Data Cleaning for Decision Support

Michael Benedikt Philip Bohannon Glenn Bruns

Bell Laboratories, Lucent Technologies

Abstract

Data cleaning may involve the acquisition, at some effort or expense, of high-quality data. Such data can serve not only to correct individual errors, but also to improve the reliability model for data sources. However, there has been little research into this latter role for acquired data. In this short paper we define a new data cleaning model that allows a user to estimate the value of further data acquisition in the face of specific business decisions. As data is acquired, the reliability model of sources is updated using Bayesian techniques, thus aiding the user in both developing reasonable probability models for uncertain data and in improving the quality of that data. Although we do not deal here with the problem of finding optimal methods for utilizing external data sources, we do show how our formalization reduces cleaning to a well-studied optimization problem.

1 Introduction

When decisions must be made without good information, it is critical to know how much the information at hand can be trusted. To this end, techniques to manage and improve data quality have been developed by the database research community. Data correctness can be captured with integrity constraints [1], while data quality and confidence can be captured in probabilistic databases (e.g. [11, 12]). Work on constraint repair (e.g. [3, 4]) and on purging duplicates (e.g. [5, 9]) supports the automated improvement of data quality, for example by removing unlikely tuples or identifying dissimilar representations of the same entity. However, the problem of connecting data cleaning to the decision making process it supports has not been studied in any detail. Similarly, little attention has been focused on *how* to develop probabilistic models for data or *when* to expend effort or money to acquire good data. We now introduce a running example to illustrate these issues and motivate a model of data cleaning driven by *business decisions*.

Example 1. Consider a hypothetical telecommunications service provider, ACME Telecom. ACME is interested in building a wireless network in Alberta, which requires the construction of cellular towers. To see if it can build a network with sufficient capacity, ACME requires data on the sites available for cellular towers, and the towers they can support.

ACME has obtained catalogs of possible sites from two consulting companies, and represents this information in the two tables of Figure 1. The tables conform to the following schema:

site(Locid: int, Towertype: ttype)

Locid is the deed number for a site and ttype gives the type of towers it can support: "High" for a high-capacity tower, "Low" for a low-capacity tower, and "CU" if the site is Currently Unavailable, but may be able support a cell tower in the future.

The data suggests that one or both lists are of low quality. To what extent can existing data cleaning techniques be applied to ACME's problem? As mentioned above, one class of techniques deals with finding *approximate matches* and reconciling entities across data sources [5, 9]. However, such "record linkage" problems are not an issue here, as we have assumed that the Locid information is correct. (If this assumption does not hold, approximate matches might be used to bring the data to the state shown in the example.)

A second class of techniques that might be applied are *constraint repair* techniques [3, 4]. To formalize this example as a constraint repair problem, one might compute the union of **Smith** and **Jones**, and then assert a key constraint from Locid to Towertype on the resulting table. The data could then be repaired by either deleting tuples or modifying values. Consider applying either approach to Site 121. Certainly, the key constraint can be repaired by deleting either the **Smith** or **Jones** tuple for Site 121, or by changing a "Low" value to "High", or vice versa. However, since there is no evidence to favor any of these repairs, neither delete-tuple constraint repair nor attribute-modification repair is likely to help ACME in this case.

Good repair decisions could be made if the source instances were enriched with reliability information

Smith

Locid	Towertype
121	"High"
128	"Low"
146	"CU"
177	"CU"
199	"Low"
203	"Low"

Jones

Locid	Towertype
121	"Low"
128	"Low"
146	"High"
177	"CU"
199	"Low"

Figure 1: Example site data.

giving the probability that a tuple or attribute value is correct. Such information could be modeled, for example, in a probabilistic data model [11, 12]. A critical question, however, is how this reliability information would be obtained. What probabilities should be assigned to each possible value for ttype for site 121? Furthermore, how can we assign confidence to our reliability estimate to reflect the amount of supporting data we have seen? Clearly, we must distinguish the case in which much historical data supports the unreliability of **Jones** from the case where there is little data available. To help ACME, a data cleaning system must have both a strategy to develop a reliability model, and a way to estimate and adjust the confidence in the model.

Data reliability estimates can be improved by obtaining data via external sources. For example, by obtaining the correct answer for site 121, ACME can help improve its reliability model. If **Smith** turns out to be correct, ACME might assume it is more likely that **Smith** is correct on site 146, and if this too turns out to be the case (or perhaps for several more tuples), ACME may choose to trust **Smith**'s value for site 203.

An important trade-off is thus between the expected improvement in data quality obtained by consulting external sources and the expense of data acquisition from these sources. Existing systems offer neither a way to quantify the expected improvement nor a way to estimate the value of that improvement. If a credible source offered ACME a perfect list, how much should ACME be willing to pay? We argue that this question can only be answered concretely with a model of the *business decisions* faced. To illustrate this point, we return to our example:

Example 1, continued. ACME has a critical decision to make: Should it build a new network in Alberta? There is a fixed overhead of 10,000,000 Canadian dollars for building the network, while once the network is built there will be a net profit of $100,000 for each high-capacity tower and $50,000 for each low-capacity tower. Clearly, errors in the data can have a dramatic impact on the correct choice for this decision. There is no doubt that ACME can determine the quality of this data and correct it: it can send a representative out to survey each site, and contact each site owner. But this is at an average cost of $1000 per record.

How should ACME proceed? Should it ignore the risk caused by the low quality data and make this decision before taking any cleaning action? Should it query a small number of tuples and then decide? Which tuples should be sampled, and from what source? Or should it sample more than once, dependent on the results of prior samples? The question of *what data to clean* and when is a crucial one for any organization managing decision-critical data. In general, the data may impact many decisions, and there may be many external sources available with which to validate data, each with a different reliability. We see that even in this simple example the correct cleaning policy is far from obvious.

In this work we propose a new data cleaning framework geared towards deciding which external data to query, based on its impact on business decisions and its value in updating the data reliability model. Our framework includes a formal model of **external sources**, in which a query on an external source entails a cost. We model **organizational decisions** as actions whose rewards are based on queries. This allows us to quantify the benefit of data cleaning actions. Our model of **source reliability** includes a parameterized data model in which the parameters can be estimated based on current knowledge, including data from external sources. A solution to the cleaning problem will then consist of a *sampling policy* telling which external data to query based on the current reliability estimates and the organization value of data.

In this short paper we concentrate on giving the formal model, and do not discuss solution techniques. However, we do indicate how our notion of a cleaning policy reduces the cleaning problem to a well-studied problem area, that of *Markov Decision Processes* [10].

2 Decision-driven Cleaning Framework

In this section we describe a framework for Decision-driven Cleaning, in the form of a *Cleaning Problem*, consisting of the following components:

- a relational signature for the observed data and true data
- an error model giving a distribution on errors, possibly with hidden parameters
- a set of data sources that can be queried to get higher-quality information on the target data, along with cost and quality information for these sources
- a set of business decisions with associated reward functions

A solution to a cleaning problem is an optimal *cleaning policy* that decides, for every possible history of source data sampling, whether to take a sampling action, or to make a choice for one or more business decisions.

We now discuss each of these elements in detail.

Relational Data Model Our data model is a simple model for integrated relational data. It consists of a fixed schema $\mathcal{R}$ and a set Src of sources. The schema $\mathcal{R}$ consists of a finite set of relations, a mapping att associating with each relation R a set $\mathsf{att}(R)$ of attributes, a mapping Dom associating with each a in $\mathsf{att}(R)$ a domain $\mathsf{Dom}(a)$, which can be either infinite ('string', 'int', 'real', etc.) or finite, and a subset $\mathsf{key}(R) \subset \mathsf{att}(R)$ representing a distinguished key of R.

An *instance* of a relation R is a set of tuples, where a tuple includes a value in $\mathsf{Dom}(a)$ for each a in $\mathsf{att}(R)$. A *global instance* includes a relation instance for each relation R in $\mathcal{R}$ and each source src in Src. We can equivalently present a global instance as one set of tuples, where each tuple t has a source annotation, $\mathsf{src}(t) \in \mathsf{Src}$. A global instance is required to have each of its relation instances satisfy the key constraints.

There is a distinguished source $\mathsf{src}_{\mathsf{true}}$ representing the true, or actual, data values. We let GLOB be the set of global instances for a given schema. We use GI to range over global instances, src to range over sources, and R to range over relations. We write GI.src for the instances with source src in global instance GI, and $\mathsf{GI.src}(R)$ for the instance of R in GI with source src. Given a source instance I for a relation R we let $\mathsf{key}(I)$ denote the set of key values witnessed.

Data Reliability Framework We first develop a general model of data quality in which source data quality models can be computed based on standard Bayesian reasoning. We then discuss a specific model which could be used in a concrete setting.

Our error models are probability distributions over global instances [7, 12]. An alternative would be to attach probabilities to attribute values [11]. For example, consider the tuple for site 121 in the **Smith** table of Figure 1. One could explicitly state that there is an 80% chance that the value of Towertype is actually 'Low', a 15% chance that it is 'CU', and a 5% chance that it is 'High'. However, such a distribution on attribute values can be derived from a distribution on global instances.

Clearly, a decision maker will not generally know the probability distribution P on global instances. As discussed earlier, we seek to support a) uncertainty about the probability distribution P and b) the ability to characterize and update that uncertainty. To do this we adopt a standard Bayesian approach. We assume that the errors in the observed instance are generated by some probabilistic process, where the parameters controlling this process are unknown. For example, one parameter might represent the accuracy of data entry clerks at **Jones**. Given n such *hidden parameters* in an application, we let PAR be the product space consisting of all n-tuples of hidden parameter values, and consider PAR as a probability space. Thus, the values of the hidden parameters control the probability of global instances, and the parameter values themselves lie in a probability space.

An *error model* over a schema $\mathcal{R}$ consists of a probability distribution $G(\theta)$ on GLOB indexed by an element θ in PAR, along with a smooth probability on PAR, called the *prior*. For example, in tossing a coin with an unknown bias, the probability of a head might be θ, and the prior on θ may be a uniform distribution (representing the case where we have no knowledge of the bias). For some set S of global instances, and some θ in PAR, we write $P(S \mid \theta)$ for $G(\theta)(S)$, the probability of seeing the set S of global instances given hidden parameters θ. Also, we write $P(\mathsf{GI} \mid \theta)$ for $P(\{\mathsf{GI}\} \mid \theta)$.

Although we cannot observe the hidden parameters directly, we can gain information about them by looking at the results of sampling. The updated density function represents our improved knowledge of the hidden parameters given the observation of a set S of global instances:

$$\mathsf{POST}_S(\theta) = P(\mathsf{par} = \theta \mid \mathsf{GI} \in S)$$

This is the *posterior distribution* on the hidden parameter space. For example, in coin tossing, the updated density function tells us how to adjust our prior as a result of observing some coin tosses.

Concrete Data Reliability Model We now specialize our general model to one having independent tuple-level error probabilities. For simplicity, we consider only schemas in which the domain of every non-key attribute is finite. Other schemas can be handled by replacing the uniform distribution over the finite domain used below by a probability distribution appropriate for the particular domain.

Our error model has the following hidden parameters. The parameter $\theta^R_{\mathsf{src},a}$ gives the probability that the value of attribute a of relation R in source src has been modified. The parameter $\theta^R_{\mathsf{src},\mathsf{Ins}}$ gives the probability that a tuple has been inserted into relation R of source src. The parameter $\theta^R_{\mathsf{src},\mathsf{Del}}$ gives the probability that a tuple has been deleted from relation R of source src.

A tuple in the parameter space PAR then contains a value for each instance of these parameters for every source src in Src, every relation R in $\mathcal{R}$, and every attribute a in $\mathsf{att}(R)$. The prior distribution on PAR is a β distribution, which is known to be convenient for calculation. In our examples, we generally assume a uniform distribution over the parameter space as the prior. However, we could easily accommodate other priors that represent error information gathered using historical data or other means.

Now we explain how to compute the probability of seeing an instance I' of a relation R for source src, assuming that we are given src, the true instance I for R, and the hidden parameters θ. For every key value kv appearing in I, the probability that kv is absent from I' is $\theta^R_{\mathsf{src},\mathsf{Del}}$. For every key value kv remaining in

I', the probability that the non-key attributes of kv are set in I' to particular values is as follows: for a non-key attribute a, the probability that the value for a in I' agrees with the value in I is $1-\theta^R_{\mathsf{src},a}$. The probability that the value for a in I' will be a particular value in $\mathsf{Dom}(a)$ different from the one in I is $\theta^R_{\mathsf{src},a}/(|\mathsf{Dom}(a)|-1)$. The probability that k new key values are inserted into I' is $\binom{|I|}{k}(\theta^R_{\mathsf{src},\mathsf{Ins}})^k(1-\theta^R_{\mathsf{src},\mathsf{Ins}})^{|I|}$ (each tuple in I leads to a new spurious tuple in I' with probability $\theta^R_{\mathsf{src},\mathsf{Ins}}$). If k new tuples are to be inserted, the key values for each of these are chosen randomly from the initial values in every interval in $\mathsf{Dom}(\mathsf{key}(R))-I$; for each of these tuples, the non-key value for attribute a is chosen to be a particular value from $\mathsf{Dom}(a)$ with probability $1/|\mathsf{Dom}(a)|$.

This calculation gives the probability $P(\mathsf{GI}.\mathsf{src}_{\mathsf{true}}(R) = I \wedge \mathsf{GI}.\mathsf{src}(R) = I' \mid \theta)$ for a source src and an element θ of PAR. The probability of a full global instance GI_0 given θ is then obtained by: $\Pi_{R\in\mathcal{R},\mathsf{src}\in\mathsf{Src}-\{\mathsf{src}_{\mathsf{true}}\}}P(\mathsf{GI}.\mathsf{src}_{\mathsf{true}}(R) = \mathsf{GI}_0.\mathsf{src}_{\mathsf{true}}(R) \wedge \mathsf{GI}.\mathsf{src}(R) = \mathsf{GI}.\mathsf{src}(R) \mid \theta)$

The model above allows for no correlation between errors in attributes. We can generalize to broader error models by fixing a Bayesian Network [13] with unknown weights, specifying the conditional probability of one attribute being correct given that another attribute is or is not correct. The use of such networks for modeling errors is the subject of future work.

Cleaning Model During the data cleaning process, a decision maker will sample data sources, paying to do so. A global instance in our data model represents the data to be potentially sampled, not the data already sampled in this process. The data already sampled is modeled here as a *sampling history*: a function SH that maps a source src and a relation R to a set $\mathsf{SH}(\mathsf{src}, R)$ of tuples. A tuple in the set either has a value for each attribute of R, or has a value for the key attributes and the distinguished value *null* for all other attributes. A sampling history SH is *consistent* with a global instance GI if, for every relation R and source src, 1) every tuple in $\mathsf{SH}(\mathsf{src}, R)$ not containing *null* also appears in $\mathsf{GI}.\mathsf{src}(R)$, and 2) if a tuple in $\mathsf{SH}(\mathsf{src}, R)$ does contain *null*, then no tuple having the same key appears in $\mathsf{GI}.\mathsf{src}(R)$.

We assume that our decision maker begins with some sampling history, which we refer to as the *initial sample*, denoted Init. This would typically contain all the data from some readily-available sources, and no data from expensive sources.

Each source src has an associated *cost function* C_{src} : $N \to \mathbb{Z}$, where $C_{\mathsf{src}}(n)$ gives the cost of sampling src for a group of n tuples. The sampling cost might be linear in n, or might perhaps reflect that sampling can be done more cost-efficiently in bulk. Normally we expect that a user will not be able to sample the actual data $\mathsf{src}_{\mathsf{true}}$, but will be able to sample sources where the parameters $\theta^R_{\mathsf{src},\mathsf{Del}}$, $\theta^R_{\mathsf{src},\mathsf{Ins}}$, and $\theta^R_{\mathsf{src},a}$, are known and small.

A *sampling action* for a source src is a relation, R, and a set $\{\mathsf{kv}_1, \ldots, \mathsf{kv}_n\}$ of key values of appropriate type for R. The result of a sampling action is a sequence $\langle t_1, \ldots, t_n\rangle$ where t_i is either a tuple for R with key value kv_i, or *null*, indicating that no tuple with this value exists in the instance of R.

Example 2. Consider again the ACME example. For the data from Smith Inc. there is an (unknown) probability $\theta^{\mathsf{SM}}_{\mathsf{Towertype}}$ that the Towertype attribute is correct. In addition, there is a probability $\theta^{\mathsf{SM}}_{\mathsf{Del}}$ that a given true lot was deleted, and a parameter $\theta^{\mathsf{SM}}_{\mathsf{Ins}}$ controlling how many spurious tuples are inserted. Similarly, for the data from Jones, we have $\theta^{\mathsf{JS}}_{\mathsf{Towertype}}$, $\theta^{\mathsf{JS}}_{\mathsf{Del}}$, and $\theta^{\mathsf{JS}}_{\mathsf{Ins}}$ parameterizing the probability of an error.

We also assume that we have a correct source, with a linear cost per tuple of sampling.

Suppose we know from historical data that for each consulting company between 0 and 1% of the available lots for the coming year are missed, and between 0 and .5% of the Locid values correspond to non-existent lots. Hence we can take a prior distribution on $\theta^{\mathsf{SM}}_{\mathsf{Del}}$ and $\theta^{\mathsf{JS}}_{\mathsf{Del}}$ a uniform distribution on the interval $[0, .01]$, while taking a prior on $\theta^{\mathsf{SM}}_{\mathsf{Ins}}$ and $\theta^{\mathsf{JS}}_{\mathsf{Ins}}$ as a uniform distribution on $[0, .005]$. For the rate of modification of the Towertype attribute, we have no historical data, so we take these to have a prior that is uniformly distributed on $[0, 1]$.

Suppose that we have sampled our oracle on 100 key values, and determined that all 100 Locid values for the Smith data are valid lots, and that for 99 of these the Smith report has the correct value for Towertype.

Then we can estimate a new posterior distribution on $\theta^{\mathsf{SM}}_{\mathsf{Towertype}}$ given this sample data and the observed data, with the density of $\theta^{\mathsf{SM}}_{\mathsf{Towertype}}$ now $(1-\theta^{\mathsf{SM}}_{\mathsf{Towertype}})^{99}\theta^{\mathsf{SM}}_{\mathsf{Towertype}}/\int_0^1(1-x)^{99}x\,dx$.

Business Decisions Associated with a cleaning problem is a finite set $\{\tau_1 \ldots \tau_r\}$ of *business decisions*, where each decision τ has an associated finite set $\mathsf{Choices}(\tau)$ of choices. Each decision τ and choice ρ in $\mathsf{Choices}(\tau)$ has associated with it a relational query $Q_{\tau,\rho}$ over the signature $\mathcal{R}$. Evaluating $Q_{\tau,\rho}$ on a (true) instance I gives the outcome if the decision maker chooses ρ for decision τ.

In the ACME example, our business decision τ is Build, with choices ρ_1 =Yes and ρ_2 =No. The reward query associated with Build = Yes is given by Q_{τ,ρ_1}:

```
select (100,000 * High.tot + 50,000 * Low.tot -
10,000,000) as Profit
from
(select cnt(*) as tot from Site where
Towertype='High') High,
(select 50,000 cnt(*) as tot from Site where
Towertype='Low') Low
```

The reward query associated with Build = No is $Q_{\tau,\rho_2} = 0$.

Cleaning Policies We can now take a *cleaning problem* to be a tuple $(\mathcal{R}, \mathsf{Src}, \mathsf{EMOD}, C, \mathsf{BD})$, where $\mathcal{R}$ is a schema, Src a set of sources, EMOD an error model giving a distribution over global instances of $\mathcal{R}$ for the sources in Src, C a cost function, and BD a set of business decisions. A *policy* for a cleaning problem is a function deciding, for each sampling history, either a sampling action, or a choice for one or more of the business decisions. A policy is a recipe telling what should be sampled at any state: given an initial sampling history Init, and a policy λ, one can apply λ repeatedly to get a sequence of histories and choices for business decisions: $\mathsf{SH}_1 = \mathsf{Init}$ unioned with the response of $\lambda(\mathsf{Init})$ on GI, $\mathsf{SH}_2 = \mathsf{SH}_1$ unioned with the response of $\lambda(\mathsf{SH}_1)$ on GI, etc. In the process, we obtain larger and larger sampling histories and some sequence of choices for business decisions. A policy is *valid* if on every global instance GI it produces a sequence such that every business decision $\tau_i : i \leq r$ is decided exactly once. For a valid policy, we can evaluate its effectiveness on a global instance GI via the *cumulative reward*: if the policy produces sampling actions $S_1 \ldots S_k$ when applied on GI, then the reward is $\Sigma_{i \leq r} Q_{\tau_i,\rho}(\mathsf{GI}.\mathsf{src}_{\mathsf{true}}) - \Sigma_{j \leq k} C_j(|S_j|)$, where ρ is the choice selected for τ_i when running policy λ on GI and $C_j(|S_j|)$ is the cost of the j^{th} sample. That is, the reward is the gain from the business decisions minus the total cost of sampling.

The goal of cleaning (in our sense) is then to find the *optimal policy* for a cleaning problem, given an initial history Init. This optimal policy maximizes the expected value of the reward, conditioned on the event that the global instance is consistent with Init. The optimal policy tells the cleaner the "best" data to sample in a precise sense.

3 Ongoing Work

In our framework, a solution to the cleaning problem is an optimal strategy for a certain game. Optimization strategies for more general planning problems, such as those for Markov Decision Processes (MDPs) [8], are applicable here. We now very briefly review MDPs and their application to cleaning.

An MDP describes a game between a player and the environment in which the player chooses an action and the environment chooses a resulting state according to a probability distribution associated with the action. Each action has an associated reward function, and the goal of the player is to choose a strategy that maximizes her expected cumulative reward. It is easy to translate the cleaning problem here into an MDP: the states of the MDP are the sampling histories, while actions are sampling actions and choices for the business decisions. The rewards for sampling actions are the negative of the cost of sampling (based on the associated cost function of the sources), while the rewards for decision actions are the expected values of the corresponding queries, where the expectation is conditioned on the information known from the sampling history. The naive translation will yield a very large MDP (exponential in the size of the data). Given the fact that the best algorithms for solving general MDPs (based on dynamic programming) are quadratic, one cannot hope to use the straightforward approach in practice. In ongoing work we are investigating the use of abstraction techniques, along the lines of [6, 2], which may yield a more manageable problem.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] J. Baum and A. E. Nicholson. Dynamic non-uniform abstractions for approximate planning in large structured stocastic domains. In *Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence*, pages 587–598, 1998.

[3] A. Cali, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, 2003.

[4] J. Chomicki and J. Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In L. Bertossi, A. Hunter, and T. Schaub, editors, *Integrity Tolerance.* Springer, 2004.

[5] W. Cohen, P. Ravikumar, and S. Feinberg. A comparison of string-distance metrics for name-matching tasks. In *IIWeb*, 2003.

[6] T. Dean and R. Givan. Model minimization in markov decision processes. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 106–111, 1997.

[7] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[8] G. E. Monahan. A survey of partialy observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28:1–16, 1982.

[9] A. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *DKMD*, 1997.

[10] M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.

[11] A. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. of ICDE 2006.* IEEE Computer Society, 2006.

[12] D. Suciu and N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 963–963, New York, NY, USA, 2005. ACM Press.

[13] R. L. Winkler. *An Introduction to Bayesian Inference and Decision.* Holt, Rinehart, and Winston, 1972.