Speeding Up Data Science: From a Data Management Perspective

JIANNAN WANG
SIMON FRASER UNIVERSITY

Database Group @ UBC
July 24, 2017
Our Lab’s Mission

Speeding Up Data Science
# Computer Science vs. Data Science

<table>
<thead>
<tr>
<th>What</th>
<th>When</th>
<th>Who</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science</td>
<td>1950-</td>
<td>Software Engineer</td>
<td>Write software to make computers work</td>
</tr>
</tbody>
</table>

Plan → Design → Develop → Test → Deploy → Maintain

<table>
<thead>
<tr>
<th>What</th>
<th>When</th>
<th>Who</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Science</td>
<td>2010-</td>
<td>Data Scientist</td>
<td>Extract insights from data to answer questions</td>
</tr>
</tbody>
</table>

Collect → Clean → Integrate → Analyze → Visualize → Communicate
Where is the bottleneck?

Data scientists spend 60% of their time on cleaning and organizing data.

(Source: Cloudera)
DeepER’s Key Idea

Leveraging Deep Web To Speed Up Data Cleaning and Data Enrichment
Deep Web

Hidden Database

Invaluable External Resource

- **Big**: Consisting of a substantial number of entities
- **Rich**: Having rich Information about each entity
- **High-quality**: Being trustful and up-to-date
A real-world example

### Customer Location Data

<table>
<thead>
<tr>
<th>User ID</th>
<th>Location</th>
<th>Zip Code</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>U12345</td>
<td>Lotus of Siam</td>
<td>891004</td>
<td>20 visits</td>
</tr>
</tbody>
</table>

### 1. Data Enrichment

<table>
<thead>
<tr>
<th>User ID</th>
<th>Location</th>
<th>Zip Code</th>
<th>Frequency</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>U12345</td>
<td>Lotus of Siam</td>
<td>891004</td>
<td>20 visits</td>
<td>Thai, Wine Bars</td>
</tr>
</tbody>
</table>

### 2. Data Cleaning

<table>
<thead>
<tr>
<th>User ID</th>
<th>Location</th>
<th>Zip Code</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>U12345</td>
<td>Lotus of Siam</td>
<td>89104</td>
<td>20 visits</td>
</tr>
</tbody>
</table>
Entity Resolution

**Local Database (D)**

<table>
<thead>
<tr>
<th>Restaurant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thai Noodle House</td>
</tr>
<tr>
<td>Thai Pot</td>
</tr>
<tr>
<td>Thai House</td>
</tr>
<tr>
<td>BBQ Noodle House</td>
</tr>
</tbody>
</table>

**Hidden Database (H)**

<table>
<thead>
<tr>
<th>Restaurant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thai Noodle House</td>
</tr>
<tr>
<td>Thai Pot</td>
</tr>
<tr>
<td>Thai House</td>
</tr>
<tr>
<td>BBQ Noodle House</td>
</tr>
<tr>
<td>Monta Ramen</td>
</tr>
<tr>
<td>Steak House</td>
</tr>
<tr>
<td>Yard House</td>
</tr>
<tr>
<td>Ramen Bar</td>
</tr>
<tr>
<td>Ramen House</td>
</tr>
</tbody>
</table>
Deep Entity Resolution

Local Database \((D)\)

- Restaurant
  - Thai Noodle House
  - Thai Pot
  - Thai House
  - BBQ Noodle House

Hidden Database \((H)\)

- Restaurant
  - Thai Noodle House
  - Thai Pot
  - Thai House
  - BBQ Noodle House
  - Monta Ramen
  - Steak House
  - Yard House
  - Ramen Bar
  - Ramen House

Keyword Search
1. Conjunctive Query
2. Top-k Constraint
3. Deterministic Query Processing

Local and Hidden DBs
1. \(D\) has no duplicate record
2. \(H\) has no duplicate record
New Challenges

Limited Query Budget
- Yelp API is restricted to 25,000 free requests per day
- Google Maps API only allows 2,500 free requests per day

Top-k Constraint
- Return top-k results based on an unknown ranking function
NaiveCrawl

Enumerate each record in $D$ and then generate a query to cover it

**Limitation**

- Cover one record at a time

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Keyword Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thai Noodle House</td>
<td>$q_1 = \text{“Thai Noodle House”}$</td>
</tr>
<tr>
<td>Thai Pot</td>
<td>$q_2 = \text{“Thai Pot”}$</td>
</tr>
<tr>
<td>Thai House</td>
<td>$q_3 = \text{“Thai House”}$</td>
</tr>
<tr>
<td>BBQ Noodle House</td>
<td>$q_4 = \text{“BBQ Noodle House”}$</td>
</tr>
</tbody>
</table>
FullCrawl

1. Try to crawl the entire hidden database $H_{\text{crawled}}$
2. Perform entity resolution between $D$ and $H_{\text{crawled}}$

Limitation

- Not aware of the existence of a local database
Insight 1. Query Sharing

Keyword Queries

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thai Noodle House</td>
<td>( q_1 = \text{“Thai Noodle House”} )</td>
</tr>
<tr>
<td>Thai Pot</td>
<td>( q_2 = \text{“Thai Pot”} )</td>
</tr>
<tr>
<td>Thai House</td>
<td>( q_3 = \text{“Thai House”} )</td>
</tr>
<tr>
<td>BBQ Noodle House</td>
<td>( q_4 = \text{“BBQ Noodle House”} )</td>
</tr>
<tr>
<td></td>
<td>( q_5 = \text{“Noodle House”} )</td>
</tr>
<tr>
<td></td>
<td>( q_6 = \text{“House”} )</td>
</tr>
<tr>
<td></td>
<td>( q_7 = \text{“Thai”} )</td>
</tr>
</tbody>
</table>

Cover multiple records at a time
Insight 2. Local-database-aware crawling

$q_5 = \text{"Noodle House"}$

$q_6 = \text{"House"}$
SmartCrawl Framework

1. Generate a query pool \( Q \)

2. Select at most \( b \) queries from \( Q \) such that \( |H_{crawled} \cap D| \) is maximized

3. Perform entity resolution between \( H_{crawled} \) and \( D \)
Query Pool Generation

Basic Idea
- Only need to consider the queries in $D$

<table>
<thead>
<tr>
<th>Restaurant</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Thai Noodle House</td>
<td></td>
</tr>
<tr>
<td>Thai Pot</td>
<td></td>
</tr>
<tr>
<td>Thai House</td>
<td></td>
</tr>
<tr>
<td>BBQ Noodle House</td>
<td></td>
</tr>
</tbody>
</table>

$q$ = “Sushi”

Keyword Queries
- $q_1$ = “Thai Noodle House”
- $q_2$ = “Thai Pot”
- $q_3$ = “Thai House”
- $q_4$ = “BBQ Noodle House”
- $q_5$ = “Noodle House”
- $q_6$ = “House”
- $q_7$ = “Thai”
SmartCrawl Framework

1. Generate a query pool $Q$

2. Select at most $b$ queries from $Q$ such that $|H_{crawled} \cap D|$ is maximized

3. Perform entity resolution between $H_{crawled}$ and $D$
Query Selection

NP-Hard Problem
- Can be proved by a reduction from the maximum coverage problem

Greedy Algorithm
- Suffers from a chicken-and-egg problem
Sampling and Estimation

Deep Web Sampling [Zhang et al. SIGMOD 2011]

- $H_S$ is a random sample of $H$
- $\theta$ is the sampling ratio

Two classes of queries

- Solid Query
- Overflowing Query

\[
\text{IF } \left| \frac{q(H_S)}{\theta} \right| \leq k \text{ THEN } \]
\[
q \text{ is a solid query}
\]

\[
\text{ELSE }
\]
\[
q \text{ is an overflowing query}
\]

\[
\text{END}
\]
Solid Query

How to estimate $|q(D) \cap q(H)|$?

Unbiased Estimator: $\frac{|q(D) \cap q(H_s)|}{\theta}$

Key Observation: $|q(D) - q(H)|$ is small

Biased Estimator: $|q(D)|$
Overflowing Query

How to estimate $|q(D) \cap q(H)^k|?$

Basic Idea

How to estimate $\frac{k}{|q(H)|} \times |q(D) \cap q(H)|?$
## A Summary of Estimators

<table>
<thead>
<tr>
<th></th>
<th>Unbiased</th>
<th>Biased (w/ small biases)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Solid</strong></td>
<td>$\frac{</td>
<td>q(D) \cap q(H_s)</td>
</tr>
<tr>
<td><strong>Overflowing</strong></td>
<td>$</td>
<td>q(D) \cap q(H_s)</td>
</tr>
</tbody>
</table>
Other Contributions

1. Theoretical Analysis

2. Efficient Implementations

3. Inadequate Sample Size

4. Fuzzy Matching
Experimental Settings

Simulation
- Hidden Database: DBLP
- Local Database: Database Researchers’ publications

Real-world
- Hidden Database: Yelp
- Local Database: 3000 restaurants in AZ
- Ground-Truth: Manually Labeled
$|D| = 10,000, |H| = 100,000, K = 100, \theta = 0.2\%$

1. SmartCrawl performed very well with a small sampling ratio

2. SmartCrawl outperforms straightforward solutions
1. SmartCrawl outperformed straightforward solutions
2. SmartCrawl was more robust to the fuzzy-matching situation than NaiveCrawl
DeepER Conclusion

We are the first to study the DeepER problem

SmartCrawl outperforms NaiveCrawl and FullCrawl by a factor of $2 - 7\times$

SmartCrawl is more robust to the fuzzy-matching situation than NaiveCrawl
Today’s Talk

DeepER

Collect → Clean → Integrate → Analyze → Visualize → Communicate

AQP++
Interactive Analytics

Tableau

Jupyter

Power BI

Databricks

Apache Zeppelin
Two Separate Ideas

Approximate Query Processing (AQP)
- Trade answer quality for interactive response time

Aggregate Precomputation (AggPre)
- Trade preprocessing cost for interactive response time
AQP++: Connecting AQP with AggPre

![Diagram showing AQP, AQP++, AggPre in a 3D space with Response Time, Preprocessing Cost, and Query Error axes.](image-url)
## Experimental Result

**TPCD-Skew (10GB, z = 2, 0.3% sample)**

<table>
<thead>
<tr>
<th>Preprocessing</th>
<th>Preprocessing Cost</th>
<th>Response Time</th>
<th>Answer Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Space</td>
<td>Time</td>
<td>Avg Err.</td>
</tr>
<tr>
<td>AQP</td>
<td>30.72 MB</td>
<td>1.71 min</td>
<td>3.28%</td>
</tr>
<tr>
<td>AggPre</td>
<td>10.91 TB</td>
<td>&gt; 1 day</td>
<td>0.00%</td>
</tr>
<tr>
<td>AQP++</td>
<td>30.74 MB</td>
<td>2.97 min</td>
<td>0.41%</td>
</tr>
</tbody>
</table>
## On-going Projects

<table>
<thead>
<tr>
<th>Students</th>
<th>Stages</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pei &amp; Yongjun</td>
<td>Clean &amp; Integrate</td>
<td>DeepER: Deep Entity Resolution</td>
</tr>
<tr>
<td>Jinglin Peng</td>
<td>Analyze &amp; Visualize</td>
<td>AQP++: Connecting AQP with AggPre</td>
</tr>
<tr>
<td>Mathew &amp; Mohamad</td>
<td>Clean &amp; Analyze</td>
<td>Data Cleaning Advisor for ML</td>
</tr>
<tr>
<td>Changbo &amp; Ruochen</td>
<td>Collect &amp; Clean</td>
<td>Live Video Highlight Detection using Crowdsourced User Comments</td>
</tr>
<tr>
<td>Nathan Yan</td>
<td>Clean</td>
<td>Data Cleaning with Statistical Constraints</td>
</tr>
<tr>
<td>Young Woo</td>
<td>Analyze</td>
<td>ML Explanation and Debugging</td>
</tr>
</tbody>
</table>
Two Take-away Messages

Data scientists waste a lot of time on data processing

Collect → Clean → Integrate → Analyze → Visualize → Communicate

Database researchers play a central role to speed up data science