f_2 and f_3 satisfy both monotonicity and indifference to redundancy.

Proposition 4.6.1. *The functions f_1, f_2, and f_3 are monotonic.*

Proof. The denominator does not depend on φ in any of the three functions; hence, monotonicity only depends on the numerator. Clearly, the function f_1 is monotonic, as adding more predicates to φ can only increase the number of tuple pairs that satisfy the DC. For that same reason, the number of tuples $t \in D$ for which we have that for every $t' \in D$ both $\langle t, t' \rangle$ and $\langle t', t \rangle$ satisfy φ can only increase, and the function f_2 is also monotonic. Finally, we prove that f_3 is monotonic. Let D' be a subinstance of D such that $D' \models \varphi$ and there is no other subinstance D'' of D that also satisfies this property such that $|D''| > |D'|$. Clearly, for each φ' such that $S_\varphi \subseteq S_{\varphi'}$ it holds that $D' \models \varphi'$ as well. Thus, D' also satisfies the condition in the numerator of f_3 for φ' (although D' is not necessarily maximal in this case), and the value $f_3(D, S_{\varphi'})$ cannot be lower than $f_3(D, S_\varphi)$. □

Proposition 4.6.2. *The functions f_1, f_2, and f_3 are indifferent to redundancy.*

Proof. The fact that this property is satisfied by f_1 and f_2 is rather straightforward. If the same tuple pairs satisfy both φ and φ', then clearly the function f_1 that counts such pairs assigns the same value to both DCs. This also implies that the tuples involved in violations of both DCs are exactly the same, which means that $f_2(D, S_\varphi) = f_2(D, S_{\varphi'})$ as well. To prove indifference to redundancy for f_3, we will show that every subinstance D' of D satisfies φ if and only it satisfies φ'. This holds since every subinstance D' satisfying one of these DCs does not contain any pair of tuples from D that jointly violate the DC, and since the exact same pairs of tuples from D violate both DCs, it means that it does not contain any tuple pair violating the other DC. □

We also prove the following result regarding the relationships between the functions f_2, f_3 and the function f_1. As will be seen in the next section, throughout the algorithm we always keep track of the sets in $\mathbf{Evi}(D)$ that have an empty intersection with $\widehat{S_\varphi}$; hence, we can compute the function f_1 faster than computing f_2 or f_3. The next proposition allows us to reduce the number of times we are required to compute f_2 or f_3 using the function f_1.

Proposition 4.6.3. *Let D be a database, φ a DC, and $\epsilon \geq 0$. For $i \in \{2, 3\}$, if $1 - f_i(D, S_\varphi) \leq \epsilon$ then $1 - f_1(D, S_\varphi) \leq 2\epsilon$.*

Proof. The evidence set $\mathbf{Evi}(D)$ contains $2(|D| - 1)$ sets for every tuple $t \in D$ (two sets, $\mathbf{Sat}(t, t')$ and $\mathbf{Sat}(t', t)$, for every tuple $t' \in D$). If $1 - f_2(D, S_\varphi) \leq \epsilon$, then at most $\epsilon|D|$ tuples appear in a violating pair. Thus, the number of violating pairs is at most $2\epsilon|D|(|D|-1)$, which is exactly 2ϵ of the tuple pairs. We conclude that $1 - f_1(D, S_\varphi) \leq 2\epsilon$. As for the function f_3, when we remove a tuple from D, we remove $2(|D| - 1)$ sets from $\mathbf{Evi}(D)$. If $1 - f_3(D, S_\varphi) \leq \epsilon$, then there is a subinstance D' of D that is obtained by removing at most $\epsilon|D|$ tuples from D such that $D' \models \varphi$. This observation implies that $\mathbf{Evi}(D')$ contains every set in $\mathbf{Evi}(D)$ except for at most $2\epsilon|D|(|D| - 1)$ sets. Since D' satisfies φ, at most $2\epsilon|D|(|D| - 1)$ pairs violate φ, which is at most 2ϵ of the tuple pairs, and again we have that $1 - f_1(D, S_\varphi) \leq 2\epsilon$. □

Finally, we discuss the computational complexity of the three functions. Unlike the functions f_1 and f_2 that can be computed in polynomial time for both FDs and DCs, the function f_3 can be computed in polynomial time for FDs [138], but not for DCs. Livshits et al. [137] have shown that this problem is NP-hard even when considering simple DCs over a single relation symbol (e.g., the DC $\forall t, t' \neg(t[A] \neq t'[B])$). Hence, we cannot efficiently compute f_3. However, there is a simple reduction from the problem of computing $1 - f_3(D, S_\varphi)$ to the minimum vertex cover problem (where the goal is to find a minimal set of vertices that intersects with all the edges), based on the concept of a *conflict graph*, in which vertices represent tuples and edges represent violations. Since vertex cover is 2-approximable in polynomial time [9], this is also the case for our problem. Thus, to generate ADCs w.r.t. f_3 we could use the 2-approximation algorithm with the threshold 2ϵ. Note that we will return all ADCs, but we may also return some DCs for which it holds that $1 - f_3(D, S_\varphi) \leq 2\epsilon$ but $1 - f_3(D, S_\varphi) > \epsilon$.

Algorithm 10: A greedy algorithm replacing f_3 $D, S_\varphi, \mathbf{vios}, \epsilon$

```
(T, v) = SortTuples(D, S_φ, vios)
u = |S ∈ Evi(D) | S ∩ S_φ = ∅|
c = 0, R = ∅
while c < u do
    let t be the first tuple in T
    c = c + v(t)
    remove t from T and add it to R
end
return (|R|/|D| ≤ ε)
```

In practice, the 2-approximation algorithms for minimum vertex cover assume an explicit representation of the graph. In our case, this requires storing, for every set S in $\mathbf{Evi}(D)$, all pairs $\langle t, t' \rangle$ of tuples such that $\mathbf{Sat}(t, t') = S$. As the number of tuple pairs is quadratic in the size

Algorithm 11: $SortTuples(D, S_\varphi, \mathbf{vios})$

```
v(t) = 0 for all t ∈ D
for S ∈ Evi(D) such that S ∩ S_φ = ∅ do
    for t ∈ vios[S] do
        v(t) = v(t) + vios[S][t]
    end
end
return (tuples of D in descending order of v(t), v(t))
```

of the database, storing this information with reasonable memory usage is infeasible for large databases. Hence, in our experimental evaluation, we implement a greedy algorithm (Alg. 10) instead. This greedy algorithm is inspired by the greedy $O(\log n)$-approximation algorithm for minimum vertex cover, that, in each iteration, selects a vertex that is adjacent to the maximal number of uncovered edges, and then marks each one of these edges as covered. However, our algorithm does not require an explicit representation of the graph; hence, we do not know which edges are covered. While we do not provide any theoretical guarantees on the result of this algorithm, our experimental evaluation shows that using this algorithm we often obtain more accurate results than the ones obtained using the function f_2.

In the algorithm, we sort the tuples in descending order according to the number of violations they participate in. For that, we use the data structure $\mathbf{vios}$ that stores, for every set $S \in \mathbf{Evi}(D)$ and tuple $t \in D$, the number of violations of type S that t is involved in (that is, the number of tuple pairs $\langle t_1, t_2 \rangle$ such that $\mathbf{Sat}(t_1, t_2) = S$ and either $t_1 = t$ or $t_2 = t$). Then, we start selecting these tuples, one by one, while recording the change to the number of violations covered by the selected tuples. That is, with every tuple that we select, we add the number of violations it participates in to the number of covered violations c. We stop this process when the number of covered violations c is at least the number of total violations u. The number of covered violations can be higher than the number of total violations, as if two tuples t, t' jointly violate the DC and are both added to the result, we count this violation twice. Finally, we return the DC if the ratio between the number of tuples in the result and the total number of tuples is lower than the threshold.

The most time consuming process is Alg. 11; hence, the time complexity is $O(|D| \cdot n)$ where n is the number of *distinct* sets in $\mathbf{Evi}(D)$ (recall that we treat $\mathbf{Evi}(D)$ as a bag), and the space complexity, which depends on the size of $\mathbf{vios}$, is the same. In all of our experiments, the number of distinct sets in $\mathbf{Evi}(D)$ is orders of magnitude smaller than the number of tuple pairs; hence, storing this data structure requires significantly less space than storing data for every pair of tuples.

4.7 Enumeration Algorithm

In this section, we introduce an algorithm for enumerating minimal ADCs. Following Chu et al. [40], we reduce our problem to that of enumerating *minimal approximate hitting sets*. The hitting set problem is the following: given a finite set K and a family M of subsets of K, find all subsets of K that intersect every one of the subsets in M. A subset F is a *minimal* hitting set if no proper subset of F is a hitting set. As mentioned in the preliminaries, a pair $\langle t, t' \rangle$ of tuples satisfies a DC φ if $\widehat{P} \in \mathbf{Sat}(t, t')$ for some $P \in S_\varphi$. Hence, it is rather straightforward that φ is a valid DC if $\widehat{S_\varphi}$ is a hitting set of $\mathbf{Evi}(D)$. Note that the other direction does not necessarily hold, as a hitting set may not correspond to a nontrivial DC. For example, the set $\{t[A] = t'[A], t[A] \neq t'[A]\}$ is clearly a hitting set of $\mathbf{Evi}(D)$, but the corresponding DC is trivial. Hence, the reduction is essentially to the hitting set problem with restrictions rather than the general hitting set problem.

Although the complexity of enumerating minimal hitting sets or, equivalently, hypergraph transversals is still an open problem (after decades of research), many algorithms have been proposed for this task (see [78] for a survey). Yet, to the best of our knowledge, the problem of enumerating minimal *approximate* hitting sets has not received much attention. Here, we refer to a set $F \subseteq K$ that satisfies $1 - f(M, F) \leq \epsilon$ for a given valid approximation function f and a threshold ϵ as an approximate hitting set. Researches typically refer to one of two problems as computing approximate hitting sets: *(1)* enumerating hitting sets, but not necessarily all of them (and not necessarily minimal) [4, 26, 151], *and (2)* computing an approximate hitting set of minimum cardinality [25, 28, 196]. However, we focus on the problem of generating minimal approximate hitting sets for a given approximation function. Hence, we devise an algorithm for enumerating minimal approximate hitting sets, building upon an algorithm for enumerating minimal hitting sets by Murakami and Uno [148]. In Section 4.9, we compare the performance of our algorithm to the discovery algorithm used in [40, 160, 161], and show that even though our algorithm is more general, we are able to significantly reduce the running time.

4.7.1 Enumerating Minimal Hitting Sets

We now introduce the algorithm of Murakami and Uno [148] for enumeraing minimal hitting sets. In the next subsection, we will explain how we adapt the algorithm to the approximation problem.

The algorithm is depicted in Figure 12. The input consists of a set K of elements and a set M of subsets of K. Those are used to initialize three data structures, namely $\mathbf{uncov}$, $\mathbf{cand}$ and $\mathbf{crit}$, maintained by the algorithm. The algorithm is a recursive algorithm that builds the hitting

Algorithm 12: An algorithm for enumerating minimal hitting sets $MMCS(S, \mathbf{crit}, \mathbf{uncov}, \mathbf{cand})$ [148]

```
1  if uncov = ∅ then
2  |  output S
3  |  return
4  end
5  choose a set F from uncov
6  C = cand ∩ F
7  cand = cand \ C
8  for e ∈ C do
9  |  UpdateCritUncov(e, S, crit, uncov)[Alg. 13]
10 |  if crit[u] ≠ ∅ for each u ∈ S then
11 |  |  MMCS(S ∪ {e}, crit, uncov, cand)
12 |  |  cand = cand ∪ {e}
13 |  end
14 |  recover the changes to crit and uncov done in 8
15 end
16 recover the change to cand done in 6
```

Algorithm 13: $UpdateCritUncov(e, S, \mathbf{crit}, \mathbf{uncov})$

```
1  for F ∈ uncov do
2  |  if e ∈ F then
3  |  |  crit[e] = crit[e] ∪ {F}
4  |  |  uncov = uncov \ {F}
5  |  end
6  end
7  for u ∈ S do
8  |  for F ∈ crit[u] do
9  |  |  if e ∈ F then
10 |  |  |  crit[u] = crit[u] \ {F}
11 |  |  end
12 |  end
13 end
```

sets incrementally. It starts with an empty set S, and adds elements to S until it has a nonempty intersection with each one of the subsets in M; that is, until S is a hitting set. The data structure $\mathbf{uncov}$ stores the subsets in M that are not yet covered, that is, have an empty intersection with the intermediate S. Since we start with an empty S, initially, $\mathbf{uncov}$ contains all the subsets in M. The second data structure, $\mathbf{cand}$, stores the elements of K that can be added to S in the next iterations of the algorithm. Initially, $\mathbf{cand}$ contains every element of K. Finally, $\mathbf{crit}$ stores, for each element e in the intermediate S, all the subsets in M for which e is critical (i.e., all the subsets that contain e, but do not contain any other element of S). The importance of each one of these data structures will become clear soon.

At each iteration, the algorithm selects a subset F from $\mathbf{uncov}$. The goal is then to add at least one element of F to S, so that the two sets have a nonempty intersection. In line 5 of the algorithm, we store the intersection of F and $\mathbf{cand}$ in C. The set C thus contains all the elements of F that we are allowed to add to S. Then, every element of F is removed from $\mathbf{cand}$. Some of these elements will be added back to $\mathbf{cand}$ later on, while some are permanently removed from this list. The idea is the following. Let $\{e_1, \ldots, e_n\}$ be the set of elements in C. First, we add e_1 to S, and the other elements of C still do not belong to $\mathbf{cand}$; hence, we are able to generate minimal hitting sets that contain e_1, but do not contain any other element of C. Then, we add e_2 to S and we add e_1 to $\mathbf{cand}$ (if some condition holds, as we will explain later). Thus, we are now able to generate minimal hitting sets that contain only e_2, or contain both e_2 and e_1, but do not contain any other element of C. Then, we add e_3 to S and both e_1 and e_2 appear in the list of candidates, and so on. This allows us to avoid generating the same hitting set twice, but it also allows us to prune branches in the search tree early on, as we now explain.

Observe that a set S is a *minimal* hitting set only if *every* element of S is critical to at least one subset. Thus, after adding an element e of F to S, the UpdateCritUncov subroutine is called. This subroutine updates the data structures in the following way: *(a)* every subset in $\mathbf{uncov}$ that contains e is removed from $\mathbf{uncov}$, as it no longer has an empty intersection with S, *(b)* every subset that has been removed from $\mathbf{uncov}$ is added to the list of subsets for which e is critical, as it does not contain any other element of S, and *(c)* for every element u in S, and for every subset F that belongs to the list of subsets for which u is critical, F is removed from this list if it contains e (as it now contains other elements of S).

The purpose of calling UpdateCritUncov is twofold. First, it updates the data structures after adding a new element to S. Second, it is used to prune branches in the search tree early on. In line 9 of the algorithm, after the call to the subroutine, the algorithm tests whether for every element of S, the list of subsets for which it is critical is nonempty. Otherwise, as explained above, this branch will never result in a minimal hitting set. Hence, if the test of line 9 fails, we recover all changes to $\mathbf{crit}$ and $\mathbf{uncov}$, and move on to the next element of C in the iteration in line 7. Observe that in this case, the element e is not added back to $\mathbf{cand}$ due to the observation

that if an element is not critical for any subset w.r.t. S, then it cannot be critical for any subset w.r.t. a set S' such that $S \subseteq S'$. If, on the other hand, the test of line 9 succeeds, then we add e back to $\mathbf{cand}$; thus, it could be added to S later on. Murakami and Uno [148] proved the following about the algorithm MMCS: *(a)* it returns only minimal hitting sets, *(b)* it returns all the minimal hitting sets, and *(c)* it returns each minimal hitting set once. Moreover, they have shown that the time complexity of the algorithm is $O(\|M\|)$ per iteration, where $\|M\|$ is the sum of sizes of sets in M. The same holds for the space complexity.

4.7.2 Enumerating Approximate Hitting Sets

One may suggest to adapt the algorithm of Alg. 12 to generate minimal approximate hitting sets by modifying the base case. Instead of stopping when all the subsets have a nonempty intersection with S, we will stop when our condition for minimal approximate hitting sets holds (i.e., when $1 - f(D, S) \leq \epsilon$ for the function f and threshold ϵ). It is straightforward that this will return only minimal approximate hitting sets w.r.t. f and ϵ, but will it return all of them? The answer to this question is negative. The problem with this approach, which also applies to many other algorithms for enumerating minimal hitting sets [78], is that when we select a new subset at each iteration and try to "hit" it, we define a certain order over the subsets. An easy observation is that we will never return a set that has an empty intersection with the first chosen subset, even if it has a nonempty intersection with any other subset.

Our algorithm ADCEnum for enumerating minimal ADCs is depicted in Alg. 14. We modify the algorithm MMCS in the following way. First, we change the base case, as aforementioned; that is, we print S only if $1 - f(D, S) \leq \epsilon$. However, we also have to explicitly check for minimality before printing S. This is due to the fact that while a set S of elements where each $e \in S$ is critical for at least one subset of M is guaranteed to be minimal when considering hitting sets, this is not the case when considering approximate hitting sets, as our S is allowed to have an empty intersection with some subsets of M. Due to the indifference to redundancy property, this condition is still necessary when considering approximate hitting sets, since we can remove elements that are not critical for any subset without affecting the set of tuple pairs that have a non-empty intersection with S, and, consequently, without affecting the value of the approximation function. However, this condition is no longer sufficient. Therefore, we check whether S is minimal with IsMinimal subroutine depicted in Alg. 15. There, we go over all sets S' of elements obtained from S by removing a single element, and for each S' we check whether $1 - f(D, S') \leq \epsilon$. Recall that the approximation functions that we consider are monotonic; hence, if for a subset S' of S it holds that $1 - f(D, S') > \epsilon$, then we have that $1 - f(D, S'') > \epsilon$ for any $S'' \subset S'$, and we do not need to go over the subsets of S obtained by removing more than one element.

Algorithm 14: Enumerating minimal ADCs: $ADCEnum(S, \mathbf{crit}, \mathbf{uncov}, \mathbf{cand}, \mathbf{canHit}, f, \epsilon)$

1 **if** $1 - f(D, S) \leq \epsilon$ *and* IsMinimal(S, f, ϵ)*[Alg. 15]* **then**
2 | output DC from S
3 | **return**
4 **end**
5 choose a set $F \in \mathbf{uncov}$ s.t. $\mathbf{canHit}[F] =$ true
6 **if** *such a set F does not exist* **then**
7 | **return**
8 **end**
9 $\mathbf{cand} = \mathbf{cand} \setminus F$
10 UpdateCanCover$(\mathbf{uncov}, \mathbf{cand}, \mathbf{canHit})$
11 **if** WillCover$(S, \mathbf{cand}, f, \epsilon)$*[Alg. 17]* **then**
12 | ADCEnum$(S, \mathbf{crit}, \mathbf{uncov}, \mathbf{cand}, \mathbf{canHit})$
13 **end**
14 recover the change to $\mathbf{cand}$ done in 7
15 recover the change to $\mathbf{canHit}$ done in 8
16 $C = \mathbf{cand} \cap F$
17 $\mathbf{cand} = \mathbf{cand} \setminus C$
18 **for** $e \in C$ **do**
19 | UpdateCritUncov$(e, S, \mathbf{crit}, \mathbf{uncov})$[Alg. 13]
20 | **if** $\mathbf{crit}[u] \neq \emptyset$ *for each* $u \in S$ **then**
21 | | RemoveRedundantPreds(e, $\mathbf{cand}$)
22 | | ADCEnum$(S \cup \{e\}, \mathbf{crit}, \mathbf{uncov}, \mathbf{cand}, \mathbf{canHit})$ [Alg. 16]
23 | | $\mathbf{cand} = \mathbf{cand} \cup \{e\}$
24 | **end**
25 | recover the changes to $\mathbf{crit}$ and $\mathbf{uncov}$ done in 16
26 **end**
27 recover the change to $\mathbf{cand}$ done in 14

Algorithm 15: $IsMinimal(S, f, \epsilon)$

```
for e ∈ S do
    if 1 − f(D, S \ {e}) ≤ ε then
        return false
    end
end
return true
```

Algorithm 16: $UpdateCanCover(\mathbf{uncov}, \mathbf{cand}, \mathbf{canHit})$

```
for F ∈ uncov do
    for e ∈ cand do
        if e ∈ F then
            continue outer loop
        end
    end
    canHit[F] = false
end
```

Algorithm 17: $WillCover(S, \mathbf{cand}, f, \epsilon)$

```
S' = S ∪ cand if 1 − f(D, S') ≤ ε then
    return true
end
return false
```

Next, we choose a subset $F \in \mathbf{uncov}$ and make two recursive calls – one that "hits" the chosen F (i.e., adds an element of F to S) and one that does not. We start with the second one. Observe that our algorithm contains an additional data structure, namely $\mathbf{canHit}$. It is used for the additional recursive call and it contains a single value, true or false, for every subset of M. Initially, the value is true for all subsets. The idea is the following. Whenever we choose not to hit F, this set remains in $\mathbf{uncov}$. To avoid choosing it again in a future iteration of the algorithm (which may result in an infinite recursion), we update $\mathbf{canHit}[F] = \text{false}$ in the UpdateCanCover depicted in Alg. 16. However, F may not be the only subset in $\mathbf{uncov}$ that has an empty intersection with $\mathbf{cand}$ after removing all the elements of F from $\mathbf{cand}$ in line 7. Hence, in this subroutine, we mark every subset of M that is still in $\mathbf{uncov}$ and does not contain any element of $\mathbf{cand}$. This way, we avoid selecting these subsets in future iterations, which significantly reduces the number of unnecessary recursive calls. We make the recursive call after checking whether it can result in an approximate hitting set. We check that in Alg. 17, WillCover subroutine adds all the elements of $\mathbf{cand}$ to S and checks whether the result S' satisfies $1 - f(D, S') \leq \epsilon$. If this is not the case, the monotonicity property ensures that this branch will never result in an approximate hitting set (since we cannot increase the value of the approximation function by adding less predicates), and we do not make the recursive call.

The second recursive call (where we hit the selected F) is identical to the recursive call of the original algorithm and we do not explain it again here. Note that if we did not assume indifference to redundancy, we could not prune branches based on the $\mathbf{crit}$ data structure (line 17) as done in the original algorithm, since it could be the case that adding predicates that are not critical for any subset actually increases the value of the approximation function (while having no impact on the set of tuple pairs satisfying the DC).

While the Alg. 14 can be used as a general algorithm for enumerating minimal approximate hitting sets, there are two aspects that are specific to our setting. First, we do not return the hitting set S itself, but the DC obtained from S; Second, before making the recursive call of line 19, and after adding an element u to S,

we remove from $\mathbf{cand}$ all the predicates that differ from u only by the operator. This way, we avoid developing branches that will result in trivial DCs, such as $\forall t, t' \neg(t[A] < t'[A] \wedge t[A] \geq t'[A])$, and avoid developing some branches that will fail the minimality condition, such as $\forall t, t' \neg(t[A] < t'[A] \wedge t[A] \leq t'[A])$ (this is again based on the assumption that the approximation function is indifferent to redundancy, and the addition of the predicate $t[A] \leq t'[A]$ cannot affect the value of the approximation function on a set that already contains the predicate $t[A] < t'[A]$).

Finally, to improve the running time of the algorithm, we do not select a random set F in line 4. Murakami and Uno [148] suggested to select the set that has the minimum size intersection with the candidate list. Doing so, we minimize the number of iterations in the loop of line 15,

and decrease the number of recursive calls. The problem with this approach is that when we select such a set, we remove less predicates from the candidate list in line 7; thus, the chances of the condition of line 9 to be satisfied increase. Hence, while we decrease the number of recursive calls in line 20, we increase the number of recursive calls in line 10. In our implementation, we select the set that maximizes the intersection with the candidates list. Our experiments show that this choice decreases the running times, as the total number of recursive call decreases compared to the approach in [148].

4.7.3 Proof of Correctness

The correctness of ADCEnum is stated in the following theorem.

Theorem 4.7.1. *Let D be a database. Let f be a valid approximation function and let $\epsilon \geq 0$. Then, the following hold for* ADCEnum *w.r.t. f and ϵ: (a) it returns only minimal ADCs of D, (b) it returns all the minimal ADCs of D, and (c) it returns every minimal ADC of D once.*

Proof. The first claim is rather straightforward, as we return a set S only if $1 - f(D, S) \leq \epsilon$ and S is minimal. For the last claim, observe that the two recursive calls in each iteration cannot result in the same S (since in the first one S will always have an empty intersection with the selected F, while in the second one S will have a nonempty intersection with F). Moreover, the first recursive call does not modify S, while the second one is identical to the recursive call of the algorithm MMSC. We conclude that since MMSC returns every minimal hitting set once, our algorithm returns every minimal ADC once.

We prove by induction on n, the depth of the recursion, that
$\mathsf{ADCEnum}(S, \mathbf{crit}, \mathbf{uncov}, \mathbf{cand}, \mathbf{canHit})$ returns every minimal ADC φ that satisfies:

- $S \subseteq S_\varphi$ and $S_\varphi \subseteq (S \cup \mathbf{cand})$,
- S_φ has an empty intersection with all the sets $F \in \mathbf{Evi}(D)$ for which $\mathbf{canHit}[F] =$ false.

Note that the sets for which $\mathbf{canHit}[F] =$ true can either have an empty or a nonempty intersection with S_φ. Since at the beginning, $\mathbf{cand}$ contains all the predicates of $\mathcal{P}_R$ and we have that $\mathbf{canHit}[F] =$ true for each $F \in \mathbf{Evi}(D)$, we will conclude that
$\mathsf{ADCEnum}(\emptyset, \mathbf{crit}, \mathbf{uncov}, \mathbf{cand}, \mathbf{canHit})$ returns every ADC φ such that $\emptyset \subseteq S_\varphi$ and $S_\varphi \subseteq \mathbf{cand}$; that is, all the ADCs.

For the basis of the induction, $n = 0$, one possible case is that the condition of line 1 holds. Then, the constraint corresponding to S itself is a minimal ADC; thus, the only S_φ that contains S such that φ is a minimal ADC is S itself, and we indeed return S. Note that the sets F for which it holds that $\mathbf{canHit}[F] =$ false have an empty intersection with S, as we update the

value $\mathbf{canHit}[F]$ for a set F to false only when $\mathbf{cand}$ no longer contains any predicate of F. If the condition of line 1 does not hold, then the only other option is that the condition of line 5 holds. In this case, no S_φ that contains S is such that φ is an ADC, as it does not holds that $1 - f(D, S) \leq \epsilon$ and the remaining candidate predicates do not appear in any of the remaining sets in $\mathbf{uncov}$.

For the inductive step, we prove that if the claim holds for all $n \in \{1, \ldots, k-1\}$, it also holds for $n = k$. Let us consider an iteration from which the depth of the recursion is k. In line 4, we choose a set F for which $\mathbf{canHit}[F] = \text{true}$. Each S_φ that contains S either has a nonempty intersection with F or an empty one. In the first case, let S_φ be a minimal ADC that has a nonempty intersection with F and satisfies the two conditions. In line 14, we go over all the predicates of F and try to add each one of them to S. Clearly, S_φ contains at least one of these predicates. Let $\{p_1, \ldots, p_k\} = S_\varphi \cap F$ and assume that $p_1, \ldots, p_k$ is the order by which they are selected in line 14. We claim that S_φ is generated in the recursive call made when p_k is selected in line 14.

Each predicate of $\{p_1, \ldots, p_k\}$ is in $\mathbf{cand}$ at this point, since we assume that $S_\varphi \subseteq (S \cup \mathbf{cand})$ and none of these predicates can violate the minimality condition of line 17, as this will imply that S_φ contains an element that is not critical for any subset, which is a contradiction to the fact that S_φ is minimal (due to indifference to redundancy). Hence, every predicate in $\{p_1, \ldots, p_k\}$ is added back to $\mathbf{cand}$ in line 20, and since p_k is the last predicate selected in the loop of line 15, in that iteration all the other predicates already belong to $\mathbf{cand}$. From the inductive assumption, we know that ADCEnum generates every minimal ADC that contains $S \cup \{p_k\}$, is contained in $S \cup \{p_k\} \cup \mathbf{cand}$, and has an empty intersection with every F' for which $\mathbf{canHit}[F'] = \text{false}$, when given the set $S \cup \{p_k\}$ as input; among them is S_φ (observe that we do not change the data structure $\mathbf{canHit}$ for the recursive call of line 19).

For the second case, let S_φ be a minimal ADC that has an empty intersection with F and satisfies the two conditions. We claim that S_φ is generated in the recursive call of line 10. Here, we make a recursive call with the same S after removing all the predicates of F from $\mathbf{cand}$, and updating $\mathbf{canHit}[F']$ to false for each F' that no longer contains any predicates from $\mathbf{cand}$. From the inductive assumption, we know that this recursive call generates every minimal ADC that contains S and has an empty intersection with F (as no predicate of F appears in $\mathbf{cand}$); among them is S_φ. That concludes our proof of correctness for the algorithm. □

Finally, we discuss the complexity of ADCEnum. There are two components of the algorithm that affect the time complexity compared to the complexity of MMCS—the additional recursive call in line 10, and the computation of the function f that affects the complexity per iteration. Recall that the complexity of MMCS per iteration is $O(\|M\|)$. In our case, we have that $\|M\|$ is bounded by $|\mathcal{P}| \cdot n$, where n is the number of distinct sets in $\mathbf{Evi}(D)$. We compute the function

f in the algorithm $|S| + 2$ times, and since $|S|$ is bounded by $|\mathcal{P}|$, we conclude that the time complexity per iteration is $O\left(|\mathcal{P}| \cdot n + |\mathcal{P}| \cdot f(|\mathcal{P}|, |\mathbf{Evi}(D)|)\right)$, where $f(|\mathcal{P}|, |\mathbf{Evi}(D)|)$ is the time required to compute f. The space complexity is not affected compared to MMCS and remains $O(|\mathcal{P}| \cdot n)$.

4.8 Mining ADCs From a Sample

The input to our algorithm is the evidence set and the complexity of building it is quadratic in the size of the database (as we have to go over all pairs of tuples), which can be prohibitively expensive for large databases. In this section, we show how to use a sample from the database to produce ADCs with probabilistic guarantees, while avoiding the cost of building the evidence set for the entire database [94]. For simplicity, we limit our discussion to a simple approximation function, namely, the function f_1 introduced in Section 4.6. Recall that the function f_1 is based on the number of tuple pairs violating the DC in the database.

Let J be a sample uniformly drawn from a database D and let $\epsilon \geq 0$. Let φ be a DC. We address the following problems: *(1)* how to estimate the number of violations of φ in D from J; and *(2)* how to use this estimate to decide on the right threshold (or approximation function) to use when enumerating ADCs from J.

4.8.1 Estimating the Number of Violations

Since we consider the function f_1 that is based on the number of violations of the DC in the database, we now show how to estimate this number from a sample J uniformly drawn from D. We represent the violations of an ADC φ as a conflict graph $G(V, E)$ [36], where V is the set of vertices corresponding to the tuples in D, and E is the set of edges corresponding to violations of the DC, where an edge (t_1, t_2) exists if the pair $\langle t_1, t_2 \rangle$ violates the DC. Note that this is a directed graph since a pair $\langle t_1, t_2 \rangle$ may violate a DC that is satisfied by $\langle t_2, t_1 \rangle$. Hence, the problem that we consider here is that of estimating the *density* of a graph from a given sample.

To the best of our knowledge, most works on the density of random graphs focus on the generation of samples with density requirements [6, 20, 75, 101, 141, 183], which seems to be a harder problem. Hence, the methods proposed in these works are too robust for our problem, which reflects in the high computational complexity of the proposed solutions. In our case, the graph that we obtain is different for every DC, and we need to estimate the density for a different graph in every iteration of the algorithm; hence, using solutions with a high computational cost is infeasible. There is also a line of work that focuses on the related problem of estimating the

average degree of a graph, given the degree of some of the vertices [71, 84]; however, a basic requirement in the proposed solutions is to be able to query the actual degree of at least $O(\sqrt{|V|})$ vertices. To obtain this information, we will need to go over $O(|V| \cdot \sqrt{|V|})$ pairs of tuples in each iteration of the algorithm, which is again too expensive. Hence, we propose a simple method for estimating the graph density from a sample, that has no significant impact on the computational cost of our algorithm.

Let $p = \frac{|E|}{2 \cdot \binom{|V|}{2}}$ (that is, $p = 1 - f_1(D, S_\varphi)$). Let $G_J(V_J, E_J)$ be the conflict graph of J. To estimate p from J, we use the value $\hat{p} = \frac{|E_J|}{2 \cdot \binom{|V_J|}{2}}$. We define the random variable x_i for each pair of nodes in V_J, where $x_i = 1$ with probability p and $x_i = 0$ with probability $1-p$. It can be easily shown that $E(\hat{p}) = p$, so it is an unbiased estimator of p. Note that we do not make assumptions about the structure of the conflict graph or about the dependencies between the edges.

We further derive error bounds on our estimator to help us derive our guarantees. To compute error bounds, various methods can be used, including Chebyshev's inequality and the normal distribution assumption. Most of them require estimating the variance of our estimator. Here, we use Chebyshev's inequality:

$$\Pr(|\hat{p} - E(\hat{p})| > a) \leq \frac{1}{a^2} \cdot \mathrm{var}(\hat{p})$$

We know that $p = E(\hat{p})$; hence, we now compute an upper bound on $\mathrm{var}(\hat{p})$.

$$\begin{aligned}\mathrm{var}(\hat{p}) = \mathrm{var}\left(\frac{|E_J|}{\binom{|V_J|}{2}}\right) &= \frac{1}{\binom{|V_J|}{2}^2}\left[E(E_J^2) - E(E_J)^2\right] \\ &= \frac{1}{\binom{|V_J|}{2}^2}\left[E(E_J^2) - \binom{|V_J|}{2}^2 \cdot p^2\right]\end{aligned}$$

We now expand the term $E(E_J^2)$ using the random variables $x_1, \ldots, x_{\binom{|V_J|}{2}}$ as follows.

$$E(E_J^2) = E\left(\left(\sum_{i=1}^{\binom{|V_J|}{2}} x_i\right)^2\right) = \sum_{i=1}^{\binom{|V_J|}{2}} E(x_i^2) + \sum_{i \neq j \in \{1,\ldots,\binom{|V_J|}{2}\}} E(x_i \cdot x_j)$$

Since we do not assume anything about the dependencies between the variables x_i, we cannot calculate the exact value of $E(x_i \cdot x_j)$; however, we can derive an upper bound for this value.

We know that $x_i \cdot x_j = 1$ if and only if $x_i = x_j = 1$, and the value of $E(x_i \cdot x_j)$ depends of the number of these events. Hence, if we can find an upper bound for the probability of $x_i \cdot x_j = 1$, this will be an upper bound for $E(x_i \cdot x_j)$. We have the following.

$$\begin{aligned} &E(x_i \cdot x_j) = \Pr(x_i = 1, x_j = 1) = \\ &= \Pr(x_i = 1 | x_j = 1) \cdot \Pr(x_j = 1) \leq \Pr(x_j = 1) = p \end{aligned}$$

Clearly, for $i = j$ it holds that $\Pr(x_i = 1, x_j = 1) = p$. Hence, we obtain the following upper bound on $E(E_J^2)$.

$$\begin{aligned} &E(E_J^2) = \sum_{i=1}^{\binom{|V_J|}{2}} E(x_i^2) + \sum_{i \neq j \in \{1,\ldots,\binom{|V_J|}{2}\}} E(x_i \cdot x_j) = \\ &\leq \binom{|V_J|}{2} \cdot p + \binom{\binom{|V_J|}{2}}{2} \cdot p \end{aligned}$$

Next, we use the upper bound on $E(E_J^2)$ to obtain an upper bound for $\mathrm{var}(\hat{p})$.

$$\mathrm{var}(\hat{p}) \leq p \cdot \left[\frac{\binom{|V_J|}{2} + \binom{\binom{|V_J|}{2}}{2}}{\binom{|V_J|}{2}^2} - p \right]$$

Using Chebyshev's inequality we obtain the following:

$$\Pr(|\hat{p} - p| > a) \leq \frac{p}{a^2} \cdot \left[\frac{\binom{|S|}{2} + \binom{\binom{|S|}{2}}{2}}{\binom{|S|}{2}^2} - p \right]$$

The obtained bounds are loose since we did not assume anything about the structure of the conflict graph and the dependencies among the violations. We show that better bounds can be obtained under the assumption that violations (or, equivalently, edges) are introduced randomly and independently.

We first introduce the rationale behind random violations as follows. Assume a random polluter which is a probability distribution over graphs on n labeled vertices, where each directed

edge appears independently with probability p. Each violation (edge) independently occurs between two tuples without following any specific pattern. Under this assumption, the number of edges in a sample J produces a binomial distribution.

$$\Pr[E_J = i] = \binom{2 \cdot \binom{|V_J|}{2}}{i} \cdot p^i \cdot (1-p)^{2 \cdot \binom{|V_J|}{2} - i}$$

For simplicity, we assume that the sample size is not too small and p is not too close to 0 or 1; hence, we can approximate the binomial $B(n, p)$ under the mentioned conditions using the normal distribution $N(np, np(1-p))$, and we can define a confidence interval parameterized by a confidence level $1 - 2\alpha$, and $n = 2 \cdot \binom{|V_J|}{2}$. The confidence interval of normal distribution is given by the following equation.

$$\Pr\left[|p - \hat{p}| \leq z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right] \geq 1 - 2\alpha \tag{4.1}$$

In the next subsection, we elaborate on how to use this idea to decide which threshold ϵ_J should be used on the sample, assuming that the desired threshold for the database is ϵ.

4.8.2 Computing the Sample Threshold

We now focus on the following problem. Given a sample J, a threshold ϵ and an error bound α, find the thresholds that should be used on the sample to obtain accurate ADCs with high probability. Note that the threshold may depend on the DC itself, since different DCs are violated by different tuple pairs, and, consequently, the conflict graphs of different DCs are different. That is, if φ is an ADC on the sample J w.r.t. ϵ_J^{φ}, then we require that with probability at least $1 - \alpha$, it holds that φ is an ADC on the entire database w.r.t. ϵ. We use Inequality 4.1 for this task.

Using the symmetry of the normal distribution we obtain the following.

$$\Pr\left[p - \hat{p} \leq z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right] \geq 1 - \alpha$$

Next, we add 1 and subtract 1 from the expression $p - \hat{p}$ and multiply both sides of the inner inequality by -1. Clearly, none of these operations affects the outer inequality and we have that:

$$\Pr\left[(1-p) - (1-\hat{p}) \geq -z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right] \geq 1 - \alpha$$

Finally, we move the term $(1 - \hat{p})$ to the other side of the inner inequality to obtain the following result.

$$\Pr\left[(1-p) \geq (1-\hat{p}) - z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right] \geq 1 - \alpha$$

Recall that our goal is to find an ϵ_J^{φ} such that if $1-\hat{p} \geq 1-\epsilon_J^{\varphi}$ then $\Pr(1-p \geq 1-\epsilon) > 1-\alpha$. Thus, all we need to do now is to set:

$$(1-\hat{p}) - z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \geq 1 - \epsilon$$

Or, equivalently:

$$(1-\hat{p}) \geq z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} + (1-\epsilon) \tag{4.2}$$

Consequently, if we define $\epsilon_J^{\varphi} = 1 - z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} + (1-\epsilon)$, and accept the DC φ if $1-\hat{p} \geq 1-\epsilon_J^{\varphi}$, then with probability at least $1-\alpha$, this DC is an ADC on the entire database w.r.t. the threshold ϵ. We conclude that we can use inequality 4.2 as criteria for accepting or rejecting an ADC on the sample.

Note that we can also look at Inequality 4.2 from a different point of view. Rather than defining a different threshold ϵ_J^{φ} for every DC, we can define the following approximation function:

$$f_1' = (1-\hat{p}) - z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Then, Inequality 4.2 implies that the DC φ is an ADC on the entire database w.r.t. the threshold ϵ if it is an ADC on the sample w.r.t. the approximation function f_1' and the same ϵ. Note that as the size of the sample increases, the value n increases as well, and the difference between f_1 and f_1' becomes very small, as expected.

4.9 Experimental Evaluation

In this section, we provide an experimental evaluation of our ADC discovery algorithm.

Dataset	#Tuples	#Attributes	#Golden DCs
Tax	1M	15	9
Stock	123K	7	6
Hospital	115K	19	7
Food	200K	17	10
Airport	55K	12	9
Adult	32K	15	3
Flight	582K	20	13
Voter	950K	25	12

Table 4.4: Datasets.

4.9.1 Experimental Setup

We implemented our enumeration algorithm, including the functions f_1 and f_2 in Java. As explained in Section 4.6, the function f_3 is hard to compute for DCs; hence, we implemented the algorithm in Alg. 10, and we refer to this algorithm when mentioning the function f_3. We also used the Java implementation of the algorithm AFASTDC by Chu et al. [40] and the Java implementation of the algorithm DCFinder provided by the authors of [160].

All experiments were executed on a machine with an Intel Xeon CPU E5-2603 v3 (1.60GHz, 12 cores) with 64GB of RAM running Ubuntu 14.04.3 LTS. All the experiments were repeated ten times and the average values are reported.

Following previous works on the problem of discovering DCs [15, 40, 160], we evaluate our algorithm on seven real-world datasets (**SP Stock**, **Hospital**, **Food Inspection**, **Airport**, **Adult**, **Flight**, and **NCVoter**), and one synthetic dataset (**Tax**). Table 4.4 depicts the number of tuples, attributes, and golden DCs (i.e., DCs obtained by human experts) for each one of the datasets.

4.9.2 Running Time

We evaluate the running time of our algorithm on the aforementioned datasets and compare them to the running times of the algorithm AFASTDC [40]. As we do not propose a new technique for constructing the evidence set, we only compare the running times of the DC enumeration algorithms (that is, we compare our algorithm ADCEnum with the algorithm SearchMinimalCovers used in [40, 160, 161], that we denote here by SearchMC). We discuss the running time of the evidence set construction later.

In the experiments, we used the approximation function f_1 (which is the function SearchMC is designed for) with the threshold $\epsilon = 0.1$. Figure 4.1 depicts the running times of both algo-

rithms. Note that the y axis is in log scale. The results show that our algorithm is two to three times faster than SearchMC on most of the datasets. As an example, it took SearchMC 5750 seconds (96 minutes) to generate all ADCs on the entire Tax dataset, while ADCEnum finished after 2373 seconds (39 minutes); that is, about 2.5 times faster.

We have also conducted a running time comparison between ADCEnum and SearchMC on different sample sizes. The results are depicted in Figure 4.4. Note that in some cases, the running times for higher sample sizes are slightly higher than the running times for smaller sample sizes (e.g., the running time on a sample that consists of 60% of the tuples in the Hospital dataset is higher than the running time on a sample that consists of 80% of the tuples). This is due to the fact that while increasing the number of tuples in the database significantly increases the number of tuple pairs, which, in turn, significantly increases the total running time, the number of *distinct* sets in the evidence set becomes relatively stable at some point, and does not change much when more tuples are added to the database. Since the running time of ADCEnum depends on the number of distinct sets in the evidence set, the running times for different sample sizes are usually very close. Therefore, we do not reason about these small differences.

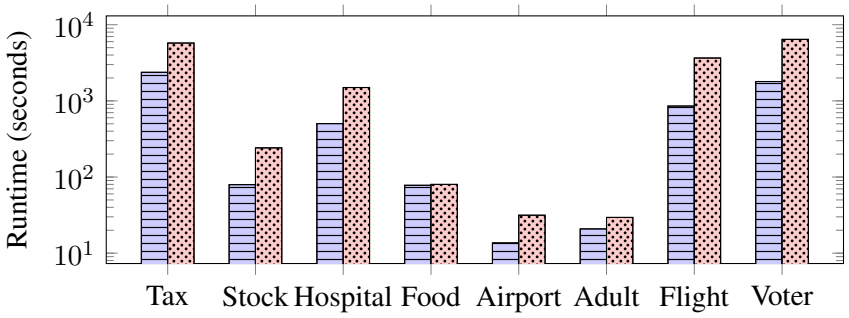


Figure 4.1: Running times of ADCEnum ( ) and SearchMC ( ).

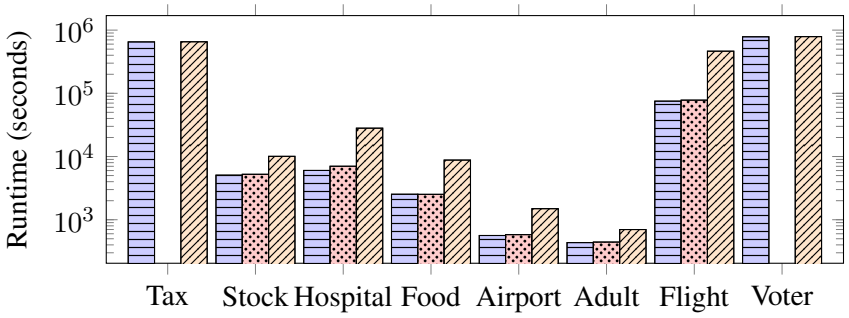


Figure 4.2: Running times of ADCMiner ( ), DCFinder ( ), and AFASTDC ( ).

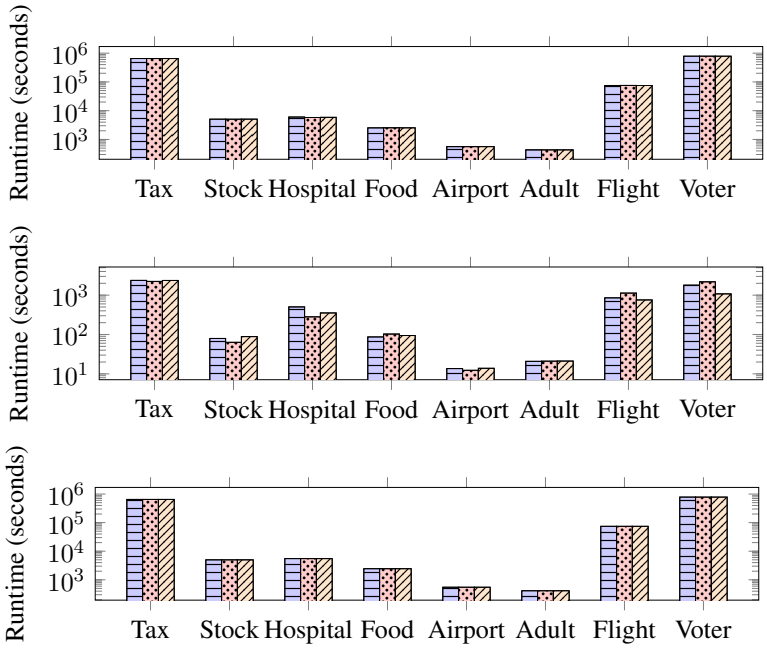


Figure 4.3: Running times of ADCMiner for f_1 ( ), f_2 ( ), and f_3 ( ). Top: total running time, middle: running time of ADCEnum, bottom: running time of evidence set construction.

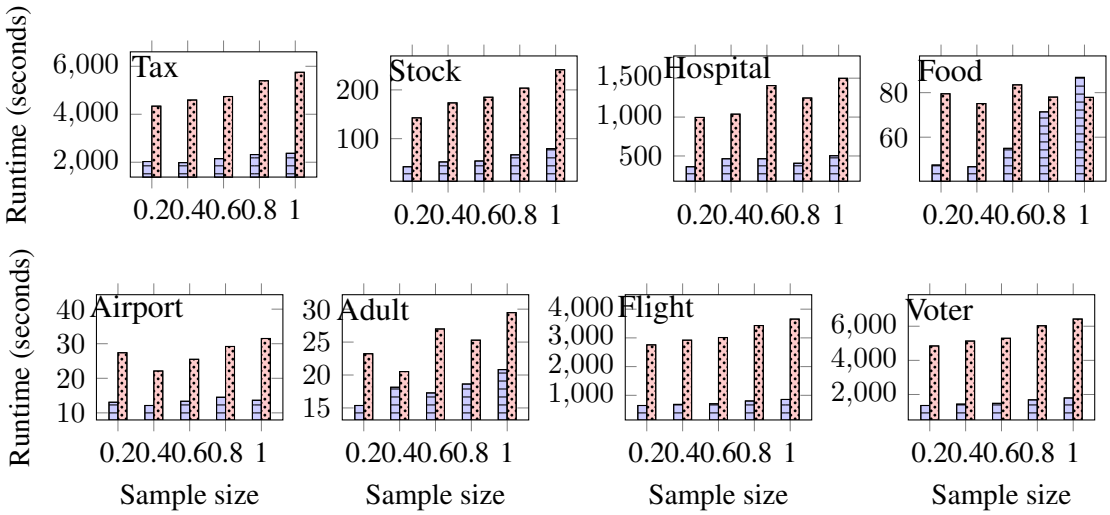


Figure 4.4: Running times in seconds of ADCEnum ( ) and SearchMC ( ) for varying sample sizes.

In Figure 4.2, we compare the total running times of ADCMiner, AFASTDC [40], and DCFinder [160]. Note that we do not report on the running times of DCFinder on the Tax and Voter datasets since we were unable to generate the evidence set with their algorithm (using the parameters recommended by the authors) even when dedicating almost the entire memory of our machine to the Java heap. While our algorithm is faster than the other two algorithms, the running time is mainly affected by the evidence set construction, which has a high computational cost in all three algorithms; hence, there is no drastic difference in the running times between our algorithm and DCFinder. Sampling allows us to significantly reduce the running times compared to the other solutions, and we show that in the next subsection.

In Figure 4.3, we present the running times of ADCMiner on all datasets for all three approximation functions. The top, middle, and bottom diagrams depict the total running time, the running time of ADCEnum, and the running time of the evidence set construction, respectively. Note that the running times of ADCEnum (which is the only part that depends on the choice of the approximation function) are very close for all three functions, and the total running time mostly depends on the evidence set construction. To construct the evidence set, we used the algorithm introduced by Pena et al. [160], which is the fastest algorithm for that task. However, since, as aforementioned, their algorithm was not able to process the Tax and NCVoter datasets, for these datasets, we used the algorithm of Chu et al. [40] to construct the evidence set. While for the Adult dataset, building the entire evidence set takes seven minutes, the evidence set construction requires almost an hour and a half on the SP Stock dataset, and more than twenty hours on the Flight dataset. This highlights the importance of incorporating sampling in our algorithm, as we are able to reduce the running times by as much as 90%, as we explain in the next subsection.

Finally, as discussed in Section 4.7, we do not select a random set from $\mathbf{uncov}$ in each iteration of ADCEnum, but rather the set that has the maximal intersection with the candidate list, as this choice decreases the running times, compared to the approach of Murakami and Uno [148] who select the set that minimizes this intersection. In Figure 4.5, we report the running times of ADCEnum on $60k$ tuples from the Tax, SP Stock, and Hospital datasets, for both approaches. We see that the running times are indeed lower when we choose the set with the maximal intersection, for all three approximation functions.

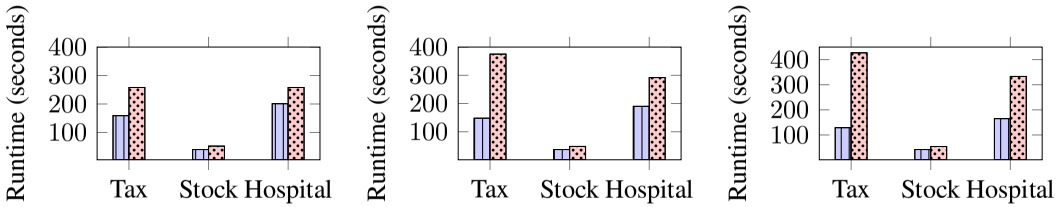


Figure 4.5: Running times of ADCEnum choosing the set F with the maximal ( ) and minimal ( ) intersection with **cand**, for the functions f_1 (top), f_2 (middle), and f_3 (bottom).

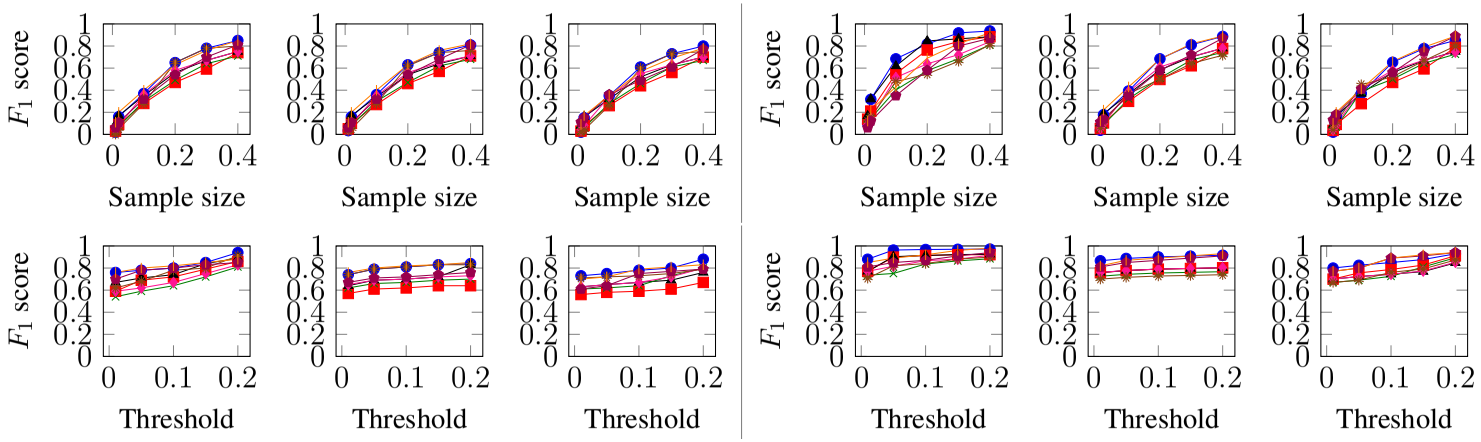


Figure 4.6: F_1 score for varying sample sizes and fixed $\epsilon = 0.01$ (top left) and $\epsilon = 0.1$ (top right), and varying thresholds and fixed sample size 0.3 (bottom left) and 0.4 (bottom right), under f_1 (left), f_2 (middle), and f_3 (right). Datasets: Tax (—●—), Stock (—▲—), Hospital (—■—), Food (—×—), Airport (—♦—), Adult (—✳—), Flight (—⬟—), and Voter (—+—).

4.9.3 Sampling

We now report on the quality of the ADCs obtained from a sample. In all of our experiments, the sample size is big enough so that the term $z_{1-2\alpha} \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$ in the approximation function f_1' defined in Section 4.8 has practically no impact on the function. Therefore, we use the same approximation function and threshold on both the sample and the entire dataset. In the experiments reported in Figure 4.6, we use a standard measure of quality, namely the F_1 score (i.e., $2 \cdot \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$). We compare the ADCs obtained from the sample with the ADCs obtained from the entire dataset.

We first fix a threshold and consider different sample sizes. The first three charts on the top of Figure 4.6 show the F_1 score for a fixed threshold $\epsilon = 0.01$ for varying sample sizes, ranging from 1% to 40% of the tuples in the dataset, for all three approximation functions. The last three charts show the F_1 score for a fixed threshold $\epsilon = 0.1$ for varying sample sizes. Clearly, the larger the sample is, the more accurate the results we obtain. Generally, we see that in order to obtain an F_1 score of about 0.7 or above we need to see about 40% of the tuples in the dataset. Note that we obtain a higher F_1-score on larger datasets (for which sampling is particularly important), as for such datasets a relatively small sample allows us to see enough tuples to obtain accurate results. For example, on the Tax and NCVoter datasets we consistently obtain an F_1-score of at least 0.7 or 0.8 when seeing 30% or 40% of the tuples, respectively.

The first three charts on the bottom of Figure 4.6 depict the F_1 score for a fixed sample size of 30% and varying thresholds, ranging from 0.01 to 0.2, for all three approximation functions. The last three charts depict the F_1 score for a fixed sample size of 40% and varying thresholds. Here, we can see that we obtain more accurate results when considering a higher threshold. This is due to the fact that a higher ϵ allows for more exceptions, and the DCs obtained using a higher threshold can be seen as obtained using a smaller sample of the database (as a smaller part of the database satisfies them). Hence, we are able to obtain result with high accuracy when considering a relatively small random sample. We conclude that the choice of the right threshold and sample size should be based on the size of the original dataset and the approximation function (as we discuss in the next subsection).

Next, we show the improvement in running times obtained when considering a sample. Figure 4.7 depicts the running times of ADCMiner for varying sample sizes on all datasets for the function f_1 (as shown in the previous subsection, the running times for all three functions are very close; hence, all three functions follow a similar trend). On the SP Stock dataset, we are able to reduce the total running time from eighty five to thirty two minutes when considering a sample that consists of 40% of the tuples—a reduction of more than 60%. For the Flight dataset, the running time goes down from almost twenty one hours to seventy minutes—a reduction of

almost 95%. For the Tax dataset, we are not able to use the same algorithm for constructing the evidence set on 100% and 40% of the tuples in the database. Using the original algorithm for constructing the evidence set by Chu et al. [40] we obtain a reduction of more than 94%—from 7.5 days to 10.5 hours. Using the algorithm BFASTDC to construct the evidence set we can obtain a similar reduction (of almost 90%) in the running time [161].

Finally, we validate the theoretical analysis of Section 4.8 as follows. For each dataset, we run our algorithm with the approximation function f_1 on varying sample sizes ranging from 5% to 80% of the tuples in the dataset. For each such sample, we compute the average value of $\epsilon - \hat{p}$ over the discovered ADCs (recall that $\hat{p}$ is the proportion of violating tuple pairs). Figure 4.8 depicts the values obtained in this experiment. Note that the actual numbers are very small; hence, the reported numbers are scaled up for each dataset (i.e., multiplied by a constant 10^x, where x depends on the dataset). We see that as the sample size increases, the value $\epsilon - \hat{p}$ decreases. Moreover, for each dataset, we have that $(\epsilon - \hat{p}) \sim \frac{1}{\sqrt{n}}$ (where $\sim$ denotes asymptotic equivalence and n is defined as in Section 4.8), which supports our main result of Section 4.8 (i.e., Inequality 4.2).

4.9.4 Qualitative Analysis

We compare the three approximation functions discussed in Section 4.6 in the following way. For each one of the datasets, we have a set of "golden" DCs; i.e., DCs obtained by domain experts. We take a sample of $10K$ tuples from each one of the datasets and add noise to the resulting dataset, such that each value has a probability of 0.001 to be modified, and if it is modified, then it has 50% chance of being changed to a new value from the active domain of the corresponding column and 50% chance to being changed to a typo. We also generate another dirty dataset in a similar way, but in this case, we only allow changing values in 0.001 of the tuples. Hence, in the first dataset, the errors are distributed among the tuples (and the number of modified tuples is

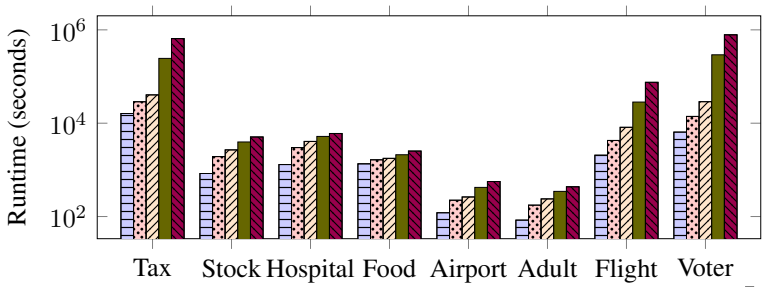


Figure 4.7: Running times of ADCMiner for varying sample sizes—20% ( ), 40% ( ), 60% ( ), 80% ( ), and 100% ( ).

usually very close to the number of modified values), while in the second dataset, the errors are concentrated in a small subset of the tuples.

Then, we run our algorithm on the two dirty datasets obtained from each one of the original datasets, with varying approximation thresholds ϵ (ranging from 10^{-6} to 10^{-1}). For each ϵ, we compute the G-recall, that is, the number of golden DC returned divided by the total number of golden DCs. We report the results in Figure 4.9. We also report the G-recall for $\epsilon = 0$ (i.e., when considering valid DCs) above each diagram (in parentheses). We observe the following phenomena. First, the G-recall for valid DCs is consistently zero, or very close to zero, which highlights the importance of considering approximate DCs. Second, the function f_1 produces results with a higher G-recall on smaller thresholds (i.e., $10^{-5} - 10^{-3}$), while the other two functions have a higher G-recall on the larger thresholds (i.e., $10^{-2} - 10^{-1}$). This is due to the fact that the functions f_2 and f_3 are more sensitive in the sense that a single tuple adds $\frac{1}{n}$ to the value of the functions f_2 and f_3 (where n is the number of tuples), while a pair of tuples adds $\frac{1}{n^2}$ to the value of the function f_1.

Another interesting phenomenon is that for we consistently obtain a higher G-recall on the error-concentrated datasets (especially for the functions f_2 and f_3). This is expected, especially for the function f_3, as when the errors are concentrated in a small subset of the tuples, these tuples will participate in every violation of the DC, and we only need to remove them from the database to satisfy the DC. The function f_3 (or, more accurately, our greedy approximation algorithm for this function) usually behaves better than the function f_2, especially on the error-concentrated datasets, and we are able to obtain a higher G-recall for a larger range of thresholds. As explained in Section 4.6, this is due to the fact that one erroneous tuple may result in a set of problematic tuples that contains every tuple in the database, while if we just remove this tuple, the DC will be satisfied. For this same reason, while with the function f_2 we constantly obtain the best accuracy using $\epsilon = 10^{-1}$, with the function f_3, we sometimes obtain better results with

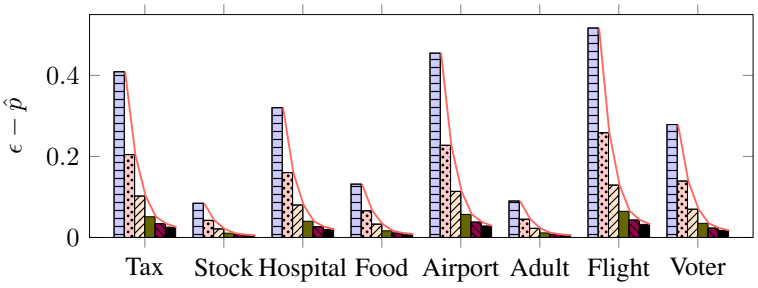


Figure 4.8: The average difference between ϵ and $\hat{p}$ over the ADCs obtained from varying sample sizes—5% (), 10% (), 20% (), 40% (), 60% (), and , 80% ().

the smaller threshold $\epsilon = 10^{-2}$.

Observe that in the experiments reported in Figure 4.6, we have used six specific thresholds, with which we are not always able to obtain the highest possible G-recall. If we conduct a more refined analysis, we find that using the threshold 5×10^{-5} for the function f_1 on the Tax dataset, for example, we are able to obtain a G-recall of 1. When we increase the threshold, we are able to obtain more general DCs (i.e., DCs consisting of less predicates) that we cannot obtain using smaller thresholds; however, some DCs become "too general", and we also lose some of the good DCs that we obtained using the smaller threshold. Hence, we need to find the threshold that will generate the best results with high probability. Using the above insights, we can choose a certain threshold (that depends on the approximation function), that will generate good results with high probability. Based on Figure 4.9, the best thresholds in that sense are 10^{-4}, 10^{-2}, and 10^{-1} for the functions f_1, f_2, and f_3, respectively. Using these thresholds we obtained an average G-recall of 0.71, 0.72, and 0.97, respectively.

Finally, Table 4.5 presents some of the golden DCs that we were able to obtain with the three approximation functions using the best threshold according to Figure 4.9, as well as an example of a corresponding valid DC from the same dirty dataset, obtained with the threshold $\epsilon = 0$. The DCs were obtained from the Tax, SP Stock, Hospital, Food, Flight, and NCVoter datasets. Many valid DCs are obtained from a single approximate DC by adding more predicates to cover for the errors in the database, which results in longer and less general DCs. Therefore, we often obtain less DCs and shorter DCs when considering ADCs. However, this is not always the case, as in some cases we also discover constraints that are approximate DCs, but cannot be extended to any minimal valid DC.

For example, the DC stating that the same zip code cannot correspond to two states (obtained from the Food dataset) becomes the DC stating that the same zip code cannot correspond to two states if the name and the type of the facility are the same. Clearly, we do not expect to obtain such complicated constraints, which strengthens our motivation for considering ADCs rather than valid DCs. In fact, while this DC generally holds, there are a few multi-state US zip codes (e.g., the zip code 84536 belongs to Utah and Arizona). If our original database contained two tuples with the same zip code and different states we would not be able to discover this DC unless considering ADCs. This example shows that ADCs are meaningful even when the database is clean, as they allow us to discover rules that are generally correct, but may have a few exceptions (about 0.03% of zip codes in the US cross states).

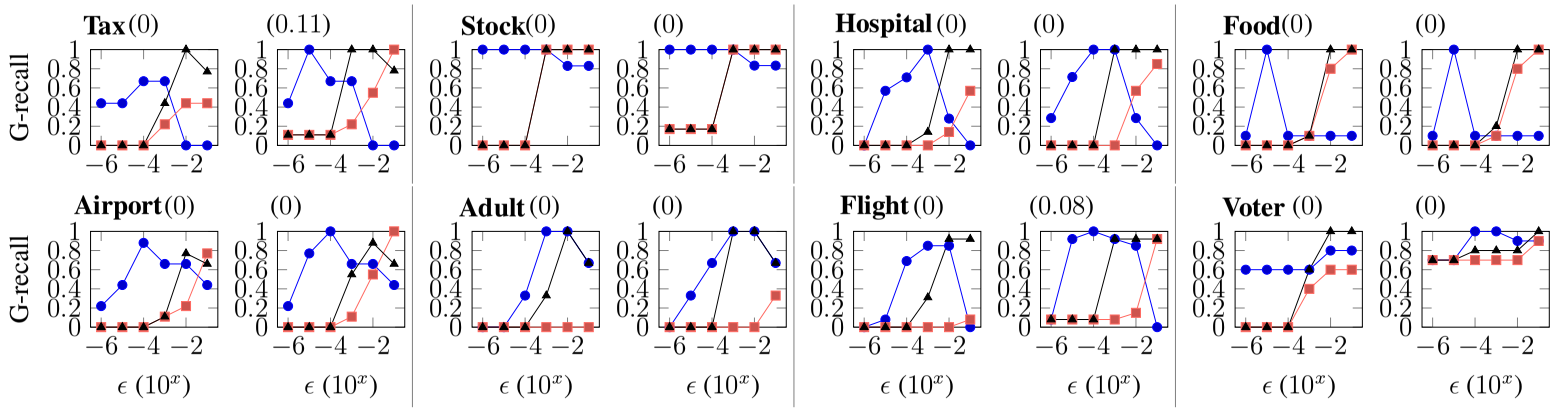


Figure 4.9: G-recall for varying thresholds under f_1 (—●—), f_2 (—■—) and f_3 (—▲—) for spread (left) and skewed (right) noise.

Approximate DC	Valid DC
$\forall t, t' \neg(t[\mathsf{St}] = t'[\mathsf{St}] \wedge t[\mathsf{Salary}] > t'[\mathsf{Salary}] \wedge t[\mathsf{Tax}] < t'[\mathsf{Tax}])$	$\forall t, t' \neg(t[\mathsf{St}] = t'[\mathsf{St}] \wedge t[\mathsf{Salary}] > t'[\mathsf{Salary}] \wedge t[\mathsf{Tax}] < t'[\mathsf{Tax}] \wedge t[\mathsf{G}] = t'[\mathsf{G}] \wedge t[\mathsf{SE}] \geq t'[\mathsf{SE}] \wedge t[\mathsf{Ph}] = t'[\mathsf{Ph}])$
$\forall t, t' \neg(t[\mathsf{High}] < t[\mathsf{Low}])$	$\forall t, t' \neg(t[\mathsf{High}] < t[\mathsf{Low}] \wedge t[\mathsf{Open}] < t[\mathsf{High}] \wedge t[\mathsf{Low}] \leq t[\mathsf{Close}])$
$\forall t, t' \neg(t[\mathsf{St}] = t'[\mathsf{St}] \wedge t[\mathsf{MC}] = t'[\mathsf{MC}] \wedge t[\mathsf{StAvg}] \neq t'[\mathsf{StAvg}])$	$\forall t, t' \neg(t[\mathsf{St}] = t'[\mathsf{St}] \wedge t[\mathsf{MC}] = t'[\mathsf{MC}] \wedge t[\mathsf{StAvg}] \neq t'[\mathsf{StAvg}] \wedge t[\mathsf{City}] = t'[\mathsf{City}] \wedge t[\mathsf{Sample}] = t'[\mathsf{Sample}])$
$\forall t, t' \neg(t[\mathsf{Zip}] = t'[\mathsf{Zip}] \wedge t[\mathsf{St}] \neq t'[\mathsf{St}])$	$\forall t, t' \neg(t[\mathsf{Zip}] = t'[\mathsf{Zip}] \wedge t[\mathsf{St}] \neq t'[\mathsf{St}]) \wedge t[\mathsf{Name}] = t'[\mathsf{Name}] \wedge t[\mathsf{FacilityType}] \neq t'[\mathsf{FacilityType}])$
$\forall t, t' \neg(t[\mathsf{OSt}] = t'[\mathsf{OSt}] \wedge t[\mathsf{DSt}] = t'[\mathsf{DSt}] \wedge t[\mathsf{DTime}] \geq t'[\mathsf{DTime}]) \wedge t[\mathsf{ATime}] \leq t'[\mathsf{ATime}]) \wedge t[\mathsf{ETime}] > t'[\mathsf{ETime}])$	$\forall t, t' \neg(t[\mathsf{St}] = t'[\mathsf{St}] \wedge t[\mathsf{MC}] = t'[\mathsf{MC}] \wedge t[\mathsf{SA}] = t'[\mathsf{SA}] \wedge t[\mathsf{PN}] = t'[\mathsf{PN}] \wedge t[\mathsf{Owner}] \neq t'[\mathsf{Owner}])$
$\forall t, t' \neg(t[\mathsf{Age}] < t'[\mathsf{Age}] \wedge t[\mathsf{BirthYear}] < t'[\mathsf{BirthYear}])$	$\forall t, t' \neg(t[\mathsf{Age}] < t'[\mathsf{Age}] \wedge t[\mathsf{BirthYear}] < t'[\mathsf{BirthYear}] \wedge t[\mathsf{C}] \neq t'[\mathsf{C}] \wedge t[\mathsf{Status}] \neq t'[\mathsf{Status}] \wedge t[\mathsf{Reason}] = t'[\mathsf{Reason}])$

Table 4.5: Approximate vs Valid DCs. Attributes: St – state, Ph – phone, G – gender, SE – single exemption, MC – measure code, OSt, DSt – origin and destination state, Dtime, ATime, ETime – departure, arrival, and elapsed time, C – county.

Chapter 5

Sampling from Data with Duplicated Records

In various domains, people rely on data to make critical decisions. For example, businesses use data analyses to decide about operations, sales, and marketing; Hospitals maintain patient records to track their treatments; Governments keep a census of their population to determine various aspects of public policy. Due to the complexity of the acquisition system and various human errors, this data is often noisy. These errors also affect the statistical properties of the dataset, which compromises its utility in various analytics.

Presence of multiple records that correspond to the same real-world entity is a common type of error in databases. Entity duplication often caused by collecting data from multiple sources or curated by different experts [94].

Duplication in records causes various problems for the downstream analytical tasks. For example, consider the problem of estimating the mean and variance of some columns of a table. The presence of duplicates can lead to inaccurate estimates or the presence of duplicates can lead to errors in k-means clustering where the presence of duplicates can perturb the computed centers and drastically change the clustering output [136]. Similarly, in the supervised learning, data duplication changes the optimization function leading to undesired behaviour. Thus, to ensure the correctness of downstream analytical results, it is important to repair such errors from our database. This process of de-duplicating the data is also referred to as record linkage [200], reference matching [144], and copy detection [187].

One common approach for removing the duplicates from a given dataset is compute a similarity (or distance) score for each pair of records, then clustering techniques are usually used to generate groups of records (a cluster represents duplicate records). Generating all record-pairs

has a time complexity of $O(|X|^2)$ where X denotes the dataset. To reduce the quadratic dependence on the dataset-size, *blocking* techniques are used in the database community [5, 14, 96, 99]. Locality-sensitive hashing are a class of methods, which partition a dataset into blocks such that similar points share a block and dis-similar points are partitioned across different blocks with very high probability. The problem of detecting duplicates across the entire dataset now reduces to the problem of detecting duplicates across the blocks. Hence, the time complexity of these methods is $O(|X|^2/B)$ where B is the number of blocks into which the data is partitioned.

The complexity of these methods can still be prohibitive or unnecessary in many scenarios. Consider the following thought experiment, where we are given an unclean dataset X. Let E be the set containing the distinct unique entities in X. For estimating the mean of E, one approach is detecting the duplicates in X using the previous known methods, and remove them from X to construct E and compute the mean. However, for the purpose of estimation, full construction of E is unnecessary. If we had a procedure $\mathcal{A}$, which generates a uniform sample from E (without constructing the full data set), we can use the sampled set S to estimate the mean. Note that the above discussion is equally applicable to machine learning tasks such as classification and duplications negatively impacts the model performance. Finding such a sampling algorithm $\mathcal{A}$, is the goal of this work. We lay down the theoretical foundations for this framework. Formally,

Given an unclean dataset X with duplicate entities, find a method $\mathcal{A}$ which can sample uniformly from E, the clean version of the dataset (or the set of unique distinct entities of X).

In previous approaches, Sample-and-Clean [197] uses sampling approaches for data deduplication; if the frequency for all the entities $e \in E$ are known, the authors proposed a method to sample uniformly from the set E. However, the assumption that the frequencies are known is extremely restrictive. In almost all practical situations, we do not expect to have access to such information. Sample debiasing [156] proposed an iterative algorithm based on resampling from the dataset for estimating the frequencies of entities. In extreme cases (when one entity has high frequency and other entities are unique), this method fails, and generally, it can be proved that the frequency estimator is an unbiased estimator iff resample the whole dataset or infinity big sample. Moreover, the analysis of the amount of bias is missing.

In General, to solve this problem, the following two-stage approach is required. In the first stage, the frequency of all the entities from a small sample are estimated. In the second stage, these estimations are used to obtain a set sampled uniformly at random from E. However, it is well-established that estimating frequencies (the first stage) is a non-trivial task [53, 170]. A simple application of the fundamental theorem of learning shows that no such $\mathcal{A}$ (which is based on estimating frequencies) can exist in general for arbitrary datasets. Observation 5.0.1 asserts that to get a reliable estimate of the frequency for an entity with low frequency, we need a linear number of samples.

Observation 5.0.1 (First impossibility result for sampling)**.** *Let* $X = \{ \overbrace{e_1, \ldots, e_1}^{f_1 \text{ times}}, \overbrace{e_2, \ldots, e_2}^{f_2 \text{ times}}\}$ *be a dataset with two entities with frequencies* f_1 *and* f_2 *respectively such that* $f_1 \leq \sqrt{n}$*. Let* $\mathcal{A}$ *be any algorithm which receives a sample* S *of size* m *and tries to estimate the frequency of the entities* e_1 *and* e_2*. Let* $\hat{f}_1, \hat{f}_2$ *be the estimated frequencies. If* $|\hat{f}_1 - f_1| \leq f_1\epsilon$ *then we have that*

$$m > C\frac{n}{\epsilon^2}$$

Proof. The proof follows from the fundamental theorem of learning (Thm. 6.7 in [185]). □

In this chapter, we investigate certain properties of the data under which it is possible to construct efficient (both statistical and computational) procedures that can sample uniformly at random form the set of entities E. We consider three categories of datasets/methods: (1) in Section 5.2, we consider datasets that are 'balanced' (Defn. 5.1.2) and show how that can help us estimate the frequencies of all the entities from a 'small' sample; (2) in Section 5.3, we consider datasets that can be successfully partitioned into hashing blocks, and we show how access to such blocks can help us estimate the frequencies and then sample uniformly from the set of entities E; and (3) in section 5.4, we consider the case when the dataset is generated by a mixture of k-spherical Gaussian distributions. For all the three cases, we provide mathematical bounds to prove correctness of our approach. Finally, in Section 5.5, we provide extensive experimental evaluation of our approach on both synthetic and real-world datasets. Considering each method assumptions, we inject duplicate data into the real datasets and then we evaluate our methods by comparing the average of the output sample with the original data average.

In Table 5, we give an overview of all methods that we cover in this chapter. The Gaussian prior method assumes a property of whole space and the generative process which is stronger than the LSH-based method. The LSH-based method assumes the boundary of each cluster is known. The balanced dataset method needs the lower-bound on the smallest cluster size, which is a weaker assumption than LSH-based method.

5.1 Preliminaries and solution overview

We denote by X, the original unclean dataset. Let E be the set of entities in X. That is, E is the set of distinct elements(unique entities) in X. The frequency of an entity $e \in E$ is defined as $freq(e) = |x \in X : x = e|$ and the probability of an entity is defined as $prob(e) = \frac{freq(e)}{|X|}$. We denote by $\mathcal{T}_X$, the uniform distribution over the entities of X. In this work, our goal is to sample (approximately) according to $\mathcal{T}_X$. Thus, we need a metric of distance between two distributions

Method	Goal	Assumptions	Sample Complexity	References
Goodman	Unbiased estimator for $\|E\|$	$\|E\| \ll n$	$\Omega\Big(\|E\|\Big)$	[87]
Valiant	Distribution support size	Large $\|E\|$	$\Omega\Big(\frac{\|E\|}{\log \|E\|}\Big)$	[192]
Sample-and-Clean	Unbiased *avg* estimator	$f_1, \ldots, f_{\|E\|}$ are given	NA	[197]
Sample debiasing	Uniform Sample	Linear Space	NA	[156]
Balanced Datasets	Uniform Sample	η-balance	$O\Big(\frac{1}{\epsilon^2\eta^2}\big(\log \|E\| \log \frac{\log \|E\|}{\epsilon\eta} + \log \frac{1}{\delta}\big)\Big)$	Thm 5.2.1&5.2.2
LSH-based	Uniform Sample	δ-isotropic set	$O\Big(q\big[\log s + \log(\frac{2q}{\delta})\big]\Big)$	Thm 5.3.3
Gaussian prior	Uniform Sample	Well-Separated GMM	$O\Big(\frac{d^3(\log k^2 + \log\log \epsilon^{-1} + \log \delta^{-1})}{\eta_{\min}\epsilon^2}\Big)$	Thm 5.4.3

Table 5.1: Bounds on the sample complexities of learning rejection process of generating uniform distribution. These results promise error of ϵ with probability δ. $|E|$ determine the number of entities. s is the maximum distance of lower and upper boundaries. q is the number of blocks. d is the dimension of Gaussians and k determine the number of them. $\eta_{\min}$ is the smallest weight parameter.

to quantify how far we are from our goal. For this, we use the total variation distance which is defined as $d_{TV}(\mathcal{P}, \mathcal{Q}) = \sup_{A \subseteq X} |\mathcal{P}(A) - \mathcal{Q}(A)|$

Definition 5.1.1 (Cleanable)**.** *Given set X and parameters $\epsilon, \delta \in (0, 1)$. Let E be the set of entities of X. We say that X is cleanable if there exists an algorithm $\mathcal{A}$ and function f such that we have the following. If $\mathcal{A}$ receives a sample S of size $m \geq f(\epsilon, \delta)$ then with probability atleast $1 - \delta$ (over the choice of S), $\mathcal{A}$ outputs a distribution $\mathcal{P}$ such that $d_{TV}(\mathcal{P}, \mathcal{T}_X) \leq \epsilon$*

If such an algorithm $\mathcal{A}$ exists for a dataset, we say that $\mathcal{A}$ cleans X. Furthermore, X is cleanable with sample complexity given by f. Our general two-stage approach for constructing $\mathcal{A}$ is described below. Note that instead of outputting a distribution $\mathcal{P}$, we output a set P of size p, sampled according to $\mathcal{P}$.

The second stage is a rejection sampling step, where we accept a point with probability inversely proportional to its (estimated) frequency. Thus, if the estimates are accurate, each point has an (approximately) equal probability of getting selected. The key component of our approach is the procedure $\mathcal{F}$ used during the first stage. Since, it is not possible to have an $\mathcal{F}$ in the general case, depending upon different properties of the dataset X, we use different procedures, as described above.

Algorithm 18: Uniform sampling from the clean data

Input: Dataset $X = \{x_1, \ldots, x_n\}$, sample size p
Output: Sample P

First stage
Use a procedure $\mathcal{F}$, to estimate the probabilities (or frequencies) for all $e \in E$.
Let $\hat{p}(e)$ be the estimated probabilities and let $m = \min \hat{p}(e)$
Second stage
while $|P| \neq p$ **do**
    Sample $v \in X$ uniformly at random and let a be a uniform random number in $[0, 1]$
    if $a < \frac{m}{\hat{p}(v)}$ **then**
        Add v to P
    end
end
return P

Case 1: Balanced datasets The data balance property asserts that the probability of each entity is atleast η. In Section 5.2, we describe a cleaning procedure for when the data has this property.

Definition 5.1.2 (η-balance)**.** *Given a set X and the corresponding set of entities E. We say that X is η-balanced w.r.t E if* $\min_{e \in E} \; prob(e) \geq \eta$.

Case 2: Blocked datasets Next, let us consider the opposite spectrum for de-duplication applications; a common scenario described below where each entity has a small (atmost a constant) number of duplicates. To uniformly sample in the such scenarios, we turn our attention towards Locality Sensitive Hashing or LSH-based methods. LSH is a popular technique that aims to partition a given dataset (and an associated similarity or distance metric) into blocks such that two points whose similarity is above a certain threshold lie in the same block. [1] A generic definition of LSH and related methods is given in Section 5.3.1. In Section 5.3, we assume that the dataset has been partitioned into blocks by a suitable LSH-based method. We cluster each block using the framework of regularized k-means algorithm [128].

Definition 5.1.3 (Regularized k-means objective)**.** *Given a clustering instance (X, d) and the number of clusters k. Partition X into $k+1$ subsets $\mathcal{C} = \{C_1, \ldots, C_k, C_{k+1}\}$ so as to minimize $\sum_{i=1}^{k} \sum_{x \in C_i} d^2(x, \mu_i) \; + \; \lambda |C_{k+1}|$. Here μ_i represents the center of C_i where the cluster centres*

[1] The actual definition says that two points lie in the same block with probability proportional to their similarity. But a non-probabilistic treatment suffices for this section.

$\mu_1, \ldots, \mu_k$. *In this framework, the algorithm is allowed to 'discard' points into a garbage cluster* C_{k+1}.

We then combine the clustering of each of these blocks into a clustering of the whole dataset. In Section 5.3, we further prove that if the dataset has the δ-isotropic property (defined next), then our LSH and clustering based method cleans X.

Definition 5.1.4 (δ-isotropic set)**.** *Let $\mathcal{D}$ be an isotropic distribution on the unit ball centred at the origin. Let $E = \{e_1, \ldots, e_n\}$ be points such that $\|e_i - e_j\| > \delta > 2$. Let $\mathcal{D}_i$ be the measure $\mathcal{D}$ translated w.r.t e_i. Let X_i be a set of size n_i generated according to the distribution $\mathcal{D}_i$. We say that $X = \cup X_i$ is a δ-isotropic set and E is the set of entities of X.*

Some common example of isotropic distributions include standard Gaussian distribution, Bernoulli distribution, spherical distributions, uniform distribution and many more [195].

Case 3: Gaussian prior Finally, in Section 5.4, we look at the same problem from a generative process perspective. That is when the probability of each entity is well-approximated by a mixture of k Gaussians (Defn. 5.1.6).

Definition 5.1.5 (Well-Separated Gaussian mixture models)**.** *For $i \in \{1, \ldots, k\}$, let μ_i, Σ_i be the parameters of k different Gaussian distributions. Also, let the mixing weights $\eta_i \in [0, 1]$ be such that $\sum_i \eta_i = 1$. A mixture of k Gaussians is well-separated if for all $i \neq j$, we have that $\|\mu_i - \mu_j\| \geq C \max(\sigma_i, \sigma_j)\sqrt{\log(\rho_\sigma/\eta_{min})} = O(\sqrt{\log k})$*

We say that a database X has ξ-gmm property if it can be 'well-approximated' by a mixture of k Gaussian distributions. We describe this intuition formally below.

Definition 5.1.6 (ξ-GMM property)**.** *Given a finite dataset X. Let η_i, μ_i and σ_i be the parameters of a well-separated mixture of k spherical Gaussians with density function $\mathcal{N}$. We say that X has ξ-GMM property if for all $x \in X$, we have that $|prob(x) - \mathcal{N}(x)| \leq \mathcal{N}(x)\xi$*

5.2 Sampling for balanced datasets

In this section, we consider datasets that satisfy the η-balanced property (Defn. 5.1.2). The cleaning algorithm is as described in Alg. 18. Procedure $\mathcal{F}$ (which estimates the frequencies) works as follows. We first sample a set T of size m uniformly at random from X. We compute

Algorithm 19: Probability estimates of all the entities for balanced datasets

Input: Dataset $X = \{x_1, \ldots, x_n\}$
Output: Probability estimates $\hat{p}_1, \ldots, \hat{p}_E$ for all the entities in X.

Let $W = \{v_1, v_2, \ldots, v_m\}$ be a set of size m sampled uniformly at random from X.
For all $v_i \in W$, let $\hat{p}(v_i) := \frac{|\{w \in W : x = v_i\}|}{m}$.
For all $v_i \in X \setminus W$, let $\hat{p}(v_i) = \min_{w \in W} \hat{p}(w)$
return $\hat{p}$

the count of all entities in our sample T and use the counts (divided by m) as probability estimates for these sampled points (Alg. 19).

Theorem 5.2.1 establishes rigorous bounds on the approximation guarantees of our sampling procedure. It shows that the sampling distribution approximates the uniform distribution (where the distance between two distributions is measured by total variation distance). Thm. 5.2.2 analyses the time complexity of our approach and shows that the time taken to sample one point is constant in expectation.

Theorem 5.2.1. *Given a finite dataset $X = \{x_1, \ldots, x_n\}$ which satisfies η-balance property w.r.t its set of entities E. Let $\mathcal{A}$ be as described in Alg. 18 with procedure $\mathcal{F}$ as described in Alg. 19. If $\mathcal{F}$ receives a sample of size $m \geq f(\epsilon, \delta) := \frac{a}{\epsilon^2 \eta^2}\Big(\log|E| \log\frac{\log|E|}{\epsilon\eta} + \log\frac{1}{\delta}\Big)$ then $\mathcal{A}$ cleans X with sample-complexity given by the function f.*

Proof. Let m, W and $\mathcal{A}$ be as defined in the description of Alg. 19. Let $h_x = \{x_i \in X : x_i = x\}$. And let $H = \{h_x : x \in X\}$ be a set of subsets of X. Now, $|H| = |E| = r$. Hence, we get that the $\mathrm{vcdim}(H) \leq \log r$. Now, using the classical result from learning theory (Thm. 5.3.4), we get that if

$$m \geq \frac{a}{\epsilon^2}\Big(d \log\frac{d}{\epsilon} + \log\frac{1}{\delta}\Big) =: M$$

where $d = \log r$ is an upper-bound on the $\mathrm{vcdim}(H)$, then with probability atleast $1 - \delta$, we have that for all $h_x \in H$, $\Big|\frac{|h_x \cap W|}{|W|} - p(h_x)\Big| \leq \epsilon$. Now, denote by $q(x) := \frac{|h_x \cap W|}{|W|}$. Then, we get that for $m \geq M$, with probability $1 - \delta$, for all $x \in X, \big|q(x) - p(x)\big| \leq \epsilon$. Now, for all $x \notin W$, we have that $p(x) \leq \epsilon$ which contradicts the fact that $p(x) > \eta$. Thus, we see that the sampling procedure samples a point with probability $q(x) \propto \frac{p(x)}{\hat{p}(x)}$ where $1 - \frac{\epsilon}{\eta - \epsilon} \leq \frac{p(x)}{\hat{p}(x)} \leq 1 + \frac{\epsilon}{\eta - \epsilon}$ Choosing $\epsilon = \frac{\epsilon}{(1+\epsilon)}\eta$ gives us the result of the theorem. □

Theorem 5.2.2. *Let the framework be as in Thm. 5.2.1. And define $\eta_1 = \max_{e \in E} prob(e)$ and $\eta_2 = \min_{e \in E} prob(e)$. Then the preprocessing time of algorithm $\mathcal{A}$ is $O(\log^2|E|)$ and the expected time taken to sample one point is $O(\frac{\eta_1}{\eta_2})$.*

Proof. Note that the preprocessing time depends linearly on m. Using the bound on m from Thm. 5.2.1 the result on pre-processing time follows. Similarly, the expected time taken to sample a point is upper bounded by $\frac{\eta_1+\epsilon}{\eta-\epsilon}$ where ϵ is as defined in the proof of Thm. 5.2.1. Substituting the values of $\epsilon = \frac{\epsilon}{(1+\epsilon)}\eta$, we get that the expectation is upper bounded by $\epsilon + (1+\epsilon)\frac{\eta_1}{\eta_2}$. □

5.3 LSH-based sampling

In Section 5.2, we saw a method that samples approximately according to the uniform distribution (over the entities) if the given dataset has η-niceness. The number of samples required to construct this distribution is $O(\frac{1}{\eta})$. In situations when $\eta = O(\frac{1}{n})$, the bounds from the previous section are vacuous. In this section, we assume that the data has been partitioned into hash-blocks such that all the duplicates are within the same block. That is for all $x \in X$, all y which correspond to the same entity as x, share the same hash-block. Hence, we can treat each block as a separate instance of the cleaning (or the de-duplication) problem.

Our goal is to estimate the frequency (or probability) of each entity in E. To achieve this goal, we cluster the set X and then estimate the frequency of an entity e as the number points that belong to the same cluster as e. Since each block can be treated independently, we focus on clustering a hash-block rather than the whole set. Clustering a hash block, although easier than clustering the entire dataset, still has some issues: the number of clusters is still unknown and hence standard clustering formulations are inapplicable for our setting. In Section 5.3.2, we describe our regularized clustering algorithm which can cluster each hash block given k, the number of non-singleton clusters.

The exact knowledge of the number of non-singleton clusters for each hash block can still be restrictive in many applications. A weaker assumption is the knowledge of an upper and lower bound on the *number of non-singleton clusters* within each hash-block. That is for each block, we know that $k \in [k_1, k_2]$ where k is the number of non-singleton clusters and k_1 and k_2 are known. In Section 5.3.3, we describe a principled approach to select the right value of k based on the framework of SSC (semi-supervised clustering) introduced in [126, 127] and describe our complete sampling approach.

5.3.1 Locality Sensitive Hashing

Definition 5.3.1 (Hash function)**.** *Given a set X. A hash function $h : X \to \{1, \ldots, k\}$ partitions the set X into k blocks.*

Definition 5.3.2 (LSH). *Given a set X and a similarity measure $s : X \times X \to [0, 1]$. Let $\mathcal{H}$ be a set of functions over X. An LSH for the similarity measure s is a probability distribution over $\mathcal{H}$ such that for all $x_1, x_2 \in X$*

$$\mathop{\mathbf{P}}_{h \in \mathcal{H}}[h(x_1) = h(x_2)] = s(x_1, x_2)$$

Note that the above definition is terms of similarity function s. For our purposes, it will be more comfortable to talk in terms of the distance metric d, rather than a similarity function. Note that a given distance metric implies a similarity function and vice-versa. Hence, given (X, d), if there exists a such a probability distribution, then we say that the given metric d is LSH-able.

We are now ready to describe a generic hashing scheme based on a LSH-able metric. Sample hash functions $h_1, \ldots, h_k$ and group them into r groups $H_1, \ldots, H_r$ of size s each, that is, $rs = k$. Now, two points x_1, x_2 end in the same block if they have same hash value on either one of the r groups. The approach is described below.

Algorithm 20: A generic LSH based hashing algorithm [29, 109]

Input: (X, d), a class of hash functions $\mathcal{H}$ and integers r, s.
Output: Partition Q of the set X.

1. Let D be a distribution over H which satisfies Defn. 5.3.2 and let $k = rs$.
2. Sample hash functions $h_1, \ldots, h_k$ iid using D.
3. Group the hash functions into s bands. Each band contains r hash functions.
4. For all x and $0 \leq i \leq s - 1$, let $g_i(x) = (h_{is+1}(x), \ldots, h_{is+r}(x))$. That is, $g_i(x)$ represents the i^{th} signature of x.
5. Let Q be the partition induced by g_i's. That is, if there exists $0 \leq i < s$ such that $g_i(x_1) = g_i(x_2)$ then x_1 and x_2 belong to the same group in Q.
6. Output Q.

Theorem 5.3.1. *Given a set X, a distance function $d : X \times X \to [0, 1]$, a class of hash functions H, threshold parameter λ and a parameter δ. Let $\mathcal{A}$ be a generic LSH based algorithm (Alg. 20) Choose r, s such that $\frac{1}{2\lambda} < r < \frac{1}{-\ln(1-\lambda)}$ and $s = \lceil 2.2 \ln(\frac{1}{\delta}) \rceil$. Define $\delta' := s \ln(1 + \delta)$. Then for $x_1, x_2 \in X$*

- *If $d(x_1, x_2) \leq \lambda$ then $\mathbf{P}_{h \in H}$ [$q(x_1, x_2) = 1$] $> 1 - \delta$*

where $q(x, y) = 1$ iff x, y belong to the same group in Q.

Proof. Observe that

$$\mathbf{P}[b(x,y)=0] = \mathbf{P}\ [\bigcap_{i=1}^{s} g_i(x) \neq g_i(y)] = \prod_i \Big(1 - \prod_{j=1}^{r} \mathbf{P}[h_{(i-1)r+j}(x) = h_{(i-1)r+j}(y)]\Big)$$

$$= \prod_{i=1}^{s}(1 - \prod_{j=1}^{r} f(x,y)) = (1 - f(x,y)^r)^s$$

Consider the case when $d(x,y) \leq \lambda$. From the choice of s, we know that $s \geq 2.2\ln(1/\delta) \implies s \geq \frac{\ln(1/\delta)}{1-\ln(e-1)} \iff 1 - \frac{1}{e} \leq \delta^{1/s}$. From the choice of r, we know that $r < \frac{1}{-\ln(1-\lambda)} \iff r\ln(\frac{1}{1-\lambda}) < 1 \iff (1-\lambda)^r > \frac{1}{e}$. Hence, then we have that $\mathbf{P}[b(x,y) = 0] = (1 - (1 - d(x,y))^r)^s \leq (1 - (1-\lambda)^r)^s < \delta$. □

For the simplicity of analysis, for the rest of the subsections, we will assume that if $x_1, x_2 \in X$ are duplicates of one another then $q(x_1, x_2) = 1$. In the probabilistic case our results hold true with the corresponding probability.

5.3.2 Regularized k-means clustering

To solve the optimization problem in Defn. 5.1.3, we use the following strategy. We first decide which points go into the set C_{k+1} (that is which points belong to singleton clusters). We remove those points from the set and k-cluster the remaining points using an SDP based algorithm (same as in [128]). Our approach is described in Alg. 21.

Next, Thm. 5.3.2 shows that under δ-isotropic assumption and if the number of non-singleton clusters k is known, then Alg. 21 finds the desired clustering solution.

Theorem 5.3.2. *Given a clustering instance (X, d) where $x_i \in X$ has dimension p. Let X be a δ-isotropic set and let $E = \{e_1, \ldots, e_n\}$ be the set of entities of X. Let $e_1, \ldots, e_k$ be the set of non-singleton entities of X. In addition, let $e_i \in X$. Denote by B_i all the records in X which correspond to the entity e_i and $C_{k+1} = \{e_{k+1}, \ldots, e_n\}$. If $\delta > 2 + O(\sqrt{k/p})$ then there exists a constant $c > 0$ such that with probability at least $1 - 2p\exp(\frac{-cN\theta}{p\log^2 N})$ Alg. 21 finds the intended cluster solution $\mathcal{C}^* = \{B_1, \ldots, B_k, C_{k+1}\}$ when given X, k and $\mu = 1$ as input.*

Proof. We know that X satisfies δ-isotropic condition. Hence, for all the entities $e \in U := \{e_{k+1}, \ldots, e_n\}$, we have that $|S_e| = 1$. Also, for $e \in U$ we have that $e \notin S_{e'}$ (cause of δ-isotropy). Thus, we have that $e \notin X'$ and $e \in C_{k+1}$. Hence, we get that $U \subseteq C_{k+1}$.

Algorithm 21: Regularized k-means clustering

Input: Clustering instance (X, d), the number of non-singleton clusters k and constant μ
Output: Partition into $k+1$ clusters.

For all x, compute $S_x = \{y : d(x,y) \leq \mu\}$. If $|S_x| > 1$ then $X' = X' \cup S_x$.
$C_{k+1} = X \setminus X'$ and $X = X'$.
If $|X| \leq constant$, execute a brute force search for all possible k partitions.
For all $x_i \in X$, compute the matrix $D_{ij} = \|x_i - x_j\|_2^2$.
Set $\lambda = \infty$ and $y = 0$ and solve Eqn. 5.1 using any standard SDP solver and obtain matrix Z.

$$\textbf{SDP} \begin{cases} \min_{Z,y} & \text{Tr}(DZ) + \lambda \langle \mathbf{1}, y \rangle \\ \text{s.t.} & \text{Tr}(Z) = k \\ & \left(\frac{Z+Z^T}{2}\right) \cdot \mathbf{1} + y = \mathbf{1} \\ & Z \geq 0, y \geq 0, Z \succeq 0 \end{cases} \tag{5.1}$$

k-cluster the columns of $X^T Z$ to obtain clusters $C_1, \ldots, C_k$.
Output $\mathcal{C}' = \{C_1, \ldots, C_k, C_{k+1}\}$.

Now, we will show that all $x \in X \setminus U$, doesn't belong to C_{k+1}. For the sake of contradiction, assume that there exists $x \in X \setminus U$ such that $x \in C_{k+1}$. WLOG, let $x \in e_i$ where $1 \leq i \leq k$. This implies that for all $x' \in X$, $x \notin S'_x$. This is a contradiction as $x \in S_{e_i}$. Thus, we get that $U = C_{k+1}$. Now, using Thm. 5.7 from [125] completes the proof of the theorem. □

5.3.3 Semi-supervised clustering

In the previous section, we discussed an algorithm, which finds the target clustering when the number of non-singleton clusters k is known. In this section, we extend it to the case when it is given that $k \in [k_1, k_2]$. We use the framework of semi-supervised clustering selection (SSC).

Definition 5.3.3 (Clustering loss)**.** *Given a clustering C of a set X and an unknown target clustering $\mathcal{C}^*$. Denote by P^+ the uniform distribution over $\{(x,y) \in X^2 : C^*(x,y) = 1\}$ and P^- the uniform distribution over $\{(x,y) \in X^2 : C^*(x,y) = 0\}$. The loss of clustering C is defined as*

$$L_{C^*}(C) = \mu \mathop{\mathbf{P}}_{(x,y)\sim P^+}\big[C(x,y) = 0\big] + (1-\mu) \mathop{\mathbf{P}}_{(x,y)\sim P^-}\big[C(x,y) = 1\big]$$

where $C(x,y) = 1$ iff x, y belong to the same cluster according to C.

Definition 5.3.4 (Semi-Supervised Clustering (SSC) [127])**.** *Given a clustering instance (X, d). Let C^* be an unknown target clustering of X. Find $\hat{C} \in \mathcal{G} := \{C_1, \ldots, C_p\}$ such that $\hat{C} = \arg\min_{C \in \mathcal{G}} \; L_{C^*}(C)$ where $L_{C^*}(C)$ measures the (weighted) average of the fraction of pairs of points which belong to the same cluster according to C^* but belong to different clusters in C plus the fraction of pairs which belong to the different clusters in C^* but belong to same cluster in C.*

For each value of k from $k_1, \ldots, k_2$, we use Alg. 21, to generate clusterings $\mathcal{G} = \{\mathcal{C}_{k_1}, \ldots, C_{k_2}\}$. Note the each $\mathcal{C}_{k_i}$ is a clustering of the given dataset. We then use the SSC framework (Alg. 22) to select the best clustering from $\mathcal{G}$. We describe our "clustering and hashing" based sampling algorithm and then prove the main result from this section.

Algorithm 22: Empirical Risk Minimization for SSC

Input: (X, d), a set of clusterings $\mathcal{F}$, a C^*-oracle and size m.
Output: $C \in \mathcal{F}$

1 Sample a pair (x, y) uniformly at random from X^2. If $C^*(x, y) = 1$ then $S_+ = S_+ \cup (x, y)$ else $S_- = S_- \cup (x, y)$.
2 Repeat till at least one of $|S_+|$ and $|S_-|$ is less than m.
3 Define $\hat{pl}(C) = \frac{|\{(x,y) \in S_+ : C(x,y)=0\}|}{|S_+|}$ and $\hat{nl}(C) = \frac{|\{(x,y) \in S_- : C(x,y)=0\}|}{|S_-|}$
4 Define $\hat{L}(C) = \mu \hat{pl}(C) + (1 - \mu)\hat{nl}(C)$.
5 Output $\arg\min_{C \in \mathcal{F}} \; \hat{L}(C)$

Theorem 5.3.3. *Given a finite dataset $X = \{x_1, \ldots, x_n\}$ which has the δ-isotropic property w.r.t its set of entities E. Let x_i have dimension g. Let X be partitioned into blocks $X_1, \ldots, X_q$ such that all records corresponding to the same entity lie within the same hash block. For each of the blocks X_i let k_i be the number of entities with number of corresponding records greater than 1 (or non-singleton clusters). Let $\mathcal{C}_i^*$ be the corresponding clustering of the non-singleton entities of X_i be such that any other clustering C of X_i has loss $L_{\mathcal{C}_i^*}(C) > o(\alpha)$. Let $\mathcal{A}$ be as described in Alg. 18 with procedure $\mathcal{F}$ as described in Alg. 23. If $\mathcal{F}$ receives a sample of size $m \geq aq\frac{\log s + \log(\frac{2q}{\delta})}{\alpha^2}$ where a is a universal constant and $s = \max_i(k_{i2} - k_{i1})$ where k_{i2}, k_{i1} are as defined in Alg. 23. Then with probability atleast $1 - \delta - 2gq\exp(\frac{-cN\theta}{g\log^2 N})$[2], $\mathcal{A}$ samples a set P of size p such that*

$$d_{TV}(\mathcal{P}, \mathcal{T}_X) = 0.$$

Proof. Before start to proof this theorem, we need the following lemmas,

[2] c is a global constant and $N = \min B_i$ where B_i is the total number of points in non-single clusters for $1 \leq i \leq q$. The minimum is over all B_i greater than a large global constant.

Algorithm 23: Probability estimates for all entities under LSH

Input: Dataset X which has been partitioned in to blocks $X_1, \ldots, X_q$ and sample size m
Output: $\hat{p}$.

$\mathcal{G}_{i \in [1 \ldots q]} = \emptyset$
while $1 \leq i \leq q$ **do**
    let the number of non-singleton clusters $\in [k_{i1}, k_{i2}]$.
    for $k \in [k_{i1}, k_{i2}]$ **do**
        Use Alg. 21 with input X_i, d, k to obtain clustering $\mathcal{C}$.
        Let $\mathcal{G}_i = \mathcal{G}_i \cup \mathcal{C}$.
    end
    Use the SSC framework (Alg. 22) on $\mathcal{G}_i$ with sample $\frac{m}{q}$ to obtain $\hat{\mathcal{C}}_i$ of X_i.
end
Combine the clusterings to obtain a clustering $\hat{\mathcal{C}}$ of the whole set X.
Define $\hat{p}(e) = \frac{1}{|\hat{\mathcal{C}}(e)|}$ where $C(e)$ denotes the number of points which belong to the same cluster as e.

Lemma 5.3.1 (Sample Complexity)**.** *Given metric space (X, d), a class of clusterings $\mathcal{F}$ of size s and a threshold parameter λ. Given $\epsilon, \delta \in (0, 1)$ and a C^*-oracle. Let $\mathcal{A}$ be the ERM-based approach as described in Alg. 22 and $\bar{C}$ be the output of $\mathcal{A}$. Let $C^* \in \mathcal{F}$. If*

$$m \ \geq a \frac{\log s + \log(\frac{2}{\delta})}{\epsilon^2} \tag{5.2}$$

where a is a global constant then with probability at least $1 - \delta$ (over the randomness in the sampling procedure), we have that

$$L_{C^*}(\bar{C}) \ \leq \ \epsilon$$

Proof. The proof of the theorem involves a straightforward application of the fundamental theorem of learning. If $m > a \frac{\text{vcdim}(\mathcal{F}) + \log(\frac{1}{\delta})}{\epsilon^2}$, then with probability at least $1 - \delta$, we have that $|\hat{n}l(C) - nl(C)| < \epsilon$. Similarly, we have that with probability at least $1 - \delta$, we have that $|\hat{p}l(C) - pl(C)| < \epsilon$. Combining these two equations, we get that with probability at least $1 - 2\delta$, $|\hat{l}(C) - l(C)| < \epsilon$. Now, $l(\bar{C}) \leq \hat{l}(\bar{C}) + \epsilon \leq \hat{l}(C^*) + \epsilon \leq l(C^*) + 2\epsilon$. Substituting, $\delta = \delta/2$ and $\epsilon = \epsilon/2$ completes the result of the theorem. □

Next we prove an upper bound on the number of queries to the oracle to sample m_+ positive and m_- negative pairs.

Lemma 5.3.2 (Query Complexity)**.** *Let the framework be as in Lemma 5.3.1. In addition, let* $\gamma = \mathbf{P}[C^*(x,y) = 0]$. *With probability at least* $1 - \exp\big(-\frac{\nu^2 m_-}{4}\big) - \exp\big(-\frac{\nu^2 m_+}{4}\big)$ *over the randomness in the sampling procedure, the number of same-cluster queries* q *made by* $\mathcal{A}$ *is*

$$q \leq (1+\nu)\left(\frac{m_-}{\gamma} + \frac{m_+}{1-\gamma}\right)$$

Proof. Let q_- denote the number queries to sample the set S_-. Now, $\mathbf{E}[q_-] = \frac{1}{\gamma}$. Thus, using Thm. 5.3.5, we get that $q_- \leq \frac{(1+\nu)m_-}{\beta(1-\epsilon)}$ with probability at least $1 - \exp(\frac{-\nu^2 m_-}{4})$. □

Now, can continue our proof by putting all these result together. Let m be as in the statement of the theorem. Then, Lemma 5.3.1 implies that with probability at least $1-\delta$, we have that for all i, $L_{\mathcal{C}_i^*}(\hat{\mathcal{C}}_i) \leq \epsilon$. However, we know that for all clusterings $\hat{\mathcal{C}}_i$ of X_i, we have that $L_{\mathcal{C}_i^*}(\hat{\mathcal{C}}_i) > \epsilon$. Hence, $\hat{\mathcal{C}}_i = \mathcal{C}_i^*$. Hence, we get that $\hat{C} = C^*$. In other words, Alg. 23 recovers the target clustering. Once the target clustering is known, the rest of the algorithm samples a point uniformly at random and accepts it with probability proportional to $\frac{1}{|C(x)|}$ where $C(x)$ denotes the cluster to which x belongs. Hence, for any entity e, we have that

$$\mathbf{P}[e] \propto \frac{|C^*(e)|}{|C(e)|} = 1.$$

The extra $2gq\exp(\frac{-cN\theta}{g\log^2 N})$ term is due to the success probability of the regularized SDP algorithm. □

5.3.4 Classical theorems and results

Theorem 5.3.4 (Vapnik and Chervonenkis [193])**.** *Let* X *be a domain set and* D *a probability distribution over* X. *Let* H *be a class of subsets of* X *of finite VC-dimension* d. *Let* $\epsilon, \delta \in (0,1)$. *Let* $S \subseteq X$ *be picked i.i.d according to* D *of size* m. *If* $m > \frac{c}{\epsilon^2}(d\log\frac{d}{\epsilon} + \log\frac{1}{\delta})$, *then with probability* $1-\delta$ *over the choice of* S, *we have that* $\forall h \in H$

$$\left|\frac{|h \cap S|}{|S|} - P(h)\right| < \epsilon$$

Theorem 5.3.5 (Concentration inequality for sum of geometric random variables [22])**.** *Let* $X = X_1 + \ldots + X_n$ *be* n *geometrically distributed random variables such that* $\mathbf{E}[X_i] = \mu$. *Then*

$$\mathbf{P}[X > (1+\nu)n\mu] \leq \exp\left(\frac{-\nu^2\mu n}{2(1+\nu)}\right)$$

5.4 Sampling Under a Gaussian Prior

In this section, we consider datasets X, which have the ξ-GMM property (Definition 5.1.6). ξ-GMM property informally means the dataset a mixture of k distributions that each component is quasi-normal has at most ξ total variation (TV) error from a normal distribution.

The GMM is the probabilities of all the entities can be well approximated by a mixture of k-Gaussian distributions. We use the EM algorithm to estimate the parameters of the mixture model. Next, we prove that if the generative model is a mixture of k well-separated spherical Gaussians then the sampling approach described in Section 5.1 takes a point approximately according to the uniform distribution.

Here is the algorithm,

Algorithm 24: Probability estimates for all entities under GMM prior

Input: Dataset $X \subseteq \mathbf{R}^d$, the number of mixtures k, sample size m and number of steps T
Output: Sample S

1 Run the EM algorithm for T steps with X as input and obtain parameters $\theta_i = (\hat{\eta}_i, \hat{\mu}_i, \hat{\sigma}_i)$.
2 Define $\hat{\mathcal{N}}(x) = \sum_i \hat{\eta}_i \mathcal{N}(x; \hat{\mu}_i, \hat{\sigma}_i)$ where $\mathcal{N}(x; \mu, \sigma)$ is the Gaussian with mean μ and variance σ^2.
3 Draw m data point with rejection probability $\hat{\mathcal{N}}(x)$.

The Alg. 24 initialized with point provided by the following theorem.

Theorem 5.4.1 (Thm 3.6 in [129])**.** *Given a well-separated mixture of k-spherical Gaussians. There exists initializations for $\mu_1^{(0)}, \ldots, \mu_k^{(0)}$ for the means and $\eta_1^{(0)}, \ldots, \eta_k^{(0)}$ for the mixing weights such that if the EM algorithm is initialized with these parameters, and if each step of the EM algorithm receives a sample of size $m > C' \frac{d(\log(k^2 T) + \log(\frac{1}{\delta}))}{\eta_{\min} \epsilon^2}$ then in $T = O(\log(1/\epsilon))$ iterations, converges to parameters $\hat{\eta}_i, \hat{\mu}_i$ and $\hat{\sigma}_i$ such that for all i,*

$$\|\hat{\mu}_i - \mu_i\| \leq \sigma_i \epsilon \quad \text{and} \quad |\hat{\eta}_i - \eta_i| \leq \eta_i \epsilon \quad \text{and} \quad |\hat{\sigma}_i^2 - \sigma_i^2| \leq \sigma_i^2 \epsilon / \sqrt{d}$$

with probability at least $1 - \delta - \frac{T}{k^{30} n^{C-2}}$.

5.4.1 Maximum Likelihood Estimation

The EM algorithm in second stage of Alg. 24 is composed of two steps, the E-step that constructs the expectation of the log-likelihood on the current estimators, and the M-step that maximizes this

Algorithm 25: EM estimation

Input: Dataset $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, Sample $\boldsymbol{T} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_m)$, Allocation $\boldsymbol{e} = \{e_1, \ldots, e_m\}$, Number of distinct value K, Threshold ϵ

Output: GMM estimation

Repeat until the total parameters change is less than ϵ:

Weak labeling of each observation $\boldsymbol{x}_i$, for $i = 1, ..., m$, for each $k \in [K]$:

$$\text{E step} \quad l_k = \frac{\eta_k e^{\frac{-\|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|^2}{2\sigma_k^2} - \frac{d}{2}\log(\sigma_k^2)}}{\sum_{j=1}^{K} \eta_j e^{\frac{-\|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2} - \frac{d}{2}\log(\sigma_j^2)}}$$

$$\text{M-step} \quad \eta_k^+ = \mathbb{E}_{\boldsymbol{T}}[l_k], \boldsymbol{\mu}_k^+ = \frac{\mathbb{E}_{\boldsymbol{T}}[l_k \boldsymbol{x}_i]}{\mathbb{E}_{\boldsymbol{T}}[l_k]}, \sigma_k^{+2} = \frac{\mathbb{E}_{\boldsymbol{T}}[l_k \|\boldsymbol{x}_i - \boldsymbol{\mu}_k^+\|^2]}{d\mathbb{E}_{\boldsymbol{T}}[l_k]}$$

return $\hat{\mathcal{N}}(\boldsymbol{x}|\eta_k, \boldsymbol{\mu}_k, \sigma_k^2) = \sum_{k=1}^{K} \eta_k \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \sigma_k^2 \boldsymbol{I}_r)$

expectation. For $\mathcal{D}$, we have the following algorithm. Algorithm 25 runs until MM parameters converge with error ϵ.

In the above notation, we use $\mathbb{E}_{\boldsymbol{T}}[.]$ to denote the expectation over the entire sample of mixture distribution. In the E-step, l_k represents the probability of the sample $\boldsymbol{x}_i$ being generated from the k^{th} component as computed using the current estimates of parameters, and $(.)^+$ denotes the corresponding updated estimators.

5.4.2 Uniform Distribution Using Rejection Sampling

$\hat{\mathcal{N}}(\boldsymbol{x}|\eta_k, \boldsymbol{\mu}_k, \sigma_k^2)$ obtained from Algorithm 25 is used to specify rejection probabilities and make a uniform sample from G. In Theorem 5.4.2, we prove that the rejection sampling in stage three of Alg. 24 is almost uniform.

Theorem 5.4.2. *The sample S returned in Algorithm 24 is uniform over distinct values (E) of given dataset.*

Proof. We know that if v is a random variable whose support is a subset of $[0, 1]$, and a is a standard uniform random variable independent of v, then $\int_0^1 Pr(a \leq v) = \mathbb{E}[v]$. Let $\boldsymbol{\nu}$ be the

distribution parameters. The probability of acceptance of v is,

$$Pr(v \text{ is accepted}) = \int_0^1 Pr\big(a \leq \frac{\min_{\boldsymbol{x}\in E} f(\boldsymbol{x}|\boldsymbol{\nu})}{f(v|\boldsymbol{\nu})}\big) =$$
$$\mathbb{E}\big[\frac{\min_{\boldsymbol{x}\in E} f(\boldsymbol{x}|\boldsymbol{\nu})}{f(v|\boldsymbol{\nu})}\big] = \min_{\boldsymbol{x}\in E} f(\boldsymbol{x}|\boldsymbol{\nu}) \int_{-\infty}^{+\infty} \frac{1}{f(x|\boldsymbol{\nu})} f(x|\boldsymbol{\nu})dx = K.\gamma$$

The sampling procedure described produces draws from E with density uniform. We must show that the conditional distribution of v given that $a \leq \frac{\gamma}{p(v|\boldsymbol{\nu})}$,is indeed uniform in E;

$$Pr(x \leq v|v \text{ is accepted}) = \frac{Pr(v \text{ is accepted}|x \leq v).Pr(x \leq v)}{Pr(v \text{ is accepted}} =$$
$$Pr(v \text{ is accepted}|x \leq v).\frac{F(v)}{K.\gamma} = \frac{F(v)}{K.\gamma}.\frac{Pr(v \text{ is accepted}, x \leq v)}{F(v)} =$$
$$\frac{1}{K.\gamma}\int_{-\infty}^{v}\int_0^1 Pr(a \leq \frac{\gamma}{f(v|\boldsymbol{\nu})}, w \leq v) f(w|\boldsymbol{\nu})da\ dw$$
$$\frac{1}{K.\gamma}\int_0^1\int_0^v \frac{\gamma}{f(w|\boldsymbol{\nu})} f(w|\boldsymbol{\nu})dw\ da = \frac{v}{K} = \text{Unif}(0,k)(v)$$

The discrete case is analogous to the continuous case and we follow the same proof sketch. □

5.4.3 Correctness and Error Bound of Using Estimated Mixture

Theorem 5.4.3. *Given a finite dataset X which has the ξ-GMM property w.r.t an unknown density function $\mathcal{N}$ with parameters η_i, μ_i, σ_i for $i \in [1, \ldots, k]$ and $\gamma = \min \mathcal{N}$. Let E be the set of entities of X.Let $\mathcal{A}$ be as described in Alg. 18 with procedure $\mathcal{F}$ as described in Alg. 24. If $\mathcal{F}$ receives a sample of size $m > C' \frac{d^3(\log(k^2T)+\log(\frac{1}{\delta}))}{\eta_{\min}\epsilon^2}$ and $T = O(\log(1/\epsilon))$ as input, then $\mathcal{A}$ samples a set P according to a distribution $\mathcal{P}$ such that*

$$d_{TV}(\mathcal{P}(e), \mathcal{T}_X) \leq O(\epsilon/\gamma) + \xi$$

with probability atleast $1 - \delta - \frac{T}{k^{30}n^{C-2}}$ where C is the separation parameter for the spherical Gaussians. For EM algorithm initialization, the parameters provided in Thm. 5.4.1 are used.

In [129], it has been proved that if for each pair of Gaussians $\mathcal{N}(\boldsymbol{\mu}, \sigma\boldsymbol{I})$ and $\mathcal{N}(\boldsymbol{\mu}', \sigma'\boldsymbol{I})$, we know their means have distance $\Omega(\max(\sigma, \sigma')\sqrt{\log(\rho_\sigma/\eta_{min})})$ where $\rho_\sigma = \frac{max_{i\in[K]}\sigma_i}{min_{i\in[K]}\sigma_i}$ then with a good parameters initialization, EM algorithm with sample complexity

$$O(r\eta_{min}^{-1}\log^2(K^2T/\delta)/\epsilon^2)$$

can converge to optimal parameters with probability $1-\delta-T/n^{c-2}K^{30}$ where $T = O(\log(1/\epsilon))$ which means

$$\forall k \in [K] \;\; \| \hat{\eta}_k^{(T)} - \eta_k^* \|_2 \leq \eta_k^* \epsilon, \;\; \| \hat{\boldsymbol{\mu}}_k^{(T)} - \boldsymbol{\mu}_k^* \|_2 \leq \sigma_k^* \epsilon, \;\; \| (\hat{\sigma}_k^{(T)})^2 - \sigma_k^{*2} \|_2 \leq \sigma_k^{*2} \epsilon / \sqrt{r}. \quad (5.3)$$

Now, we approximate the error of the approximation.

Theorem 5.4.4. *Suppose a mixture of k d-dimensional Gaussian's f has parameters such that the separation of the means are $\Omega(c\alpha \max(\sigma, \sigma')\sqrt{\log(\rho_\sigma/\eta_{min})})$ with some given constant $c > 2$ and $\alpha = 2.297$. Suppose we use*

$$m \geq O\left(\frac{d\big(\log(K^2T/\delta)[2d + \epsilon + \alpha\beta] \big)^2}{\eta_{min}\alpha^2\epsilon^2} \right)$$

samples where $\beta = 0.0084$. Then, with a proper initialization, EM algorithm in $T = O(\log(\frac{2d+\epsilon+\alpha\beta}{\alpha\epsilon}))$ iterations approximation $\hat{f}$,

$$TV(f, \hat{f}) \leq \epsilon$$

with probability at least $1 - \delta - T/n^{c-2}K^{30}$.

Proof. For this goal, we know that the area between two normal distribution does not have close form, so we approximate it each normal distribution with Triangular distribution. We use L^2 norm for error so we have to solve. Since we have spherical assumption, we can decompose dimensions and solve the optimal point for each dimension independently.

$$\frac{d}{dx}\left[\frac{1}{2\pi} \int_{|x| \geq \alpha} e^{-x^2} dx + \int_{-\alpha}^{\alpha} \left(\frac{1 - |x/\alpha|}{\alpha} - \frac{e^{-x^2/2}}{\sqrt{2\pi}} \right) \right] = 0 \quad (5.4)$$

so we have $\alpha = 2.2975$ with maximum error of 0.042. Therefore, the best Triangular approximation for the normal distribution $\eta\mathcal{N}(\mu, \sigma)$ is between $[\mu - \beta, \mu + \beta]$ with $\beta = \alpha\sigma/\eta$, So we have to compute the maximum possible error which is the area between two Triangular distributions $[\mu - \frac{\alpha\sigma}{\eta}, \mu + \frac{\alpha\sigma}{\eta}]$ and $[\mu + \epsilon\sigma - \frac{\alpha(1-\epsilon)\sigma}{\eta(1+\epsilon)}, \mu + \epsilon\sigma + \frac{\alpha(1-\epsilon)\sigma}{\eta(1+\epsilon)}]$. Since the components can be considered independently so triangles separation follows the same well-separation property of the Gaussians. we choose approximation parameters in 5.3 (see Fig. 5.1) such that the triangular distribution makes the minimum overlap with respect to the triangle of the correct of the normal distribution.

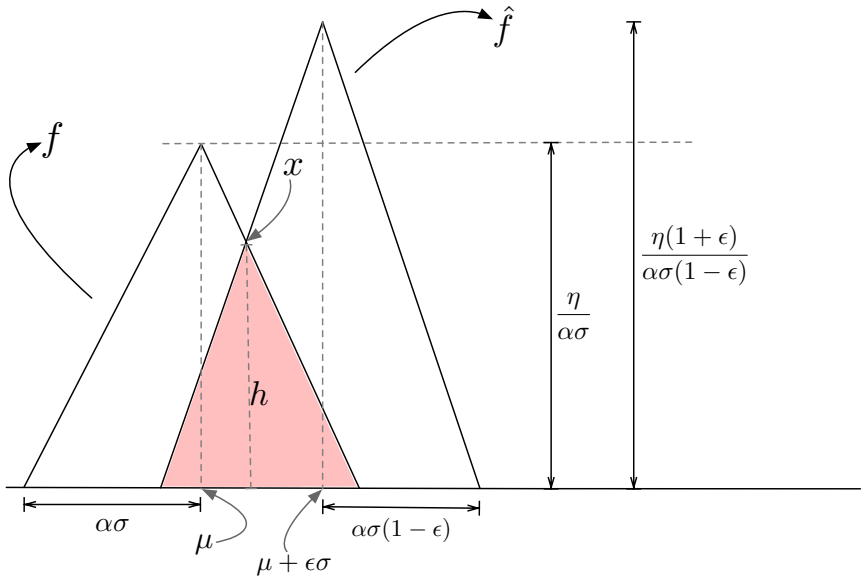


Figure 5.1: Triangular distribution and its worst-case approximation.

The total variation of i-th component is the area that true distribution and its approximation are not overlapped. Therefore, we need to compute the filled area(Fig. 5.1), A_i, then the error upper-bound is,

$$error_i \leq 2\eta_i + \eta_i\epsilon - 2A_i$$

First we should know that the mean of normal distribution and correspondingly the triangular distribution does not change the error, so we consider $\mu = 0$. To obtain the point x, we need intersect two lines that we can obtain by the given properties of distribution, so we obtain

$$y_1 = \frac{\eta(1+\epsilon)}{\big(\alpha\sigma(1-\epsilon)\big)^2}x - \frac{\epsilon\sigma\eta(1+\epsilon)}{\big(\alpha\sigma(1-\epsilon)\big)^2} + \frac{\eta(1+\epsilon)}{\alpha\sigma(1-\epsilon)}$$
$$y_2 = -\frac{\eta}{(\alpha\sigma)^2}x + \frac{\eta}{\alpha\sigma}$$

The intersection is,

$$x = \alpha\sigma\left(\frac{1 + \frac{\epsilon(1+\epsilon)}{\alpha(1-\epsilon)^2} - \frac{1+\epsilon}{1-\epsilon}}{\frac{1+\epsilon}{(1-\epsilon)^2} + 1}\right)$$

Now, we can compute h, the height of overlapped area.

$$h = \frac{\eta}{\alpha\sigma}\left[\frac{2 + \big(1 - \frac{1}{\alpha}\big)\epsilon - \big(1 + \frac{1}{\alpha}\big)\epsilon^2}{2 - \epsilon + \epsilon^2}\right]$$

The base of the triangle is,

$$b = \big((2-\epsilon)\alpha - \epsilon\big)\sigma$$

Therefore, the area is,

$$\begin{aligned} A = \frac{1}{2}bh &= \frac{1}{2}\sigma\big((2-\epsilon)\alpha-\epsilon\big).\frac{\eta}{\alpha\sigma}\left[\frac{2+\big(1-\frac{1}{\alpha}\big)\epsilon-\big(1+\frac{1}{\alpha}\big)\epsilon^2}{2-\epsilon+\epsilon^2}\right] \\ &= \frac{1}{2}\eta\big(2-\epsilon-\frac{\epsilon}{\alpha}\big)\left[\frac{2+\big(1-\frac{1}{\alpha}\big)\epsilon-\big(1+\frac{1}{\alpha}\big)\epsilon^2}{2-\epsilon+\epsilon^2}\right] \end{aligned}$$

We assume $\epsilon^2 \approx 0$, so we have,

$$A = \frac{1}{2}\eta\big(2-\epsilon-\frac{\epsilon}{\alpha}\big)\left[\frac{2+\big(1-\frac{1}{\alpha}\big)\epsilon}{2-\epsilon}\right] = \frac{1}{2}\eta\left[2+\big(1-\frac{1}{\alpha}\big)\epsilon-\frac{2\epsilon}{\alpha(2-\epsilon)}\right]$$

Therefore, the overlapped area of component i-th is $A_i = \frac{1}{2}\eta_i\big[2+\big(1-\frac{1}{\alpha}\big)\epsilon-\frac{2\epsilon}{\alpha(2-\epsilon)}\big]$. Since we are given a proper separation between components, the total error is the sum of each component error. Therefore, we have,

$$\begin{aligned} TV(f,\hat{f}) = \sum_i error_i &\leq \sum_i 2\eta_i + \eta_i\epsilon - 2A_i = 2+\epsilon-2\sum_i A_i \\ &= 2+\epsilon-\sum_i \eta_i\big[2+\big(1-\frac{1}{\alpha}\big)\epsilon-\frac{2\epsilon}{\alpha(2-\epsilon)}\big] \\ &= 2+\epsilon-\big[2+\big(1-\frac{1}{\alpha}\big)\epsilon-\frac{2\epsilon}{\alpha(2-\epsilon)}\big] = \epsilon\big[\frac{2}{\alpha(2-\epsilon)}+\frac{1}{\alpha}\big] \\ &= \frac{4\epsilon}{\alpha(2-\epsilon)} = O(\epsilon). \end{aligned}$$

When ϵ is small the triangle is an lower bound of the actual overlap between two normal so the error we get is an upper bound. For dimension d, because our model is spherical, the joint distribution is the product of distribution of each dimension. Therefore, the error of the tale of Gaussian decreasing by increasing the dimension when the dimension is large the data concentrates around the mean. For the right and left tale, if we have ϵ movement to right, using Taylor expansion, we have $0.0084\epsilon + 0.0039\epsilon^2$ for the area between normal and its approximation, so

$$TV(f,\hat{f}) \leq d\left(\frac{4\epsilon}{\alpha(2-\epsilon)}\right) + \beta\epsilon$$

where $\beta = 0.0084$. We determine $\epsilon' = d\big(\frac{4\epsilon}{\alpha(2-\epsilon)}\big) + \beta\epsilon$ so $\epsilon = \frac{\alpha\epsilon'}{2d+\epsilon'+\alpha\beta}$. We replace this into the sample complexity of parameters, $O(d\eta_{min}^{-1}\log^2(K^2T/\delta)/\epsilon^2)$ from [129], the we achieve the result. □

Corollary 5.4.1. *The expected number of draws in rejection sampling (Alg. 24) is less than* $\frac{m}{K.\gamma}$

Proof. We know that $\min_{\boldsymbol{x}\in E} f(\boldsymbol{x}|\boldsymbol{\nu}) \leq \frac{1}{K}$.From the proof of the Theorem 5.4.2, $Pr(v \text{ is accepted}) = K.\gamma$, so if we consider each success as independent geometry distribution then the average number is less than m-times of largest geometry distribution success, so it is $\frac{m}{K.\gamma}$. □

5.5 Experimental Results

In this section, experiments have been divided into two parts, the experiments that show behaviours of our framework and experiments compare our estimator to Sample-and-Clean [197] on some real datasets. For evaluating the performance of different methods over our datasets, we define $Error = |RealAvg - EstimatedAvg|/RealAvg$. We repeated each experiment until, we see convergence in the average of the errors. We use five real datasets which they are publicly available.

TPC-H Dataset[3] It contains 1.5GB TPC-H benchmark3 dataset (8,609,880 Records in lineitem table). The line item table schema simulates industrial purchase order records. We used this dataset to model errors where the purchase orders were digitized using optical character recognition (OCR). We similar to Sample-and-Clean [197] randomly duplicated 20% of tuples with the following distribution: 80% one duplicate, 15% two duplicates, 5% three duplicates.

Sensor Dataset[4] We also applied our approach to a dataset of indoor temperature, humidity, and light sensor readings in the Intel Berkeley Research Lab. The dataset is publicly available for data cleaning and sensor network research from MIT CSAIL5.

Publications Dataset This dataset is a real-world bibliographical information of scientific publications [64]. The dataset has 1,879 publication records with duplicates. The ground truth of duplicates is available. To perform clustering on this dataset we first tokenized each publication record and extracted 3-grams from them. Then, on 3-grams we used Jaccard distance to define distance between two records.

E-commerce products I[5] This dataset contains $1,363$ products from Amazon, and $3,226$ products from Google, and the ground truth has $1,300$ matching products.

E-commerce products II[6] This dataset contains 1,082 products from Abt, and 1,093 products from Buy, and the ground truth has 1,098 matching products.

[3] http://www.tpc.org/tpch

[4] http://db.csail.mit.edu/labdata/labdata.html

[5] https://dbs.uni-leipzig.de/en

[6] https://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution

Restaurants Dataset[7] The fifth dataset is a list of 864 restaurants from the Fodor's and Zagat's restaurant guides that contains 112 duplicates.

5.5.1 Effect of Sampling Size and Dataset Balance

We evaluate our sampling method for balance dataset under different sample sizes and perform this evaluation for different duplication ratios. For this experience, we use *TPC-H* dataset and we inject duplicated values manually. Table 5.2 shows that for different duplication rates, the method has a similar behaviour and the error decreases as the sample size increases, the error is strictly smaller than the theoretical upper bound. In Table 5.3, we generate an arbitrary distribution for entities frequencies. From Table 5.2 and Table 5.3, as Thm 5.2.1 suggests, we confirm that the imbalance dataset weaken the uniform sample generation.

dup	0.01	0.02	0.04	0.06	0.08	0.1
0.1	2.12[#]	1.64	1.41	1.23	1.11	1.16
0.15	3.12	2.03	1.84	1.92	1.76	1.69
0.2	3.14	2.24	1.97	1.84	1.80	1.74
0.25	4.26	4.02	3.65	2.93	2.61	2.17
0.3	5.21	4.94	4.57	3.84	3.33	2.89

[#] Values $\times 10^{-3}$.

Table 5.2: The precision changes in different sample sizes under generative process for duplication with uniform distribution. By increasing the duplication ratio, the error of our framework increases. *dup* presents duplication rate.

dup	0.01	0.02	0.04	0.06	0.08	0.1
0.1	2.58[#]	2.03	1.69	1.38	1.26	1.19
0.15	3.66	3.17	2.50	2.03	1.89	1.79
0.2	4.62	4.14	3.61	3.08	2.59	2.42
0.25	5.32	4.88	4.37	3.84	3.29	2.73
0.3	5.94	5.45	4.99	4.73	4.03	3.79

[#] Values $\times 10^{-3}$.

Table 5.3: Our estimator is independent from duplication distribution of entities. The datasets that considered has non-uniform duplication over their entities.

[7] http://www.cs.utexas.edu/users/ml/riddle/data.html

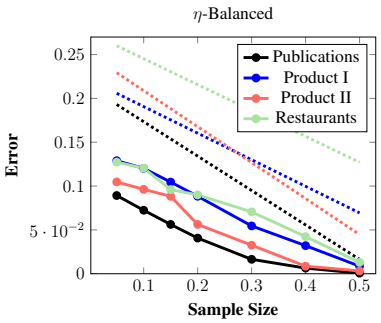


Figure 5.2: Applying balanced method on real datasets. Dotted lines show the linear regression of theoretical bounds.

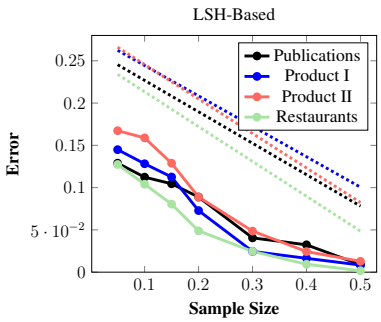


Figure 5.3: Applying LSH method on real datasets. Dotted lines show the linear regression of theoretical bounds.

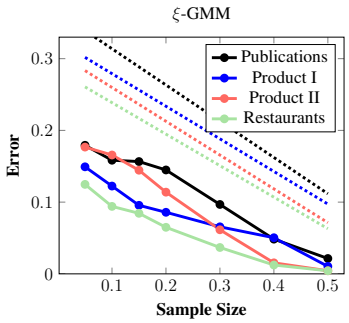


Figure 5.4: Applying GMM method on real datasets. Dotted lines show the linear regression of theoretical bounds.

5.5.2 Our Methods Over Real Datasets

We conducted a set of experiments on real datasets to evaluate our method and evaluate our theoretical bounds. We set $\delta = 0.9$ and obtain all information each method needs directly from data. For each sample size, we repeat for 100 times and calculate the average of the errors. Fig 5.2 shows the result of the Alg. 18 for four real datasets, and the dashed line is linear regression of the upper-bound suggested by Thm. 5.2.1. Figure 5.3 shows the result of Alg. 23 and the dashed lines show the upper bound given by Thm. 5.3.3. In Fig 5.4, we used the Alg 24, and computed the upper bound by using Thm. 5.4.3. As we know the assumption of Gaussian prior is stronger than LSH and LSH is stronger that balanced dataset. Gaussian method on a random dataset has weaker performance, which can approved by comparing Fig 5.4 with Fig 5.3 and Fig 5.2.

5.5.3 Comparison to Other Methods Over Real Data

In this section, we compare out methods to *RawSC* and *NormalizedSC* in Sample-and-Clean [197] on real and synthetic datasets.

- **RawSC Estimation**: Propose a correction function for count, sum, and average estimator and then return the average of corrected element of the sample with a confidence interval.
- **NormalizedSC Estimation**: Similar to RawSC but uses the difference between correct value and unclean version in the sample to justify the average of the measure on the unclean dataset.

We use samples with size 30% of the dataset, and measure $accuracy = 1 - error$. We use the optimal blocking function for *RawSC* and *NormalizedSC*.

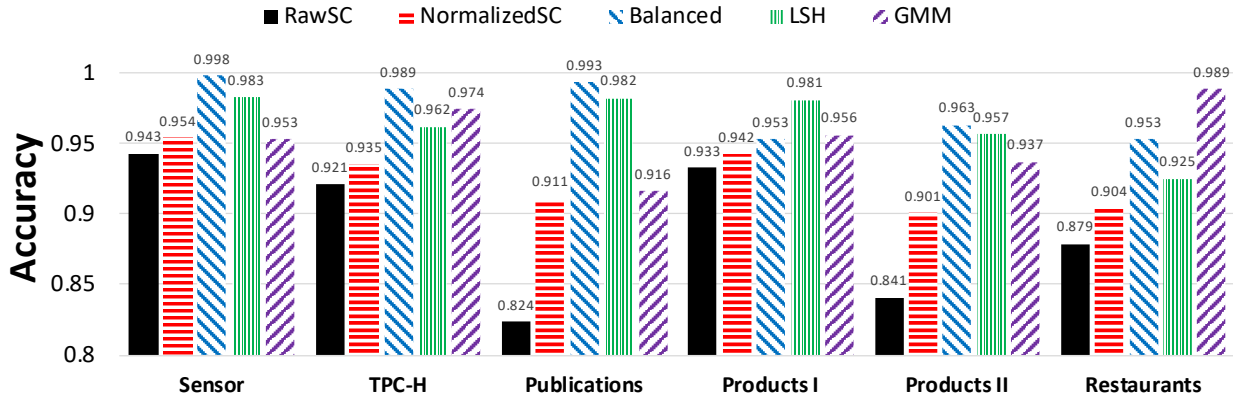


Figure 5.5: $accuracy$ of proposed methods and methods in [197]. In Sensor and TPC-H, duplicated records are added manually.

As you can see in Figure 5.5, $accuracy$ of our methods outperforms the the-state-of-art sample cleaning framework. In each method, we computed the assumed information as the method input. For each experiment, we computed the average of 100 experiments.

5.6 Conclusion and Open Problems

Obtaining correct information from data with repetitive records is an important problem. Deduplicating the entire dataset is computationally expensive. We solve the problem by approximating the uniform distribution over the clean data. Generating uniform sample from such a data is not always possible. Knowing additional data properties can make the problem feasible and solvable. We consider three approaches that work under different circumstances. These methods return samples that can be used for any downstream analytical purpose, because it has the same properties as a uniform sample from the cleaned version of the data. There are some open-ended questions in our research. One direction of research can explore other weaker and/or natural assumptions under which the problem is still solvable. Another direction of research involves providing tighter bounds and/or lower bounds for the methods presented in this work. Another important question, is if it possible to verify that a given dataset satisfies any of the niceness conditions. For example, is it possible to estimate the value of η for a balanced dataset (Section 5.2). Moreover, We have introduced a method which provides a lower bound estimate for the balance parameter η. However, the number of samples needed (upper and lower bound) to obtain this

estimate is an open question. Similar questions can be posed for the other niceness conditions introduced in this work.

Chapter 6

Record Fusion via Inference and Data Augmentation

In a variety of domains, people rely on multiple sources to fulfill their information requirements. However, diverse information sources often provide contradictory data, some being out-of-date, inexact, or incorrect. The increasing demand to ingest and acquire a large number of heterogeneous data sources via inexpensive connectors has been the driving force behind many studies in the field of entity consolidation and data integration [23, 60–62, 94, 132, 134, 204]. An integrated dataset can have a considerable positive impact on the quality of downstream analytics [27]. For the rest of the chapter, we will refer to aggregating multiple records into a unified representation by "Record Fusion".

A good example of record fusion is *the golden record problem*; after deduplicating [38,51,65] a dataset, a set of record clusters is created, which needs to be consolidated to produce one representation. Another example is *data fusion*, which sometimes referred to as "Single-Choice Tasks" in crowdsource community [208], where the task is to aggregate multiple sources that contain information about the same set of entities, with possibly conflicting attribute values [60, 62, 134, 204].

Example 6.0.1. *In Table 6.1, the data have been gathered from* 4 *different databases. Each database can be viewed as a source that makes a claim about the actual value of real-word entities. In this example, there are three clusters, where each tuple is generated from a unique source in the cluster. The correct values of each cluster/attribute are shown in* ***Bold****. However, pieces of information that are gathered from different sources can be conflicting. For instance, the first source of cluster* c_3 *claims that the Amazon headquarters is located in the state "WA", while, the last source claims that the state of the entity is "New York". In the record fusion*

problem, we want to resolve such conflicts and obtain the correct values for the attributes of each real-world entity.

Cluster Id	Company	ZIP	City	State	Webpage
c_1	Google	**94043**	**Mountain View**	**CA**	google.ca
c_1	**Google Inc.**	940 43	Mountains View	California	**Google.com**
c_2	Microsoft	**98052**	**Redmond**	**WA**	MS.com
c_2	MICROSOFT	**98052**	Seattle	Washington	**msn.com**
c_2	**Microsoft Corp.**	56419	**Redmond**	**WA**	**microsoft.com**
c_3	Amazon	<*Null*>	**Seattle**	**WA**	amazon.co
c_3	**Amazon Inc.**	**98109**	**Seattle**	<*Null*>	amazon.ca
c_3	AMZN INC	**98109**	**Seattle**	Washington	**amazon.com**
c_3	Amazon	98052	<*Null*>	New York	**amazon.com**

Figure 6.1: An example of clusters with conflicting values for each cluster-attribute.

Many probabilistic fusion methods depend on the availability of the records source information (or source information in short) as part of the schema, and often build models to estimate the "trustworthiness" of sources [8, 48, 50, 60, 63, 116, 175, 194, 199, 202, 204, 205, 209]. However, calculating the trustworthiness of a source is not trivial and can significantly vary in different parts of the data. For example, in some instances, limited ground truth is used for calculating an initial estimation of the sources "trust score" [59, 79]. In other scenarios, all sources might be equally good/bad and these source-dependant methods are reduced to simple majority voting and fail to take other effective signals into account. Furthermore, in many real-world scenarios, we may not have any explicit source information altogether, e.g., in deduplicating a single-source dataset as we show in the following example.

Example 6.0.2. *Figure 6.2 shows a tabular data containing tuple id's, person names, occupation, and address and illustrates the task of a typical deduplication approach. A deduplication technique produces a table with the clustering of those records, where each cluster refers to the same real-world entities. The process of finding the true records for entities is called the "golden record problem". In this setting, there is no information about sources, as all records might have come from the same source.*

In the golden record problem, the concept of "source" might not be even applicable since the data from one source can have duplicates. Current approaches either use techniques such

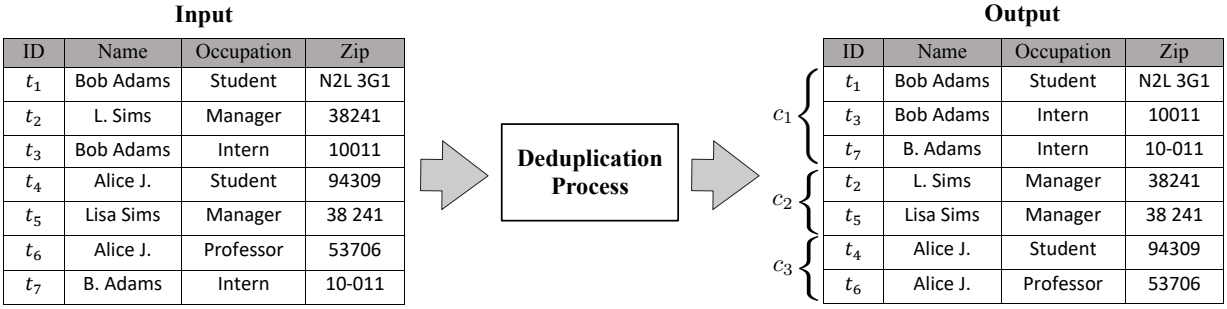


Figure 6.2: A typical deduplication task.

as heuristic aggregation rules, majority vote or involve humans annotators to resolve conflicts via learned transformations [51]. In this case, We show experimentally that naïve methods like choosing the value provided by the majority vote of sources often lead to inaccurate and unreliable results (see Section 6.6.2). Human-in-the-loop approaches have multiple challenges, including (1) they can be prohibitively expensive and time-consuming depending on the number of clusters, and difficult to resolve conflicts; (2) they often assume users do not make mistakes, or they need to involve multiple voters introducing new types of conflicts that need additional resolution mechanisms; and (3) a global process to aggregate human decision in a consistent way may still be needed. For example, when the human votes are conflicting, the challenge to conform their suggestions and selecting the correct vote still remains. In summary, the common challenge between "data fusion" and "golden record" is fusing multiple records into a smaller set of true records, which we call it "record fusion".

Recently, multiple principled machine learning (ML) approaches have been proposed for the data cleaning problems [42, 96, 177]. In this chapter, we show that an ML inference model that takes into account all available signals (statistical properties and integrity constraints) can solve the record fusion problem. The problem shares some technical challenges with these ML-cleaning approaches. These include: (1) designing a rich representation to capture the structure of the data and all the signals needed to impute the fused values; and (2) learning from noisy and partial information. While we show how to solve these challenges in the clustered data settings, an additional new challenge that we also address in this work is generating enough "cluster" training data (via data augmentation) to train the rich representation model.

6.1 Approach and Technical Challenges

We propose a learning framework for record fusion based on weak supervision approach [173]; our framework automatically fuses records in the clusters by leveraging all related information,

i.e., integrity constraints and data rules, quantitative statistics, and source information if available. Our ML-approach addresses multiple technical challenges:

I [Model] To infer the correct values, an expressive model is required to capture all data characteristics. Specifically, the representations models should cover all data context features that describe the distribution governing the generation of the dataset.

II [Training Data] An expressive model needs enough training data to achieve an acceptable level of confidence. However, gathering enough labeled examples is a costly, error-prone, and sometimes impossible process. In addition, each training example is a cluster of records, which needs different labeling mechanism. Hence, we need a way to automatically generate additional training data from the limited ground truth available.

III [Partial Information] For designing a reliable ML-approach, we need to have complete information to generate appropriate features. Labeled data gives us limited observations of the whole truth. The model need to learn from partial and noisy estimates of the correct values and an appropriate mechanism to improve prediction iteratively.

We address the aforementioned challenges and propose an ML-based framework for record fusion. The framework iterates over multiple modules to learn the representation of the input data, generate enough training data to learn the model parameters, and produce a probabilistic representation of the final fused record (Section 6.2.2). We highlight the following concrete contributions:

(1) We introduce representation models that capture various characteristics of the data, with or without source information, providing a rich input to the inference model for predicting the values of the fused record (Section 6.3). **(2)** We introduce a novel generative data augmentation process, which generates additional artificial *training data clusters* for learning the model parameters. This process resolves the problem of limited training data (Section 6.4). **(3)** To address the partial information challenge, we introduce an iterative mechanism that works as a *Hard-Assignment Expectation Maximization* approximation scheme to estimate model parameters and provide robust estimates (Section 6.5).

Finally, we evaluate our framework on multiple real-world datasets, where we demonstrate its ability to determine the correct values for real-world entities, and we show that probabilistic inferences with sufficient training data are a valid modeling tool for the record fusion problem (Section 6.6). We highlight relevant previous works in Section 6.7.

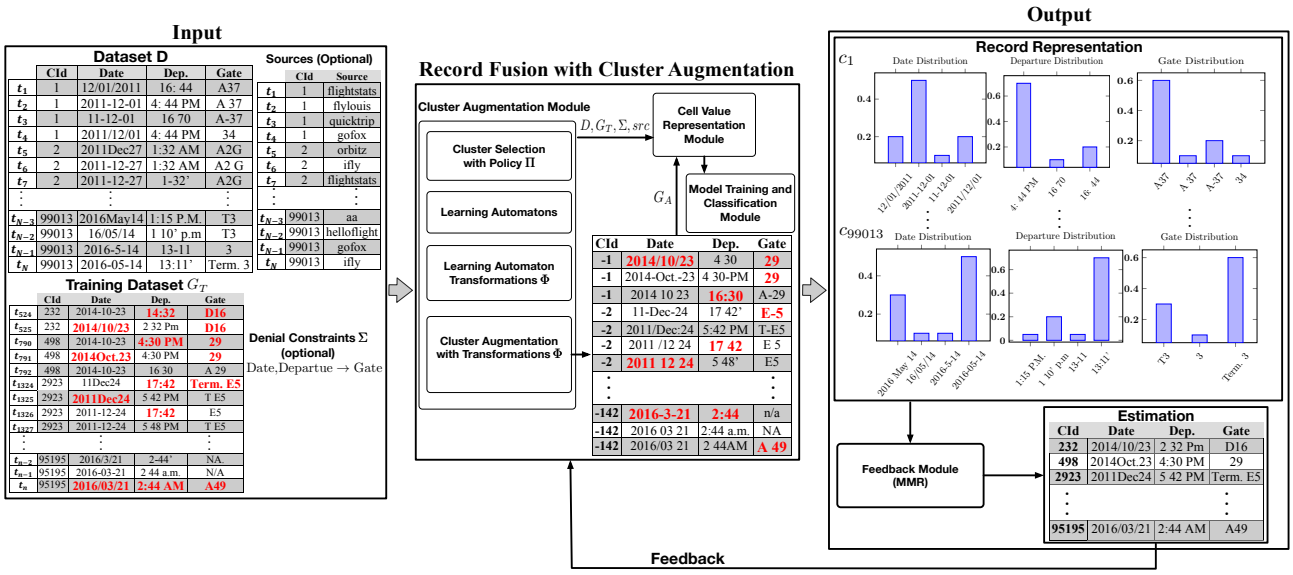


Figure 6.3: Overview of Record Fusion system. The system can learn from heterogeneous formats of correct data. For example, the correct Date is shown in different formats and the inference model still can approximate the correct record representations.

6.2 Problem Definition and Solution Overview

We define "Record Fusion" as the process that takes a collection of clusters as input, where each cluster contains a set of tuples and outputs a record representation for each cluster. The record representation follows a probabilistic semantics that we formally describe in the following section.

6.2.1 Problem Definition

The input is a relational dataset D, with a clustering $\mathcal{C}$ that defines partitions over tuples in D. D has a schema $S = R \cup \{cid\}$, where $R = \{r_1, r_2, \ldots, r_N\}$ denotes attributes of D, and cid identifies the cluster of each tuple. Let Cluster $c_i \in D$ denotes the set of tuples in D with the same $cid = i$, and $c_i(r_j)$ denotes the set of unique values of Attribute $r_j \in R$ in Cluster $c_i \in D$.

Let G be a data set with the same schema as D that provides a probabilistic representation of the clusters in D. For each Tuple $g_i \in G$ that corresponds to Cluster $c_i \in D$ and Attribute $r_j \in R$, $g_i.r_j$ is a probability distribution $V^*_{c_i,r_j}$ over the values $c_i(r_j)$. Finally, besides D, we are also provided with a training dataset $G_T \subset G$. $V^*_{c_i,r_j}$ is given for each Attribute $r_j \in R$ and every Tuple $t_i \in G_T$. Optionally, a set of data rules expressed as denial constraints Σ are provided. We

define $G_U = G \backslash G_T$ as the set of unknown true record representations. The problem definition is stated as follows:

Definition 6.2.1 (Record Fusion)**.** *For each Cluster $c_i \in D$, Record Fusion produces a unified representation tuple $g_i = (f_{i,1}, f_{i,2}, \ldots, f_{i,N})$, where $f_{i,j}$ is a probability distribution over values $c_i(r_j)$. Specifically, $f_{i,j}$ is an estimator for the ground truth $V^*_{c_i,r_j}$ that minimizes a given distance function $\Delta(f_{i,j}, V^*_{c_i,r_j})$ conditioned on the observed input data D, the partial ground truth G_T, and the set of data rules Σ (if given).*

Since the record fusion provides a probability distribution over possible values in each attribute of each cluster, there are multiple ways to consume it. One way is to provide the most likely value(s) of each attribute in each cluster as our best approximation for the golden record problem.

Definition 6.2.2 (Marginal MAP Record (MMR))**.** *Given the output of Record Fusion (Definition 6.2.1) for a cluster $c_i \in D$ and a record representation $g_i = (f_{i,1}, f_{i,2}, \ldots, f_{i,N})$; $\hat{g}_i = (\hat{v}_{i,1}, \hat{v}_{i,2}, \ldots, \hat{v}_{i,N})$ can be produced, where $\hat{v}_{i,j}$ is the maximum likelihood values from the distribution $f_{i,j}$.*

Note that if the partial ground truth $G_T \subset G$ was provided as a set of correct values for each attribute and tuple in G_T, we can induce a uniform distribution over these values. Note also that one of the attributes $r_i \in R$ can correspond to the source of records in D.

6.2.2 Solution Overview

We now present the architecture of our framework. Theses modules are used (i) to learn a generative process to produce more training data by adding artificial clusters; and (ii) use an iterative approach to learn a representation model of signals jointly with a classifier that is used to find record representations of the input dataset. Our framework takes as input a noisy dataset D, a training data G_T, (optionally) a set of denial constraints Σ, and (optionally) source information. The four core modules are described in follow:

Module 1: Representation Learning This module combines different properties and signals to capture the distribution of true record representation, which maps each cell in D to a fixed-dimension real-valued vector. To obtain this vector, we concentrate on the output of different representation models, each of which targets a specific context (i.e., attribute, tuple, or dataset context) (see Section 6.3.1).

Module 2: Model Training and Classification This module is responsible for training a multi-class classifier, which given the representation of cells of a cluster, generates a distribution with the size of the distinct values of each attribute of clusters (see Section 6.3.2).

Module 3: Cluster Augmentation Due to the need for training data, this module learns a generative process to generate additional training data by sampling and transforming the correct representation records, G_T. The output of this module is a set of additional clusters G_A that are legitimate to be used for training fusion models parameters. This module generally can be used for any fusion algorithm that uses statistical properties of data. (see Section 6.4).

Module 4: Feedback Due to learning with noisy and partial information (correct predictions of other attributes of cluster), we perform multiple iterations of a *Hard-Assignment Expectation Maximization* process [122] (see Section 6.5).

An overview of how the different modules are connected is shown in Figure 6.3. First, Module 1 augments training data with additional artificial clusters. Then, Module 2 grounds the representation model of our record fusion model. Subsequently, the representation model is connected with the multi-class classifier model in Module 3. After generating a record representation, the model gets feedback from Module 4, and so it can modify the representation and the predictions.

6.3 Representation of Data Cells

From Definition 6.2.1, given D and the training data $G_T \subset G$, we want to estimate the probability distribution $V^*_{c_i,r_j}$ for each cell $g_{i,j} \in G \backslash G_T$. To estimate $V^*_{c_i,r_j}$, we learn a representation model Q that approximates data distribution properties on the attribute, tuple, and cluster level, and couple them with a prediction model M that maps a value representation based on Q to a probability distribution. The model Q builds upon a variety of features computed either based on the initially observed values (static features) or based on intermediate inferences of the true cluster values (dynamic features). We require Q to be such that the likelihood of true record representations in G_T given Q to be maximum, while the likelihood of other record representations be low. This property is necessary for a model M to assign high-probability to correct cluster values and low-probability to incorrect cluster values. We rely on representation learning techniques to learn Q jointly with model M.

6.3.1 Representation Models

The representation model Q needs to capture the statistical characteristics of cells inside each clusters in attribute-level, tuple-level, and cluster-level contexts. Q is formed by concatenating the outputs of different models and signals, and maps each cell D_{t_i,r_j} of tuple t_i and attribute r_j to a fixed-dimension real-valued vector $h_{t,r} \in \mathbb{R}^d$ with dimension d. Next, we review the

representation models used in each of these three contexts. It should be noted that the introduced models correspond to a bare-bone set that captures all settings and is currently implemented in our prototype. Our architecture can trivially accommodate additional models (e.g. auto-encoders) or more expressive variants of the current representation models.

Attribute-level Representation Models for this context capture the distributions governing the values and format for an attribute. For example, the value "New Yorx" in attribute *City* has a low frequency and with respect to language models, also obtains low scores. Separate models are used for each attribute $r_i \in R$ in dataset D. We consider three types of models: (1) *Format model*, which captures the probability distribution governing the format of the values. In our implementation, we consider an n-gram model for each attribute that captures the format sequence over the cell value (empirically, we set $n = 2$). Each n-gram is associated with a probability learned directly from the dataset by using a reduced model (for example we replace all the letters with A, all the numbers with N, special characters represent themselves, etc.). For each value $c_i(r_j)$, we calculate a probability by taking the product of the likelihood of each n-gram constructing from the reduced model string value. (2) *Running cluster-value feature* , as our model uses an iterative algorithm, this signal captures the compatibility of the attribute value in a record with the running cluster value in $c_i(r_j)$ (after inference). For each value in $c_i(r_j)$, we have a flag that indicates if it was the predicted value in an earlier iteration. In the first iteration, the current Running cluster-value feature is set to zero for each value in $c_i(r_j)$ to prevent any bias to the learning parameters. (3) *Character and token sequence* is a distance metric on the character-based representation of each value in D with the average representation of all the values for each entity. We train a word-embedding model for each attribute, where each column in the dataset D is considered to be a corpus and each possible value a sentence. We decompose each possible value into its respective characters, and we learn an embedding vector for each value. Furthermore, after the model is trained for each attribute of an cluster, we compute the average embedding vectors of Attribute r_p in Cluster c_q, $avg_{c_q(r_p)} = \frac{\sum_{j \in c_q(r_p)} embedding_j}{|c_q(r_p)|}$. Finally, we calculate the distance of the embedding vector of each value from the average embedding vector using the following equation, $h_i = -2 \times \cos(embedding_i, avg_{c_q(r_p)}) + 1$ and if $h_i > 0$ we set $h_i = h_i/3$ to normalize the feature value.

Tuple-level Representation: Models for this context capture the joint distribution of different attributes. These representations perform weak predictions for the possible values of each cell. For example, if the value "New York" in attribute *City* appeared more often with "United States" of attribute *Country*, the Tuple-level signals reveal this effect. We consider two types of models: (1) *Co-occurrence model* that captures the empirical joint distribution over pairs of attributes. As we have multiple initial values for each attribute, we use the values that are predicted in a previous iteration for each attribute in our calculations. This model is updated in every iteration. In the first iteration, we select the majority vote value as the initial value. Finally, it should be

noted that we use one co-occurrence feature per pair of attributes. (2) *Vote model* that captures the empirical distribution of the values $c_i(r_j)$ of Attribute r_j associated with Cluster c_i. These can be learned directly from the input dataset D. This representation is a scalar, which is the empirical probability of the values in each cluster.

Dataset-level Representation: Models for this context capture a distribution that governs the compatibility of tuples and values in the dataset D. Specifically, any functional dependency that is valid on our correct records, Dataset-level signals capture this data integrity rule. We consider three types of models: (1) *Neighborhood-based representation* that is a distance metric on the neighborhood-based representation of each value in D with the average representation of all the values for each entity. We train a word-embedding model, where the input dataset D is considered to be a document, and each tuple in D is a sentence. As we mentioned in *character and token sequence* signal, after the model is trained for each attribute of a cluster, we calculate the average embedding vector of all the values for this specific attribute-cluster pair. Finally, we can calculate the distance of the embedding vector of each value from the average embedding vector by using the equation that we described in *character and token sequence* signal. In our dataset, if the source information is given we can follow the same procedure. The semantic of the source information signal is the level of irregularity of the given information of a source. (2) *Source model* captures the source from which each value in $c_i(r_j)$ came from. Specifically, for each value, we have an indicator that specifies the source that it came from. This signal is available if the source information is provided. (3) *Constraint-based models* capture the number of violations of every possible value with the predicted values that compose the current cluster values table. For each denial constraint in Σ (if given), we create the equivalent relaxed denial constraints [136, 177], and for each random variable (pair of cluster-attribute) and values in $c_i(r_j)$, we have one feature per denial constraint that captures the number of violations. As we have multiple initial values, we use the values that are predicted in a previous iteration for each random variable in our calculations. This model is updated in every iteration, and for the first iteration, we consider the most frequent value as the initial value. The outputs of all signals and representations are concatenated into a single vector that is given as input to classifier M. To achieve a high-quality record fusion model, features from all contexts need to be combined to form model Q.

6.3.2 Record Representation Distributions

The classifier M uses a dense layer followed by *Softmax* to find record representations for random variables in the dataset D. In this setting, for each attribute $r_j \in R$ and cluster $c_i \in D$, we consider a random variable v_{c_i,r_j} which takes values from $c_i(r_j)$. Let Z denotes random variables for attributes in R and clusters in $D \cup D_A$ where D_A is an additional artificial clusters which

is generated by augmentation module (Section 6.4). The training process sets the dense layer parameters θ to values that maximize the likelihood, $\mathcal{L}(\theta) = \log P(G_T \cup G_A|Z;\theta)$, where G_A is the set of augmented clusters (see Section 6.4). The model uses random variables that are observed in the training data G_T and augmented training data G_A as labeled examples to learn the parameters θ. After obtaining the record representations, probabilistic inference is used to estimate the true values of tuples in G_U. To this end, we rely on an Expectation-Maximization (EM) algorithm to learn the distribution (given the unobserved random variables). To perform EM we set the unobserved cluster values to their maximum a posteriori (MAP) values.

More importantly, we calibrate the confidence of the predictions of M using *Platt Scaling* [91, 164] in order to remove the false prior effect of the artificial training clusters that we used as training data. Specifically, an additional linear layer is added after *Softmax* with size equal to the maximum random variables domain, $\max_{c_i \in D, r_j \in R} |c_i(r_j)|$. The parameter of this layer are learned by using data from the original distribution, G_T after θ have been learned and fixed. In prediction, we use all layers as an end-to-end entity.

6.4 Data Augmentation

We propose a data augmentation approach to leverage the amount of training data available in the training phase of our model by learning a generative process. The augmentation process introduces new clusters in the dataset, by transforming an existing cluster to another cluster format of values. The statistical properties of Attribute r in Cluster c reflect signals like frequency, co-occurrence, and column-based language models. Thus, assuming that the generative process has fixed parameters during the cluster generation, we can observe an empirical distribution $\hat{I}_{c_i,r_j}$ for Cluster c_i and Attribute r_j in the original data. Using $\hat{I}_{c_i,r_j}$, we can generate a different cluster c_i' that gives us a new format for the incorrect values in c_i, and c_i' keeps all the statistical properties of c_i. This approach helps us discriminate between the actual distribution of records and other possible distributions of the cluster in more general forms. A format translation process which generates these new values is important for the performance as biases the system to identify syntactic variations and does not focus only on instance level variations. Hence, it promotes generalization. Our goal is to find a generative process that produces valid values based on a given Cluster c_i. Since changing the source authority introduces bias with format translation, we suggested a separate setup for generating source information attribute (see Section 6.4.4) and this approach applies on attributes other than source information.

We present our augmentation policy Π in Section 6.4.1. In Section 6.4.2, we discussed how to design a format translator function. Generating a cluster with a new format is described in

Algorithm 26: Overview of Data Augmentation

Input: Original cluster c_i with $g_i \in G_T$, policy Π
Output: Augmented cluster c_i'

1 Sample from $D \backslash c_i$ with policy Π to get candidate format;
2 Learn format translation process Φ;
3 Apply Φ on c_i to produce c_i';
4 [OPTIONAL] Add source information to the generated cluster

Section 6.4.3. Finally, we describe how to add the source information attribute to augmented clusters in Section 6.4.4.

6.4.1 Augmentation Policy

We pick a cluster following a sampling policy over the initial clusters and generate a new cluster of values with a new representation format. We have a set of policies $\Pi = \{\Pi_c, \Pi_{c_t}, \Pi_a, \Pi_v\}$, which supports the training process by selecting the clusters that in prediction have lowest confidence. The policy selects two clusters, an original cluster c and a target cluster c_t, using the sampling policy Π_c and Π_{c_t} respectively. Π_c selects the first cluster which is used to generate a new artificial cluster c' from its values and Π_{c_t} selects the cluster that the algorithm wants to capture its format for cluster c. As we said, we use the output of the running model M to select the clusters with lowest confidence in predictions. We select cluster c_i with probability $1 - \min_{r_j \in R} \ \min_{v' \in c_i(r_j) \backslash \arg\max_{c_i(r_j)} M(v)} |M(v') - \max_{v \in c_i(r_j)} M(v)|$ which is the gap between the two most probable values, and it has direct relationship to the model confidence. In the first iteration (Section 6.5), the model is not trained, so we select the original cluster(c) randomly. For each value in c, a new c_t is selected randomly, because we assume no prior information. Π_a is the selecting policy of an attribute among the attributes of c. In this work, to prevent generating inconvenient formats, we use the same attribute of the value currently is selected to change in c. Π_v is the policy of choosing a value among distinct values in the selected attribute of c_t.

6.4.2 Format Transformation

To produce a translation function of string formats, we need to generalize string values to a more format-informative string and then learn a function that takes as input the original format and produces a string in the target format. We also need to determine a policy which will apply this translation to a corresponding string.

Automaton Representation

Each $v \in c_i(r_j)$ in cluster c_i and attribute r_j is considered as a string. It should be noted that many regular expressions (RE) can generate v.

Example 6.4.1. *All the following regular expressions parse the string 'aab'.* $A_1 : a^* - b$, $A_2 : a - a^* - b$, $A_3 : a^* - a^* - b$, $A_4 : a - a - b$, $A_5 : a^* - a - b$. *Also, we can replace* b *with* b^*, $b - b^*$ *and* $b^* - b$. *In this model, we only considered two alphabets, and "+" operation was not used.*

Therefore, learning formats from strings is not a well-defined task. To solve this problem, we need an unambiguous grammar which for every valid string has a unique leftmost derivation or parse tree. We use a similar approach presented in [72], where we combine the elements that cannot be parsed, known as the *Glushkov automaton* [23]. For a language L over alphabet Λ, we define: (1) $first(L) = \{b \in \Lambda | \exists w \in \Lambda^* : bw \in L\}$(where * determines kleene star). (2) $last(L) = \{b \in \Lambda | \exists w \in \Lambda^* : wb \in L\}$. (3) $follow(L, a) = \{b \in \Lambda | \exists v, w \in \Lambda^* : vabw \in L\}$. Let Q_E be a finite set of states, Λ be a finite set of alphabet symbols, δ_E be a transition function, $q_l \in Q_E$ determines the start state, and $F_E \subseteq Q_E$ be a set of accept states. The Glushkov automaton $G_E = (Q_E, \Lambda, \delta_E, q_l, F_E)$ has the following properties: (I) $Q_E = \mu(\Lambda) \cup \{q_l\}$, where $q_l \notin \mu(\Lambda)$ is the initial state. $\mu(.)$ puts marking index on element of a set e.g. $\mu((a|b)^*.a.(b^+.a)^*) = (a_1|b_1)^*.a_2.(b_2^+.a_3)^*$. (II) For $a \in \Lambda$, $\delta_E(q_l, a) = \{\chi \in first(\mu(E)) | \sigma(\chi) = a\}$. $\sigma(.)$ is the dropping of subscripts reverse of $\mu(.)$. (III) For $a \in \Lambda$, $\chi \in \mu(\chi)$, $\delta_E(\chi, a) = \{y \in follow(\mu(E), \chi) | \sigma(y) = a\}$. (IV) $F_E = last(\mu(E))$ is the set of final states; add q_l to F_E iff $\lambda \in L(E)$. We also know that the regular expression L is unambiguous iff G_E is deterministic. For all the types of finite automatons considered in this work, we use notions like initial or final state as introduced for the Glushkov automatons.

Learning Automaton

Since we have a unique automaton for each string under the assumption of unambiguous regular expressions, we can learn formats from strings. The learning goal is to obtain a regular expression that represents string's format. Let $\alpha = \{a, A, \ldots, z, Z\}$, $\beta = \{0, 1, \ldots, 8, 9\}$, and $C_s = \{?, !, \#, \$, \%, ;, :, \textit{space}, \ldots\}$ be the set of special characters. We define the following sets $\mathcal{A} = \{w \in \alpha^+ | |w| = 1\}$, $\mathcal{B} = \{w \in \alpha^+ | |w| > 1\}$, $\mathcal{N} = \{w \in \beta^+ | |w| = 1\}$, and $\mathcal{M} = \{w \in \beta^+ | |w| > 1\}$. We assume $\Lambda = \{\mathcal{A}, \mathcal{B}, \mathcal{N}, \mathcal{M}, \} \cup C_s$ and $Q_E = \mu(\Lambda) \cup \{q_l\}$. We assume all examples generated from the Glushkov automaton with Λ and Q_E.

Prepossessing of training data The example of the introduced model should conform with alphabet Λ, so when we get value v in a cluster, we consider it as *string* and we apply the process

$\tau : \left[[a-z]^*.[A-Z]^*.[0-9]^*.C_s\right]^+ \rightarrow \Lambda^+$ to translate the v to a $\hat{v} = \tau(v) \in \Lambda^*$. This process is outlined in Algorithm 27. We keep the previous character l_s and a Kleene star flag f_{Kleene}, and based on seeing more characters, numbers, or special characters, we determine the states. This process is called *blocking training examples*. For example string $s = "New\ York - \#401H3"$ blocks to $\tau(s) = \left[[New][\ ][York][-][\#][401][H][3]\right] = \mathcal{B}space\mathcal{B} - \#\mathcal{MAN}$.

Algorithm 27: String To States (τ)

Input: String $v = s_0 s_1 s_2 \dots s_n$, Automaton Alphabet Λ, The Set of alphabet α, The set of numbers β

Output: String $\hat{v} \in \Lambda^*$,Function $\tau_v : String \rightarrow \Lambda^*$

$l_s \leftarrow s_0$, $f_{Kleene} \leftarrow False$, and $\hat{v} \leftarrow \emptyset$;
for $s \in [s_1, s_2, \dots, s_n, \emptyset]$ **do**
  if $l_s \neq s$ **then**
    if $l_s \in \alpha \wedge f_{Kleene}$ **then**
      $\hat{v} \leftarrow \hat{v}.\mathcal{B}$
    else if $l_s \in \alpha \wedge \neg f_{Kleene}$ **then**
      $\hat{v} \leftarrow \hat{v}.\mathcal{A}$
    else if $l_s \in \beta \wedge f_{Kleene}$ **then**
      $\hat{v} \leftarrow \hat{v}.\mathcal{M}$
    else if $l_s \in \beta \wedge \neg f_{Kleene}$ **then**
      $\hat{v} \leftarrow \hat{v}.\mathcal{N}$
    else
      $\hat{v} \leftarrow \hat{v}.C_s$
    end
    $f_{Kleene} \leftarrow False$;
  else
    $f_{Kleene} \leftarrow True$;
  end
  $l_s \leftarrow s$;
end
return $\hat{v}, \tau_v$

String matching After translating the input string to state strings, we have only one halt state, so the generated regular expressions are linear. Using the linear property, we can consider the representation of the states of the automaton as strings. We generate a *state string* for each value. Informally, the similarity between state strings indicates that they have similar formats. Algorithm 28 performs a transition between the two state strings. From Algorithm 28, we have a

translation from automaton A to the second automaton A'. We use this translation as a protocol to transform the corresponding strings that are generated from these automatons to each other.

Transformation Policy

Applying translations between machines might lose some string information because some parts of the original automaton string are not compatible with the target automaton format. On the other hand, we do not inject any information to the original automaton string, because we do not want to alter the original value distribution (adding symbols make bias in the target distribution especially in the language models). Therefore, we normalize the transformations, by ignoring the transformations that inject information. This is equivalent to $r \longmapsto r'$ where $r \neq \emptyset$ at the index pos, so the set of these transformations make a surjective function.

After we apply a transformation to automaton A, we apply the same transformation to the corresponding original string. The new string is a translation of the original string parsed with the target automaton (new format). If the string new format is the same as before we reject that and repeat the process with another value. Note that the source of the new formats can be given as input by domain expert, in that case Π_{c_t}, Π_a and Π_v are useless, and we select new formats from external information provided.

Formally, let string s be the original string with corresponding learned automaton (RE) A_s. Applying Algorithm 28 gives us $\Phi_{(A_s,A_t)}$ a set of transformations from A_s to a target automaton A_t. We normalize the transformations and obtain $\Phi^{norm}_{(A_s,A_t)}$. If $A'_s = \Phi^{norm}_{(A_s,A_t)}(A_s)$ be the automaton after translation, since we already have τ_s, the function that translates string to state, so we can calculate $s' = \tau_s^{-1}(A'_s)$, which is a new format of the string.

Example 6.4.2. *We have the string* $s =$ "$2016 - 04 - 12$"*, so* $A_s = \tau_s(2016 - 04 - 12) = \mathcal{M} - \mathcal{M} - \mathcal{M}$ *and our target automaton is* $A_t : \mathcal{M}/\mathcal{M}/\mathcal{M}$ *then we obtain two transformations* $\Phi_{A_s \to A_t} = \{[- \to /, [5, 6)], [- \to /, [8, 9)]\}$*. The normal version of* $\Phi_{(A_s,A_t)}$ *is the same, so* $s' = \tau_s^{-1}(\Phi^{norm}_{(A_s,A_t)}(A_s)) =$ "$2016/04/12$"*. As another example, Let* $s =$ "$2016\ May\ 12$" *and the same* A_t*, so we have,* $A_s = \tau_s($*2016 May 12*$) = \mathcal{M}_{\text{space}}\mathcal{B}_{\text{space}}\mathcal{M}$ *and* $\Phi_{A_s \to A_t} = \{[\text{space} \to /, [5, 6)], [\text{space} \to /, [9, 10)]\}$*, so* $s' = \tau_s^{-1}(\Phi^{norm}_{(A_s,A_t)}(A_s)) =$
"$2016\ 04\ 12$".

In summary, using transformations creates the surjective function $\Phi^{norm}_{(A_s,A_t)}(.)$.

Algorithm 28: Transformation Learning (TL)

Input: Automaton pair $e = (A, A')$, Alphabet Λ, The set of special characters C_s
Output: A list of valid transformations Φ_e

if $A = \varnothing \wedge A' = \varnothing$ **return** $\emptyset$;
$l \leftarrow$ Longest Common Substring(A, A');
if $l = \emptyset$ **then**
  if $(Length(A) \geq 2 \wedge \{\exists a \in A \wedge a \notin C_s\}) \vee (Length(A') \geq 2 \wedge \{\exists a \in A' \wedge a \notin C_s\})$ **then**
    $A_c \leftarrow replace_A(a, "C") : \forall a \in A \wedge a \notin C_s$;
    $A'_c \leftarrow replace_{A'}(a, "C") : \forall a \in A \wedge a \notin C_s$;
    $norm(A_c) \leftarrow replace_{A_c}(C^+, "C")$ and $norm(A'_c) \leftarrow replace_{A'_c}(C^+, "C")$;
    TL$(norm(A_c), norm(A'_c))$];
  else
    Add $[(A \longmapsto A', pos)]$ in Φ_e /*pos shows the position of applying the transformation */;
  end
else
  $l_A, r_A \leftarrow A \setminus l$ /* Generate left and right substrings */;
  $l_{A'}, r_{A'} \leftarrow A' \setminus l$;
  if *similarity*$(l_A, l_{A'})$ + *similarity*$(r_A, r_{A'})$ > *similarity*$(l_A, r_{A'})$ + *similarity*$(l_{A'}, r_A)$ **then**
    TL$(l_A, l_{A'})$ and TL$(r_A, r_{A'})$;
  else
    TL$(l_A, r_{A'})$ and TL$(r_A, l_{A'})$;
  end
end
Remove all identity transformations from Φ_e;
return Φ_e

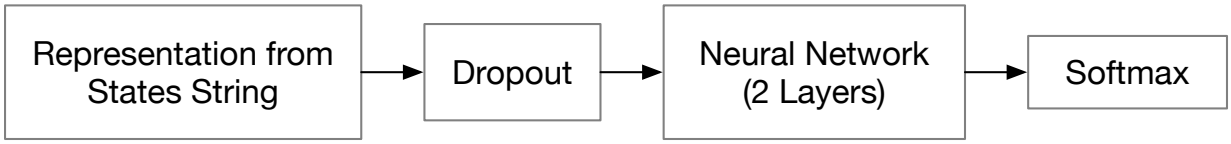


Figure 6.4: The architecture of source classifier

6.4.3 Cluster Augmentation Algorithm

Let $c_i[r_j]$ denote all records in attribute r_j in cluster c_i. For each attribute $r_j \in R$, we compute the number of occurrences of each $v \in c_i(r_j)$ in $c_i[r_j]$, as with $n_{(c_i,r_j)}(v)$. The automaton translation applies to all values $v \in c_i(r_j)$ except the correct value of $c_i(r_j)$. Afterwards, we reconstruct the statistical representation of each augmented value $v' = \tau_v^{-1}(\Phi^{norm}_{(A_v,A_{\Pi(G_T\setminus\{c_i\})})}(A_v))$ with information of $n_{(c_i,r_j)}(v), v \in c_i(r_j)$ which $\sum_{v\in c_i(r_j)} n_{(c_i,r_j)}(v) = |c_i[r_j]| = |c_i|$. This process is outlined in Algorithm 29.

In Algorithm 29, $gt(c, r)$ provides the correct value of Attribute r in Cluster c. The statistical property of the augmented cluster is the same with the original cluster which gives us robustness for the calculation of the statistical signals. If we have some information $\mathcal{I}$, we can apply a conditional distribution $\Pi \sim \mathcal{D}(G_T|\mathcal{I})$ for our selection policies.

6.4.4 Source Information Augmentation

The augmentation algorithm generates artificial clusters. As you can imagine, the generative process cannot be used for the source information attribute. Therefore, After the augmentation algorithm generates all attributes but the source information, we can use language models to extract the format of generated rows and then use the sources' signature to find the most probable source for them. We use *format* and *Character and token models* over state strings of the original clusters, and use these representations to train a 2-layers neural network with a *Softmax* that generates a probability distribution over all sources of the dataset (see Figure 6.4). For model optimization, we use logistic loss.

In the prediction, we generate the representation model for the augmented cluster's row using the representation models, and by passing them through the learned model, we obtain a distribution over all sources. We use MAP to assign a source to the input row. We do this prediction for every row in the new cluster to fill its source attribute. The augmented data with source information attribute could also be used for previous approaches. Specifically, approaches that use statistical property of input dataset for data fusion.

Algorithm 29: Cluster Augmentation (CA)

Input: Dataset D, Training Dataset G_T, Automaton Transformation Φ, String to State Function τ, Cluster Selection Policy Π

Output: Augmented cluster c'

Empty cluster c' with Schema S;
$c_i, c_t \sim \Pi(G_T | c_i \neq c_t)$;
for $r_j \in R$ **do**
    Dictionary $Cor \leftarrow \emptyset$;
    $tr_{c_i} \leftarrow gt(c_i, r_j), tr_{c_t} \leftarrow gt(c_t, r_j)$;
    for $v \in c_i(r_j)$ **do**
        if $v \neq tr_{c_i}$ **then**
            Select $r_k \in R$, then $v_t \in c_t(r_k) \backslash tr_{c_t}$ with policy Π;
            $A_v \leftarrow \tau(v), A_t \leftarrow \tau(v_t)$;
            $old \leftarrow True$;
            while *old* **do**
                $v' \leftarrow \tau_v^{-1}(\Phi^{norm}_{(A_v, A_t)}(A_v))$;
                if $v' \notin D$ **then**
                    $c'(r_j) \leftarrow c'(r_j) \cup v'$;
                    Append $Cor[v] \leftarrow v'$;
                    $old \leftarrow False$;
                end
            end
        else
            $c'(r_j) \leftarrow c'(r_j) \cup v$;
            Append $Cor[v] \leftarrow v$;
        end
    end
    for $v \in c_i[r_j]$ **do**
        Append $Cor[v]$ to $c'[r_j]$;
    end
end
return c'

6.5 Iterative Learning

In our framework, we use an iterative process to improve the learning performance; the key idea here is to use *Hard-Assignment Expectation Maximization* [122]. The framework, at the end of each learning phase, evaluates its output using Hard EM, to maximize the probability of expected values.

6.5.1 Hard Expectation Maximization

In the case of complete data, we have enough *statistical sufficiency* to obtain a conditional probability distribution, but this might not be the case when we have missing values. When some observations are missing, one approach is to randomly choose a value from the random variables domain or use default values. However, such approach introduces a bias, and the estimation is skewed toward the applied policy for filling in missing values. Moreover, in the random assignment, we will not learn any dependencies between hidden variables because they are independently generated.

An alternative algorithm for optimizing a likelihood function is *Expectation Maximization*. This approach is useful when we have a hidden random variable or incomplete data. The *Expectation Maximization* algorithm starts with an arbitrary point, either parameters θ or hidden variables $u_o \in U$, where U is the set of cells in G that are missing observations. The algorithm then repeats two steps: first, it uses the current parameters to complete the data, using a probabilistic completion of the different data instances to estimate the expected value of the sufficient statistics (Expectation step or E-step), then it treats the completed data as if it were observed and learn a new set of parameters for the next iteration (Maximization step or M-step).

In our framework, we assume the true value of a record to be a random variable, and the classes (for the prediction task of model M) to correspond to the different values from the domain of the record (the values $c_i(r_j)$). Each class $c \in c_i(r_j)$ is associated with a probability distribution over the features of the instances in the class, $P(\mathbf{x}|c)$. This approach views the data as the result of a *mixture distribution* of classes and attempts to use the hidden variable to separate the mixture into its components. In this setup, the E-step involves computing the probability of different values of the class variables for each instance, but since we only need one value from the estimation of true record representation for the next iteration, we should avoid a soft classification. Therefore, we consider "hard clustering", where each instance contributes all of its weight to the cluster to which it is most likely belongs to and if we have more than one we select one randomly. Given parameters θ^t, the cluster of instance m is $c_m = random(\arg\max_c P(c|h(m), \theta^t))$, where $h(.)$ is the featurization model described in Section 6.3.1, and m is an attribute of R in a cluster of

$\mathcal{C}$. By filling the unobserved values (which are clusters) with no true record representation ($\mathbf{u_o}$) and add them to the set of observed true record representation, $G_{\mathcal{C}\setminus T}$, to have $(\mathcal{D}^+)^t$, which is the completed set of the data at step t. In M-step, we calculate θ^{t+1} using $(\mathcal{D}^+)^t$ by maximizing likelihood estimation (MLE), $\theta^{t+1} = \arg\max_\theta \mathcal{L}(\theta : (\mathcal{D}^+)^t)$.

6.5.2 Recurrent Process

As we mentioned above, our model is built using a variety of static and dynamic features, which use current cluster values, as they change in each iteration (see Section 6.3.1). In fact, in every iteration, we will get more accurate predictions, so we will be able to recalculate all the dynamic features more accurately and proceed to the inference of classification variables once again. This approach is inspired by Neville et al. [150], which uses upon the framework's prediction.

In EM, one can start by initializing parameters or use an auxiliary method to complete the data. The first method can make trouble if the initial point does not converge into the global optima. Therefore, we use *Majority Vote* to complete the data with the most probable answers. Notice that *Majority Vote* is not a perfect algorithm, but it can deal with the problem of the parameters initialization that are far from feasible space. Given the input source observations G_T, augmented training data G_A and U, the framework firstly using *Majority Vote* (MV) estimates true record representation for unobserved random variables, $G_U = MV(D_U)$, and the completed data initialized as $(\mathcal{D}^+)^0 = G_T \cup G_A \cup MV(U)$. Then, the model M parameters θ can be obtained by MLE, $\mathcal{L}_M(\theta) = \log P(\theta|(\mathcal{D}^+)^0)$. Afterward, the probabilistic inference is used to predict true record representations from the domain of objects $u_o \in U$. We need all true record representations to calculate dynamic representation models, but for some random variables, the true record representation is unknown. Algorithm 30 shows how we calculate the expectation of our model M^t at iteration t to predict its true record representations and how we use this prediction to maximizes its likelihood. In all iterations, we fix the true record representation of the observed random variables.

6.6 Experiments

We evaluate our record fusion framework using real datasets with various rules. We answer the following questions: (1) how well does the record fusion framework work as a data fusion system compared to the state-of-the-art data fusion systems when the source information of the entries is known. (2) how well does it perform when the source of entries are not available compared to [51] and Majority Vote. (3) what is the impact of different representation contexts on data

Algorithm 30: Iterative Learning (IL)

Input: Set of observed random variables G_T, Set of augmented random variables G_A, Set of unknown random variable U, Domain of Random Variables P, Number of Iterations I, Featurizer Function h, Classifier M

Output: Empirical estimation $\hat{Y}$

Initialize $(\mathcal{D}^+)^0 \leftarrow G_T \cup G_A \cup MV(U)$;
for $t = 1$ *to* I **do**
  $X^{(t-1)} \leftarrow h((\mathcal{D}^+)^{(t-1)}, P)$ (calculate the representation vectors);
  Obtaining MLE of $\mathcal{L}_M(\theta) = \log P(\theta : X^{(t-1)})$;
  $(\mathcal{D}^+)^t \leftarrow M(X^{(t-1)}, P; \theta)$ (new predictions using model M);
  $(\mathcal{D}^+)^t \leftarrow (\mathcal{D}^+)^0[G_T \cup G_A]$ (replace original values of observed random variables);
end
$\hat{Y} \leftarrow (\mathcal{D}^+)^I$;
return $\hat{Y}$

Table 6.1: Datasets used in our experiments.

Dataset	Size	Clusters	Attributes	# Sources	has constraints?
Flight	57,222	2,313	6	37	No
Stock 1	113,379	2,066	10	55	No
Stock 2	107,260	1,954	8	55	No
Weather	43,003	13,689	6	11	Yes
Address	3,287	494	6	N/A*	Yes
S_Rel	28,291	4,460	2	523	No
S_Adult	10,537	1,517	2	603	No

* N/A = Address dataset basically has been generated without sources.

fusion. (4) is cluster augmentation a valid approach to solve the lack of training data for record fusion problems. (5) how well our iterative algorithm can solve the problem of learning from noisy and incomplete data.

Table 6.2: Precision's Median, Average, and Standard Error of different methods for different datasets.

Dataset (G_T size)	Prec	RF_S	NB	ACCU	CATD	SSTF	SF	DS [48]	ZC [50]	GLAD [199]	MM [209]	BCC [116]	CBCC [194]	LCF [175]	PM [8]	RF_W^+	RF_W	MV	UM
Flight (5%)	Med	**0.958**	0.885	0.878	0.912	0.739	0.220	0.333	0.492	0.449	0.354	0.424	0.423	0.449	0.394	**0.939**	0.853	0.296	0.305
	Avg	0.949	0.882	0.892	0.909	0.732	0.241	0.376	0.517	0.418	0.333	0.374	0.419	0.449	0.369	0.947	0.859	0.176	0.337
	StE	0.001	0.009	0.005	0.013	0.021	0.003	0.036	0.024	0.028	0.018	0.046	0.003	2×10^{-4}	0.024	0.008	0.041	0.202	0.105
Stock 1 (5%)	Med	**0.985**	0.815	0.906	0.921	0.688	0.323	0.429	0.331	0.394	0.329	0.347	0.424	0.352	0.307	**0.935**	0.921	0.051	0.180
	Avg	0.977	0.862	0.867	0.941	0.632	0.343	0.399	0.370	0.377	0.376	0.353	0.460	0.351	0.260	0.929	0.899	0.035	0.062
	StE	0.002	0.021	0.029	0.023	0.005	0.006	0.025	0.038	0.015	0.042	0.015	0.036	0.021	0.044	0.003	0.017	0.040	0.034
Stock 2 (5%)	Med	**0.938**	0.840	0.853	0.823	0.779	0.767	0.35	0.325	0.317	0.36	0.428	0.397	0.361	0.325	**0.939**	0.889	0.745	0.796
	Avg	0.941	0.825	0.812	0.737	0.652	0.795	0.396	0.356	0.283	0.335	0.438	0.431	0.343	0.304	0.945	0.901	0.782	0.856
	StE	0.009	0.016	0.013	0.009	0.031	0.028	0.036	0.028	0.029	0.025	0.008	0.029	0.015	0.018	0.013	0.029	0.064	0.107
Weather (5%)	Med	**0.797**	0.669	0.602	0.606	0.437	0.417	0.349	0.479	0.334	0.322	0.31	0.455	0.405	0.329	**0.774**	0.794	0.511	0.558
	Avg	0.822	0.719	0.662	0.607	0.513	0.441	0.347	0.430	0.368	0.367	0.346	0.451	0.405	0.353	0.867	0.787	0.641	0.672
	StE	0.004	0.018	0.009	0.017	0.011	0.003	0.002	0.044	0.034	0.035	0.025	0.003	3×10^{-4}	0.023	0.021	0.044	0.091	0.053
Address (10%)	Med	n/a*	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	**0.936**	0.883	0.757	0.772
	Avg	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.919	0.894	0.740	0.780
	StE	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.013	0.037	0.231	0.183
S_Rel (5%)	Med	**0.932**	0.719	0.592	0.693	0.301	0.247	0.615	0.486	0.538	0.581	0.607	0.564	0.657	0.610	**0.843**	0.754	0.542	0.615
	Avg	0.927	0.699	0.562	0.707	0.283	0.211	0.631	0.453	0.515	0.579	0.614	0.526	0.647	0.593	0.847	0.761	0.542	0.652
	StE	0.003	0.018	0.009	0.017	0.011	0.003	0.020	0.017	0.026	0.037	0.013	0.008	0.021	0.024	0.014	0.024	0.000	0.043
S_Adult (5%)	Med	**0.871**	0.479	0.562	0.552	0.337	0.227	0.338	0.351	0.35	0.365	0.347	0.339	0.355	0.361	**0.807**	0.734	0.361	0.452
	Avg	0.892	0.359	0.551	0.561	0.323	0.241	0.324	0.354	0.354	0.363	0.361	0.352	0.366	0.372	0.827	0.727	0.359	0.422
	StE	0.011	0.032	0.027	0.070	0.053	0.025	0.009	0.002	0.003	0.001	0.012	0.013	0.008	0.009	0.016	0.023	0.003	0.013

* n/a = Address dataset basically has source information, so source-needed algorithms cannot be executed.

Table 6.3: Precision's Median, Average, and Standard Error of different methods for different datasets with source information augmentation.(+ shows the method process perform over augmented data)

Dataset (G_T size)	Prec	RF_S^+	NB^+	$ACCU^+$	$CATD^+$	$SSTF^+$	SF^+	DS^+ [48]	ZC^+ [50]	$GLAD^+$ [199]	MM^+ [209]	BCC^+ [116]	$CBCC^+$ [194]	LCF^+ [175]	PM^+ [8]
Flight (5%)	Med	**0.981**	0.907	0.883	0.951	0.875	0.417	0.426	0.527	0.513	0.368	0.441	0.477	0.508	0.416
	Avg	0.979	0.911	0.931	0.949	0.885	0.429	0.465	0.553	0.470	0.337	0.382	0.441	0.537	0.415
	StE	7×10^{-4}	0.002	0.004	0.01	0.018	0.001	0.031	0.019	0.026	0.015	0.041	0.002	4×10^{-5}	0.019
Stock 1 (5%)	Med	**0.988**	0.855	0.952	0.942	0.823	0.519	0.523	0.416	0.437	0.405	0.362	0.451	0.398	0.363
	Avg	0.989	0.872	0.903	0.962	0.803	0.530	0.484	0.426	0.422	0.359	0.358	0.494	0.453	0.308
	StE	8×10^{-4}	0.017	0.024	0.023	0.005	0.004	0.021	0.030	0.013	0.035	0.011	0.032	0.017	0.033
Stock 2 (5%)	Med	**0.984**	0.859	0.905	0.852	0.906	0.899	0.556	0.406	0.350	0.368	0.447	0.467	0.440	0.350
	Avg	0.980	0.862	0.855	0.779	0.652	0.942	0.574	0.391	0.352	0.339	0.443	0.467	0.424	0.360
	StE	0.006	0.013	0.008	0.005	0.027	0.022	0.029	0.025	0.021	0.020	0.002	0.023	0.014	0.016
Weather (5%)	Med	**0.886**	0.780	0.753	0.636	0.591	0.512	0.424	0.497	0.393	0.338	0.342	0.498	0.462	0.351
	Avg	0.941	0.869	0.799	0.732	0.784	0.745	0.417	0.463	0.435	0.360	0.360	0.486	0.498	0.419
	StE	0.003	0.015	0.007	0.01	0.007	0.002	0.002	0.039	0.028	0.032	0.024	0.002	7×10^{-5}	0.023
S_Rel (5%)	Med	**0.959**	0.797	0.639	0.723	0.454	0.417	0.696	0.526	0.608	0.518	0.624	0.601	0.723	0.648
	Avg	0.961	0.773	0.606	0.741	0.459	0.396	0.725	0.490	0.564	0.563	0.625	0.559	0.733	0.647
	StE	0.003	0.016	0.005	0.014	0.010	0.002	0.017	0.013	0.022	0.034	0.012	0.006	0.018	0.022
S_Adult (5%)	Med	**0.892**	0.531	0.621	0.578	0.480	0.441	0.429	0.385	0.436	0.360	0.361	0.379	0.417	0.389
	Avg	0.902	0.440	0.596	0.594	0.507	0.432	0.409	0.395	0.416	0.367	0.373	0.384	0.564	0.432
	StE	0.009	0.029	0.021	0.063	0.044	0.019	0.006	0.002	0.002	9×10^{-4}	0.009	0.012	0.005	0.008

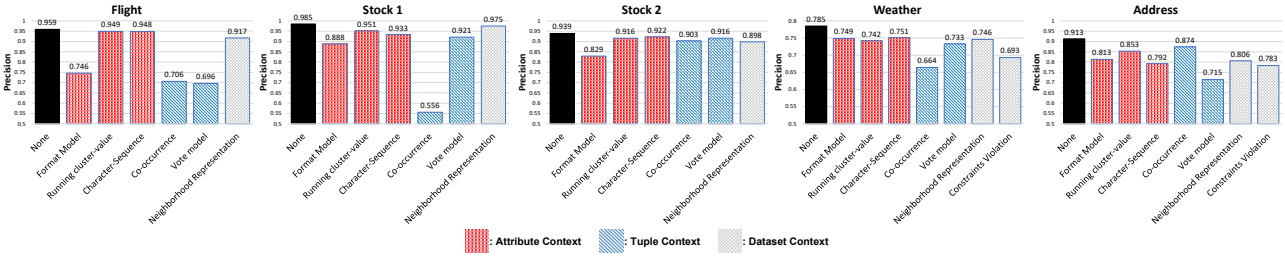


Figure 6.5: Ablation studies to evaluate the effect of different representation models.

6.6.1 Experimental Setup

We describe the datasets, metrics, and settings that we use in our experiments.[1] We use seven datasets with different domain properties and usage described in Table 6.1.

Flight [134] is a benchmark dataset that contains flights information from the flight domain. The dataset focused on $2,313$ flights departing from or arriving at the hub airports of the three airlines. The ground truth is created by domain experts. *Stock 1 and 2* [134] contain data from popular financial aggregators such as *Yahoo! Finance*, *Google Finance*, and *MSN Money*, official stock-market websites such as *NASDAQ*, and financial-news websites such as *Bloomberg* and *MarketWatch*. the ground truth of *Stock 1* is created by assuming that *NASDAQ* always provides the correct value. the ground truth of *Stock 2* is created by taking the majority value provided by five stock data providers. *Weather* is collected for 30 major USA cities from 11 websites about every 45 minutes. We consider *(city, time)* as the (cluster) key. There are in total 32 collections in a day. The attributes are manually mapped, and there are 6 distinct attributes. The ground truth is provided by domain experts. *Address* reflects applications for discretionary funding to be allocated by the New York City Council. For each record, the attributes that represent legal information, address, and geographical properties of location are selected. The minimum size of each cluster is two and the ground truth has been extracted from the *ISBNsearch* organization website. An interesting feature of the *Address* dataset is that it does not contain any source information (*S_Rel*) [24]. Each cluster contains a topic and a document, and sources are asked to choose the relevance of the topic w.r.t. the document by selecting one out of four choices: 'highly relevant', 'relevant', 'non-relevant', and 'broken link'. *S_Adult* [102]. Each task contains a website, and workers are asked to identify the adult level of the website by selecting one out of four choices: 'G' (General Audience), 'PG' (Parental Guidance), 'R' (Restricted), and 'X' (Porn). For the last two datasets, we only select clusters that we have their ground truth.

[1]The implementation and datasets are publicly available on Github. It is removed for double-blind policy and will appear in the final version.

Methods We compare our approach, referred to as RF_S when we have source information, RF_W when source information is not available and there is no data augmentation, and RF_W^+ when source information is not available and we use data augmentation, against several data fusion methods. First, we consider 13 baseline data fusion and crowdsourcing models that need source information (i.e. which source generated each row in the cluster). *NB*: This corresponds to Naive Bayes. Source accuracies are estimated as the fraction of times a source provides the correct value for an object in the ground truth. *ACCU* [59] is the Bayesian data fusion method without source information copying. *CATD* [132] extends source reliability scores with confidence intervals to account for sparsity in source observations. *SSTF* [205] leverages semi-supervised graph learning to exploit the presence of ground truth data. *SlimFast[SF]* [180] is a data fusion based on statistical learning over discriminative probabilistic models. *DS* [48] maximizes the likelihood of the observed labels given sources using the EM algorithm. *ZC* [50] uses PGMs without considering priors and estimates the correctness of sources. *GLAD* [199] extends the *ZC* idea with a model that gives difficulty coefficient to each cluster. *MM* [209] gives a different score to each source and uses *Minimax* algorithm to learn a probability for each source. *BCC* [116] maximizes the posterior joint probability distribution to find the correct labels. *CBCC* [194] extends *BCC* by considering community of sources and calculating a membership score for each cluster into these communities. *LCF* [175] extends *DS* to incorporate prior distribution on source score modeling. *PM* [8] gives a score $[0, +\infty)$ to each source and iteratively determines the score to maximize the likelihood of the observation.

We also compared our method with two approaches that require no source information. *Majority Vote (MV)* considers the maximum frequency value as the true record representation in each cluster-attribute. *UM* [51] is an entity consolidation method which uses human-in-the-loop to request the user to verify the equivalence of records, *Majority Vote* can also be used to obtain correct records. To see the rest of the methods refer to [208].

Evaluation Setup: To measure precision, we use Precision (P) defined as the fraction of true record representation predictions that are correct. For training, we split the available ground truth into three disjoint sets: (1) a training set T, used to find model parameters; (2) a validation set, which is used for hyperparameter tuning; and (3) a test set, which is used for evaluation. To evaluate different dataset splits, *we perform* 50 *runs with different random seeds for each experiment* to ensure that we gain robust results for Precision, we report the median performance. The mean performance along with standard error measurements is also reported. *Seeds are sampled at the beginning of each experiment, and hence, a different set of random seeds can be used for different experiments.* We use *ADAM* [119] as the optimization algorithm for all learning-based models and train all models for 500 epochs with a batch size of ten examples. We run Platt Scaling for 50 epochs. All experiments were executed on a 12-core Intel(R) Xeon(R) CPU E5-2603 v3 @ 1.60GHz with 64GB of RAM running Ubuntu 14.04.3 LTS.

6.6.2 End-to-End Performance

We evaluate the performance of our approach and competing approaches on data fusion in all five datasets. Table 6.3 summarizes the precision's Median, Average, and Variance of each method. For *Flight*,*Stock 1*, *Stock 2*, and *Weather*, we set the amount of training data to be 5% of the total dataset. For Address, we set the percentage of training data to be 10% (corresponding to 40 clusters) since it is fairly small. The number of iterations is 15, and in the case of using the data augmentation, the amount is 5% of clusters.

As Table 6.3 shows, our method consistently outperforms all other methods. In the no sources information case, we see improvements of $\sim 65/55$ points for *Flight* with/without data augmentation. More importantly, we find that our method is able to achieve low standard error in all datasets despite the different clusters and true record representation distributions in each dataset. This is something that seems challenging for prior data fusion methods and reduce the consistency and reliability of their results. Despite the fact that source information is an important factor for other algorithms, *RF* can obtain high precision. This is because *RF* model estimates the actual data distribution by extracting source signatures using attribute correlation from datasets. For instance, for *Address*, we see that MV can find many true record representations—it has high precision—indicating that most true record representations correspond to the statistical frequency. Overall, our method achieves an average precision of $93/87\%$ without sources information with/without data augmentation, and an average precision of 99% when sources information is available across these diverse datasets, while the performance of competing methods varies significantly.

Table 6.3 shows the result of the data augmentation with source information, as you can see the data augmentation on average improve precision of methods by 6 percents. The method that more rely on the statistical sufficiency of the data or their model are more complicated get more increase.

For *CATD*, we see that it achieves relatively high-precision results in *Flight* and *Stock 1*, but it is not consistent on all datasets. Similar performance is exhibited by *Count*. We see that *Count* achieves high precision when the correct answers have significant Bayesian support (i.e., occur relatively often), due to the fact that this dataset has a strong Co-occurrence signal (see Figure 6.5). Finally, the results of *SlimFast* and *SSTF* varies drastically from 0.220 for *Flight* to 0.779 for *Stock 2*, and under no settings they give the best performance.

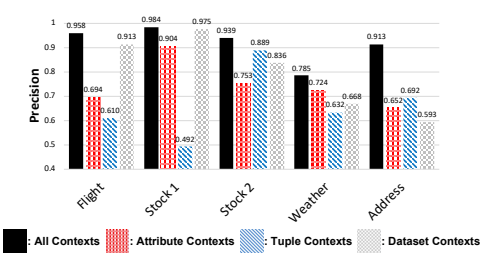


Figure 6.6: Ablation studies to evaluate the effect of different representation model groups with data augmentation.

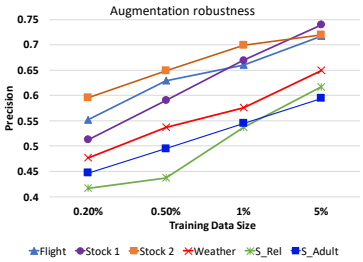


Figure 6.7: The effect of increasing the number of clusters via data augmentation without iteration.

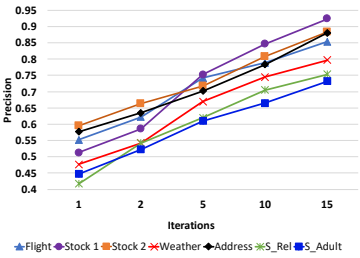


Figure 6.8: The effect of increasing the number of iterations without data augmentation.

6.6.3 Representation Ablation Study

We perform an ablation study to evaluate the effect of different representation models on the quality of our system. Specifically, we compare the performance of *RF* when all representation models are used versus variants of *RF*, where one or a set of representation models is removed at a time.

Single Representation Effect

In Figure 6.5, we report the precision of the different variants as well as the original *RF*. It is shown that removing any feature has an impact on the quality of the predictions of our model. More importantly, we find that different representation models have a different impact on various datasets. For instance, the most significant drop for *Stock1* and *Weather* is achieved when the co-occurrence model is removed, while for *Flight* and *Address*, the highest drop is achieved when the voting model is removed. Therefore, the representation models that we considered have a positive impact on the performance of our system. As it can be observed in Figure 6.5, for example, in the dataset *Fight*, removing *Running-cluster value* representation from the model has a minimum impact on its performance, and the parameters of this representation after the model is trained, have values that have a minimum impact.

Group Contexts Effect

Figure 6.6 shows the effect of a group of representation models corresponding to different contexts. It can be observed that removing any contexts group has an impact on the quality of

predictions of our model. Furthermore, for datasets that have various properties, different context groups have the most prominent effect on the performance of *RF*. For *Flight*, *Stock 1*, and *Weather*, the largest drop is achieved when the tuple contexts group models are removed, while for *Address*, the highest drop is achieved when the dataset contexts group models are removed. This validates our design of considering representation models from different contexts.

Takeaway: It is necessary to leverage cluster representations that are informed by different contexts to provide robust and high-quality data fusion solutions.

6.6.4 Effects of Augmentation on Performance

We evaluate the effectiveness of data augmentation to counteract the lack of training data. Figure 6.7 shows that using data augmentation yields high-quality record fusion models for datasets with varying sizes and properties (as they were described in section 6.6.1). Hence, data augmentation is robust to different domains of properties. We ignore Address dataset because it is small to make an adequate augmentation model. We also evaluate the effect of excessive data augmentation: We manually set the ratio between the initial clusters and the lately generated cluster in the final training examples and use augmentation to materialize this ratio. Our results are reported in Figure 6.7. We see that peak performance is achieved when the ratio between the two types of clusters is about 10% to 30% for all datasets.

Takeaway: Data augmentation is an effective and robust way to counteract the lack of enough training data.

6.6.5 Effects of Iterations on Performance

In this experiment, we validate the importance of the iterative process to improve learning performance. Figure 6.8 shows the results of *RF* for a various number of iterations. The results validate that as the number of iterations increases, we were able to get more accurate predictions. This observation has significant meaning for the performance of *RF* as getting more accurate predictions in each iteration results in recalculating the dynamic features more accurately in each round. For instance, in Weather, *RF* was able to achieve precision less than 0.7 with only one iteration; however, after 15 iterations, the precision was improved over 10 points.

Takeaway: The recurrent process is an effective approach for calculating accurately dynamic features.

6.7 Related Work

Several pieces of research have been done on combining data from multiple sources. Bleiholder et al. [16] surveyed existing strategies for resolving inconsistencies in structured data. Specifically, the data fusion methods can be categorized into four main regimes:

I **Naïve method**: In this method, all sources have a vote, and the correct value for each object is decided by choosing the value that has maximum votes among all the conflicting values. In these models, there is no worker(source) modeling, so there is no need of source information to discriminate records.

II **Source-based** main goal is to calculate how accurate is each source(worker). More specifically, in this framework, the votes of the sources(workers) do not have the same "weight" or "quality", so it assigns a weight q^w to worker w and estimates the quality of the worker based on D. The importance of each vote depends on the quality of the source. In the crowdsourcing community, this is called worker modeling [110,208]. The necessity of using these methods is to have a minimal source information that the model can discriminate records.

III **Relation-based** methods use the main idea of Source-based methods. They also consider the correlation between the sources(worker) (e.g., if a pair of sources copy from each other).

IV **Transformation-based** methods reduce cluster size by transforming values to each other and use human-in-the-loop to fuse the remained set.

V **Signature-based** methods use representation models to find the latent signature of the truth pattern. Some of these methods (like our model) can learn from data without any source information and extract the truth pattern from a proportion of data annotated by humans and given as training data.

In the field of discovering dependencies between data sources, a lot of work has been done as well. In [48,59,116,194], an Bayesian analysis is applied to decide on the dependencies between sources. In some researches [50,58,199], they consider various types of copying on different data records. Moreover, [60] explores the idea of integrating data and determining the way the sources are interacting with each other by examining the updated history of the sources. There has been a lot of research in the field of evaluating the trustworthiness of the sources resulting in algorithms such as *PageRank* which assigns trust based on link analysis [209] and TrustFinder [204] which decides about the importance of a source based on its behavior in a P2P network. Moreover, in [62], Dong et al. examine the problem of selecting a subset of sources before integration. The

authors claim that by choosing only the sources that can be beneficial for their algorithm, they can achieve higher performance than by using all the available sources and data.

Furthermore, in [159], Pasternack and Roth solve the data fusion problem by creating an iterative model. The main idea of their fact-finding algorithm was to incorporate prior knowledge from the users into their process, to integrate data from conflicting claims. The SLiMFast [180] proposes a framework to solve the data fusion problem as a learning and inference problem over discriminative probabilistic graphical models. SliMFast was also the first that came with guarantees on its error rate for the estimation of the source accuracy.

In [51, 124] human-in-the-loop is used to solve entity consolidation; instead of using sources, the system simulates source information for entity resolution by asking information from an oracle.

Chapter 7

Conclusion and Future Work

In this chapter, we conclude the dissertation and discuss some promising future works.

7.1 Conclusion

There are two aspects for considering the structure of the data. First is the case that we have structure information that can help to retrieve the clean version of data. The second type is when we have a structure on the noises, and then our data cleaning model should consider the effect of this structured noise on the cleaning process.

We made five contributions in structured prediction in this dissertation following these two aspects. First, we proposed an error detection framework based on ML modeling, which used structured features like data rules in the format of denial constraints. This modeling allowed us to improve the performance of detecting erroneous cells and enabled us to augment training data using few erroneous examples. Second, we explored the correctness of inference on the graphical models, and we obtained promising theoretical results for using graphical models on categorical data. Third, we studied the problem of sampling from structured information. Specifically, when we have data rules, they define a graph over data points. If we want to detect approximately correct data rules by sampling from these graphs, we need to have analytical guarantees for rules that we detected by sampling from these structures. Next, we presented a method for sampling from data with duplicated records. Data duplications imply a structure on the data where each pair that refer to the same real-world entity are connected. Sampling from this structure assigns a higher probability to larger connected components. We have provided three methods that, under some assumptions, provide uniform samples over entities. Subsequently, when we merge

multiple datasets, we might face conflicting information that requires a process to solve these conflicts. We presented an ML framework that uses arithmetic- and structure-based features for merging conflicting data. We also provided a process to augment data for learning purposes. For all these contributions, we provided experimental studies to show how they work in real-world scenarios.

7.2 Future work

Many aspects of structured prediction require profound studies and more researches. In this dissertation, we considered only some aspects of structured prediction. We plan to investigate some new ideas and improve upon some of our existing contributions in the future. In particular, we consider the following tasks to be promising directions for future work.

First, building a data cleaning framework based on structured prediction on graphs or data representations other than relational data. Graph data have some properties that can help the structured prediction be applied more conveniently.

Second, In this dissertation hypothesis (Section 1.5), we assumed that the input data does not have any noises, which means the domain experts have perfect knowledge. This assumption can be relaxed, and These contributions can employ machine learning (ML) under agnostic learning properties.

Third, current data cleaning frameworks do not take into account the effect of data evaluation. Many datasets can change over time, and the distribution of the data is not the same. Improving the data cleaning systems so they can recognize the data is unclean, or the model is expired.

References

[1] Ziawasch Abedjan, Cuneyt Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. Temporal rules discovery for web data cleaning. *PVLDB*, 9(4):336 –347, 2015.

[2] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12):993–1004, 2016.

[3] Ziawasch Abedjan, John Morcos, Ihab F. Ilyas, Paolo Papotti, Mourad Ouzzani, and Michael Stonebraker. DataXFormer: A robust transformation discovery system. In *ICDE*, 2016.

[4] Rui Abreu and Arjan J. C. van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *SARA*, 2009.

[5] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 586–597. Elsevier, 2002.

[6] Phipps Arabie, Scott A Boorman, and Paul R Levitt. Constructing blockmodels: How and why. *Journal of mathematical psychology*, 17(1):21–63, 1978.

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

[8] Bahadir Ismail Aydin, Yavuz Selim Yilmaz, Yaliang Li, Qi Li, Jing Gao, and Murat Demirbas. Crowdsourcing for multiple-choice question answering. In *AAAI*, pages 2946–2953. Citeseer, 2014.

[9] Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2(2):198–203, 1981.

[10] James Vere Beck and Kenneth J Arnold. *Parameter estimation in engineering and science*. James Beck, 1977.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, August 2013.

[12] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[13] James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[14] Mikhail Bilenko, Beena Kamath, and Raymond J Mooney. Adaptive blocking: Learning to scale up record linkage. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 87–96. IEEE, 2006.

[15] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. Efficient denial constraint discovery with hydra. *PVLDB*, 11(3):311–323, 2017.

[16] Jens Bleiholder and Felix Naumann. *Conflict handling strategies in an integrated information system*. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät ..., 2006.

[17] Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In Timo Lepistö and Arto Salomaa, editors, *Automata, Languages and Programming*, pages 105–118. Springer Berlin Heidelberg, 1988.

[18] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[19] Stéphane Boucheron, Gábor Lugosi, Pascal Massart, et al. Concentration inequalities using the entropy method. *The Annals of Probability*, 31(3):1583–1614, 2003.

[20] Marc Boullé. Universal approximation of edge density in large graphs. *arXiv preprint arXiv:1508.01340*, 2015.

[21] R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941.

[22] Daniel G Brown. How i wasted too long finding a concentration inequality for sums of geometric variables. *Found at https://cs. uwaterloo. ca/˜ browndg/negbin. pdf*, 6, 2011.

[23] Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Information and computation*, 142(2):182–206, 1998.

[24] Chris Buckley, Matthew Lease, Mark D Smucker, Hyun Joon Jung, Catherine Grady, Chris Buckley, Matthew Lease, Mark D Smucker, Catherine Grady, Matthew Lease, et al. Overview of the trec 2010 relevance feedback track (notebook). In *The nineteenth text retrieval conference (TREC) notebook*, 2010.

[25] Norbert Bus, Nabil H. Mustafa, and Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018.

[26] Nuno Cardoso and Rui Abreu. MHS2: A map-reduce heuristic-driven minimal hitting set search algorithm. In *MUSEPAT*, pages 25–36, 2013.

[27] Chengliang Chai, Ju Fan, Guoliang Li, Jiannan Wang, and Yudian Zheng. Crowdsourcing database systems: Overview and challenges. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2052–2055. IEEE, 2019.

[28] Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno-Centeno, and Santosh Vempala. Algorithms for implicit hitting set problems. In *SODA*, pages 614–629, 2011.

[29] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[30] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter*, 6(1):1–6, 2004.

[31] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[32] Yuxin Chen and Andrea J Goldsmith. Information recovery from pairwise measurements. In *2014 IEEE International Symposium on Information Theory*, pages 2012–2016. IEEE, 2014.

[33] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, 2016.

[34] Fei Chiang and Renée J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

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

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

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

[38] Xu Chu, Ihab F Ilyas, and Paraschos Koutris. Distributed data deduplication. *Proceedings of the VLDB Endowment*, 9(11):864–875, 2016.

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

[40] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[41] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1247–1261. ACM, 2015.

[42] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1247–1261, 2015.

[43] Samuel Clemens. 7 facts about data quality. 2018.

[44] Carlo Combi, Matteo Mantovani, Alberto Sabaini, Pietro Sala, Francesco Amaddeo, Ugo Moretti, and Giuseppe Pozzi. Mining approximate temporal functional dependencies with pure temporal grouping in clinical databases. *Comp. in Bio. and Med.*, 62:306–324, 2015.

[45] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[46] 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.

[47] Tamraparni Dasu and Ji Meng Loh. Statistical distortion: Consequences of data cleaning. *PVLDB*, 5(11):1674–1683, 2012.

[48] Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(1):20–28, 1979.

[49] Simon de Givry, Thomas Schiex, and Gerard Verfaillie. Exploiting tree decomposition and soft local consistency in weighted csp. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 22–27. AAAI Press, 2006.

[50] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st international conference on World Wide Web*, pages 469–478, 2012.

[51] Dong Deng, Wenbo Tao, Ziawasch Abedjan, Ahmed Elmagarmid, Guoliang Li, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Unsupervised string transformation learning for entity consolidation. *arXiv preprint arXiv:1709.10436*, 2017.

[52] Artan Dermaku, Tobias Ganzow, Georg Gottlob, Ben Mcmahan, Nysret Musliu, and Marko Samer. Heuristic methods for hypertree decomposition. In *Proceedings of the 7th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence*, MICAI '08, pages 1–11, Berlin, Heidelberg, 2008. Springer-Verlag.

[53] Ilias Diakonikolas. Learning structured distributions. *Handbook of Big Data*, 267, 2016.

[54] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.

[55] Reinhard Diestel. *Graph theory*. Springer Publishing Company, Incorporated, 2018.

[56] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[57] Piotr Dollár and C Lawrence Zitnick. Structured forests for fast edge detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1841–1848, 2013.

[58] Xin Luna Dong, Laure Berti-Equille, Yifan Hu, and Divesh Srivastava. Global detection of complex copying relationships between sources. *Proceedings of the VLDB Endowment*, 3(1-2):1358–1369, 2010.

[59] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Integrating conflicting data: the role of source dependence. *Proceedings of the VLDB Endowment*, 2(1):550–561, 2009.

[60] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Truth discovery and copying detection in a dynamic world. *Proceedings of the VLDB Endowment*, 2(1):562–573, 2009.

[61] Xin Luna Dong and Theodoros Rekatsinas. Data integration and machine learning: A natural synergy. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1645–1650. ACM, 2018.

[62] Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: Selecting sources wisely for integration. *Proceedings of the VLDB Endowment*, 6(2):37–48, 2012.

[63] Xin Luna Dong and Divesh Srivastava. Compact explanation of data fusion decisions. In *Proceedings of the 22nd international conference on World Wide Web*, pages 379–390. ACM, 2013.

[64] Uwe Draisbach and Felix Naumann. Dude: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, volume 100000, page 10000000, 2010.

[65] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2006.

[66] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. 19(1), 2007.

[67] Seyda Ertekin, Jian Huang, and C. Lee Giles. Active learning for class imbalance problem. SIGIR '07, pages 823–824, New York, NY, USA, 2007. ACM.

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

[69] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011.

[70] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. Towards certain fixes with editing rules and master data. 21(2):213–238, 2012.

[71] Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.

[72] Henning Fernau. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521–541, 2009.

[73] Peter A. Flach and Iztok Savnik. Database dependency discovery: A machine learning approach. *AI Commun.*, 12(3):139–160, 1999.

[74] Gerald B Folland. *Introduction to partial differential equations*. Princeton university press, 2020.

[75] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[76] Dylan Foster, Karthik Sridharan, and Daniel Reichman. Inference in sparse graphs with pairwise measurements and side information. In *International Conference on Artificial Intelligence and Statistics*, pages 1810–1818. PMLR, 2018.

[77] Dylan J. Foster, Karthik Sridharan, and Daniel Reichman. Inference in sparse graphs with pairwise measurements and side information. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, pages 1810–1818, 2018.

[78] Andrew Gainer-Dewar and Paola Vera-Licona. The minimal hitting set generation problem: Algorithms and computation. *SIAM J. Discrete Math.*, 31(1):63–100, 2017.

[79] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. Corroborating information from disagreeing views. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 131–140. ACM, 2010.

[80] Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1167–1176. Society for Industrial and Applied Mathematics, 2006.

[81] Amir Globerson, Gal Chechik, Fernando Pereira, and Naftali Tishby. Euclidean embedding of co-occurrence data. *JMLR*, 8:2265–2295, December 2007.

[82] Amir Globerson, Tim Roughgarden, David Sontag, and Cafer Yildirim. How hard is inference for structured prediction? In *International Conference on Machine Learning*, pages 2181–2190. PMLR, 2015.

[83] Amir Globerson, Tim Roughgarden, David Sontag, and Cafer Yildirim. How hard is inference for structured prediction? In *International Conference on Machine Learning*, pages 2181–2190, 2015.

[84] Oded Goldreich and Dana Ron. On estimating the average degree of a graph. *Electronic Colloquim on Computational Complexity (ECCC)*, 2004.

[85] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[86] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[87] Leo A Goodman et al. On the estimation of the number of classes in a population. *The Annals of Mathematical Statistics*, 20(4):572–579, 1949.

[88] Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The most probable database problem. 2014.

[89] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[90] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1321–1330, 2017.

[91] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

[92] Haibo He and Yunqian Ma. *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, 1st edition, 2013.

[93] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[94] Alireza Heidari, Ihab F Ilyas, and Theodoros Rekatsinas. Approximate inference in structured instances with noisy categorical observations. *arXiv preprint arXiv:1907.00141*, 2019.

[95] Alireza Heidari, Shrinu Kushagra, and Ihab F Ilyas. On sampling from data with duplicate records. *arXiv preprint arXiv:2008.10549*, 2020.

[96] 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.

[97] Alireza Heidari, George Michalopoulos, Shrinu Kushagra, Ihab F Ilyas, and Theodoros Rekatsinas. Record fusion: A learning approach. *arXiv preprint arXiv:2006.10208*, 2020.

[98] Joseph M Hellerstein. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*, 2008.

[99] Mauricio A Hernández and Salvatore J Stolfo. The merge/purge problem for large databases. In *ACM Sigmod Record*, volume 24, pages 127–138. ACM, 1995.

[100] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Distributed Representations, pages 77–109. MIT Press, Cambridge, MA, USA, 1986.

[101] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[102] https://github.com/ipeirotis/Get Another-Label/tree/master/data. Adult datset.

[103] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

[104] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[105] Zhipeng Huang and Yeye He. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1377–1392, 2018.

[106] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

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

[108] Ihab F Ilyas and Xu Chu. *Data Cleaning*. Morgan & Claypool, 2019.

[109] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[110] Yuan Jin, Mark Carman, Ye Zhu, and Yong Xiang. A technical survey on statistical modelling and design methods for crowdsourcing quality control. *Artificial Intelligence*, page 103351, 2020.

[111] Shantanu Joshi and Christopher Jermaine. Materialized sample views for database approximation. *IEEE Transactions on Knowledge and Data Engineering*, 20(3):337–351, 2008.

[112] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[113] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

[114] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[115] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230, 2015.

[116] Hyun-Chul Kim and Zoubin Ghahramani. Bayesian classifier combination. In *Artificial Intelligence and Statistics*, pages 619–627, 2012.

[117] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[118] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[119] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[120] Jyrki Kivinen and Heikki Mannila. *Approximate dependency inference from relations*, pages 86–98. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

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

[122] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[123] Sanjay Krishnan, Jiannan Wang, Michael J Franklin, Ken Goldberg, Tim Kraska, Tova Milo, and Eugene Wu. Sampleclean: Fast and reliable analytics on dirty data. *IEEE Data Eng. Bull.*, 38(3):59–75, 2015.

[124] Evgeny Krivosheev, Siarhei Bykau, Fabio Casati, and Sunil Prabhakar. Detecting and preventing confused labels in crowdsourced data. *Proceedings of the VLDB Endowment*, 13(12):2522–2535, 2020.

[125] Shrinu Kushagra. Theoretical foundations for efficient clustering. 2019.

[126] Shrinu Kushagra, Shai Ben-David, and Ihab Ilyas. Semi-supervised clustering for de-duplication. *arXiv preprint arXiv:1810.04361*, 2018.

[127] Shrinu Kushagra, Hemant Saxena, Ihab F Ilyas, and Shai Ben-David. A semi-supervised framework of clustering selection for de-duplication. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 208–219. IEEE, 2019.

[128] Shrinu Kushagra, Yaoliang Yu, and Shai Ben-David. Provably noise-robust, regularised k-means clustering. *arXiv preprint arXiv:1711.11247*, 2017.

[129] Jeongyeol Kwon and Constantine Caramanis. Em algorithm is sample-optimal for learning mixtures of well-separated gaussians. *arXiv preprint arXiv:2002.00329*, 2020.

[130] Rémi Lebret and Ronan Collobert. Word embeddings through hellinger pca. EACL, 2014.

[131] Ming-Hui Li, Lei Lin, Xiao-Long Wang, and Tao Liu. Protein–protein interaction site prediction based on conditional random fields. *Bioinformatics*, 23(5):597–604, 2007.

[132] Qi Li, Yaliang Li, Jing Gao, Lu Su, Bo Zhao, Murat Demirbas, Wei Fan, and Jiawei Han. A confidence-aware approach for truth discovery on long-tail data. *Proceedings of the VLDB Endowment*, 8(4):425–436, 2014.

[133] Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Zhilei Yin. Discovering approximate functional dependencies from distributed big data. In *APWeb*, pages 289–301, 2016.

[134] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? *Proceedings of the VLDB Endowment*, 6(2):97–108, 2012.

[135] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. Discover dependencies from data - A review. *IEEE Trans. Knowl. Data Eng.*, 24(2):251–264, 2012.

[136] Ester Livshits, Alireza Heidari, Ihab F Ilyas, and Benny Kimelfeld. Approximate denial constraints. *arXiv preprint arXiv:2005.08540*, 2020.

[137] Ester Livshits, Ihab F. Ilyas, Benny Kimelfeld, and Sudeepa Roy. Principles of progress indicators for database repairing. *CoRR*, abs/1904.06492, 2019.

[138] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing optimal repairs for functional dependencies. In *PODS*, pages 225–237, 2018.

[139] Andrei Lopatenko and Leopoldo E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, pages 179–193, 2007.

[140] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *EDBT*, pages 350–364, 2000.

[141] Francois Lorrain and Harrison C White. Structural equivalence of individuals in social networks. *The Journal of mathematical sociology*, 1(1):49–80, 1971.

[142] Xinghua Lu, Bin Zheng, Atulya Velivelli, and ChengXiang Zhai. Enhancing text categorization with semantic-enriched representation and training data augmentation. *Journal of the American Medical Informatics Association*, 13(5):526–535, 2006.

[143] Jonas Lukasczyk, Eric Kinner, James Ahrens, Heike Leitte, and Christoph Garth. Voidga: A view-approximation oriented image database generation approach. In *2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 12–22. IEEE, 2018.

[144] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. Citeseer, 2000.

[145] Tomas Mikolov, Ilya Sutskever, Kai Chen, et al. Distributed representations of words and phrases and their compositionality. NIPS, 2013.

[146] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[147] Stefan Uhlich; Marcello Porcu; Franck Giron; Michael Enenkl; Thomas Kemp; Naoya Takahashi; Yuki Mitsufuji. Improving music source separation based on dnns through data augmentation and network blending. 2017.

[148] Keisuke Murakami and Takeaki Uno. Efficient algorithms for dualizing large-scale hypergraphs. *Discrete Applied Mathematics*, 170:83–94, 2014.

[149] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[150] Jennifer Neville and David Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.

[151] Lhouari Nourine, Alain Quilliot, and Hélène Toussaint. Partial enumeration of minimal transversals of a hypergraph. In *CLA*, pages 123–134, 2015.

[152] Noel Novelli and Rosine Cicchetti. FUN: an efficient algorithm for mining functional and embedded dependencies. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, pages 189–203, 2001.

[153] Sebastian Nowozin, Peter V Gehler, Jeremy Jancsary, and Christoph H Lampert. *Advanced Structured Prediction*. MIT Press, 2014.

[154] Sebastian Nowozin, Christoph H Lampert, et al. Structured learning and prediction in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

[155] Adrian Weller Ofer Meshi, Mehrdad Mahdavi and David Sontag. Train and test tightness of lp relaxations in structured prediction. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1776–1785, 2016.

[156] Laurel Orr, Magdalena Balazinska, and Dan Suciu. Sample debiasing in the themis open world database system. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 257–268, 2020.

[157] Jason W Osborne. *Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data*. Sage, 2013.

[158] 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. *PVLDB*, 8(10):1082–1093, 2015.

[159] Jeff Pasternack and Dan Roth. Knowing what to believe (when you already know something). In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 877–885. Association for Computational Linguistics, 2010.

[160] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. Discovery of approximate (and exact) denial constraints. *PVLDB*, 13(3), 2019.

[161] Eduardo H. M. Pena and Eduardo Cunha de Almeida. BFASTDC: A bitwise algorithm for mining denial constraints. In *DEXA*, pages 53–68, 2018.

[162] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[163] J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In *Advances in Large Margin Classifiers*, 2000.

[164] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[165] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J Miller, and Divesh Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4):300–311, 2015.

[166] Erhard Rahm and Hong-Hai Do. Data cleaning: Problems and current approaches. *DE*, 23(4):3–13, 2000.

[167] Joeri Rammelaere and Floris Geerts. Explaining repaired data with cfds. *Proc. VLDB Endow.*, 11(11):1387–1399, July 2018.

[168] Joeri Rammelaere and Floris Geerts. Revisiting conditional functional dependency discovery: Splitting the "c" from the "fd". In *ECML/PKDD (2)*, volume 11052 of *Lecture Notes in Computer Science*, pages 552–568. Springer, 2018.

[169] Joeri Rammelaere, Floris Geerts, and Bart Goethals. Cleaning data with forbidden itemsets. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 897–908. IEEE, 2017.

[170] Sofya Raskhodnikova, Dana Ron, Amir Shpilka, and Adam Smith. Strong lower bounds for approximating distribution support size and the distinct elements problem. *SIAM Journal on Computing*, 39(3):813–842, 2009.

[171] John W. Ratcliff and David E. Metzener. Pattern matching: The gestalt approach. *Dr. Dobb's Journal of Software Tools*, 13(7):46, 47, 59–51, 68–72, July 1988.

[172] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

[173] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

[174] Alexander J. Ratner, Henry R. Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. Learning to compose domain-specific transformations for data augmentation. In *NIPS* [174], pages 3239–3249.

[175] Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. *Journal of Machine Learning Research*, 11(4), 2010.

[176] Christopher Ré. Software 2.0 and snorkel: Beyond hand-labeled data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2876–2876. ACM, 2018.

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

[178] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11):1190–1201, 2017.

[179] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11):1190–1201, 2017.

[180] Theodoros Rekatsinas, Manas Joglekar, Hector Garcia-Molina, Aditya Parameswaran, and Christopher Ré. Slimfast: Guaranteed results for data fusion and source reliability. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1399–1414. ACM, 2017.

[181] Michael Renardy and Robert C Rogers. *An introduction to partial differential equations*, volume 13. Springer Science & Business Media, 2006.

[182] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A formal framework for probabilistic unclean databases. ICDT, 2019.

[183] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[184] Burr Settles. Active learning. *Synthesis lectures on artificial intelligence and machine learning*, 6(1):1–114, 2012.

[185] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[186] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment*, 8(11):1310–1321, 2015.

[187] Narayanan Shivakumar and Hector Garcia-Molina. Scam: A copy detection mechanism for digital documents. 1995.

[188] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006.

[189] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

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

[191] Nikki Swartz. Gartner warns firms of'dirty data'. *Information Management*, 41(3):6, 2007.

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

[193] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[194] Matteo Venanzi, John Guiver, Gabriella Kazai, Pushmeet Kohli, and Milad Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *Proceedings of the 23rd international conference on World wide web*, pages 155–164, 2014.

[195] Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010.

[196] Staal A. Vinterbo and Aleksander Öhrn. Minimal approximate hitting sets and rule templates. *Int. J. Approx. Reasoning*, 25(2):123–143, 2000.

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

[198] Jiannan Wang and Nan Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, pages 457–468, 2014.

[199] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[200] William E Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.

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

[202] Minji Wu and Amélie Marian. A framework for corroborating answers from multiple web sources. *Information Systems*, 36(2):431–449, 2011.

[203] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *DaWaK*, pages 101–110, 2001.

[204] Xiaoxin Yin, Jiawei Han, and S Yu Philip. Truth discovery with multiple conflicting information providers on the web. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):796–808, 2008.

[205] Xiaoxin Yin and Wenzhao Tan. Semi-supervised truth discovery. In *Proceedings of the 20th international conference on World wide web*, pages 217–226. ACM, 2011.

[206] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. 2017.

[207] Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. Highway long short-term memory rnns for distant speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5755–5759. IEEE, 2016.

[208] Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. Truth inference in crowdsourcing: Is the problem solved? *Proceedings of the VLDB Endowment*, 10(5):541–552, 2017.

[209] Dengyong Zhou, Sumit Basu, Yi Mao, and John C Platt. Learning from the wisdom of crowds by minimax entropy. In *Advances in neural information processing systems*, pages 2195–2203, 2012.

[210] Xiaojin Zhu. Semi-supervised learning tutorial. In *International Conference on Machine Learning (ICML)*, pages 1–135, 2007.