SeeDB: Supporting Fast Visual Analytics

Manasi Vartak
MIT Database Group | MIT CSAIL

Joint work w/Sam Madden, Aditya Parameswaran, Neoklis Polyzotis
Motivating Example

• Visualization: First step in analytics

• Census data\(^1\): age, education, marital-status, sex, race, income, hours-worked etc.
  – \( A = \# \) attributes in table

• Task: Socioeconomics of adults who have never-been-married

\(^1\)https://archive.ics.uci.edu/ml/datasets/Census+Income
Visualizing Census Data

Histograms: $O(A)$
Multi-attribute: $O(2^A)$
Visualizing Census Data

Pairwise Scatterplot: $O(A^2)$
Visualizing Census Data

Aggregate View: $O(A^2)$
Visualizing Census Data

Comparative visualizations: $O(A^2)$
Too Many Possible Visualizations
SeeDB: a visualization recommender
Related Work

• Visualization tools:
  – ManyEyes, Tableau/Polaris, Fusion Tables, Spotfire
  – Tableau and Spotfire recommendations (Aesthetics)

• Some Automation: VizDeck, Profiler, Voyager

• Scagnostics: interesting-ness of scatterplots
  – Graph-theoretic metrics
Recommending Visualizations

I. How to find relevant visualizations?
   – Interesting-ness or utility metric

I. How to make recommendations efficiently?
   – Scale to large number of rows
   – Curse of dimensionality
   – Interactive time scales
I. How to find relevant visualizations?
Visualization Utility

• Utility depends on data (distribution), query, metadata, context, user preferences, aesthetics

• $U = f(\text{data, query, metadata})$

• Why?
  – Captures significant part of user interest
  – Independent of “external” data
  – Necessary for cold starts
SeeDB Visualizations

Bar charts/Accumulate Visualizations

- Aggregates essential for large datasets
- Large fraction of common visualizations
SeeDB Visualizations

- $V_i = (d : \text{dimension}, m : \text{measure}, f : \text{aggregate})$
  - X-axis
  - Y-axis

- AGGREGATE + GROUP BY queries

```sql
SELECT d, f(m) FROM table GROUP BY d WHERE selection_predicate
```
The greatest value of a picture is when it forces us to notice what we never expected to see.

Tukey, Exploratory Data Analysis 1977

What is unexpected (different from expected trends) is interesting.
Deviation-based Utility Metric

\[ Q : \text{SELECT * from census WHERE marital-status = 'Never-married'} \]

Table: \( D \), Data selected by \( Q \): \( Q_D \)

\{d\} : race, work-type, sex etc.
\{m\} : capital-gain, capital-loss, hours-per-week
\{f\} : COUNT, SUM, AVG
Computing Expected Trend

$V_i$: Race vs. AVG(capital-gain)

SELECT race, AVG(capital-gain) FROM census GROUP BY race

$P[V_i(D)]$

Expected Distribution
Computing Actual Trend

$V_i$: Race vs. AVG(capital-gain)

```sql
SELECT race, AVG(capital-gain) FROM census
GROUP BY race WHERE marital-status='Never-married'
```

$P[V_i(D_Q)]$

Actual Distribution
Computing Utility

\[ V_i : P[V_i(D)] \text{ (expected)}, \quad P[V_i(D_Q)] \text{ (actual)} \]

\[ U(V_i) = \Delta(P[V_i(Q_D)], P[V_i(D)]) \]

\[ \Delta = \text{EMD, L}_2 \text{ etc.} \]
Low Utility Visualization

race vs. AVG(capital-gain)

<table>
<thead>
<tr>
<th>race</th>
<th>Actual</th>
<th>Expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amer-Indian-Eskimo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asian-Pac-Islander</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>Black</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>White</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>
High Utility Visualization

race vs. AVG(capital-gain)

- Amer-Indian-Eskimo
- Asian-Pac-Islander
- Black
- Other
- White

AVG(capital-gain)

Actual
Expected
II. How to make recommendations efficiently?
SeeDB Architecture

• Middleware on top of DB

• Data processing delegated to DBMS

• Any SQL-compliant DBMS
Challenges

\begin{align*}
D &= \text{# dimensions, } M = \text{# measures,} \\
F &= \text{# aggregate functions}
\end{align*}

- $D \times M \times F$ potential visualizations
- $2 \times D \times M \times F$ queries to the DBMS
- Each query (potentially) scans full dataset
- Computation wasted on low-utility views
How do we make recommendations efficiently?

Goals:

• Interactive latencies
• No wasted resources on low-utility views

Strategies:

1. Run-time pruning framework
2. Systems-level optimizations
Run-time Pruning Framework

- Identify views with low-utility early, weed out

- Running estimates of utility based on samples

- Techniques:
  - Confidence Interval-based Pruning
  - Multi-Armed Bandit Pruning*
Multi-Armed Bandit

Which machines to play and in what order to maximize reward?
Multi-Armed Bandit

Slot Machines: S₁, S₂, S₃, S₄, ..., Sₙ

Reward Distributions: R₁, R₂, R₃, R₄, ..., Rₙ

Estimates: r₁, r₂, r₃, r₄, ..., rₙ

Use estimates to guide choice of machine
Multi-Armed Bandit Pruning

- Rank views by utility
- Find maximum utility views
- Compute various \( \Delta_i \)
- \( \text{Discard } V_n \) or accept \( V_1 \)
Systems-level optimizations

• Each visualization = 2 SQL queries

• Latency > 100s

• Minimize number of queries and scans
Systems-level optimizations

• Combine aggregate queries on $Q_D$ and $D$

• Combine multiple aggregates
  
  $(d_1, m_1, f_1), (d_1, m_2, f_1) \rightarrow (d_1, [m_1, m_2], f_1)$

• Combine multiple group-bys$^*$
  
  $(d_1, m_1, f_1), (d_2, m_1, f_1) \rightarrow ([d_1, d_2], m_1, f_1)$

• Parallel Query Execution
Combining Multiple Group-bys

• Too few group-bys leads to many table scans

• Too many group-bys hurt performance
  – # groups = \( \Pi (# \text{ distinct values per attributes}) \)

• Optimal group-by combination \( \approx \) bin-packing
  – Bin volume = \( \log S \) (max number of groups)
  – Volume of items (attributes) = \( \log (|a_i|) \)
  – Minimize # bins s.t.
    \[ \Sigma_i \log (|a_i|) \leq \log S \]
Interleaving Optimizations

Phase 1
- Query Execution
- Pruning

Phase 2
- Query Execution
- Pruning

Phase 3
- Query Execution
Evaluation
Evaluation

• Performance Study
  – Latency, accuracy
  – Variety of synthetic and real datasets

• User Study (ongoing)
  – Controlled Study
  – Trends, Interactions, Surveys
Pruning Optimizations

Utility Distance for BANK

Utility Distance = \[ \Delta \left( \text{Utility of real top-}k, \text{Utility of SeeDB top-}k \right) \]
Pruning Optimizations

Pruning can reduce latency by 50 - 90% w/o significant hit in accuracy
Systems-level Optimizations

Combination of systems optimizations reduce latency 25X to ~ 10s
Putting it together

SeeDB returns results in < 4 s for small and medium-sized data
User Study
User Quotes

“I liked the recommendations because it allowed for a quick starting point”

“They showed a compelling abstract of the data. It made the data comprehensible”

“It’s a great tool for proposing an initial set of queries for a new dataset”

“Recommendations helped in finding some interesting aspects of the data I wouldn’t have checked myself”

“I was more likely to stick to suggestions than come up with my own thoughtful or new ideas”
User study stats: I

• Prefer tool with recommendations: 100%
  – High bar on quality

• Recommendations sped up analysis: 85%

• Recommendations Rating: 75% (>= helpful)
  – Deviation-based metric performs well
User study stats: II

• More charts explored with recommendations
  – No_rec (11) vs. Rec (14) [p-value: 0.1]

• Number of bookmarks
  – No_rec (3.5) vs. Rec (3.6) [p-value: 0.5]
    – Context and user preferences

• Expert users expect sophisticated statistics
SeeDB Summary

• Visualization recommender for analytics
• Deviation-based utility to capture relevance
• Run-time pruning can provide a 4X speedup and systems-level optimizations 25X
• Users prefer a tool with recommendations
• Ongoing work: incorporate other relevance metrics (other distance metrics, context etc.)
Thanks and Questions!

mvartak@mit.edu  |  @DataCereal