EXTRACTING AND UTILIZING HIDDEN STRUCTURES IN LARGE DATASETS

BY

YIHAN GAO

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:

Assistant Professor Aditya Parameswaran, Chair
Professor Kevin Chang
Associate Professor Hari Sundaram
Assistant Professor Jiannan Wang, Simon Fraser University

ABSTRACT

The hidden structure within datasets — capturing the inherent structure within the data not explicitly captured or encoded in the data format — can often be automatically extracted and used to improve various data processing applications. Utilizing such hidden structure enables us to potentially surpass traditional algorithms that do not take this structure into account. In this thesis, we propose a general framework for algorithms that automatically extract and employ hidden structures to improve data processing performance, and discuss a set of design principles for developing such algorithms. We provide three examples to demonstrate the power of this framework in practice, showcasing how we can use hidden structures to either outperform state-of-the-art methods, or enable new applications that are previously impossible. We believe that this framework can offer new opportunities for the design of algorithms that surpass the current limit, and empower new applications in database research and many other data-centric disciplines.

To my father, whose love and support helped me get through difficult times.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor Aditya Parameswaran for his tremendous help during my years under his supervision. His valuable research insights have made this thesis possible, and I will also benefit from the life experiences he shared with me for the rest of my career.

I would like to thank Dr. Kevin Chang, Dr. Hari Sundaram, and Dr. Jiannan Wang, for taking their valuable time to be on my thesis committee and for providing helpful suggestions on various issues.

I would also like to thank all my friends who have given me a variety of advice on thesis writing during the past year. Their suggestions have improved the quality of this work in a number of ways.

Finally, I would like to thank all the anonymous reviewers of my papers. Their suggestions have greatly improved the quality of my prior papers, which is reflected in this work.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
1.1 Thesis Outline 3

CHAPTER 2 THE GENERAL FRAMEWORK 4
2.1 Design Principles for the Structure Hypothesis Space 5
2.2 Common Design Patterns 7

CHAPTER 3 EXTRACTING STRUCTURE FOR COMPRESSION 9
3.1 Preliminaries 11
3.2 Bayesian Network Structure Extraction 13
3.3 SquID: Supporting Complex Attribute Types 15
3.4 Precision-Aware Compression & Decompression 20
3.5 Discussion: Examples, Optimality, and Streaming 26
3.6 Experiments 31
3.7 Bibliographical Notes 38
3.8 Conclusion 39

CHAPTER 4 EXTRACTING STRUCTURE FOR ORGANIZATION 40
4.1 Structure Hypothesis Space 42
4.2 Overview of the Datamaran Algorithm 45
4.3 The Generation Step 47
4.4 The Pruning Step 51
4.5 Structure Refinement 53
4.6 Regularity Score Function 55
4.7 Theoretical Analysis 56
4.8 Experiments 57
4.9 Bibliographical Notes 71
4.10 Conclusion 75

CHAPTER 5 EXTRACTING STRUCTURE FOR INSIGHT DISCOVERY 78
5.1 The Calibration Property of Conditional Probabilities 79
5.2 Approaches for Relational Learning 92
5.3 Preliminary Experiments 96
5.4 Discussion 99

CHAPTER 6 CONCLUSION 101

REFERENCES 102

CHAPTER 1: INTRODUCTION

Most real-world datasets are intrinsically structured: their data contents encode real world objects and/or their relationships, which necessarily follows the natural orders that restrict the data contents in various ways. Another reason for structure is the programmatic generation of data, where the data is restricted to a fixed structure encoding semantics of the application as specified by the programmer. These restrictions, when properly captured, summarized and formally expressed, become the *structures* of datasets that can be used by subsequent data processing applications. Dataset structures can be broadly categorized into two types: *explicit structures* are straightforward implications of the data formatting (e.g., XML datasets follow a tree-style structure, or relational datasets consist of tuples with the same schema), while *hidden structures* are statistical/logical restrictions on the actual data contents, which are usually not as apparent, and require additional effort to be discovered and extracted from the datasets.

Even though the general problem of extracting hidden structures from datasets has been an active research topic for many years, most of the existing algorithms are designed for consumption by human analysts, with applications primarily in the form of data summarization or visualization. These algorithms can find useful, human-interpretable structures such as association rules [1] or data point clusters [2] that provide analysis with insights into the contents of the datasets, and thus deepen their understanding on the nature of those datasets. For these algorithms, the data analysts serve as a bridge between the structure extraction algorithm and the actual application: the insights gained from the extracted structures are manually encoded into the actual application programs, with the intention of improving their efficiency/performance on similar datasets from the same source.

The above design philosophy is deeply rooted in the design of most existing structure extraction algorithms. However, in recent years, many new applications have begun to include structure extraction modules as part of their fully automated data processing pipelines. In these applications, the discovered structures are automatically used to support downstream data processing, without ever being exposed to any human during the process. Initially, such a shifted paradigm resulted from the usage of increasingly complex structures (e.g., deep neural networks) that are difficult for human analysts to understand but provide superior utility for subsequent data processing applications. However, some researchers have now begun to realize another equally important benefit of not involving data analysts in the process: since the discovered structures do not need to be exposed, it enables the usage of structure extraction modules in applications that are traditionally fully automated. For

instance, Kraska et al. [3] have argued that in some cases "learned" indexing structures (e.g., neural networks) that are built specifically from the input dataset can potentially outperform traditional B-tree based indexing structures.

Extracting hidden structures that are directly used in downstream data processing applications presents a very different set of requirements for algorithm design:

- *The extracted structures do not need to be intuitively interpretable.* Since the extracted structures are never directly exposed to users, their simplicity or interpretability are somewhat irrelevant. Complicated and unintuitive structures may in fact be preferable if they have better modeling capabilities.

- *The extraction procedure needs to be fully automated.* Since human analysts can no longer validate the extracted structures via manual inspection, the structure extraction method needs to ensure that it can extract meaningful structures for any input dataset, without requiring users to perform "quality checks" and adjustments.

- *The structures need to be useful for subsequent data processing.* We need to design the structure extraction module in conjunction with the actual data processing method, such that the extracted structures can be used naturally in the subsequent steps to improve the overall effectiveness/efficiency of the application.

Therefore, designing structure extraction modules for automated data processing can be drastically different from existing approaches, and an interesting research question would be: *are there any general philosophies/frameworks that can help us with the design of such modules?* In this thesis we try to answer this question by summarizing the common characteristics of several example structure extraction algorithms for automated data processing. As we shall see, to utilize automatically extracted hidden structures for data processing, most algorithms will follow the generic two-step framework below:

- **Extraction Step.** During this step, we extract the hidden structure of datasets by searching through the hypothesis space to identify the most appropriate candidate that best fits the dataset.

- **Data Processing Step.** During this step, we utilize the extracted structure to accomplish the actual data processing task.

Among the two steps, the extraction step is usually the focus of algorithm design, while the actual data processing step is relatively simple in most cases. Generally speaking, we want to design the extraction step so that it can reliably extract the hidden structure of any

input dataset, which is achieved by properly choosing and limiting the structure hypothesis space (i.e., the total collection of candidates that we want to consider as our extracted structure) and developing efficient and effective learning methods for selecting from within this hypothesis space. While the exact algorithm design is usually task dependent, there are several general principles that are commonly applicable, and we shall explain them in detail and discuss how they were applied in each of our example applications to lead us towards our final algorithm design.

1.1 THESIS OUTLINE

The rest of this thesis is organized as follows: in Chapter 2, the general algorithmic framework for hidden structure based data processing is presented, and we will list some general design principles/patterns along with the reasoning behind them. In Chapter 3 and 4, two of our published works are presented as examples, and we will discuss how the general principles were applied and affected our algorithm design. In Chapter 5, we will talk about our ongoing work on relational data imputation; we will go through several different approaches for this task, and present some preliminary results. Finally, we will conclude in Chapter 6. Related works will be covered in the appropriate chapters.

Remark: Most of the contents in Chapter 3 was published in [4], and Chapter 4 primarily of contents from [5]. Part of the contents in Chapter 5 was originally published in [6]. Readers are welcome to skip directly to Chapter 3, 4 or 5 for more concrete examples, then revisit the general principles in Chapter 2 as needed.

CHAPTER 2: THE GENERAL FRAMEWORK

Broadly speaking, structures can be viewed as constraints that restrict the value ranges or forms of the data contents. By understanding the structures of datasets, one essentially reduces the space of "valid" inputs to be considered, which is generally beneficial in the sense that more specialized algorithms can be designed to achieve potentially better performance. For instance, while binary compression algorithms can be used to compress any kind of dataset, the specialized compression algorithms (e.g., for images or movies) usually perform much better.

Traditional data processing applications often use only the explicit structure of datasets, and are usually designed to optimize the worst-case time complexity. Consequently, these algorithms can work reasonably well for any input dataset conforming to the corresponding explicit structure, and their performance/efficiency variance is relatively low. While such an approach has its own advantages, it completely gives up any opportunity to perform better when the input datasets have other properties in addition to their explicit structure. In fact, most real world datasets do have hidden structures that naturally result from the semantics of their data contents. If we can identify and utilize these hidden structures in our data processing procedures, we may be able to go beyond the limits of traditional approaches.

Unfortunately, even though hidden structures are clearly present in the datasets, they are usually not directly accessible: for instance, functional dependencies in relational datasets are often not explicitly described in the meta-data; the clustering property of data points (i.e., that most data points are originally from a small number of clusters) that exists in many datasets are also not expressed directly in most cases. Generally speaking, an extraction step needs to be carried out first before hidden structures can be used in the data processing step, and as a result most of the hidden structure based algorithms will follow the generic two-step framework as shown in Figure 2.1.

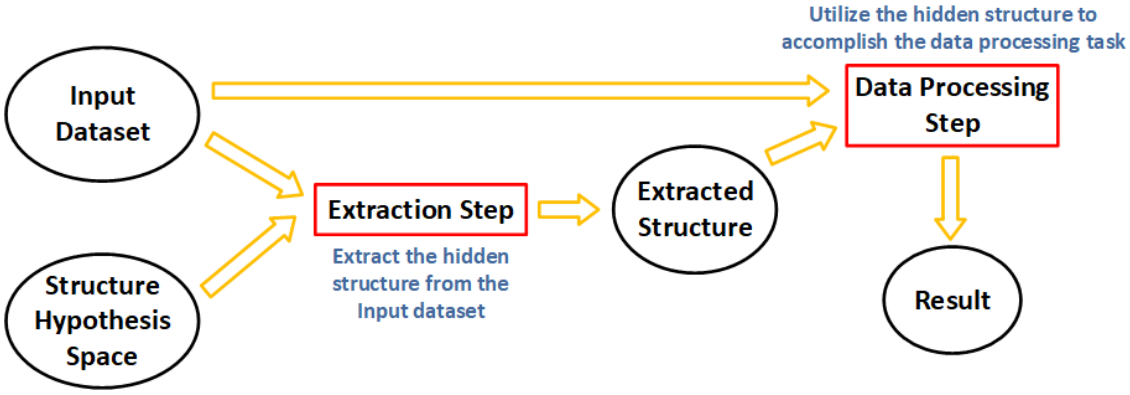


Figure 2.1: General framework for hidden structure based algorithms

There are three major components that we need to make decisions on when designing algorithms within such a framework: the *structure hypothesis space*, the *extraction method*, and the *data processing method*. Intuitively, the structure hypothesis space captures what type of hidden structures we want to discover from the datasets, and the extraction/data processing method captures how we plan to extract/utilize these hidden structures. For clarity, the terminologies used in this chapter are listed in Table 2.1 (with explanations) and demonstrated via a simple example in Figure 2.2, in which the left hand side shows an example tabular dataset with two functionally dependent columns, and the right hand side demonstrates 4 of the concepts using this example. Here, the input space consists of all tabular datasets with two numerical attributes X and Y, and the structure extraction algorithm tries to identify the functional dependency relationship between them. The structure hypothesis space specifies $Y = f(X, \theta)$, the parametric form of functions considered in the structure extraction algorithm, and Θ, the range of parameters. Each structure candidate, $Y = f(X, \theta_0)$, is one particular instantiation of parameters, and the extracted structure, $Y = 0.5X + 0.5$, is the function (within the hypothesis space) that best fits the input dataset.

Concept	Explanation
Input Space	The collection of all datasets that can potentially appear as input to the structure extraction algorithm
Input Dataset	The specific dataset that appears as the input to the algorithm (for one specific run)
Structure Candidate	A hidden structure that could potentially appear in an input dataset
Structure Hypothesis Space	The collection of all possible structure candidates that we are going to consider in the structure extraction algorithm
Extracted Structure	The structure candidate extracted from the structure hypothesis space using the input dataset
Extraction Method	The procedure for searching over for the structure hypothesis space to find the extracted structure
Data Processing Method	The procedure for accomplishing the intended data processing task using the extracted structure

Table 2.1: Terminology used in this Chapter

2.1 DESIGN PRINCIPLES FOR THE STRUCTURE HYPOTHESIS SPACE

Choosing an appropriate structure hypothesis space is the most critical step of the entire structure extraction algorithm design. Ideally, the hypothesis space should be vast and comprehensive, while each individual structure candidate should be as precise and informative as possible (i.e., restricting the value flexibility of the data contents). At the same time, we need to take into account the feasibility of efficient extraction in practice. More specifically,

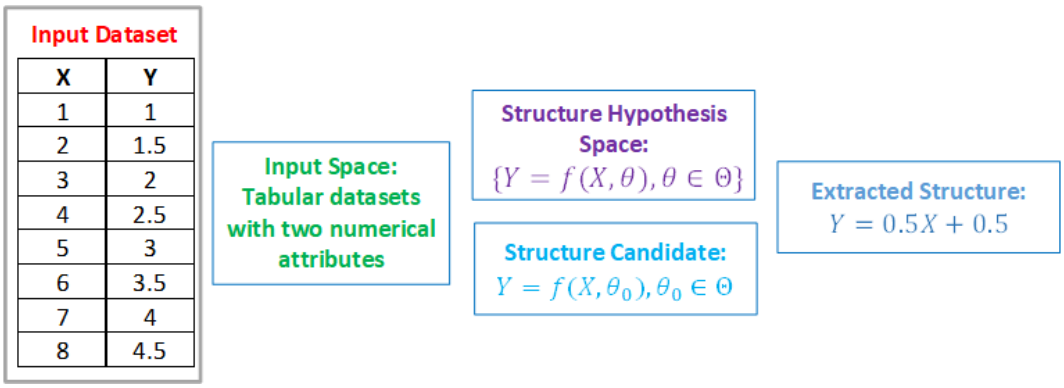


Figure 2.2: Example Tabular Dataset with a Functional Dependency

there are three general principles that we need to consider when designing the structure hypothesis space:

- **Representativeness.** When designing the structure hypothesis space, it is important to ensure that for any potential input dataset, there exists a structure candidate that can capture its essential characteristics. For instance, structures that capture the correlations between adjacent data entries would be always applicable for any continuous time series signal. On the other hand, if we want to extract repetitive patterns as structures, such an approach is only applicable if the input signal is actually periodic, and the representativeness principle would be violated if the input space also contains non-periodic signals.

- **Learnability.** While complex structures can be precise and informative for describing the input dataset, it comes with the cost of being more difficult to extract: complex structures are usually associated with a larger and more complicated structure hypothesis space, which makes it much more difficult to search through/optimize over this space. Generally speaking, the difficulty of extraction increases proportionally with respect to the complexity of structure candidates, and it is necessary to limit the complexity of the structure candidates when choosing the structure hypothesis space.

- **Utility.** It is also important to ensure that the extracted structure can be useful for the subsequent data processing step. For many datasets, it is possible to capture their structures from many different perspectives. However, not all of them would contain all of the essential information necessary for the downstream data processing task. Therefore, it is important to ensure that the structure candidates are capable of capturing all of the relevant characteristics of the input dataset. For instance, if we are detecting grammatical errors in natural language documents, then unigram (i.e.,

bag-of-word) structures will not help us much, since it ignores all of the dependencies between adjacent words, which are critical for our application. In addition, the exact representation of structure candidates should be designed in such a way that makes the data processing step as easy as possible: we should always choose the most appropriate structural form for capturing dataset characteristics, and a well-chosen structural form can lead to significant efficiency improvements over a poorly-chosen one.

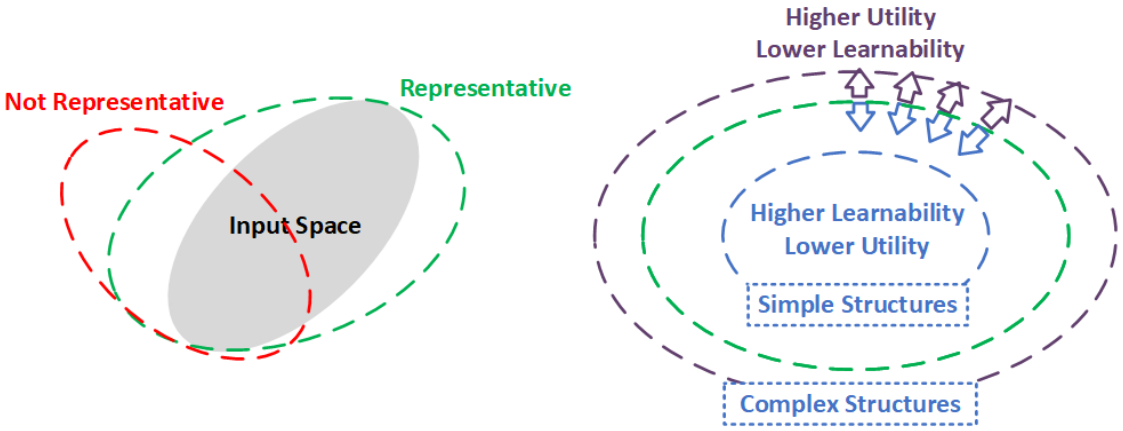


Figure 2.3: Structure Hypothesis Space Design Principles

These three principles are demonstrated in Figure 2.3 along with the relationships between them. Generally speaking, the utility principle favors complex structure candidates and large hypothesis spaces, while the learnability principle favors small structure hypothesis spaces and simple candidates. The representativeness principle requires us to identify the type of structures that are ubiquitous for the input space, and usually determines the basic form of extracted structure. In most cases, it is easier to start from the representativeness principle, and then try to balance between the utility and learnability principles afterwards.

2.2 COMMON DESIGN PATTERNS

While the details of each hidden structure based algorithm are task dependent, there are several common design patterns for such algorithms, which are summarized from our example applications and listed in the following:

- **Use Universal Structures.** The representativeness principle requires us to be able to extract a proper hidden structure for most input datasets. For applications with a large input space, the characteristics of input datasets can vary significantly from one

another, making it difficult to satisfy the representativeness requirement. One common approach that we adopt in such scenarios is to use *universal structures*[1], which are highly flexible and can adapt to many different situations. Compared to the alternative of considering multiple types of structures at once, universal structures allow us to handle all scenarios in a consistent manner, reducing the engineering complexity.

- **Use Sampling Technique for Structure Extraction.** The structure extraction algorithm's overall efficiency depends on the amount of time we spend on both the extraction and data processing steps. While the data processing step usually requires a fixed amount of time (which only depends the size of input dataset), the amount of time we spend on structure extraction is often flexible. Thus, tuning the total running time of structure extraction step is key to achieving an optimal trade-off between effectiveness and efficiency. One common way to achieve this is to use sampling: intuitively, it is not necessary for us to use the entire input dataset for structure extraction, and a subset is often sufficient. Sampling just the right amount of the input dataset allows us to significantly improve overall efficiency while not sacrificing the quality of extracted structure by much.

- **Use Approximation Algorithms.** Recall that the goal of the extraction step is essentially to search over the structure hypothesis space and find the optimal candidate. However, for large structure hypothesis spaces, searching over the entire space is often too time consuming and impractical. One common technique to be used in such scenarios is to adopt approximation algorithms: for the eventual purpose of improving data processing, it is not really necessary to use the optimal structure. Rather, any hidden structure that offers sufficient details would suffice. When the time budget is limited and exhaustive search is impractical, approximation algorithms can be a good alternative.

These design patterns are summarized from our example applications, and we will see how these design patterns are applied in practice in later chapters, which should help readers better understand the intuition behind them, so that they can successfully apply these ideas in other applications as well.

[1]Universal structure is a concept in Machine Learning. Intuitively, it denotes the type of structure that can approximate most other functions with carefully chosen parameters. Neural network is the most well-known universal structure.

CHAPTER 3: EXTRACTING STRUCTURE FOR COMPRESSION

Today, data is being generated at an alarming rate from all sorts of activities ranging from social media and commercial transactions to scientific simulations and internet-of-things. Consequently, the volume of structured (relational) datasets is also growing rapidly. Having effective compression for these relational datasets can significantly reduce their storage cost, which would benefit most organizations, including companies, governments and universities.

While the task of generic data compression [7, 8] has been extensively studied in the past, compression of tabular datasets has received limited attention. Generic algorithms (such as Lempel-Ziv [7]) do not exploit the relational structure of the datasets at all. Some paper provide partial solutions: Spartan [9] uses the relational structure by identifying functional dependencies between attributes; ItCompress [10] finds the clustering structure of tuples in the dataset to achieve better compression; column-based approaches [11] exploit the skewness of attribute values instead.

Clearly, these existing papers on tabular dataset compression are attempting to identify a certain type of hidden structure within the dataset. However, because of the flexibility of relational formats, the input space is extremely large and we don't really know what kind of hidden structure would be most useful for any given dataset. Therefore, as we have discussed in Chapter 2, a better strategy would be to use universal structures that are flexible and can be adapted to approximate other structures. Universal structures for tabular datasets are probabilistic models for the joint-distribution of multiple random variables: if we think of each attribute as a random variable, then tuples in the input dataset will be their joint-instantiations. Note that all other hidden structures we previously mentioned can be expressed as special cases of this universal structure: functional dependencies can be viewed as deterministic conditional distributions; clustering of tuples can be expressed via mixture models; and skewness of attribute values can be denoted via skewed marginal distributions.

Probabilistic Graphical Models [12] (PGMs) are the canonical structures for capturing the joint distribution of multiple random variables. PGMs use graphs (directed or undirected) to capture the dependencies between random variables, and additional structures called factors are used to compactly represent the distributional information. While there are many different types of PGMs, Bayesian Networks are particularly suitable for our setting, since its directed acyclic graph structure provides a natural order among attributes, allowing us to employ a sequential encoding strategy. Our tabular dataset compression algorithm SQUISHis based on this exact idea, wherein we extract the hidden Bayesian Network structure of the input dataset and utilize this structure to compress the dataset via arithmetic coding [13].

However, there are several challenges in using Bayesian Networks and Arithmetic Coding for compression. First, we need to identify a new objective function for learning a Bayesian Network, since conventional objectives like Bayesian Information Criterion [14] are not designed to minimize the size of the compressed dataset. Another challenge is to design a mechanism to support attributes with an infinite range (e.g., numerical and string attributes), since Arithmetic Coding assumes a finite alphabet for symbols, and therefore cannot be applied to those attributes. To be applicable to the wide variety of real-world datasets, it is essential to be able to handle numbers and strings. We have addressed these challenges when developing SQUISH. As we shall see in this chapter, the compression rate of SQUISH is near-optimal for all datasets that can be efficiently described using a Bayesian Network. This theoretical optimality reflects in our experiments as well: SQUISH *achieves a reduction in storage on real datasets of over 50% compared to the nearest competitor.*

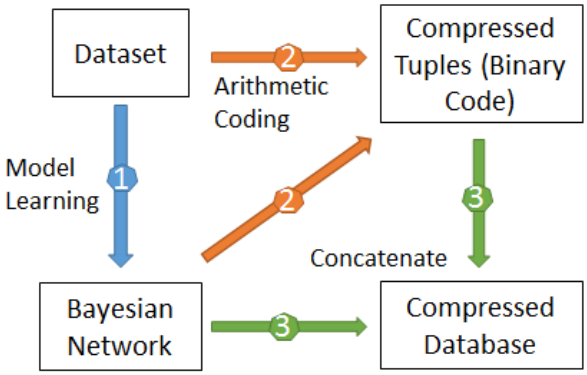


Figure 3.1: Workflow of the Compression and Decompression Algorithm

Figure 3.1 illustrates the overall workflow of SQUISH, and the details will be described in the rest of this chapter: in Section 3.1, we review the problem definition of tabular dataset compression, and related concepts such as Bayesian Networks and arithmetic coding; in Section 3.2, we discuss the details of Bayesian Network structure extraction; in Section 3.3, we introduce the SQUID mechanism for handling complex attributes, and discuss how it is used in the actual compression step; in Section 3.4, we introduce the detailed encoding and decoding procedures and discuss mechanisms for handling finite precision issue; in Section 3.5, we use examples to illustrate the effectiveness of SQUISH and prove the asymptotic optimality of the compression algorithm; experimental results are shown in Section 3.6 to demonstrate the superiority of SQUISH compared to prior methods.

3.1 PRELIMINARIES

In this section, we formally define the problem of tabular datatset compression, and provide some background on Bayesian networks and arithmetic coding.

3.1.1 Problem Definition

Suppose our dataset consists of a single relational table T, with n rows (tuples) and m columns (attributes). We want to design an encoding procedure A with its associated decoding procedure B, such that A takes T as input to generate its compressed form $C(T)$, while B reconstructs T' from $C(T)$ where T' is an approximation of T. The goal is to minimize the file size of $C(T)$ under the constraint that T' and T are close enough.

The closeness constraint of T' to T is defined as follows: For ith attribute of the table, if it is numerical, then for each tuple t and the reconstructed tuple t', $|t_i - t'_i| \leq \epsilon_i$, where ϵ_i is the error threshold provided by the user. For non-numerical attributes, the original and reconstructed attribute values must be exactly the same: $t_i = t'_i$. Note that our problem definition subsumes lossless compression as a special case with $\epsilon_i = 0$.

3.1.2 Bayesian Network

Bayesian networks [12] are widely used probabilistic models for the joint-distribution of multiple random variables. Formally, each Bayesian Network $\mathcal{B} = (\mathcal{G}, (\mathcal{M}_1, \ldots, \mathcal{M}_n))$ consists of two components: the structure graph $\mathcal{G} = (V, E)$, and the conditional probability models $\mathcal{M}_1, \ldots, \mathcal{M}_n$. $\mathcal{G}$ is a directed acyclic graph with n vertices $v_1, \ldots, v_n$ corresponding to n random variables (denote them as $X_1, \ldots, X_n$). $\mathcal{M}_i$ is the conditional probability model describing the distribution of X_i conditioned on $\mathbf{X}_{parent(i)} = \{X_j : j \in parent(i)\}$, where $parent(i) = \{j : (v_j, v_i) \in E\}$ is called the set of parent nodes of v_i in $\mathcal{G}$.

Figure 3.2 shows an example Bayesian Network with three random variables X_1, X_2, X_3. The structure graph $\mathcal{G}$ is shown at top of the figure and the bottom side shows the models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$.

In SQUISH, attributes of the input datasets are associated with random variables, and Bayesian Networks are used to model their joint probability distribution. Each tuple is viewed as a joint-instantiation of all the random variables, or equivalently, a sample from the probability distribution described by the Bayesian Network.

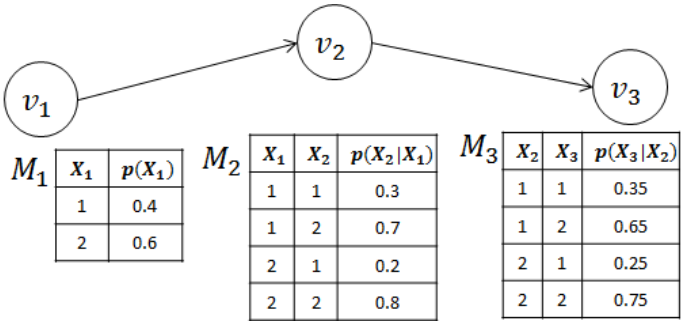


Figure 3.2: Bayesian Network Example

3.1.3 Arithmetic Coding

Arithmetic coding [13] is a state-of-the-art adaptive compression technique for a sequence of dependent characters. Formally, arithmetic coding is defined by a finite ordered alphabet $\mathcal{A}$, and a probabilistic model for a sequence of characters that specifies the probability distribution of each character X_k conditioned on all precedent characters $X_1, \ldots, X_{k-1}$. Let $\{a_n\}$ be any string of length n. To compute the encoded string for $\{a_n\}$, we first compute a probability interval for each character a_k:

$$\begin{aligned}[l_k, r_k] =&[p(X_k < a_k | X_1 = a_1, \ldots, X_{k-1} = a_{k-1}), \\ &p(X_k \leq a_k | X_1 = a_1, \ldots, X_{k-1} = a_{k-1})]\end{aligned} \tag{3.1}$$

We define the product of two probability interval as:

$$[l_1, r_1] \circ [l_2, r_2] = [l_1 + (r_1 - l_1)l_2, l_1 + (r_1 - l_1)r_2] \tag{3.2}$$

The probability interval for string $\{a_n\}$ is the product of probability intervals of all the characters in the string:

$$[l, r] = [l_1, r_1] \circ [l_2, r_2] \circ \ldots \circ [l_n, r_n] \tag{3.3}$$

Let k be the smallest integer such that there exists a non-negative integer $0 \leq M < 2^k$ satisfying:

$$l \leq 2^{-k}M, r \geq 2^{-k}(M + 1) \tag{3.4}$$

Then the k-bit binary representation of M is the encoded bit string of $\{a_n\}$.

An example to demonstrate how arithmetic coding works can be found in Figure 3.3. The three tables at the right hand side specify the probability distribution of the string $a_1a_2a_3$. The blocks at the left hand show the associated probability intervals for the strings: for example, "aba" corresponds to $[0.12, 0.204] = [0, 0.4] \circ [0.3, 1] \circ [0, 0.3]$. As we can see,

arithmetic coding maps all possible strings to disjoint probability intervals, which ensures that code words can not be the prefix of one another.

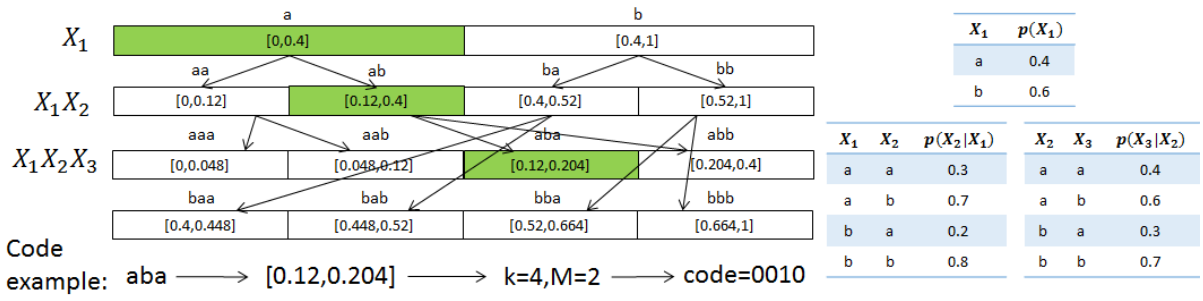


Figure 3.3: Arithmetic Coding Example

3.2 BAYESIAN NETWORK STRUCTURE EXTRACTION

Although the problem of learning Bayesian Networks has been extensively studied in literature [12], most existing methods are not ideal for our compression purpose: (a) conventional objectives like Bayesian Information Criterion (BIC) [14] are suboptimal for optimizing compression ratio; (b) commonly used combinatorial search techniques are too slow in our compression scenario, where efficiency is an equally important factor as compression rate. In Section 3.2.1, we derive the correct objective function for optimizing the compression ratio. In Section 3.2.2, we describe an efficient greedy algorithm for constructing good Bayesian Network structures and explain why it fits well in our scenario.

3.2.1 Objective Function for Bayesian Network Learning

Suppose our dataset D consists of n tuples and m attributes (denoted as $attr_1, \ldots, attr_m$). Let $\mathcal{B} = (\mathcal{G}, (\mathcal{M}_1, \ldots, \mathcal{M}_m))$ be a Bayesian network with m nodes that captures the joint probability distribution of all attributes. The total description length of D using $\mathcal{B}$ is $\mathbf{S}(D|\mathcal{B}) = \mathbf{S}(\mathcal{B}) + \mathbf{S}(\mathit{Tuples}|\mathcal{B})$, where $\mathbf{S}(\mathcal{B})$ is the description length of $\mathcal{B}$, and $\mathbf{S}(\mathit{Tuples}|\mathcal{B})$ is the total length of encoded binary strings of tuples $\{t_i\}$ (using arithmetic coding):

$$\mathbf{S}(\mathcal{B}) = \sum_{i=1}^{m} \mathbf{S}(\mathcal{M}_i) \qquad \mathbf{S}(\mathit{Tuples}|\mathcal{B}) = \sum_{i} \mathbf{S}(t_i|\mathcal{B}) \tag{3.5}$$

Here $\mathbf{S}(t_i|\mathcal{B})$ is the length of the code string of t_i, which has the following decomposition:

$$\mathbf{S}(t_i|\mathcal{B}) \approx -\sum_{j=1}^{m} \log_2 \mathbf{Pr}(a_{ij}|a_{i,parent(j)}, \mathcal{M}_j) - \sum_{j=1}^{m} num(j) \log_2 \epsilon_j + \text{const} \tag{3.6}$$

in which a_{ij} is the value of $attr_j$ in t_i, and $parent(j) = \{i : (i,j) \in \mathcal{G}\}$ is the set of parent nodes of $attr_j$ (as defined in Section 3.1.2), $num(j)$ is the indicator function of whether the $attr_j$ is numerical or not, and ϵ_j is the maximum tolerable error for $attr_j$. The reasoning behind this decomposition will be explained in Section 3.3.2. The total description length $\mathbf{S}(D|\mathcal{B})$ can then be decomposed as follows:

$$\mathbf{S}(D|\mathcal{B}) \approx \sum_{j=1}^{m} [\mathbf{S}(\mathcal{M}_j) - \sum_{i=1}^{n} \log_2 \mathbf{Pr}(a_{ij}|a_{i,parent(j)}, \mathcal{M}_j)] - n(\sum_{j=1}^{m} num(j) \log_2 \epsilon_j + \text{const}) \tag{3.7}$$

3.2.2 Greedy Structure Extraction

When optimizing Eqn (3.7), the second term (i.e., $n(\sum_{j=1}^{m} num(j) \log_2 \epsilon_j + \text{const})$) does not really involve $\mathcal{B}$. Thus for optimization purposes we only need to consider the first summation. Denote each term in the summation as obj_j:

$$obj_j = \mathbf{S}(\mathcal{M}_j) - \sum_{i=1}^{n} \log \mathbf{Pr}(a_{ij}|a_{i,parent(j)}, \mathcal{M}_j) \tag{3.8}$$

Note that once the the network structure $\mathcal{G}$ is fixed, each individual obj_j will only depend on $\mathcal{M}_j$. In that case, optimizing $\mathbf{S}(D|\mathcal{B})$ is equivalent to optimizing each obj_j individually. In other words, if we fix the graph structure $\mathcal{G}$ in advance, then the parameters of each $\mathcal{M}_j$ can be learned separately.

The general problem of finding the exact optimal graph structure $\mathcal{G}$ for Bayesian Network is NP-hard [12]. In SQUISH, we implemented a simple greedy procedure for this task. The procedure starts with an empty seed set, and repeatedly finds new attributes with the lowest obj_j, and adds these new attributes to the seed set. We also employed a sampling technique to further improve the efficiency: we only estimate obj_j using a subset of tuples, which is generally good enough for the purpose of comparing different graph structures.

The pseudo-code of the greedy procedure is shown in Algorithm 3.1, which has a worst case time complexity of $O(m^4 n)$. Here m is the number of columns and n is the total number of tuples used for Bayesian Network learning. In practice, the running time is usually around $O(m^3 n)$, which is efficient enough for practical scenarios (with sampling technique employed).

Algorithm 3.1 Greedy Structure Learning Procedure

```
function LearnStructure
    seed ← ∅
    for i = 1 to m do
        for j = 1 to m do
            parent(j) ← ∅
            while true do
                best_model_score_j ← obj_j
                best_model_j ← parent(j)
                for k ∈ seed do
                    parent(j) ← parent(j) ∪ {k}
                    Compute the value of obj_j
                    if obj_j < best_model_score_j then
                        best_model_score_j ← obj_j
                        best_model_j ← parent(j)
                    end if
                end for
                if best_model have never been updated during the for loop then
                    break
                else
                    parent(j) ← best_model_j
                end if
            end while
            parent(j) ← ∅
        end for
        Find j with minimum best_model_score_j
        seed ← seed ∪ {j}
        parent(j) ← best_model_j
    end for
end function
```

3.3 SQUID: SUPPORTING COMPLEX ATTRIBUTE TYPES

The SquID (short for Squish Interface for Data types) mechanism is introduced in Squish to address two specific issues:

- Arithmetic Coding requires a finite alphabet for each symbol. However, it is natural for attributes in a dataset to have infinite range (e.g., numerical/string attributes).
- In order to support user-defined data types, we need a mechanism for specifying the probability distributions over such data type.

A SquID is a (possibly infinite) decision tree with probability values associated with its edges, such that for every node v, the total probability of edges connecting v and v's children is equal to one. Figure 3.4 shows an example (infinite) SquID for numerical attribute, and

we can see that every edge is associated with a decision rule and a probability value: for each non-leaf node v_{2k-1}, the decision rules for edges (v_{2k-1}, v_{2k}) and (v_{2k-1}, v_{2k+1}) are $x \leq k$ and $x > k$, with probability values 0.9 and 0.1 respectively.

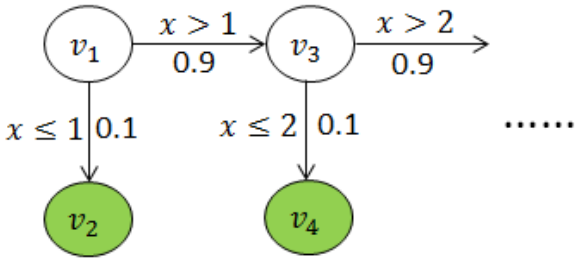


Figure 3.4: SQUID Example

SQUID is the unified interface for all different types of attributes that are supported in SQUISH, which allows us to use the same encoding & decoding procedure regardless of the exact attribute type. It also controls the maximum tolerable error in lossy compression. Denote $A(v)$ be the set of attribute values corresponding to leaf node v:

$$A(v) = \{a : \text{starting from root, } a \text{ will reach } v \text{ by following decision rules}\} \tag{3.9}$$

and let $a(v)$ be the reconstructed attribute value for v, then the maximum reconstruction error err is qual to $\sup_{v \in T} \sup_{a \in A(v)} |a - a(v)|$ for SQUID T. For instance, in Figure 3.4 we have $A(v_{2k}) = (k-1, k]$, and if we let $a(v_{2k}) = k - 0.5$, then we will have $err = 0.5$.

3.3.1 Compressing Tuples via SQUIDs

Suppose tuple t contains m attributes: $t = (a_1, \ldots, a_m)$ and we assume without loss of generally that $parent(i) \subseteq \{a_1, \ldots, a_{i-1}\}$. Let us denote T_i to be the SQUID associated with a_i (conditioned on $a_{parent(i)}$). In the following, we explain how to compute the code string of t using these SQUIDs.

To begin with, let us associate all edges in each SQUID with probability intervals. For each non-leaf node v, consider all of its children nodes $u_1, \ldots, u_k$, and denote the probability values associated with the onnecting edges as $p(v \to u_i)$. Using this notation, we define PI_T to be the following mapping from edges of SQUID T to probability intervals:

$$\text{PI}_T(v \to u_i) = [\sum_{j<i} p(v \to u_j), \sum_{j \leq i} p(v \to u_j)] \tag{3.10}$$

Now we can determine the code string of t. Let v_j be the leaf node in T_j such that

$a_j \in A(v_j)$, and denote the path from root to v_j to be $(u_{j1} \to u_{j2} \to \ldots \to u_{jk_j} \to v_j)$. Then, based on the idea of arithmetic coding, the code string of tuple t can be derived from the following probability interval:

$$\begin{aligned}[L, R] =& \mathrm{PI}_{T_1}(u_{11} \to u_{12}) \circ \ldots \circ \mathrm{PI}_{T_1}(u_{1k_1} \to v_1) \circ \\ & \mathrm{PI}_{T_2}(u_{21} \to u_{22}) \circ \ldots \circ \mathrm{PI}_{T_2}(u_{2k_2} \to v_2) \circ \ldots \circ \\ & \mathrm{PI}_{T_m}(u_{m1} \to u_{m2}) \circ \ldots \circ \mathrm{PI}_{T_m}(u_{mk_m} \to v_m) \end{aligned} \tag{3.11}$$

where $\circ$ is the probability interval multiplication operator (defined in Section 3.1.3).

3.3.2 SquIDs for Numerical Attributes

In Squish, we have implemented SquID for three primitive data types: (a) Categorical attributes with finite range; (b) Numerical attributes, either integer or float number; (c) String attributes. It is trivial to use SquID to represent the distributions of categorical attributes, and the design for string attributes is also relatively straightforward (using k-gram models).

For numerical attributes, we construct the SquID using the idea of *bisection*. Each node v is marked with an upper bound v_r and a lower bound v_l, so that every attribute value in range $(v_l, v_r]$ will pass by v on its path from the root to the corresponding leaf node. Each node has two children and a bisecting point v_m, such that the two children have ranges $(v_l, v_m]$ and $(v_m, v_r]$ respectively. The branching process stops when the range of the interval is less than 2ϵ, where ϵ is the maximum tolerable error. Figure 3.5 shows an example SquID for numerical attributes.

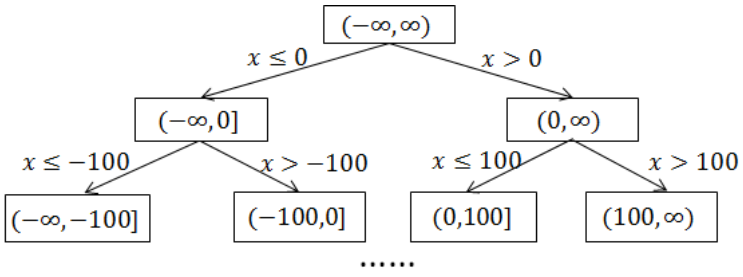


Figure 3.5: SquID for numerical attributes

Since each node represents a continuous interval, we can compute its probability using the

cumulative distribution function. The branching probability of each node is:

$$(\mathbf{Pr}(\textit{left branch}), \mathbf{Pr}(\textit{right branch})) = (\frac{\mathbf{Pr}(v_l < X \leq v_m)}{\mathbf{Pr}(v_l < X \leq v_r)}, \frac{\mathbf{Pr}(v_m < X \leq v_r)}{\mathbf{Pr}(v_l < X \leq v_r)}) \tag{3.12}$$

Clearly, the average number of bits that is needed to encode a numerical attribute depends on both the probability distribution of the attribute and the maximum tolerable error ϵ. The following theorem [4] provides a lower bound for the average number of bits required for encoding numerical attributes:

Theorem 3.1. *Let $X \in \mathcal{X} \subseteq \mathbb{R}$ be a numerical random variable with continuous support $\mathcal{X}$ and probability density function $f(X)$. Let $g : \mathcal{X} \to \{0,1\}^*$ be any uniquely decodable encoding function, and $h : \{0,1\}^* \to \mathcal{X}$ be any decoding function. If there exists a function $\rho : \mathcal{X} \to \mathbb{R}^+$ such that:*

$$\forall x, y \in \mathcal{X}, |x - y| < 2\epsilon \Rightarrow |f(x) - f(y)| \leq \rho(x) f(x) |x - y| \tag{3.13}$$

and g, h satisfies the ϵ-closeness constraint: $\forall x \in \mathcal{X}, |h(g(x)) - x| \leq \epsilon$. Then

$$E_X[len(g(X))] \geq E_X[-\log_2 f(X)] - E_X[\log_2(2\epsilon\rho(X) + 1)] - \log_2 \epsilon - 2 \tag{3.14}$$

Furthermore, if g is the bisecting code described above, then

$$E_X[len(g(X))] \leq E_X[-\log_2 f(X)] - \log_2 l + E_X[\max(\log_2 \rho(X) + \log_2 l, 0)] + 4 \tag{3.15}$$

where $l = \min_v(v_r - v_l)$ is the minimum length of probability intervals in the tree.

Proof. For any $x \in \mathcal{X}$, define $S(x)$ as $S(x) = \{y \in \mathcal{X} : g(y) = g(x)\}$, then the probability that $g(x)$ is the code word is $\mathbf{Pr}(g(x)) = \int_{y \in S(x)} f(y)dy$, and the entropy of $g(x)$ is $H(g(x)) = E_X[-\log_2 \mathbf{Pr}(g(X))]$. Since $g(x)$ is uniquely decodable, the average length must be greater than or equal to the entropy: $E_X[len(g(X))] \geq H(g(x))$. For the first part, since $-\log x$ is decreasing function, it suffices to prove that

$$\forall x \in \mathcal{X}, \mathbf{Pr}(g(x)) \leq (2\rho(x)\epsilon + 1) 4\epsilon f(x) \tag{3.16}$$

Note that due to closeness constraint, we have $S(x) \subseteq [h(g(x)) - \epsilon, h(g(x)) + \epsilon]$, which combined with the fact that $|h(g(x)) - x| \leq \epsilon$, we have $S(x) \subseteq [x - 2\epsilon, x + 2\epsilon]$ and

$$\mathbf{Pr}(g(x)) = \int_{y \in S(x)} f(y)dy \leq \int_{x-2\epsilon}^{x+2\epsilon} f(y)dy \tag{3.17}$$

The right hand side can be upper bounded by

$$\int_{x-2\epsilon}^{x+2\epsilon} f(y)dy \leq 4\epsilon(2\epsilon\rho(x)+1)f(x) \tag{3.18}$$

since by Equation (3.13),

$$\forall y \in \mathcal{X}, |x-y| < 2\epsilon \Rightarrow f(y) \leq (2\epsilon\rho(x)+1)f(x) \tag{3.19}$$

For the second part, by the property of Arithmetic Coding, we have

$$E_X[len(g(X))] \leq H(g(x)) + 2 \tag{3.20}$$

We will prove

$$\forall x \in \mathcal{X}, \mathbf{Pr}(g(x)) \geq \min(\frac{l}{2}, \frac{1}{4\rho(x)})f(x) \tag{3.21}$$

Let us suppose Equation (3.21) is true for now, then we have

$$\begin{aligned}
H(g(X)) =& E_X[-\log_2 \mathbf{Pr}(g(x))] \\
\leq& E_X[-\log_2 \min(\frac{l}{2}, \frac{1}{4\rho(x)})f(x)] \\
=& E_X[-\log_2 f(x)] - E_X[\log_2 l + \min(\log_2 \frac{1}{2\rho(x)} - \log_2 l, 0)] + 1 \\
\leq& E_X[-\log_2 f(x)] - \log_2 l + E_X[\max(\log_2 \rho(x) + \log_2 l, 0)] + 2
\end{aligned} \tag{3.22}$$

To prove Equation (3.21), let v be the leaf node that x corresponds to, we consider two cases:

1. If $v_r - v_l < \frac{1}{2\rho(x)}$, then

$$\forall y \in [v_l, v_r], f(y) \geq (1-(v_r-v_l)\rho(x))f(x) \geq \frac{1}{2}f(x) \tag{3.23}$$

   thus we have

$$\mathbf{Pr}(g(x)) \geq \frac{1}{2}(v_r - v_l)f(x) \geq \frac{1}{2}lf(x) \tag{3.24}$$

2. If $v_r - v_l \geq \frac{1}{2\rho(x)}$, then

$$\forall y \in [x - \frac{1}{2\rho(x)}, x + \frac{1}{2\rho(x)}] \cap [v_l, v_r], f(y) \geq \frac{1}{2}f(x) \tag{3.25}$$

Therefore,

$$\mathbf{Pr}(g(x)) \geq \frac{1}{2} f(x) len([v_l, v_r] \cap [x - \frac{1}{2\rho(x)}, x + \frac{1}{2\rho(x)}]) \tag{3.26}$$

Since $x \in [v_l, v_r]$, $[x - \frac{1}{2\rho(x)}, x + \frac{1}{2\rho(x)}]$ covers at least $\frac{1}{2\rho(x)}$ length of $[v_l, v_r]$, therefore,

$$\mathbf{Pr}(g(x)) \geq \frac{1}{4\rho(x)} f(x) \tag{3.27}$$

Since $\mathbf{Pr}(g(x))$ is always greater than or equal to at least one of terms, it must always be greater than or equal to the minimum of two. Thus Equation (3.21) is proved.

The tuple description length decomposition equation (i.e., Eqn (3.6)) in Section 3.2.1 was derived from Theorem 3.1, where we used the following approximation: $len(g(X)) \approx -\log_2 f(X) - \log_2 \epsilon + \text{const}$. Compared to either the upper bound or lower bound in Theorem 3.1, the only term we omitted is the term related to $\rho(X)$, which is approximately zero for most common distributions (e.g., uniform/Gaussian/Laplace) when ϵ is small [4].

Remark: Equation (3.13) is a mild assumption that holds for many common probability distributions, including uniform, Gaussian, and Laplace distributions [15].

3.4 PRECISION-AWARE COMPRESSION & DECOMPRESSION

Although we have described the detailed procedure for encoding attributes using SquID, but that procedure is only "theoretically applicable": in practice, we cannot directly compute the final probability interval of a tuple, since there could be hundreds of probability intervals in the product, and the result can easily exceed the precision limit of a floating-point number.

In our actual implementation of Squish, we implemented a precision-aware encoding module for handling the finite precision issue. Algorithm 3.2 shows the pseudo-code of this precision-aware module, in which we leveraged two tricks to deal with the finite precision problem:

- Section 3.4.1 describes the classic early bits emission trick [16].
- Section 3.4.2 describes the new deterministic approximation trick

We also employed an existing technique [17] for exploiting the orderless of tuples in tabular dataset to improve the compression. For completeness, the details of this technique will be described in Section 3.4.3.

Algorithm 3.2 Encoding Algorithm

function ARITHMETICCODING($[l_1, r_1], \ldots, [l_n, r_n]$)
  code $\leftarrow \emptyset$
  $I_t \leftarrow [0, 1]$
  for $i = 1$ to n **do**
    $I_t \leftarrow I_t \diamond [l_i, r_i]$
    while $\exists k = 0$ or $1, I_t \subseteq [\frac{k}{2}, \frac{k+1}{2}]$ **do**
      code $\leftarrow$ code $+ k$
      $I_t \leftarrow [2I_t.l - k, 2I_t.r - k]$
    end while
  end for
  Find smallest k such that
    $\exists M, [2^{-k}M, 2^{-k}(M+1)] \subseteq I_t$
  return code $+ M$
end function

3.4.1 Early Bits Emission

Without loss of generality, suppose $[L, R] = [l_1, r_1] \circ [l_2, r_2] \circ \ldots \circ [l_n, r_n]$. Define $[L_i, R_i]$ as the product of first i probability intervals: $[L_i, R_i] = [l_1, r_1] \circ [l_2, r_2] \circ \ldots \circ [l_i, r_i]$. If there exist positive integer k_i and non-negative integer M_i such that $2^{-k_i}M_i \leq L_i < R_i \leq 2^{-k_i}(M_i + 1)$, then the first k_i bits of the code string of t must be the binary representation of M_i.

Define $[L_i', R_i'] = [2^{k_i}L_i - M_i, 2^{k_i}R_i - M_i]$, then it can be verified that

$$code([L, R]) = code(M_i) + code([L_i', R_i'] \circ [l_{i+1}, r_{i+1}] \ldots \circ [l_n, r_n]) \tag{3.28}$$

Therefore, we can immediately output the first k_i bits of the code string. After that, we compute the product: $[L_i', R_i'] \circ [l_{i+1}, r_{i+1}] \ldots \circ [l_n, r_n]$, and we can recursively use the same early bit emitting scheme for this product. In this way, we can greatly reduce the likelihood of precision overflow.

3.4.2 Deterministic Approximation

For probability intervals containing 0.5, we cannot emit any bits early. In rare cases, such a probability interval would exceed the precision limit, and the correctness of our algorithm would be compromised.

To address this problem, we introduce the deterministic approximation trick. Recall that the correctness of arithmetic coding relies on the non-overlapping property of the probability intervals. Therefore, we do not need to compute probability intervals with perfect accuracy: the correctness is guaranteed as long as we ensure these probability intervals do not overlap

with each other.

Formally, let t_1, t_2 be two different tuples, and suppose their probability intervals are:

$$\begin{aligned} \text{PI}(t_1) &= [l_1, r_1] = [l_{11}, r_{11}] \circ [l_{12}, r_{12}] \circ \ldots \circ [l_{1n_1}, r_{1n_1}] \\ \text{PI}(t_2) &= [l_2, r_2] = [l_{21}, r_{21}] \circ [l_{22}, r_{22}] \circ \ldots \circ [l_{2n_2}, r_{2n_2}] \end{aligned} \tag{3.29}$$

The deterministic approximation trick is to replace $\circ$ operator with a deterministic operator $\diamond$ that approximates $\circ$ and has the following properties:

- For any two probability intervals $[a, b]$ and $[c, d]$:

$$[a, b] \diamond [c, d] \subseteq [a, b] \circ [c, d] \tag{3.30}$$

- For any two probability intervals $[a, b]$ and $[c, d]$ with $b - a \geq \epsilon$ and $d - c \geq \epsilon$. Let $[l, r] = [a, b] \diamond [c, d]$, then:

$$\exists k, M, 2^{-k}M \leq l < r \leq 2^{-k}(M + 1), 2^k(r - l) \geq \epsilon \tag{3.31}$$

In other words, the product computed by $\diamond$ operator is always a subset of the product computed by $\circ$ operator, and $\diamond$ operator always ensures that the product probability interval has length greater than or equal to ϵ after emitting bits. The first property guarantees the non-overlapping property still holds, and the second property prevents potential precision overflow. As we will see in Section 3.4.4, these two properties are sufficient to guarantee the correctness of arithmetic coding.

3.4.3 Delta Coding

Notice that our compression scheme thus far has focused on compressing "horizontally", i.e., reducing the size of each tuple, independent of each other. In addition to this, we could also compress "vertically", where we compress tuples relative to each other. For this, we directly leverage an algorithm developed in prior work. Raman and Swart [17] developed an optimal method for compressing a set of binary code strings. This coding scheme (called "Delta Coding" in their paper) achieves $O(n \log n)$ space saving where n is the number of code strings. For completeness, the pseudo-code of a variant of their method that is used in our system is listed in Algorithm 3.3.

In Algorithm 3.3, $Unary(s)$ is the unary code of s (i.e., $0 \to 0$, $1 \to 10$, $2 \to 110$, etc.). Delta Coding replaces the $\lfloor \log n \rfloor$-bit prefix of each tuple by an unary code with at most 2

Algorithm 3.3 Delta Coding

function DELTACODING($s_1, \ldots, s_n$)
  // $s_1, \ldots, s_n$ are binary codes of $t_1, \ldots, t_n$
  Sort $s_1, \ldots, s_n$
  $l \leftarrow \lfloor \log n \rfloor$
  If $len(s_i) < l$, pad s_i with trailing zeros
  Let $s_i = a_i b_i$ where a_i is l-bit prefix of s_i
  $s'_i \leftarrow Unary(a_i - a_{i-1}) + b_i$
  return $\{s'_1, \ldots, s'_n\}$
end function

bits on average. Thus it saves about $n(\log_2 n - 2)$ bits storage space in total.

3.4.4 Decompression

When decompressing, SQUISH first reads in the dataset schema and all of the model information, and stores them in the main memory. After that, it scans over the compressed dataset, extracts and decodes the binary code strings to recover the original tuples.

Algorithm 3.4 Decoding Algorithm

function DECODER.INITIALIZATION
  $I_b \leftarrow [0, 1]$
  $I_t \leftarrow [0, 1]$
end function
function DECODER.GETNEXTBRANCH($branches$)
  while not $\exists br \in branches, I_b \subseteq I_t \diamond \mathrm{PI}(br)$ **do**
    Read in the next bit x
    $I_b \leftarrow I_b \circ [\frac{x}{2}, \frac{x+1}{2}]$
  end while
  if $I_b \subseteq I_t \diamond \mathrm{PI}(br), br \in branches$ **then**
    $I_t \leftarrow I_t \diamond \mathrm{PI}(br)$
    while $\exists k = 0$ or $1, I_t \subseteq [\frac{k}{2}, \frac{k+1}{2}]$ **do**
      $I_t \leftarrow [2I_t.l - k, 2I_t.r - k]$
      $I_b \leftarrow [2I_b.r - k, 2I_b.r - k]$
    end while
    return br
  end if
end function

Algorithm 3.4 describes the procedure to decide the next branch. The decoder maintains two probability intervals I_b and I_t. I_b is the probability interval corresponding to all the bits that the algorithm has read in so far. I_t is the probability interval corresponding to all decoded attributes. At each step, the algorithm computes the product of I_t and

the probability interval for every possible attribute value, and then checks whether I_b is contained by one of those probability intervals. If so, we can decide the next branch, and update I_t accordingly. If not, we continue reading in the next bit and update I_b.

As an illustration of the behavior of Algorithm 3.4, Table 3.1 shows the step by step execution for the following example: $t = (a_1, a_2, a_3), [l, r] = [\frac{1}{3}, \frac{1}{2}] \circ [\frac{1}{4}, \frac{1}{2}] \circ [\frac{1}{2}, \frac{2}{3}], \text{code} = 01100110$.

Step	I_t	I_b	Input	Output
1	$[0, 1]$	$[0, 1]$		
2	$[0, 1]$	$[0, \frac{1}{2}]$	0	
3	$[0, 1]$	$[\frac{1}{4}, \frac{1}{2}]$	01	
4	$[\frac{1}{3}, \frac{1}{2}]$	$[\frac{1}{4}, \frac{1}{2}]$	01	a_1
5	$[\frac{2}{3}, 1]$	$[\frac{1}{2}, 1]$	01	a_1
6	$[\frac{1}{3}, 1]$	$[0, 1]$	01	a_1
7	$[\frac{1}{3}, 1]$	$[\frac{1}{2}, 1]$	011	a_1
8	$[\frac{1}{3}, 1]$	$[\frac{1}{2}, \frac{3}{4}]$	0110	a_1
9	$[\frac{1}{3}, 1]$	$[\frac{1}{2}, \frac{5}{8}]$	01100	a_1
10	$[\frac{1}{2}, \frac{2}{3}]$	$[\frac{1}{2}, \frac{5}{8}]$	01100	a_1, a_2
11	$[0, \frac{1}{3}]$	$[0, \frac{1}{4}]$	01100	a_1, a_2
12	$[0, \frac{2}{3}]$	$[0, \frac{1}{2}]$	01100	a_1, a_2
13	$[0, \frac{2}{3}]$	$[\frac{1}{4}, \frac{1}{2}]$	011001	a_1, a_2
14	$[0, \frac{2}{3}]$	$[\frac{3}{8}, \frac{1}{2}]$	0110011	a_1, a_2
15	$[0, \frac{2}{3}]$	$[\frac{3}{8}, \frac{7}{16}]$	01100110	a_1, a_2
16	$[\frac{1}{3}, \frac{4}{9}]$	$[\frac{3}{8}, \frac{7}{16}]$	01100110	a_1, a_2, a_3
17	$[\frac{2}{3}, \frac{8}{9}]$	$[\frac{3}{4}, \frac{7}{8}]$	01100110	a_1, a_2, a_3
18	$[\frac{1}{3}, \frac{7}{9}]$	$[\frac{1}{2}, \frac{3}{4}]$	01100110	a_1, a_2, a_3

Table 3.1: Decoding Example

Notice that Algorithm 3.4 mirrors Algorithm 3.2 in the way it computes probability interval products. This design is to ensure that the encoding and decoding algorithm always apply the same deterministic approximation that we described in Section 3.4.2. Theorem 3.2 in the following proves the correctness of the algorithm:

Theorem 3.2. *Let $[l_1, r_1], \ldots, [l_n, r_n]$ be probability intervals with $r_i - l_i \geq \epsilon$ where ϵ is the small constant defined in Section 3.4.2. Let s be the output of Algorithm 3.2 on these probability intervals. Then Algorithm 3.4 can always determine the correct branch from alternatives using s as input:*

$$PI(Decoder.GetNextBranch(branch_i)) = [l_i, r_i] \tag{3.32}$$

Proof. First we briefly examine Algorithm 3.2. Define a_i, b_i, c_i as follows:

$$a_i = b_{i-1} \diamond [l_i, r_i] \quad (b_0 = [0, 1]) \tag{3.33}$$

$$b_i = [2^{k_i} a_i.l - M_i, 2^{k_i} a_i.r - M_i] \tag{3.34}$$

$$c_i = c_{i-1} \circ [2^{-k_i} M_i, 2^{-k_i}(M_i + 1)] \quad (c_0 = [0, 1]) \tag{3.35}$$

where k_i is the largest integer such that $a_i \subseteq [2^{-k_i} M_i, 2^{-k_i}(M_i + 1)]$.'

Then, we have the following observations:

- a_i is the value of I_t after executing the first step of ith iteration of the loop, b_i is the value of I_t at the end of ith iteration of the loop, and c_i is the probability interval corresponding to code.
- s is the binary code of the probability interval $c_n \circ b_n$. If we define $\mathrm{PI}_s = [2^{-len(s)} s, 2^{-len(s)}(s+1)]$, then $\mathrm{PI}_s \subseteq c_n \circ b_n \subseteq c_i \circ b_i = c_{i-1} \circ a_i$

We can prove the following statements by induction on the steps of Algorithm 3.4 (see Table 3.1 for reference):

- During the procedure of determining ith branch, the value of I_t is:
  - b_{i-1}, during the first while loop.
  - a_i, after executing the first statement inside the if block.
  - b_i, at the end of the if block.
- During the procedure of determining ith branch, let d be the input that the algorithm has read in. Define $\mathrm{PI}_d = [2^{-len(d)} d, 2^{-len(d)}(d + 1)]$. Then the value of I_b satisfies:
  - $\mathrm{PI}_d = c_{i-1} \circ I_b$, during the first while loop.
  - $\mathrm{PI}_d = c_i \circ I_b$, at the end of the if block.
- $I_b \subseteq I_t$ always holds.

The induction step is easy for most parts, we will only prove the nontrivial part that the loop condition is satisfied before reaching the end of input. i.e., after certain steps we must have $I_b \subseteq I_t \diamond [l_i, r_i]$. To prove this, note that

$$\mathrm{PI}_s \subseteq \mathrm{PI}_d = c_{i-1} \circ I_b \qquad \mathrm{PI}_s \subseteq c_{i-1} \circ a_i \tag{3.36}$$

Therefore,

$$(c_{i-1} \circ I_b) \cap (c_{i-1} \circ a_i) \neq \emptyset \Leftrightarrow I_b \cap a_i \neq \emptyset \tag{3.37}$$

Note that $a_i = b_{i-1} \diamond [l_i, r_i] = I_t \diamond [l_i, r_i]$ during the while loop. Thus for any other branch $[l', r']$, $I_b \not\subseteq I_t \diamond [l', r']$ due to the fact that $[l', r'] \cap [l_i, r_i] = \emptyset$. Also note that as we continue reading new bits, $\text{PI}_d = c_{i-1} \circ I_b$ approaches PI_s, thus eventually we would have:

$$c_{i-1} \circ I_b \subseteq c_{i-1} \circ a_i \Leftrightarrow I_b \subseteq a_i \tag{3.38}$$

3.5 DISCUSSION: EXAMPLES, OPTIMALITY, AND STREAMING

In this section, we use three example types of datasets to illustrate how our compression algorithm can effectively find compact representations. These examples also demonstrates the wide applicability of the SQUISH approach. We then describe the overall optimality of our algorithm. Finally, we discuss how to extend SQUISH into the streaming scenario (i.e., tuples arrive continuously over time).

3.5.1 Illustrative Examples

We now describe three types of datasets in turn.

Pairwise Dependent Attributes

Consider a dataset with 100 binary attributes $a_1, \ldots, a_{100}$, where $a_1, \ldots, a_{50}$ are independent and uniformly distributed, and $a_{51}, \ldots, a_{100}$ are identical copies of $a_1, \ldots, a_{50}$ (i.e., $a_{i+50} = a_i$).

Let us consider a tuple $(x_1, x_2, \ldots, x_{100})$. The probability intervals for the first 50 attributes are $[\frac{x_i}{2}, \frac{x_i+1}{2}]$ (i.e., $[0, \frac{1}{2}]$ if $x_i = 0$ and $[\frac{1}{2}, 1]$ otherwise). For the last 50 attributes, since they deterministically depend on the first 50 attributes, the probability interval is always $[0, 1]$. Therefore the probability interval for the whole tuple is:

$$[\frac{x_1}{2}, \frac{x_1+1}{2}] \circ \ldots \circ [\frac{x_{50}}{2}, \frac{x_{50}+1}{2}] \circ [0,1] \circ \ldots \circ [0,1] \tag{3.39}$$

It is easy to verify that the binary code string of the tuple is exactly the 50-bits binary string $x_1 x_2 \ldots x_{50}$ (recall that each x_i is either 0 or 1).

We also need to store the model information, which consists of the probability distribution

of $x_1, \ldots, x_{50}$, the parent node of $x_{51}, \ldots, x_{100}$ and the conditional probability distribution of x_{i+50} conditioned on x_i.

Assuming all the model parameters are stored using an 8-bit code, then the model can be described using $200 \times 8 = 1600$ bits. Since each tuple uses 50 bits, in total SQUISH would use $1600 + 50n$ bits for the whole dataset, where n is the number of tuples. Note that our compression algorithm achieves much better compression rate than Huffman Coding [18], which uses at least 100 bits per tuple.

Dependent attributes exist in many datasets. While they are usually only softly dependent (i.e., one attribute influences but does not completely determine the other attribute), our algorithm can still exploit these dependencies in compression.

Markov Chain

Figure 3.6 shows an example dataset with 1000 categorical attributes, in which each attribute a_i depends on the preceding attribute a_{i-1}, and a_1 is uniformly distributed. This kind of dependency is called a *Markov Chain*, and frequently occurs in time-series datasets.

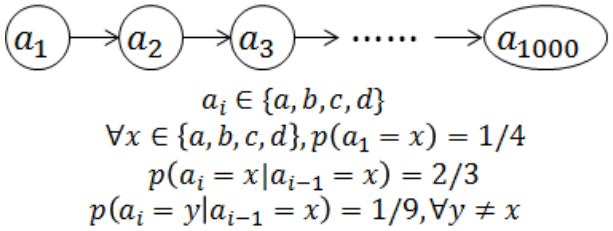


Figure 3.6: Markov Chain Example

The probability interval of tuple $t = (x_1, \ldots, x_{1000})$ is:

$$[\frac{x_1}{4}, \frac{x_1 + 1}{4}] \circ g(x_1, x_2) \circ g(x_2, x_3) \circ \ldots \circ g(x_{999}, x_{1000}) \tag{3.40}$$

where mapping $g(x, y)$ is listed in Table 3.2.

	y=1	y=2	y=3	y=4
x=1	$[0, 2/3]$	$[2/3, 7/9]$	$[7/9, 8/9]$	$[8/9, 1]$
x=2	$[0, 1/9]$	$[1/9, 7/9]$	$[7/9, 8/9]$	$[8/9, 1]$
x=3	$[0, 1/9]$	$[1/9, 2/9]$	$[2/9, 8/9]$	$[8/9, 1]$
x=4	$[0, 1/9]$	$[1/9, 2/9]$	$[2/9, 1/3]$	$[1/3, 1]$

Table 3.2: Mapping Probability Intervals of Markov Chain Example

On average, for each tuple our algorithm uses about

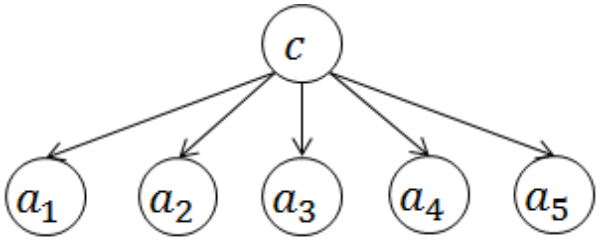
$$1000 \times (\frac{2}{3}\log_2 \frac{3}{2} + 3 \times \frac{1}{9}\log_2 9) \approx 1443 \text{ bits} \tag{3.41}$$

while standard Huffman Coding [18] uses 2000 bits.

Time series datasets usually contain a lot of redundancy that can be used to achieve significant compression. As an example, most electrocardiography (ECG) waveforms [19] can be restored using a little extra information if we know the cardiac cycle. Our algorithm offers effective ways to utilize such redundancies for compression.

Clustered Tuples

Figure 3.7 shows an example with 100 binary attributes: $a_1, \ldots, a_{100}$. In this example, c is the hidden cluster index attribute and all other attributes are dependent on it.



c	$p(c)$
1	0.5
2	0.5

c	a_i	$p(a_i \mid c)$
1	a	0.2
1	b	0.8
2	a	0.8
2	b	0.2

Figure 3.7: Clustered Tuples Example

If we compress the cluster index together with all attributes using our algorithm, we will need about

$$H(t_i) = 1 + 100 \times (0.2\log_2 \frac{1}{0.2} + 0.8\log_2 \frac{1}{0.8}) \approx 73 \text{ bits} \tag{3.42}$$

for each tuple. Note that it is less than the 100-bits used by the plain binary code.

Many real datasets have a clustering property. To compute the cluster index, we need to choose an existing clustering algorithm that is most appropriate to the dataset at hand [2]. The extra cost of storing a cluster index is usually small compared to the overall saving in compression.

3.5.2 Asymptotic Optimality

We can prove that SQUISH achieves asymptotic near-optimal compression rate for lossless compression if the dataset only contains categorical attributes and can be described efficiently using a Bayesian network:

Theorem 3.3. *Let $a_1, a_2, \ldots, a_m$ be categorical attributes with joint probability distribution $P(a_1, \ldots, a_m)$ that decomposes as*

$$P(a_1, \ldots, a_m) = \prod_{i=1}^{m} P(a_i | parent_i) \tag{3.43}$$

such that

$$parent_i \subseteq \{a_1, \ldots, a_{i-1}\}, card(parent_i) \leq c \tag{3.44}$$

Suppose the dataset $\mathcal{D}$ contains n tuples that are i.i.d. samples from P. Let $M = \max_i card(a_i)$ be the maximum cardinality of attribute range. Then SQUISH *can compress $\mathcal{D}$ using less than $H(\mathcal{D}) + 4n + 32mM^{c+1}$ bits on average, where $H(\mathcal{D})$ is the entropy [20] of the dataset $\mathcal{D}$.*

Proof. Since our compression algorithm searches for the Bayesian Network with minimum description length, it suffices to prove that the size of compressed dataset using the correct Bayesian Network is less than $H(\mathcal{D}) + 4n + 32mM^{c+1}$.

Let S be the set of tuples in $\mathcal{D}$ such that its probability is less than 2^{-n}:

$$S = \{i \in [n] : P(t_i) < 2^{-n}\} \tag{3.45}$$

Then, the size of the compressed dataset can be expressed as:

$$\begin{aligned} \text{compressed size} \leq & \sum_{i \in S} (-\log_2 P(t_i) - (\log_2 n - 2) + 2) + 2(n - |S|) + (\text{BN description cost}) \\ \leq & \sum_{i \in S} (-\log_2 P(t_i)) - |S| \log_2 |S| + 4n + 32mM^{c+1} \end{aligned} \tag{3.46}$$

where we assume that the model parameters are stored using single precision float numbers.

On the other hand, since S deterministically depends on $\mathcal{D}$, therefore by the chain rule of entropy we have:

$$\begin{aligned} H(\mathcal{D}) = & H(\mathcal{D}) + H(S|\mathcal{D}) = H(\mathcal{D}, S) \\ = & H(S) + H(t_S|S) + H(t_{[n]\setminus S}|S) \geq H(t_S|S) \end{aligned} \tag{3.47}$$

and since t_S is a multi-set consisting of $\{t_i : i \in S\}$, we have

$$H(t_S|S) \geq -|S|E[\log_2 P(t)|P(t) < 2^{-n}] - |S| \log_2 |S| \tag{3.48}$$

Combining these two directions, we conclude that

$$\text{compressed size} \leq H(\mathcal{D}) + 4n + 32mM^{c+1} \tag{3.49}$$

Thus, when n is large, the difference between the size of the compressed dataset using our system and the entropy[1] of $\mathcal{D}$ is at most $5n$, that is only 5 bits per tuple. This indicates that SQUISH is asymptotically near-optimal for this setting.

When the dataset $\mathcal{D}$ contains numerical attributes, the entropy $H(\mathcal{D})$ is not defined, and the techniques we used to prove Theorem 3.3 no longer apply. However, in light of Theorem 3.1, it is likely that SQUISH still achieves asymptotic near-optimal compression.

3.5.3 Extension to a Streaming Scenario

So far, we have assumed that the entire tabular dataset is given at the beginning, it is also possible to extend SQUISH to the data stream scenario wherein tuples arrive continuously over time. In the following we briefly discuss a potential extension to a streaming case.

The simplest scenario is that all arriving tuples are i.i.d. samples from the same joint-distribution. In this case, it is sufficient to learn a Bayesian Network structure using the earliest arriving tuples, and then apply it to encode all later tuples. Since every tuple comes from the same underlying distribution, the Bayesian Network learned from a subset of tuples will apply equally well to other tuples.

However, the more common and complex scenario is when the underlying distribution would also slowly drift away over time. In this case, the Bayesian Network structure learned from the earliest arriving tuples will not naturally apply to the ones that arrive later, and it becomes necessary for us to adapt the existing structure incrementally to handle the distribution drifting phenomenon. The problem of learning Bayesian Network incrementally have been studied in a few recent papers [21, 22], but it is not clear whether these methods can be readily plugged into SQUISH, or we need to develop more efficient methods for our scenario. We will leave the investigation of this issue for future work.

[1] By Shannon's source coding theorem [20], there is no algorithm that can achieve compression rate higher than entropy asymptotically.

3.6 EXPERIMENTS

In this section, we evaluate the performance of SQUISH against the state-of-the-art tabular dataset compression algorithms SPARTAN [9] and ItCompress [10]. For reference we also include the performance of gzip [7], a well-known generic compression algorithm. We use the following four publicly available datasets:

- *Corel* (http://kdd.ics.uci.edu/databases/CorelFeatures) is a 20 MB dataset containing 68,040 tuples with 32 numerical color histogram attributes.
- *Forest-Cover* (http://kdd.ics.uci.edu/databases/covertype) is a 75 MB dataset containing 581,000 tuples with 10 numerical and 44 categorical attributes.
- *Census* (http://thedataweb.rm.census.gov/ftp/cps_ftp.html) is a 610MB dataset containing 676,000 tuples with 36 numerical and 332 categorical attributes.
- *Genomes* (ftp://ftp.1000genomes.ebi.ac.uk) is a 18.2GB dataset containing 1,832,506 tuples with about 10 numerical and 2500 categorical attributes.[2]

The first three datasets have been used in prior papers [9, 10], and the compression ratio achieved by SPARTAN, ItCompress and gzip on these datasets have been reported in Jagadish et al.'s work [10]. We did not reproduce these numbers and only used their reported performance numbers for comparison. For the *Census* dataset, the previous papers only used a subset of the attributes in the experiments (7 categorical attributes and 7 numerical attributes), and we are unaware of the exact selection criteria. Therefore, we will only report the comparison with gzip on this dataset.

For the *Corel* and *Forest-Cover* datasets, we set the error tolerance as a percentage (1% by default) of the width of the range for numerical attributes as in previous works. For the *Census* dataset, the compression is lossless. For the *Genomes* dataset, we set the error tolerance for integer attributes to 0 and float-point numerical attributes to 10^{-8}.

3.6.1 Compression Rate Comparison

Figure 3.8 shows the comparison of compression rate on the *Corel* and *Forest-Cover* datasets. In these figures, X axis is the error tolerance for numerical attributes (% of the

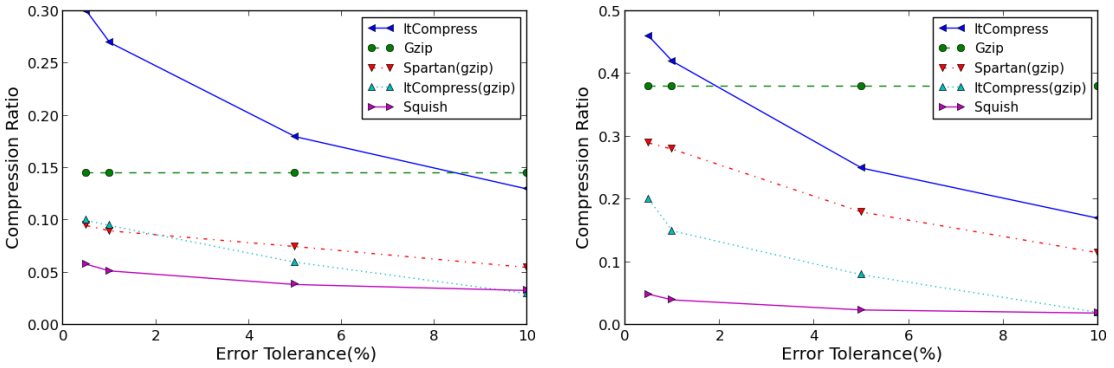
[2] In this dataset, many attributes are optional and these numbers indicate the average number of attributes that appear in each tuple.

width of range), and Y axis is the compression ratio, defined as follows:

$$\text{compression ratio} = \frac{\text{data size with compression}}{\text{data size without compression}} \tag{3.50}$$

As we can see from the figures, SQUISH significantly outperforms the other algorithms. When the error tolerance threshold is small (0.5%), SQUISH achieves about *50% reduction in compression ratio on the Forest Cover dataset and 75% reduction on the Corel dataset, compared to the nearest competitor ItCompress (gzip)*, which applies gzip algorithm on top of the result of ItCompress. The benefit of not using gzip as a post-processing step is that we can still permit tuple-level access without decompressing a larger unit.

The remarkable improvements that our system achieved in the Corel dataset reflects the superiority of SQUISH for compressing numerical attributes, which is known to be a hard problem [23] and none of the previous systems have effectively addressed it. In contrast, our encoding scheme can leverage the skewness of the distribution and achieve near-optimal performance.



(a) Forest Cover (b) Corel

Figure 3.8: Error Threshold vs Compression Ratio

Figure 3.9 shows the comparison of compression rate on the *Census* and *Genomes* datasets. Note that in these two datasets, we set the error tolerance threshold to be extremely small, so that the compression is essentially lossless. As we can see, even in the lossless compression scenario, our algorithm still outperforms gzip significantly. *Compared to gzip,* SQUISH *achieves* 48% *reduction in compression ratio in Census dataset and* 56% *reduction in Genomes dataset.*

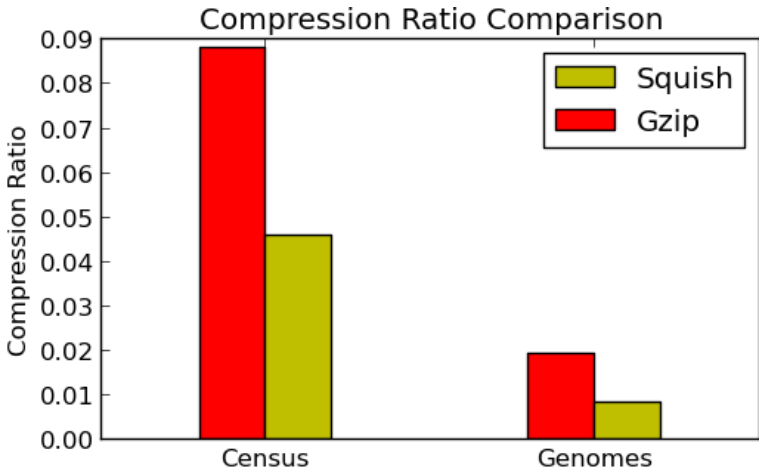


Figure 3.9: Compression Ratio Comparison

3.6.2 Compression Breakdown

As we have seen in the last section, SQUISH achieved superior compression ratio in all four datasets. In this section, we use detailed case studies to illustrate the reason behind the significant improvement over previous papers.

Categorical Attributes

Here we study the source of the compression in SQUISH for categorical attributes. We will use three different treatments for the categorical attributes and see how much compression is achieved for each of these treatments:

- Domain Code: We replace the categorical attribute values with short binary code strings. Each code string has length $\lceil \log_2 N \rceil$, where N is the total number of possible categorical values for the attribute.
- Column: We ignore the correlations between categorical attributes and treat all the categorical attributes as independent.
- Full: We use both the correlations between attributes and the skewness of attribute values in our compression algorithm.

We will use the *Genomes* and *Census* dataset here since they consist of mostly categorical attributes. We keep the compression algorithm for numerical attributes unchanged in all treatments. Figure 3.10 shows the compression ratio of the three treatments:

As we can see, the compression ratio of the basic domain coding scheme can be improved up to 70% if we take into account the skewness of the distribution in attribute values.

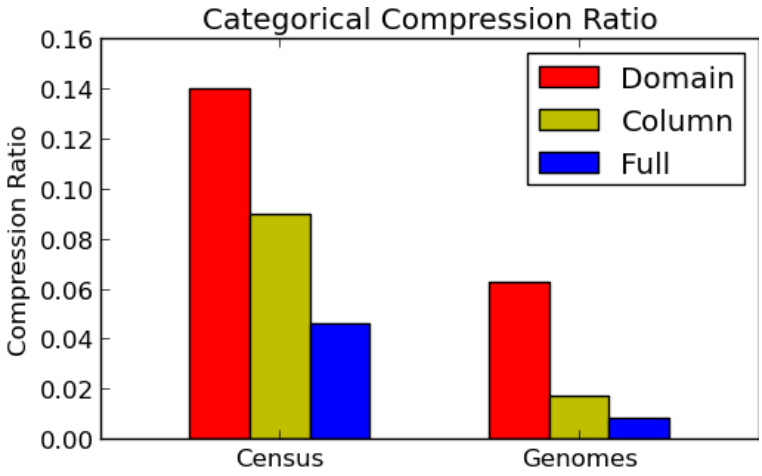


Figure 3.10: Compression Ratio Comparison for Categorical Attributes

Furthermore, the correlation between attributes is another opportunity for compression, which improved the compression ratio by 50% in both datasets.

An interesting observation is that the Column treatment achieves comparable compression ratio as gzip in both datasets, which suggests that gzip is in general capable of capturing the skewness of distribution for categorical attributes, but unable to capture the correlation between attributes.

Numerical Attributes

Here we study the source of the compression in SQUISH for numerical attributes. We use the following five treatments for the numerical attributes:

- IEEE Float: We use the IEEE Single Precision Floating Point standard to store all attributes.
- Discrete: Since all attributes in the dataset have value between 0 and 1, we use integer i to represent a float number in range $[\frac{i}{10^7}, \frac{i+1}{10^7}]$, and then store each integer using its 24-bit binary representation.
- Column: We ignore the correlation between numerical attributes and treat all attributes as independent.
- Full: We use both the correlations between attributes and distribution information about attribute values.
- Lossy: The same as the Full treatment, but we set the error tolerance at 10^{-4} instead.

We use the *Corel* dataset here since it contains only numerical attributes. The error tolerance in all treatments except the last are set to be 10^{-7} to make sure the comparison is fair (IEEE single format has precision about 10^{-7}). All the numerical attributes in this dataset are in range $[0, 1]$, with a distribution peaked at 0. Figure 3.11 shows the compression ratio of the five treatments.

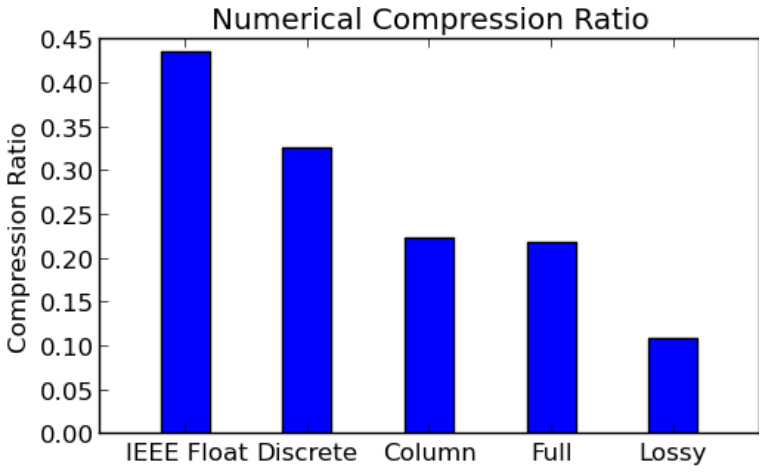


Figure 3.11: Compression Ratio Comparison for Numerical Attributes

As we can see, storing numerical attributes as float numbers instead of strings gives us about 55% compression. However, the compression rate can be improved by another 50% if we recognize distributional properties (i.e., range and skewness). Utilizing the correlation between attributes in the *Corel* dataset only slightly improved the compression ratio by 3%. Finally, we see that the benefit of lossy compression is significant: even though we only reduced the precision from 10^{-7} to 10^{-4}, the compression ratio has already been improved by 50%.

3.6.3 Running Time

In this section we evaluate the running time of Squish. Note that the time complexity of the compression and decompression components are both $O(nm)$, where m is the number of attributes and n is the number of tuples. In other words, the running time of these two components are linear to the size of the dataset. Therefore, the algorithm should scale well to large datasets in theory.

Table 3.3 lists the running time of the five components in Squish. All experiments are performed on a computer with eight[3] 3.4GHz Intel Xeon processors. For the *Genomes*

[3]The implementation is single-threaded, so only one processor is used.

dataset, which contains 2500 attributes—an extremely large number—we constructed the Bayesian Network manually. Note that none of the prior works have been applied on a dataset with the magnitude of the *Genomes* dataset (both in number of tuples and number of attributes).

	Forest Cov.	Corel	Census	Genomes
Struct. Learning	5.5 sec	2.5 sec	20 min	N/A
Param. Tuning	140 sec	15 sec	100 min	40 min
Compression	48 sec	6 sec	6 min	50 min
Writing to File	7 sec	2 sec	40 sec	7 min
Decompression	53 sec	7.5 sec	6 min	50 min

Table 3.3: Running Time of Different Components

As we can see from Table 3.3, our compression algorithm scales reasonably: even with the largest dataset *Genomes*, the compression can still be finished within hours. Recall that since our algorithm is designed for archival not online query processing, and *our goal is therefore to minimize storage as much as possible*, a few hours for large datasets is adequate.

Random Access: Unlike gzip [7], SQUISH allows random access of tuples without decompressing the whole dataset. Therefore, if users only need to access a few tuples in the dataset, then they will only need to decode those tuples, which would require far less time than decoding the whole dataset.

Running Time Remark: Due to a suboptimal implementation of the parameter tuning component in our current code, the actual time complexity of the parameter tuning component is $O(nmd)$ where d is the depth of the Bayesian network. Therefore, Table 3.3 may not reflect the best possible running time of a fully optimized version of our compression algorithm. Also, the running time of the parameter tuning component can be greatly reduced if we utilize sampling technique (as we did for structure learning). The only potential bottleneck is structure learning, which scales badly with respect to the number of attributes ($O(m^4)$). To handle datasets of this scale, another approach is to partition the dataset column-wise, and apply the compression algorithm on each partition separately. We plan to investigate this in future work.

3.6.4 Sensitivity to Bayesian Network Learning

We now investigate the sensitivity of the performance of our algorithm with respect to the Bayesian network learning. We use the *Census* dataset here since the correlation between

attributes in this dataset is stronger than other datasets, so the quality of the Bayesian network can be directly reflected in the compression ratio.

Since our structure learning algorithm only uses a subset of the training data, one might question whether the selection of tuples in the structure learning component would affect the compression ratio. To test this, we run the algorithm for five times, and randomly choose the tuples participating in the structure learning. Table 3.4 shows the compression ratio of the five runs. As we can see, the variation between runs are insignificant, suggesting that our compression algorithm is robust.

No. of Exp.	1	2	3	4	5
Comp. Ratio	0.0460	0.0472	0.0471	0.0468	0.0476

Table 3.4: Sensitivity of the Structure Learning

We also study the sensitivity of our algorithm with respect to the number of tuples used for structure learning. Table 3.5 shows the compression ratio when we use 1000, 2000 and 5000 tuples in the structure learning algorithm respectively. As we can see, the compression ratio improves gradually as we use more tuples for structure learning.

Number of Tuples	1000	2000	5000
Comp. Ratio	0.0474	0.0460	0.0427

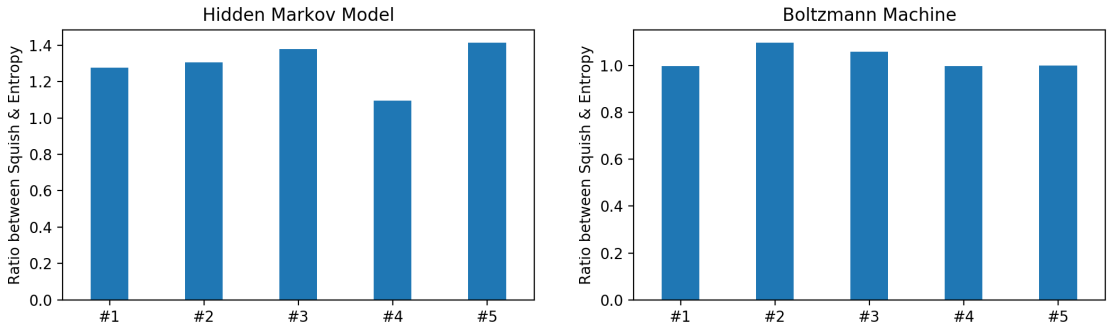
Table 3.5: Sensitivity to Number of Tuples

3.6.5 Loss of Compressibility from Bayesian Network

Although Bayesian Network structure is a very commonly used model for capturing the joint-distribution of multiple random variables, there are situations in which the Bayesian Network structures cannot perfectly model the exact underlying distribution. In this section, we investigate how much compressibility we could potentially lose from using a Bayesian Network structure instead of full joint-distribution. Here, we synthetically generate several datasets and compare the compression ratio of Squish with the dataset entropy (which, by Shannon's coding theorem [20], is the theoretical lower bound[4] of the compression ratio).

We generated two sets of synthetic datasets using Hidden Markov Model (HMM) [24] (4 states, 32 possible observations, 50 columns, 1,000,000 tuples) and Boltzmann Machine [25] (4 hidden units, 10 observed units/columns, 1,000,000 tuples) respectively. Figure 3.12 shows

[4]We remark that such a lower bound only holds asymptotically, and with a finite dataset size it is actually possible to achieve slightly better compression compared to entropy.



(a) Hidden Markov Model (b) Boltzmann Machine

Figure 3.12: Compression Ratio: SQUISH vs. Theoretical Lower Bound

the ratio between SQUISH's compression result and the entropy of these datasets. As we can see, even though SQUISH cannot achieve the optimal compression ratio exactly, it is not too far either: the compression ratio of SQUISH is only 29% and 3% worse on average compared to the theoretical lower bound for both sets of datasets respectively, which demonstrates that even in the cases where the true distribution cannot be perfectly captured by Bayesian Networks, the compression result of SQUISH is still nearly optimal.

3.7 BIBLIOGRAPHICAL NOTES

Although compression of datasets is a classical research topic in the database research community [23], the idea of exploiting attribute correlations (a.k.a. semantic compression) is relatively new. Babu et al. [9] used functional dependencies among attributes to avoid storing them explicitly. Jagadish et al. [10] used a clustering algorithm for tuples. Their compression scheme stores, for each cluster of tuples, a representative tuple and the differences between the representative tuple and other tuples in the cluster. These two types of dependencies are special cases of the more general Bayesian network style dependencies used in SQUISH.

The idea of applying arithmetic coding on Bayesian networks was first proposed by Davies and Moore [26]. However, their work only supports categorical attributes (a simple case). Further, the authors did not justify their approach by either theoretically or experimentally comparing their algorithm with other semantic compression algorithms. Lastly, they used conventional BIC [14] score for learning a Bayesian Network, which is suboptimal, and their technique does not apply to the lossy setting.

The compression algorithm developed by Raman and Swart [17] used Huffman Coding to compress attributes. Therefore, their work can only be applied to categorical attributes and can not fully utilize attribute correlation (the authors only mentioned that they can exploit

attribute correlations by encoding multiple attributes at once). The major contribution of Raman's work [17] is that they formalized the old idea of compressing ordered sequences by storing the difference between adjacent elements, which has been used in search engines to compress inverted indexes [27] and also in column-oriented database systems [11].

Bayesian networks are well-known general-purpose probabilistic models to characterize dependencies between random variables. For reference, the textbook written by Koller and Friedman [12] covers many recent developments. Arithmetic coding was first introduced by Rissanen [28] and further developed by Witten et al. [13]. An introductory paper written by Langdon Jr. [16] covers most of the basic concepts of Arithmetic Coding, including the early bit emission trick. The deterministic approximation trick is original. Compared to the overflow prevention mechanism in Witten et al.'s work [13], the deterministic approximation trick is simpler and easier to implement.

3.8 CONCLUSION

In this chapter, we presented Squish, an hidden structure based compression algorithm for tabular datasets. Squish follows the general framework we summarized in Chapter 2: the Bayesian Network learning module corresponds to the extraction step, and the arithmetic coding module corresponds to the data processing step. The algorithm design primarily focuses on the details of the structure hypothesis space: (a) we used Bayesian Networks as the basic structure form, which is much more representative than structures used in prior papers; (b) the use of a greedy approximation algorithm and sampling technique allows us to efficiently extract structures from the input dataset; (c) the SquID mechanism allows us to handle different types of attributes consistently, making it easy for the extracted structure to be utilized in the compression step. Due to the careful design of structure hypothesis space, Squish is able to surpass prior compression algorithms [9, 10] that also use hidden structures, achieving significantly better results.

CHAPTER 4: EXTRACTING STRUCTURE FOR ORGANIZATION

Log datasets, as generated by computer programs, contain a large volume of useful information including the behavior of users, the activities within systems and the usage of resources over time. While this information can potentially be very valuable from an analytic standpoint, to summarize, cluster, or identify anomalies, they are unfortunately buried in arbitrarily formatted log files, preventing them from being readily used. Most log datasets are streams of log entries continuously generated by print statements scattered across the computer program. These log entries are often implicitly structured (i.e., they follow a predefined but unknown data format). Once we uncover their structures, we can easily import the information within into a relational format. We can then infer relationships across datasets, and put them to use to aid analysis, search, or browsing.

Even though techniques for extracting structures from text-formatted datasets have been studied in many other contexts (e.g., from HTML files [29, 30], HTML lists [31], Network Protocols [32]), most existing techniques rely on the specific characteristics of the target dataset (e.g., using the DOM tree structure for extraction from HTML files), which makes them not readily applicable to log datasets. On the other hand, program synthesis techniques [33] are general-purpose methods for devising transformation rules from provided examples and are applicable in our context, but requires extensive user supervision. Fisher et al. [34] take one step towards automation by only requiring users to provide record boundaries: they assume that the data is already *chunked* (i.e., partitioned into small blocks such that each block contains exactly one record) beforehand using external tools. This chunking step is assumed to be a simple form of supervision (e.g., when each record contains exactly k lines), and their work primarily focus on learning structure given these blocks as input. Recordbreaker [35] is a simple automated implementation of Fisher et al.'s technique that assumes that each record occupies exactly one line. Indeed, this is far too drastic an assumption to retain applicability for log datasets, as we will see below.

Example 4.1 (Importance of Recognizing Multi-line Records)**.** *Consider the example log dataset in Figure 4.1, where each record occupies multiple lines. One promising approach to extract from such a dataset is to use an unsupervised extraction scheme such as RecordBreaker [35] that extracts contents from each line independently, resulting in tables T_1, T_2, T_3 as displayed, such that a multi-line record can be viewed to be a union of multiple single-line records. While such a single-line extraction approach indeed can potentially "extract" all relevant content, the association between records are completely lost in the generated tables. The loss of record association information makes it very hard for users to interpret or utilize*

such results.

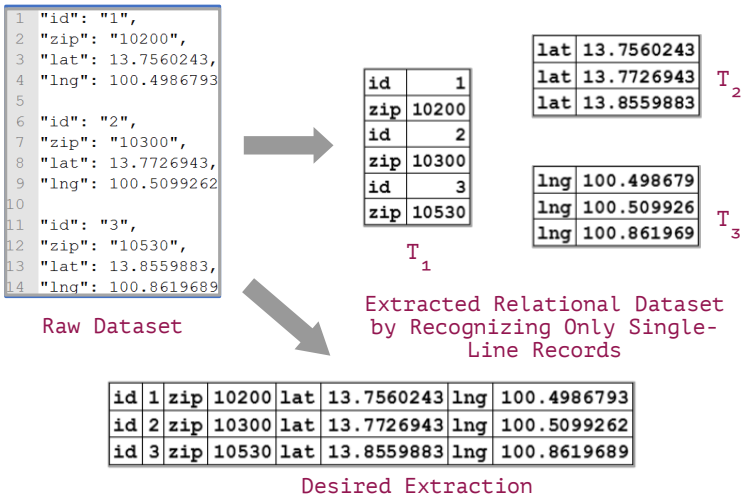


Figure 4.1: Sample multi-line record dataset, along with the single-line extraction results

Example 4.2 (Importance of Recognizing Multiple Record Types)**.** *Consider the example log dataset in Figure 4.2, in which there are two types of records (A and B) consisting of* 7 *and* 9 *lines respectively, randomly interspersed with each other. Since the sequence of record types can be arbitrary, it is no longer possible to identify the boundaries of records using simple rules, rendering prior unsupervised log structure extraction algorithms non-applicable*[1]*.*

In this chapter, we present DATAMARAN [5], an automatic log dataset structure extraction algorithm. At a high-level, DATAMARAN follows the general framework in Chapter 2: we first identify the optimal structure of the dataset, then use it to extract the data contents within the log dataset and then import them into structured format. However, due to the extreme efficiency requirement of this task, the design of both the structure hypothesis space and the extraction method are highly nontrivial, as we shall see in the rest of this chapter: in Section 4.1, we present the design of structure hypothesis space in DATAMARAN and discuss the intuition behind this design; Section 4.2 – 4.6 are dedicated to the details of the DATAMARAN algorithm for finding the optimal structure candidate within this hypothesis space; in Section 4.7, we present a theoretical analysis of DATAMARAN; in Section 4.8, some experimental results are shown to demonstrate the utility of DATAMARAN.

[1]Note, although that in this example, the boundaries of records are represented as the special "——" lines, an unsupervised chunker cannot utilize this information without human guidance

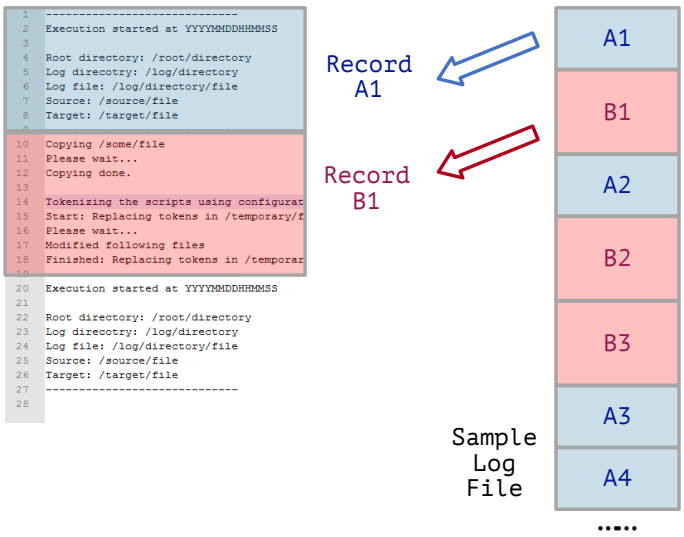


Figure 4.2: Sample log dataset from GitHub with contents anonymized

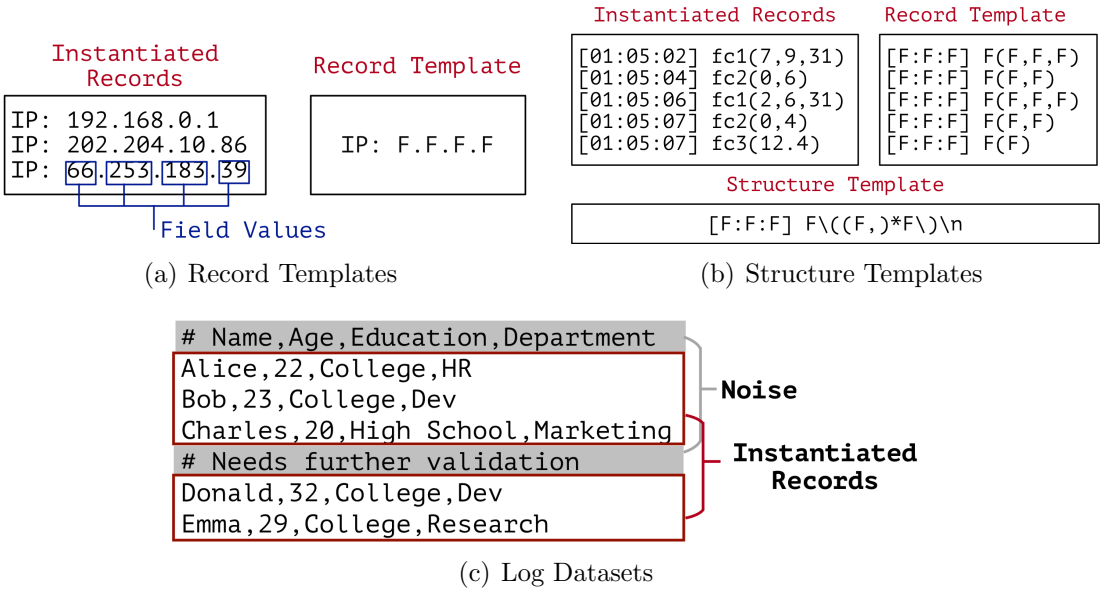
4.1 STRUCTURE HYPOTHESIS SPACE

Establishing the structure hypothesis space for log dataset formats is tricky since the concept of a log dataset is itself ill-specified and does not indicate clear characteristics or structures. In the following, we first need to identify the defining characteristics of common log datasets encountered in practice. The concept of a log dataset is defined via the following series of definitions:

Definition 4.1 (Record Template/Instantiated Record)**.** *A* record template *is a string that contains one or more instances of the field placeholder character—a special type of character, denoted as 'F'—along with other characters. An* instantiated record *generated from a record template is a string constructed by replacing field placeholder characters in the record template with strings containing no field placeholder characters.*

Definition 4.2 (Structure Template)**.** *A* structure template *is a regular expression [36] for record templates. We say the record template RT* can be generated *from the structure template ST iff the regular expression of ST matches the string form of RT.*

Definition 4.3 (Log Dataset)**.** *A log dataset $\mathcal{D} = \{T, S\}$ consists of two components: the textual component T and the structural component S. $S = \{ST_1, ST_2, \ldots, ST_k\}$ is a collection of structure templates, and T is a text dataset with the following structure: T can be partitioned into several blocks separated by the end-of-line character '\n', and each block is either an instantiated record generated from one of the structure templates in S, or corresponds to a noise string with no structure.*



(a) Record Templates (b) Structure Templates

(c) Log Datasets

Figure 4.3: Log Dataset Illustration

These definitions are illustrated in Figure 4.3. Intuitively, a structure template captures minor variations in the structure of records within a dataset via a regular expression: as shown in Figure 4.3(b), the example structure template captured minor differences in the record templates such as one, two, or three arguments within parentheses.

Based on these definitions, in order to complete the structure hypothesis space design, we need to restrict the set of valid structure templates that we want to consider (since the set of all regular expressions is obviously too large to allow efficient extraction). Generally speaking, the structure hypothesis space should have good representativeness (i.e., flexible enough to be applicable for most log datasets). In addition, the complexity level of structure templates should offer a good trade-off between utility and learnability (i.e., more complicated structures offer finer-grained extraction but are more costly to extract). In DATAMARAN, the structure templates are restricted to have the following tree-style form:

Assumption 4.1 (Tree-Style Structure Candidates)**.** *We only consider structure templates with one of the following forms:*

1. *Array:* `({regexA}x)*{regexA}y`
   where `{regexA}` *is another regular expression satisfying Assumption 4.1, and* `x` *and* `y` *are different characters.*
2. *Struct:* `{regexA}{regexB}{regexC}....`
   where `{regexA}{regexB}{regexC}....` *is a sequence of regular expressions, and each*

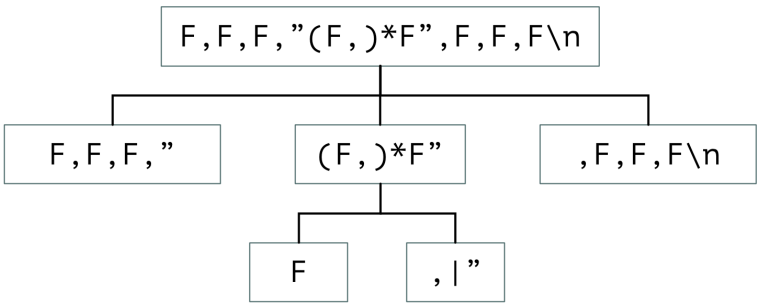


Figure 4.4: Structural Form Assumption

of them is either a simple string or another regular expression satisfying Assumption 4.1.

Assumption 4.1 states that records in log datasets are laid out from left to right, with nesting. Formally, the Array-type regular expression is intended to characterize lists of objects. For example, the structure template `[F,F,F,...,F]` can be represented by a prefix character `'['` and an array-type regular expression `"(F,)*F]"`. Based on Assumption 4.1, each structure template essentially follows a special tree-style form. Figure 4.4 illustrates the associated tree form of an example structure template `F,F,F,"(F,)*F",F,F,F\n`. As we can see, the root node in this tree is a Struct node, while the node in the middle (2nd node in level 2) is an Array node with two children nodes: the left one is the repetitive regular expression, and the right one denotes the separating/terminating characters.

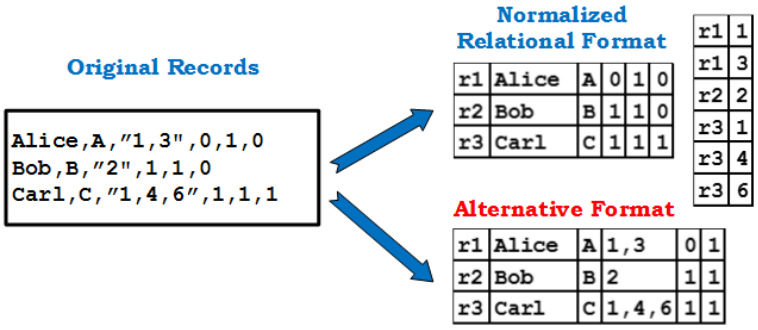


Figure 4.5: Extracted Relational Dataset

We can store all extracted records in a relational format based on Assumption 4.1, which is demonstrated in Figure 4.5: the instantiated records (left hand side) are generated from the structure template in Figure 4.4, and the right hand side depicts two representations for the relational dataset. DATAMARAN can generate either representation, both of which contain all of the extracted information, and can be utilized by downstream applications.

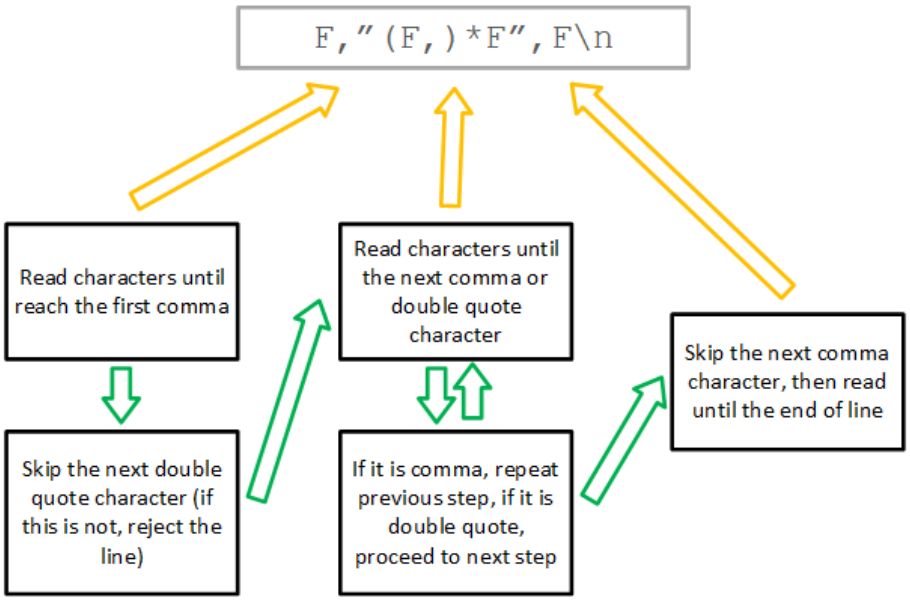


Figure 4.6: Parser for Example Structure Template

Remark: Assumption 4.1 is adapted from the record structure assumption in [34]. The two major differences are: (a) we removed the union-type node in Fisher's original design to allow more efficient extraction; (b) the array-type nodes are enhanced with separating and terminating characters to facilitate efficient parsing: the tree-style structure form allows one to implement a LL(1) parser, which only requires a single linear scan over the whole dataset. Figure 4.6 demonstrates this parsing procedure for a simple example structure template.

4.2 OVERVIEW OF THE DATAMARAN ALGORITHM

Most prior unsupervised structure extraction algorithms [34, 35, 31] assume that the record boundaries are known beforehand. These algorithms are usually based on the idea of summarization, in which they take all the examples generated from the structure template as input, and then try to find the structure template by seeking out the common patterns among records. However, in many real-world log datasets, the record boundaries are usually unknown, which makes these algorithms not directly applicable. Furthermore, the task of finding record boundaries itself is also not easy: without knowing the record characteristics first, it is very difficult to pinpoint the exact location of record boundaries, especially with the presence of heavy noise.

Given the difficulty associated with identifying record boundaries, a different approach is used by Datamaran: Datamaran first generates a large collection of structure template candidates directly from the dataset (without actually identifying the record boundaries),

and then evaluates the most promising ones to find the optimal structure template. Figure 4.7 illustrates the conceptual differences between DATAMARAN and prior approaches such as RecordBreaker [35]. Concretely, DATAMARAN algorithm consists of the following three steps, as illustrated in Figure 4.8:

- *Generation.* The first step is to generate a large collection of candidate structure templates directly from the log dataset. To achieve this, we first extract a large collection of structure templates from potential records (i.e., consecutive lines in the dataset), then insert these structure templates into a hash-table to find repeated ones.
- *Pruning.* The second step is to prune out most of the candidates found in the previous step, such that we only need to evaluate a small number of candidates to find the optimal one. To achieve this, we designed an *assimilation score function*, which is used to filter out all of the redundant structure templates derived by removing some structural details from the true structure templates. We then retain the candidates with highest assimilation score for the final evaluation.
- *Evaluation.* During the final step, we evaluate the remaining candidates using a MDL [37] based *regularity score function*[2] to find the optimal one.

The primary algorithmic contributions of DATAMARAN are the implementations of *generation* and *pruning* step: (a) for the *generation* step, extracting structure templates directly from potential records is highly nontrivial due to the possible variations of field values and record template structures (see Assumption 4.1); (b) for the *pruning* step, the *assimilation score function* requires careful design: it has to be simple enough so that we can evaluate it efficiently, while being effective enough to be able to prune out most of the low-quality redundant candidates. Our final design is based on several iterations, and is not straightforward at first glance.

Notation. Table 4.1 lists the notations used in DATAMARAN. The first 3 symbols are parameters in DATAMARAN, while the last 5 symbols represent dataset-dependent values. We will describe each of these parameters later on.

[2]The exact design of the regularity score is not the primary focus of DATAMARAN. In fact, we assume that the regularity score function is given, and we can access it through a function call. In this sense, the primary contribution of DATAMARAN is an efficient and scalable method to optimize any reasonable scoring function.

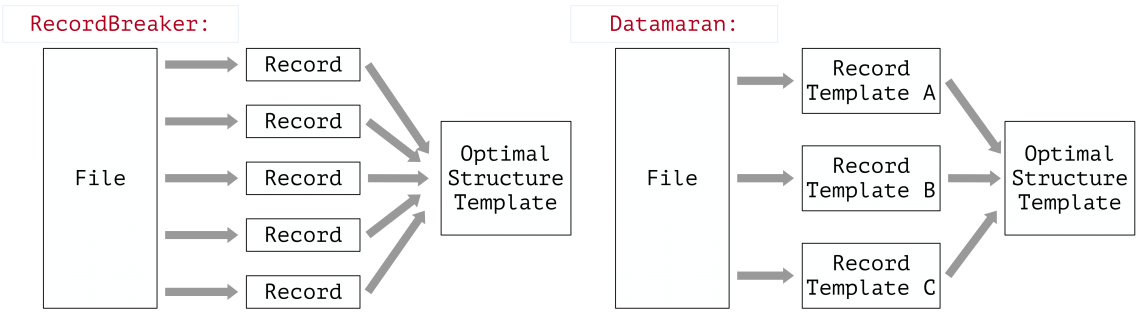


Figure 4.7: The difference between DATAMARAN and earlier work

Symbol	Description
M	The number of structure templates retained after pruning
L	The maximum span of records (i.e., the maximum number of lines each record can span)
α	The minimum coverage threshold for records
n	The total number of lines in the dataset
K	The number of structure templates retained after generation
T_{data}	The total size of the dataset
S_{data}	The amount of data sampled during all three steps
c	The number of special characters (i.e., characters in *RT-CharSet-Candidate*) appearing in the dataset

Table 4.1: Notation Summary

4.3 THE GENERATION STEP

In the generation step, we generate a large collection of structure template candidates directly from the input log dataset. More concretely, we will generate all structure templates that cover a sufficient percentage of the original log dataset. The correctness of the generation step relies on the following coverage assumption:

Assumption 4.2 (Coverage Threshold)**.** *The coverage of every structure template $ST_i \in S$ should be at least α%. The coverage of structure template ST is defined as the total length (i.e., total number of characters) of the instantiated records of ST.*

In order to find all structure templates with at least α% coverage, DATAMARAN makes an additional assumption regarding the structure templates:

Assumption 4.3 (Non-Overlapping)**.** *For any structural template ST and any character c, one of the following is true:*

- *for any record template RT generated from ST, $c \notin RT$.*

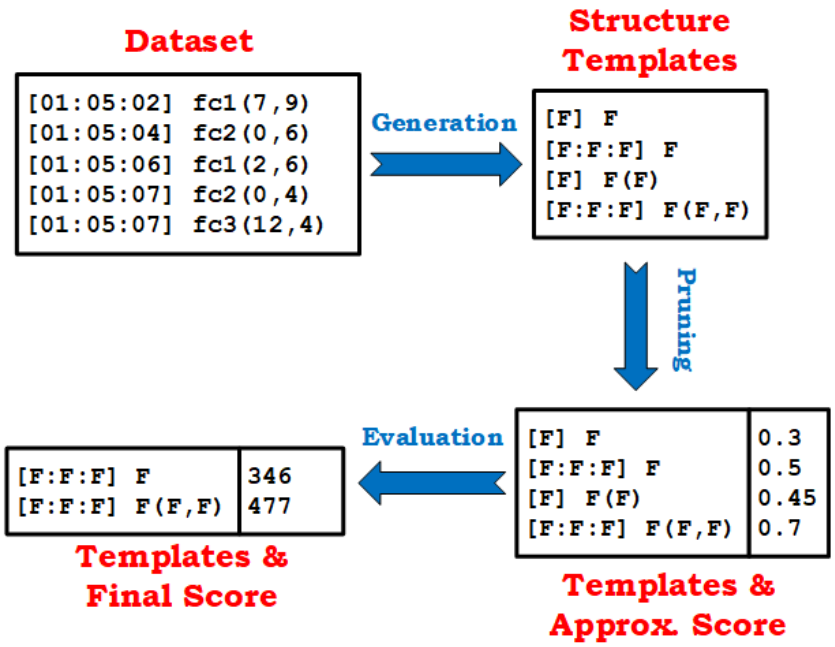


Figure 4.8: The Workflow of DATAMARAN

- *for any instantiated record R generated from ST, no field values of R contains c.*

Alternatively, let RT-CharSet *denote the set of characters in record templates, while* F-CharSet *denotes the set of characters in field values. Then for any structure template ST, there exists two disjoint character sets $A(ST)$ and $B(ST)$, such that for any instantiated record R of ST, we have* RT-CharSet$(R) \subseteq A(ST)$ *and* F-CharSet$(R) \subseteq B(ST)$.

With Assumption 4.3, the following five step approach can be used to generate all structure templates with at least $\alpha\%$ coverage:

1. Enumerate possible values of *RT-CharSet* (i.e., the character set in the record templates), and for each such value of *RT-CharSet*, run through steps 2-5.
2. Enumerate all $O(nL)$ pairs of end-of-line characters `'\n'` that are close to each other (i.e., at most L lines are between them) in the textual component T. For each such pair, treat the content between each pair as an instantiated record, and run steps 3-4.
3. Extract the record template from the instantiated record using the value of *RT-CharSet*.
4. Reduce the record template into a structure template (with the form defined in Assumption 4.1).

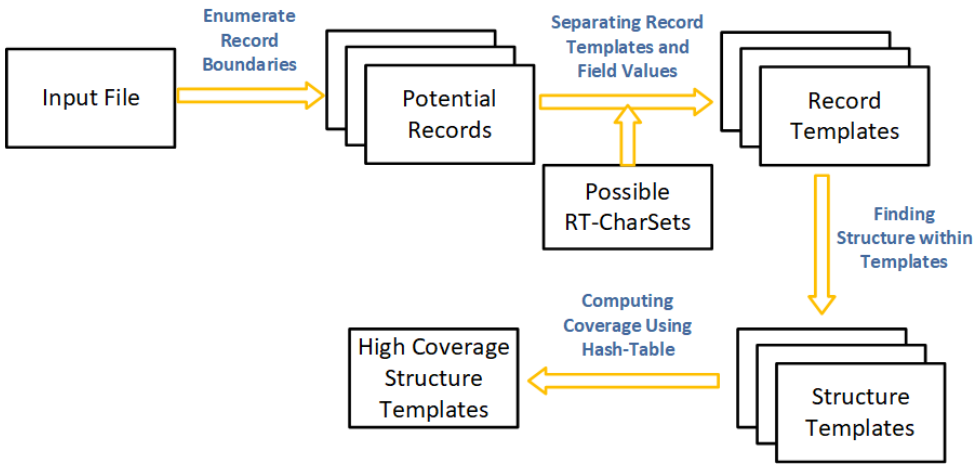


Figure 4.9: The Generation Step Workflow

5. Store all of the structure templates generated in step 4 within a hash-table, and then find the ones that satisfy the coverage threshold assumption.

Figure 4.9 illustrates the workflow of the generation step. The basic idea behind the generation step is very simple: we first enumerate all possible record boundaries (Step 2), then extract structure templates from the contents between them (Step 3, 4), and finally use a hash-table to find the ones with sufficient coverage (Step 5). Assumption 4.3, which states that $\textit{RT-CharSet} \cap \textit{F-CharSet} = \emptyset$, allows us to extract record templates directly from instantiated records (Step 3). Using this assumption, we can separate the field values from formatting characters after enumerating possible values of *RT-CharSet* (Step 1).

Finding the optimal RT-CharSet. We implemented two searching procedures in DATAMARAN for finding the optimal *RT-CharSet*. Both searching procedures require *RT-CharSet-Candidate*, the set of characters that can potentially be in *RT-CharSet*, as an input.

Suppose there are c different characters in *RT-CharSet-Candidate* that appeared in the dataset. The exhaustive search would enumerate all 2^c subsets. On the other hand, the greedy search procedure would only enumerate $O(c^2)$ of them. The greedy search procedure operates in the following way: initially, *RT-CharSet* is set to be empty; then in each step, one of the characters in *RT-CharSet-Candidate* is added to *RT-CharSet*; the decision for choosing which character to add is made greedily by choosing the character generating the structure template with highest approximation score.

The following example helps illustrate the two searching procedures. Consider a dataset with the following structure template: `[F:F:F] F(F,F)`. There are 7 special characters in total: `[]:(),`(space character). Thus, the exhaustive search would enumerate 128 possible

subsets for this example. As for the greedy search, it starts from the empty set and gradually adds new characters into it:

- in the first step, it enumerates all the subsets containing only one character, and computes the corresponding structure templates (i.e., invoking steps 2-5).
- it then decides which subset to proceed based on which one has the structure template with the highest approximation score (for this example, it is "`F:F:F`").
- then in the second step, it enumerates all 6 subsets consisting of the character ':' and one additional character.
- this procedure repeats until either the subset is full or we can no longer find any structure template with at least $\alpha\%$ coverage.

It is easy to see that, for this example, the maximum number of subsets that the greedy search would have enumerated is 29 (also counting the empty subset here). On the other hand, the exhaustive search would have enumerated 128 subsets. Note that if the field values do not contain any special characters in *RT-CharSet-Candidate*, then the correct *RT-CharSet* would contain all characters in *RT-CharSet-Candidate* that appeared in the dataset. In this case, the greedy search procedure is guaranteed to find the correct *RT-CharSet* since it will always consider the full subset at the end of the searching procedure.

Extracting Record Template From Instantiated Record. The non-overlapping assumption (Assumption 4.3) states that there exists two disjoint sets of characters A and B, such that for any instantiated record R, *RT-CharSet*(R) (i.e., the record template character set) is a subset of A, and *F-CharSet*(R) (i.e., the field value character set) is a subset of B. By this assumption, the record template can be uniquely extracted from any of its instantiated records given the value of A and B.

For example, if $A = \{$ `','`, `'\n'` $\}$, then the instantiated record `1,2,3,45,6,78,9,a,bc\n` can be transformed into the record template `F,F,F,F,F,F,F,F,F\n` by replacing characters not in A with the field placeholder.

Reducing Record Templates to Structure Templates. We identify the corresponding minimum structure template that can generate each extracted record template. This is achieved by repeatedly reducing repeated patterns into array regular expressions. For example, the record template `F,F,F,F,F,F,F\n` is reduced into the structure template `(F,)*F\n`. If there are conflicting reduction steps (i.e., reduction steps that cannot be performed simultaneously), we choose one of them arbitrarily. The reduction process only guarantees to

find a minimal structure template (i.e., structure template that cannot be reduced further), which means that not all instantiated records are reduced back to the same structure template. As a result, the coverage estimate during the generation step is an underestimate. However, in our experiments, the initial coverage estimate is usually still well above the $\alpha\%$ threshold, thereby not affecting the correctness of the generation step.

Sampling Technique. In the actual implementation of DATAMARAN, sampling is used instead of simply scanning through the entire dataset in both the generation and evaluation step. For large datasets, scanning the whole dataset during these steps is not feasible: the total number of whole dataset scans is equal to the number of *RT-CharSet*s enumerated in the generation step plus M in the evaluation step. Our sampling implementation is cache-aware: we sample several large chunks of data and concatenate them in the memory. Both the generation/evaluation steps are performed on the concatenated chunks instead.

Pseudocode. The pseudocode of the generation step can be found in Algorithm 4.1. The two searching procedures correspond to function *GreedySearch* and *ExhaustiveSearch* respectively. The function *GenST* finds structure templates with at least $\alpha\%$ coverage given the value of *RT-CharSet*.

4.4 THE PRUNING STEP

Even with the coverage threshold assumption, there are often far too many structure template candidates remaining after the generation step. As a result, it is impossible to evaluate the MDL regularity score for every single one. In the pruning step, we identify a small promising subset of these candidates to be evaluated in the final evaluation step, and discard the rest. To achieve this, we use *assimilation score function* to order the structure templates, so that only the top ones need to be evaluated explicitly in the evaluation step. The assimilation score estimates the amount of data "assimilated" by the structure template (i.e., the amount of data that can be explained by the structure template). Therefore, structure templates with a higher assimilation score are more likely to also have a higher regularity score.

Before we describe the actual design of our assimilation score function, it is helpful to first understand why there are so many structure templates remaining after the generation step. It turns out that most of the redundant structure templates come from two sources as demonstrated in Figure 4.10: (a) when the structure template consists of multiple lines (line 1-5 in Figure 4.10 left), any subset of such a structure template would also be captured by the generation step as a legitimate structure template (line 2-4 in Figure 4.10 right); (b)

when the structure template uses multiple types of characters to separate the field values, simpler structure templates can be recognized if some of those characters are treated as field values as illustrated in Figure 4.10 (bottom).

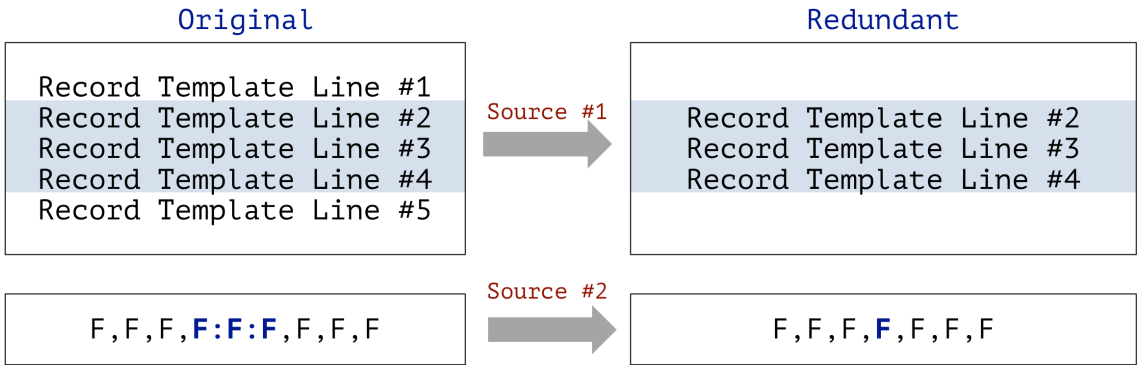


Figure 4.10: Two sources of redundancies: (1) subsets of multi-line structure templates; (2) structural parts recognized as field values.

Therefore, a good assimilation score should be able to distinguish both types of redundancies, and rank the true structure template(s) higher than the redundant ones. At the same time, it should be relatively lightweight to compute. To achieve this, our first component uses the coverage value of structure templates, which has already been computed during the generation step.

However, while the coverage value can effectively distinguish the first source of redundancy, it is not capable of distinguishing the second one. To address this shortcoming, we introduce another component into the assimilation score: the *Non-Field-Coverage* term, which is defined as the total coverage of the structure template minus the total coverage of all field values of the structure template (i.e., the total length covered by field values in the instantiated records). This term computes the total coverage achieved by "non-field" characters in the template, and can effectively distinguish the second source of redundancy. The final assimilation score function used in DATAMARAN is the following, which filters out all structure templates with either low coverage or low non-field-coverage:

$$Assimilation(S) = Cov(S) \times Non\_Field\_Cov(S) \tag{4.1}$$

4.5 STRUCTURE REFINEMENT

In order to further improve the extraction accuracy of DATAMARAN, we developed two techniques to refine the structure templates. These techniques are applied to all of the top M structure templates during the evaluation step: we revise these structure templates, and compare the revised structure templates against the original ones, using the regularity score function, replacing them if the score is improved.

Array Unfolding

During the generation step, all of the records are transformed into minimal structure templates, which allowed us to detect repetitive patterns within the dataset. However, there are cases where the minimum structure template is not the optimal structure template.

For instance, in comma-separated values files (*.csv files), all of the records have the form `"F,F,F,....,F,F\n"` (i.e., a fixed number of field values separated by commas). There are two possible structure templates for these records: the plain struct-type `"F,F,F,....,F,F\n"` and the array-type `"(F,)*F\n"`. The plain struct-type template offers a better semantic interpretation in this case (since it implies that the field values are of different types), and also leads to a better regularity score.

More generally, because of the structure template reduction procedure (step 4 in the generation step), when the optimal structure template is not a minimal structure template, only its reduced form will be found during the generation step. To address this, we designed the *array unfolding* technique: for each array-type regular expression in the structure template, we attempt to unfold it by expanding it into a struct-type. Figure 4.11 demonstrates this process: the array-type regular expression at the top of the figure will be unfolded into one of the struct-type regular expression at the bottom of the figure. If any of these unfolded structure templates has a better score than the original one, the unfolding would be finalized.

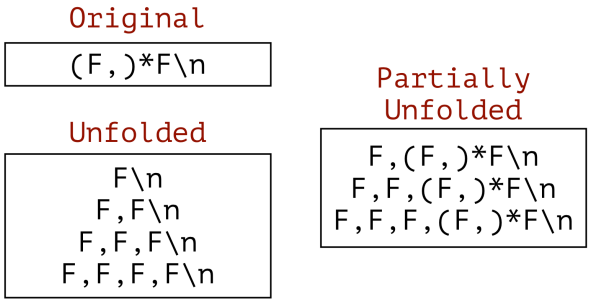


Figure 4.11: Array Unfolding

Partial unfolding, another unfolding mechanism implemented in DATAMARAN, is also

demonstrated in Figure 4.11. Here, we expand the array-type regular expression while retaining the non-deterministic array-type suffix. Partial unfolding is used to handle the cases where regular field values are "mixed in" with text field values, as in the following example:

```
Apr 24 04:02:24 srv7 snort shutdown succeeded
Apr 24 04:02:24 srv7 snort startup succeeded
Apr 24 14:44:28 srv7 Disabling nightly yum
```

In this example, the first four fields are regular fields, but the last one is a free-text field. The ideal structure template for this example is `F F F F (F )*F\n`, which can be obtained by applying partial unfolding to the minimum structure template `(F )*F\n`.

Structure Shifting

Typically, the regularity score function evaluates the quality of structure templates using statistics such as coverage value or minimum description length. For most cases, these kinds of score functions can distinguish good structure templates from bad ones. However, there is one ambiguity among structure templates that such a regularity score would fail to detect: the cyclic shifting of structure templates. Figure 4.12 illustrates this: the regularity score of the shifted structure template (right hand side in Figure 4.12) and the score of the correct structure template (left hand side in Figure 4.12) are usually approximately equal to each other.

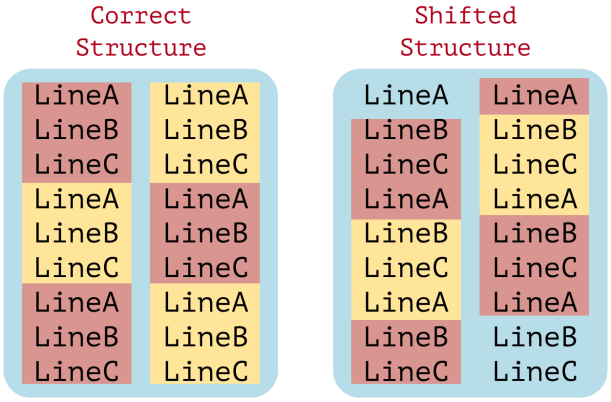


Figure 4.12: Structure Shifting

The structure shifting mechanism in DATAMARAN is designed to distinguish such ambiguities: for each structure template, we consider all possible shifted variants, and then find the position of first occurrence for each one of them. We then pick the one with the earliest first occurrence, which intuitively is most likely the correct structure.

4.6 REGULARITY SCORE FUNCTION

The design of DATAMARAN is independent of the choice of the regularity scoring function: we can plug in any reasonable scoring function into DATAMARAN, and the algorithm would function as before. However, for completeness, we will present the details of the regularity score function that we use in our implementation in this section. The regularity score function we implemented is based on the minimum description length principle [37]: we design a record generation procedure from the structure template, and the regularity score is equal to the total amount of information needed for describing all the instantiated records using the structure template, plus the additional information needed to describe the noise blocks. Describing the record using the structure template is straightforward given the structural form assumption (Assumption 4.1):

- For arrays, we first describe the number of blocks, then each block individually.
- For structs, we describe each component individually.
- For fields, the description scheme depends on its value type.

For the field value description, we associate each field in the structure template with one of the following four value-types: enumerated type, integer, real number, or string. The description schemes for field values depend on the data-type—which can be determined by analyzing the field values in the group; the details of these schemes are listed as follows:

- The enumerated type fields are described using $\lceil \log_2 n\_value \rceil$ bits, where n_value is the total number of unique values in this field.
- The integer fields are described using $\lceil \log_2(max\_value - min\_value + 1) \rceil$ bits, where max_value and min_value are the upper bound and lower bound of the field value, which can be determined by scanning through the dataset.
- The real number fields are described using $\lceil \log_2[(max\_value - min\_value) \times 10^{exp} + 1] \rceil$ bits, where max_value and min_value are the same as above, and exp is the maximum number of digits after the decimal point.
- The string fields are described directly using $(len(s) + 1) \times 8$ bits, where $len(s)$ is the length of the field value. The $+1$ term is to include the end-of-string `'\0'` character, and each character needs 8 bits to describe.

Using the description schemes above, the total description length can be computed as $D(dataset) = len(ST) \times 8 + 32 + m + \sum_{i=1}^{m} D(block_i)$. The first $len(ST) \times 8$ bits describe the the structure template, and the next $32 + m$ bits describe the total number of blocks in the dataset and whether each block is a noise block or a record. $D(block_i)$ is the description length of ith block: for noise blocks, it is simply the block length times 8; for record blocks, we compute its description length accordingly.

The pseudocode for computing the description score can be found in Algorithm 4.2, with the following 3 steps:

1. extract all the instantiated records from the dataset.
2. estimate the data-type parameters from the extracted records.
3. compute the description length using the formulae above.

4.7 THEORETICAL ANALYSIS

Time Complexity

Table 4.2 lists the time complexity of the three steps in DATAMARAN respectively[3]. An explanation for the symbols can be found in Table 4.1. Note that for large datasets, we would utilize sampling for both the generation and evaluation step (details in Section 4.3), and therefore S_{data} is upper-bounded by a large constant. In such cases, the running time of our algorithm is dominated by the actual data extraction procedure.

Step	Time Complexity
Generation Step	$O(S_{data}L2^c)$ or $O(S_{data}Lc^2)$
Pruning Step	$O(K \log K)$
Evaluation Step	$O(MS_{data})$
Data Extraction	$O(T_{data})$

Table 4.2: Time Complexity of the Three Steps in DATAMARAN

Correctness Guarantee

DATAMARAN is designed to tolerate noise blocks and variations within record structures and field values. Here we characterize three conditions that are sufficient for guaranteeing the correctness of DATAMARAN:

[3]There are two variants of the search procedure for enumerating RT-CharSet in the generation step as described in Section 4.3.

Theorem 4.1. *For a log dataset $D = \{T, S\}$ with only T observed, if the following conditions are all met:*

(a) One of the structure templates in S (denote it as ST_0) has the highest coverage and non-field-coverage (defined in Section 4.4) among all structure templates.

(b) For at least $\alpha\%$ of the instantiated records, the minimum structure template is ST_0.

(c) ST_0 has the best regularity score among all structure templates.

Then DATAMARAN *is guaranteed to return ST_0 as the optimal structure template.*

Proof. First of all, condition (b) ensures that ST_0 can be found during the generation step. Then, using condition (a), we can ensure that ST_0 to be the top structure template during the pruning step. Finally, condition (c) ensures that ST_0 will be chosen during the evaluation step. Combining all arguments, we can see that DATAMARAN is guaranteed to return ST_0 as the optimal structure template.

For most practical settings, condition (b) is automatically met. Condition (c) requires a carefully designed score function, which is not the focus of DATAMARAN. As for condition (a), intuitively it requires the structure templates in S to be sufficiently different from each other, and the field values and noise blocks are sufficiently random. If all of these conditions are satisfied, then Theorem 4.1 would guarantee the correctness of DATAMARAN.

4.8 EXPERIMENTS

In this section, we present experimental results that evaluate the performance of DATAMARAN. Three sets of experiments are conducted serving different purposes:

- **Manually Collected Log Datasets (Section 4.8.1).** We collected 25 datasets, including the entire set of 15 datasets used by Fisher et al. [34] and 10 from other sources. These datasets cover a wide variety of structural formats and possess different characteristics (e.g., file size or structural complexity). We use these datasets to demonstrate the effectiveness and stability of DATAMARAN across the board.

- **GitHub Log Datasets (Section 4.8.2).** We crawled a collection of 100 log datasets automatically from public GitHub repositories. These datasets reflect the properties of real-world data lakes. We use these datasets to study the properties of data lakes "in the wild", as well as the utility of DATAMARAN in such settings.

- **User Study (Section 4.8.3).** we conducted a user study on five representative log datasets, wherein the participants were asked to transform them into the desired target structure, starting from (a) our results, (b) the extraction results of RecordBreaker [35], and (c) the raw datasets as three different tasks. Their experience on each task is reported to reflect the utility of DATAMARAN in practice.

DATAMARAN **Settings:** DATAMARAN is implemented in C++ and compiled under Visual Studio 2015. The default values for the three parameters in DATAMARAN are: $\alpha = 10\%$ (the coverage threshold parameter); $L = 10$ (the upper bound of record span); $M = 50$ (the number of remaining structure templates after the pruning step). These default values are used in all of our experiments except for our parameter sensitivity experiments.

RecordBreaker [35] Settings: Despite our best attempts, we were unable to install or run the open-source version of RecordBreaker [35]. Therefore, we decided to faithfully reimplement RecordBreaker in C++ for our comparison. At the first step, RecordBreaker relies on a lexer to break up each record into tokens. We use the open source software Flex [38] as the lexer in our implementation. Accordingly, users need to write a Flex specification file tailored to their dataset in order to obtain a better tokenization scheme. We will compare against RecordBreaker in Section 4.8.2 and Section 4.8.3.

Experiment Settings: All experiments were conducted on a 64-bit Windows machine with 8-core Intel Xeon 3.40GHz CPU and 8GB RAM. All executions are single-threaded.

4.8.1 Manually Collected Datasets

Fisher et al. [34] used 15 manually collected datasets to demonstrate the effectiveness of their structure extraction method, and we followed this tradition by applying DATAMARAN on the same set of datasets. Furthermore, since Fisher's collection lacks large or complex datasets (i.e., datasets with multiple types of records or multi-line records), we also collected 10 additional datasets from the internet (e.g., the stack exchange data dump [39]) as well as from our genomics collaborators.

Table 4.3 lists the sources and characteristics of the 25 manually collected datasets[4]. The first 15 datasets are from Fisher et al.'s paper [34] (marked with "*" in Table 4.3).

Some example extraction results of DATAMARAN are demonstrated in Figure 4.13, along with the results of RecordBreaker [35]. From the experiments, we see that DATAMARAN can successfully extract structures from all manually collected datasets. The extraction results are also finer grained (compared to RecordBreaker), which is generally better for practical

[4]For crash log datasets, there are two valid structures with max record span 1 and 3 respectively

Data source	File size(MB)	# of rec. types	Max rec. span
*transaction records	0.07	1	1
*comma-sep records	0.02	1	1
*web server log	0.29	1	1
*log file of Mac ASL	0.28	1	1
*Mac OS boot log	0.02	1	1
*crash log	0.05	1	1(3)
*crash log (modified in [34])	0.05	1	1(3)
*ls -l output	0.01	1	1
*netstat output	0.01	2	1
*printer logs	0.02	1	1
*personal income records	0.01	1	1
*US railroad info	0.01	1	1
*application log	0.06	1	1
*LoginWindow server log	0.05	1	1
*pkg install log	0.02	1	1
Thailand district info	0.19	1	8
stackexchange xml data	20	1	1
vcf genetic format	167.4	1	1
fastq genetic format	29.9	1	4
blog xml data	0.06	1	10
log file (1)	0.03	2	9
log file (2)	0.01	1	3
log file (3)	0.19	2	1
log file (4)	0.07	2	10
log file (5)	0.09	1	4

Table 4.3: Sources and characteristics of manually collected datasets.

purposes. These experiments demonstrated the effectiveness of DATAMARAN on a wide range of datasets with different properties.

Running Time

Here we study the efficiency of DATAMARAN. We first run DATAMARAN on the 25 datasets using the default parameters to study the connection between the characteristics of datasets (size/structural complexity) and the running time of DATAMARAN. Then, we vary the parameters to study their impact on the efficiency of DATAMARAN.

Running Time vs. Dataset Size: Figure 4.14(a) depicts the impact of the size of the dataset on the running time of DATAMARAN (using either exhaustive search or greedy search). The running time on small datasets (less than 50MB) is dominated by the generation and evaluation step. For these datasets, the average running time is 17 seconds for greedy search and 37 seconds for exhaustive search. It takes about 7 minutes for DATAMA-

(a) Raw File(A)

(b) Raw File(B)

(c) Datamaran(A)

(d) Datamaran(B)

(e) RecordBreaker(A)

(f) RB(B1)

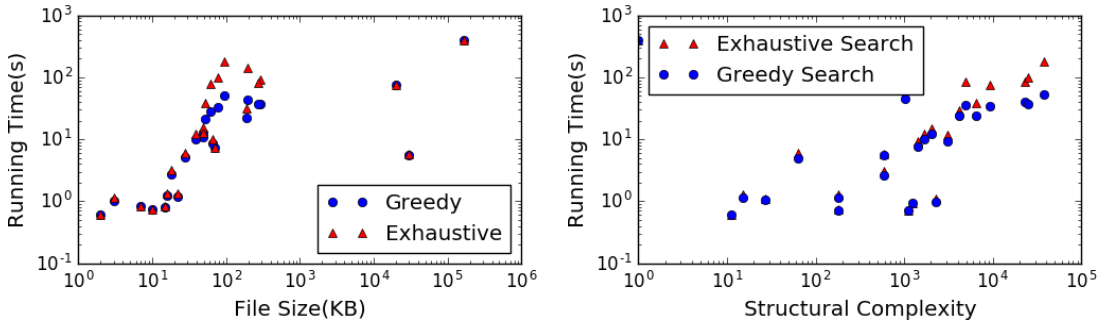
(g) RB(B2)

(h) RB(B3)

Figure 4.13: Example Extraction Results of DATAMARAN and RecordBreaker

RAN to process the largest dataset here (with size 167MB), where the majority of the running time is spent on running the LL(1) parser [40] for the actual data extraction. Note that the running time of the three major steps of DATAMARAN is not affected by dataset size for large datasets (as discussed in Section 4.7). As we can see in Figure 4.14(a), the extraction time is already dominated by the running time of LL(1) parser [40] (which is a necessary step for all structure extraction algorithms) even when the dataset is only moderately large (i.e., about 167MB). Further, this step is easily parallelizable. Therefore, we conclude that DATAMARAN is efficient enough in practice.

Running Time vs. Structural Complexity: Figure 4.14(b) depicts the impact of the structural complexity of the dataset on the running time of DATAMARAN. The structural complexity of datasets are characterized using the total number of structure templates with at least 10% coverage. In general, it takes a longer time for DATAMARAN to extract datasets



(a) Running Time vs. Dataset Size (b) Running Time vs. Structural Complexity

Figure 4.14: Running Time vs. (a) Dataset Size and (b) Structural Complexity. x axis in (b) is the number of structure templates with at least 10% coverage.

with higher structural complexity, and the efficiency benefits of greedy search is more significant on these datasets.

Running Time vs. Parameters: Figure 4.15 shows the impact of parameters on the running time of DATAMARAN (exhaustive search). Recall that M is the number of remaining structure templates after pruning step. As we can see in the left figure, the value of M directly affects the overall running time, and this effect is more significant for larger datasets. In the right figure, we can see that changing parameters α or L also affect the efficiency of DATAMARAN.

Note that if we evaluate all structure templates with at least α% coverage (i.e., skipping the pruning step by setting $M = \infty$), the average running time would be longer than 6 minutes even for small datasets. Therefore, it is necessary to use assimilation score to prune out structure templates.

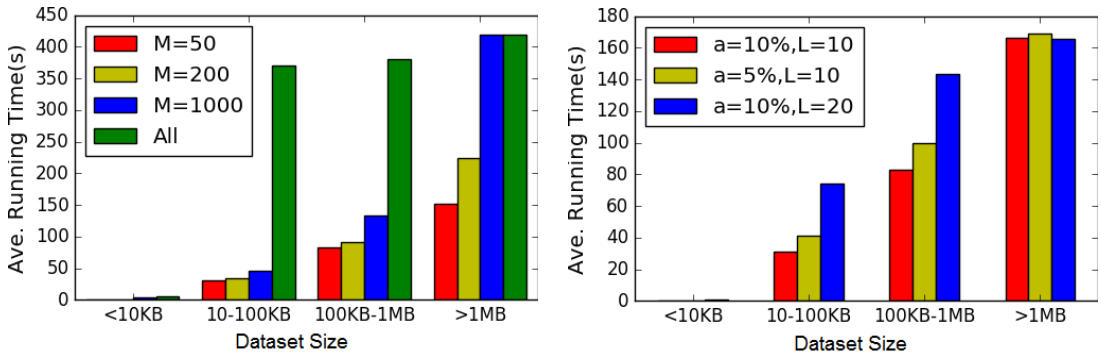


Figure 4.15: The impact of parameters on the running time.

Parameter Sensitivity

Here we evaluate the impact of parameters by checking whether DATAMARAN can find

the optimal structure template (i.e., the structure template with best regularity score, this is found by evaluating the regularity score of every structure template with at least $\alpha\%$ coverage). Figure 4.16 shows the percentage of datasets in which DATAMARAN can find the optimal structure template on different parameter combinations. As we can see, DATAMARAN is very robust with respect to the parameter settings: for example, changing the value of parameter M from 50 to 1000 only increased the likelihood of finding the optimal structure by about 10%. Figure 4.16 also verifies the effectiveness of the assimilation score in practice: for 40% of the datasets, the optimal structure also has the best assimilation score.

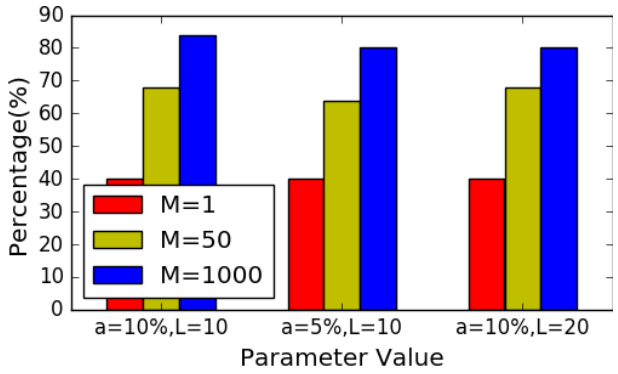


Figure 4.16: The percentage of datasets in which DATAMARAN can find the optimal structure on different parameters

Note that it is not necessary for DATAMARAN to find the optimal structure, and the metric used in this section is solely for comparison purposes. Based on the results throughout this section, we suggest using the following default parameter configuration in practice: $\alpha = 10\%$, $L = 10$, $M = 1000$.

4.8.2 GitHub Datasets

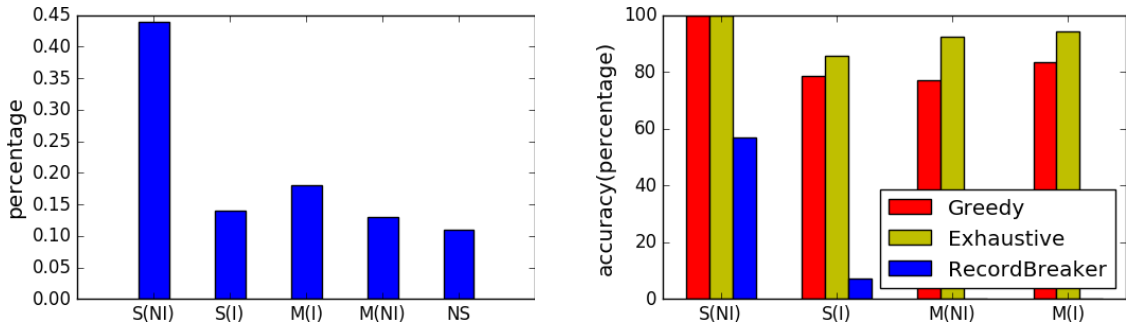
GitHub contains a large quantity of log datasets generated by programmers across the world. We collected 100[5] of such datasets by uniformly sampling from the first 1000 search results using the following three criteria: (a) files end with ".log" (b) with length greater than 20000 (c) contains one of the following keywords[6] : "db", "2016", "system", "query", "user". The datasets are sampled before any follow-up analysis is conducted, so it represents an unbiased subset of the whole dataset.

[5]The scale is limited to 100 since we have to manually inspect the datasets and the extraction results. DATAMARAN can be automatically applied to thousands of datasets without any problem.

[6]GitHub search function requires at least one search keyword, and we used multiple keywords to improve the diversity of our selection.

Label	Description
S (Single-line)	Dataset consists of only single-line records.
M (Multi-line)	Dataset contains records spanning multiple lines
NI (Non-Interleaved)	Dataset consists of only one type of records.
I (Interleaved)	Dataset contains more than one types of records.
NS (No Structure)	Dataset has no structure or its structure does not follow Assumption 4.1, 4.2 or 4.3.

Table 4.4: GitHub Dataset Labels



(a) Characteristics (b) Accuracy

Figure 4.17: GitHub Datasets: Characteristics and Accuracy

Dataset Characteristics

The sampled datasets are categorized based on three criteria:

- *whether the dataset contains multi-line records*
- *whether the dataset consists of multiple types of records*
- *whether the dataset has any structure at all*

There are five possible labels of datasets based on the above criteria, which are listed in Table 4.4. The distribution of labels among the 100 sampled log files is shown in Figure 4.17(a), from which we can draw the following conclusions:

- **Validity of Structural Assumptions:** 89% of datasets follow the structure assumptions in DATAMARAN (Assumption 4.1, 4.2 and 4.3). Note that among the remaining 11%, 10% of the datasets have no structure at all (nothing can be extracted from these datasets), and only 1% dataset have structure that cannot be described within our framework. These statistics suggest that our structural assumptions are well-justified for log datasets.
- **Necessity for Multi-line Record Handling:** 31% of datasets contains at least one type of record spanning multiple lines. The optimal structure in these datasets cannot be successfully extracted if the extraction system cannot handle multi-line records.

- **Necessity for Interleaved Records Handling:** 32% of datasets contains more than one type of records. If the extraction system cannot recognize the existence of multiple types of records, only one type of record can be extracted (the rest will be regarded as noise), resulting information loss.

Evaluation Criteria

Recall that the structure extraction problem is not well-posed, and the validity of the extracted structure solely depends on the end-user. For many datasets, there are usually multiple structures that can potentially be deemed as valid. For example, the dataset `[01:05:02] 192.168.0.1` has at least the following 4 valid structure templates:

```
[F] F\n          [F] F.F.F.F\n          [F:F:F] F\n          [F:F:F] F.F.F.F\n
```

Thus, it is not possible to directly compare the extracted structure with a manually labeled structure. Here we define the following evaluation criteria: for each dataset, we first identify several different types of records within the dataset, then identify as many intended extraction targets as possible for each type of record (i.e., observable fields with potentially interesting information). The extraction is considered successful *if both of the following two criteria are met: (a) all of the record boundaries and record types are correctly identified; (b) for each type of intended extraction target, we can select several fields from the structure template, such that all of the intended extraction targets (of this type) can be reconstructed by concatenating the selected fields from the corresponding record.* Figure 4.18 demonstrates an example successful extraction, in which we have two types of intended extraction targets (i.e., time and IP address), and Datamaran returns the structure template as shown in the middle of the figure. In this example, the extraction is considered successful because both types of intended extraction targets can be reconstructed by concatenating field values at specific positions for all extracted records. If, instead, the targets were extracted together, reconstructing them via concatenation would not be possible.

Extraction Accuracy

We applied Datamaran to extract structured information from GitHub datasets. Figure 4.17(b) shows extraction accuracy for different types of datasets (based on the above evaluation criteria). Overall, Datamaran successfully extracted structure from 85 datasets. The accuracy is 95.5% if we exclude datasets with no structure.

As we can see in Figure 4.17(b), Datamaran achieved 100% accuracy on single-line non-interleaved datasets, the simplest type of dataset. The accuracy of Datamaran for the other three types of datasets are 85.7%, 92.3% and 94.4% for exhaustive search, and 78.6%,

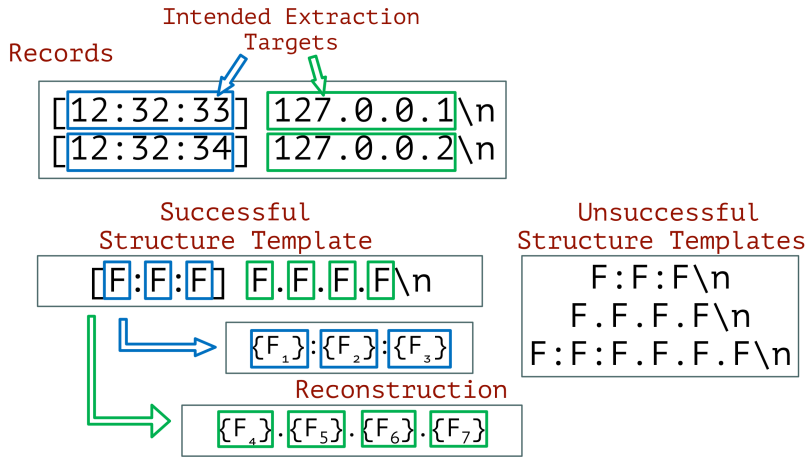


Figure 4.18: Successful and Unsuccessful Extraction Examples

76.9%, 83.3% for greedy search. Therefore, we conclude that DATAMARAN is effective for most of the log datasets in practice.

Figure 4.17(b) also shows the extraction accuracy of RecordBreaker [35] with default configurations and parameters for comparison. As we can see, RecordBreaker performs very poorly on log datasets with accuracy 56.8% and 7.1% on S(NI) and S(I) respectively and 0% on M(NI) and M(I), for a total of 29.2% accuracy, which is not very surprising: RecordBreaker is originally designed for well-structured datasets, and cannot handle the noise-heavy log datasets very well. Furthermore, the resulting structure templates depend a lot on the Flex configurations and the tuning of two parameters in RecordBreaker (i.e., MaxMass and MinCoverage). This is because Flex configurations decide the quality of tokenization, while the other two parameters determine the datatype (i.e., *struct*, *array* or *union*) for a given list of records. However, there are no generic configurations or parameter values that work for all datasets, *which makes RecordBreaker less desirable in an unsupervised setting and incomparable to* DATAMARAN.

Figure 4.17(a) and Figure 4.17(b) also demonstrates why prior work such as RecordBreaker [35] is not well-suited for extracting structure from log datasets: for any dataset containing multi-line records, the task of partitioning such dataset into collection of records is nontrivial (due to the presence of noise & the fact that record span is unknown). From Figure 4.17(a), we see that at least 31% of datasets cannot be handled by RecordBreaker [35] as demonstrated by M(NI) and M(I) in Figure 4.17(b).

Causes for Inaccurate Extraction

There are primarily two causes for inaccurate extraction from GitHub log datasets. There are 4 log files where even the exhaustive search version of DATAMARAN failed to find a valid structure. In the following, we list the two causes for these inaccurate extractions.

Fail to recognize "long" records: The maximum range of records is set to be 10 lines during the experiments. In some datasets, there are some extremely "long" records that exceeds this limit. If we increase the range limit, the efficiency of DATAMARAN would suffer. As the records in practice can be arbitrarily long, we are still unaware of methods that can completely solve this problem.

The greedy approach for interleaved datasets: In DATAMARAN, we handle interleaved datasets by repeatedly applying the algorithm on the dataset. However, this greedy procedure does not always find the correct structure for interleaved dataset. Instead, sometimes we would find structure templates with characteristics of multiple types of records. The following example demonstrates this phenomenon: Suppose we have two types of records with templates: `F: F F F\n` and `F: F F F F F F\n`, then DATAMARAN could potentially settle on the wrong structure template `F: (F )*F\n`, when this generic structure template has a lower regularity score compared to the two correct record templates.

4.8.3 User Study

To further evaluate the quality of the structure extracted by DATAMARAN, we conducted a user study on five representative log datasets, comparing our results against the raw datasets as well as the extracted results of RecordBreaker.

Study Design

Our user study simulates the following scenario, where a participant is presented with a log file, and they want to extract some information of interest, prior to analysis. One straightforward way to do so is to import the log file into a spreadsheet tool like Microsoft Excel, and then use Excel functionalities to extract this information. Alternatively, the participant can first use either DATAMARAN or RecordBreaker to extract the structure, and then refine the results using Excel to obtain the desired structure and filter out anything that is not of interest. We will compare these three methods (i.e., from the raw log file, from the result of DATAMARAN/RecordBreaker) in our user study, and an optimal extraction result is created based on our best judgement so that we can quantify the manual effort taken to reach the desired extraction result. For each dataset, we show the raw log file as well as the extraction results of DATAMARAN and RecordBreaker to the participants, and ask them to transform each file into the desired target structure.

Methodology

The user study consists of three phases:

(1) Introduction phase: We first show the participants an example of the raw file, extraction results from DATAMARAN and RecordBreaker, along with the target file, denoted as R, A, B and T respectively. Then, we introduce four popular Excel data wrangling functionalities that may be used for transforming those three files into the target file, *Concatenate, Split, FlashFill* and *Offset*. *Concatenate* and *Split* are straightforward; *Flashfill* autocompletes columns from a few user examples [41]; and *Offset* can be used to copy contents every K rows while skipping the $(K-1)$ rows in-between.

(2) Quiz phase: We present five folders to the participant, one for each dataset, where each folder contains the raw file (R), two extraction files (A and B) and the target file (T). One dataset is a single-line dataset while the other four are multi-line datasets. For each dataset, the participant is asked to transform R, A and B into T using the functionalities[7] in Excel. The whole process takes around one and half hours per participant.

(3) Survey phase: we conduct a survey to understand the participant's experience in structure extraction using the raw file R and the two extraction files (A and B).

Detail of introduction phase: in the introduction phase, we give an tutorial on the usage of four common data wrangling features in Excel: *Concatenate, Split, FlashFill* and *Offset* in Microsoft Excel. The functionality of each operation is listed as follows:

(a) *Concatenate* merges the strings from multiple cells into a combined string.

(b) *Split* splits a string into multiple cells via delimiters.

(c) *Offset* can be used to copy contents every K rows while skipping the $(K-1)$ rows in-between. For example, $\mathit{offset}(B\$1, (\mathit{row}()-1)\times 5, 0, 1, 1)$ refers to the cell with $(\mathit{row}()-1)\times 5$ row offset and 0 column offset from the reference cell $B\$1$, where $\mathit{row}()$ is the row id. By specifying this formula, Excel extracts content every 5 rows and skips the 4 rows in between. In our user study, *Offset* helps reconstruct records spanning multiple rows.

(d) *FlashFill* is different from the above cell-based operations, and is content-based. It can automatically fill the data if it detects a pattern between input examples and the original data in Excel. In some sense, the functionalities of *FlashFill* subsume those provided by both *Concatenate* and *Split*. However, compared to *Split* which splits each column into multiple columns simultaneously, *FlashFill* can only fill in one column at a time.

[7]The participant can use any functionality in Excel, not limited to the ones we teach in the introduction phase.

Furthermore, *FlashFill* sometimes detects the wrong pattern, but by providing a few more examples, *FlashFill* can correct the mistake and provide the correct results.

The complexity for using each operation is not uniform: *Concatenate*, *Split* and *FlashFill* are very easy to use, while *Offset* requires more thought and effort in writing the formula since it involves the manipulations over multiple rows and is not very intuitive.

Detail of quiz phase: in the quiz phase, among the representative five datasets we present to the participants, one of them is a single-line dataset while the other four are multi-line datasets. Among the four multi-line datasets, two of them have a regular pattern, while the other two have noise. The raw dataset R, the extracted result using DATAMARAN A and the target result T is stored in a single file each, while there may be multiple files for the extracted results using RecordBreaker due to "union" structure type in their algorithm. We output the extraction results using RecordBreaker into multiple files if it recognizes multiple structures.

Participants: Six users participated in our study, including four graduate students from Computer Science and one graduate student from Electrical and Computer Engineering. Three out of six work with data very often (daily), one often (weekly) and one rarely (yearly or fewer). In addition, every participant has used spreadsheets and scripting language(s), like Python and Matlab, for data analysis, while two participants had also used business analytics tools like Tableau and Power BI.

Results

For each dataset, we recorded the action sequences performed by each participant during the transformation. In total, there are $6 \times 3 \times 5 = 90$ sequences (six participants, three file types, and five datasets). Each sequence is depicted by a horizontal line in Figure 4.19, where each colored circle denotes a specific operation[8] performed by the participant, as shown in the legend. The x-axis is the operation's index in the sequence, and y-axis shows the participant id and the file type. For instance, $R.u_1$ refers to the first participant (u_1) and the task is to transform the raw file (R) into the target file.

As shown in Figure 4.19, participants took more operations to transform the raw file (if no failure occurred) as opposed to extracted files using DATAMARAN and RecordBreaker. This verifies the usefulness of automated extraction tools. Furthermore, participants always took the least number of steps to reach the target file T when using DATAMARAN, with no failure. On the contrary, they were often unable to transform the raw file R and the extracted file using RecordBreaker B, as shown in Figure 4.19(b,d-e). This occurred mostly

[8]We ignore the simple operations like Delete, Copy, Paste.

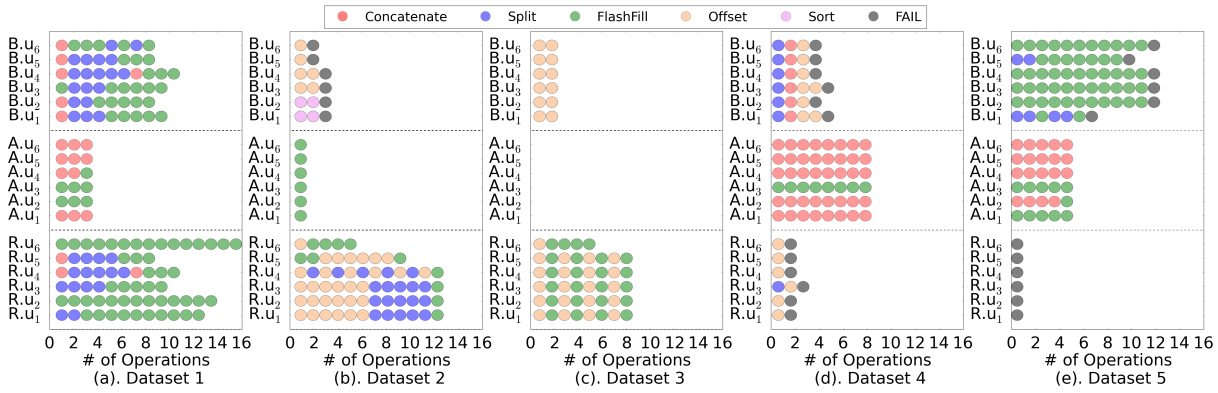


Figure 4.19: Sequence of Operations for Transformation

when the records span multiple-lines and when the dataset is noisy. Next, we will discuss the findings for each dataset briefly. More details can be found in our technical report [42].

Dataset 1 is a single-line dataset, and the extraction results of both RecordBreaker and DATAMARAN are much better structured than the raw file. Compared to R and B, A took the smallest number of steps in order to be transformed to T, as illustrated in Figure 4.19(a). When it comes to multi-line record datasets, i.e., datasets 2-5, DATAMARAN exhibits a much more substantial advantage over RecordBreaker and the raw file. First, when there is noise or incomplete records in the dataset (dataset 4 and 5), participants needed to either manually filter the incomplete records one by one, or write some sophisticated code to remove the noise and reconstruct the records. This step is often laborious or hard to implement. Second, RecordBreaker treats each single line as a record unit, and would recognize each line as a different structure, which are then stored into different files. Hence, the participants often found themselves losing context for reconstructing the records when each record spanned multiple files. As a consequence, participants often failed to transform B and R into T after some trials, as shown by the black circles in Figure 4.19(b,d-e). Due to the context missing in B, participants could only figure out that they failed to reconstruct the rows after a number of operations, as illustrated in Figure 4.19(e).

The major findings from Figure 4.19 are summarized as follows:

(a) Starting from the extracted results of RecordBreaker/DATAMARAN helps the participant "fast-forward" to the desired target structure, compared to the raw file R.

(b) The extracted results of DATAMARAN are already in a very fine-grained clean format, requiring very simple operations, i.e., Concatenate or FlashFill, to concatenate the fine-grained results to the most desirable target format T.

(c) For multi-line datasets, it is hard to obtain information from both the raw file R and the

results of RecordBreaker B, as evidenced by the **failure** (black circle) in Figure 4.19.

Survey and Interview

Most participants (5/6) reported that A (DATAMARAN) is very easy to use, requiring only merging (i.e., *Concatenate* and *FlashFill*) and deleting operations most of the time. But some participants also complained that A still requires a bunch of manual work, like repeating *Concatenate*. This is because the extracted result of DATAMARAN is very fine-grained. The large number of repeating operations on *Concatenate* or *FlashFill* is captured in Figure 4.19(d). On the other hand, all participants (6/6) complained that the raw file is hard to begin with, since it looks messy and is difficult to find the pattern inside. In addition, participants were not satisfied with the extracted results by RecordBreaker, since they were annoyed by the multi-file and multi-line merge operations like Offset. On average, participants rated the difficulty of performing transformation from A, B, and R to T as 1.8, 7.8, and 9.3 respectively, where 1 indicates the easiest and 10 indicates the most difficult.

In particular, one participant (u_4) said the following-"For A, it is ready to use, involving mostly merge and delete operations. For B, there is lots of extra operations. It's hard to carefully use Offset to merge lines and merging across rows could be painful and error prone. For R, it is impossible to do manually. I prefer to write code, but need to make sure the code is bug free." Another participant (u_6) said the following–"No major difficulty for A. Each row corresponds to exactly one record. For B, there is information lost during processing, hard (impossible?) to join disparate partially processed items together. R requires significant manual effort to identify anomalous records before automatic techniques can be applied to put data in structured format." There were also some limitations identified for DATAMARAN (A). One participant (u_1) said the following–"For A, it only involves single file operators, easier to track, but still a lot of manual work. For B, it requires cross file operations, difficult to track, and sometimes you end up choosing sub-optimal operations. For R, it is unstructured, need to create tuple using *Offset* first, most laborious among the three."

Summary: All participants ranked the extracted results by DATAMARAN (A) easiest to use, and the raw file R most difficult to use. This is mostly because the structure in the raw file is unclear, while DATAMARAN provides very clear structure. From the user study, we conclude that DATAMARAN has better extraction result than RecordBreaker, and both tools are a better starting point than the raw file.

Limitations: However, since our user study is limited to the comparison between two automated structure extraction tools, i.e., DATAMARAN and RecordBreaker, and one supervised

extraction feature in Microsoft Excel, i.e., FlashFill, it remains to be seen whether unsupervised tools can perform comparably well as other more advanced supervised tools. Also, the many concatenate operations (e.g., assembling IPs from fragments) can be tedious. For such domain-specific data types. DATAMARAN should be enhanced with type awareness (e.g., for phone numbers, IPs, URLs).

4.9 BIBLIOGRAPHICAL NOTES

DATAMARAN is related to the vast bodies of work on general information extraction, as well as the more limited work on log dataset extraction, and string transformation. Sarawagi [43] provides an excellent summary of the information extraction area.

Example-Driven HTML Wrapper Induction. There has been a long line of work on inducing or learning a "wrapper" to extract content from HTML pages, e.g., [44, 30, 45, 46, 47, 29]. The majority of these papers crucially rely on both the web-page structure in the form of the DOM, as well as on text (e.g., extract the piece of text immediately following "Price:"). Examples are provided in the form of entities that belong to the concept class that are to be extracted, or in the form of explicit annotations (e.g., this location contains an item of interest to be extracted). Often, the eventual relational schema is known in advance. Some papers do not rely on the HTML structure, opting instead to use NLP [48, 49]. In our case, we do not require any seed entities or annotations.

Unsupervised HTML Wrapper Induction. A few papers attempt to extract from HTML pages directly, without requiring any training examples [50, 51, 52, 53, 54, 55, 56]. In the following, we discuss the core ideas of these papers and explain why their ideas cannot be applied to log datasets without substantial modification.

First, we briefly summarize the contents of these papers as follows:

- **Roadrunner [51]** takes a collection of HTML pages as input, and the output of their algorithm is a minimal union-free regular expression that generates all the pages in the collection. Their algorithm repeatedly applies pairwise reduction on each page and the current template to reduce all of them into a single regular expression.

- **ExAlg [50]** takes a collection of HTML pages as input, and their output is a minimal page template (which also has tree-style form and is very similar to union-free regular expression). Their algorithm relies on the idea of using equivalence classes (i.e., sets of tokens that appear exactly the same number of times in each page) to identify tokens

that have the same role in the template. Additionally, tree structures are used to differentiate multiple roles of the same type of tokens.

- **TEX [53, 52]** takes a collection of documents as input, and their output is a collection of field value lists. Their algorithm works by searching for the longest shared pattern among all documents in the collection, and this shared pattern is then used to partition the documents in the collection to create finer grained collections (on which their algorithm can be applied recursively). Their algorithm does not guarantee that all the field values of the same type will be included in a single list, and their lists can also have different sizes. As a result, their output cannot be easily converted into a relational format.

- **MDR [54, 55]** takes a single HTML page as input, and identifies boundaries of data records as well as the record templates for each record type within that HTML page. Their algorithm first identifies several data regions, such that each data region contains exactly one type of record and no noise. Then it identifies record boundaries and data fields using tree alignment. A data region is identified by enumerating the region boundary and the span of each "generalized node" (i.e., the number of tag nodes each such generalized node contains).

- **FiVaTech [56]** takes a collection of HTML pages as input, and uses tree alignment to find the optimal template that generates all pages in the collection.

From the above descriptions, we can see that there are primarily five major techniques used in these papers, and each of them rely on some characteristics of HTML pages that do not exist in log datasets:

- **Equivalence Class Identification.** Equivalence classes are sets of tokens that appear exactly the same number of times in each document. Finding such equivalence classes requires the input to be a collection of documents to begin with. However, a log dataset is initially one single document, and we need to know at least a subset of actual record boundaries in order to partition the log dataset into a collection of documents. Furthermore, the equivalence class technique requires each document in the collection to be noise-free, but most real-world log datasets are very noisy.

- **Tree Alignment.** A tree alignment method is used to compute the tree template of multiple tree-style records. Performing tree alignment requires the documents to have tree-structures to begin with: while tree structures arise naturally from HTML pages, log datasets do not have tree-structures initially.

- **Token Role Differentiation.** ExAlg [50] relies on the tree-paths of tokens to differentiate their roles in the template. This technique requires the documents to have tree-structures to begin with. Similar to the discussion of "Tree Alignment" technique, log datasets do not have tree-structure typically.

- **Shared Pattern Identification.** The implicit assumption in the shared pattern identification approach is that the collection is "uniform" and noise-free: that is, even though the structural form of each document in the collection is relatively flexible (e.g., "ABB...BC", where A,B,C are record types), all of the documents in the collection must have exactly the same form (i.e., starting from A, followed by any number of Bs, and ends with C). Similar to the discussion of "Equivalence Class" technique, log datasets cannot be easily partitioned into such collections.

- **Data Region Identification.** Identifying the data region requires the input document to actually have multiple data regions, such that within each data region, there is only one type of "generalized node" and is otherwise noise-free. Note that each generalized node can contain multiple types of records, but all generalized nodes must have the same format (e.g., ABABAB... is a legitimate data region, in which A and B are two different types of records, and AB is the generalized node in the region). Unfortunately, log datasets do not have such a property, since in a log dataset different types of records can appear in an arbitrary order (e.g., ABAABABB).

Table 4.5 summarizes the techniques used by each method and the assumptions made for the input documents. The four key assumptions are: (a) Partial Record Boundaries (PRB); (b) Non-Interleave (NI); (c) Noise-Free[9] (NF); (d) Tree-Structure (TS). As we can see, all these methods make multiple assumptions that do not hold for log datasets, and as a result there is no easy way to adapt these techniques for log dataset structure extraction.

Method	Core Techniques	PRB	NI	NF	TS
Roadrunner [51]	Tree Alignment	Y	Y	Y	Y
ExAlg [50]	Equivalence Class & Token Role	Y	Y	Y	Y
TEX [53, 52]	Shared Pattern Identification	Y	Y	Y	N
MDR [54, 55]	Data Region & Tree Alignment	N	Y	Partial	Y
FiVaTech [56]	Tree Alignment	Y	Y	N	Y

Table 4.5: Unsupervised Wrapper Induction: Techniques and Assumptions.

However, we remark that it might be possible that some of these techniques can potentially be adapted and integrated into DATAMARAN to further improve its effectiveness. For

[9]MDR requires each data region to be noise-free, but noise can appear between data regions.

example, a tree alignment technique can be potentially integrated into the generation step to further enhance recall. We leave the exploration of such possibilities as future work.

Extracting Structure From Other Media. There is work [32, 57, 58, 59] on extracting structure from other types of media (i.e., other than text-formatted log datasets). The extraction strategies adopted by these papers crucially rely on characteristics of the target dataset type. For instance, in security research [32, 57], the network traces consist of continuous communication between server and client, best modeled as a deterministic state machine (i.e., messages between server and client represent transitions in the global state), and reliant on indicators that signal the start of a new message, e.g., the presence of an IP address; in either case, the record boundaries are clear. On the other hand, in the field of natural language processing [58, 59], the structure is usually restricted to local context (i.e., within each sentence), and can be captured using probabilistic language models. In particular, Cohen et al. [58] employs language models from other languages to learn the structure of a new language, while Spitkovsky et al. [59] uses clustering based on local context (neighboring words to a given word) to infer dependency structures to inform a sentence parser, where parsing is delimited based on periods. In our case, the fundamental characteristics of log datasets are captured in Definition 4.3, and our whole extraction strategy revolves around this definition.

Extraction from Web Documents. There has been some work on extraction from other forms of documents, or portions of Web documents, typically leveraging example concepts [60] or a knowledge-base [61, 62, 63] to extract entities and attributes from text files.

List extraction, i.e., extraction from lists on the web is another area that has seen some work [31, 64, 65, 66]. Some of these papers require both the eventual relational schema as well as candidate examples to be provided [65, 66]. Some papers attempt fully-automated list extraction [31, 64, 55]. These papers make the crucial assumption of each record corresponding to a single list item, making it easy to extract the boundaries of the records. Our space of datasets—log files—do not admit any such assumption.

Log Dataset Extraction and Transformation. Wrangler [67] supports the interactive specification of log dataset cleaning operations, drawing from the transformations in Raman et al. [68]. Instead of operator specification, other work relies on user-provided input-output examples [69, 41, 33, 70, 71] to transform one semi-structured dataset to another. In our case, we do not require any intervention from the user. The PADS project [34] relies on a user-provided chunker and tokenizer to identify the boundaries of records/field values, while RecordBreaker is a line-by-line unsupervised implementation, with a fixed lexer configuration

which makes it inflexible for real log datasets. Recent work by Raza and Gulwani [72] describe an automatic text extraction DSL for single-line extraction, generalizing to both web-pages and text documents.

Other work clusters event logs [73, 74] by treating the lines of the log dataset as data points and assigning them to clusters. Compared to our work, these papers do not attempt to identify the structure within records, and they do not consider the possibility of multi-line records.

4.10 CONCLUSION

The structure hypothesis space design in DATAMARAN represents a careful balance between learnability and utility: note that Fisher et al.'s work [34] already includes a tree-style structure that is very similar to Assumption 4.1. However, the crucial difference is that we removed the union-type node from Fisher's original design, which significantly reduced the complexity of the structure hypothesis space. In our experiments, we have seen that Fisher's original structure design not only caused a severe learnability issue (the extraction algorithm described in [34] does not really work for complex structures based on our experiments), but also makes it difficult to import the contents into relational format. On the other hand, the removal of union-type structure does not really hinder the effectiveness of our method, and the procedure for importing data into relational format also becomes much more natural as a result.

Even though the assumptions in DATAMARAN, especially Assumption 4.3, may seem too restrictive, they are still an acceptable compromise we came up with to work around the difficulties, and they actually do not hinder the algorithm's utility by too much. For most real-world log datasets, there are more than one valid structure template that can be used for parsing the dataset, and among all the valid structure templates, usually at least one of them would satisfy both assumptions. Therefore, DATAMARAN can still work effectively as long as it can find one structure template that satisfies all assumptions.

Algorithm 4.1 The Generation Step

```
    function GenST(char_set)
        n ← Total Number of Lines
        for i ← 1 to n do
            for j ← i + 1 to i + L do
 5:             left_boundary ← i
                right_boundary ← j
                r ← ExtractRecord(left_boundary, right_boundary)
                rt ← ExtractRecordTemplate(r, char_set)
                st ← GenerateStructureTemplate(rt)
10:             k ← ComputeHashKey(st)
                cov(k) ← cov(k) + length(r)
                st_set(k) ← st_set(k) ∪ {r}
            end for
        end for
15:     Find all hash keys with more than α% coverage.
        return the associated structure templates.
    end function
    function ExhaustiveSearch(char_candidates)
        ST_set ← ∅
20:     for char_set ⊆ char_candidates do
            ST_set ← ST_set ∪ GenST(char_set)
        end for
        return ST_set
    end function
25: function GreedySearch(char_candidates)
        char_set ← ∅
        ST_set ← ∅
        repeat
            new_best_char_set ← ∅
30:         best_f ← 0
            for c ∈ char_candidates \ char_set do
                new_char_set ← char_set + c
                new_ST_set ← GenST(new_char_set)
                ST_set ← ST_set ∪ new_ST_set
35:             for st ∈ new_ST_set do
                    if AssScore(st) > best_score then
                        best_score ← AssScore(st)
                        new_best_char_set ← new_char_set
                    end if
40:             end for
            end for
            char_set ← new_best_char_set
        until no structure template has at least α% coverage
        return ST_set
45: end function
```

Algorithm 4.2 The MDL Regularity Score Function

```
    function EvalST(ST)
        (RecordBlocks, NoiseBlocks) ← ParseData(ST)
        Determine the data types of field values
        Learn the distributional parameters
 5:     TotalDL ← len(ST) × 8 + 32 + NumBlocks
        for record ∈ RecordBlocks do
            RT ← GetRecordTemplate(record)
            TotalDL ← TotalDL + D(RT|ST)
            TotalDL ← TotalDL + D(record|RT)
10:     end for
        for block ∈ NoiseBlocks do
            TotalDL ← TotalDL + len(block) × 8
        end for
        return TotalDL
15: end function
```

CHAPTER 5: EXTRACTING STRUCTURE FOR INSIGHT DISCOVERY

Relational datasets often have missing values encoded as NULLs. Filling in such NULL values can greatly enhance the completeness of the dataset, and also provide insights into missing entries that are impossible to observe (e.g., due to the limitations of data gathering, or because they represent hypothetical scenarios). Unfortunately, most existing work on predicting missing values target specific scenarios (e.g., the dataset contains only a single table [75], or has a star-style schema [76]), and there aren't many papers investigating the design of generic algorithmic procedure for arbitrary relational datasets. In practice, data analysts usually start by identifying the core characteristics of the target dataset, and then design a specialized algorithm that utilizes the identified characteristics: for example, if we want to predict the missing entries of social network user profiles, we can rely the fact that people connected with each other in the network usually have similar ages, live in the same place, or share some other similarities. This can indicate to the designer how the dataset and prediction task must be constructed. While such a strategy often works extremely well in general, it requires the algorithm designer to have both a good understanding of the dataset semantics, as well as the knowledge of various prediction techniques developed in literature, so that they can adapt suitable ones for the task at hand.

A good prediction algorithm that works over a collection of arbitrary relational datasets can serve as an affordable alternative to the specialized prediction algorithm developed by professional data analysts. It can be useful in scenarios that demand automation at the cost of slightly reduced accuracy (e.g., for performing an exploratory investigation on external datasets in a collaborative research scenario). However, the development of such an algorithm is much more challenging, due to the fact that the characteristics of datasets are unknown to the prediction algorithm. It is immediately clear that a hidden-structure extraction phase is necessary for such an algorithm, and there are two separate but equally important challenges for designing this prediction algorithm:

- **How do we evaluate the quality and trustworthiness of predictions?** As the hidden structure extraction step operates in an automated fashion, sometimes the algorithm will not be able to discover any meaningful structure, and the predictions will have low accuracy as a result. In such a case, it is important to inform the users so that they are aware of the situation. In other words, the algorithm should be able to evaluate the reliability of predictions, and quantify it in a meaningful manner.
- **What are the hidden structures that are capable of capturing important characteristics for helping with the prediction task?** In order to choose a

good structure hypothesis space for a hidden-structure based algorithm, we need to first summarize the dataset characteristics that are potentially useful for our purpose. However, this step is not very straightforward as we are considering the input space of all possible relational datasets (which can have arbitrarily complex schema). There are simply too many possibilities one can think of: for instance, while correlation often happen between attributes within the same tuple, in the previously mentioned social network prediction task, correlations occur between attributes of different tuples. Because of the extreme flexibility of relational data model, summarizing the most prevalent and essential characteristics of datasets is often challenging.

These two challenges will be discussed separately in the rest of this chapter: in Section 5.1, we introduce the calibration property of conditional probability estimates, and discuss how to obtain such a property in practice; in Section 5.2, we discuss three major approaches for relational learning in literature (i.e., methods that data analysts have employed for specific tasks in the past), and generalize them to handle relational datasets with arbitrary schema. After discussing these two challenges, we will present several experimental results in Section 5.3, which can help us understand and evaluate the pros and cons of the prototype methods.

5.1 THE CALIBRATION PROPERTY OF CONDITIONAL PROBABILITIES

Recall that our goal is to design a fully automated prediction algorithm for an arbitrary relational dataset: regardless of what kind of input dataset we are given, our algorithm should always be able to output predictions for the missing entries within such a dataset. Obviously, depending on the nature of the dataset as well as the prediction method that we use, the confidence level of our predictions could be drastically different from instance to instance: sometimes we may be fairly certain about our predictions, while in other cases we can be simply making wild guesses. Due to this, the ability to evaluate the confidence level of predictions and quantify it in a meaningful manner is of crucial importance.

One possible approach is to designate a separate validation dataset before all the training and prediction procedures begin, and use it to provide global statistics: accuracy/F_1 score/Mean Squared Error (MSE), etc. While such global statistics do offer important insights into the reliability of our predictions, these statistics are not exactly ideal in the sense that they only report "averaged" reliability over all of our predictions. In practice, the predictions usually do not have uniform confidence levels: we might be fairly certain about some of the predictions, while entirely unsure about others. If we can quantify our confidence level

on a per-item basis, it could enable much more flexible use of our predictions: users could utilize only the predictions with a high confidence level, while disregarding the others.

Conditional Probabilities and The Calibration Property

The conditional probabilities of labels are often viewed as a convenient tool for assessing the per-item confidence level: for each missing entry in the dataset, we require the algorithm to predict not only the most probable value, but also the likelihood of every value appearing in that slot. The prediction of conditional probabilities has been extensively studied in the literature, and most existing machine learning algorithms either naturally (e.g., logistic regression, neural networks) or can be tuned to provide conditional probabilities as well (e.g., SVM [77], decision tree [78] or nearest neighbor). Unfortunately, while the accuracy of global statistics is naturally guaranteed by the law of large numbers, the conditional probability estimates do not naturally enjoy such formal guarantees. In other words, while our algorithm can provide conditional probabilities as the assessments of confidence levels for our predictions, such assessments can be completely unreliable and their usefulness is severely compromised as a result.

Now, one may wonder *whether it is possible to achieve any formal guarantee(s) regarding conditional probabilities (similar to how the reliability of global statistics is guaranteed by the law of large numbers)?* To answer this question, we provide some preliminary results for properly interpreting the confidence measures generated by conditional probability estimators [6]. Clearly, it would be ideal if the predicted conditional probabilities are accurate on a per-item basis (i.e., the predicted probability equals to the true value according to the underlying generative model), but unfortunately, this has been proven to be impossible in agnostic scenarios [6], wherein we are unaware of the exact semantics of data entries in a dataset. However, it is still possible to guarantee a certain calibration property for the conditional probability estimates, defined as follows:

Definition 5.1 (Calibration Property)**.** *A supervised classification task is characterized by four elements: the feature space $\mathcal{X}$, the finite target set $\mathcal{Y}$, the data generation distribution $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$, and a training dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ consisting of i.i.d. samples from $\mathcal{P}$. A conditional probability estimator $f : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ estimates the value of $\mathbf{Pr}(Y|X)$, the conditional distribution of Y given X, for each feature-target pair.*

Let $f : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ be one such conditional probability estimator, we say f is (perfectly) calibrated with respect to $\mathcal{P}$ if for any probability interval $(p_1, p_2]$ ($p_1, p_2 \in [0, 1]$) and any target value $y_0 \in \mathcal{Y}$, the estimated relative frequency of attribute value y_0 among all data instances satisfying $p_1 < f(X, y_0) \leq p_2$ is equal to the actual relative frequency of y_0 among

these data instances:

$$\forall p_1, p_2 \in [0,1], y_0 \in \mathcal{Y}, \quad \mathbb{E}_{(X,Y)\sim\mathcal{P}}[f(X,y_0)\mathbb{1}_{p_1<f(X,y_0)\leq p_2}] = \mathbb{E}_{(X,Y)\sim\mathcal{P}}[\mathbb{1}_{Y=y_0}\mathbb{1}_{p_1<f(X,y_0)\leq p_2}] \tag{5.1}$$

and we say f is (perfectly) calibrated with respect to a dataset D if

$$\forall p_1, p_2 \in [0,1], y_0 \in \mathcal{Y}, \quad \frac{1}{n}\sum_{i=1}^{n} f(x_i,y_0)\mathbb{1}_{p_1<f(x_i,y_0)\leq p_2} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i=y_0}\mathbb{1}_{p_1<f(x_i,y_0)\leq p_2} \tag{5.2}$$

When f is not perfectly calibrated, the supreme of differences between the two quantities in the above equation is called the calibration error of f with respect to the distribution $\mathcal{P}$ or the dataset D:

$$c_P(f) = \sup_{p_1,p_2,y_0} |\mathbb{E}_{(X,Y)\sim\mathcal{P}}[f(X,y_0)\mathbb{1}_{p_1<f(X,y_0)\leq p_2}] - \mathbb{E}_{(X,Y)\sim\mathcal{P}}[\mathbb{1}_{Y=y_0}\mathbb{1}_{p_1<f(X,y_0)\leq p_2}]| \tag{5.3}$$

$$c_D(f) = \sup_{p_1,p_2,y_0} |\frac{1}{n}\sum_{i=1}^{n} f(x_i,y_0)\mathbb{1}_{p_1<f(x_i,y_0)\leq p_2} - \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i=y_0}\mathbb{1}_{p_1<f(x_i,y_0)\leq p_2}| \tag{5.4}$$

Intuitively, f has the calibration property if for any probability value p and target value y_0, if we consider all data points X satisfying $f(Y = y_0|X) = p$, then exactly p portion of them would actually have label y_0. In other words, f does not have to agree with the true conditional probabilities $Pr(Y|X)$ on every item, as long as they agree on average within each respective group (i.e., items with the same prediction value).

The Motivation of the Calibration Property

To motivate the calibration property, let us consider the following example:

Example 5.1. *Denote Z to be the collection of all English words. In this problem the feature space $\mathcal{X} = Z^*$ is the collection of all possible word sequences, and $\mathcal{Y}$ denotes whether this document belongs to a certain topic (say, football). Let $\mathcal{P}$ be the following data generation process: X is generate from the Latent Dirichlet Allocation model [79], and Y is chosen randomly according to the topic mixture.*

We use logistic regression for computing the conditional probability of each label Y, which is parameterized by a weight function $w_Y : Z \to \mathbb{R}$, and two additional parameters a_Y and b_Y. For each document $X = z_1 z_2 \dots z_k$, the output of the conditional probability estimator is:

$$f(X,Y) = \frac{1}{1+\exp(-a\sum_{i=1}^{k} w(z_i) - b)} \tag{5.5}$$

Now consider the case where we fix the word weight function w_Y. In this case, every document X can be represented using a single parameter $w_Y(X) = \sum_i w(z_i)$, and we search for the optimal a_Y and b_Y such that the log-likelihood is maximized. This is illustrated in Figure 5.1.

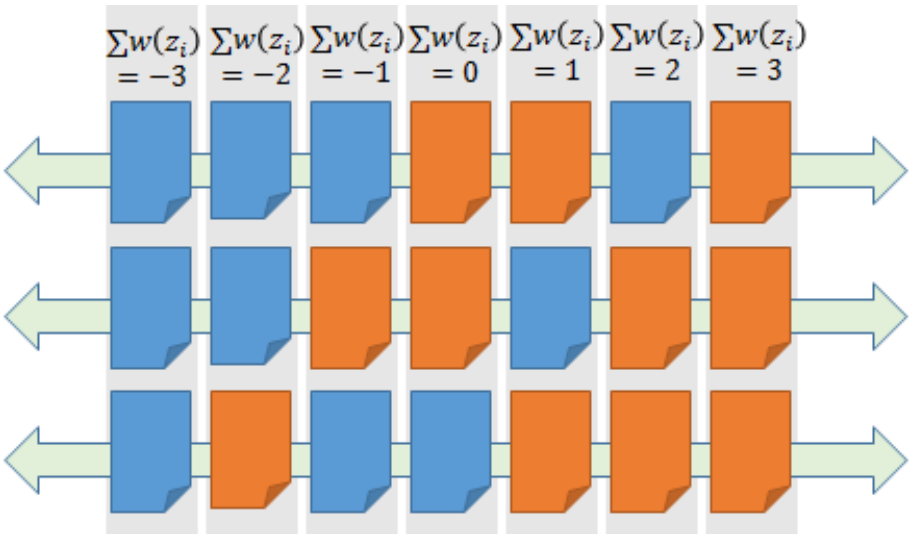


Figure 5.1: Example 5.1 with Fixed Word Weight

Intuitively, to maximize the log-likelihood, we need the sigmoid function $(1+\exp(-a_Y w_Y(X)-b_Y))^{-1}$ to match the conditional probability of Y conditioned on $w(X)$: $\mathcal{P}(Y|w_Y(X))$. Therefore, for the optimal a_Y and b_Y, we could say that the following property is roughly correct:

$$\mathcal{P}(Y|w_Y(X)) \approx \frac{1}{1+\exp(-a_Y w_Y(X) - b_Y)} \tag{5.6}$$

In other words,

$$\forall 0 \leq p \leq 1, \mathbb{E}[\mathcal{P}(Y|X)|f(X,Y) = p] \approx p \tag{5.7}$$

Let us examine this example more closely. The reason why the logistic regression classifier tells us that $f(X,Y) \approx p$ is because of the following: among all the documents with similar weight $w_Y(X)$, about p portion of them actually belong to the topic in the training dataset. This leads to an important observation: logistic regression classifiers estimate the conditional probabilities by computing the relative frequency of labels among documents it believes to be similar. Furthermore, this behavior is not unique to logistic regression. Many other algorithms, including decision tree classifiers, nearest neighbor (NN) classifiers, and neural networks, exhibit similar behavior:

- In decision trees, all the data points reaching the same decision leaf are considered similar.
- In NN classifiers, all the data points with the same nearest neighbors are considered similar.
- In neural networks, all the data points reaching the same output layer values are considered similar.

We can abstract the above conditional probability estimators as the following two-step process:

1. Partition the feature space $\mathcal{X}$ into several regions.
2. Estimate the relative frequency of labels among all data points inside each region.

The definition of the calibration property follows easily from the above two-step process. We can argue that the classifier is approximately calibrated, if for each region S in the feature space $\mathcal{X}$, the output conditional probability of data points in S is close to the actual relative frequency of labels in S. The definition for the calibration property then follows from the fact that all data points inside each region have the same output conditional probabilities:

$$\forall p_1 < p_2, Y \in \mathcal{Y}, \quad \mathcal{P}(Y|p_1 < f(X,Y) \leq p_2) = \mathbb{E}_{X\sim\mathcal{P}}[f(X)|p_1 < f(X,Y) \leq p_2] \tag{5.8}$$

The Uniform Convergence Result

Let $\mathcal{G}$ be a collection of functions from $\mathcal{X} \times \mathcal{Y}$ to $[0,1]$, the Rademacher Complexity [80][1] of $\mathcal{G}$ with respect to D is defined as [81]:

$$R_D(\mathcal{G}) = \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}[\sup_{g\in\mathcal{G}} \sum_{i=1}^{n} \sigma_i g(X_i, Y_i)] \tag{5.9}$$

Then regarding the calibration property, we have the following result:

Theorem 5.1. *Let $\mathcal{F}$ be a set of conditional probability estimators (i.e., functions from $\mathcal{X}, \mathcal{Y}$ to $[0,1]$). Let $\mathcal{H}$ be the set of binary classifiers obtained by thresholding the output of functions in $\mathcal{F}$:*

$$\mathcal{H} = \{\mathbb{1}_{p_1<f(X,y)\leq p_2} : p_1, p_2 \in \mathbb{R}, y \in \mathcal{Y}, f \in \mathcal{F}\} \tag{5.10}$$

[1]Our definition of Rademacher Complexity comes from Shalev-Shwartz and Ben-David's textbook [81], which is slightly different from the original definition in Bartlett and Mendelson's paper [80].

Suppose the Rademacher Complexity of $\mathcal{H}$ satisfies:

$$2\mathbb{E}_D R_D(\mathcal{H}) + \sqrt{\frac{2\ln(8/\delta)}{n}} < \frac{\epsilon}{2} \tag{5.11}$$

Then,

$$\mathbf{Pr}_D(\sup_{f\in\mathcal{F}} |c_{\mathcal{P}}(f) - c_D(f)| > \epsilon) < \delta \tag{5.12}$$

Proof. We will use the following uniform convergence result [81]:

Theorem 5.2 (Uniform Convergence of Functions [81])**.** *Let D be i.i.d. samples of $(\mathcal{X} \times \mathcal{Y}, \mathcal{P})$, then with probability at least $1-\delta$,*

$$\sup_{g\in\mathcal{G}} |\frac{1}{n}\sum_{i=1}^{n} g(X_i, Y_i) - \mathbb{E}g(X, Y)| \leq 2\mathbb{E}_D R_D(\mathcal{G}) + \sqrt{\frac{2\ln(4/\delta)}{n}} \tag{5.13}$$

In the following we sometimes allow $\mathcal{G}$ to be a collection of functions from $\mathcal{X}$ to $[0,1]$ in the above results. When used in this sense, we assume that the function will not use y label: $g(x,y) = g(x)$.

Define $\mathcal{F}_{D,p_1,p_2,y}(f)$ to be the relative frequency of event $\{p_1 < f(X,y) \leq p_2, Y = y\}$:

$$\mathcal{F}_{D,p_1,p_2,y}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{p_1<f(X_i)\leq p_2, Y_i=y} \tag{5.14}$$

Define $\mathcal{F}_{\mathcal{P},p_1,p_2,y}(f)$ to be the probability of the same event:

$$\mathcal{F}_{\mathcal{P},p_1,p_2,y}(f) = \mathcal{P}(p_1 < f(X,y) \leq p_2, Y = y) \tag{5.15}$$

Define $\mathcal{E}_{D,p_1,p_2,y}(f)$ as the empirical expectation of $f(X,y)\mathbb{1}_{p_1<f(X,y)\leq p_2}$:

$$\mathcal{E}_{D,p_1,p_2,y}(f) = \frac{1}{n}\sum_{i=1}^{n} f(X_i, y)\mathbb{1}_{p_1<f(X_i,y)\leq p_2} \tag{5.16}$$

Define $\mathcal{E}_{\mathcal{P},p_1,p_2,y}(f)$ as the expectation of the same function:

$$\mathcal{E}_{\mathcal{P},p_1,p_2,y}(f) = \mathbb{E}[f(X,y)\mathbb{1}_{p_1<f(X,y)\leq p_2}] \tag{5.17}$$

When the context is clear, subscripts p_1, p_2 and y can be dropped. Using this notation,

we can rewrite $c_{\mathcal{P}}(f)$ and $c_D(f)$ as follows:

$$c_{\mathcal{P}}(f) = \sup_{p_1,p_2,y} |\mathcal{F}_{\mathcal{P}}(f) - \mathcal{E}_{\mathcal{P}}(f)| \qquad c_D(f) = \sup_{p_1,p_2,y} |\mathcal{F}_D(f) - \mathcal{E}_D(f)| \tag{5.18}$$

Note that:

$$\begin{aligned}
&|\sup_{p_1,p_2,y} |\mathcal{F}_D(f) - \mathcal{E}_D(f)| - \sup_{p_1,p_2,y} |\mathcal{F}_S(f) - \mathcal{E}_S(f)|| \\
&\leq \sup_{p_1,p_2,y} ||\mathcal{F}_D(f) - \mathcal{E}_D(f)| - |\mathcal{F}_S(f) - \mathcal{E}_S(f)|| \\
&\leq \sup_{p_1,p_2,y} |\mathcal{F}_D(f) - \mathcal{E}_D(f) - \mathcal{F}_S(f) + \mathcal{E}_S(f)| \\
&\leq \sup_{p_1,p_2,y} (|\mathcal{F}_D(f) - \mathcal{F}_S(f)| + |\mathcal{E}_D(f) - \mathcal{E}_S(f)|) \\
&\leq \sup_{p_1,p_2,y} |\mathcal{F}_D(f) - \mathcal{F}_S(f)| + \sup_{p_1,p_2,y} |\mathcal{E}_D(f) - \mathcal{E}_S(f)|
\end{aligned} \tag{5.19}$$

Therefore it suffices to show that

$$\mathbf{P}(\sup_{f,p_1,p_2,y} |\mathcal{F}_D(f) - \mathcal{F}_S(f)| + \sup_{f,p_1,p_2,y} |\mathcal{E}_D(f) - \mathcal{E}_S(f)| > \epsilon) < \delta \tag{5.20}$$

Define

$$\mathcal{H}_1 = \{\mathbb{1}_{p_1<f(X,y)\leq p_2, Y=y} : p_1, p_2 \in \mathbb{R}, y \in \mathcal{Y}, f \in \mathcal{F}\} \tag{5.21}$$

$$\mathcal{H}_2 = \{f(X,y)\mathbb{1}_{p_1<f(X,y)\leq p_2} : p_1, p_2 \in \mathbb{R}, y \in \mathcal{Y}, f \in \mathcal{F}\} \tag{5.22}$$

Then we have the following lemma:

Lemma 5.1. *Let $\mathcal{H}_1, \mathcal{H}_2$ as defined above, then:*

$$R_D(\mathcal{H}_1) \leq R_D(\mathcal{H}) \quad R_D(\mathcal{H}_2) \leq R_D(\mathcal{H}) \tag{5.23}$$

Proof. For $R_D(\mathcal{H}_1)$, we have:

$$R_D(\mathcal{H}_1) = \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}[\sup_{p_1,p_2,y,f} \sum_{i=1}^{n} \sigma_i \mathbb{1}_{p_1<f(X_i,y)\leq p_2, Y_i=y}] \tag{5.24}$$

We can replace $\mathbb{1}_{Y_i=y}$ with $\mathbb{E}_{Z_i\in\{\pm1\}} \max(Z_i, 2\mathbb{1}_{Y_i=y} - 1)$:

$$R_D(\mathcal{H}_1) = \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}[\sup_{p_1,p_2,y,f} \sum_{i=1}^{n} \sigma_i \mathbb{1}_{p_1<f(X_i,y)\leq p_2} \mathbb{E}_{Z_i\in\{\pm1\}} \max(Z_i, 2\mathbb{1}_{Y_i=y} - 1)] \tag{5.25}$$

Move the expectation over Z out of the supremum operator, we have:

$$R_D(\mathcal{H}_1) \leq \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n, Z\in\{\pm1\}}[\sup_{p_1,p_2,y,f}\sum_{i=1}^{n}\mathbb{1}_{p_1<f(X_i)\leq p_2}\sigma_i\max(Z_i, 2\mathbb{1}_{Y_i=y}-1)] \tag{5.26}$$

Now define $T_i = \sigma_i\max(Z_i, 2\mathbb{1}_{Y_i=y}-1)$, then

$$R_D(\mathcal{H}_1) \leq \frac{1}{n}\mathbb{E}_{Z\in\{\pm1\}}\mathbb{E}_{\sigma\sim\{\pm1\}^n}[\sup_{p_1,p_2,y,f}\sum_{i=1}^{n}\mathbb{1}_{p_1<f(X_i)\leq p_2}T_i] \tag{5.27}$$

Note that T_i is always uniformly distributed over $\{\pm1\}$, which is independent of Z_i and Y_i. Therefore,

$$R_D(\mathcal{H}_1) \leq \frac{1}{n}\mathbb{E}_{T\sim\{\pm1\}^n}[\sup_{p_1,p_2,y,f}\sum_{i=1}^{n}\mathbb{1}_{p_1<f(X_i,y)\leq p_2}T_i] = R_D(\mathcal{H}) \tag{5.28}$$

For $R_D(\mathcal{H}_2)$, we have:

$$R_D(\mathcal{H}_2) = \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}[\sup_{p_1,p_2,y,f}\sum_{i=1}^{n}\sigma_i f(X_i,y)\mathbb{1}_{p_1<f(X_i,y)\leq p_2}] \tag{5.29}$$

Replace $f(X_i,y)$ with $\int_0^1\mathbb{1}_{t<f(X_i,y)}dt$, we have

$$R_D(\mathcal{H}_2) = \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}[\sup_{p_1,p_2,y,f}\int_0^1\sum_{i=1}^{n}\sigma_i\mathbb{1}_{t<f(X_i,y)}\mathbb{1}_{p_1<f(X_i,y)\leq p_2}dt] \tag{5.30}$$

Move the integral out of the supremum operator, we have:

$$R_D(\mathcal{H}_2) \leq \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}\int_0^1[\sup_{p_1,p_2,y,f}\sum_{i=1}^{n}\sigma_i\mathbb{1}_{\max(p_1,t)<f(X_i,y)\leq p_2}]dt \tag{5.31}$$

Define $p_1' = \max(t,p_1)$, then we have:

$$R_D(\mathcal{H}_2) \leq \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}\int_0^1[\sup_{p_1'\geq t,p_2,y,f}\sum_{i=1}^{n}\sigma_i\mathbb{1}_{p_1'<f(X_i,y)\leq p_2}]dt \tag{5.32}$$

Remove the restriction over p_1':

$$R_D(\mathcal{H}_2) \leq \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm1\}^n}\int_0^1[\sup_{p_1',p_2,y,f}\sum_{i=1}^{n}\sigma_i\mathbb{1}_{p_1'<f(X_i,y)\leq p_2}]dt \tag{5.33}$$

Now the expression inside the bracket no longer depends on the value of t, therefore we

conclude:

$$R_D(\mathcal{H}_2) \leq \frac{1}{n}\mathbb{E}_{\sigma\sim\{\pm 1\}^n}[\sup_{p_1',p_2,y,f}\sum_{i=1}^{n}\sigma_i \mathbb{1}_{p_1'<f(X_i,y)\leq p_2}] = R_D(\mathcal{H}) \tag{5.34}$$

Combining this lemma with the assumptions in the theorem:

$$\mathbb{E}_D R_D(\mathcal{H}_1) + \sqrt{\frac{2\ln(8/\delta)}{n}} < \frac{\epsilon}{2} \qquad \mathbb{E}_D R_D(\mathcal{H}_2) + \sqrt{\frac{2\ln(8/\delta)}{n}} < \frac{\epsilon}{2} \tag{5.35}$$

By Equation (5.13):

$$\mathbf{P}(\sup_{f,y,p_1,p_2}|\mathcal{F}_D(f) - \mathcal{F}_S(f)| > \frac{\epsilon}{2}) < \frac{\delta}{2} \qquad \mathbf{P}(\sup_{f,y,p_1,p_2}|\mathcal{E}_D(f) - \mathcal{E}_S(f)| > \frac{\epsilon}{2}) < \frac{\delta}{2} \tag{5.36}$$

In Theorem 5.1, $\mathcal{H}$ is the collection of binary classifiers obtained by thresholding the output of condtional probability estimators in $\mathcal{F}$. For many hypothesis classes $\mathcal{F}$, the Rademacher Complexity of $\mathcal{H}$ can be naturally bounded. For instance, if $\mathcal{F}$ is the d-dimensional generalized linear classifiers with monotone link function, then $\mathbb{E}_D R_D(\mathcal{H})$ can be bounded by $O(\sqrt{d\log n/n})$.

Verifying the Calibration Property

The first application of Theorem 5.1 is that we can verify whether the learned classifier f is calibrated. For simple hypothesis classes $\mathcal{F}$ (e.g., logistic regression), the corresponding hypothesis space $\mathcal{H}$ has low Rademacher Complexity. In this case, Theorem 5.1 naturally guarantees the generalization of calibration measure.

There are also cases where the Rademacher Complexity of $\mathcal{H}$ is not small. One notable example is SVM classifiers with Platt Scaling [77]. In the case of SVM, the dimensionality of the feature space is usually much larger than the training dataset size (this is especially true for kernel SVM). In such situation, we can no longer verify the calibration property using only the training data, and a separate validation dataset is needed to calibrate the classifier (as suggested by Platt [77]). When verifying the calibration of a classifier on a validation dataset, we have the following result:

Claim 5.1. *Let f be any conditional probability estimator, and D be a* ***validation*** *dataset*

consisting of i.i.d. samples from $\mathcal{P}$ (i.e., D is not used when training f), then:

$$\mathbf{Pr}(c_{\mathcal{P}}(f) \leq c_D(f) + \sqrt{\frac{\ln \frac{2}{\delta}}{2n}}) \geq 1 - \delta \tag{5.37}$$

$$\mathbf{Pr}(c_{\mathcal{P}}(f) \geq c_D(f) - [16\sqrt{2\pi} \cdot |\mathcal{Y}| + 2\sqrt{2 \ln \frac{8}{\delta}}]\sqrt{\frac{1}{n}}) \geq 1 - \delta \tag{5.38}$$

Proof. We first prove the first inequality. Since D consists of i.i.d. samples from $\mathcal{P}$, for any p_1, p_2, y, we have the following:

$$\begin{aligned} &\mathbb{E}_{X \sim \mathcal{P}}[\mathbb{1}_{p_1 < f(X,y) \leq p_2} f(X,y)] - \mathcal{P}(p_1 < f(X,y) \leq p_2, Y = y) \\ =&\mathbb{E}_D \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{p_1 < f(X_i,y) \leq p_2}[f(X_i,y) - \mathbb{1}_{Y_i = y}] \end{aligned} \tag{5.39}$$

Therefore, by Hoeffdding's inequality, with probability $1 - \delta$:

$$\begin{aligned} &|\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{p_1 < f(X_i,y) \leq p_2}[f(X_i,y) - \mathbb{1}_{Y_i = y}] \\ &-\mathbb{E}_{X \sim \mathcal{P}}[\mathbb{1}_{p_1 < f(X,y) \leq p_2} f(X,y)] + \mathcal{P}(p_1 < f(X,y) \leq p_2, Y = y)| \leq \sqrt{\frac{\ln \frac{2}{\delta}}{2n}} \end{aligned} \tag{5.40}$$

Thus with probability $1 - \delta$,

$$\begin{aligned} &|\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{p_1 < f(X_i,y) \leq p_2}[f(X_i,y) - \mathbb{1}_{Y_i = y}]| \\ \geq &|\mathbb{E}_{X \sim \mathcal{P}}[\mathbb{1}_{p_1 < f(X,y) \leq p_2} f(X,y)] - \mathcal{P}(p_1 < f(X,y) \leq p_2, Y = y)| - \sqrt{\frac{\ln \frac{2}{\delta}}{2n}} \end{aligned} \tag{5.41}$$

For any $\epsilon > 0$, we can choose p_1, p_2 and y such that

$$|\mathbb{E}_{X \sim \mathcal{P}}[\mathbb{1}_{p_1 < f(X,y) \leq p_2} f(X,y)] - \mathcal{P}(p_1 < f(X,y) \leq p_2, Y = y)| > c_{\mathcal{P}}(f) - \epsilon \tag{5.42}$$

Then with probability $1 - \delta$,

$$c_D(f) \geq |\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{p_1 < f(X_i,y) \leq p_2}[f(X_i,y) - \mathbb{1}_{Y_i = y}]| > c_{\mathcal{P}}(f) - \epsilon - \sqrt{\frac{\ln \frac{2}{\delta}}{2n}} \tag{5.43}$$

Since ϵ can be any positive real number, the desired result follows immediately.

To prove the second inequality, by Theorem 5.1, it suffices to show that

$$\forall D, R_D(\mathcal{H}) \leq \sqrt{\frac{32\pi}{n}}|\mathcal{Y}| \tag{5.44}$$

for $\mathcal{F} = \{f\}$. Now for each $y \in \mathcal{Y}$, define the $\rho_y(i)$ to be the permutation of $\{1, \ldots, n\}$ satisfying $f(x_{\rho_y(1)}, y) \leq f(x_{\rho_y(2)}, y) \leq \ldots \leq f(x_{\rho_y(n)}, y)$. Then we have,

$$R_D(\mathcal{H}) = \frac{1}{n}\mathbb{E}_\sigma \sup_{p_1,p_2,y} |\sum_{i=1}^{n} \mathbb{1}_{p_1 < f(x_i,y) \leq p_2}\sigma_i| \leq \frac{1}{n}\mathbb{E}_\sigma \max_{y,a,b} |\sum_{a<i\leq b} \sigma_{\rho_y(i)}| \tag{5.45}$$

Denote $S_y(i) = \sum_{j\leq i} \sigma_{\rho_y(j)}$, then S_y is a simple one-dimensional random walk, by the reflection principle of symmetric random walk [82], we have:

$$\forall C \geq 0, \mathbf{Pr}(\sup_i |S_y(i)| > C) \leq 2\mathbf{Pr}(|S_y(n)| > C) \tag{5.46}$$

Therefore,

$$\mathbb{E}_\sigma[\sup_{i,j,y} |S_y(i) - S_y(j)|] \leq 2\mathbb{E}_\sigma[\sup_{i,y} |S_y(i)|] \leq 4\mathbb{E}_\sigma \sup_y |S_y(n)| \tag{5.47}$$

By Hoeffding's inequality,

$$\forall C \geq 0, y \in \mathcal{Y}, \qquad \mathbf{Pr}(|S_y(n)| \geq C\sqrt{n}) \leq 2\exp(-\frac{1}{2}C^2) \tag{5.48}$$

Therefore,

$$\forall C \geq 0, \qquad \mathbf{Pr}(\sup_y |S_y(n)| \geq C\sqrt{n}) \leq 2\exp(-\frac{1}{2}C^2)|\mathcal{Y}| \tag{5.49}$$

Thus,

$$\begin{aligned} R_D(\mathcal{H}) \leq \frac{4}{n}\mathbb{E}_\sigma \sup_y |S_y(n)| =& \sqrt{\frac{16}{n}} \int_0^\infty \mathbf{Pr}(|S_n| \geq x\sqrt{n})dx \\ \leq& \sqrt{\frac{64}{n}}|\mathcal{Y}| \int_0^\infty e^{-\frac{1}{2}x^2}dx = \sqrt{\frac{32\pi}{n}}|\mathcal{Y}| \end{aligned} \tag{5.50}$$

Obtaining Calibrated Conditional Probabilities

Suppose that for each label $y \in \mathcal{Y}$, we are given an uncalibrated conditional probability estimator $f_y : \mathcal{X} \to [0, 1]$, and we want to find a function $g_y : [0, 1] \to [0, 1]$, so that $g_y \circ f_y$ presents a better conditional probability estimation. This is the problem of classifier calibration, which has been studied in many papers [83, 77].

Traditionally, calibration algorithms find the best link function g_y by maximizing likelihood or minimizing squared loss. In this chapter, we suggest a different approach to the calibration problem. We can find the best g_y by minimizing the calibration measure[2] $c_{D,y}(g_y \circ f_y)$ with respect to the validation dataset. Let us assume w.l.o.g. that the validation dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ satisfies $f_y(x_1) \leq \ldots \leq f_y(x_n)$ and that g_y is monotonically nondecreasing. Then we have,

$$\begin{aligned} c_{D,y}(g_y \circ f_y, D) =& \frac{1}{n} \sup_{p_1,p_2} |\sum_{i=1}^{n} \mathbb{1}_{p_1<g_y(f_y(x_i))\leq p_2}(\mathbb{1}_{y_i=y} - g_y(f_y(x_i)))| \\ \leq& \frac{1}{n} \max_{a,b} |\sum_{a<i\leq b} (\mathbb{1}_{y_i=y} - g_y(f_y(x_i)))| \end{aligned} \tag{5.51}$$

This expression can be used as the objective function for calibration: we search over the space of hypothesis $\mathcal{G}$ to find a function g_y that minimizes this objective function. Compared to other loss functions, the benefits of minimizing this objective function is that the resulting classifier is more likely to be calibrated, and therefore provides more interpretable conditional probability estimates. In fact, one of the most well-known calibration algorithms, the isotonic regression algorithm (Algorithm 5.1), can be viewed as minimizing this objective function (proof can be found in [6]):

Claim 5.2. *Let $\mathcal{G}$ be the set of all continuous nondecreasing functions from $[0, 1]$ to $[0, 1]$. Then the optimal solution found by the isotonic regression algorithm (Algorithm 5.1) not only minimizes the squared loss*

$$\mathcal{L}_2(g_y) = \sum_{i=1}^{n} (\mathbb{1}_{y_i=y} - g_y(f_y(x_i)))^2 \tag{5.52}$$

as shown in [84], but also minimizes

$$\mathcal{L}_c(g_y) = \max_{a,b} |\sum_{a<i\leq b} (\mathbb{1}_{y_i=y} - g_y(f_y(x_i)))| \tag{5.53}$$

Usefulness of the Calibration Property

Although we have shown that it is possible to obtain calibrated conditional probabilities as assessments for confidence levels, it is not immediately clear why calibrated conditional probabilities would be more useful to users than uncalibrated ones in practice. To demonstrate

[2]Although c_D is originally defined w.r.t. all labels, it is also possible to define the calibration w.r.t. specific label y, we omit the detailed definition here for simplicity.

Algorithm 5.1 Isotonic Regression Calibration (PAV Algorithm) [84]

1. Order the data points so that $f_y(x_1) \leq f_y(x_2) \leq \ldots \leq f_y(x_n)$
2. For $i = 0, \ldots, n$, Compute $P_i = (i, S_i = \sum_{j \leq i} \mathbb{1}_{y_j = y})$
3. Let $cv(P)$ be the lower boundary of the convex hull of the set of points P_i
   Remark: Implementing this step using the Graham's algorithm [85] would result in the exact same algorithmic procedure as in [84].
4. For $i = 0, \ldots, n$, Let $Z_i =$ intersection of $cv(P)$ and the line $x = i$
5. Compute $z_i = Z_i - Z_{i-1}$
6. Let $g_y(f_y(x_i)) = z_i$, extrapolate these points to get continuous nondecreasing function g_y.

the usefulness of calibration property, let us consider the following example scenario:

Example 5.2. *Suppose that we have a list of residents and a product, the prediction algorithm (trained using the history of purchase activities) has predicted the calibrated conditional probabilities of each resident buying our product (if we dispatch a salesperson). Obviously, the strategy to maximize our profit while minimizing the cost is to send the salesperson only to residents with relatively high probability of buying.*

Now the question is: *how to decide the optimal threshold probability value for whether to send the salesperson or not?* Without the calibration property, it can be hard to answer such question. However, with the calibration property, the answer is simple: we should only send the salesperson to residents with the purchasing probability greater than p, where p is defined as:

$$p = \frac{\text{the cost of sending the salesperson}}{\text{the profit of each sale}} \tag{5.54}$$

To see this, note that since the conditional probabilities are calibrated, we can reliably estimate the expected number of people actually buying our product once we send the salesperson (denoted as S_p) using the following formula:

$$S_p = \sum_{r \in \text{residents}} \mathbf{Pr}(\text{buying}|r) \mathbb{1}_{\mathbf{Pr}(\text{buying}|r) > p} \tag{5.55}$$

and the calibration property ensures that the real value is close to S_p (see Definition 5.1). From this, it is easy to see that our previously mentioned strategy would maximize the expected net profit (i.e. profits from sales minus cost of salesperson).

Remark: Currently, our calibration property is defined with respect to each individual target label y_0 (i.e., the conditional probabilities are essentially calibrated for each y_0 separately). Although it is possible to define more restricted version of the *total calibration* property (i.e., designating probability intervals for all target labels simutaneously), it becomes unclear as how to actually guarantee such a total calibration property in practice. We hope that future papers in this direction would give us a more complete answer.

5.2 APPROACHES FOR RELATIONAL LEARNING

Now that we have discussed the calibration property of conditional probabilities and methods for obtaining it, we can move on to discuss the actual prediction algorithms for relational datasets. As our goal is to develop a hidden structure based prediction algorithm that works for arbitrary relational datasets, a good starting point is to review the existing specialized approaches for specific relational learning tasks, and understand the high-level philosophy behind those approaches, so that we can generalize them into algorithms that work for arbitrary relational datasets.

To the best of our knowledge, there are primarily three distinct lines of approaches for relational learning tasks:

- **Join-based Learning and Prediction.** The first, and probably most commonly used approach for relational learning is to reduce it to a standard machine learning task with independent training data points. Such a reduction is usually achieved via foreign-key join operations, with the goal of generating one single large table to be used as input for standard machine learning techniques (e.g., SVMs, Decision Trees, etc.) [76].

- **Probabilistic Models.** Many researchers have also been developing generative probabilistic models for large-scale relational datasets to capture the probabilistic dependencies between attributes in the dataset. These techniques are mostly based on possible world semantics [86], and the most notable one involves Markov Logic Networks [87]. These probabilistic models naturally induce conditional probabilities for each missing entry, although exact inference is NP-hard [12], necessitating approximations [88].

- **Representation Learning.** Popularized by the seminal work of word2vec [89], the idea of learning a vector space representation for entities in relational datasets is at the core of many recent relational learning techniques. These methods are mostly trying to optimize a certain objective function that revolves around using vector representations

of entities to predict observed information in the dataset, and the learned representations [90] are then used as features for other prediction tasks (via standard machine learning techniques such as SVMs and neural networks).

In the following, we show the three different methods that we designed by generalizing the above approaches to handle relational datasets with arbitrary schema.

5.2.1 Feature Inheritance

The most straightforward approach for predicting missing entries in relational datasets is to reduce this task into a standard machine learning task, which is typically achieved by joining information across relations via foreign-key references. Feature Inheritance (FI) is an algorithmic procedure that we developed for joining information across relations in relational datasets with arbitrary schema. After the FI procedure, each tuple will "inherit" features from other tuples that are connected to it via foreign-key references. Let us first demonstrate the general idea of feature inheritance via a simple example:

Example 5.3 (Targeted Advertisement with FI)**.** *Consider the e-commerce dataset in Figure 5.2, which consists of three different tables containing information about advertisements, users, and historical click-throughs respectively. As we can see, the joined click-through table (right hand side of Figure 5.2) "inherits" all relevant features from the user and advertisement tuples in the base tables. Hence, it is now possible to use standard machine learning methods on this table to predict click-through probabilities.*

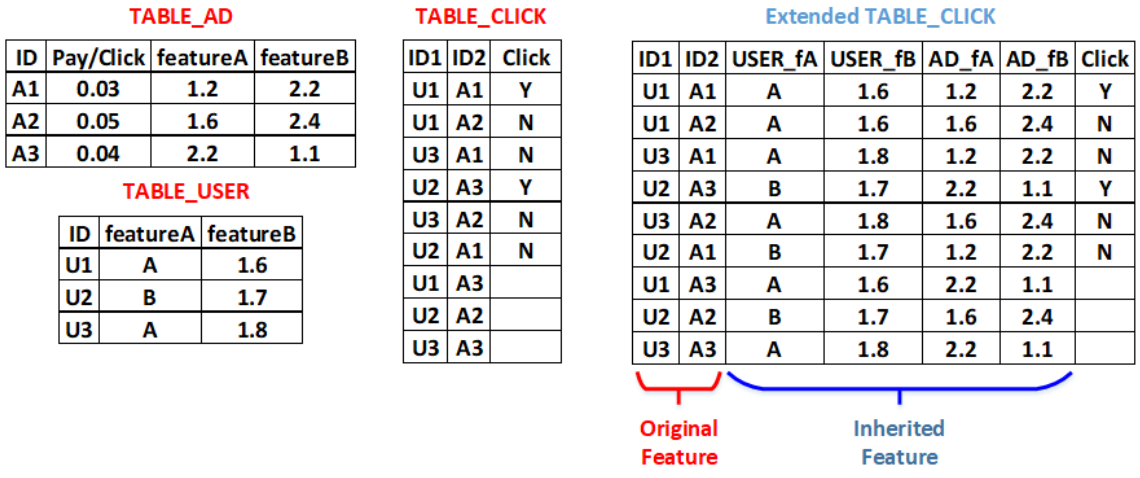


Figure 5.2: Targeted Advertisement (Feature Inheritance)

More generally, suppose the database contains k relations $\mathcal{R} = \{R_1, \ldots, R_k\}$, in which each relation R_i contains n_i tuples $R_i = \{t_{i,1}, \ldots, t_{i,n_i}\}$. There are also m different foreign-key attributes in the database: $\mathcal{F} = \{f_1, \ldots, f_m\}$, and each foreign-key attribute $f_i : R_{a_i} \to R_{b_i}$ is defined as a mapping function between two relations[3]. Under such a notation, the FI procedure uses the following steps to decide the exact feature inheritance setup:

1. For each relation R_{a_0}, consider all possible join-paths starting from R_{a_0} that has length $l \leq L$:

$$\mathcal{P}_{a_0,l} = \{R_{a_0} \xrightarrow{g_1} R_{a_1} \xrightarrow{g_2} \ldots \xrightarrow{g_l} R_{a_l} : g_1, \ldots, g_l \in \mathcal{F}\} \tag{5.56}$$

   where $g_i \in \mathcal{F} : R_{a_{i-1}} \to R_{a_i}$ denotes the foreign-key attribute of $R_{a_{i-1}}$ referencing tuples of R_{a_i}.

2. For each such join-path $p = (g_1, \ldots, g_l) \in \mathcal{P}_{a_0,l}$, we augment R_{a_0} with all the (base) attributes of R_{a_l}: for each attribute $\mathcal{A} : R_{a_l} \to \mathcal{Y}$ of R_{a_l}, the following new attribute $\mathcal{A}_p : \mathcal{R}_{a_0} \to \mathcal{Y}$ is added to R_{a_0}:

$$\forall t \in R_{a_0}, \mathcal{A}_p(t) = \mathcal{A} \circ g_l \circ \ldots \circ g_1(t) \tag{5.57}$$

   In other words, for each $t \in R_{a_0}$, let $t_l \in R_{a_l}$ be the corresponding tuple that t is referencing through the join path p, then the new attribute of t has value equals the corresponding attribute value of t_l.

After running the FI procedure, each relation in $\mathcal{R}$ will be augmented with features from other relations. Thus, it is now possible to apply standard machine learning algorithms on the augmented relations to obtain calibrated conditional probability estimates regarding the attributes of interest. The upper limit L for the length of join-path controls the trade-off between efficiency and accuracy: larger value of L can lead to situations where too many features are joined together, reducing the efficiency of learning procedure; smaller value on the other hand can miss important correlations occurring between far-away attributes. The value of this hyper-parameter can be either specified by user or tuned automatically via cross-validation.

5.2.2 Calibrated Markov Logic Network

The second approach utilizes existing generative probabilistic models for relational datasets: although these models naturally output marginal probabilities for the missing entries, these

[3]This is consistent with the standard definition of attributes throughout this chapter, in which each attribute $\mathcal{A} : \{t_1, \ldots, t_n\} \to \mathcal{Y}$ is defined as a mapping function from the set of tuples to the target domain.

marginal probabilities do not naturally have the calibration property. However, it is possible to convert them into calibrated CPEs using a calibration step. Here, we use the Markov Logic Network [87] model as our probabilistic model for relational datasets. The detailed steps of this method are described in the following:

1. Construct a parametric generative probabilistic model $\mathcal{P}(D|\Theta)$ (e.g., a Markov Logic Network) for the given relational dataset D. This probabilistic model describes the joint distribution of all attributes in the dataset (regardless of whether they are observed or missing), and the parameter Θ controls the likelihood of each particular joint instantiation of attribute values.

2. Divide the observed attribute values of D into two separate datasets: the training dataset D_{train} and the validation dataset D_{val}. Search for the parameter Θ_{MLE} that maximizes the likelihood[4] of observing D_{train} given Θ:

$$\Theta_{MLE} = \arg\max_{\Theta} = \mathcal{P}(D_{train}, \mathbf{A}|\Theta) \tag{5.58}$$

3. Compute the marginal distribution of all attributes with missing values in D_{train}, conditioned on the surrounding context and the estimated parameter Θ_{MLE}.

$$\forall y \in \mathcal{Y}_i, f(i, y) = \mathcal{P}(A_i = y|D_{train}, \Theta_{MLE}) \tag{5.59}$$

4. Apply a calibration algorithm on the marginal probabilities $f(i, y)$, and use observed attribute values in D_{val} as labels. The calibration algorithm takes the uncalibrated probability estimates as input[5] and adjusts their values so that the calibration property is satisfied.

5.2.3 Latent Feature Inheritance

The third approach for computing calibrated CPEs is inspired by the recent developments in the field of representation learning: we try to learn vector-form representation for each individual tuple in the relational dataset. The embedding vectors are intended to preserve all structural and attribute information in the dataset, while having the benefits of being easier to be utilized by standard machine learning algorithms.

[4] In practice, optimizing exact likelihood is infeasible due to efficiency reasons, and approximations such as pseudo-likelihood are commonly used

[5] The ground truth label of a small subset of data instances is also required for the calibration algorithm to work.

Assuming that each tuple is associated with an embedding vector (a.k.a., the latent feature), the representation learning approach requires us to design an appropriate objective function, which should involve all the latent feature vectors, as well as the structure and attributes of dataset. In order to design an objective function that is applicable to all different kinds of database schema, we can utilize the feature inheritance (FI) technique, allowing each tuple to inherit other tuple's latent feature vector. Formally, the objective function can be designed as follows:

1. Initially, associate each tuple t with an latent feature vector v_t[6].

2. For each tuple t, find the collection of tuples $C(t)$ that t should inherit feature from (based on the FI technique in Section 5.2.1). Define the extended latent feature vector form of t as $ev_t = \{v_t\} \cup \{v_u : u \in C(t)\}$.

3. For each attribute A_i, we associate it with a prediction function $f_i(ev_t, \theta_i)$ which takes an extended latent feature vector ev_t as input (θ_i is the parameter to be optimized). We also need a loss function $l_i(A_i(t), f(ev_t, \theta_i))$, which compares the true attribute value $A_i(t)$ with the prediction $f(ev_t, \theta_i)$ and incur loss based on the difference.

4. The final objective function is the sum of all individual loss function components:

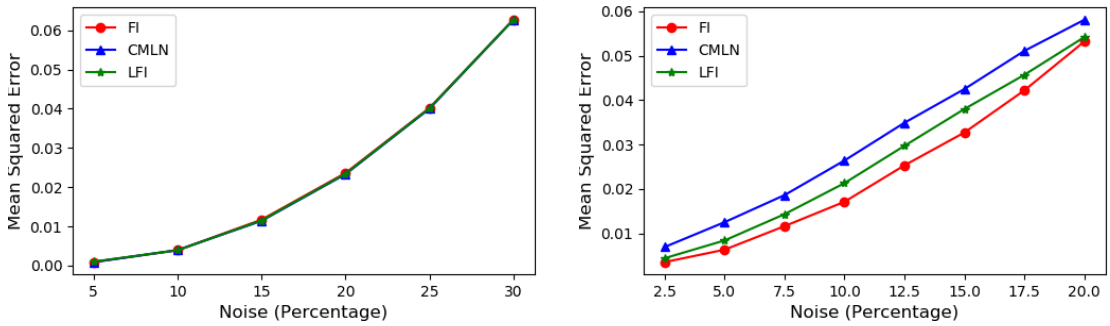
$$L(\Theta, V) = \sum_{A_i} \sum_{t} l(A_i(t), f(ev_t, \theta_i)) \tag{5.60}$$

Optimizing the final objective function $L(\Theta, V)$ should lead us to meaningful vector representations for all tuples, as well as proper predictions for all the missing entries. The parametric prediction function design can be dataset dependent, but it is also possible to use universal classifiers (e.g., SVMs), which is the default choice in our implementation.

5.3 PRELIMINARY EXPERIMENTS

Here, we evaluate the above three methods on several synthetic datasets as well as one real-world dataset. These experiments should help us understand what kind of datasets each method works well with, and in what situations they won't give us reasonable predictions.

[6]The latent feature includes the explicit feature as part of it.



(a) Synthetic Bayes

(b) Synthetic Cluster

Figure 5.3: Single Table Synthetic Datasets

5.3.1 Synthetic Bayes Dataset

This dataset contains a single table with 6 binary attributes. The first 5 attributes are independent noisy version of the last one (with probability p being flipped), and we want to predict the last attribute given the others. In this experiment, the dataset contains 10,000 tuples, and 50% of them have the last attribute observed. Figure 5.3(a) shows the mean squared error of the three methods, where all three overlap entirely.

In this dataset, the performance of three methods are basically the same: all three methods are capable of capturing the logistic regression model as a special case, which is actually the optimal prediction method for this dataset. Simple linear classification methods are special cases of all three methods, and the optimization procedure can find the existence of such simple models using relatively few data points.

5.3.2 Synthetic Cluster Dataset

This dataset contains one single table with 11 binary attributes. The dataset is generated using the following procedure:

- We first generate 10 template tuples, for which each of the 11 binary attributes are generated from independent coin flips.
- Then, each tuple is generated from one of the 10 tuple templates (randomly chosen): the first 10 attributes are noisy version of the corresponding attribute in the template tuple; the last attribute (11th) is always equal to the corresponding (11th) attribute of the template tuple, and is also the prediction target of this dataset.

In this experiment, the dataset contains 10,000 tuples, and 50% of them have the last attribute observed. Figure 5.3(b) shows the mean squared error of the three methods. For

this dataset, the FI method performs the best since it essentially treats the prediction task as a standard machine learning task (i.e., no feature inheritance occurs in this dataset). On the other hand, the CMLN method performs the worst due to the fact that the logistic regression model mimicked by CMLN is not the optimal model for this dataset. The performance of the LFI method lies somewhere in between, which is due to fact that the vector space latent features behave like pure noise for this dataset.

5.3.3 Synthetic Graph Dataset

This dataset simulates a graph or network scenario with nodes and edges. It contains two relations: the first relation contains only 1 column, which specifies the node group information (i.e., which group does each node belongs to); the second relation has 2 columns, which specifies the edge information (i.e., two end points of the edges). Here we try to predict the group label of some nodes given others. Among all edges, p of them are noise edges (i.e., connecting random nodes), while the remaining ones only connect nodes with the same label. In this experiment, the dataset contains 5,000 nodes (divided into 5 groups) and $100,000$ edges, 50% of the nodes have their labels observed, and the labels of the other 50% nodes are prediction targets. Figure 5.4 shows the mean squared error of the LFI and CMLN methods. The performance of the FI method is not reported because it is not applicable to this dataset.

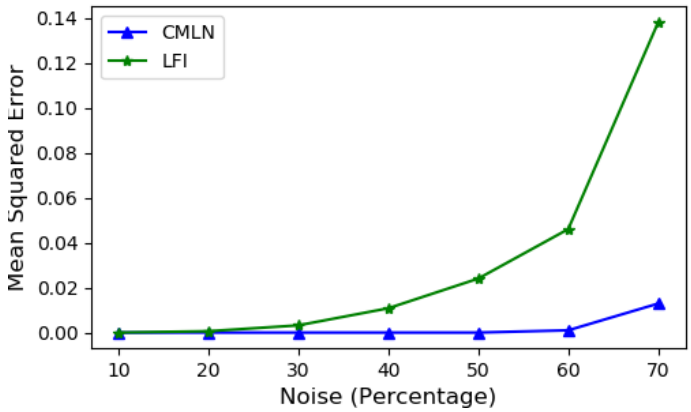


Figure 5.4: Synthetic Graph Dataset

As we can see, while LFI can perform reasonably well on this dataset with low noise level, CMLN performs much better due to the existence of human guidance. This experiment demonstrates the superiority of CMLN when the user is capable of providing precise and accurate first-order logical predictors. Although such a scenario is relatively rare in practice, it does exist in some cases.

5.3.4 StackExchange Dataset

This dataset comes from the StackExchange data dump [39], and we used part of the data collected from cs.stackexchange.com, which consists of 28,210 questions, 33,411 answers, 17,608 users and 461 tags[7]. The dataset schema is illustrated in Figure 5.5. There are a total of 65,781 pairs of question-tag associations, and the same number of negative associations are randomly generated. In the experiment, 50% of the question-tag associations have their labels observed, while the labels of the other 50% are the prediction targets.

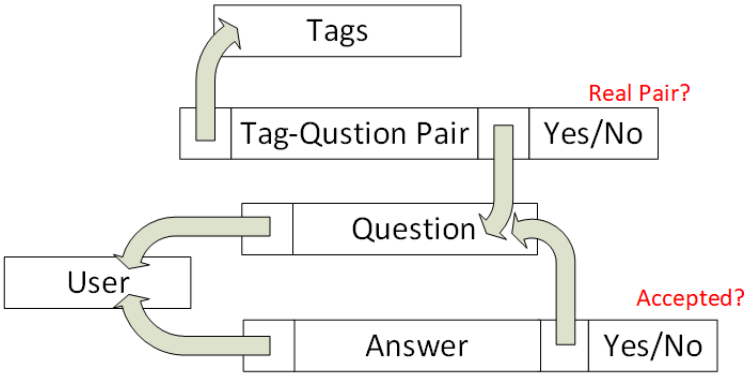


Figure 5.5: StackExchange Dataset Schema

The AUC(Area Under Curve) score[8] obtained by the FI and LFI methods are 0.806 and 0.631 respectively. The CMLN method do not scale well to this large dataset, so we are unable to report its performance.

5.4 DISCUSSION

From the experiments, we have seen that all three methods can perform well if given suitable datasets. Generally speaking, FI works best when the dataset is "flat", in the sense that predictions can be made without relying on the relationships between entities. CMLN performs well if user can provide precise first-order logic predictors, but scales badly to large datasets. LFI performs decently across the board, and excels on datasets with complex relationships (e.g., StackExchange dataset). From these experiments, it appears that it would be beneficial to combine all these approaches in some way, so that we can always achieve good performance regardless of the kind of input dataset we are given. We remark

[7]Tags with less than 50 associated questions are removed to reduce noise.

[8]AUC score is the standard evaluation metrics for ranking-based algorithms, which computes the total area under the ROC curve.

that the work presented in this chapter is still ongoing, and it would be interesting to see if it is possible to design an algorithm that combines the advantages of these three approaches.

CHAPTER 6: CONCLUSION

In this thesis, we introduced a general framework for utilizing the hidden structures of datasets to improve automated data processing. In our framework, the extracted structures are directly used for the subsequent data processing steps, and do not need manual verification or intervention beforehand. Due to the lack of human supervision, we usually need to consider different trade-offs when designing the structure extraction modules, compared to traditional scenarios where structure extraction is for the purpose of human consumption or data analysis. We summarized three major design principles for structure extraction in a data processing scenario: (a) the representativeness principle requires the structure hypothesis space to be able to cover most cases, since the structure extraction procedure must work for every potential dataset without any human supervision; (b) the learnability principle restricts the size of structure hypothesis space, as we must be able to efficiently find the appropriate structure candidate from this space; (c) the extracted structure must provide good utility for the subsequent data processing step, and thus the structure hypothesis space must be designed in conjunction with the data processing procedure. We also discussed three example algorithms in our recent and ongoing work, and connected the design decisions in practice with the general principles above.

Automatically using hidden structure to improve data processing represents the future trend of integration between artificial intelligence and data management. As such a framework does not require the presence of experienced data analysts "in the loop", it can be used by anyone with basic data management experience in a plug and play manner. The "intelligent" part is hidden from the user in our framework, and therefore the algorithm design for our framework generally requires a deeper understanding of the behavior of machine learning algorithms. Recent advances in the field of machine learning have made it possible to design generic structure extraction algorithms without pre-specifying a target dataset, and we can definitely hope for new developments following this trend in the future, so that data processing algorithms will become more and more "intelligent", approach human expertise, and lead to substantially reduced effort for end-users.

REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Acm sigmod record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[3] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 489–504.

[4] Y. Gao and A. Parameswaran, "Squish: Near-optimal compression for archival of relational datasets," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1575–1584.

[5] Y. Gao, S. Huang, and A. Parameswaran, "Navigating the data lake with datamaran: Automatically extracting structure from log datasets," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 943–958.

[6] Y. Gao, A. Parameswaran, and J. Peng, "On the interpretability of conditional probability estimates in the agnostic setting," in *Artificial Intelligence and Statistics*, 2017, pp. 1367–1374.

[7] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[8] F. M. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: basic properties," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, 1995.

[9] S. Babu, M. Garofalakis, and R. Rastogi, "Spartan: A model-based semantic compression system for massive data tables," in *ACM SIGMOD Record*, vol. 30, no. 2. ACM, 2001, pp. 283–294.

[10] H. Jagadish, R. T. Ng, B. C. Ooi, and A. K. Tung, "Itcompress: An iterative semantic compression algorithm," in *Data Engineering, 2004. Proceedings. 20th International Conference on*. IEEE, 2004, pp. 646–657.

[11] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil et al., "C-store: a column-oriented dbms," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 553–564.

[12] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[13] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[14] G. Schwarz et al., "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[15] M. Abramowitz and I. Stegun, "Handbook of mathematical functions: With formulas, graphs, and mathematical tables applied mathematics series," *National Bureau of Standards, Washington, DC*, 1964.

[16] G. G. Langdon, "An introduction to arithmetic coding," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[17] V. Raman and G. Swart, "How to wring a table dry: Entropy compression of relations and querying of compressed relations," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 858–869.

[18] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[19] S. M. Jalaleddine, C. G. Hutchens, R. D. Strattan, and W. A. Coberly, "Ecg data compression techniques-a unified approach," *IEEE transactions on Biomedical Engineering*, vol. 37, no. 4, pp. 329–343, 1990.

[20] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[21] J. Roure and R. Sangüesa, "Incremental methods for bayesian network learning," in *Department de*. Citeseer, 1999.

[22] J. R. Alcobé, "Incremental hill-climbing search applied to bayesian network structure learning," in *Proceedings of the 15th European conference on machine learning, Pisa, Italy*, 2004.

[23] M. A. Roth and S. J. Van Horn, "Database compression," *ACM Sigmod Record*, vol. 22, no. 3, pp. 31–39, 1993.

[24] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[25] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[26] S. Davies and A. Moore, "Bayesian networks for lossless dataset compression," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. Citeseer, 1999, pp. 387–391.

[27] R. Baeza-Yates, B. d. A. N. Ribeiro et al., *Modern information retrieval*. New York: ACM Press; Harlow, England: Addison-Wesley,, 2011.

[28] J. J. Rissanen, "Generalized kraft inequality and arithmetic coding," *IBM Journal of research and development*, vol. 20, no. 3, pp. 198–203, 1976.

[29] N. Kushmerick, D. S. Weld, and R. Doorenbos, "Wrapper induction for information extraction," 1997.

[30] D. Freitag and N. Kushmerick, "Boosted wrapper induction," in *AAAI/IAAI*, 2000, pp. 577–583.

[31] H. Elmeleegy, J. Madhavan, and A. Halevy, "Harvesting relational tables from lists on the web," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1078–1089, 2009.

[32] J. Antunes, N. Neves, and P. Verissimo, "Reverse engineering of protocols from network traces," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 2011, pp. 169–178.

[33] S. Gulwani, W. R. Harris, and R. Singh, "Spreadsheet data manipulation using examples," *Communications of the ACM*, vol. 55, no. 8, pp. 97–105, 2012.

[34] K. Fisher, D. Walker, K. Q. Zhu, and P. White, "From dirt to shovels: fully automatic tool generation from ad hoc data," in *ACM SIGPLAN Notices*, vol. 43, no. 1. ACM, 2008, pp. 421–434.

[35] "Recordbreaker: Automatic structure for your text-formatted data," http://cloudera.github.io/RecordBreaker/.

[36] M. Sipser, *Introduction to the Theory of Computation*. Thomson Course Technology Boston, 2006, vol. 2.

[37] A. Barron, J. Rissanen, and B. Yu, "The minimum description length principle in coding and modeling," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2743–2760, 1998.

[38] "Flex: lexical analyzer generator," https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator).

[39] "Stack exchange data dump," https://archive.org/details/stackexchange, accessed: 2017-07-13.

[40] D. Grune and C. J. Jacobs, "Parsing techniques," *Monographs in Computer Science. Springer,*, p. 13, 2007.

[41] S. Gulwani, "Automating string processing in spreadsheets using input-output examples," in *ACM SIGPLAN Notices*, vol. 46, no. 1. ACM, 2011, pp. 317–330.

[42] "Datamaran technical report," https://arxiv.org/pdf/1708.08905.pdf.

[43] S. Sarawagi et al., "Information extraction," *Foundations and Trends® in Databases*, vol. 1, no. 3, pp. 261–377, 2008.

[44] N. Dalvi, P. Bohannon, and F. Sha, "Robust web extraction: an approach based on a probabilistic tree-edit model," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 335–348.

[45] W. Han, D. Buttler, and C. Pu, "Wrapping web data into xml," *ACM SIGMOD Record*, vol. 30, no. 3, pp. 33–38, 2001.

[46] C.-N. Hsu and M.-T. Dung, "Generating finite-state transducers for semi-structured data extraction from the web," *Information systems*, vol. 23, no. 8, pp. 521–538, 1998.

[47] I. Muslea, S. Minton, and C. Knoblock, "Stalker: Learning extraction rules for semistructured, web-based information sources," in *Proceedings of AAAI-98 Workshop on AI and Information Integration*. AAAI Press, 1998, pp. 74–81.

[48] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Web-scale information extraction in knowitall:(preliminary results)," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 100–110.

[49] E. Agichtein and L. Gravano, "Snowball: Extracting relations from large plain-text collections," in *Proceedings of the fifth ACM conference on Digital libraries*. ACM, 2000, pp. 85–94.

[50] A. Arasu and H. Garcia-Molina, "Extracting structured data from web pages," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 337–348.

[51] V. Crescenzi, G. Mecca, P. Merialdo et al., "Roadrunner: Towards automatic data extraction from large web sites," in *VLDB*, vol. 1, 2001, pp. 109–118.

[52] H. A. Sleiman and R. Corchuelo, "Trinity: on using trinary trees for unsupervised web data extraction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 6, pp. 1544–1556, 2014.

[53] H. A. Sleiman and R. Corchuelo, "Tex: An efficient and effective unsupervised web information extractor," *Knowledge-Based Systems*, vol. 39, pp. 109–123, 2013.

[54] B. Liu, R. Grossman, and Y. Zhai, "Mining data records in web pages," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 601–606.

[55] Y. Zhai and B. Liu, "Web data extraction based on partial tree alignment," in *Proceedings of the 14th international conference on World Wide Web*. ACM, 2005, pp. 76–85.

[56] M. Kayed and C.-H. Chang, "Fivatech: Page-level web data extraction from template pages," *IEEE transactions on knowledge and data engineering*, vol. 22, no. 2, pp. 249–263, 2010.

[57] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces." in *USENIX Security Symposium*, 2007, pp. 1–14.

[58] S. B. Cohen, D. Das, and N. A. Smith, "Unsupervised structure prediction with non-parallel multilingual guidance," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 50–61.

[59] V. I. Spitkovsky, H. Alshawi, A. X. Chang, and D. Jurafsky, "Unsupervised dependency parsing without gold part-of-speech tags," in *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2011, pp. 1281–1290.

[60] P. Senellart, A. Mittal, D. Muschick, R. Gilleron, and M. Tommasi, "Automatic wrapper induction from hidden-web sources with domain knowledge," in *Proceedings of the 10th ACM workshop on Web information and data management*. ACM, 2008, pp. 9–16.

[61] E. Agichtein and V. Ganti, "Mining reference tables for automatic text segmentation," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 20–29.

[62] E. Cortez, D. Oliveira, A. S. da Silva, E. S. de Moura, and A. H. Laender, "Joint unsupervised structure discovery and information extraction," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 541–552.

[63] C. Zhao, J. Mahmud, and I. Ramakrishnan, "Exploiting structured reference data for unsupervised text segmentation with conditional random fields," in *Proceedings of the 2008 SIAM International Conference on Data Mining*. SIAM, 2008, pp. 420–431.

[64] W. W. Cohen, M. Hurst, and L. S. Jensen, "A flexible learning system for wrapping tables and lists in html documents," in *WWW*. New York, NY, USA: ACM, 2002, pp. 232–241.

[65] R. Gupta and S. Sarawagi, "Answering table augmentation queries from unstructured lists on the web," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 289–300, 2009.

[66] A. Machanavajjhala, A. S. Iyer, P. Bohannon, and S. Merugu, "Collective extraction from heterogeneous web lists," in *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 2011, pp. 445–454.

[67] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3363–3372.

[68] V. Raman and J. M. Hellerstein, "Potter's wheel: An interactive data cleaning system," in *VLDB*, vol. 1, 2001, pp. 381–390.

[69] Z. Jin, M. R. Anderson, M. Cafarella, and H. Jagadish, "Foofah: Transforming data by example," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 683–698.

[70] V. Le and S. Gulwani, "Flashextract: a framework for data extraction by examples," in *ACM SIGPLAN Notices*, vol. 49, no. 6. ACM, 2014, pp. 542–553.

[71] D. W. Barowy, S. Gulwani, T. Hart, and B. G. Zorn, "Flashrelate: extracting relational data from semi-structured spreadsheets using examples," in *PLDI'15*, 2015, pp. 218–228.

[72] M. Raza and S. Gulwani, "Automated data extraction using predictive program synthesis," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[73] R. Vaarandi, "A breadth-first algorithm for mining frequent patterns from event logs," *Intelligence in Communication Systems*, pp. 293–308, 2004.

[74] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 1255–1264.

[75] K. Lakshminarayan, S. A. Harp, R. P. Goldman, T. Samad et al., "Imputation of missing data using machine learning techniques." in *KDD*, 1996, pp. 140–145.

[76] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu, "To join or not to join?: Thinking twice about joins before feature selection," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 19–34.

[77] J. Platt et al., "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[78] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[79] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[80] P. L. Bartlett and S. Mendelson, "Rademacher and gaussian complexities: Risk bounds and structural results," *The Journal of Machine Learning Research*, vol. 3, pp. 463–482, 2003.

[81] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[82] R. Durrett, *Probability: Theory and Examples*. Cambridge University Press, 2010.

[83] B. Zadrozny and C. Elkan, "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers," in *ICML*, vol. 1. Citeseer, 2001, pp. 609–616.

[84] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22nd International Conference on Machine Learning*. ACM, 2005, pp. 625–632.

[85] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, no. 4, pp. 132–133, 1972.

[86] D. Suciu, "Probabilistic databases," in *Encyclopedia of Database Systems*. Springer, 2009, pp. 2150–2155.

[87] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[88] L. Getoor and B. Taskar, *Introduction to statistical relational learning*. MIT press, 2007.

[89] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[90] R. Castro Fernandez, E. Mansour, A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Seeping semantics: Linking datasets using word embeddings for data discovery," 04 2018.