

Estimating the Quality of Databases

Ami Motro

Igor Rakov

George Mason University

May 1998

Outline:

1. Introduction
2. Simple quality estimation
3. Refined quality estimation
4. Computing the quality of answers to arbitrary queries
5. Practical considerations
6. Application: Multidatabase harmonization of inconsistencies
7. Experimentation
8. Research directions

1. Introduction

Data quality: What is it?

- *Correctness*, precision, accuracy, soundness, validity.
- *Completeness*: all the data is available.
 - Completeness of files: no missing records.
 - Completeness of records: no missing fields.
- *Integrity and consistency*:
 - Integrity constraints are satisfied.
 - Proper relationships among different values are maintained.
- *Currency*: all data is up-to-date.
- Other?

Items 1 and 2 are “orthogonal”; the others are “related”.

Data quality: What can be done about it?

1. Protect

- Improve handling of data, enforce quality control procedures.
- Analogous to quality control in a production process.

2. Measure

- Adopt a data quality measure.
- Estimate the quality of the data according to the measure.

3. Improve

- Find obvious errors (“outliers”).
- Estimate, infer missing values (“imputation”).

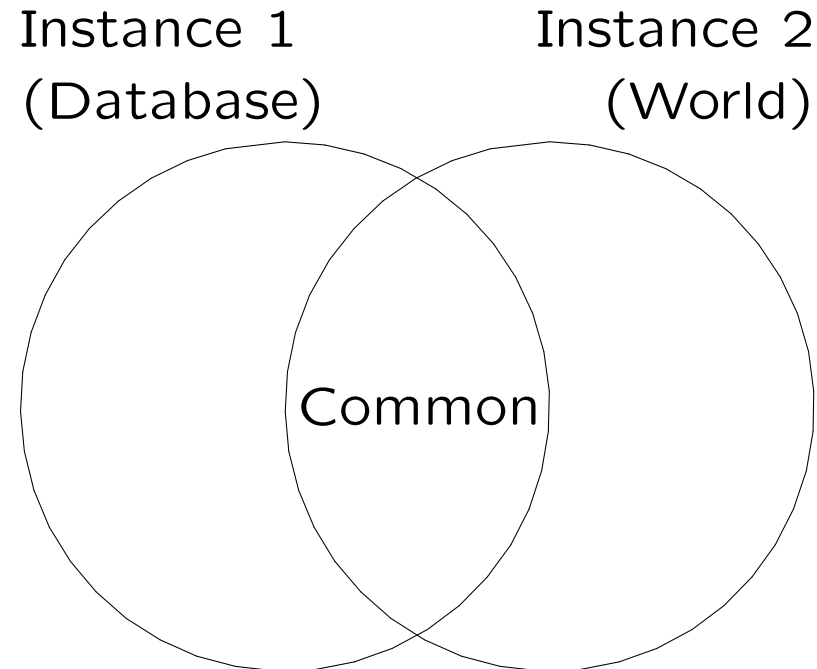
Advantages of data quality estimation

- Better decision-making when quality of data is known.
- Alternative sources of information may be compared, and the better source chosen.
- Value (price) of information is related to its quality.

2. Simple Quality Estimation

- Assume a database scheme with two instances:
 1. W : The *ideal (real-world)* instance; hypothetical, unavailable.
 2. D : The *actual (stored)* instance.
- The stored instance is an *approximation* of the ideal instance.
- How *good* is this approximation?
 - We need to *measure* the similarity of two database instances.
 - When one instance is the stored database and the other is the real-world instance, this measure will, in effect, calculate the *quality* of the stored instance.

The soundness and completeness measures



- Soundness (of Database relative to World): $\frac{|D \cap W|}{|D|}$
- Completeness (of Database relative to World): $\frac{|D \cap W|}{|W|}$

The soundness and completeness measures (cont.)

- Similar to *precision* and *recall* measures in information retrieval, where they are used to evaluate the goodness of answers.
- An intersection object (a database tuple that is also in the world) contributes to *both* soundness and completeness.
- A database tuple and a world tuple that differ (even in one attribute) adversely affect *both* soundness and completeness.
- Solution: All relations are *decomposed* to (key,nonkey) pairs. This provides more *refined* measures.
- Practicality: All measurements will be made on *samples*.

Estimating soundness

1. Sample D .
2. For each $x \in D_{\text{sample}}$ determine if $x \in W$.
3. Calculate soundness estimate.

Step 1 is *simple*; step 2 is *difficult*.

Step 2 requires *human verification* of the sample elements (determine their presence in W without actually constructing W).

Estimating completeness

1. Sample W .
2. For each $x \in W_{\text{sample}}$ determine if $x \in D$.
3. Calculate completeness estimate.

Step 1 is *difficult*; step 2 is *simple*.

Step 1 constructs a *fair challenge* to the database, by using various resources (e.g., by sampling alternative databases).

3. Refined Quality Estimation

- **Problem:** The quality of information sources is often *heterogenous*:
 - An information source may provide excellent information in one “area”, and poor information in another “area”.
- **Solution:** *Partition* the database to “areas” that are *homogeneous* with respect to their quality:
 - Any “subarea” of a homogeneous “area” would inherit roughly the same quality.
- Separate partitions needed for soundness and completeness.
- The partitions (and the corresponding quality measurements of their views) provide a *quality specification (quality basis)* for the database.

Quality specification

- “Areas” and “subareas” are defined with *views*.
- Views (and, later, queries) may involve projection, selection, and Cartesian product; projections must retain key attributes.
- *Homogeneity (wrt soundness)* of a view v is defined by

$$H_S(v) = 1/N \sum_{i=1}^N |s(v) - s(v_i)|$$

where v_1, \dots, v_n are the possible subviews of v , and $s(v)$ is the soundness of v .

- $H_S(v) = 0$ if the view elements are all true or all false.
- Analogously, for homogeneity wrt completeness.
- Problem: prohibitively expensive to compute.
- For now, assume we have a method to derive quality (soundness and completeness) bases.

4. Estimating the Quality of Answers to Queries

- Each query is a *chain* of individual relational operations.
- Each relational operation is extended to produce (in addition to the result) a *quality basis* of the result.
- Each operation in the chain receives a quality basis from the previous operation and uses it to derive a quality basis for the next operation.
- This allows us to calculate the quality of *sequences* of relational operations (a general query).

Estimating the quality of answers to queries (cont.)

At the final step, the *overall quality* of the answer is computed from the final quality basis:

- A quality basis *partitions* the answer.
- Hence, each answer is partitioned into views with known quality.
- Overall quality is computed as a *weighted sum* of the quality of these views.
- If we have only *estimates* of quality, then the *variance* of these estimates is also calculated

List of algorithms/computations

1. Construct a quality basis for the result
 - Projection-selection
 - Cartesian product
2. Estimate the quality of each view of the basis
 - Projection-selection
 - Cartesian product
3. Estimate the overall quality of the result
 - Projection-selection
 - Cartesian product
4. Compute the variance of the overall quality
 - Projection-selection
 - Cartesian product

5. Practical Considerations

- By our definitions, bases are partitions of perfectly homogeneous views.
 - The homogeneity measure H is expensive to compute.
 - Possibly, the only homogeneous views to be found will have very few tuples.
- Instead, apply the same techniques to views that are only “highly homogeneous” .
 1. The result of a projection-selection on a highly homogeneous view will be (with high probability) also highly homogeneous.
 2. Homogeneity of the result of a Cartesian product will be close to the largest (worst) of the homogeneities of the input views.

Gini Index

- A simple-to-compute alternative from statistical classification theory to our homogeneity measure.
- Gini index of a view v :

$$2p_0 \cdot p_1 = 2p_1(1 - p_1) = 2p_0(1 - p_0)$$

p_0 : proportion of the incorrect elements in v

p_1 : proportion of the correct elements in v

- Analogously, for completeness.

Finding quality bases with approximately homogeneous views

- Done separately for soundness and completeness.
- Done separately for each relation.
- Performed only on samples.
- Basis views are restricted to projections (that retain key attributes) and selections.

Algorithm

Input: A sample of the relation; a threshold value.

Output: Set of views on the relation that are (approximately) homogeneous with respect to a quality measure.

Step 1: Label the relation as the root node, make it the current node v , and put it in a queue Q .

Step 2: Consider all possible splits of the current node into two subnodes v_1 and v_2 and measure the reduction of each split.

Reduction of a split: $\Delta G = G(v) - \alpha_1 G(v_1) - \alpha_2 G(v_2)$,
where $G(v)$ is the Gini index of v , and $\alpha_i = |v_i|/|v|$.

Step 3: Select a split that maximizes the reduction, and split v .

Algorithm (cont.)

Step 4: If the reduction of this split is greater than the threshold value, put subnodes v_1 and v_2 in Q .

Otherwise, make v a leaf node and go to Step 5.

Step 5: If Q is not empty, go to Step 2. Otherwise, stop.

Comments:

- The threshold value is needed to prevent splitting down to individual elements.
- Step 2 is computationally expensive and should be refined to reduce search space and make use of heuristic information.

6. Multidatabase Harmonization of Inconsistencies

- **Multidatabase Integration:** How to provide integrated access to a collection of autonomous and heterogeneous databases.
- **The usual focus:** How to resolve *intensional inconsistencies* among the participating databases (semantic heterogeneity).
- **The Multiplex project** also addresses the problem of *extensional inconsistencies*:
 - Do not assume overlapping sources are mutually consistent.
 - Develop methods for *harmonizing* inconsistent answers.

Quality-based approach to harmonization

- Assume a query Q has answers A_1, \dots, A_n .
- Assume there is no information regarding quality:
 $s(A_i) = 1.0$ and $c(A_i) = 1.0$ for $i = 1, \dots, n$
(obviously, these claims are inconsistent).
- Heuristic: Answer providers *vote* on the answer space $\bigcup_{i=1}^n A_i$:
If $x \in A_i$ then provider i votes "yes" on x (soundness).
If $x \notin A_i$ then provider i votes "no" on x (completeness).
- Define:
 $L = \{x \mid x \text{ received only "yes" votes}\}$
 $U = \{x \mid x \text{ received at least one "yes" vote}\}$
- The real answer A is "sandwiched" between L and U :
 $L \subseteq A \subseteq U$.
- In-between layers can be created:
 $E_i = \{x \mid x \text{ received at least } i \text{ "yes" votes}\}$

Quality-based approach to harmonization (cont.)

- Assume now that each answer A_i has its individual quality specifications: $s(A_i) = s_i$ and $c(A_i) = c_i$.
- Using probabilistic arguments, the voting scheme is extended:
If $x \in A_i$ then provider i votes on x

$$P(x \in A | x \in A_i) = s_i$$

If $x \notin A_i$ then provider i votes on x

$$P(x \in A | x \notin A_i) = \frac{(s_i/c_i) - s_i}{(n/|A_i|) - 1}$$

where n is the size of the entire answer space.

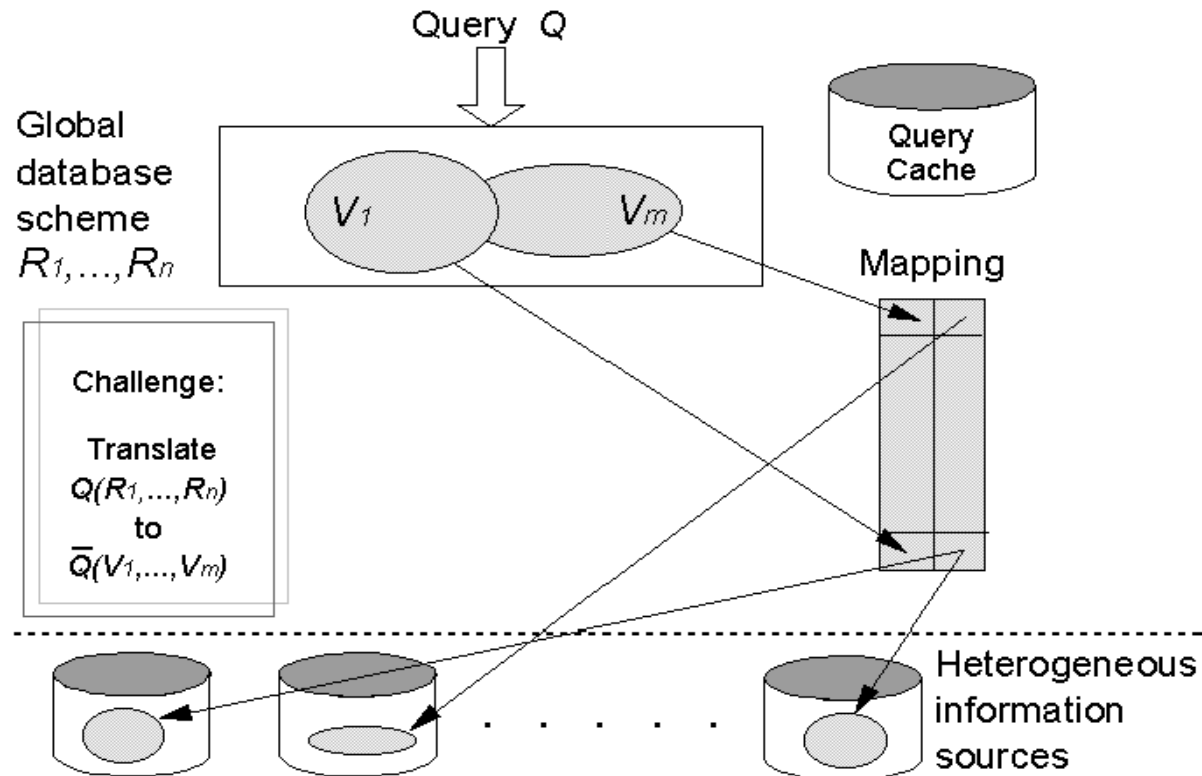
- Votes are combined to provide *ranking* of the answer.
- Allows handling of queries that prespecify answer cardinality.
- **Note:** Data quality is just one way to resolve inconsistencies in Multiplex. Essentially, the multidatabase designer specifies a *conflict resolution strategy*, whenever the potential for inconsistencies exists (i.e., contributed views overlap).

The Multiplex model: important features

1. **Extension** of the relational model.
2. Formal assumptions of **schema and instance integrability**.
3. **Abstraction** of many ad-hoc solutions.
4. Full support for **heterogeneity**.
5. **Flexibility**: global to local scheme mappings are neither *total*, nor *onto*, nor *single-valued*.
6. Support for **approximative answers** when some information is either *unavailable* or *inconsistent*.
7. Quick adaptation to **evolving environments**.
8. **Practical platform** for implementation.

The Multiplex software system

Diagram



URL: <http://www.isse.gmu.edu/~multiplex>

The Multiplex system (cont.)

Highlights

- **WWW application:** Uses HTTP Protocol to communicate with contributing systems; user interface is accessible via WWW browser; contributing systems invoked by cgi-scripts.
- **Heterogeneity:** Presently integrates information from (1) relational (Oracle); (2) object-oriented (Ode); (3) simple files (Unix shell scripts); (4) Wide Area Information Services (WAIS), (5) spreadsheets (Excell), (6) menu-based (Xlibris).
- **Answer format:** Presently, answers assumed to claim perfect soundness and completeness; combined answers are presented (using colors) as two relations: (1) the sound estimate; (2) the tuples that augment it to a complete estimate.
- **Query language:** Conjunctive queries with aggregates and ordering (both SQL and point-and-click interfaces available).

7. Experimentation

- Designed to test viability and performance of our methods.
- Used synthetic data.
- Relative error (relative difference between computed soundness and real soundness) was used as the measure of performance.

The quality toolkit demo

1. Select a data set, an error pattern, and a sampling rate.
2. Sample the data set.
3. Apply algorithm to find a quality (soundness) specification.
4. Submit an arbitrary query.
5. Compute its answer.
6. Apply algorithm to estimate answer quality.
7. Compare the estimate with the actual quality rate.

<http://www.isse.gmu.edu/~irakov/demo/demo.html>

Performance

- Pre-assigning errors is the only deviation from actual estimation.
 1. It enables skipping manual verification.
 2. It provides the actual quality for comparison.
- Observations
 1. Error decreases as sample size increases.
 2. Error decreases as answer cardinality increases.
 3. Experimentation needed to find the optimal threshold value (often in the range 0.3–0.5).
- With sample size of about 15% and answer cardinality of about 400, the relative error is in the range of 4%–6%.

8. Research Directions

1. Reduce the lab work: “discover” quality “automatically”.
2. Dynamic databases: update the quality estimates.
3. We only considered whether the database values are *identical* to the real values. How about basing quality on their *similarity* to the real values?