Deriving Effectiveness Measures for Data Quality Rules

Lei Jiang
University of Toronto
Canada
leijiang@cs.toronto.edu

Alex Borgida[*]
Rutgers University
USA
borgida@cs.rutgers.edu

Daniele Barone
University of Toronto
Canada
barone@cs.toronto.edu

John Mylopoulos[†]
University of Trento
Italy
jm@cs.toronto.edu

ABSTRACT

The poor quality of data constitutes a major concern worldwide, and an obstacle to data integration and analysis efforts. Detecting errors and inconsistencies using application specific data quality rules play an important role in data quality assessment. These rules have different efficacy and cost under different circumstances. In our previous work, we have proposed a quantitative framework for measuring and comparing data quality rules in terms of their effectiveness. Effectiveness formulas are built from variables that represent probabilistic assumptions about the occurrence of errors in data values, and our earlier work gave examples of how to derive these formulas in an ad-hoc fashion.

This paper lays the foundations of a workbench-approach for systematically deriving effectiveness formulas. The approach involves several steps, including building Bayesian network graphs, adding (symbolic) probabilities to the nodes in the graph, and deriving effectiveness formulas. The graphs are built algorithmically, for a large and useful class of data quality rules. We present this approach and its implementation in Python, and report its evaluation results, which show that the resulting formulas give reasonable estimates of effectiveness scores under various scenarios.

1. INTRODUCTION

The poor quality of data constitutes a major concern world-wide, and an obstacle to data integration and analysis efforts. The first necessary step towards better quality is being able to effectively detect "problematic" data. DQ assessment techniques in practice heavily rely on application-specific *data quality (DQ) rules* (or simply *rules*), which specify what data values are (likely to be) consistent and error-free [2, 1]. In DBMS, this is achieved through the use of SQL check constraints, or code embedded in transactions.

[*]Also affiliated to University of Toronto

[†]Also affiliated to University of Toronto

It is not unusual that a piece of data could be assessed in multiple ways. For example, consider a *CustomerOnt(sin, name, city, street, pcode)* table, which stores information about the customers of a company living in Ontario, including their social insurance number (*sin*) and postal-code (*pcode*). Postal-code values in the table could be assessed using (i) a regular expression, (ii) a pre-compiled list of valid postal-codes for the application of interest, (iii) an extended conditional functional dependency [5] such as$pcode \neq$ "M8V 4G1" $\rightarrow$ *city* [1], or (iv) a separate table of postal-code / phone area-code matches (since the correlation cannot be captured using a functional dependency – but only with a separate mapping table [12]). If there were another table about customers available (e.g., by a different department of the same company, or by another company), we can also assess values of postal-code (and other attributes) by (v) linking customers from different, independently-created tables.

Ideally we would assess data using *all* possible alternatives: together they provide better coverage of detectable errors. For example, (i) can be used to detects syntactically invalid Canadian postal-codes, while (ii) can be used to further identify customers with non-Ontario postal-codes; furthermore, (iii) and (iv) can be used to detect additional errors in Ontario postal-codes (when they do not match with the corresponding city or phone area-code properly). However, using all possible rules is likely unfeasible in practice because each of them incurs some costs, including human/organizational costs in entering and maintaining data that is not strictly needed for the application, and computer-related costs for checking constraints. For example, (iv) would require an additional attribute, *area-code*, to be added and updated for the *CustomerOnt* schema, and as well as a separate postal-code / phone area-code mapping table.

Our ultimate goal is to provide a workbench which supports comparative cost-benefit analyses for DQ rules. This workbench would be useful to designers during the initial

[1]*pcode* values uniquely determines *city* values except for "M8V 4G1", which lies the boundary between the cities of Toronto and Etobicoke

design of a database (since some alternatives require different schemas), as well as in a data cleaning initiative during administration (e.g., deciding which alternative to carry out depending on available resource). In our previous work [11], we have introduced the concept of "effectiveness" as a measure of benefit, and proposed a quantitative framework for measuring and comparing DQ rules in terms of their effectiveness. The following example illustrates effectiveness of DQ rules and its measurement.

Example 1: Consider the *CustomerOnt* table defined earlier, and a DQ rule ϕ_0 which checks *CustomerOnt.pcode* values against a list of pre-compiled Ontario postal-codes (stored in the table L_{Ont} with a single attribute *pcode*). Whenever the *pcode* value v in a *CustomerOnt* tuple is not found in L_{Ont}, the rule is violated and the value v is assessed as erroneous. Intuitively, effectiveness of ϕ_0 measures the likelihood that ϕ_0 produces a correct / incorrect assessment. More specifically, effectiveness of ϕ_0 can be quantified using the number of true positive (TP), false positive (FP) and false negative FN.

For example, a TP occurs when *CustomerOnt.pcode* contains some factual error and it is signaled erroneous by ϕ_0 (i.e., ϕ_0 is violated). There are actually two cases where this can happen (assuming L_{Ont} contains only valid Ontario postal-codes):

- v is not the correct *pcode* in that tuple, and is an invalid Ontario postal-code (and therefore is not contained in L_{Ont}).
- v is not the correct *pcode* in that tuple but happens to be a *different valid* Ontario postal-code, which luckily is missing from L_{Ont}.

To estimate the number of TP, we need to assign probabilities (symbolic ones, rather than specific numeric values) to various events whose occurrence (or absence) leads to one of these two cases:

- p: the probability that a postal-code value in the *CustomerOnt* table is wrong;
- c: the probability that a factually incorrect postal-code value in the *CustomerOnt* table happens to still be a valid Ontario postal-code;
- s: the probability that a valid Ontario postal-code is missing from L_{Ont}

The expected number of TP is then expressed by the following formula:

$$(p \times (1-c) + p \times c \times s) \times N$$

where $p \times (1-c)$ is the probability of the first case, $p \times c \times s$ is the probability of the second case, and N is the total number of tuples in *CustomerOnt*. In a similar way, we can obtain the expected number of FP (i.e., v is factually correct but ϕ_0 is violated):

$$((1-p) \times s) \times N$$

and the expected total number of FN (i.e., v is factually incorrect yet ϕ_0 is not violated):

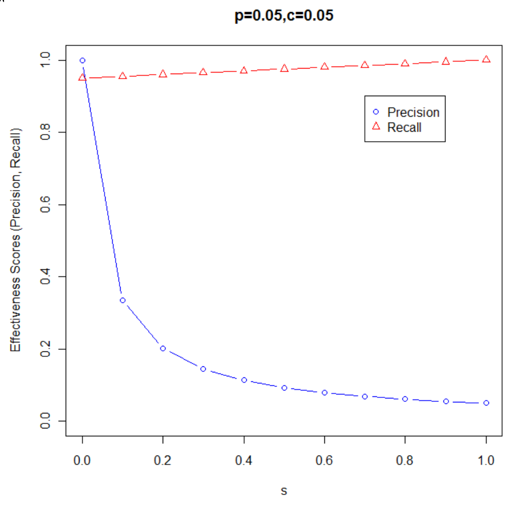
$$(p \times c \times (1-s)) \times N.$$

The expected number of TP, FP, FN can then be combined to produce measures such as precision, recall and F-measure, which we call the *effectiveness formulas* for ϕ_0.

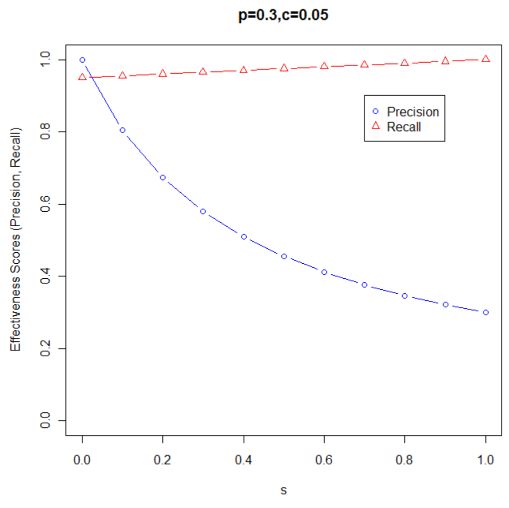
■

Effectiveness formulas are parametrized by *symbolic* variables for the various probabilities (e.g., p, c). This allows us (i) to evaluate the effectiveness of a rule as a function of changing values of these probability parameters (i.e., ranging from 0 to 1), (ii) to identify the most effective region(s) for a rule, (iii) to compare rules in terms of their effectiveness, and (iv) to obtain *reasonable* effectiveness scores for a rule by estimating values for these parameters via sampling. We emphasize again, that the ultimate goal of these formulas is to perform cost-benefit analysis of alternative DQ rules when we cannot implement them all.

As an example of rule evaluation, consider Figure 1 (taken from [11]), which shows how the effectiveness of ϕ_0 chances with two parameters: s and p. We can see as s increases, precision drops dramatically while recall increases slightly. Moreover, by comparing 1(a) and 1(b), we can observe that p has much larger influence on precision than on recall: recall remains almost unchanged in the relative clean (i.e., $p = 0.05$) and dirty ($p = 0.3$) database; however, in the relative dirty database, precision decreases considerably slower as s increa



(a) A relative clean database



(b) A relative dirty database

Figure 1: Evaluation of ϕ_0.

From our experience in [11], derivation of effectiveness formulas is a complex and error prone process even for a simple rule as shown in the previous example. The analysis in the example will be further complicated if we remove the assumption that "L_{Ont} contains only valid Ontario postal-codes"; new probabilistic variables need to be introduced to accommodate the cases where L_{Ont} contains non-Ontario postal-codes and where L_{Ont} contains plain invalid Canadian postal-codes.

This particular paper concentrates the derivation of effectiveness formulas for DQ rules. The main contributions of the paper include:

- a characterization of the main factors that determine effectiveness of DQ rules (Section 2),
- a novel, semi-automatic approach to derive effectiveness formulas for a large and useful class of DQ rules (including both equality- and tuple-generating dependencies) (Section 3); it consists three phases:
  - construction of a Bayesian network graph of events of interest,
  - assignment of (symbolic) probabilities to the events, and,
  - generation of effectiveness formulas from the graph;
- an evaluation of the proposed approach using simulations (Section 4).

2. THE DETERMINING FACTORS FOR EFFECTIVENESS OF DQ RULES

We consider DQ rules that can be expressed in the following form

$$H(\bar{z}) \leftarrow R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m), M_1(\bar{y}_1), \ldots, M_n(\bar{y}_n)$$

where each R_i $(i = 1 \ldots m)$ is the name of a database relation, each M_j $(j = 1 \ldots n)$ is an arithmetic (in)equality and $H(\bar{z})$ is either another relation $R_0(\bar{x}_0)$ or (in)equality $M_0(\bar{y}_0)$; $\bar{x}_i$, $\bar{y}_j$ and $\bar{z}$ are tuples of variables and/or constants[2]. As usual, we require that a DQ rule be "safe" in the sense every variable appearing in some $\bar{y}_j$ must also appears in some $\bar{x}_i$.

Without loss of generality, we impose two additional conditions: (i) every $\bar{x}_i$ contains variables only (e.g., $R(x, 6)$ is expressed as $R(x, y), y = 6$) and (ii) variable occurrences in x_i must be distinct, so each variable uniquely identifies a database relation occurrence and an attribute/column in it. Furthermore, we assume every rule is put into a standard form where: all tautologies (e.g., $x = x$), contradictions ($x > y$ and $x < y$) and redundancies (e.g., $x = z$ given $x = y$ and $y = z$) are removed. Variables that appear *only* in $\bar{x}_i$ are irrelevant as far as effectiveness is concerned, hence are replaced by _.

DQ rules defined in this way can be used to express a large and useful class of integrity constraints that includes both tuple-generating and equality-generating dependencies, as illustrated by the following examples.

[2]Variables that appear *only* in the body of a rule are implicitly existentially quantified, while others are implicitly universally quantified.

Example 2: Consider the Ontario customer schema *CustomerOnt* again. The rule ϕ_0 discussed in Example 1 can be expressed as:

$$\phi_0 : \quad L_{Ont}(\mathit{pcode} : x_2) \quad \leftarrow \quad \mathit{CustomerOnt(sin{:}\_,name{:}\_,city{:}\_,street{:}\_,\ pcode{:}x_1)}, \; x_1 = x_2.$$

The conditional FD (iii) from Section 1 is expressed as:

$$\phi_1 : \quad y_1 = y_2 \quad \leftarrow \quad \mathit{CustomerOnt(sin{:}\_,name{:}\_,city{:}y_1,street{:}\_,pcode{:}x_1)}, \; \mathit{CustomerOnt(sin{:}\_,name{:}\_,city{:}y_2,street{:}\_,pcode{:}x_2)}, \; x_1 = x_2,\ x_1 \neq \text{"M8V 4G1"}.$$

■

The effectiveness of a rule is intended to measure the likelihood it produces a correct/incorrect assessment result. To understand why a rule may produce an incorrect result, it helps to note that data used to *evaluate* the rule is not necessary the same as data *assessed* by the rule. For example, to evaluate a rule which asserts that employees always earn less than their managers, we need values for the salary of employees and their managers; but we almost always use this rule to detect/assess cases where the employee was assigned a salary that was too high, rather than the manager's salary was too low or that the manager assignment was incorrect.

Given a DQ rule ϕ, let X_ϕ be the set of attributes corresponding to the named variables in $\bar{x}_j$; X_ϕ specifies exactly what data is needed to evaluate ϕ. For example, for ϕ_1, $X_{\phi_1} = \{\mathit{CustomerOnt.pcode}, \mathit{CustomerOnt.city}\}$; this means we need both *pcode* and *city* values from the *CustomerOnt* table in order to evaluate ϕ_1.

The data assessed by ϕ is a nonempty subset Y_ϕ of X_ϕ. We make this explicit by adding the set of the named variables corresponding to the attributes in Y_ϕ as the arguments of ϕ. For example, for ϕ_1, we have several choices, including $Y_{\phi_1} = \{\mathit{CustomerOnt.pcode}\}$ (assessing *CustomerOnt.pcode* values only, in which case we write $\phi_1(x_1,\ x_2)$), or $Y_{\phi_1} = \{\mathit{CustomerOnt.pcode}, \mathit{CustomerOnt.city}\}$ (i.e., assessing both *CustomerOnt.pcode* and *CustomerOnt.city* values, in which case we write $\phi_1(x_1,\ x_2,\ y_1,\ y_2)$).

What data is assessed by a rule is the choice of the person who performs the DQ assessment activity. The algorithm to be presented in Section 3 can be equally applied no matter what choice is made.

The rule ϕ produces an incorrect assessment result exactly when

- the data values assessed by ϕ are factually correct, but ϕ is still violated (i.e., an *FP*), or,
- the data values assessed by ϕ are factually incorrect, yet ϕ is not violated (i.e., an *FN*).

Otherwise, ϕ produces a correct result. For example, a violation of ϕ_0 caused *solely* by erroneous (or missing) L_{Ont}*.pcode* values is an incorrect assessment result; similarly, ϕ_0 produces an incorrect result when an erroneous *CustomerOnt.pcode* value does not trigger its violation (because it is "cancelled out" by erroneous (or missing) values in L_{Ont}*.pcode*).

The likelihood that a rule produces an incorrect result (i.e. *FP* or *FN*) depends on the specification of the rule, the data used to evaluate the rule, and the data assessed by the rule.

More specifically, as we shall see below, the effectiveness of a rule ϕ is effected by the conjuncts in rule, the sets X_ϕ and Y_ϕ, and chances of errors in them, as well as the probability of missing tuples from relations. Another factor for effectiveness is whether or not the rule has exceptions (i.e., its violation does not caused by erroneous data). In this paper, we ignore this factor, assuming rules to be always valid.

3. SEMI-AUTOMATIC DERIVATION OF EFFECTIVENESS FORMULAS

We describe a novel, semi-automatic approach to derive effectiveness formulas for the class of DQ rules as defined in Section 2. It generates the effectiveness formulas for a rule in three phases:

- construction of a Bayesian network graph of events of interest,
- assignment of (symbolic) probabilities to the events in the graph, and,
- formulation of effectiveness formulas from them.

3.1 Phase 1: Construct a Bayesian network graph of events

A Belief/Bayes Network graph is a directed acyclic graph $G = (V, E)$, where V is a set of nodes representing the events (in this case, those deemed relevant in determining the effectiveness of a rule ϕ) and E is a set of edges representing probabilistic relationships among these events: there is an edge (v_i, v_j) if the probability distribution of node v_j (called the child node) is directly influenced by that of node v_i (called the parent node). A missing edge between two nodes means they are (conditionally) independent.

In our case, the relevant events will be comparisons between *real*/*stored* versions of database table "cells" — attribute fields of relation tuples. Given a specific attribute A of a specific tuple t, we represent the "cell" using x^t where x is the variable in the position of attribute A. We use $real(x^t)$ to refer to the value that is *supposed* to be stored at x^t, and use $stored(x^t)$ to refer the value that is *actually* stored at x^t. So each event is a (conjunction of) comparisons involving terms like $real(x^t)$, $stored(x^t)$ and constants.

For example, consider the rule ϕ_1. Events of interest include: (i) $real(x_1^{t_1}) \neq stored(x_1^{t_1})$? (whether or not the *pcode* value supposed to be stored in $x_1^{t_1}$ is the different from the one actually stored), and (ii) $stored(x_1^{t_1}) = store(x_2^{t_2}) \wedge stored(y_1^{t_1}) \neq stored(y_2^{t_2})$? (whether or not the tuples t_1 and t_2 have the same stored *pcode* and *city* values). The question marks emphasize that each event has two possible output: true or false. We use $Error(x^t)$? as a shorthand for the event $real(x^t) \neq stored(x^t)$?. We ignore the superscript in the events when it is clear from the context. Finally, if $E(\bar{w})$ is an expression in a rule, we'll use $E(real(\bar{w}))$ for $E(\bar{w})$ with each variable w in $\bar{w}$ being replaced by $real(w)$ (similarly for $E(stored(\bar{w}))$).

The algorithm for constructing the Bayesian network graph for a rule ϕ starts with a node, to be called *Root*?, which is intended to represent the evaluation of the rule over the database. It stands for the event $R_1(stored(\bar{x}_1)) \wedge \ldots \wedge R_m(stored(\bar{x}_m)) \wedge M_1(stored(\bar{y}_1)) \wedge \ldots \wedge M_n(stored(\bar{y}_n)) \wedge \neg H(stored(\bar{z}))$?, which is true exactly when the rule ϕ is violated.

The algorithm builds the graph backwards by repeatedly applying the following actions, depending on the type of node:

1. For a conjunction node $\bigwedge v_i$?, add parent nodes v_1?, v_2?, $\ldots$;
2. For each $E(stored(\bar{w}))$?, where $E(\bar{w})$ is either a $R_i(\bar{x}_i)$, $M_j(\bar{y}_j)$ or $\neg H(\bar{z})$, add parent nodes $E(real(\bar{w}))$?
3. For each $E(stored(\bar{w}))$?, where $E(\bar{w})$ is either a $M_j(\bar{y}_j)$ or $\neg H(\bar{z})$, add parent nodes $Error(w_i)$?
4. The rule itself tells us that the truth of $H(real(\bar{z}))$? is implied by the real body atoms, thereby requiring edges to it from each $R_i(real(\bar{x}_i))$? and $M_j(real(\bar{y}_j))$?.

The above graph represents the minimal dependencies expected to hold between the events of interest in calculating effectiveness of a DQ rule. Additional nodes and edges could be introduced based on specific domain knowledge in an application. For example, a person's gender may influence the likelihood whether or not this person is a smoker. Therefore, we need to add an edge from $real(x)$="male"? to $real(y)$="smoker"? where x and y are variables in the positions for attributes $Person.gender$ and $Person.smokingStatus$ respectively.

Example 3: Figure 2 shows the output of this algorithm applied to ϕ_1. The node 0 is the root of the graph. The nodes 1-3, and the edges (1,0), (2,0), and (3,0) are introduced by the first action in the algorithm. The nodes 4-10, and the edges (4,1), (5,1), (6,2), (7,2), (8,2), (5,3), (9,3), (10,3) are introduced by the second action. The edges (4,6), (9,6) are introduced by the last action.

An important property of this algorithm is that it is insensitive to the data assessed by the rule(e.g., $\phi_1(x_1, x_2)$, $\phi_1(y_1, y_2)$ and $\phi_1(x_1, x_2, y_1, y_2)$ all share the same graph).

■

3.2 Phase 2: Assign (symbolic) probabilities to events

The second phase generates, for each node in the graph G, a conditional probability table (CPT) [10], which specifies the conditional probability distributions for the node given all its parent nodes. A CPT of a node v is generated by (i) finding the parent nodes $parents(v)$ of v, and (ii) inserting a row in the CPT for every possible combination of the values of v and $parents(v)$. There are 2^{n+1} possible rows in the CPT where n is the size of $parents(v)$. One of the computation advantage of using the graph notation is that normally n is much small than the total number of nodes in G. For example, the average of n in Fig. 2 is around 1.18 compared to 11, the number total of nodes in the graph.

Each row in a CPT is associated with a constant (if the probability for that configuration is known with certainty), or a *probabilistic variable*, as illustrated in the following example.

Example 4: Consider the graph generated for ϕ_1 (Fig. 2). Part of the CPT for $stored(x_1) = stored(x_2)$? is shown in Table 1.

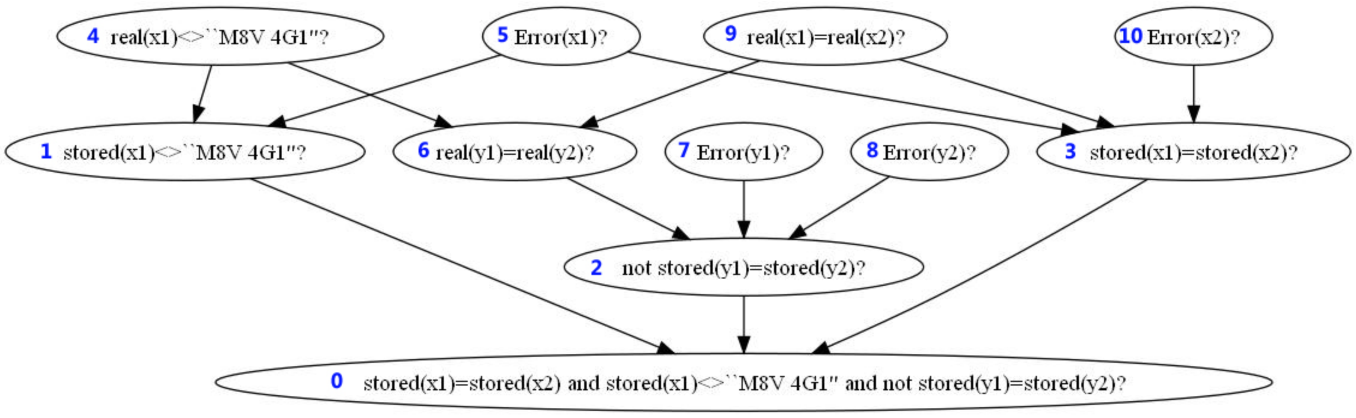


Figure 2: The Bayesian network graph for ϕ_1.

Table 1: The CPT for $stored(x_1) = stored(x_2)$?.

	$Error(x_1)$?	$Error(x_2)$?	$real(x_1) = real(x_2)$?	$stored(x_1) = stored(x_2)$?	prob.
1	*true*	*true*	*true*	*true*	s_2
2	*true*	*true*	*false*	*true*	s_6
3	*true*	*false*	*true*	*true*	0
4	*true*	*false*	*false*	*true*	s_5
5	*false*	*true*	*true*	*true*	0
6	*false*	*true*	*false*	*true*	s_5
7	*false*	*false*	*true*	*true*	1
8	*false*	*false*	*false*	*true*	0

A row in Table 1 is associated with "1" (respectively "0") when the corresponding configuration always (respectively never) happens. For example, two *pcode* values actually stored in the *CustomerOnt* table must be the same if the real values supposed to be stored there are identical, and both values are error-free (i.e., Row 7 of Table 1); on the other hand, when only one of the stored values is erroneous, they must be different (i.e., Row 3 and 5).

A row in Table 1 is associated with a probabilistic variable when the configuration cannot be predicted with certainty. For example, we use s_2 to denote the probability of the event that both stored *pcode* values contain exactly the same error (e.g., Row 1 of Table 1).

The rest eight rows in the CPT for ϕ_1 contains the configurations in which $stored(x_1) = stored(x_2)$? is false, and can be assigned probabilities automatically based on the fact that the sum of the conditional probabilities of $stored(x_1) = stored(x_2)$? being true and false (given a particular configuration of its parent nodes) is always 1. For example, the probability assigned to the row, which is identical to Row 1 except for $stored(x_1) = stored(x_2)$? being false, is 1 - s_2.

■

3.3 Phase 3: Formulate effectiveness formulas

The last phase calculates the probabilities of three events, corresponding to the occurrence of a *TP*, *FN* and *FP*, and uses the result to formulate the effectiveness formulas. For the sake of presentation, in what follows we write v (instead of $v? =$ true) to mean the corresponding event being true (i.e., it occurs), and write $\neg v$ (instead of $v? =$ false) otherwise. The three events are:

- $v_{TP} \Leftrightarrow v_F \wedge v_D$
- $v_{FN} \Leftrightarrow v_F \wedge \neg v_D$
- $v_{FP} \Leftrightarrow \neg v_F \wedge v_D$

where v_F means at lease one of the data values assessed by the rule is factually erroneous, and v_D means the rule is violated.

Example 5: : For the DQ rule $\phi_1(y_1, y_2)$ considered earlier, v_F and v_D are defined as follows:

- $v_F \Leftrightarrow Error(y_1) \vee Error(y_2)$
- $v_D \Leftrightarrow stored(x_1) = stored(x_2) \wedge stored(y_1) \neq stored(y_2) \wedge stored(x_1) \neq$ "$M8V4G1''$

■

One may be tempted to calculate the probabilities of these events (e.g., v_{TP}), according to the definition of joint probability. For example, $pr(v_{TP}) = pr(v_F \wedge v_D) = pr(v_F) \times pr(v_D|v_F)$. However, in many cases (especially when the rule is complex), it is difficult to directly obtain pr($v_D|v_F$) (i.e., the probability that a rule is violated given that some value being assessed by it is factually erroneous).

Instead, we take a detour in calculating these probabilities, by using nodes from the graph G as intermediate events; they allow us to divide the calculation into smaller pieces. For example, suppose v_1? and v_2? are two nodes in G and are used as intermediate events for $pr(v_{TP})$ such that

$$\begin{aligned} pr(v_{TP}) &= pr(v_F \wedge v_D) \\ &= pr(v_F \wedge v_1 \wedge v_2 \wedge v_D) \;+ \\ &\quad pr(v_F \wedge v_1 \wedge \neg v_2 \wedge v_D) \;+ \\ &\quad pr(v_F \wedge \neg v_1 \wedge v_2 \wedge v_D) \;+ \\ &\quad pr(v_F \wedge \neg v_1 \wedge \neg v_2 \wedge v_D) \end{aligned}$$

G allows to dramatically simplify the calculation of these joint probabilities if the average number of parents of a node is much smaller than the total number of nodes in the graph, thanks to the chain rule for Bayesian networks [10],

$$pr(v_1 \wedge \ldots \wedge v_n) = \prod_{i=1}^{n} pr(v_i|parents(v_i))$$

where each $pr(v_i|parents(v_i))$ is given by the CPT of the node v_i.

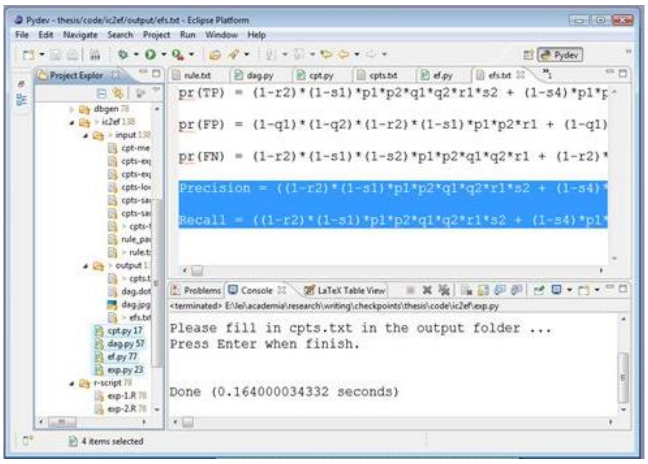
In the above case, if (v_F, v_1), (v_1, v_D), (v_F, v_2), (v_2, v_D) are the only edges in G, the calculation of $pr(v_F \wedge v_1 \wedge v_2 \wedge v_D)$ can be reduced to $pr(v_D|v_1 \wedge v_2) \times pr(v_1|v_F) \times pr(v_2|v_F) \times pr(v_F)$.

This approach is based on the premise that probabilities of $pr(v_D|v_1 \wedge v_2)$, $pr(v_1|v_F)$, $pr(v_2|v_F)$, $pr(v_F)$ are much easier to obtain than that of pr($v_D|v_F$).

Example 5 (continued): For the rule $\phi_1(y_1, y_2)$ (whose graph is shown in Fig. 2), the nodes that can be used to for the intermediate events include $stored(x_1) = stored(x_2)$?, $stored(x_1) \neq$ "$M8V4G1''$? and $stored(y_1) \neq stored(y_2)$?. ■

3.4 Implementation

This approach has been implemented in Python in three modules. The first module *dag.py* parses a DQ rule and generates a Bayesian network graph for it in the DOT language[3], which can be visualized using graphviz[4] (see Figure 2). The second module *cpt.py* reads the graph and generates a file which contains CPTs for all the nodes in the graph; each row in a CPT is assigned either a constant or probabilistic variable (see Table 1). Finally, these CPTs are read in by the third module *ef.py*, which generates effectiveness formulas (*precision* and *recall*). Figure 3 shows a screenshot of a particular run of the modules for ϕ_1. The generated formulas can then be fed into a tool (such as the R framework[5]) for plotting (see the next section for such examples).


Pydev - thesis/code/ic2ef/output/efs.txt - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Pydev
Project Explor
rule.txt
dag.py
cpt.py
cpts.txt
ef.py
efs.txt
Problems
Console
LaTeX Table View
4 items selected

```
pr(TP) = (1-r2)*(1-s1)*p1*p2*q1*q2*r1*s2 + (1-s4)*p1*p
pr(FP) = (1-q1)*(1-q2)*(1-r2)*(1-s1)*p1*p2*r1 + (1-q1)
pr(FN) = (1-r2)*(1-s1)*(1-s2)*p1*p2*q1*q2*r1 + (1-r2)*
Precision = ((1-r2)*(1-s1)*p1*p2*q1*q2*r1*s2 + (1-s4)*
Recall = ((1-r2)*(1-s1)*p1*p2*q1*q2*r1*s2 + (1-s4)*p1*
```

```
<terminated> E:\lei\academia\research\writing\checkpoints\thesis\code\ic2ef\exp.py
Please fill in cpts.txt in the output folder ...
Press Enter when finish.


Done (0.164000034332 seconds)
```

Figure 3: A screenshot of the tool for deriving effectiveness formulas.

4. EXPERIMENTAL EVALUATION

We evaluate our proposed approach by comparing its output for a rule with the effectiveness scores one would obtain when actually applying the rule to assess a database. As in the evaluation of duplicate detection algorithms (e.g., [9]), we use a data generator to inject errors to synthesized clean data. The advantage of using synthesized data is that we have complete control over (and knowledge of) the type and percentage of errors in the data; this makes it much easier to perform a comprehensive evaluation by synthesizing, in a systematic way, datasets of varying characteristics (e.g., a *CustomerOnt* table containing 1%, 5%, 10%, ...of erroneous *pcode* values).

4.1 Settings

We evaluate our approach using two rules of different nature. The first one is a tuple-generating dependency, specified with respect to the relation schema *Address(sin, fsa, ldu, prov, city)*. This schema is used to record the address of a person (identified by his/her social insurance number (*sin*)), including postal code (*fsa*[6] and *ldu*[7]) and province (*prov*) and city (*city*) .

ϕ_2. $L_{FSAOnt}(y)$ $\leftarrow$ *Address(sin:_, fsa:y, ldu:_, prov:x, city:_), x='Ontario'.*

This rule asserts that if the province value in an address tuple is 'Ontario', its FSA code value must appear in a list L_{FSAOnt} of valid Ontario FSA codes.

The second rule is an equality-generating dependency, which is specified with respect to the relation schema *Restaurant(name, addr, desc)*; this schema is used to record the name, address and description of a restaurant.

ϕ_3. $y_1 = y_2$ $\leftarrow$ *Restaurant(name:*y_1*, addr:*x_1*, desc:_),*
Restaurant(name:y_2*, addr:*x_2*, desc:_),* $x_1 = x_2$.

This rule asserts that if two restaurants have the same address, they must have the same name.

The evaluation is carried out in the following way. Given a DQ rule ϕ and a relation schema R, (i) we first generate a clean instance I of R, using real data as seed (e.g., Canada Post's FSA Map[8] for *Address*, and the dataset obtained from the Fodor's and Zagat's restaurant guides [3] for *Restaurant*); (ii) we then inject errors whose type and percentage is determined in according to a set of adjustable parameters; (iii) next we apply ϕ to I and count the number of TP, FP and FN (this is possible since we have the complete knowledge of the ground truth of I), and use these three numbers to calculate specific precision, recall and FMeasure scores for ϕ with respect to I (we call them *measured* effectiveness scores) (iii) finally we use the effectiveness formulas (generated using the proposed approach) to obtain specific precision, recall and FMeasure scores (we call them *estimated* effectiveness scores), by assigning values to the probabilistic variables in the formulas. This process is repeated for 10 times, and in the end we compare the measured scores with the estimated ones.

4.2 Results

The comparison of measured and estimated scores is carried out in different scenarios, each of which examines the impact of certain type/percentage of errors on the effectiveness of a rule. We give a summary of some of these scenarios

[3]http://www.graphviz.org/doc/info/lang.html
[4]http://www.graphviz.org/
[5]http://www.r-project.org/

[6]stands for "forward sortation area", the first three characters in a Canadian postal code identifying a geographical region
[7]stands for "local delivery unit", the last three characters denoting a specific single address or range of addresses
[8]www.canadapost.ca/CPC2/addrm/hh/maps/FSA/CA01.pdf

#	**Impact of . . . on the effectiveness of ϕ_2**
1.1	the percentage of persons with an Ontario address (with error rate = 0.5%)
1.2	the percentage of persons with an Ontario address (with error rate = 2%)
1.3	the ratio of erroneous FSA to Province values (1:1)
1.4	the ratio of erroneous FSA to Province values (1:2)
1.5	the percentage of missing values in the list of valid Ontario FSA codes
1.6	the percentage of extraneous values in the list of valid Ontario FSA codes
#	**Impact of . . . on the effectiveness of ϕ_3**
2.1	the percentage of duplicated restaurant tuples (with error rate = 0.5%)
2.2	the percentage of duplicated restaurant tuples (with error rate = 2%)
2.3	the percentage of erroneous name values
2.4	the percentage of erroneous address values
2.5	the percentage of error-masking for name values[9].
2.6	the percentage of error-masking for address values

Table 2: Summary of scenarios for ϕ_2 and ϕ_3

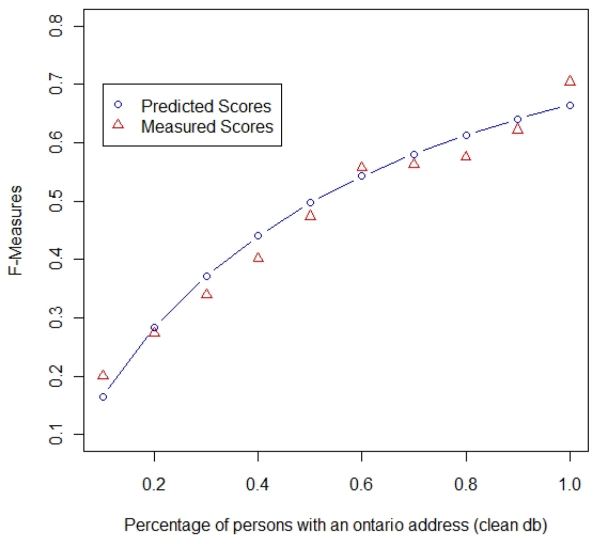
in Table 2. In all scenarios, we generate a total of 100,000 tuples. We show the measured and estimated effectiveness scores (expressed using F-Measure) for the first two scenarios for each rule (Figure 4 and 5). These results (and others not shown here) confirm that our approach generates formulas that give reasonable estimates of effectiveness scores under various circumstances.

5. RELATED WORK

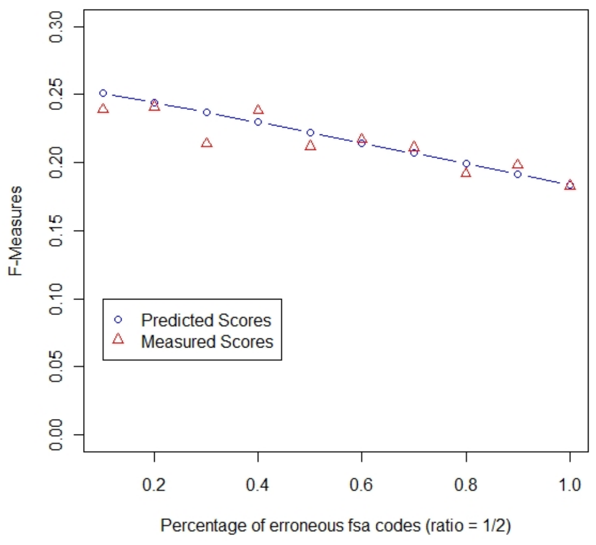
Constraint-based approaches for DQ assessment and improvement have recently received much attention both in the academic and professional communities. For example, unlike the traditional dependency theory, which was developed mainly for schema design, conditional functional dependencies (CFDs) have been studied for the purpose of data cleaning [4]. A CFD is functional dependency that holds only in a context where the context is specified using constants in a pattern tableau [4]. To cope with potentially large number of CFDs, several discovery algorithms have also been proposed [6, 8]. On the practical front, DQ rules (which may or may not be expressed as CFDs) are often derived from domain-specific business rules and through data profiling activities [14, 1].

Given a set of DQ rules, either discovered automatically or derived manually, a nature question is how we could quantify the usefulness of these rules in a DQ assessment effort; this is important when applying all available rules is unfeasible (since each rule incurs some costs as discussed in Section 1). In fact, [6] has already noted the lack of universal objective measures for DQ rules. But also notice that the measures in [6] are used to evaluate the validity of DQ rules while ours are used to evaluate their usefulness assuming they are valid. Our work is therefore a step toward objective cost-benefit evaluation of DQ rules.

The work closest to ours is the use of performance measures to evaluate and compare record linkage and deduplication algorithms [7]. For example, [9] proposed a framework for evaluating clustering algorithms in duplicate detection, in which precision, recall and F-Measures are used to measure the quality of output clusters (compared to ground truth clusters). However, performance measures in these approaches are obtained for a particular run of an algorithm on an actual dataset, and are often used as a mechanism to tune the parameters (e.g., matching threshold) that affect the performance of the algorithm. Our work instead focuses on the derivation of formulas that allow us to evaluate and compare rules under different circumstances without

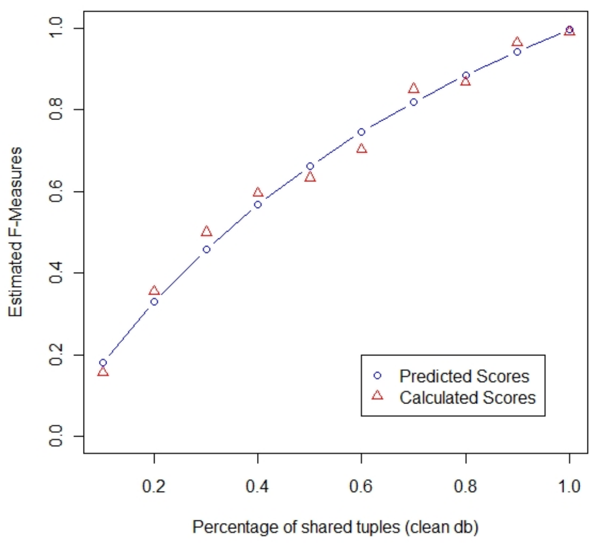


(a) Scenario 1.1

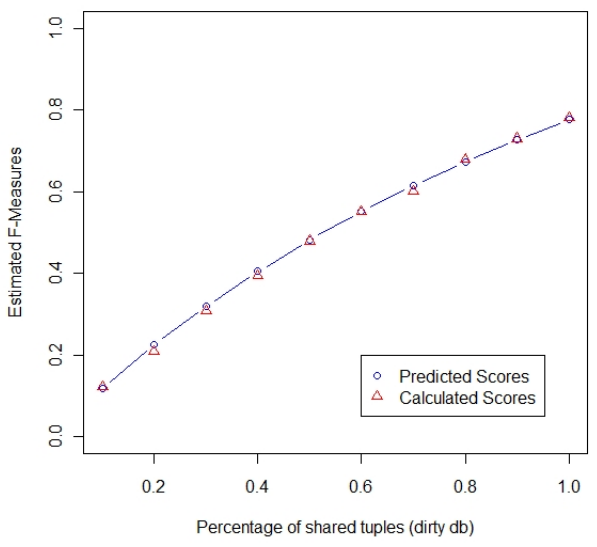


(b) Scenario 1.2

Figure 4: Canadian Postal Code Scenarios.



(a) Scenario 2.1



(b) Scenario 2.2

Figure 5: Restaurant Address Scenarios.

the need of actual datasets (as illustrated in Section 1; see [11] for more details).

Our approach aims at building, in a semi-automatic way, a probabilistic graphic model. However, our approach differs from the traditional applications of such models in an important way: the purpose of building a Bayesian network in most applications [13] is to perform probabilistic inference and learning that concern *individual* entities. For example, a Bayesian network can be used to update the probability of a DQ rule being violated (or a patient having a disease) when certain type of errors (or symptoms) are observed. In contrast, Our approach aims at enabling the *comparison* of two or more entities (i.e., DQ rules in our case) in terms of their probabilities in some special events (e.g., v_{TP}, v_{FN}, v_{FP}).

6. CONCLUSION

In this paper we have presented a systematic approach to derive effectiveness formulas for DQ rules. We introduced the concept of effectiveness for DQ rules, and identified the main factors that affect a rule's effectiveness. Our approach derives an effectiveness formula in three steps including the construction of a Belief Network graph, the assignment of symbolic probabilities to the events in the graph, and the generation of the formulas from the graph. We have also implemented and evaluated our approach — the evaluation results show that our approach generates formulas that give reasonable estimates of effectiveness scores under various circumstances.

In this paper we have been focusing on *internal events*, which concern directly values in a database. However, probabilities of these events are inevitably affected by those of *external events*, which concern the processes that generate, record and update these values. One advantage of considering external events is that they help to further break the estimation of probabilities of internal events into more manageable pieces. For example we may talk about what is the probability of a value being erroneous, given the fact it was entered by a trained professional (or a casual user). As the future work, we are aiming at building an ontology of such external events and extend our algorithm accordingly.

7. REFERENCES

[1] Maydanchik Arkady. Data quality assessment. 2007.

[2] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer, 1st edition, 2006.

[3] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48, New York, NY, USA, 2003. ACM.

[4] P. Bohannon, Wenfei Fan, F. Geerts, Xibei Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. *IEEE 23rd International Conference on Data Engineering*, pages 746–755, April 2007.

[5] Loreto Bravo, Wenfei Fan, Floris Geerts, and Shuai Ma. Increasing the expressivity of conditional

functional dependencies without extra complexity. *Data Engineering, International Conference on*, 0:516–525, 2008.

[6] Fei Chiang and Renée J. Miller. Discovering data quality rules. *Proc. VLDB Endow.*, 1(1):1166–1177, 2008.

[7] Peter Christen and Karl Goiser. Quality and complexity measures for data linkage and deduplication. In Fabrice Guillet and Howard J. Hamilton, editors, *Quality Measures in Data Mining*, volume 43 of *Studies in Computational Intelligence*, pages 127–151. Springer, 2007.

[8] Wenfei Fan, Floris Geerts, Laks V. S. Lakshmanan, and Ming Xiong. Discovering conditional functional dependencies. pages 1231–1234, 2009.

[9] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.

[10] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[11] Lei Jiang, Alexander Borgida, and John Mylopoulos. Measuring the effectiveness of data quality by design. In *the 21th International conference on Advanced Information Systems (CAiSE'09)*, 2009.

[12] Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping data in peer-to-peer systems: semantics and algorithmic issues. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 325–336, New York, NY, USA, 2003. ACM.

[13] Daphne Koller and Nir Friedman. Probabilistic graphical models: Principles and techniques (adaptive computation and machine learning). August 2009.

[14] Jack Olson. *Data Quality: The Accuracy Dimension*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.