Unsupervised Bayesian Data Cleaning Techniques For Structured Data

by

Sushovan De

A Proposal Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved May 2013 by the
Graduate Supervisory Committee:

Dr. Subbarao Kambhampati, Chair
Dr. Selçuk Candan
Dr Yi Chen
Dr Huan Liu

ARIZONA STATE UNIVERSITY

December 2013

ABSTRACT

Data Cleaning, despite being a long standing problem, has occupied the center stage again thanks to the mass of uncurated web data and big data. State of the art approaches for data cleaning suffer from two critical shortcomings: they depend on the availability of clean master data (to learn their data generative models), and they assume the feasibility of offline data rectification (so they can use traditional query processing over clean data at run time). To handle these shortcomings, in this paper I propose a novel mediator system called BayesWipe which employs an end-to-end probabilistic framework to eliminate dependence on clean master data, and a novel query rewriting model to go beyond off-line rectification to on demand cleaning. I present the details of the framework and demonstrate its effectiveness in supporting query processing over inconsistent data in an unsupervised fashion.

ACKNOWLEDGMENTS

I thank Dr. Subbarao Kambhampati, who was an excellent advisor to me.

— Other acknowledgements here —

TABLE OF CONTENTS

Page

LIST OF TABLES . . . vi

LIST OF FIGURES . . . vii

CHAPTER

1 INTRODUCTION . . . 1

1.1 A motivating example . . . 1

1.2 Limitations of existing techniques . . . 1

1.3 BayesWipe approach . . . 2

2 RELATED WORK . . . 4

2.1 Data Cleaning . . . 4

2.2 Query Rewriting . . . 5

3 BAYESWIPE OVERVIEW . . . 7

4 MODEL LEARNING . . . 9

4.1 Data Source Model . . . 9

4.2 Error Model . . . 9

4.2.1 Edit distance similarity: . . . 10

4.2.2 Distributional similarity feature: . . . 10

4.2.3 Unified error model: . . . 11

4.3 Finding the Candidate Set . . . 11

5 OFFLINE CLEANING . . . 13

5.1 Cleaning to a Deterministic Database . . . 13

5.2 Cleaning to a Probabilistic Database . . . 13

6 QUERY REWRITING FOR ONLINE QUERY PROCESSING . . . 16

6.1 Increasing the precision of rewritten queries . . . 16

6.2 Increasing the recall . . . 17

7 EMPIRICAL EVALUATION . . . 20

7.1 Experimental Setup . . . 20

7.2 Experiments . . . 20

8 PROPOSED WORK . . . 26

8.1 Expanded user studies with real data . . . 26

8.2 Extended error model . . . 27

8.3 Explanations . . . 27

Chapter Page

8.4 Query imprecision 29
8.5 Schedule 29
9 CONCLUSION 31
REFERENCES 32

LIST OF TABLES

Table Page

1.1 A snapshot of car data extracted from cars.com using information extraction techniques . . 1

5.1 Cleaned probabilistic database 14

5.2 Result probabilistic database 14

7.1 Results of the Mechanical Turk Experiment, showing the percentage of tuples for which the users picked the results obtained by BayesWipe as against the original tuple. 24

LIST OF FIGURES

Figure Page

3.1 The architecture of BayesWipe. My framework learns both data source model and error model from the raw data during the model learning phase. It can perform offline cleaning or query processing to provide clean data. 8

4.1 Learned Bayes Network: Auto dataset 10

4.2 Learned Bayes Network: Census dataset 10

6.1 Query Expansion Example. The tree shows the candidate constraints that can be added to a query, and the rectangles show the expanded queries with the computed probability values. . 17

6.2 Query Relaxation Example. 19

7.1 Results of probabilistic method. 21

7.2 Average precision vs recall, 20% noise. 22

7.3 Performance evaluations 23

7.4 A fragment of the questionnaire provided to the Mechanical Turk workers. 24

7.5 Average precision vs recall, 20% noise. 24

7.6 Time vs. #Tuples 25

7.7 Time vs. %Noise 25

Chapter 1

INTRODUCTION

Although data cleaning has been a long standing problem, it has become critical again because of the increased interest in web data and big data. The need to efficiently handle structured data that is rife with inconsistency and incompleteness is now more important than ever. Indeed multiple studies [2] emphasize the importance of effective and efficient methods for handling "dirty data" at scale. Although this problem has received significant attention over the years in the traditional database literature, the state-of-the-art approaches fall far short of an effective solution for big data and web data.

1.1 A motivating example

Most of the current techniques are based on deterministic rules, which have a number of problems:

Table 1.1: A snapshot of car data extracted from cars.com using information extraction techniques

TID	Model	Make	Orig	Size	Engine	Condition
t_1	Civic	Honda	JPN	Mid-size	V4	NEW
t_2	Focus	Ford	USA	Compact	V4	USED
t_3	Civik	Honda	JPN	Mid-size	V4	USED
t_4	Civic	Ford	USA	Compact	V4	USED
t_5		Honda	JPN	Mid-size	V4	NEW
t_6	Accord	Honda	JPN	Full-size	V6	NEW

Suppose that the user is interested in finding 'Civic' cars from Table 1.1. Traditional data retrieval systems would return tuples t_1 and t_4 for the query, because they are the only ones that are a match for the query term. Thus, they completely miss the fact that t_4 is in fact a dirty tuple — A Ford Focus car mislabeled as a Civic. Additionally, tuple t_3 and t_5 would not be returned as a result tuples since they have a typos or missing values, although they represent desirable results. My objective is to provide the true result set (t_1, t_3, t_5) to the user. □

1.2 Limitations of existing techniques

A variety of data cleaning approaches have been proposed over the years, from traditional methods (e.g., outlier detection [29], noise removal [42], entity resolution [42, 21], and imputation[23]) to recent efforts on examining integrity constraints, Although these methods are efficient in their own scenarios, their dependence on clean master data is a significant drawback.

Specifically, state of the art approaches (e.g., [9, 20, 5] attempt to clean data by exploiting patterns in the data, which they express in the form of conditional functional dependencies (or CFDs). In my motivating example, the fact that Honda cars have 'JPN' as the origin of the manufacturer would

be an example of such a pattern. However, these approaches depend on the availability of a clean data corpus or an external reference table to learn data quality rules or patterns before fixing the errors in the dirty data. Systems such as ConQuer [24] depend upon a set of clean constraints provided by the user. Such clean corpora or constraints may be easy to establish in a tightly controlled enterprise environment but are infeasible for web data and big data. One may attempt to learn data quality rules directly from the noisy data. Unfortunately however, my experimental evaluation shows that even small amounts of noise severely impairs the ability to learn useful constraints from the data.

1.3 BayesWipe approach

To avoid dependence on clean master data, in this paper, I propose a novel system called BayesWipe that assumes that a statistical process underlies the generation of clean data (which I call the *data source model*) as well as the corruption of data (which I call the data *error model*). The noisy data itself is used to learn the parameters of these the generative and error models, eliminating dependence on clean master data. Then, by treating the clean value as a latent random variable, BayesWipe leverages these two learned models and automatically infers its value through a Bayesian estimation.

I designed BayesWipe so that it can be used in two different modes: a traditional *offline cleaning* mode, and a novel *online query processing* mode. The offline cleaning mode of BayesWipe follows the classical data cleaning model, where the entire database is accessible and can be cleaned *in situ*. This mode is particularly useful for cleaning data crawled from the web, or aggregated from various noisy sources. The cleaned data can be stored either in a deterministic database, or in a probabilistic database. If a probabilistic database is chosen as the output mode, BayesWipe stores not only the clean version of the tuple it believes to be most likely correct one, but the entire distribution over possible clean tuples. This mode is most useful for those scenarios where recall is very important for further data processing on the cleaned tuples.

The online query processing mode of BayesWipe is motivated by web data scenarios where it is impractical to create a local copy of the data and clean it offline, either due to large size, high frequency of change, or access restrictions. In such cases, the best way to obtain clean answers is to clean the resultset as I retrieve it, which also provides me the opportunity of improving the efficiency of the system, since I can now ignore entire portions of the database which are likely to be unclean or irrelevant to the top-k. BayesWipe uses a query rewriting system that enables it to efficiently retrieve only those tuples that are important to the top-k result set. This rewriting approach is inspired by, and is a significant extension of our earlier work on QPIAD system for handling data incompleteness [41].

The rest of the proposal is organized as follows. I describe the architecture of BayesWipe in

the next section. Section 4 describes the learning phase of BayesWipe, where I find the data and error models. Section 5 describes the offline cleaning mode, and the next section details the query rewriting and online data processing. I describe the results of my empirical evaluation in Section 7, and then conclude by summarizing my contributions.

Chapter 2

RELATED WORK

2.1 Data Cleaning

The current state of the art in data cleaning focuses on deterministic dependency relations such as FD, CFD, and INDs.

CFDs: Bohannon *et al.* proposed [10, 19] using Conditional Functional Dependencies (CFD) to clean data. Indeed, CFDs are very effective in cleaning data. However, the precision and recall of cleaning data with CFDs completely depends on the quality of the set of dependencies used for the cleaning. As my experiments show, learning CFDs from dirty data produces very unsatisfactory results. In order for CFD-based methods to perform well, they need to be learned from a clean sample of the database [20]. Learning CFDs are more difficult than learning plain FDs. For FDs, the search space is of the order of the number of all possible combinations of the attributes. In the case of CFDs, each such dependency is further adorned with a pattern tableau, that determines specific patterns in tuples to which the dependency applies. Thus, the search space for mining CFDs is extended by all combinations of all possible *values* that can appear in the attributes. Not only does this make learning CFDs from dirty data more infeasible — this also shows that the clean sample of the database from which CFDs are learn must be large enough to be representative of all the patterns in the data. Finding such a large corpus of clean master data is a non-trivial problem, and is infeasible in all but the most controlled of environments (like a corporation with high quality data).

Indeed, all these deterministic dependency based solutions were focused towards the business data problem, where it is well known that the error rate lies between 1% – 5% [38]. BayesWipe can handle much higher rates of error, which makes this technique applicable for web data and user-generated data scenarios, which are much more relevant today. This is because BayesWipe learns both the generative and error model from the dirty data itself using Bayes networks and not deterministic rules; the system is a lot more forgiving of dirtiness in the training sample.

Even if a curated set of integrity constraints are provided, existing methods do not use a probabilistically principled method of choosing a candidate correction. They resort to either heuristic based methods, finding an approximate algorithm for the least-cost repair of the database [1, 9, 13]; using a human-guided repair [43], or sampling from a space of possible repairs [8]. There has been work that attempts to guarantee a correct repair of the database [22], but they can only provide guarantees for corrections of those tuples that are supported by data from a perfectly clean master database. Recently, Beskales *et al.* have shown [7] how the relative trust one places on the constraints and the data itself

plays into the choice of cleaning tuples. A Bayesian source model of data was used by Dong *et al.*[17], but was limited in scope to figuring out the evolution over time of the data value.

Most of the existing work has been focused on deterministic rules, and as a result, the repairs to the database they perform do not have any probabilistic semantics. On the other hand, BayesWipe provides confidence numbers to each of the repairs it performs, which is the posterior probability (in a Bayesian sense) of the corrected tuple given the input database and error models. Recent work [6] shows the use of a principled probabilistic method for two scenarios: (1) using a probabilistic database to perform data de-duplication, and (2) to fix violations of functional dependencies. Similar to (1), I allow the use of a probabilistic database, however, I use it to store the outcome of my cleaning of corrupted data. As for (2), it has been shown that CFDs are far more effective in data cleaning that FDs [10], and I show in this paper that my approach is superior to CFDs as well.

Kubica and Moore also use a probabilistic model that attempts to learn the generative and error model [31], and apply it to a image processing domain. However, this paper separates the noise model into two parts, the noise itself, and the corruption given the noise. Additionally, Kubica and Moore do not specify how the generative and error models were learned.

Recent work has also focused on the metrics to use to evaluate data cleaning techniques [15]. In this paper, I focus on evaluating my method against ground truth (when the ground truth is known), and user studies (when the ground truth is not known).

2.2 Query Rewriting

The classic problem of query rewriting [36] is to take a SQL query Q that was written against the full database D, and reformulate it to work on a set of views $\mathcal{V}$ so that it produces the same output. I use a similar approach in this paper — when it is not possible to clean the entire database in place, I use query rewriting to efficiently obtain those tuples that are most likely to be relevant to the user by exploiting all the views that the database *does* expose.

The query rewriting part of this paper is inspired by the QPIAD system [41], but significantly improves upon it. QPIAD performed query rewriting over incomplete databases using approximate functional dependencies (AFD). Unlike QPIAD, BayesWipe supports cleaning databases that have both null values as well as *wrong* values. The problem I are attempting to solve in this paper would not be solvable by QPIAD, since it needs to know the exact attribute that is dirty (QPIAD assumed any non-null value in a tuple was correct). The inference problem I solve is much harder, since I have to infer both the location as well as the value of the error in the tuples.

Researchers have also suggested query rewriting techniques to get clean answers over inconsistent databases. However, there are significant differences between the problem that I solve and the problem typically solved by query rewriting techniques. Arenas *et al.* show [1] a method to generate rewritten queries to obtain clean tuples from an inconsistent database. However, the query rewriting algorithm in that paper is driven by the deterministic integrity dependencies, and not the generative or error model. Since this system requires a set of curated deterministic dependencies, it is not directly applicable to the problem solved in this paper. Further, due to the use of Bayes networks to model the generative model, BayesWipe is able to incorporate richer types of dependencies.

Recently, performing cleaning of the top-k results of a query has gained interest. Mo *et al.* propose a system [35] that cleans just the top-k returned tuples — similar to what I do in this paper. However, their definition of cleaning the data is very different from ours; while I algorithmically find the best correction for a given tuple, they query the real world for a cleaner sample of any tuple that the system flags as ambiguous.

Chapter 3

BAYESWIPE OVERVIEW

BayesWipe views the data cleaning problem as a statistical inference problem over the structured data. Let $\mathcal{D} = \{T_1, ..., T_n\}$ be the input structured data which contains a certain number of corruptions. $T_i \in \mathcal{D}$ is a tuple with m attributes $\{A_1, ..., A_m\}$ which may have one or more corruptions in its attribute values. So given a correction candidate set $\mathcal{C}$ for a possibly corrupted tuple T in $\mathcal{D}$, I can clean the database by replacing T with the candidate clean tuple $T^* \in \mathcal{C}$ that has the maximum $P(T^*|T)$. Using Bayes rule (and dropping the common denominator), I can rewrite this to

$$T^*_{best} = \arg\max[P(T|T^*)P(T^*)] \tag{3.1}$$

If I wish to create a probabilistic database (PDB), I don't take an $\arg\max$ over the $P(T^*|T)$, instead I store the entire distribution over the T^* in the resulting PDB.

For online query processing I take the user query Q^*, and find the relevance score of a tuple T as

$$Score(T) = \sum_{T^* \in \mathcal{C}} \underbrace{P(T^*)}_{\text{source model}} \underbrace{P(T|T^*)}_{\text{error model}} \underbrace{R(T^*|Q^*)}_{\text{relevance}} \tag{3.2}$$

In this paper, I used a binary relevance model, where R is 1 if T^* is relevant to the user's query, and 0 otherwise. Note that R is the relevance of the query Q^* to the candidate clean tuple T^* and not the observed tuple T. This allows the query rewriting phase of BayesWipe, which aims to retrieve tuples with highest $Score(.)$ to achieve the non-lossy effect of using a PDB without explicitly rectifying the entire database.

Architecture: Figure 3.1 shows the system architecture for BayesWipe. During the model learning phase (Section 4), I first obtain a sample database by sending some queries to the database. On this sample data, I learn the generative model of the data as a Bayes network (Section 4.1). In parallel, I define and learn an error model which incorporates three types of errors (Section 4.2). I also create an index to quickly propose candidate T^*s.

I can then choose to do either offline cleaning (Section 5) or online query processing (Section 6), as per the scenario. In the offline cleaning mode, I can choose whether to store the resulting cleaned tuple in a deterministic database (where I store only the T^* with the maximum posterior probability) or probabilistic database (where I store the entire distribution over the T^*). In the online query processing mode, I obtain a query from the user, and do query rewriting in order to find a set of queries that are

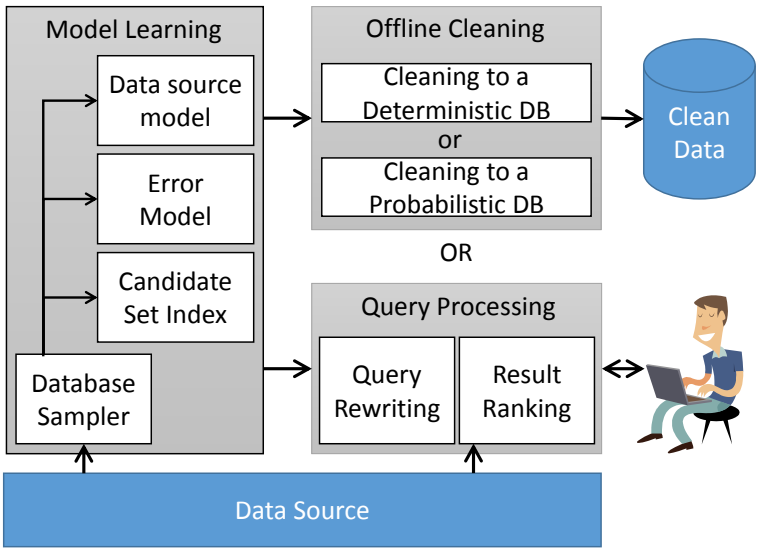


Figure 3.1: The architecture of BayesWipe. My framework learns both data source model and error model from the raw data during the model learning phase. It can perform offline cleaning or query processing to provide clean data.

likely to retrieve a set of highly relevant tuples. I execute these queries and re-rank the results, and then display them to the user.

Chapter 4

MODEL LEARNING

This chapter details the process by which I estimate the components of Equation 3.2: the data source model $P(T^*)$ and the error model $P(T|T^*)$

4.1 Data Source Model

The data that I work with can have dependencies among various attributes (e.g., a car's *engine* depends on its *make*). Therefore, I represent the data source model as a Bayes network, since it naturally captures relationships between the attributes via structure learning and infers probability distributions over values of the input tuple instances.

Constructing a Bayes network over $\mathcal{D}$ requires two steps: first, the induction of the graph structure of the network, which encodes the conditional independences between the m attributes of $\mathcal{D}$'s schema; and second, the estimation of the parameters of the resulting network. The resulting model allows me to compute probability distributions over an arbitrary input tuple T.

I observe that the structure of a Bayes network of a given dataset remains constant with small perturbations, but the parameters (CPTs) change more frequently. As a result, I spend a larger amount of time learning the structure of the network with a slower, but more accurate tool, Banjo [25]. Figures 4.1 and 4.2 show automatically learned structures for two data domains.

Then, given a learned graphical structure $\mathcal{G}$ of $\mathcal{D}$, I can estimate the conditional probability tables (CPTs) that parameterize each node in $\mathcal{G}$ using a faster package called Infer.NET [34]. This process of inferring the parameters is run offline, but more frequently than the structure learning.

Once the Bayesian network is constructed, I can infer the joint distributions for arbitrary tuple T, which can be decomposed to the multiplication of several marginal distributions of the sets of random variables, conditioned on their parent nodes depending on $\mathcal{G}$.

4.2 Error Model

Having described the data source model, I now turn to the estimation of the error model $P(T|T^*)$ from noisy data. Given a set of clean candidate tuples $\mathcal{C}$ where $T^* \in \mathcal{C}$, my error model $P(T|T^*)$ essentially measures how clean T is, or in other words, how similar T is to T^*. Unlike traditional record linkage measures [30], my similarity functions have to take into account dependencies among attributes.

I now build an error model that can estimate some of the most common kinds of errors in real

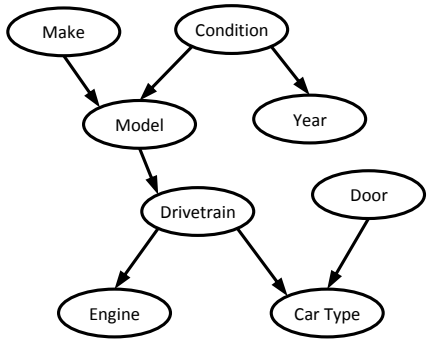


Figure 4.1: Learned Bayes Network: Auto dataset

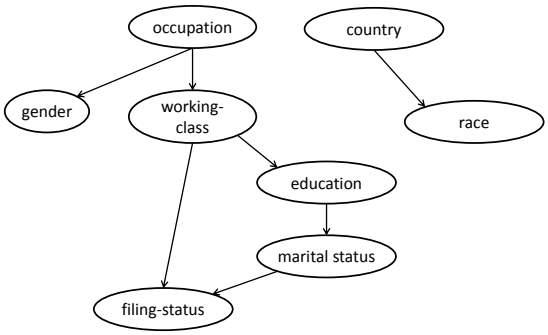


Figure 4.2: Learned Bayes Network: Census dataset

data: a combination of spelling, incompletion and substitution errors.

4.2.1 Edit distance similarity:

This similarity measure is used to detect spelling errors. Edit distance between two strings T_{A_i} and $T^*_{A_i}$ is defined as the minimum cost of edit operations applied to dirty tuple T_{A_i} transform it to clean $T^*_{A_i}$. Edit operations include character-level copy, insert, delete and substitute. The cost for each operation can be modified as required; in this paper I use the Levenshtein distance, which uses a uniform cost function. This gives me a distance, which I then convert to a probability using [39]:

$$f_{ed}(T_{A_i}, T^*_{A_i}) = \exp\{-cost_{ed}(T_{A_i}, T^*_{A_i})\} \tag{4.1}$$

4.2.2 Distributional similarity feature:

This similarity measure is used to detect both substitution and omission errors. Looking at each attribute in isolation is not enough to fix these errors. I propose a context-based similarity measure called Distributional similarity (f_{ds}), which is based on the probability of replacing one value with another under a similar context [32]. Formally, for each string T_{A_i} and $T^*_{A_i}$, I have:

$$f_{ds}(T_{A_i}, T^*_{A_i}) = \sum_{c \in C(T_{A_i}, T^*_{A_i})} \frac{\mathbf{Pr}(c|T^*_{A_i})\mathbf{Pr}(c|T_{A_i})\mathbf{Pr}(T_{A_i})}{\mathbf{Pr}(c)} \tag{4.2}$$

where $C(T_{A_i}, T^*_{A_i})$ is the context of a tuple attribute value, which is a set of attribute values that co-occur with both T_{A_i} and $T^*_{A_i}$. $\mathbf{Pr}(c|T^*_{A_i}) = (\#(c, T^*_{A_i}) + \mu)/\#(T^*_{A_i})$ is the probability that a context value c appears given the clean attribute $T^*_{A_i}$ in the sample database. Similarly, $P(T_{A_i}) = \#(T_{A_i})/\#tuples$ is the probability that a dirty attribute value appears in the sample database. I calculate $\mathbf{Pr}(c|T_{A_i})$ and $\mathbf{Pr}(T_{A_i})$ in the same way. To avoid zero estimates for attribute values that do not appear in the database sample, I use Laplace smoothing factor μ.

4.2.3 Unified error model:

In practice, I do not know beforehand which kind of error has occurred for a particular attribute; I need a unified error model which can accommodate all three types of errors (and be flexible enough to accommodate more errors when necessary). For this purpose, I use the well-known maximum entropy framework [4] to leverage all available similarity measures, including Edit distance f_{ed} and distributional similarity f_{ds}. So for the input tuple T and T^*, I have my unified error model defined on this attribute as follows:

$$\mathbf{Pr}(T|T^*) = \frac{1}{Z} \exp\left\{ \alpha \sum_{i=1}^{m} f_{ed}(T_{A_i}, T^*_{A_i}) + \beta \sum_{i=1}^{m} f_{ds}(T_{A_i}, T^*_{A_i}) \right\} \tag{4.3}$$

where α and β are the weight of each similarity measure, m is the number of attributes in the tuple. The normalization factor is $Z = \sum_{T^*} \exp\{\sum_i \lambda_i f_i(T^*, T)\}$.

4.3 Finding the Candidate Set

I need to find the set of candidate clean tuples, $\mathcal{C}$, that comprises all the tuples in the sample database that differ from T in not more than j attributes. Even with $j = 3$, the naïve approach of constructing $\mathcal{C}$ from the sample database directly is too time consuming, since it requires one to go through the sample database in its entirety once for every result tuple encountered. To make this process faster, I create indices over $(j + 1)$ attributes. If any candidate tuple T^* differs from T in less than or equal to j attributes, then it will be present in at least one of the indices, since I created $j + 1$ of them. These $j + 1$ indices are created over those attributes that have the highest cardinalities, such as Make and Model (as opposed to attributes like Condition and Doors which can take only a few values). For every possibly dirty tuple T in the database, I go over each such index and find all the tuples that match the corresponding attribute. The union of all these tuples is then examined and the candidate set $\mathcal{C}$ is constructed by keeping only those tuples from this union set that do not differ from T in more than j attributes.

Thus I can be sure that by using this method, I have obtained the entire set $\mathcal{C}$. By using those attributes that have high cardinality, I ensure that the size of the set of tuples returned from the index

would be small.

Chapter 5

OFFLINE CLEANING

5.1 Cleaning to a Deterministic Database

In order to clean the data *in situ*, I first use the techniques of the previous section to learn the data generative model, the error model and create the index. Then, I iterate over all the tuples in the database and use Equation 3.1 to find the T^* with the best score. I then replace the tuple with that T^*, thus creating a deterministic database using the offline mode of BayesWipe.

5.2 Cleaning to a Probabilistic Database

I note that many data cleaning approaches — including the one I described in the previous sections — come up with multiple alternatives for the clean version for any given tuple, and evaluate their confidence in each of the alternatives. For example, if a tuple is observed as 'Honda, Corolla', two correct alternatives for that tuple might be 'Honda, Civic' and 'Toyota, Corolla'. In such cases, where the choice of the clean tuple is not an obvious one, picking the most-likely option may lead to the wrong answer. Additionally, if one intends to do further processing on the results, such as perform aggregate queries, join with other tables, or transfer the data to someone else for processing, then storing the most likely outcome is lossy.

A better approach (also suggested by others [2]) is to store all of the alternative clean tuples along with their confidence values. Doing this, however, means that the resulting database will be a probabilistic database (PDB), even when the source database is deterministic.

It is not clear upfront whether PDB-based cleaning will have advantages over cleaning to a deterministic database. On the positive side, using a PDB helps reduce loss of information arising from discarding all alternatives to tuples that did not have the maximum confidence. On the negative side, PDB-based cleaning increases the query processing cost (as querying PDBs are harder than querying deterministic databases [14]). Another challenge is that of presentation: users usually assume that they are dealing with a deterministic source of data, and presenting all alternatives to them can be overwhelming to them.

In this section, and in the associated experiments, I investigate the potential advantages to using the BayesWipe system and storing the resulting cleaned data in a probabilistic database. For my experiments, I used Mystiq [11], a prototype probabilistic database system from University of Washington, as the substrate.

In order to create a probabilistic database from the corrections of the input data, I follow the

offline cleaning procedure described previously in Section 4. Instead of storing the most likely T^*, I store all the T^*s along with their $P(T^*|T)$ values.

When evaluating the performance of the probabilistic database, I used simple select queries on the resulting database. Since representing the results of a probabilistic database to the user is a complex task, in this paper I focus on showing just the tuple ID to the user. The rationale for my decision is that in a used car scenario, the user will be satisfied if the system provides a link to the car the user intended to purchase — the exact reasoning the system used to come up with the answer is not relevant to the user. As a result, the form of my output is a tuple-independent database. This can be better explained with an example:

Table 5.1: Cleaned probabilistic database

TID	Model	Make	Orig.	Size	Eng.	Cond.	P
t_1	Civic	Honda	JPN	Mid-size	V4	NEW	0.6
	Civic	Honda	JPN	Compact	V6	NEW	0.4
...							
t_3	Civic	Honda	JPN	Mid-size	V4	USED	0.9
	Civik	Honda	JPN	Mid-size	V4	USED	0.1

Example: Suppose I clean my running example of Table 1.1. I will obtain a tuple-disjoint independent[1] probabilistic database [40]; a fragment of which is shown in Table 5.1. Each original input tuple (t_1, t_3), has been cleaned, and their alternatives are stored along with the computed confidence values for the alternatives (0.6 and 0.4 for t_1, in this example). Suppose the user issues a query Model = Civic. Both options of tuple t_1 of the probabilistic database satisfy the constraints of the query. Since I are only interested in the tuple ID, I project out every other attribute, resulting in returning tuple t_1 in the result with a probability $0.6 + 0.4 = 1$. Only the first option in tuple t_3 matches the query. Thus the result will contain the tuple t_3 with probability 0.9. The experimental results use only the tuple ids when computing the recall of the method. The output probabilistic relation is shown in Table 5.2.

Table 5.2: Result probabilistic database

TID	P
t_1	1
t_3	0.9

The interesting fact here is that the result of any query will always be a tuple-independent database. This is because I projected out every attribute except for the tuple-ID, and the tuple-IDs are independent of each other. □

When showing the results of my experiments, I evaluate the precision and recall of the system. Since precision and recall are deterministic concepts, I have to convert the probabilistic database into

[1] A tuple-disjoint independent probabilistic database is one where every tuple, identified by its primary key, is independent of all other tuples. Each tuple is, however, allowed to have multiple alternatives with associated probabilities. In a tuple-independent database, each tuple has a single probability, which is the probability of that tuple existing.

a deterministic database (that will be shown to the user) prior to computing these values. I can do this conversion in two ways: (1) by picking only those tuples whose probability is higher than some threshold. I call this method the *threshold based determinization*. (2) by picking the top-k tuples and discarding the probability values (*top-k determinization*). The experiment section (Section 7.2) shows results with both determinizations.

Chapter 6

QUERY REWRITING FOR ONLINE QUERY PROCESSING

In this chapter I develop an online query processing method where the result tuples are cleaned at query time. Two challenges need to be addressed to do this effectively. First, certain tuples that do not satisfy the query constraints, but are relevant to the user, need to be retrieved, ranked and shown to the user. Second, the process needs to be efficient, since the time that the users are willing to wait before results are shown to them is very small. I show my query rewriting mechanisms aimed at addressing both these challenges.

I begin by executing the user's query (Q^*) on the database. I store the retrieved results, but do not show them to the user immediately. I then find rewritten queries that are most likely to retrieve clean tuples. I do that in a two-stage process: I first expand the query to increase the precision, and then relax the query by deleting some constraints (to increase the recall).

6.1 Increasing the precision of rewritten queries

Since my data sources are inherently noisy, it is important that I do not retrieve tuples that are obviously incorrect. Doing so will improve not only the quality of the result tuples, but also the efficiency of the system. I can improve precision by adding relevant constraints to the query Q^* given by the user. For example, when a user issues the query Model = Civic, I can expand the query to add relevant constraints Make = Honda, Country = Japan, Size = Mid-Size. These additions capture the essence of the query — because they limit the results to the specific kind of car the user is probably looking for. These expanded structured queries I generate from the user's query are called ESQs.

Given a Q^*, I attempt to generate multiple ESQs that maximizes both the relevance of the results and the coverage of the queries of the solution space.

Note that if there are m attributes, each of which can take n values, then the total number of possible ESQs is n^m. Searching for the ESQ that globally maximizes the objectives in this space is infeasible; I therefore approximately search for it by performing a heuristic-informed search. My objective is to create an ESQ with m attribute-value pairs as constraints. I begin with the constraints specified by the user query Q^*. I set these as evidence in the Bayes network, and then query the Markov blanket of these attributes for the attribute-value pairs with the highest posterior probability given this evidence. I take the top-k attribute-value pairs and append them to Q^* to produce k search nodes, each search node being a query fragment. If Q has p constraints in it, then the heuristic value of Q is given by $P(Q)^{m/p}$. This represents the expected joint probability of Q when expanded to m attributes, assuming

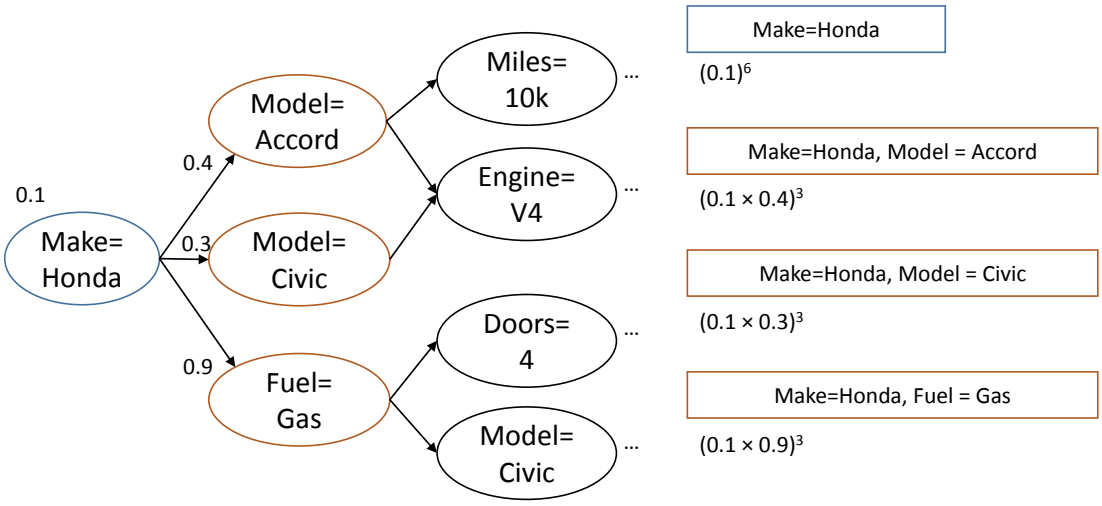


Figure 6.1: Query Expansion Example. The tree shows the candidate constraints that can be added to a query, and the rectangles show the expanded queries with the computed probability values.

that all the constraints will have the same average posterior probability. I expand them further, until I find k queries of size m with the highest probabilities.

Example: In Figure 6.1, I show an example of the query expansion. The node on the left represents the query given by the user "Make=Honda". First, I look at the Markov Blanket of the attribute Make, and determine that Model and Condition are the nodes in the Markov blanket. I then set "Make=Honda" as evidence in the Bayes network and then run an inference over the values of the attribute Model. The two values of the Model attribute with the highest posterior probability are Accord and Civic. The most probable values of the Condition attribute are "new" and "old". Using each of these values, new queries are constructed and added to the queue. Thus, the queue now consists of the 4 queries: "Make=Honda, Model=Civic", "Make=Honda, Model=Accord" and "Make=Honda, Condition=old". A fragment of these queries are shown in the middle column of Figure 6.1. I dequeue the highest probability item from the queue and repeat the process of setting the evidence, finding the Markov Blanket, and running the inference. I stop when I get the required number of ESQs with a sufficient number of constraints.

6.2 Increasing the recall

Adding constraints to the query causes the precision of the results to increase, but reduces the recall drastically. Therefore, in this stage, I choose to delete some constraints from the ESQs, thus generating relaxed queries (RQ). Notice that tuples that have corruptions in the attribute constrained by the user (recall tuples t_3 and t_5 from my running example in Table 1.1) can only be retrieved by relaxed queries that do not specify a value for those attributes. Instead, I have to depend on rewritten queries that

contain correlated values in other attributes to retrieve these tuples. Using relaxed queries can be seen as a trade-off between the recall of the resultset and the time taken, since there are an exponential number of relaxed queries for any given ESQ. As a result, an important question is the order and number of RQs to execute.

I define the rank of a query as the *expected relevance* of its result set.

$$Rank(q) = \mathbb{E}\left(\frac{\sum_{T_q} Score(T_q|Q^*)}{|T_q|}\right)$$

where T_q are the tuples returned by a query q, and Q^* is the user's query. Executing an RQ with a higher rank will have a more beneficial result on the result set because it will bring in better quality result tuples.

Estimating this quantity is difficult because I do not have complete information about the tuples that will be returned for any query q. The best I can do, therefore, is to approximate this quantity.

Let the relaxed query be Q, and the expanded query that it was relaxed from be ESQ. I wish to estimate $\mathbb{E}[P(T|T^*)]$ where T are the tuples returned by Q. Using the attribute-error independence assumption, I can rewrite that as $\prod_{i=0}^{m} P(T.A_i|T^*.A_i)$, where $T.A_i$ is the value of the i-th attribute in T. Since ESQ was obtained by expanding Q^* using the Bayes network, it has values that can be considered clean for this evaluation. Now, I divide the m attributes of the database into 3 classes: (1) The attribute is specified both in ESQ and in Q. In this case, I set $P(T.A_i|T^*.A_i)$ to 1, since $T.A_i = T^*.A_i$. (2) The attribute is specified in ESQ but not in Q. In this case, I know what $T^*.A_i$ is, but not $T.A_i$. However, I can generate an average statistic of how often $T^*.A_i$ is erroneous by looking at my sample database. Therefore, in the offline learning stage, I pre-compute tables of error statistics for every T^* that appears in my sample database, and use that value. (3) The attribute is not specified in either ESQ or Q. In this case, I know neither the attribute value in T nor in T^*. I, therefore, use the average error rate of the entire attribute as the value for $P(T.A_i|T^*.A_i)$. This statistic is also precomputed during the learning phase. This product gives me the expected rank of the tuples returned by Q.

Example: In Figure 6.2, I show an example for finding the probability values of a relaxed query. Assume that the user's query Q^* is "Civic", and the ESQ is shown in the second row. For an RQ that removes the attribute values "Civic" and "Mid-Size" from the ESQ, the probabilities are calculated as follows: For the attributes "Make, Country" and "Engine", the values are present in both the ESQ as well as the RQ, and therefore, the $P(T|T^*)$ for them is 1. For the attribute "Model" and "Type", the values are present in ESQ but not in RQ, hence the value for them can be computed from the learned error statistics. For example, for "Civic", the average value of $P(T|Civic)$ as learned from the sample database (0.8) is used. Finally, for the attribute "Condition", which is present neither in ESQ nor in RQ,

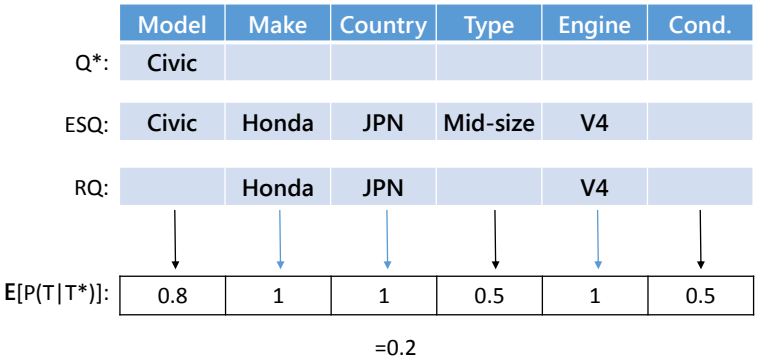


Figure 6.2: Query Relaxation Example.

I use the average error statistic for that attribute (i.e. the average of $P(T_a|T_a^*)$ for a = "Condition" which is 0.5).

The final value of $\mathbb{E}[P(T|T^*)]$ is found from the product of all these attributes as 0.2. □

Terminating the process: I begin by looking at all the RQs in descending order of their rank. If the current k-th tuple in my resultset has a relevance of λ, and the estimated rank of the Q I am about to execute is $R(T_q|Q)$, then I stop evaluating any more queries if the probability $\mathbf{Pr}(R(T_q|Q) > \lambda)$ is less than some user defined threshold $\mathcal{P}$. This ensures that I have the true top-k resultset with a probability $\mathcal{P}$.

Chapter 7

EMPIRICAL EVALUATION

I quantitatively study the performance of BayesWipe in both its modes — offline, and online, and compare it against state-of-the-art CFD approaches. I used three real datasets spanning two domains: used car data, and census data. I present experiments on evaluating the approach in terms of the effectiveness of data cleaning, efficiency and precision of query rewriting.

7.1 Experimental Setup

To perform the experiments, I obtained the real data from the web. The first dataset is *Used car sales* dataset D_{car} crawled from Google Base. The second dataset I used was adapted from the *Census Income* dataset D_{census} from the UCI machine learning repository [3]. From the fourteen available attributes, I picked the attributes that were categorical in nature, resulting in the following 8 attributes: working-class, education, marital status, occupation, race, gender, filing status. country. The same setup was used for both datasets – including parameter values and error features.

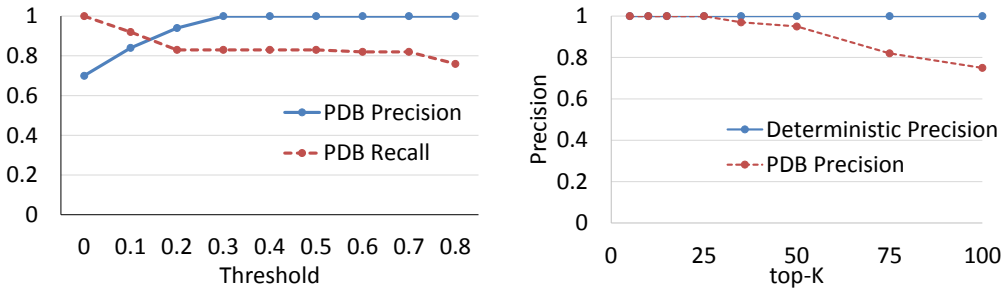
These datasets were observed to be mostly clean. I then introduced[1] three types of noise to the attributes. To add noise to an attribute, I randomly changed it either to a new value which is close in terms of string edit distance (distance between 1 and 4, simulating spelling errors) or to a new value which was from the same attribute (simulating replacement errors) or just deleted it (simulating deletion errors).

A third dataset was car inventory data crawled from the website 'cars.com'. This dataset was observed to have inaccuracies — therefore, I used this to validate my approach against real-world noise in the data, where I do not control the noise process.

7.2 Experiments

Offline Cleaning Evaluation: The first set of evaluations shows the effectiveness of the offline cleaning mode. In Figure **??**, I compare BayesWipe against CFDs [12]. The dotted line that shows the number of CFDs learned from the noisy data quickly falls to zero, which is not surprising: CFDs learning was designed with a clean training dataset in mind. Further, the only constraints learned by this algorithm are the ones that have not been violated in the dataset — unless a tuple violates some CFD, it cannot be cleaned. As a result, the CFD method cleans exactly zero tuples independent of the noise percentage. On the other hand, BayesWipe is able to clean between 20% to 40% of the data. It is interesting to note that the percentage of tuples cleaned increases initially and then slowly decreases. This is because

[1] I note that the introduction of synthetic errors into clean data for experimental evaluation purposes is common practice in data cleaning research [13, 10].



(a) Precision and recall of the PDB method using a threshold.

(b) top-k precision of PDB vs deterministic method.

Figure 7.1: Results of probabilistic method.

for very low values of noise, there aren't enough errors available for the system to learn a reliable error model from; and at larger values of noise, the data source model learned from the noisy data is of poorer quality.

While Figure **??** showed only percentages, in Figure **??** I report the actual number of tuples cleaned in the dataset along with the percentage cleaned. This curve shows that the raw number of tuples cleaned always increases with higher input noise percentages.

Setting α and β: The weight given to the distributional similarity (β), and the weight given to the edit distance (α) are parameters that can be tuned, and should be set based on which kind of error is more likely to occur. In my experiments, I performed a grid search to determine the best values of α and β to use. In Figure **??**, I show a portion of the grid search where $\alpha = 2\beta/3$.

The "values corrected" data points in the graph correspond to the number of erroneous attribute values that the algorithm successfully corrected (when checked against the ground truth). The "false positives" are the number of legitimate values that the algorithm changes to an erroneous value. When cleaning the data, my algorithm chooses a candidate tuple based on both the prior of the candidate as well as the likelihood of the correction given the evidence. Low values of α, β give a higher weight to the prior than the likelihood, allowing tuples to be changed more easily to candidates with high prior. The "overall gain" in the number of clean values is calculated as the difference of clean values between the output and input of the algorithm.

If I set the parameter values too low, I will correct most wrong tuples in the input dataset, but I will also 'overcorrect' a larger number of tuples. If the parameters are set too high, then the system will not correct many errors — but the number of 'overcorrections' will also be lower. Based on these experiments, I picked a parameter value of $\alpha = 3.7, \beta = 2.1$ and kept it constant for all my experiments.

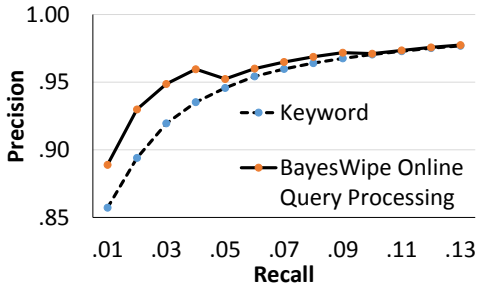


Figure 7.2: Average precision vs recall, 20% noise.

Using probabilistic databases: I empirically evaluate the PDB-mode of BayesWipe in Figure 7.1. The first figure shows the system using the threshold determinization. I plot the precision and recall as the probability threshold for inclusion of a tuple in the resultset is varied. As expected, with low values of the threshold, the system allows most tuples into the resultset, thus showing high recall and low precision. As the threshold increased, the precision increases, but the recall falls.

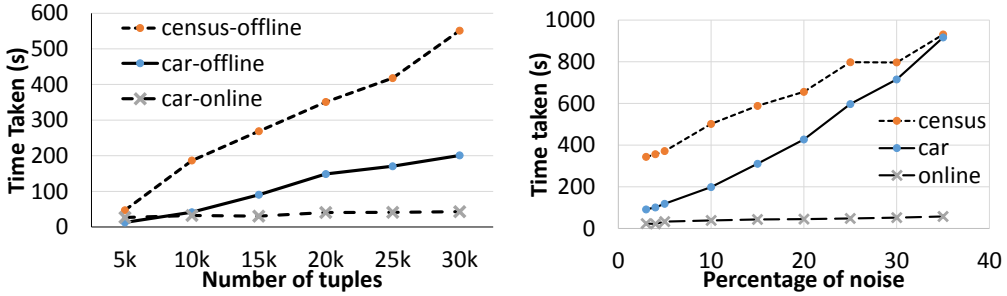
In Figure 7.1b, I compare the precision of the PDB mode using top-k determinization against the deterministic mode of BayesWipe. As expected, both the modes show high precision for low values of k, indicating that the initial results are clean and relevant to the user. For higher values of k, the PDB precision falls off, indicating that PDB methods are more useful for scenarios where high recall is important without sacrificing too much precision.

Online Query Processing: Since there is no existing work on querying autonomous data sources in the presence of data inconsistency, I consider a **keyword query** system as my baseline. I evaluate the precision and recall of my method against the ground truth and compare it with the baseline.

I issued randomly generated queries to both BayesWipe and the baseline system. Figure 7.5 shows the average precision over 10 queries at various recall values. It shows that my system outperforms the keyword query system in precision, especially since my system considers the relevance of the results when ranking them. On the other hand, the keyword search approach is oblivious to ranking and returns all tuples that satisfy the user query. Thus it may return irrelevant tuples early on, leading to a loss in precision.

This shows that my proposed query ranking strategy indeed captures the expected relevance of the to-be-retrieved tuples, and the query rewriting module is able to generate the highly ranked queries.

Efficiency: In Figure 7.6 I evaluate the time taken as the number of tuples in the database increases, and in Figure 7.7 I show the time taken as the noise varies. These graphs show that both the offline and online



(a) Time vs. #Tuples

(b) Time vs. %Noise

Figure 7.3: Performance evaluations

modes of BayesWipe complete in a reasonable time. In particular, this shows that the query processing mode shows the time taken remains mostly constant under both varying database size as well as noise. This is because the most expensive parts of the query processing algorithm (determining the error and generative models) are precomputed offline. This shows that BayesWipe can be used as a viable tool for large-scale web-data.

Evaluation on real data with naturally occurring errors: In this section I used a dataset of 1.2 million tuples crawled from the cars.com website[2] to check the performance of the system with real-world data, where the corruptions were not synthetically introduced. Since this data is large, and the noise is completely naturally occurring, I do not have ground truth for this data. To evaluate this system, I conducted an experiment on Amazon Mechanical Turk. First, I ran the offline mode of BayesWipe on the entire database. I then picked only those tuples that were changed during the cleaning, and then created an interface in mechanical turk where only those tuples were shown to the user in random order. Due to resource constraints, the experiment was run with the first 200 tuples that the system found to be unclean.

An example is shown in Figure 7.4. The turker is presented with two cars, and she does not know which of the cars was originally present in the dirty dataset, and which one was produced by BayesWipe. The turker will use her own domain knowledge, or perform a web search and discover that a Toyota RAV 4 can only have a FWD drivetrain. Then the turker will be able to declare the second tuple as the correct option with high confidence.

The results of this experiment are shown in Table 7.1. As I can see, the users consistently picked the tuples cleaned by BayesWipe more favorably compared to the original dirty tuples, proving that it is indeed effective in real-world datasets. Notice that it is not trivial to obtain a 56% rate of success in these experiments. Finding a tuple which convinces the turkers that it is better than the original requires searching through a huge space of possible corrections. An algorithm that picks a possible correction

[2] http://www.cars.com

Confidence	BayesWipe	Original
High confidence only	56.3%	43.6%
All confidence values	53.3%	46.7%

Table 7.1: Results of the Mechanical Turk Experiment, showing the percentage of tuples for which the users picked the results obtained by BayesWipe as against the original tuple.

...	make	model	cartype	fueltype	engine	transmission	drivetrain	doors	wheelbase
Car:	toyota	rav4 base	suv	gasoline	2.5l i4 16v mpfi dohc	4-speed automatic	**4wd**	4	105"
Car:	toyota	rav4 base	suv	gasoline	2.5l i4 16v mpfi dohc	4-speed automatic	**fwd**	4	105"

○ First is correct
○ Second is correct

How confident are you about your selection?
○ Very confident ○ Confident ○ Slightly confident ○ Maybe confident ○ Not confident

Figure 7.4: A fragment of the questionnaire provided to the Mechanical Turk workers.

randomly from this space is likely to get a near 0% accuracy.

The first row of Table 7.1 shows the fraction of tuples for which the turkers picked the version cleaned by BayesWipe and indicated that they were either 'very confident' or 'confident'. The second row shows the fraction of tuples for all turker confidence values, and therefore is a less reliable indicator of success.

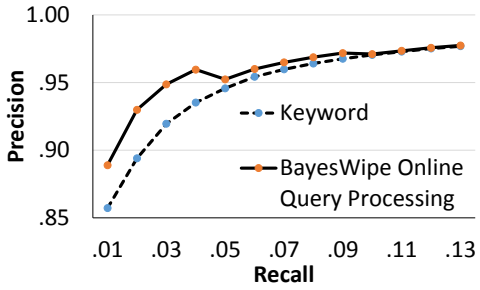


Figure 7.5: Average precision vs recall, 20% noise.

Online Query Processing: Since there is no existing work on querying autonomous data sources in the presence of data inconsistency, I consider a **keyword query** system as my baseline. I evaluate the precision and recall of my method against the ground truth and compare it with the baseline system.

I issued randomly generated queries to both BayesWipe and the baseline system. Figure 7.5

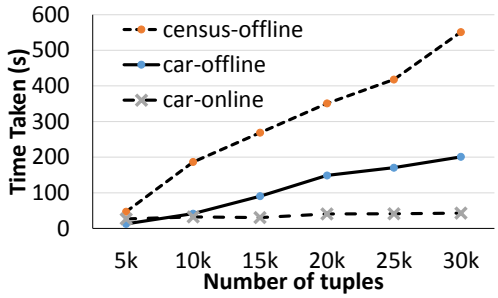


Figure 7.6: Time vs. #Tuples

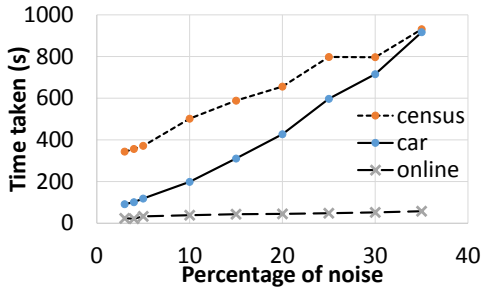


Figure 7.7: Time vs. %Noise

shows the average precision over 10 queries at various recall values. It shows that my system outperforms the keyword query system in precision, especially since my system considers the relevance of the results when ranking them. On the other hand, the keyword search approach is oblivious to ranking and returns all tuples that satisfy the user query. Thus it may return irrelevant tuples early on, leading to a loss in precision. This shows that my proposed query ranking strategy indeed captures the expected relevance of the to-be-retrieved tuples, and the query rewriting module is able to generate the highly ranked queries.

Efficiency: In Figure 7.6 I evaluate the time taken as the number of tuples in the database increases, and in Figure 7.7 I show the time taken as the noise varies. These graphs show that both the offline and online modes of BayesWipe complete in a reasonable time. In particular, this shows that the query processing mode shows the time taken remains mostly constant under both varying database size as well as noise. This is because the most expensive parts of the query processing algorithm (determining the error and data generative models) are precomputed offline. This shows that BayesWipe can be used as a viable tool for large-scale web-data.

Chapter 8

PROPOSED WORK

8.1 Expanded user studies with real data

Motivation: As noted in the experiment chapter (Chapter 7), validation of the BayesWipe system against real data has a number of challenges, mostly because ground truth is not available when the noise is real. To obtain results on data that has no ground truth, experiments have already been performed on Amazon mechanical turk, where humans were asked to evaluate the accuracy of the correction made by Kydd. I propose to further expand these experiments and make them much more robust. In short, the challenges faced by the current experiments are the following:

(1) Checking the capability of the turkers: Currently, there are no safeguards in place to check if the responses by the turkers are correct. In the future, if the turkers realize that there is no penalty for wrong responses, they may quickly enter random responses in order to maximize the amount they earn.

(2) Evaluating factors other than accuracy: In addition to the accuracy, the other factors that contribute to the quality of a system are speed, user-friendliness and overall utility. I propose to perform user studies where BayesWipe is evaluated for these parameters also.

Proposed solution:

(1) There are three ways in which the responses from turkers may be wrong: (i) the turker is knowledgable about the domain (e.g. used cars) but made an unfortunate mistake. (ii) the turker is not knowledgable about the domain, and makes best-effort guesses which are wrong. (iii) the turker disregards the questions entirely, and randomly clicks responses in order to make a lot of money in a short amount of time. Some of these problems are well-known [28, 18]. Working within the constraints of Amazon Mechanical Turk (AMT), there are two ways in which this problem can be solved: (i) creating a 'qualification' test, in which a set of questions will be prepared and a turker can achieve the qualification by solving the questions correctly. This qualification is permanent, and all future responses from the turker will be considered trusted. (ii) Each HIT will be interspersed with multiple questions for which the answer is already known. In order for the turker to be paid for that particular HIT, she must have correctly answered a minimum percentage of these known questions. The fact that this restriction is present will be made clear to the turkers upfront, thus discouraging people from randomly choosing responses.

The disadvantage of using the first solution (using qualification test), is that the turkers have to go through an additional step (passing the qualification test) before they can attempt any HITs. On the

other hand, using the second method (known questions in each HIT) is more expensive, because I have to insert dummy questions in the space where legitimate questions could have been put (assuming I keep the amount paid per question constant).

(2) In order to test the other factors of the system, I can perform a more traditional user-study, where I create a web-service using the BayesWipe codebase, and ask users to provide feedback from using the website. I will compare the satisfaction of the turkers using BayesWipe to the satisfaction of the users using a baseline system (say, a keyword query system) to evaluate the performance of BayesWipe in it's various facets (speed, user-friendliness and utility).

8.2 Extended error model

In this proposal, I presented an error model that was a mixture of two components: an edit-distance and a distributional similarity. As BayesWipe is applied to a variety of data sources, for the best accuracy, the error model will need to be tweaked to match the true error process of that particular data source. I propose to test this feature of BayesWipe by applying it to a data source where the error model is quite different. As a prototype example, consider the dataset crawled from cars.com. In addition to typos, missing values, and substituted values, it was observed that it contained many additional words in certain fields. For example, a tuple contained the following string in the 'engine' field: `3.2L V629 MPG!SUNROOF SPOILERLOADED+SHARP!`. It is clear that the author of this entry has tried to insert more information into this attribute. It is also clear that the current error models will not be able to detect the true engine '3.2L V6' with too much accuracy, since both the edit distance and the distance using distributional similarity are high. Therefore, a third error component can be proposed for this, where a substring match is given high scores. I propose to implement this error model and evaluate the increase in accuracy of the system.

8.3 Explanations

Motivation: One way of looking at the BayesWipe system is to consider it to be a recommender system for tuples — based on the query provided by the user, BayesWipe recommends the tuples that seemingly do not satisfy the constraints of the query, but that would satisfy them if cleaned properly. It has been established that providing explanations for the recommendations that are suggested by an automated recommender systems is the right thing to do [33]. In the case of BayesWipe an explanation system solves two important issues:

(1) Display of the tuple to the user: In the experiments for the online version of BayesWipe, the criteria for success was getting the correct tuple-id in response to a query. However, when I display a

tuple to the user, I am faced with a choice: I can either show the tuple as it originally existed in the (dirty) database, thus giving the user an opportunity to see the raw data and make an inference for herself, but confusing her as to how the tuple is relevant to the query; or I can show the cleaned version of the tuple, which shows why the tuple was a possible solution to the query, but might confuse the user in case the system over-corrected any attribute. Having an explanation system solves the problem, by allowing me to display the original, raw tuple followed by an explanation of why it was returned.

(2) Providing confidence to the user: In addition to solving the display problems, explanations also increase user confidence in the system, since they can easily distinguish tuples that were corrected by the system from tuples that were left untouched, and also use their own background knowledge to identify explanations that don't make sense, thus finding tuples that should not have been returned for that query.

Proposed solutions: In order to provide explanations, the first idea is to simply show the percentage confidence of the system next to the original tuple from the dirty dataset without alluding to the how the system decided that the tuple was relevant to the query. This has the advantage of not confusing the user or overloading her with too much information, but on the other hand does not provide much meaningful information for the user to make an informed choice about the tuple.

A second idea is to show the original tuple first, followed by the most likely cleaned version along with the confidence in the correction. However, this approach is not ideal since the score of a tuple is built on the contribution of all the possible clean options that match the query — a single option may contribute very little to the score.

The third idea is to explain the reasoning behind the system's choice for returning the tuple. This can be done in two ways:

(1) I can learn a set of deterministic constraints (such as CFDs or INDs) based on the cleaned version of the data — either directly induced from the Bayes network, or by running a constraint learning algorithm on the cleaned version of the data. I can then pick the set of correct options that satisfy both the query and the learned constraints, and display them to the user along with a set of explanations derived from the constraints [27, 16].

(2) I can use Bayesian sensitivity analysis to directly infer the most important nodes that contributed to a particular inference. For example, if a particular attribute value was changed by the system, then using sensitivity analysis, it is possible to figure out which other values in the tuple contributed to the change. This information can be used as an explanation [26].

8.4 Query imprecision

There are two components to handling errors in an information retrieval scenario: the error due the corruptions in the data, and the errors due to corruption in the query. While this proposal dealt mostly with corruptions in data, similar principles can also be applied to the query side. Handling query imprecision allows me to find and fix issues with the query in the following ways:

(1) Fixing typos: Just as typos occur in the data, they can also occur in the query. As I did for the data, I can create a generative and error model for the query as well. Ideally, this model will be learned from the query logs and click logs. In the absence of query logs the data generative and error model can be substituted as an approximation for the query models.

(2) Phrases: When certain words occur together, their meaning may be changed. For example, if a query 'Honda Civic Malibu' is entered, it probably means that the user is searching for a Honda Civic car near the location Malibu, California. On the other hand, if the query entered is 'Chevrolet Malibu', then the most likely meaning of the query is a car with Make 'chevrolet' and model 'Malibu' (since people often describe a car by their make and model). The system can adjust its rewritten queries to take this into account.

(3) Suggested queries: Often when the user searches for a particular car, she may not be interested in just that make and model, but also all other cars that are similar to the one she searched for. For example, if the query is 'civic', then it is likely that a 'corolla' would also be a satisfactory car for the user, since these cars are both similarly priced and have similar attributes. This information can be used to recommend more tuples or queries to the user.

8.5 Schedule

July – September 2013: User studies. I will extend the current user-study experiments to incorporate checks for user honesty. I will test both the methods mentioned above (qualification test and known answers) to determine which method to use. In addition to testing accuracy, I will also test the overall quality of the results against a naïve baseline system.

October – November 2013: Extended error model. I will identify a dataset that cannot be cleaned effectively using the current error model. If such a dataset cannot be found, I will use the current car dataset that has real noise and attempt to improve the accuracy. In these experiments, I will need to come up with an extended error model that uses additional signals in addition to edit distance and distributional similarity. The parameters for these signals will also be learned. I will thoroughly test the error model

using this dataset.

December 2013 – January 2014: Explanations. I will create a user friendly module that shows explanations for the results of BayesWipe. This includes creating a UI for the results as well as the algorithms for generating the results. I will also create simpler baseline systems for generating explanations and show the effectiveness of the explanations module by performing user-studies on Amazon Mechanical Turk.

February 2014: Journal paper. I will write a paper that incorporates the advances made since this proposal and write a journal paper, targeting either VLDB journal or TKDE.

March – April 2014: Query imprecision. I will implement query imprecision based on the model learned from the data itself. In the course of the previous experiments, if sufficient query logs are collected, then I will learn the query imprecision model from those query logs, as described above.

April – May 2014: Dissertation I will write my dissertation and then schedule and defend my thesis.

Chapter 9

CONCLUSION

In this paper I presented a novel system, BayesWipe that works using end-to-end probabilistic semantics, and without access to clean master data. This system can be used in two modalities: (1) offline data cleaning, an *in situ* rectification of data and (2) online query processing mode, a highly efficient way to obtain clean query results over inconsistent data. I empirically showed that BayesWipe outperformed existing baseline techniques in quality of results, and was highly efficient.

REFERENCES

[1] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 68–79. ACM, 1999. 4, 6

[2] C. R. Association. Challenges and opportunities with big data. http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf, 2012. 1, 13

[3] A. Asuncion and D. Newman. Uci machine learning repository, 2007. 20

[4] A. Berger, V. Pietra, and S. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 1996. 11

[5] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *International Conference on Database Theory*, 2011. 1

[6] G. Beskales. Modeling and querying uncertainty in data cleaning, 2012. 5

[7] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*. IEEE, 2013. 4

[8] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. Sampling from repairs of conditional functional dependency violations. *The VLDB Journal*, pages 1–26, 2013. 4

[9] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005. 1, 4

[10] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 746–755. IEEE, 2007. 4, 5, 20

[11] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD*, pages 891–893, 2005. 13

[12] F. Chiang and R. Miller. Discovering data quality rules. *Proceedings of the VLDB Endowment*, 2008. 20

[13] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd international conference on Very large data bases*, pages 315–326. VLDB Endowment, 2007. 4, 20

[14] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, volume 30, pages 864–875. VLDB Endowment, 2004. 13

[15] T. Dasu and J. M. Loh. Statistical distortion: Consequences of data cleaning. *VLDB*, 5(11):1674–1683, 2012. 5

[16] I. Davidson and Z. Qi. Finding alternative clusterings using constraints. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 773–778. IEEE, 2008. 28

[17] X. L. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. *VLDB*, 2(1):562–573, 2009. 5

[18] J. S. Downs, M. B. Holbrook, S. Sheng, and L. F. Cranor. Are your participants gaming the system?: screening mechanical turk workers. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2399–2402. ACM, 2010. 26

[19] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems (TODS)*, 33(2):6, 2008. 4

[20] W. Fan, F. Geerts, L. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. Ieee, 2009. 1, 4

[21] W. Fan, F. Geerts, N. Tang, and W. Yu. Inferring data currency and consistency for conflict resolution. In *ICDE*. IEEE, 2013. 1

[22] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment*, 2010. 4

[23] I. Fellegi and D. Holt. A systematic approach to automatic edit and imputation. *Journal of the American Statistical association*, pages 17–35, 1976. 1

[24] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD*, pages 155–166. ACM, 2005. 2

[25] A. Hartemink. Banjo: Bayesian network inference with java objects. Available: http://www.cs.duke.edu/ amink/software/banjo. 9

[26] F. V. Jensen, S. H. Aldenryd, and K. B. Jensen. Sensitivity analysis in bayesian networks. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 243–250. Springer, 1995. 28

[27] L. Kagal, C. Hanson, and D. Weitzner. Using dependency tracking to provide explanations for policy management. In *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*, pages 54–61. IEEE, 2008. 28

[28] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 453–456. ACM, 2008. 26

[29] E. Knorr, R. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3):237–253, 2000. 1

[30] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 802–803. ACM, 2006. 9

[31] J. Kubica and A. Moore. Probabilistic noise identification and data cleaning. In X. Wu, A. Tuzhilin, and J. Shavlik, editors, *The Third IEEE International Conference on Data Mining*, pages 131–138. IEEE Computer Society, November 2003. 5

[32] M. Li, Y. Zhang, M. Zhu, and M. Zhou. Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational*

Linguistics and the 44th annual meeting of the Association for Computational Linguistics, pages 1025–1032. Association for Computational Linguistics, 2006. 10

[33] D. McSherry. Explanation in recommender systems. *Artificial Intelligence Review*, 24(2):179–197, 2005. 27

[34] T. Minka, W. J.M., J. Guiver, and D. Knowles. Infer.NET 2.4, 2010. Microsoft Research Cambridge. http://research.microsoft.com/infernet. 9

[35] L. Mo, R. Cheng, X. Li, D. Cheung, and X. Yang. Cleaning uncertain data for top-k queries. In *ICDE*. IEEE, 2013. 6

[36] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *ACM SIGMOD Record*, volume 28, pages 455–466. ACM, 1999. 5

[37] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausble Inference*. Morgan Kaufmann Pub, 1988.

[38] T. Redman. The impact of poor data quality on the typical enterprise. *Communications of the ACM*, 1998. 4

[39] E. Ristad and P. Yianilos. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1998. 10

[40] D. Suciu and N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD*, volume 14, pages 963–963, 2005. 14

[41] G. Wolf, A. Kalavagattu, H. Khatri, R. Balakrishnan, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati. Query processing over incomplete autonomous databases: query rewriting using learned data dependencies. *The VLDB Journal*, 2009. 2, 5

[42] H. Xiong, G. Pandey, M. Steinbach, and V. Kumar. Enhancing data analysis with noise removal. *Knowledge and Data Engineering, IEEE Transactions on*, 2006. 1

[43] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *VLDB*, 4(5):279–289, 2011. 4