GOOGLE WEBTABLES

Wout Bittremieux
Sibren Polders
Introduction

- Web is a corpus of unstructured documents
- Some structure
  - Hierarchical URLs
  - Hyperlink graph
- Web pages contain
  - Text
  - Tabular data
Introduction

- Table tag
- Not explicitly declared schema consisting of labeled columns
- Navigation bars
Motivation

• Enable analysis and integration of data on the web
• User demand for structured data
  • Google: Daily for 30M queries users clicked on results containing tables

• Difficulties with relation ranking
  = Sort relations by relevance in response to a query
  • Mixture of structural and content elements
  • Ranking methods (PageRank) do not give optimum results
Data

• WebTables system considers HTML tables that are already surfaced and crawlable
• Deep Web refers to the content that is made available through filling HTML forms
• Corpus
  • 14.1 billion raw HTML tables
  • 154 million distinct relational databases
    • Around 1.1% of all raw HTML tables
  • 60% of data from non-deep-web sources
  • 40% of data from parameterized URLs
Extracting Relations

• Most HTML tables are used for page layouts
• Filter relational from non-relational tables
  • Scale of ‘relational quality’ by human judges
  • Statistically trained classifier
• High recall and low precision
  → Lose relatively few true relations

Figure 2: The WebTables relation extraction pipeline. About 1.1% of the raw HTML tables are true relations.
# Data Model

<table>
<thead>
<tr>
<th>$\mathcal{R}$</th>
<th>Corpus of databases where each database is a single relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>Is a relation; $R \in \mathcal{R}$</td>
</tr>
<tr>
<td>$R_u$ and $R_i$ uniquely define $R$</td>
<td></td>
</tr>
<tr>
<td>$R_u$</td>
<td>URL from the page from which $R$ was extracted</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Offset from $R$ within the page</td>
</tr>
<tr>
<td>$R_S$</td>
<td>Schema of $R$, which is an ordered list of attribute labels</td>
</tr>
<tr>
<td>$R_T$</td>
<td>A list of tuples</td>
</tr>
<tr>
<td>$t$</td>
<td>A tuple, which is a list of data strings</td>
</tr>
<tr>
<td>$\mathcal{A}$</td>
<td>Attribute Correlation Statistics Database (ACSDb)</td>
</tr>
</tbody>
</table>
Attribute Correlation Statistics Database (ACSDb)

- For each unique schema $R_S$, the ACSDb contains a frequency count
- $A = \{(R_{S1}, C_1), (R_{S2}, C_2), (R_{S3}, C_3), \ldots\}$
- Two schemas are identical if they contain the same attributes
- If a schema appears multiple times under the same domain, it is only counted once
Attribute Correlation Statistics Database (ACSDb)

• The ACSDb contains:
  • 5.4M unique attribute names
  • 2.6M unique schemas

• A small number of schemas appear frequently
• Most schemas are rare

Figure 3: Distribution of frequency-ordered unique schemas in the ACSDb, with rank-order on the x-axis, and schema frequency on the y-axis. Both rank and frequency axes have a log scale.
Attribute Correlation Statistics Database (ACSDb)

- The ACSDb allows to:
  - Compute the probability of seeing various attributes in a schema
  - Detect relationships between attribute names

- F.e.: \( p(\text{address}|\text{name}) = \) all the schemas in which ‘address’ appears along with ‘name’
Relation Search

• The WebTables search engine allows users to rank relations by relevance

• Structured nature of the data allows additional services:
  • Query-appropriate visualizations
  • Traditional structured operations (selection, projection) over search results
### Largest cities in the world by population (1 to 125)

<table>
<thead>
<tr>
<th>City / Urban area</th>
<th>Country</th>
<th>Population</th>
<th>Land area (in sqKm)</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokyo, Yokohama</td>
<td>Japan</td>
<td>33,200,000</td>
<td>5.68 (5688.3)</td>
<td>5,868</td>
</tr>
<tr>
<td>New York Metro</td>
<td>USA</td>
<td>17,800,000</td>
<td>8,680</td>
<td>2,100</td>
</tr>
<tr>
<td>Sao Paulo</td>
<td>Brazil</td>
<td>17,700,000</td>
<td>1,77 (1770.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Seoul</td>
<td>South Korea</td>
<td>17,000,000</td>
<td>624</td>
<td>2,651</td>
</tr>
<tr>
<td>Mexico City</td>
<td>Mexico</td>
<td>17,000,000</td>
<td>1,74 (1740.0)</td>
<td>2,637</td>
</tr>
<tr>
<td>Osaka</td>
<td>Japan</td>
<td>16,420,000</td>
<td>2,41 (2410.0)</td>
<td>699</td>
</tr>
<tr>
<td>Manila</td>
<td>Philippines</td>
<td>14,000,000</td>
<td>1,49 (1490.0)</td>
<td>940</td>
</tr>
<tr>
<td>Delhi</td>
<td>India</td>
<td>12,000,000</td>
<td>1,20 (1200.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Jakarta</td>
<td>Indonesia</td>
<td>11,500,000</td>
<td>1,15 (1150.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Lagos</td>
<td>Nigeria</td>
<td>11,000,000</td>
<td>1,10 (1100.0)</td>
<td>738</td>
</tr>
<tr>
<td>Kolkata</td>
<td>India</td>
<td>10,700,000</td>
<td>1,07 (1070.0)</td>
<td>930</td>
</tr>
<tr>
<td>Cairo</td>
<td>Egypt</td>
<td>10,200,000</td>
<td>1,02 (1020.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>USA</td>
<td>10,000,000</td>
<td>1,00 (1000.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Buenos Aires</td>
<td>Argentina</td>
<td>10,000,000</td>
<td>1,00 (1000.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>Brazil</td>
<td>10,000,000</td>
<td>1,00 (1000.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Moscow</td>
<td>Russia</td>
<td>10,000,000</td>
<td>1,00 (1000.0)</td>
<td>2,160</td>
</tr>
<tr>
<td>Shanghi</td>
<td>China</td>
<td>9,000,000</td>
<td>0.9 (900.0)</td>
<td>748</td>
</tr>
<tr>
<td>Karachi</td>
<td>Pakistan</td>
<td>7,500,000</td>
<td>1 (750.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>Paris</td>
<td>France</td>
<td>7,000,000</td>
<td>0.7 (700.0)</td>
<td>2,723</td>
</tr>
<tr>
<td>Istanbul</td>
<td>Turkey</td>
<td>7,000,000</td>
<td>0.7 (700.0)</td>
<td>1,100</td>
</tr>
<tr>
<td>Osaka</td>
<td>Japan</td>
<td>6,000,000</td>
<td>0.6 (600.0)</td>
<td>2,975</td>
</tr>
<tr>
<td>Beijing</td>
<td>China</td>
<td>6,000,000</td>
<td>0.6 (600.0)</td>
<td>748</td>
</tr>
<tr>
<td>Cairo</td>
<td>Egypt</td>
<td>6,000,000</td>
<td>0.6 (600.0)</td>
<td>1,000</td>
</tr>
<tr>
<td>London</td>
<td>UK</td>
<td>6,000,000</td>
<td>0.6 (600.0)</td>
<td>1,000</td>
</tr>
</tbody>
</table>

### City Populations

<table>
<thead>
<tr>
<th>Region</th>
<th>People per Hectare</th>
<th>Margin of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>City of Belligacy</td>
<td>100</td>
<td>5%</td>
</tr>
<tr>
<td>City of London</td>
<td>100</td>
<td>10%</td>
</tr>
<tr>
<td>Cities of London (South and North)</td>
<td>100-115</td>
<td>10%</td>
</tr>
<tr>
<td>100-115</td>
<td>120</td>
<td>10%</td>
</tr>
<tr>
<td>120-135</td>
<td>220</td>
<td>20-25%</td>
</tr>
<tr>
<td>220-25%</td>
<td>40</td>
<td>25%</td>
</tr>
<tr>
<td>40-50%</td>
<td>60</td>
<td>25%</td>
</tr>
</tbody>
</table>

---

*Note: The data includes rank, city/urban area, country, population, land area, and density data.*

---

*Further details on scatter plot, line graph, bar graph, and highlights available.*
Ranking

- Keyword ranking for databases is a novel problem
- Challenges:
  - Relations do not exist in a domain-specific schema graph
  - Attribute labels are extremely important
  - Even a high-quality page may contain tables of varying quality
- Special features:
  - Schema elements provide a good summary
  - Tuples may have a key-like element that summarizes the row
Ranking

• Ranking functions:
  • naïveRank
  • filterRank
  • featureRank
  • schemaRank
 naïveRank

• Simply uses the top-\(k\) search engine result pages to generate relations
• If there are fewer than \(k\) relations in these pages, naïveRank will not go any deeper in the result list
• Roughly simulates what a modern search engine user must do when searching for structured data

```
1: Function naiveRank(q, k):
2: let \(\mathcal{U}\) = urls from web search for query \(q\)
3: for \(i = 0\) to \(k\) do
4:    emit getRelations(\(\mathcal{U}[i]\))
5: end for
```
filterRank

• Similar to naïveRank
• It will march down the search engine results until it finds $k$ relations

```
1: Function filterRank(q, k):
2:   let $\mathcal{U}$ = ranked urls from web search for query $q$
3:   let numEmitted = 0
4:   for all $u \in \mathcal{U}$ do
5:     for all $r \in \text{getRelations}(u)$ do
6:       if numEmitted $\geq k$ then
7:         return
8:       end if
9:     emit $r$; numEmitted++
10:   end for
11: end for
```
featureRank

- Does not rely on an existing search engine
- Uses relation-specific features to score each extracted relation in the corpus
- Sorts results by score
- Different feature scores were combined using a linear regression estimator
  - Trained on more than a thousand \((q, relation)\) pairs
  - Each pair was scored by two human judges
# rows
# cols
has-header?
# of NULLs in table
document-search rank of source page
# hits on header
# hits on leftmost column
# hits on second-to-leftmost column
# hits on table body

1: **Function** featureRank\((q, k)\):
2: let \( \mathcal{R} = \) set of all relations extracted from corpus
3: let \( \text{score}(r \in \mathcal{R}) = \) combination of per-relation features in Table 2
4: sort \( r \in \mathcal{R} \) by \( \text{score}(r) \)
5: for \( i = 0 \) to \( k \) do
6: \hspace{1em} emit \( \mathcal{R}[i] \)
7: end for
schemaRank

- Same as featureRank
- Additionally uses an **ACSDb**-based schema coherency score
  - A coherent schema is one where the attributes are tightly related to one another
- Pointwise Mutual Information (PMI)
  - Gives a sense of how strongly two items are related
- The coherency score for a schema is the average of all possible attribute-pairwise PMI scores
schemaRank

1: **Function** cohere($R$):
2:  $totalPMI = 0$
3:  **for all** $a \in attrs(R), b \in attrs(R), a \neq b$ **do**
4:      $totalPMI = PMI(a, b)$
5:  **end for**
6:  return $totalPMI / (|R| \times (|R| - 1))$

1: **Function** pmi($a, b$):
2:  return $\log\left(\frac{p(a,b)}{p(a) \times p(b)}\right)$
Indexing

- Traditional search engines use an inverted index
  \(\iff\) Inverted index cannot retrieve relational features

- Inverted index
  - Term \(\rightarrow\) sorted posting list of (docid, offset) pairs

- WebTables data exists in two dimensions
  - Term \(\rightarrow\) (docid, offset-X, offset-Y)
ACSDb Applications

- Schema Auto-Complete
- Attribute Synonym-Finding
- Join Graph Traversal
Schema Auto-Complete

- Designed to assist novice database designers when creating a relational schema
- Focus on schemas consisting of a single relation
- The user enters one or more domain-specific attributes
- The schema auto-completer guesses the rest of the attribute labels
# Schema Auto-Complete

1: Function SchemaSuggest(I, t):
2: \( S = I \)
3: while \( p(S - I|I) > t \) do
4: \( a = \max_{a \in A-S} p(a, S - I|I) \)
5: \( S = S \cup a \)
6: return \( S \)
7: end while

<table>
<thead>
<tr>
<th>Input attribute</th>
<th>Auto-completer output</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name, size, last-modified, type</td>
</tr>
<tr>
<td>instructor</td>
<td>instructor, time, title, days, room, course</td>
</tr>
<tr>
<td>elected</td>
<td>elected, party, district, incumbent, status, opponent, description</td>
</tr>
<tr>
<td>ab</td>
<td>ab, h, r, bb, so, rbi, avg, lob, hr, pos, batters</td>
</tr>
<tr>
<td>stock-symbol</td>
<td>stock-symbol, securities, pct-of-portfolio, num-of-shares, mkt-value-of-securities, ratings</td>
</tr>
<tr>
<td>company</td>
<td>company, location, date, job-summary, miles</td>
</tr>
<tr>
<td>director</td>
<td>director, title, year, country</td>
</tr>
<tr>
<td>album</td>
<td>album, artist, title, file, size, length, date/time, year, comment</td>
</tr>
<tr>
<td>sqft</td>
<td>sqft, price, baths, beds, year, type, lot-sqft, days-on-market, stories</td>
</tr>
<tr>
<td>goals</td>
<td>goals, assists, points, player, team, gp</td>
</tr>
</tbody>
</table>
Attribute Synonym-Finding

- Automatically finds synonyms between arbitrary attribute strings
- Takes a set of context attributes as input
- Assumptions
  - Synonymous attributes will never appear together in the same schema; $p(a,b) = 0$
  - Two synonyms will appear in similar contexts

\[
syn(a, b) = \frac{p(a)p(b)}{ε + \sum_{z \in A} (p(z|a, C) - p(z|b, C))^2}
\]
Attribute Synonym-Finding

```
1: Function SynFind(C, t):
2:     R = []
3:     A = all attributes that appear in ACSDb with C
4:     for a ∈ A, b ∈ B, s.t. a ≠ b do
5:         if (a, b) ∉ ACSDb then
6:             // Score candidate pair with syn function
7:             if syn(a, b) > t then
8:                 R.append(a, b)
9:         end if
10:     end if
11: end for
12: sort R in descending syn order
13: return R
```

<table>
<thead>
<tr>
<th>Input context</th>
<th>Synonym-finder outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>e-mail</td>
</tr>
<tr>
<td>instructor</td>
<td>course-title</td>
</tr>
<tr>
<td>elected</td>
<td>candidate</td>
</tr>
<tr>
<td>ab</td>
<td>k</td>
</tr>
<tr>
<td>stock-symbol</td>
<td>company</td>
</tr>
<tr>
<td>company</td>
<td>phone</td>
</tr>
<tr>
<td>director</td>
<td>film</td>
</tr>
<tr>
<td>album</td>
<td>song</td>
</tr>
<tr>
<td>sqft</td>
<td>bath</td>
</tr>
<tr>
<td>goals</td>
<td>name</td>
</tr>
</tbody>
</table>
Join Graph Traversal

- Provide a useful way of navigating the huge graph of 2.6M unique schemas
- Basic join graph
  - Contains a node for each unique schema
  - An undirected join link between any two schemas that share a label
- Cluster together similar schema neighbors to reduce the join graph clutter
- Join neighbor similarity: measure whether a shared attribute \( D \) plays a similar role in its schemas \( X \) and \( Y \)

\[
\text{neighborSim}(X, Y, D) = \frac{1}{|X||Y|} \sum_{a \in X, b \in Y} \log \left( \frac{p(a, b|D)}{p(a|D)p(b|D)} \right)
\]
Conclusion

• WebTables is the first large-scale attempt to extract relational information embedded in HTML tables

• **ACSDb** uses:
  • Relation ranking
  • Schema auto-complete
  • Attribute synonym finding

• **Future work**
  • Incorporate source-page quality signal (PageRank)
  • Include relational data derived from non-HTML table sources
Questions?