BayesWipe: A Multimodal System for Data Cleaning and Consistent Query Answering on Structured Data

Sushovan De[1] Yuheng Hu[2] Yi Chen[3] Subbarao Kambhampati[4]
[1,2,4]Computer Science and Engg, Arizona State University, Tempe, AZ
[3]School of Business, New Jersey Instt. of Technology, Newark, NJ
[1]sushovan@asu.edu, [2]yuhenghu@asu.edu, [3]yi.chen@njit.edu, [4]rao@asu.edu

ABSTRACT

Recent efforts in data cleaning have focused mainly on problems like data deduplication, record matching, and data standardization; none of these focus on fixing incorrect attribute values in tuples. Correcting values in tuples is typically performed by a minimum cost repair of tuples that violate static constraints like CFDs (which have to be provided by domain experts, or learned from a clean sample of the database). In this paper, we provide a method for correcting individual attribute values in a structured database using a Bayesian generative model and a statistical error model learned from the noisy database directly. We thus avoid the necessity for a domain expert or master data. We also show how to efficiently perform consistent query answering using this model over a dirty database, in case write permissions to the database are unavailable. We evaluate our methods over both synthetic and real data.

1. INTRODUCTION

Although data cleaning has been a long standing problem, it has become critically important again because of the increased interest in web data and big data. Among these, the need to efficiently handle structured data that is rife with inconsistency and incompleteness is also more significant than ever. Indeed multiple studies, such as [1] 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.

A variety of data cleaning approaches have been proposed over the years, from traditional methods (e.g., outlier detection [2], noise removal [3], entity resolution [4, 3], and imputation[5]) 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., [6, 7, 8]) attempt to clean data by exploiting patterns in the data, which they express in the form of conditional functional dependencies (or CFDs). 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 [9] 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, our experimental evaluation shows that even a small amount of noise severely impairs the ability to learn useful constraints from the data.

To avoid dependence on clean master data, in this paper, we propose a novel system called BayesWipe that assumes that a statistical process underlies the generation of clean data (which we call the *data source model*) as well as the corruption of data (which we call the data *error model*). The noisy data itself is used to learn the generative and error model parameters, 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. We model the data source model through a Bayesian network, and the error process as a mixture of error features (to handle typo-related errors, substitution errors, and omission errors). We make the assumption that errors are generated independently in all the attributes.

We 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 online query processing mode of BayesWipe is motivated by big 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 we retrieve it, which also provides us the opportunity of improving the efficiency of the system, since we 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
BUDA 2014 Snowbird, Utah, USA
Copyright 2014 ACM ...$15.00.

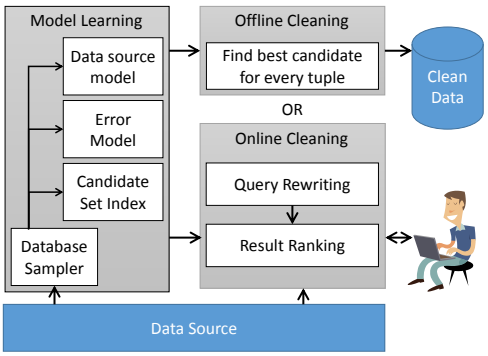


Figure 1: The architecture of BayesWipe. Our framework learns both data source model and error model from the raw data during the model learning phase. It can perform offline cleaning or query processing to provide clean data.

significant extension of the QPIAD system for handling data incompleteness [10].

To summarize our contributions, we:

- Propose that data cleaning should be done using a principled, probabilistic approach.
- Develop a novel algorithm following those principles, which uses a Bayes network as the generative model and maximum entropy as the error model of the data.
- Develop novel query rewriting techniques so that this algorithm can also be used in an online scenario.
- Empirically evaluate the performance of our algorithm using both controlled and real datasets.

The rest of the paper is organized as follows. We begin by describing the architecture of BayesWipe in the next section, where we also present the overall algorithm. Section 3 describes the learning phase of BayesWipe, where we find the generative and error models. Section 4 describes the offline cleaning mode, and the next section details the query rewriting and online data processing. We describe the results of our empirical evaluation in Section 6, and then conclude by summarizing our contributions.

2. 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 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. Given a candidate replacement set $\mathcal{C}$ for a possibly corrupted tuple T in $\mathcal{D}$, we can clean the database by replacing T with the candidate clean tuple $T^* \in \mathcal{C}$ that has the maximum $\mathbf{Pr}(T^*|T)$. Using Bayes rule (and dropping the common denominator), we can rewrite this to

$$T^*_{best} = \arg\max[\mathbf{Pr}(T|T^*)\mathbf{Pr}(T^*)] \quad (1)$$

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

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

$$Score(T) = \sum_{T^* \in \mathcal{C}} \underbrace{\mathbf{Pr}(T^*)}_{\text{source model}} \underbrace{\mathbf{Pr}(T|T^*)}_{\text{error model}} \underbrace{R(T^*|Q^*)}_{\text{relevance}} \quad (2)$$

In this paper, we 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 ensures that those tuples whose corrected versions are relevant to the query are ranked higher than the tuples that are clean, but irrelevant.

Architecture: Figure 1 shows the system architecture for BayesWipe. During the model learning phase (Section 3), we first obtain a sample database by sending some queries to the database. On this sample data, we learn the generative model of the data as a Bayes network (Section 3.1). In parallel, we define and learn an error model which incorporates common kinds of errors (Section 3.2). We also create an index to quickly propose candidate T^*s.

We can then choose to do either offline cleaning (Section 4) or online query processing (Section 5), as per the scenario. In the offline cleaning mode, we can choose whether to store the resulting cleaned tuple in a deterministic database (where we store only the T^* with the maximum posterior probability) or probabilistic database (where we store the entire distribution over the T^*). In the online query processing mode, we obtain a query from the user, and do query rewriting in order to find a set of queries that are likely to retrieve a set of highly relevant tuples. We execute these queries and re-rank the results, and then display them.

Algorithm 1: The algorithm for offline data cleaning

```
Input: D, the dirty dataset.
BN ← Learn Bayes Network (D)
foreach Tuple T ∈ D do
    C ← Find Candidate Replacements (T)
    foreach Candidate T* ∈ C do
        P(T*) ← Find Joint Probability (T*, BN)
        P(T|T*) ← Error Model (T, T*)
    end
    T ← arg max_{T*∈C} P(T*)P(T|T*)
end
```

In Algorithms 1 and 2, we present the overall algorithm for BayesWipe. In the offline mode, we show how we iterate over all the tuples in the dirty database, D and replace them with cleaned tuples. In the query processing mode, the first three operations are performed offline, and the remaining operations show how the tuples are efficiently retrieved from the database, ranked and displayed to the user.

3. MODEL LEARNING

This section details the process by which we estimate the components of Equation 2: the data source model $\mathbf{Pr}(T^*)$ and the error model $\mathbf{Pr}(T|T^*)$

3.1 Data Source Model

The data that we work with can have dependencies among various attributes (e.g., a car's *engine* depends on its *make*).

Algorithm 2: Algorithm for online query processing.

Input: D, the dirty dataset
Input: Q, the user's query
$S \leftarrow$ Sample the source dataset D
$BN \leftarrow$ Learn Bayes Network (S)
$ES \leftarrow$ Learn Error Statistics (S)
$R \leftarrow$ Query and score results (Q, D, BN)
$ESQ \leftarrow$ Get expanded queries (Q)
foreach *Expanded query* $E \in ESQ$ **do**
  $R \leftarrow R \cup$ Query and score results (E, D, BN)
  $RQ \leftarrow RQ \cup$ Get all relaxed queries (E)
end
$Sort(RQ)$ by expected relevance, using ES
while *top-k confidence not attained* **do**
  $B \leftarrow$ Pick and remove top RQ
  $R \leftarrow R \cup$ Query and score results (B, D, BN)
end
$Sort(R)$ by score
return R

Therefore, we 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 tuples. We learn both the structure and the parameters of the network using existing state of the art tools, Banjo [11] and Infer.NET [12]. For a given T^* tuple, $P(T^*)$ can then be evaluated by a constant-time computation, using the conditional probability tables of the learned Bayes network.

3.2 Error Model

The error model $\mathbf{Pr}(T|T^*)$ is also estimated from noisy data. There are many types of errors that can occur in data. We focus on the most common types of errors that occur in data that is manually entered by naïve users: typos, deletions, and substitution of one word with another. We also make an additional assumption that error in one attribute does not affect the errors in other attributes. This is a reasonable assumption to make, since we are allowing the data itself to have dependencies between attributes, while only constraining the error process to be independent across attributes. With these assumptions, we are able to come up with a simple and efficient error model, where we combine the three types of errors using a maximum entropy model.

Given a set of clean candidate tuples $\mathcal{C}$ where $T^* \in \mathcal{C}$, our error model $\mathbf{Pr}(T|T^*)$ essentially measures how clean T is, or in other words, how similar T is to T^*.

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 we use the Levenshtein distance, which uses a uniform cost function. This gives us a distance, which we then convert to a probability using [13]:

$$f_{ed}(T_{A_i}, T^*_{A_i}) = \exp\{-cost_{ed}(T_{A_i}, T^*_{A_i})\} \quad (3)$$

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. We 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 [14]. Formally, for each string T_{A_i} and $T^*_{A_i}$, we have:

$$f_{ds}(T_{A_i}, T^*_{A_i}) = \sum_{c \in C(T_{A_i}, T^*_{A_i})} \frac{\mathbf{Pr}(c|T^*_{A_i})\mathbf{Pr}(c|T_{A_i})\mathbf{Pr}(T_{A_i})}{\mathbf{Pr}(c)} \quad (4)$$

where $C(T_{A_i}, T^*_{A_i})$ is the context of an 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. We 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, we use the Laplace smoothing factor (μ).

Unified error model: We 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, we use the well-known maximum entropy framework [15]. For each attribute of the input tuple T and T^*, we have the unified error model $\mathbf{Pr}(T|T^*)$ defined as follows:

$$\mathbf{Pr}(T|T^*) = \frac{1}{Z} \exp\left\{\alpha \sum_{i=1}^{m} f_{ed}(T_{A_i}, T^*_{A_i}) + \beta \sum_{i=1}^{m} f_{ds}(T_{A_i}, T^*_{A_i})\right\} \quad (5)$$

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

3.3 Finding the Candidate Set

The set of candidate tuples, $\mathcal{C}(T)$ for a given tuple T are the possible replacement tuples that the system scores as corrections to T. The larger the set $\mathcal{C}$ is, the longer it will take for the system to perform the cleaning. If $\mathcal{C}$ contains many unclean tuples, then the system will waste time scoring tuples that are inherently unclean.

An efficient approach to finding a reasonably clean $\mathcal{C}(T)$ is to consider the set of all the tuples in the sample database that differ from T in not more than j attributes. An efficient method is to creating $j+1$ indices, and checking only candidate tuples that are retrieved by an exact lookup from these indices.

4. OFFLINE CLEANING

In order to clean the data *in situ*, we first use the techniques of the previous section to learn the data source model, the error model and create the index. Then, we iterate over all the tuples in the database and use Equation 1 to find the T^* with the best score. We then replace the tuple with that T^*, thus creating a deterministic database using the offline mode of BayesWipe.

Setting the parameter Recall from Section 3.2 that there are parameters in the error model called α and β, which need to be set. Interestingly, in addition to controlling the relative weight given to the various features in the error model, these parameters can be used to control overcorrection by the system.

Overcorrection: Any data cleaning system is vulnerable to overcorrection, where a legitimate tuple is modified by the system to an unclean value. Overcorrection can have many causes. In a traditional, deterministic system, overcorrection can be caused by erroneous rules learned from infrequent data. For example, certain makes of cars are all owned by the same conglomerate (GM owns Chevrolet). In a misguided attempt to simplify their inventory, a car salesman might list all the cars under the name of the conglomerate. This might provide enough support for the system to learn the wrong rule (Malibu $\rightarrow$ GM).

Typically, once an erroneous rule has been learned, there is no way to correct it or ignore it without a lot of oversight from domain experts. However, BayesWipe provides a way to regulate the amount of overcorrection in the system with the help of a 'degree of change' parameter. Without loss of generality, we can rewrite Equation 5 to the following:

$$\mathbf{Pr}(T|T^*) = \frac{1}{Z}\exp\left\{\gamma\Big(\delta\sum_{i=1}^{m} f_{ed}(T_{A_i}, T^*_{A_i}) + (1-\delta)\sum_{i=1}^{m} f_{ds}(T_{A_i}, T^*_{A_i})\Big)\right\}$$

where the parameters α and β have been replaced by δ and $(1-\delta)$, since we are only interested in their relative weights. To make the this transformation, a normalization constant, γ has been pulled out of the bracket. This parameter, γ, can be used to modify the degree of variation in $\mathbf{Pr}(T|T^*)$. High values of γ imply that small differences in T and T^* cause a larger difference in the value of $\mathbf{Pr}(T|T^*)$, causing the system to give higher scores to the original tuple (compared to a modified tuple).

5. QUERY REWRITING FOR ONLINE QUERY PROCESSING

In this section we extend the techniques of the previous section so that it can be used in 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. We show our query rewriting mechanisms aimed at addressing both.

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

5.1 Increasing the precision of rewritten queries

Since our data sources are inherently noisy, it is important that we 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. We 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, we 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 generated from the user's query are called ESQs.

Each user query Q^* is a select query with one or more attribute-value pairs as constraints. In order to create an ESQ, we will have to add highly correlated constraints to Q^*.

Searching for correlated constraints to add requires Bayesian inference, which is an expensive operation. Therefore, when searching for constraints to add to Q^*, we restrict the search to the union of all the attributes in the Markov blanket [16]. The Markov blanket of an attribute comprises its children, its parents, and its children's other parents. It is the set of attributes whose value being given, the node becomes independent of all other nodes in the network. Thus, it makes sense to consider these nodes when finding correlated attributes. This correlation is computed using the Bayes Network that was learned offline on a sample database (recall the architecture of BayesWipe in Figure 1.)

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

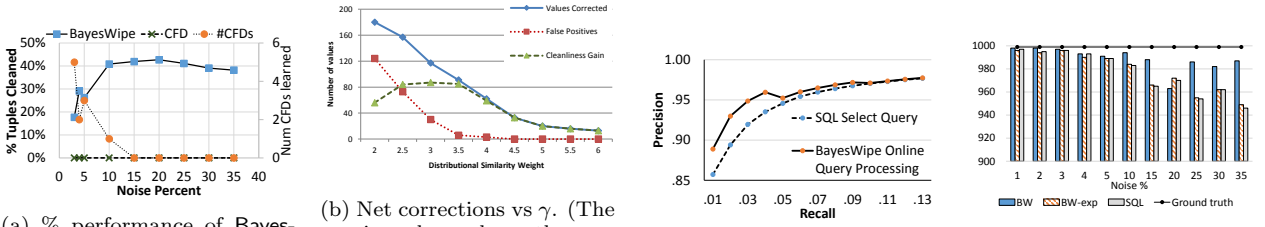
Note that if there are m attributes, each of which can take n values, then the total number of possible ESQs is n^m. Searching for the ESQ that globally maximizes the objectives in this space is infeasible; we therefore approximately search for it by performing a heuristic-informed search. Our objective is to create an ESQ with m attribute-value pairs as constraints. We begin with the constraints specified by the user query Q^*. We 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. We 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 $\mathbf{Pr}(Q)^{m/p}$. This represents the expected joint probability of Q when expanded to m attributes, assuming that all the constraints will have the same average posterior probability. We expand them further, until we find k queries of size m with the highest probabilities.

5.2 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, we 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 can only be retrieved by relaxed queries that do not specify a value for those attributes. Instead, we 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.

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

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



(a) % performance of BayesWipe compared to CFD, for the used-car dataset.

(b) Net corrections vs γ. (The x-axis values show the unnormalized distributional similarity weight, which is simply $\gamma \times 3/5$.)

(c) Average precision vs recall

(d) Improvement in data quality by running BayesWipe

Figure 2: Effectiveness of offline (a, b) and online (c, d) cleaning of BayesWipe

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

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

5.3 Terminating the process

We begin by looking at all the RQs in descending order of their rank. If the current k-th tuple in our resultset has a relevance of λ, and the estimated rank of the Q we are about to execute is $R(T_q|Q)$, then we 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 we have the true top-k resultset with a probability $\mathcal{P}$.

6. EMPIRICAL EVALUATION

We quantitatively study the performance of BayesWipe in both modes — offline, and online, and compare it against state-of-the-art CFD approaches. We used three real datasets spanning two domains: used car data (one with controlled synthetic noise, and one with real noise) and census data.

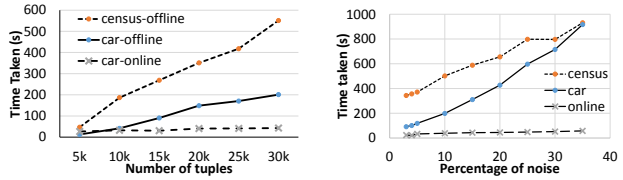
6.1 Experiments

Offline Cleaning Evaluation: In Figure 2a, we compare BayesWipe against CFDs [17]. The dotted line that shows the number of CFDs learned from the noisy data quickly falls to zero, which is not surprising: the algorithm to learn CFDs was not designed to tolerate noise in the dataset — as a result any constraints that are violated by even a single tuple in the dataset are not learnt. As a result, the CFD is unable to clean any tuples, because in order for a tuple to be cleaned, it needs to violate some constraints. On the other hand, BayesWipe is able to clean between 20% to 40% of the incorrect values. It is interesting to note that the percentage of tuples cleaned increases initially and then slowly decreases, because for very low values of noise, there aren't enough errors 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.

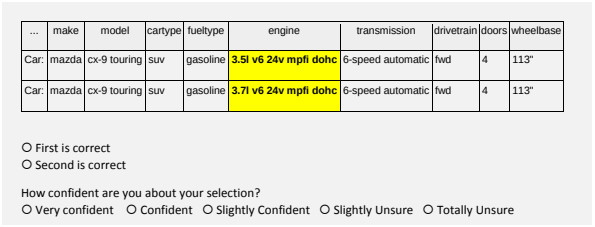
Setting γ: As explained in Section 4, the weight given to the edit distance (δ) compared to the weight given to the distributional similarity ($1 - \delta$); and the overcorrection parameter (γ) are parameters that can be tuned, and should be set based on which kind of error is more likely to occur. In our experiments, we performed a grid search to determine the best values of δ and γ to use. In Figure 2b, we vary γ, keeping $\delta = 2/5$.

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, our 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 γ 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 we set the parameter values too low, we will correct most wrong tuples in the input dataset, but we 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, we picked a parameter value of $\delta = 0.638, \gamma = 5.8$ and kept it constant for all our experiments. However, note that even if the parameters are not set optimally, the system still performs useful data cleaning.



(a) Time vs. #Tuples (b) Time vs. %Noise



(c) A fragment of the questionnaire provided to the Mechanical Turk workers.

Figure 3: Performance evaluations, and user study questionnaire

Online Query Processing: While in the offline mode, we had the luxury of changing the tuples in the database itself, in online query processing, we use query rewriting to obtain a resultset that is similar to the offline results, without modification to the database. We consider a SQL select query system as our baseline. We evaluate the precision and recall of our method against the ground truth and compare it with the baseline, using randomly generated queries.

We issued randomly generated queries to both BayesWipe and the baseline system. Figure 2c shows the average precision over 10 queries at various recall values. It shows that our system outperforms the SQL select query system in top-k precision, especially since our system considers the relevance of the results when ranking them. On the other hand, the SQL 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.

Figure 2d shows the improvement in the absolute numbers of tuples returned by the BayesWipe system. The graph shows the number of true positive tuples returned (tuples that match the query results from the ground truth) minus the number of false positives (tuples that are returned but do not appear in the ground truth result set). We also plot the number of true positive results from the ground truth, which is the theoretical maximum that any algorithm can achieve. The graph shows that the BayesWipe system outperforms the SQL query system at nearly every level of noise. Further, the graph also illustrates that — compared to a SQL query baseline — BayesWipe closes the gap to the maximum possible number of tuples to a large extent. In addition to showing the performance of BayesWipe against the SQL query baseline, we also show the performance of BayesWipe without the query relaxation part (called BW-exp[1]). We can see that the full BayesWipe system outperforms the BW-exp system significantly, showing that query relaxation plays an important role in bringing relevant tuples to the resultset, especially for higher values of noise.

This shows that our 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 3a we evaluate the time taken as the number of tuples in the database increases, and in Figure 3b we 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 generative models) are precomputed offline. This shows that BayesWipe can be used as a viable tool for large-scale web-data.

[1]BW-exp stands for BayesWipe-expanded, since the only query rewriting operation done is query expansion.

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

Table 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.

Evaluation on real data with naturally occurring errors: In this section we 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, we do not have ground truth for this data. To evaluate this system, we conducted an experiment on Amazon Mechanical Turk. First, we ran the offline mode of BayesWipe on the entire database. We 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 3c. 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 Mazda CX-9 touring is only available in a 3.7l engine, not a 3.5l. 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 1. As we 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 randomly from this space is

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

likely to get a near 0% accuracy.

The first row of Table 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.

7. CONCLUSION

In this paper we presented a novel system, BayesWipe that works using end-to-end probabilistic semantics, and without access to clean master data. We showed how to effectively learn the data source model as a Bayes network, and how to model the error as a mixture of error features. We showed the operation of this system 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. We empirically showed that BayesWipe outperformed existing baseline techniques in quality of results, and was highly efficient. We also showed the performance of the BayesWipe system at various stages of the query rewriting operation. User experiments showed that the system is useful in cleaning real-world noisy data.

Acknowledgments

The authors would like to thank Dr. K. Selçuk Candan for his valuable advice. This research is supported in part by the ONR grants N00014-13-1-0176, N0014-13-1-0519, ARO grant W911NF-13-1-0023 and a Google Research Grant.

8. REFERENCES

[1] Computing Research Association, "Challenges and opportunities with big data," http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf, 2012.

[2] E. Knorr, R. Ng, and V. Tucakov, "Distance-based outliers: algorithms and applications," *The VLDB Journal*, vol. 8, no. 3, pp. 237–253, 2000.

[3] H. Xiong, G. Pandey, M. Steinbach, and V. Kumar, "Enhancing data analysis with noise removal," *TKDE*, vol. 18, no. 3, pp. 304–319, 2006.

[4] P. Singla and P. Domingos, "Entity resolution with markov logic," in *ICDM*. IEEE, 2006, pp. 572–582.

[5] I. Fellegi and D. Holt, "A systematic approach to automatic edit and imputation," *J. American Statistical association*, pp. 17–35, 1976.

[6] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *SIGMOD*. ACM, 2005.

[7] W. Fan, F. Geerts, L. Lakshmanan, and M. Xiong, "Discovering conditional functional dependencies," in *ICDE*. IEEE, 2009.

[8] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan, "Data cleaning and query answering with matching dependencies and matching functions," in *ICDT*, 2011.

[9] A. Fuxman, E. Fazli, and R. J. Miller, "Conquer: Efficient management of inconsistent databases," in *SIGMOD*. ACM, 2005, pp. 155–166.

[10] 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.

[11] A. Hartemink., "Banjo: Bayesian network inference with java objects." http://www.cs.duke.edu/~amink/software/banjo.

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

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

[14] M. Li, Y. Zhang, M. Zhu, and M. Zhou, "Exploring distributional similarity based models for query spelling correction," in *ICCL*. Association for Computational Linguistics, 2006, pp. 1025–1032.

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

[16] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

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