ERACER: A Database Approach for Statistical Inference and Data Cleaning

Chris Mayfield       Jennifer Neville       Sunil Prabhakar

Department of Computer Science, Purdue University
West Lafayette, Indiana, USA

SIGMOD 2010, Indianapolis
Problem

Real data is *dirty*
- Inconsistent, incomplete, corrupted, outdated, etc.
- Safety measures (e.g., constraints) are often not used
- Poor decisions based on dirty data costs billions annually

Data *cleaning* is hard
- Typically ad hoc, interactive, exploratory, etc.
- Uncertain process: what to do with the “errors?”
- Maintenance of results (e.g., lineage/provenance)
- Consumes large amount of data management time

*(see Fan, Geerts, & Jia, VLDB 2008 tutorial)*
Example 1: Genealogy Data

5M people from *Pedigree Resource File*

- Person (ind_id, birth, death)
- Relative (ind_id, rel_id, role)

Integrated from many sources, e.g.:

- Census records
- Immigration lists
- Family history societies
- Individual submissions
Example 2: Sensor Networks

- 54 sensors, every 31 seconds, for 38 days
- $\approx 18\%$ obviously incorrect
- Multiple data types

2M readings of *Intel Lab Data*

- Sensor (epoch, mote_id, temp, humid, light, volt)
- Neighbor (id1, id2, distance)

Source: [http://db.csail.mit.edu/labdata/labdata.html](http://db.csail.mit.edu/labdata/labdata.html)
Insight

Correlations within tuples, e.g.:
- Birth and death years
- Temperature and humidity values

Correlations across tuples, e.g.:
- Parents and children
- Neighboring sensors

Apply statistical relational learning
- Don’t just clean tuples in isolation (e.g., functional dependencies)
- Propagate inferences multiple times

Input:
- Possible tuple dependencies
- Correlation model skeleton

Output:
- PDFs for missing data
- Flags for dirty data
Baseline Approach:

Bayesian networks

Exact inference (junction tree)

Bayes Net Toolbox for Matlab
Bayesian Network Formulation

Model *template* specifies conditional dependencies:

![Bayesian Network Diagram]

Conditional probability distribution (CPD) at each node:

\[
\begin{align*}
P(I.d \mid I.b) & \quad \text{death year, given the birth year} \\
P(I.b \mid M.b, F.b) & \quad \text{birth year, given parent birth years}
\end{align*}
\]

Prior distribution at nodes with no parents: \( P(I.b) \)

*Simplified version of Relational Bayesian Networks (see e.g., Getoor & Taskar 2007)*
1. Learn CPDs from data, e.g.:

```sql
CREATE TABLE cpt_birth AS
    SELECT birth, death, count(*)
    FROM person
    GROUP BY birth, death;
```

2. Share CPDs across all nodes:

```sql
-- P(I.d | I.b = 1750)
SELECT death, count
FROM cpt_birth
WHERE birth = 1750;
```

3. Run inference (e.g., junction tree)
   - Construct Bayesian network
   - Bind evidence (query from DB)
   - Extract results (store in DB)
Challenges and Lessons Learned

Limiting model assumptions
- Fixed CPD structure (e.g., always two parents)
- Acyclicity constraint (can’t model sensor data)

Potentially millions of parameters
- Becomes very inefficient
-Floating point underflow

Not scalable to large data sets
- DB may not fit into main memory
- Moving data in/out of R, Matlab, etc.

Not designed for data cleaning
- Propagates outliers/errors in original data
- Need to look beyond the Markov blanket
ERACER Approach:

Relational dependency networks
Approximate inference algorithm
SQL-based framework
Integrated data cleaning
Relational Dependency Networks

For example, at each sensor and time epoch:

In contrast to Bayesian networks, RDNs:

- approximate the full joint distribution
- learn CPDs locally based on component models
- allow cyclic dependencies (i.e., many-to-many)
- use aggregation to deal with heterogeneity

(see Neville & Jensen, JMLR 2007)
Component Models

Convolution (for genealogy)

- **parent age**: \( M_{PA} = P(I.b - P.b) \)
- **death age**: \( M_{DA} = P(I.d - I.b) \)

-- death age model

\[ \text{SELECT } \text{hist}(\text{death} - \text{birth}) \]
\[ \text{FROM } \text{person}; \]

Regression (for sensors)

- **mean temperature**: 
  \[ S.t \sim \beta_0 + \beta_1 \cdot S.h + \beta_2 \cdot \text{avg}(N.t) + \beta_3 \cdot \text{avg}(N.h) \]
- **mean humidity**: 
  \[ S.h \sim \gamma_0 + \gamma_1 \cdot S.t + \gamma_2 \cdot \text{avg}(N.t) + \gamma_3 \cdot \text{avg}(N.h) \]
Learning (one time, offline):
1. Extract graph structure using domain knowledge
2. RDNs aggregate existing data to learn parameters

Inference (multiple iterations):
3. Apply component models to every value in DB
4. Combine predictions to deal with heterogeneity
5. Evaluate posterior distributions for cleaning
6. Repeat 3–5 until happy (i.e., results converge)
Step 1: Extract Graphical Model

Construct nodes:

```sql
INSERT INTO node
SELECT make_nid(epoch, mote_id), -- creates simple key
      new_basis(temp), new_basis(humid),
      new_basis(light), new_basis(volt)
FROM sensor;
```

<table>
<thead>
<tr>
<th>basis data type:</th>
<th>initial</th>
<th>pdf</th>
<th>suspect</th>
<th>round</th>
</tr>
</thead>
<tbody>
<tr>
<td>original value, if any (e.g., humid)</td>
<td>current prediction (or distribution)</td>
<td>data cleaning flag (true = outlier)</td>
<td>when pdf/suspect was last updated</td>
<td></td>
</tr>
</tbody>
</table>

Construct edges:

```sql
INSERT INTO link
SELECT make_nid(a.epoch, a.mote_id),
      make_nid(b.epoch, b.mote_id)
FROM neighbor AS c -- e.g., within 6 meters
INNER JOIN sensor AS a ON c.id1 = a.mote_id
INNER JOIN sensor AS b ON c.id2 = b.mote_id
WHERE a.epoch - 30 <= b.epoch
AND a.epoch + 30 >= b.epoch;
```
Step 2: Learn RDN Parameters

Aggregate original data values:

```sql
CREATE TABLE learn AS
SELECT
    -- individual instances
    min(expect(i.t)) AS ti, min(expect(i.h)) AS hi,
    min(expect(i.l)) AS li, min(expect(i.v)) AS vi,
    -- average neighbor values
    avg(expect(n.t)) AS tn, avg(expect(n.h)) AS hn,
    avg(expect(n.l)) AS ln, avg(expect(n.v)) AS vn
FROM node AS i
    LEFT JOIN link AS l ON i.nid = l.id1
    LEFT JOIN node AS n ON l.id2 = n.nid
GROUP BY i.nid;
```

Optional: apply noise filters, sample data, etc.

Estimate applicable component models

- Convolution: use built-in `hist` aggregate
- Regression: export to R; use `lm` function
Steps 3–6: Approximate Inference

For each round of inference:

1. Update predictions via the `erase` aggregate query
   - Infers/cleans all attributes in a single function call
     
     ```sql
     SELECT `erase`(i, n)
     FROM node AS i
     LEFT JOIN link AS l ON i.nid = l.id1
     LEFT JOIN node AS n ON l.id2 = n.nid
     GROUP BY i;
     ```

   Key design choice: grouping by tuples, not attributes

2. Store results via `CREATE TABLE AS` (i.e., propagation)
   - Faster than `UPDATE` over the entire relation (MVCC)
   - Other optimizations possible (e.g., indexes on `nid`’s)
For each attribute in $i$:

- Select applicable model
- Apply/combine predictions
- Evaluate (cf. init and prev)

Data cleaning algorithm:

- Run inference for known values, as if missing
- Is original evidence within expected range?
- Replace outliers with inferred distributions
- Do not propagate suspects (rely on other data)

Many more details in the paper!
Experiments:

Generate synthetic populations
Randomly set attributes to NULL
Compare inferred values to original
Genealogy Data Results

Accuracy of birth pdfs:

Variance (uncertainty):
Sensor Data Results

Accuracy of temperature pdfs:

Accuracy of humidity pdfs:
Summary

Database-centric approach for approximate inference
  ▶ Statistical framework for correcting errors
  ▶ Efficient; no need to move data to/from R/Matlab

Synergy of imputation and data cleaning tasks
  ▶ Additional evidence identifies errors more accurately
  ▶ Corrected data values improve the quality of inference

Empirical evaluation on two real-world data sets
  ▶ Similar accuracy to Bayesian network baseline
  ▶ Significant gains in runtime performance
  ▶ Added benefit of simultaneous data cleaning