Query-Oriented Data Cleaning with Oracles

Yahui Xiong
Usama Naseer
Outline

• Biggest takeaways
• Background: Datalog
• Overview with example
• Preliminary
• Model, Architecture
• Removing a wrong answer
• Add missing answer
• Iterative cleaning algorithm
• Evaluation
Biggest takeaway points

• QOCO is a novel query-oriented framework for data cleaning with oracle crowds.
• QOCO removes incorrect answer from, and add missing data to result of a query through edits.
• Greedy approach is used for wrong answer deletion and split.
• No need for the whole dataset to be cleaned. Only query oriented data should be cleaned.
Datalog

• Logical query language for the relational model
• Datalog consists of "if-then" rules. A datalog rule is

\[ Q(args) \ :- \ R1(args1), \ R2(args2), \ ... \]

also can be written as:

\[ Q(args) \ :- \ R1(args1) \ \text{AND} \ R2(args2) \ \text{AND} \ ... \]

Datalog, cont'd

• Terminologies and definitions:

  the rule *head* = Q(args)

  the rule *body* = R1(args1), R2(args2), ...

  *atom* or *subgoal* = any one of Ri(argsi)

  head variables = the variables occurring in the head
  existential variables = all the other variables

  • Ri(argsi) evaluates to true when relation Ri contains the tuple described by argsi.
Datalog, cont'd

• Example:

Consider the following schema:
Purchase(pid, product, price, quantity)
Product(pname, manufacturer)
(Purchase.product refers to Product.pname)

\[ A(y) \leftarrow \text{Purchase}(x,y,z,u), \quad z < 9.99 \]

Meaning: Find all products under 9.99
## Overview with example

### Games

<table>
<thead>
<tr>
<th>Date</th>
<th>Winner</th>
<th>Runner-up</th>
<th>Stage</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.07.14</td>
<td>GER</td>
<td>ARG</td>
<td>Final</td>
<td>1:0</td>
</tr>
<tr>
<td>11.07.10</td>
<td>ESP</td>
<td>NED</td>
<td>Final</td>
<td>1:0</td>
</tr>
<tr>
<td>09.07.06</td>
<td>ITA</td>
<td>FRA</td>
<td>Final</td>
<td>5:3</td>
</tr>
<tr>
<td>30.06.02</td>
<td>BRA</td>
<td>GER</td>
<td>Final</td>
<td>2:0</td>
</tr>
<tr>
<td>12.07.98</td>
<td>ESP</td>
<td>NED</td>
<td>Final</td>
<td>4:2</td>
</tr>
<tr>
<td>17.07.94</td>
<td>ESP</td>
<td>NED</td>
<td>Final</td>
<td>3:1</td>
</tr>
<tr>
<td>08.07.90</td>
<td>GER</td>
<td>ARG</td>
<td>Final</td>
<td>1:0</td>
</tr>
<tr>
<td>11.07.82</td>
<td>ITA</td>
<td>GER</td>
<td>Final</td>
<td>4:1</td>
</tr>
<tr>
<td>25.06.78</td>
<td>ESP</td>
<td>NED</td>
<td>Final</td>
<td>1:0</td>
</tr>
</tbody>
</table>

### Teams

<table>
<thead>
<tr>
<th>Country</th>
<th>Continent</th>
</tr>
</thead>
<tbody>
<tr>
<td>GER</td>
<td>EU</td>
</tr>
<tr>
<td>ESP</td>
<td>EU</td>
</tr>
<tr>
<td>BRA</td>
<td>EU</td>
</tr>
<tr>
<td>NED</td>
<td>SA</td>
</tr>
<tr>
<td>ITA</td>
<td>EU</td>
</tr>
</tbody>
</table>

### Players

<table>
<thead>
<tr>
<th>Name</th>
<th>Team</th>
<th>Birth year</th>
<th>Birth place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mario Götze</td>
<td>GER</td>
<td>1992</td>
<td>GER</td>
</tr>
<tr>
<td>Andrea Pirlo</td>
<td>ITA</td>
<td>1979</td>
<td>ITA</td>
</tr>
<tr>
<td>Francesco Totti</td>
<td>ITA</td>
<td>1976</td>
<td>ITA</td>
</tr>
</tbody>
</table>

### Goals

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mario Götze</td>
<td>13.07.14</td>
</tr>
<tr>
<td>Andrea Pirlo</td>
<td>09.06.06</td>
</tr>
<tr>
<td>Francesco Totti</td>
<td>09.06.06</td>
</tr>
</tbody>
</table>
Overview with example

\[ D: \text{dirty database} \]
\[ D_G: \text{corresponding correct ground truth database} \]

\[ (x) :- \ \text{Games}(d_1, x, y, Final, u_1), \ \text{Games}(d_2, x, z, Final, u_2), \]
\[ \ \text{Teams}(x, EU), d_1 \neq d_2. \]

- Find European teams that won the World Cup at least twice.
- result: \{(GER),(ESP)\} (Spain wrong, Italy missing)
- Update D
- Asking crowd
Preliminaries

• Notations:
  \( \mathcal{V} \): a fixed set of variables
  \( \mathcal{C} \): be a fixed set of constants called the underlying vocabulary.
  A query \( Q \) over a relational schema \( S \) is an expression of the form

\[
\text{Ans}(\bar{u}_0) :: R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n), E_1, \ldots, E_m
\]
Preliminaries, cont'd

• In previous example query

\[ Var(Q_1) = \{d_1, d_2, x, y, u_1, u_2\} \]
\[ Const(Q_1) = \{Final, EU\} \]
\[ Q_1(D), \text{ contains 2 answers } \{GER, ESP\} \]

• Assignments and Results
  - An **assignment** \( \alpha : Var(Q) \rightarrow C \) for a query \( Q \) is a mapping from the variables of \( Q \) to constants
  - A **partial assignment** for \( Q \) is an assignment which may not be total

• Witness
  
  Let \( \alpha \) be an assignment for \( Q \) that is valid w.r.t. database \( D \). A **witness** for \( \alpha \) consists of all facts in \( \alpha(body(Q)) \).
Models

• (True Answer, True Result) A tuple $t$ is a true answer to a query $Q$ and database $D$ if $t \in Q(D)$ and $t \in Q(D_G)$. We call $Q(D_G)$ the true result of $Q$.

• (Missing Answer) A tuple $t$ is a missing answer to a query $Q$ and database $D$ if $t \in (Q(D_G) - Q(D))$.

• (Wrong Answer) A tuple $t$ is a wrong answer to a query $Q$ and database $D$ if $t \in (Q(D) - Q(D_G))$.

• Each question-and-answer is an interaction with the crowd and each update is called an edit.

• Insertion edit, deletion edit
Models, cont'd

• Problem to be solved:
  interact with the crowd minimally to derive a sequence of edits to achieve $D'$, $D'$ has same query result as ground truth DB.

• Result can be equal even if $D'$ is still dirty/incomplete
Architecture

• Target actions: deletion/addition
• Crowd: single crowd member, a perfect oracle/a collection of imperfect oracles
• Questions to crowd members: boolean, others
• Workflow
  - there exists a naïve strategy that guarantees that the workflow always converges for a specific target action under certain condition
Removing wrong answer

Algorithm 1: CrowdRemoveWrongAnswer

**Input**: A query $Q$, a database $D$ and a wrong tuple $t$.

**Output**: A list of deletion edits.

**Init**: DeletionEdits $= \emptyset$, $S = \text{wit}(A(t, Q, D))$

1. while $S \neq \emptyset$ do
2.   foreach Singleton $s = \{R'(\bar{a}')\}$ in $S$ do
3.     DeletionEdits $\leftarrow R'(\bar{a}')^-$
4.     Remove from $S$ all sets that contain $R'(\bar{a}')$
5.   if $S \neq \emptyset$ then
6.     $R(\bar{a}) = \text{MostFrequentTuple}(S)$
7.     if CrowdVerify($R(\bar{a})$) then
8.       $S = \{s \setminus \{R(\bar{a})\} \mid s \in S\}$
9.     else
10.      Remove from $S$ all sets that contain $R(\bar{a})$
11.      DeletionEdits $\leftarrow R(\bar{a})^-$
12. return DeletionEdits
Removing wrong answer, cont'd

• Algorithm:
  1) ask crowd about tuples that hit the largest # of witnesses
  2) find a frequent tuple and delete it
  3) repeat 2) until unique minimal hitting set exists, or all witnesses destroyed

• Relative definition: Hitting Set

  Consider the pair \((U, S)\) where \(U\) is a universe of elements and \(S\) is a set of subsets of \(U\). A set \(H \subseteq U\) is a hitting set if \(H\) hits every set in \(S\). In particular, \(H \cap S' \neq \emptyset\) for every \(S' \in S\). A minimal hitting set \(H\), is s.t. \(\forall e \in H. H \setminus \{e\}\) is not a hitting set.

• Theorem:

  Given a pair \((U, S)\), a unique minimal hitting set exists if and only if the elements of the singleton sets of \(S\) forms a hitting set for \(S\).
Removing wrong answer, cont'd

- Proof:

Let $M$ denote the elements that occur in all singleton sets of $S$. 

$M$ is a hitting set $\rightarrow M$ is also a unique minimal hitting set (because none of the elements of $M$ can be removed).

Next we prove that, a unique minimal hitting set exists $\rightarrow M$ must be a hitting for $S$.

Assume by contradiction that the hitting set must include, in addition to the elements in $M$, some other element that hits a set $s' \in S$. Then, $s'$ must contain at least 2 elements and the elements of $s'$ do not occur among $M$. If this is the case, different elements of $s'$ will constitute to different minimal hitting sets.
Removing wrong answer, cont'd

- Example

<table>
<thead>
<tr>
<th></th>
<th>Tuples of the witness</th>
</tr>
</thead>
</table>
| $w_1$ | $t_1 = \text{Games}(11.7.10, ESP, NED, Final, 1:0)$  
      | $t_2 = \text{Games}(12.7.98, ESP, NED, Final, 4:2)$  
      | $t_3 = \text{Teams}(ESP, EU)$ |
| $w_2$ | $t_2 = \text{Games}(12.7.98, ESP, NED, Final, 4:2)$  
      | $t_4 = \text{Games}(11.7.94, ESP, NED, Final, 3:1)$  
      | $t_3 = \text{Teams}(ESP, EU)$ |
| $w_3$ | $t_4 = \text{Games}(11.7.94, ESP, NED, Final, 3:1)$  
      | $t_1 = \text{Games}(11.7.10, ESP, NED, Final, 1:0)$  
      | $t_3 = \text{Teams}(ESP, EU)$ |
| $w_4$ | $t_1 = \text{Games}(11.7.10, ESP, NED, Final, 1:0)$  
      | $t_5 = \text{Games}(25.06.78, ESP, NED, Final, 1:0)$  
      | $t_3 = \text{Teams}(ESP, EU)$ |
| $w_5$ | $t_2 = \text{Games}(12.7.98, ESP, NED, Final, 4:2)$  
      | $t_5 = \text{Games}(25.06.78, ESP, NED, Final, 1:0)$  
      | $t_3 = \text{Teams}(ESP, EU)$ |
| $w_6$ | $t_4 = \text{Games}(11.7.94, ESP, NED, Final, 3:1)$  
      | $t_5 = \text{Games}(25.06.78, ESP, NED, Final, 1:0)$  
      | $t_3 = \text{Teams}(ESP, EU)$ |
Removing wrong answer, cont'd

• Process & result from last example:

  ->  \{t_1, t_2\}, \{t_2, t_4\}, \{t_4, t_1\}, \{t_1, t_5\}, \{t_2, t_5\}, \{t_4, t_5\}

  ->  \{t_1, t_2\}, \{t_2, t_4\}, \{t_4, t_1\}

  ->  \{t_2\}, \{t_2, t_4\}, \{t_4\}
Adding a missing answer

• The second action in cleaning data-set is to add missing values.
• Set of k corrective updates to add missing values to query result.
• Given $D, D_g, Q, t \in (Q(D_g) - Q(D))$, generate at most $k$ questions $q_1, ..., q_k$ of the type $\text{TRUE}(R(\overline{a}))$? s.t. $t \in Q(D \oplus \text{ans}(q_1) \oplus ... \oplus \text{ans}(q_k))$. 
NP-hard

• NP-hard, can be reduced from One-3SAT.
• Heuristics are needed in the design of an efficient algorithm to generating right questions about the relevant tuples to be inserted.
• Leverage crowd’s knowledge of ground truth.
• Authors give legitimacy to their heuristic based algorithm.
What kind of questions to ask?

• Two type of questions asked:
  • i) TRUE(t)? Is the tuple t present in Dg.
  • ii) COMPL(α,Q) where α is partial assignment for Q.
• If α is satisfiable, add something to α to make it a complete witness.
• Otherwise just ignore it.
• More of a task than a question.
Head and Body of Q

- Head – Left hand side of query. Not a relational instance from database. Contains all variables in body.
- Body – Right hand side of query. Relational instances from database. Consists of Var(Q) and Const(Q)
- For example, for query “date when a German player scored goal in worldcup”
- ans(x,y,z,w) :- Players(x, GER, y, z), Goals(x, w)
Query split

• Entire witness generation may be too large for the crowd.

• Easier for crowd if query is broken down into simpler pieces.

• Sub-query (a subset of original query such that the overall effect of all subqueries is the same as original query)

  • For example for average, sum and division can be two subqueries.

• Split Q in such a way that each relational instance is included at least once in the body of subqueries.
Algorithm for adding missing values

• Find sub-queries of Q
• If a sub-query is satisfiable using data from dirty database D, as crowd COMPL() question on this partial assignment.
• Ask the crowd to suggest a tuple to complete the assignment, using their knowledge of ground truth data base.
Example:

• Query: Find all EU players who have scored in WC final.

• Formally, $(x)$:- Players $(x, y, z, w)$, Goals$(x, d)$, Games$(d, y, v, Final, u)$, Teams$(y, EU)$.

• $x$ (player name), $y$ (team), $z$ (birth year), $w$ (birth place), $d$ (game date), $v$ (runner up), $u$ (score)

• Result contains:

Mario Gotze, Francesco Totti
Example:

- Pirlo does not appear in result since he’s missing from D.
- Let’s consider Pirlo’s case
- Split Q into sub-queries
- Next we find valid assignment for sub-queries using D.

  - Notice there is one valid assignment for $Q'$ w.r.t. $D$: $\alpha_1 = \{y = w \rightarrow \text{ITA}, z \rightarrow 1979, d \rightarrow 9.6.06, v \rightarrow \text{FRA}, u \rightarrow 5:3\}$
  - 3 valid assignment for $Q''$ w.r.t. $D$: $\alpha_2 = \{y \rightarrow \text{GER}\}$, $\alpha_3 = \{y \rightarrow \text{ESP}\}$, $\alpha_4 = \{y \rightarrow \text{BRA}\}$
  - For Pirlo to be in result, (ITA,EU) needs to be in Teams
  - The partial assignment $\alpha_1$ becomes satisfiable if (ITA,EU) tuple is added to Teams. All other partial assignments are unsatisfiable.
Example:

- Next step: Ask crowd if $\alpha_1$ is valid.
- Crowd uses its knowledge of Dg to assert that $\alpha_1$ is valid.
- QOCO concludes that (ITA, EU) needs to be in Teams for Pirlo to be in the results.
- For $\alpha_2$, $\alpha_3$, $\alpha_4$, the crowd answers in negation since these are not valid assignments. Eg, $\text{Players(Pirlo, GER, z, w)} \alpha_2(body(Q2|t))$ or $\text{Players(Pirlo, ESP, z, w)}$ in $\alpha_3(body(Q2|t))$, that cannot be completed into a fact of Dg.
- Oracle’s work is reduced to a question whether assignment is legal or not.
Caveats

• There can be an exponential number of sub-queries.
• Splitting the query can be tricky, unlike the simple example case.
Greedy Insertion Algorithm:

• Split the input query and add subqueries to queue (line 3)

• At each iteration, pop a sub-query and evaluate against crowd if it is:
  
  (i) valid total assignment (8-10)

  (ii) partial assignment that should be completed (12-15)

• If not, we split the current query (16-17)

• If the algorithm fails to find a partial assignment that can be extended into a valid assignment for $Q$ it posts to the crowd a question to provide a witness for the missing answer (line 18).

• In line 19 the algorithm executes the insertions of the true missing tuples.
Query splitting methods

• Data-directed approach
  • Exploits provenance metadata information of database. Information concerning the creation, attribution, or version history of managed data.
  • Uses the output of WhtNot? System.

• Query-directed approach
  • Takes query as a weighted graph, vertices are tuple with edge between vertices sharing common variable or inequality.
  • Find the mincut in the graph.
Input Query: \((x, y, z, w) := R_1(x, y), R_2(y, z), R_3(z, w), R_4(z, v); z \neq x, w \neq x\)

**Min-Cut**

**WhyNot?**

WhyNot? outputs a join operator of

\(O_1 = \{R_1(x, y), R_2(y, z), z \neq x\}\) and

\(O_2 = \{R_3(z, w), R_4(z, v)\}\).

Both \(O_1\) and \(O_2\) has valid assignments in \(D\), but their join filters out the missing answer \(t\).

Resulted sub-queries:

\((z, v) := R_4(z, v)\)

\((x, y, z, w) := R_1(x, y), R_2(y, z), R_3(z, w); z \neq x, w \neq x\)

\((x, y, z) := R_1(x, y), R_2(y, z); z \neq x\)

\((z, w, v) := R_3(z, w), R_4(z, v);\)

\((z, w) := R_3(z, w), R_4(z, v);\)
WhyNot?

• Explain unexpected query results (dirty or missing).
• Two explanation models why-not questions.
• The first model explains a missing tuple t in terms of modifications to the database such that t appears in the query result wrt the modified database.
• The second model explains by identifying the data manipulation operator in the query evaluation plan that is responsible for excluding t from the result.
• In this paper, we propose a new paradigm for explaining a why-not question that is based on automatically generating a refined query whose result includes both the original query’s result as well as the user-specified missing tuple(s). In contrast to the existing explanation models, our approach goes beyond merely identifying the “culprit” query operator responsible for the missing tuple(s) and is useful for applications where it is not appropriate to modify the database to obtain missing tuples.
Algorithm 3: Main Algorithm

Input: A query Q, and an underlying database D
Output: A clean and complete database D w.r.t. Q and D_G
Init: VerifiedResults = ∅, FirstIter = true

while FirstIter || Q(D) \ VerifiedResults ≠ ∅ do
  foreach Tuple t in Q(D) \ VerifiedResults do
    if CrowdVerify(Q(D), t) then
      VerifiedResults ← t
    else
      D ⊕ CrowdRemoveWrongAnswer(Q, D, t)
  endforeach
  foreach Tuple t in CrowdComplete(Q(D)) do
    CrowdAddMissingAnswer(Q, D, t)
  VerifiedResults ← t
  FirstIter = false

return
Fixing one error produces other:

• EXAMPLE. Consider again query $Q_2$, and the missing answer ($Pirlo$) in $Q_2(D)$. QOCO needs to execute the insertion edit of $\{\text{Teams}(IT\ A,\ EU)\ +\}$ on database $D$ to add ($Pirlo$) to $Q_2(D)$. Note that $D$ contains the false tuple $\text{Goals}(Totti,\ 9.6.06)$. Thus, if we add true tuple $\text{Teams}(IT\ A,\ EU)$, the wrong answer ($Totti$) will be added to the output of $Q_2$, as a side effect.
• Authors do not give any bound for these iterative errors
• Their explanation: “Since each edit by oracle brings closer to ground truth, eventually it will converge”
• For example, in the given example, crowd will recommend to remove Totti from goals table.
Multiple imperfect experts

• Aggregate the results from crowd.
• Eg, take average.
• Take vote from crowd like consensus.
• Verify an expert’s answer from crowd.
Implementation

• PHP back-end, javascript front-end, MySQL
• Amazon Mechanical Turk or CrowdFlower does not always have experts.
• Authors recruited their own crowd deemed to be expert enough.
• Tested on two datasets:
  • DBGroup database
  • Soccer database
DBGroup database

• Contains information about publications and conferences attended by researchers in their group.
• 2000 tuples
• 5 wrong answers and 7 missing values were discovered in less than one hour.
• Three users used as crowd.
• Although the db was populated manually, still it had errors
Soccer db

- Contains information about games played in worldcup
- 5000 tuples, scraped using web tools
- Cleaned db using official FIFA data (ground truth)
- Added some noise for testing purposes
- 3 kinds of crowd:
  - i) A simulated perfect oracle (uses ground truth)
  - ii) True soccer experts
  - iii) Imperfect crowd of soccer fans
• Degree of data cleanliness: ratio of number of true tuples in the dataset (i.e., \( |D \cap DG| \)) to the total number of tuples in the dataset plus the tuples missing from the dataset (\( |D| + |DG \setminus D| \)). Added false tuples and removed true tuples to the cleaned Soccer data. We vary the cleanliness of our datasets from 60% to 95%. The default value is 80%.

• Noise skewness: ratio of the number of false tuples in the dataset (i.e., \( |D \setminus DG| \)) to the number of these false tuples plus the number of the missing true tuples (\( |D \setminus DG| + |DG \setminus D| \)). We vary it from 100% where we have only false tuples and no true missing tuples, through 50% where the number of false and missing true tuples are equal, to 0% where we have only missing tuples and no false tuples.

• Degree of result cleanliness: ratio between \( |Q(D) \cap Q(DG)| \) (true tuples) and \( |Q(D)| + |Q(DG) \setminus Q(D)| \).

• Five kind of queries
Baseline Algorithms

• Compared the number of different crowd actions posed by solution to that of competing algorithms running on the same input.

• If crowd says that a result is false, find tuple that can be source of problem.

• For Deletion:
  • 1) Random - randomly picks a tuple, among the tuples in the witnesses of the wrong answer, to verify next.
  • 2) QOCO— a simplified version of our deletion algorithm that greedily picks the most frequent tuple among the tuples in the witnesses of the wrong answer, but does not identify when a unique minimal hitting set exists. Consequently, it continues posing further questions to verify the remaining tuples.
• For insertion:
• Test different splitting mechanism since it’s the core part.
• 1) Naïve - does not split the query.
• 2) Random - randomly splits the given query into two subqueries.
• 3) Min-Cut
• 4) Provenance - uses data provenance and the WhyNot? Algorithm.
Figure 4: Experimental results - real experts ($Q2$ and $Q3$)
Conclusion

• Since no perfect automatic data-cleaning mechanisms available, human interference in inevitable.
• Needs to optimize the questions asked by humans.
• Only need to clean data related to query.