WebTables: Exploring the Power of Tables on the Web

Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, Yang Zhang

Presented by: Ganesh Viswanathan
September 29th, 2011
CIS 6930 – Data Science: Large-scale Advanced Data Analysis
University of Florida, Gainesville
Outline

• Introduction to the work
• Motivation
• WebTables
  • Relation Recovery
    – Relational Filtering
    – Metadata recovery
  • *Attribute Correlation Statistics* Database (*ACSDb*)
• Relation Search
  • Ranking
  • Indexing
• ACDBb Applications
  • Schema auto-complete
  • Attribute synonym-finding
  • Join graph traversal
• Experimental Results
• Discussion
Introduction

• Age of big data:
  – Availability of massive amounts of data is driving many technical advances on the web and off
  – The web is traditionally modeled as a corpus of unstructured data
  – But web documents contain large amounts of relational data

• Questions
  – What are effective techniques for searching for structured data at search-engine scales?
  – What additional power can be derived by analyzing such a huge corpus?
Motivation

• Large demand for queries for results contained in this corpus

• Around 30 million queries from Google’s 1-day log

• Relational ranking is difficult since data is not purely unstructured, rather a mixture of “structure” and “content”

• Lacks incoming hyperlink anchor text used in traditional IR

• Ranking methods like PageRank do not give optimum results.
Example: Relational Data on the Web
WebTables

• The goals are to gather a corpus of high quality relational data on the web and make it better searchable.

• Describe a ranking method for Tables on the web based on their relational data combining traditional index and query-independent corpus-wide coherency score.

• Define an attribute correlation statistics database (ACSDb) containing statistics about corpus schemas.

• Using these statistics to create novel tools for database designers addressed later on.
WebTables

- Using Google’s English language web indexes, 14.1 billion raw HTML pages extracted

- Tables used for page layout and non-relational purposes filtered out

- Resulted in 154M distinct relational databases, i.e., around 1.1% of all raw HTML tables.
Relational Search Interface

### City Mayors: Largest cities in the world by population (1 to 125)

<table>
<thead>
<tr>
<th>Rank</th>
<th>City / Urban area</th>
<th>Country</th>
<th>Population (1)</th>
<th>Land area (in sqkm)</th>
<th>Den</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tokyo / Chiyoda</td>
<td>Japan</td>
<td>32,280,000</td>
<td>2,522</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>New York Metro</td>
<td>USA</td>
<td>10,120,000</td>
<td>123</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Sao Paulo</td>
<td>Brazil</td>
<td>10,650,000</td>
<td>235</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>Moscow</td>
<td>Russia</td>
<td>12,600,000</td>
<td>354</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>Mexico City</td>
<td>Mexico</td>
<td>2,540,000</td>
<td>354</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Osaka / Kita</td>
<td>Japan</td>
<td>2,540,000</td>
<td>354</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>Mumbai</td>
<td>India</td>
<td>2,540,000</td>
<td>354</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>Delhi</td>
<td>India</td>
<td>2,540,000</td>
<td>354</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>Jakarta</td>
<td>Indonesia</td>
<td>2,540,000</td>
<td>354</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>Lagos</td>
<td>Nigeria</td>
<td>2,540,000</td>
<td>354</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>Kolkata</td>
<td>India</td>
<td>2,540,000</td>
<td>354</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>Cairo</td>
<td>Egypt</td>
<td>2,540,000</td>
<td>354</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>Los Angeles</td>
<td>USA</td>
<td>16,650,000</td>
<td>48</td>
<td>24</td>
</tr>
<tr>
<td>14</td>
<td>Buenos Aires</td>
<td>Argentina</td>
<td>1,590,000</td>
<td>88</td>
<td>25</td>
</tr>
<tr>
<td>15</td>
<td>Rio de Janeiro</td>
<td>Brazil</td>
<td>1,590,000</td>
<td>88</td>
<td>26</td>
</tr>
<tr>
<td>16</td>
<td>Moscow</td>
<td>Russia</td>
<td>1,590,000</td>
<td>88</td>
<td>27</td>
</tr>
<tr>
<td>17</td>
<td>Shanghai</td>
<td>China</td>
<td>1,590,000</td>
<td>88</td>
<td>28</td>
</tr>
<tr>
<td>18</td>
<td>Karachi</td>
<td>Pakistan</td>
<td>1,590,000</td>
<td>88</td>
<td>29</td>
</tr>
<tr>
<td>19</td>
<td>Paris</td>
<td>France</td>
<td>1,590,000</td>
<td>88</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Moscow</td>
<td>Russia</td>
<td>1,590,000</td>
<td>88</td>
<td>31</td>
</tr>
<tr>
<td>21</td>
<td>Beijing</td>
<td>China</td>
<td>1,590,000</td>
<td>88</td>
<td>32</td>
</tr>
<tr>
<td>22</td>
<td>Shanghai</td>
<td>China</td>
<td>1,590,000</td>
<td>88</td>
<td>33</td>
</tr>
<tr>
<td>23</td>
<td>Osaka / Kita</td>
<td>Japan</td>
<td>1,590,000</td>
<td>88</td>
<td>34</td>
</tr>
<tr>
<td>24</td>
<td>Beijing</td>
<td>China</td>
<td>1,590,000</td>
<td>88</td>
<td>35</td>
</tr>
<tr>
<td>25</td>
<td>Moscow</td>
<td>Russia</td>
<td>1,590,000</td>
<td>88</td>
<td>36</td>
</tr>
</tbody>
</table>

### ESTIMATING 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 Antioch</td>
<td>450</td>
<td>10%</td>
</tr>
<tr>
<td>City of Athens</td>
<td>200</td>
<td>15%</td>
</tr>
<tr>
<td>City of Athens</td>
<td>200</td>
<td>15%</td>
</tr>
<tr>
<td>City of Europe (World)</td>
<td>150</td>
<td>10%</td>
</tr>
<tr>
<td>City of Europe (World)</td>
<td>150</td>
<td>10%</td>
</tr>
<tr>
<td>City of Europe (World)</td>
<td>150</td>
<td>10%</td>
</tr>
</tbody>
</table>
Applications

- We can leverage the ACSDb to offer solutions to following tasks:
  - Schema auto-complete tool to help database designers choose a schema
  - Attribute synonym finding tool that computes pairs of schema attributes used synonymously
  - Join graph traversal using common attributes and clustering.
The Deep Web

• Deep web refers to tables behind HTML forms

• Can be detected by detecting if the URL is parameterized

• Some deep web data is web crawlable

• Vast majority still requires sophisticated systems that can input parameters in a semantically meaningful manner

• Corpus contains around 40% from deep web sources while 60% from non-deep-web sources.
The WebTables System

- WebTables system automatically extracts databases from web crawl
- An extracted relation is a single table plus *typed and labeled columns*
- But there is no straightforward method to determine if detected table contains relational data.

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

- Two stages for extraction system:
  - Relational filtering (for “good” relations)
  - Metadata detection (in top row of table)

- Apply an HTML parser to a page crawl to obtain 14.1B instances of the <table> tag.

- Disregard tables used for layout, forms, calendars, etc., as obviously non-relational
Relational Filtering

• Next step of deciding if the tables contain relational data requires human judgment

• Training data was given to 2 independent judges

• Rated the table on its “relational quality” from 1-5. Table with average score of 4 or above was deemed relational

• Based on this sample, estimate the amount of “other non-relational” and “relational”
Relational Filtering

- The next step is a machine-learning classification problem.
- To the training data provided by the human judges, pair a set of automatically-extracted features of the table.
- This forms a supervised training set for the statistical learner.

Since the true set of relations will be much larger in size, a downstream search system would be needed after the filter has done its work.

So as not to loose true relational tables in the filtering, compromise on precision and get relatively higher recall.
## Relational Filtering Statistics

### From the raw crawl:

<table>
<thead>
<tr>
<th>Table type</th>
<th>% total</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Tiny” tables</td>
<td>88.06</td>
<td>12.34B</td>
</tr>
<tr>
<td>HTML forms</td>
<td>1.34</td>
<td>187.37M</td>
</tr>
<tr>
<td>Calendars</td>
<td>0.04</td>
<td>5.50M</td>
</tr>
<tr>
<td><strong>Filtered Non-relational, total</strong></td>
<td><strong>89.44</strong></td>
<td><strong>12.53B</strong></td>
</tr>
<tr>
<td>Other non-rel (est.)</td>
<td>9.46</td>
<td>1.33B</td>
</tr>
<tr>
<td>Relational (est.)</td>
<td>1.10</td>
<td>154.15M</td>
</tr>
</tbody>
</table>
Relational Filtering Statistics

- Extraction is tuned for lower precision and higher recall
  - Relational filtering, recall 81%, precision 41%

<table>
<thead>
<tr>
<th>#cols</th>
<th>% tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2-9</td>
<td>93.18</td>
</tr>
<tr>
<td>10-19</td>
<td>6.17</td>
</tr>
<tr>
<td>20-29</td>
<td>0.46</td>
</tr>
<tr>
<td>30+</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#rows</th>
<th>% tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2-9</td>
<td>64.07</td>
</tr>
<tr>
<td>10-19</td>
<td>15.83</td>
</tr>
<tr>
<td>20-29</td>
<td>7.61</td>
</tr>
<tr>
<td>30+</td>
<td>12.49</td>
</tr>
</tbody>
</table>
Metadata Detection

- **Per-attribute labels** are useful in improving rank quality for keyword searches on tables, for data visualization and in the construction of the ACSDb.

- To detect the header row in a table, apply a **Detect** classifier that uses the following features to make a decision:

  # rows
  # cols
  % cols w/lower-case in \textit{row}_1
  % cols w/punctuation in \textit{row}_1
  % cols w/non-string data in \textit{row}_1
  % cols w/non-string data in \textit{body}
  % cols w/\(\left|\text{len}(\text{row}_1) - \mu\right| > 2\sigma\)
  % cols w/\(\sigma \leq \left|\text{len}(\text{row}_1) - \mu\right| \leq 2\sigma\)
  % cols w/\(\sigma > \left|\text{len}(\text{row}_1) - \mu\right|\)
The classifier is trained on thousand tables marked by two human judges paired with the features listed previously.

Two heavily weighted features for header detection are 
# of columns and % of non-string data in the first row.

In the case where no header exists, try to synthesize column names from reference databases using the tuples.

The results of such a Reference-Match Algorithm are poor.
The ACSDb (covered in further slides) contains the counts of occurrence of attributes with other attributes.

We can improve the performance of Detect classifier if we use the probability values of occurrence of attributes within a given schema.

<table>
<thead>
<tr>
<th>Detector</th>
<th>header?</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detect</td>
<td>has-header</td>
<td>0.79</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>no-header</td>
<td>0.65</td>
<td>0.57</td>
</tr>
<tr>
<td>Detect-ACSDb</td>
<td>has-header</td>
<td>0.89</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>no-header</td>
<td>0.75</td>
<td>0.80</td>
</tr>
</tbody>
</table>
Structure of corpus

• Assume that corpus $\mathcal{R}$ of databases, where each database $R \in \mathcal{R}$ is a single relation.

• The URL $R_u$ and offset $R_i$ within its page uniquely define $R$.

• Schema $R_s$ is an ordered list of attributes.

• $R_t$ is the list of tuples, with tuple-size no more than $|R_s|$.
Attribute Correlation Statistics

- For each unique schema $R_s$, the ACSDb $A$ contains a pair of the form $(R_s,c)$ where $c$ is the frequency of schema $R_s$

*Function createACS(R)*

```plaintext
A = {}
seenDomains = {}
for all $R \in R$
    if getDomain(R.u) $\notin$ seenDomains[R.S] then
        seenDomains[R.S].add(getDomain(R.u))
    end if
end for
```
Attribute Correlation Statistics

• While calculating the frequency, two schemas are considered identical irrespective of the order of the attributes within the schema.

• The frequency of occurrence of a schema is counted only once for various URLs within a single domain name.

• Recovered relations contain 5.4M unique attribute labels in 2.6M unique schemas.

• Using the various counts in the ACSDb, we can calculate probabilities of seeing various attributes in a schema.
ACSDb is useful for computing attribute probabilities

- \( p(\text{"make"}) \)
- \( p(\text{"model"}) \)
- \( p(\text{"zipcode"}) \)
- \( p(\text{"make"} | \text{"model"}) \)
- \( p(\text{"make"} | \text{"zipcode"}) \)
Relational Search

- WebTables search system allows for keyword based search queries
- The results are meaningful by themselves
- Possible query-appropriate visualization of data
- But none of these extensions will be useful without good search relevance
Relational Search

• Example where query is “city population”

• The top result is a table with attributes – City/Urbam Area, Country, Population, Rank.

• WebTables extracted data from HTML tables, applied ranking, formatting and generated the visualization
Relational Ranking

• Challenges posed for ranking web-extracted relations:

  – Relations do not exist in domain-specific schema graph

  – It is not clear which table in a page is described by a frequent word

  – Attribute labels are extremely important, even if they appear infrequently

  – A high quality page may contain tables of varying quality

  – Relations have special features that must be taken into consideration – schema elements, presence of keys, size of relation, # of NULLs
Relational Ranking

- **NaïveRank**:
  - Sends user query to search engine
  - Fetches top-k pages and extracts the tables from each page
  - If more than one table exits per page, they are returned in document order
  - If less than k tables are returned, will not go deeper in search results.

1: *Function naiveRank*(q, k):
2: let U = urls from web search for query q
3: for i = 0 to k do
4:.emit getRelations(U[i])
5: end for
Relational Ranking

• **FilterRank:**

  – Similar to NaïveRank, but more sophisticated
  – If it cannot extract at least \( k \) relations from the top-\( k \) results, it will search beyond the top-\( k \) results until it extracts at least \( k \) relations.

```plaintext
1: Function filterRank(q, k):
2: let U = ranked urls from web search for query q
3: let numEmitted = 0
4: for all u ∈ U do
5:   for all r ∈ getRelations(u) do
6:     if numEmitted >= k then
7:       return
8:   end if
9:     emit r; numEmitted +=
10:   end for
11: end for
```
Relational Ranking

- **FeatureRank**
  - Does not rely on existing search engine
  - Uses query dependent and query independent features (listed on the next slide) to score each extracted relation in the corpus
  - A Linear Regression Estimator `score()` merging the above features is trained on thousands of \((q, relation)\) pairs judged by two independent judges in scale of 1-5
  - Heavily weighted features include # hits in each relation’s schema and # hits in each relation’s leftmost column
  - This matches intuition that firstly, the attribute labels are a strong indicator of the relation’s subject matter and secondly, the leftmost column is usually a “semantic key” of the relation
Relational Ranking

Query independent features:
- # rows, # cols
- has-header?
- # of NULLs in table
- document-search rank of source page

Query dependent features:
- # hits on header
- # hits on leftmost column
- # hits on second-to-leftmost column
- # hits on table body

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

- **SchemaRank**
  - Uses ACSDb-based schema coherency score

  - *Coherent Schema* is one where all attributes are tightly related to one another in the schema
  - **High**: \{make, model\}
  - **Low**: \{make, zipcode\}

  - *Pointwise Mutual Information (PMI)* determines how strongly two items are related.

    - values range from positive (strongly related) to zero (independent) to negative (negatively correlated)

  - Coherency score for schema s is average pairwise PMI scores over all pairs of attributes in the schema.
Relational Ranking

• Coherency Score

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

• Pointwise Mutual Information (PMI)

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

- Traditional IR systems use **inverted index** that maps each term to its posting list of (docid, offset)

- WebTables data exists in two dimensions

- Ranking function uses both (x,y) offsets that describe where in the table is the data

- This method supports queries with spatial operators like `samerow` and `samecol`

- Example: Can search for Paris and France on same row, Paris, London and Madrid in same column
ACSDb Applications

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

- Build to assist novice database designers when creating relational schema

- User enters one or more domain-specific attributes (example: “make”)

- The system guesses attribute labels that should be appropriate to the target domain (example: “model”, “year”, “price”, “mileage”)

- System does not use any input database or make any query operation to perform its task.
Schema Auto-Complete

• For an input $I$, the best schema $S$ of given size is one that maximizes $p(S-I | I)$

• The probability values can be obtained from the ACSDb

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

• We may use Greedy Algorithm which will select the next-most-probable attribute

• The algorithm will stop when the overall schema’s probability drops below threshold

• Does not guarantee maximal solution, but is interactive

• System never retracts previously accepted attribute

• Approach is weak when most probable attribute is common to multiple strongly distinct domains (example: “name”)

• For such cases, it is better to present domain based suggestions to the user using clustering
ACSDb Applications

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

- Traditionally done using Thesauri
- But Thesauri are difficult to compile and do not support non-natural-language strings
- Input is set of context attributes, $C$
- Output is list of attribute pairs $P$ that are likely to be synonymous in schemas that contain $C$
- Example: For attribute “artist”, output is “song/track”.
Attribute Synonym-Finding

• Synonymous attributes \( a \) and \( b \) will never appear in the same schema, \( p(a,b) = 0 \)

• Synonymous attributes must appear in schemas fairly frequently. If \( p(a,b) = 0 \) when \( p(a)p(b) \) is large, syn score should be high

• Two synonyms will appear often in similar contexts: for a third attribute \( z, z \in C, z \in A \), \( p(z|a,C) = p(z|b,C) \)
  • If \( a, b \) always “replace” each other, then the denominator will be close to zero. If they rarely “replace” each other (appearing with quite different contexts), then the denominator will be large.

• Using formula, we calculate \( \text{syn}(a,b) \) for all possible synonym pairs and return all with value > threshold \( t \)
### Attribute Synonym-Finding

\[
\text{syn}(a, b) = \frac{p(a)p(b)}{\epsilon + \sum_{z\in A}(p(z|a, C') - p(z|b, C'))^2}
\]

1: **Function SynFind(C, t):**
2: \( R = [ ] \)
3: \( A = \) all attributes that appear in ACSDb with C
4: for \( a \not\in A, b \in B, \) s.t. \( a \neq b \) do
5: \hspace{1em} if \( (a, b) \in \text{ACSDb} \) then
6: \hspace{2em} // Score candidate pair with syn function
7: \hspace{2em} if \( \text{syn}(a, b) > t \) then
8: \hspace{3em} \( R.\text{append}(a, b) \)
9: \hspace{2em} end if
10: end if
11: end for
12: sort \( R \) in descending syn order
13: return \( R \)
ACSDb Applications

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

• Join Graph N,L is created for every unique schema N and undirected join link between common attributes.

• Such join graph becomes very busy since attribute like “name” link to many schemas.

• Thus to reduce clutter, cluster together similar schema neighbors.

• Use join neighbor similarity to measure whether a shared attribute D plays similar role in schema X and Y.
Join Graph Traversal

• \textit{neighborSim} is similar to coherency score.
• If the two schemas cohere well, they are clustered together, else they are clustered separately.
• Using a starting schema S, WebTables uses neighborSim as metric to cluster similar schemas.
• Thus user will have to choose from fewer outgoing links.

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

1: Function ConstructJoinGraph(A, F):
2:    N = {}  
3:    L = {}  
4:    for (S, c) ∈ A do  
5:        N.add(S)  
6:    end for  
7:    for S, c) ∈ A do  
8:        for attr ∈ F do  
9:            if attr ∈ S then  
10:                L.add((attr,F, S))  
11:            end if  
12:        end for  
13:    end for  
14:    return N,L
Experimental Results

• Ranking: compared 4 rankers on test set
  – Naïve: Top-10 *pages* from google.com
  – Filter: Top-10 *good tables* from google.com
  – Rank: Trained ranker
  – Rank-ACSDb: Trained ranker with ACSDb score

• Fraction of top-k that are relevant shows that we do well at high grain

<table>
<thead>
<tr>
<th>k</th>
<th>Naïve</th>
<th>Filter</th>
<th>Rank</th>
<th>Rank-ACSDb</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.26</td>
<td>0.35 (34%)</td>
<td>0.43 (65%)</td>
<td>0.47 (80%)</td>
</tr>
<tr>
<td>20</td>
<td>0.33</td>
<td>0.47 (42%)</td>
<td>0.56 (70%)</td>
<td>0.59 (79%)</td>
</tr>
<tr>
<td>30</td>
<td>0.34</td>
<td>0.59 (74%)</td>
<td>0.66 (94%)</td>
<td>0.68 (100%)</td>
</tr>
</tbody>
</table>
Experimental Results

• Schema Auto-Completion:
  – Output schema is almost always coherent
  – But it is desirable to get most relevant attributes
  – 6 humans created schema for 10 test databases with input given on the next slide.
  – System was allowed to make 3 tries, each time removing all members of emitted schema $S$
## Experimental Results

- **Schema Auto-Completion**

<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>

### Table

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>0</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>instructor</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>elected</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>ab</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>stock-symbol</td>
<td>0.4</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>company</td>
<td>0.22</td>
<td>0.33</td>
<td>0.44</td>
</tr>
<tr>
<td>director</td>
<td>0.75</td>
<td>0.75</td>
<td>1.0</td>
</tr>
<tr>
<td>album</td>
<td>0.5</td>
<td>0.5</td>
<td>0.66</td>
</tr>
<tr>
<td>sqft</td>
<td>0.5</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>goals</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>Average</td>
<td>0.46</td>
<td>0.59</td>
<td>0.62</td>
</tr>
</tbody>
</table>
• Synonym-Finding:
  – Judges determined if given pair are synonyms or not
  – Then tested top-k results for accuracy
  – Accuracy is good for lower values of k
  – Accuracy falls as k value increases, since the pairs being returned are more general
## Experimental Results

- **Synonym Finding:**

<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>

<table>
<thead>
<tr>
<th>Input</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>1.0</td>
<td>0.8</td>
<td>0.67</td>
<td>0.55</td>
</tr>
<tr>
<td>instructor</td>
<td>1.0</td>
<td>1.0</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>elected</td>
<td>0.4</td>
<td>0.4</td>
<td>0.33</td>
<td>0.3</td>
</tr>
<tr>
<td>ab</td>
<td>0.6</td>
<td>0.4</td>
<td>0.33</td>
<td>0.25</td>
</tr>
<tr>
<td>stock-symbol</td>
<td>1.0</td>
<td>0.6</td>
<td>0.53</td>
<td>0.4</td>
</tr>
<tr>
<td>company</td>
<td>0.8</td>
<td>0.7</td>
<td>0.67</td>
<td>0.5</td>
</tr>
<tr>
<td>director</td>
<td>0.6</td>
<td>0.4</td>
<td>0.26</td>
<td>0.3</td>
</tr>
<tr>
<td>album</td>
<td>0.6</td>
<td>0.6</td>
<td>0.53</td>
<td>0.45</td>
</tr>
<tr>
<td>sqft</td>
<td>1.0</td>
<td>0.7</td>
<td>0.53</td>
<td>0.55</td>
</tr>
<tr>
<td>goals</td>
<td>1.0</td>
<td>0.8</td>
<td>0.73</td>
<td>0.75</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.8</td>
<td>0.64</td>
<td>0.55</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Experimental Results

- Join Graph Traversal
Conclusion

• First large-scale attempt to extract HTML table data

• Showed that there is better way to search and rank relational data on the web

• Created unique ACSDb statistics

• Showed utility of ACSDb with several novel applications
Future Scope

• Combining current scenario with a “row-centric” analogue of the ACSDb

• Using tuple-keys as an analogue to attribute labels, we can create “data-suggest” features

• Creating new data sets by integrating this corpus with user generated data

• Expanding the WebTables search engine to incorporate a page quality metric like PageRank

• Include non-HTML tables, deep web databases and HTML Lists
References and Related Work


- Cindy Xide Lin, Bo Zhao, Tim Weninger, Jiawei Han, and Bing Liu. 2010. *Entity relation discovery from web tables and links*. In Proceedings of the 19th international conference on World wide web (WWW). ACM, New York, 1145-1146


- “*Longitudinal Analytics of Web Archive Data*” - LAWA project: http://www.lawa-project.eu/index.php
Discussions