Harvesting Relational Tables from Lists on the Web

Hazem Elmeleegy, Jayant Madhavan and Alon Halevy

Presented by: Ella Bolshinsky
Lists

- Multiple lists appear on web pages
  - Plentiful source of relational data
  - Mostly manually generated lists

---

Restaurant:

- Le rendez-vous, 15 Hanadiv, Zichron Yaakov, 0542094363 (French bassari)
- Shealtieli on the Hof Ha Carmel beach 04-8550393 (meat)
- Shipudey Hatikva, Moriya 46, 04-8101100 (meat)
- Pasta Carolla, Haatzmaut 54, 0572-223333 ext 5059 (vegetarian, halavi)
- Pomodoro, Horev 16, (italian, halavi)
- El Guacho, Yefe Nof 120, 04-8370997 (bassari)
- Papagayo, Tsome Vulcan, Khoutsot Hamifrats, 04-8422666 (bassari)
- Cat blue, Tsome Beit Oren, 04-8248474, (bassari)
- Tanduka, Yokneam, 04-9590323, (bassari)
Versus Tables

- Splitting the lists into multi-column tables
  - More sophisticated querying
  - Enable advanced features
Why is it Hard?  
Lets Look at an Example...

- looking for the word “Ella” in wikipedia
  - Ella Koon, Hong Kong singer
  - Ella Maillart (1903–1997), Swiss adventurer, travel writer, photographer and sportswoman
  - Ella Mae Morse (1924–1999), American popular singer from the 1940s
  - Ella Pamfilova (born 1953), Russian politician
  - Ella (singer) (born 1966), popular Malaysian rock singer
Why is it Hard?
Inconsistent Delimiters (if Exist) and Unstructured Lines

- Ella Koon, Hong Kong singer
- Ella Maillart (1903–1997), Swiss adventurer, travel writer, photographer and sportswoman
- Ella Mae Morse (1924–1999), American popular singer from the 1940s
- Ella Pamfilova (born 1953), Russian politician
- Ella (singer) (born 1966), popular Malaysian rock singer
Why is it hard?
Missing Information - Different Number of Fields

- **Ella Koon**, Hong Kong singer
  - Name, city, job
- **Ella Maillart** (1903-1997), Swiss adventurer, travel writer, photographer and sportswoman
  - Name, birth date, death date, jobs
- **Ella Pamfilova** (born 1953), Russian politician
  - Name, birth date, job
Why is it Hard?
No Clear Notion of Columns or Cells

- **Ella Koon**, Hong Kong singer
  - First name, last name, city, job
  - Name, city, job
  - Name, job
Existing Solutions

- Rely on templates
  - Infeasible when working in web scale
- Look for patterns and HTML tags
  - In static web pages they don’t necessary exist
  - Lists are mostly manually created
- Work with documents of specific domain
  - Some require human supervision
  - For each new domain extension is needed
ListExtract
Algorithm Overview

- **Independent splitting**
  - Splitting each line in the list independently
    - Non overlapping and complete split

- **Alignment**
  - Fields are aligned into columns

- **Refinement**
  - Analyze the fields to detect and fix incorrect fields

- **Splitting all lines into records**

- **Deciding the num. of columns**

- **Re-merge and re-split long records**

- **Align short records**

- **Detect inconsistent fields / field streaks**

- **Re-merge and re-split field streaks**

- **Realign field streaks**
Independent splitting - before
Independent Splitting - After

<table>
<thead>
<tr>
<th></th>
<th>What's Opera Doc</th>
<th>Warner Bros</th>
<th>1957</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Duck Amuck</td>
<td>Warner Bros</td>
<td>1953</td>
</tr>
<tr>
<td>3</td>
<td>The Band Concert</td>
<td>Disney</td>
<td>1935</td>
</tr>
<tr>
<td>4</td>
<td>Duck Dodgers in the 24 1/2th Century (Warner Bros</td>
<td>1953</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>One Froggy Evening</td>
<td>Warner Bros</td>
<td>1956</td>
</tr>
<tr>
<td>6</td>
<td>Gertie The Dinosaur</td>
<td>McCay</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Red Hot Riding Hood</td>
<td>MGM</td>
<td>1943</td>
</tr>
<tr>
<td>8</td>
<td>Porky In Wackyland</td>
<td>Warner Bros</td>
<td>1938</td>
</tr>
<tr>
<td>9</td>
<td>Gerald McBoing Boing</td>
<td>UPA</td>
<td>1951</td>
</tr>
<tr>
<td>10</td>
<td>King-Size Canary</td>
<td>MGM</td>
<td>1947</td>
</tr>
<tr>
<td>11</td>
<td>Three Little Pigs</td>
<td>Disney</td>
<td>1933</td>
</tr>
<tr>
<td>12</td>
<td>Rabbit of Seville</td>
<td>Warner Bros</td>
<td>1950</td>
</tr>
<tr>
<td>13</td>
<td>Steamboat Willie</td>
<td>Disney</td>
<td>1928</td>
</tr>
<tr>
<td>14</td>
<td>The Old Mill</td>
<td>Disney</td>
<td>1937</td>
</tr>
<tr>
<td>15</td>
<td>Bad Luck Blackie (MGM</td>
<td>1949</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>The Great Piggy Bank Robbery</td>
<td>Warner Bros</td>
<td>1946</td>
</tr>
<tr>
<td>17</td>
<td>Popeye the Sailor</td>
<td>Meets</td>
<td>Sinbad the Sailor</td>
</tr>
</tbody>
</table>
Independent Splitting Algorithm

- Extract all sequences from the input as field candidates

  - For m words, \( \binom{m+1}{2} \) options

Input = “Bugs bunny rabbit”

“Bugs bunny rabbit”

“Bugs bunny”

“Bugs”

“bunny rabbit”

“bunny”

“rabbit”
Independent Splitting Algorithm
Cont.

- Calculate FQ for each field candidate
- Sort in descending order

<table>
<thead>
<tr>
<th>FQ</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>“Bugs bunny”</td>
</tr>
<tr>
<td>0.5</td>
<td>“Bugs bunny rabbit”</td>
</tr>
<tr>
<td>0.4</td>
<td>“rabbit”</td>
</tr>
<tr>
<td>0.3</td>
<td>“bunny rabbit”</td>
</tr>
<tr>
<td>0.1</td>
<td>“bunny”</td>
</tr>
<tr>
<td>0.1</td>
<td>“Bugs”</td>
</tr>
</tbody>
</table>

Input = “Bugs bunny rabbit”
Independent Splitting Algorithm

Cont.

Results:

<table>
<thead>
<tr>
<th>FQ</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>“Bugs bunny”</td>
</tr>
<tr>
<td>0.5</td>
<td>“Bugs bunny rabbit”</td>
</tr>
<tr>
<td>0.4</td>
<td>“rabbit”</td>
</tr>
<tr>
<td>0.3</td>
<td>“bunny rabbit”</td>
</tr>
<tr>
<td>0.1</td>
<td>“bunny”</td>
</tr>
<tr>
<td>0.1</td>
<td>“Bugs”</td>
</tr>
</tbody>
</table>
Independent Splitting - FQ

- **FQ(f)**: Field quality score for field candidate f.

\[
FQ(f) = a_{st} \times S_{st}(f) + a_{lms} \times S_{lms}(f) + a_{tcs} \times S_{tcs}(f)
\]

- \(S_{st}(f)\) – Type score
- \(S_{lms}(f)\) – Language model score
- \(S_{tcs}(f)\) – Table corpus support score
- \(a_{st}, a_{lms}, a_{tcs}\) – Weights

- Splitting all lines into records
  - Deciding the num. of columns
  - Re-merge and re-split long records
  - Align short records
  - Detect inconsistent fields / field streaks
  - Re-merge and re-split field streaks
  - Realign field streaks
Type Score

“dude@gmail.com”

Splitting all lines into records
  - Deciding the num. of columns
  - Re-merge and re-split long records
  - Align short records
  - Detect inconsistent fields / field streaks
  - Re-merge and re-split field streaks
  - Realign field streaks

Regular expressions
URL
email

Type score = 1
Language Model Score

Definition

- A Language model records the probability of occurrences of word sequences

- \( P(\text{“And nothing but the truth”}) \approx 0.001 \)

- \( P(\text{“And nuts sing on the roof”}) \approx 0 \)
Language Model Score
Usage

- The probability of each word in the field to appear after the preceding words
- The probability of the words in the “field’s margins” to appear by the words adjacent to them
- Using a large scale language model that records words co-occurrence scores
Table Corpus Support

- How many times did it appear as a field in tables in the Web
- A large corpus of automatically extracted HTML tables
FQ(f) – Field Quality Score

Summary

- Scaling components to prefer longer fields
- Normalize components to 0-1 values

\[ FQ(f) = a_{st} \times S_{st}(f) + a_{lms} \times S_{lms}(f) + a_{tcs} \times S_{tcs}(f) \]

- \( S_{st}(f) \) – Type score
- \( S_{lms}(f) \) – Language model score
- \( S_{tcs}(f) \) – Table corpus support score
- \( a_{st}, a_{lms}, a_{tcs} \) – Weights
Alignment Phase

- What did we do?
  - Split each line independently

- What will we do?
  - Decided about the columns num.
  - Align short and long records
  - Refine our solution
Deciding the Number of Columns

- Before creating a table decide what is the number of columns
- Pick the most common number of columns (will be marked by k)
  - Reasonable if there are not too many nulls
  - Lines with k fields are aligned

| 1 | What's Opera Doc | Warner Bros | 1957 |
| 2 | Duck Amuck | Warner Bros | 1953 |
| 3 | The Band Concert | Disney | 1935 |
| 4 | Duck Dodgers in the 24 1/2th Century (Warner Bros | 1953 |
| 5 | One Froggy Evening | Warner Bros | 1956 |
| 6 | Gertie The Dinosaur | McCay |
| 7 | Red Hot Riding Hood | MGM | 1943 |
| 8 | Porky In Wackyland | Warner Bros | 1938 |
| 9 | Gerald McBoing Boing | UPA | 1951 |
| 10 | King-Size Canary | MGM | 1947 |
| 11 | Three Little Pigs | Disney | 1933 |
| 12 | Rabbit of Seville | Warner Bros | 1950 |
| 13 | Steamboat Willie | Disney | 1928 |
| 14 | The Old Mill | Disney | 1937 |
| 15 | Bad Luck Blackie (MGM | 1949 |
| 16 | The Great Piggy Bank Robbery | Warner Bros | 1946 |
| 17 | Popeye the Sailor | Meets | Sinbad the Sailor | Fleischer | 1936 |
Align Long Records

- Splitting all lines into records
- Deciding the number of columns
- Re-merge and re-split long records
- Align short records
- Detecting inconsistent fields / field streaks
- Re-merge and re-split field streaks
- Realigning field streaks

(b) After re-splitting records given the number of columns
Align Long Records

- Re-split lines with more than k fields
- The same algorithm as before, but with constraint over fields number
  - Before selecting field candidate, insure that it doesn’t lead to the constraint violation.

Example: 4 column table
- 1 The Old Mile Disney 1937
### Align Short Records

<table>
<thead>
<tr>
<th>1. What's Opera Doc</th>
<th>Warner Bros</th>
<th>1957</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Duck Amuck</td>
<td>Warner Bros</td>
<td>1953</td>
</tr>
<tr>
<td>3. The Band Concert</td>
<td>Disney</td>
<td>1935</td>
</tr>
<tr>
<td>4. Duck Dodgers in the 24 1/2th Century (Warner Bros)</td>
<td>1953</td>
<td></td>
</tr>
<tr>
<td>5. One Froggy Evening</td>
<td>Warner Bros</td>
<td>1956</td>
</tr>
<tr>
<td>6. Gertie The Dinosaur</td>
<td>McCoy</td>
<td></td>
</tr>
<tr>
<td>7. Red Hot Riding Hood</td>
<td>MGM</td>
<td>1943</td>
</tr>
<tr>
<td>8. Porky In Wackylend</td>
<td>Warner Bros</td>
<td>1938</td>
</tr>
<tr>
<td>9. Gerald McBoing Boing</td>
<td>UPA</td>
<td>1951</td>
</tr>
<tr>
<td>10. King-Size Canary</td>
<td>MGM</td>
<td>1947</td>
</tr>
<tr>
<td>11. Three Little Pigs</td>
<td>Disney</td>
<td>1933</td>
</tr>
<tr>
<td>12. Rabbit of Seville</td>
<td>Warner Bros</td>
<td>1950</td>
</tr>
<tr>
<td>13. Steamboat Willie</td>
<td>Disney</td>
<td>1928</td>
</tr>
<tr>
<td>14. The Old Mill</td>
<td>Disney</td>
<td>1937</td>
</tr>
<tr>
<td>15. Bad Luck Blackie (MGM)</td>
<td></td>
<td>1949</td>
</tr>
<tr>
<td>16. The Great Piggy Bank Robbery</td>
<td>Warner Bros</td>
<td>1946</td>
</tr>
<tr>
<td>17. Popeye the Sailor Meets</td>
<td>Sinbad the Sailor</td>
<td>Fleischer</td>
</tr>
</tbody>
</table>

(c) After alignment phase (initial table $T_I$)
Align Short Records

- Nulls will be inserted in lines with less than k fields.
- Problem: Where to insert the nulls?
  - Align each field with the column most similar to it while preserving field order.

- Splitting all lines into records
- Deciding the num. of columns
- Re-merge and re-split long records
- Align short records
- Detect inconsistent fields / field streaks
- Re-merge and re-split field streaks
- Realign field streaks
Align Short Records

Algorithm

- Dynamic programming
  - Solving complex problems by breaking them down into simpler sub-problems

\[
M[i, j] = \max \left\{ M[i-1, j] + \text{Unmatched}(f_i), M[i, j-1] + \text{Unmatched}(c_j), M[i-1, j-1] + \text{Matched}(f_i, c_j) \right\}
\]

\[
M[0, 0] = 0 \quad \text{Unmatched}(c_j) = \text{some constant}
\]

\[
M[0, j] = M[0, j-1] + \text{Unmatched}(c_j) \quad \text{Unmatched}(f_i) = -\infty
\]

\[
M[j, 0] = M[j-1, 0] + \text{Unmatched}(f_i) \quad \text{Matched}(f_i, c_j) = F2FC(f_i, c_j)
\]
F2FC
Field to field consistency score

Field to field consistency score for field f and column c: \( F2FC(f, c) = \frac{1}{n} \times \sum_{i=1}^{n} F2FC(f, f^c_i) \)

\( f^c_i \) – The field on row i, column c

\[ F2FC(B,2) = \left( \frac{1}{4} \right) \times \left[ F2FC(B,A) + F2FC(B,B) + F2FC(B,C) + F2FC(B,D) \right] \]
F2FC
Field to Field Consistency Score

Field to field consistency score for fields f1,f2:

\[ F2FC(f_1, f_2) = a_{tc} \times S_{tc} (f_1, f_2) + a_{tcc} \times S_{tcc} (f_1, f_2) + a_{sc} \times S_{sc} (f_1, f_2) + a_{dc} \times S_{dc} (f_1, f_2) \]

- \( S_{tc} (f_1, f_2) \) – Type consistency score (if the fields have the same type)
- \( S_{tcc} (f_1, f_2) \) – Table corpus consistency score (the probability for \( f_1, f_2 \) to appear in the same column)
- \( S_{sc} (f_1, f_2) \) – Syntax consistency score (if the fields have the same "appearance")
- \( S_{dc} (f_1, f_2) \) – Delimiters consistency score (if the fields have the same delimiters before and after)
- \( a_{tc}, a_{tcc}, a_{sc}, a_{dc} \) – Weights
Type Consistency

- "dude@gmail.com"
- "bunny@yahoo.com"

Regular expressions

URL

email

Type Consistency = 1
### Table Corpus Consistency

<table>
<thead>
<tr>
<th>Presidents</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Nicolas Sarkozy</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ S_{tcc}("Barack Obama", "Nicolas Sarkozy") > 0 \]
Syntax Consistency

- Do the fields look similar
  - Comparing number of letters, upper/lower letters, digits, punctuation marks, etc.

- Example:
  - 05-2192111 vs. 09-2938453
  - 05-2192111 vs. Disney
Delimiter Consistency

- Same delimiter **before** the field => +0.5
- Same delimiter **after** the field => +0.5

**Example:**

- (MGM) ↔ (Disney) => Score = 1
- MGM, ↔ :Disney, => Score = 0.5
- MGM; ↔ :Disney, => Score = 0
F2FC
Field to Field Consistency Score

F2FC(f₁, f₂) = a_{tc} × S_{tc}(f₁, f₂) + a_{tcc} × S_{tcc}(f₁, f₂) + a_{sc} × S_{sc}(f₁, f₂) + a_{dc} × S_{dc}(f₁, f₂)

F2FC(f, c) = \frac{1}{n} × \sum_{i=1}^{n} F2FC(f, f_i^c)
Field Summaries

- Using n fields for F2FC calculation can be expensive
  - Create field summaries
    - Configurable number of representatives for each column
      - Selected independently from different records
    - Updated when additional records are aligned
  - Example:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Steamboat Willie</td>
<td>Disney</td>
<td>1943</td>
</tr>
<tr>
<td>10</td>
<td>Rabbit of Seville</td>
<td>MGM</td>
<td>1935</td>
</tr>
<tr>
<td>15</td>
<td>Three Little Pigs</td>
<td>Disney</td>
<td>1949</td>
</tr>
</tbody>
</table>
Refinement Phase

What did we do?
- Split each line independently
- Decided about the columns num.
- Align short and long records

What will we do?
- Detect inconsistent fields
- Try to split and align them correctly
Refinement

Splitting all lines into records
Deciding the num. of columns
Re-merge and re-split long records
Align short records
Detect inconsistent fields / field streaks
Re-merge and re-split field streaks
Realign field streaks

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>What's Opera Doc</td>
<td>Warner Bros</td>
</tr>
<tr>
<td>2</td>
<td>Duck Amuck</td>
<td>Warner Bros</td>
</tr>
<tr>
<td>3</td>
<td>The Band Concert</td>
<td>Disney</td>
</tr>
<tr>
<td>4</td>
<td>Duck Dodgers in the 24 1/2th Century</td>
<td>Warner Bros</td>
</tr>
<tr>
<td>5</td>
<td>One Froggy Evening</td>
<td>Warner Bros</td>
</tr>
<tr>
<td>6</td>
<td>Gertie the Dinosaur</td>
<td>McCay</td>
</tr>
<tr>
<td>7</td>
<td>Red Hot Riding Hood</td>
<td>MGM</td>
</tr>
<tr>
<td>8</td>
<td>Porky In Wackyland</td>
<td>Warner Bros</td>
</tr>
<tr>
<td>9</td>
<td>Gerald McBoing Boing</td>
<td>UPA</td>
</tr>
<tr>
<td>10</td>
<td>King-Size Canary</td>
<td>MGM</td>
</tr>
<tr>
<td>11</td>
<td>Three Little Pigs</td>
<td>Disney</td>
</tr>
<tr>
<td>12</td>
<td>Rabbit of Seville</td>
<td>Warner Bros</td>
</tr>
<tr>
<td>13</td>
<td>Steamboat Willie</td>
<td>Disney</td>
</tr>
<tr>
<td>14</td>
<td>The Old Mill</td>
<td>Disney</td>
</tr>
<tr>
<td>15</td>
<td>Bad Luck Blackie</td>
<td>MGM</td>
</tr>
<tr>
<td>16</td>
<td>The Great Piggy Bank Robbery</td>
<td>Warner Bros</td>
</tr>
<tr>
<td>17</td>
<td>Popeye the Sailor</td>
<td>Meets Sinbad the Sailor (Fleischer)</td>
</tr>
</tbody>
</table>

(d) After refinement phase (final table $T$)
Refinement

We assume:

- Rows on the list are related
- The number of correctly split lines is greater than incorrectly split lines
- Conclusion: Incorrect fields will align badly
Inconsistent Streaks

- Incorrect splits occur in **streaks**
  
<table>
<thead>
<tr>
<th>A</th>
<th>I am incorrect</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Individual inconsistent fields are grouped into streaks
  - Nulls streaks ignored
  - Single field streaks ignored
Detecting Inconsistent Streaks

$\% P_{inc}$ of the fields with the lowest F2FC are inconsistent

- F2FC with respect to the field summaries
- Null’s F2FC is 0

What should I do with those streaks?
Correcting Inconsistent Streaks

- **F**(i,j₁,j₂) - streak in record i from column j₁ to column j₂
  - **Re merge** \( F(i,j₁,j₂) \)
  - **Re split**
    - Set maximal number of columns (as when aligning long records)
    - Add \( S_{ls} \) – List support score to FQ calculation
      - Checks consistency with any column between j₁ and j₂
      - Formula: \( S_{ls}(f) = \max_{h=j₁}^{j₂} (F2FC(f, SF_{h})) \)
  - **Re align**
    - Nulls might be inserted

### Examples

<table>
<thead>
<tr>
<th>Danni</th>
<th>1985</th>
<th>Israel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranni</td>
<td>1987</td>
<td>USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Danni</th>
<th>1985</th>
<th>Israel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranni</td>
<td>1987</td>
<td>USA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Danni</th>
<th>1985</th>
<th>Israel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranni</td>
<td>1987</td>
<td>USA</td>
</tr>
</tbody>
</table>
We would like to know the quality of the result for instance in applications that use the extracted tables.

- TE(T) - Table extraction score for table T
  - Average FQ score for all fields in the extracted table.
Algorithm Summary

- First split each line independently
- Align all the records
  - Including the short and the long records
- Refine the solution
  - At least once
  - Usually one refinement is enough
- Evaluate the resulting table
Experiments : Before We Start

- Is the algorithm clear?
Experiments – What For?

- The ability to correctly extract relational tables
- Contribution of various constituents
- Comparison with information extraction systems
- Potential for harvesting information from the Web
Data Sources for Experiments

- In English only
- Wlists - HTML lists from the web
  - Different domains
- TDLists - Lists constructed from tables from the web
  - The constructed tables are compared to the original tables
TDList Generation

- Collapsing all cells into rows (space separators)

<table>
<thead>
<tr>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moriya 58 Haifa</td>
<td>04-8349950 / 04-8349950</td>
</tr>
<tr>
<td>hanamal 24 Haifa</td>
<td>04-8628899</td>
</tr>
<tr>
<td>Sderot ben gurion 25 Haifa</td>
<td>04-85111919</td>
</tr>
<tr>
<td>Ben Gurion Blvd 6 Haifa</td>
<td>04-8552201</td>
</tr>
<tr>
<td>Moriah Blvd 110 Haifa</td>
<td>04-8344502, 04-8667722, 04-8345548</td>
</tr>
</tbody>
</table>
Results Evaluation : Before we Start

- How would you split “Isaac Newton”?
  - Two columns:
    - First name: Isaac | Last name: Newton
  - One column:
    - Name: Isaac Newton

- Not every list
  - Constructs a relational table
  - Can be extracted
F-measure

- A measure of a test's accuracy.
- Considers:
  - **Precision** is the fraction of retrieved instances that are relevant
  - **Recall** is the fraction of relevant instances that are retrieved

\[
\text{Precision} = \frac{\{\text{Relevant} \cap \{\text{Retrieved}\}}}{\{\text{Retrieved}\}} = P
\]

\[
\text{Recall} = \frac{\{\text{Relevant} \cap \{\text{Retrieved}\}}}{\{\text{Relevant}\}} = R
\]

\[
\text{F-measure} = \frac{2RP}{R + P}
\]
### F-measure

- $T^{total}$ – The number of cells in the generated table
- $T^{total}_g$ – The number of cells in the "ground truth" table
- $T^{correct}$ – The number of correctly extracted cells.

\[
P = \frac{\{\text{Relevant}\} \cap \{\text{Retrieved}\}}{\{\text{Retrieved}\}} = \frac{T^{correct}}{T^{total}} \quad ; \quad R = \frac{\{\text{Relevant}\} \cap \{\text{Retrieved}\}}{\{\text{Relevant}\}} = \frac{T^{correct}}{T^{total}_g}
\]

\[
F\text{-measure} = \frac{2RP}{R + P} = \ldots = \frac{2T^{correct}}{T^{total} + T^{total}_g}
\]

\[
F\text{-measure} = \frac{(2 \times 7)}{(12 + 9)} = 0.66
\]
TE(T)’s Accuracy

- **X-axis**: top x% of the tables, sorted by the TE(T)
- **Y-axis**: average F-measure
**FQ - Field Quality**

- **T** – type support
- **LM** – language model support
- **WT** – table corpus support

$$FQ(f) = a_{st} \times T + a_{lms} \times LM + a_{tcs} \times WT$$
F2FC
Field to Field Consistency Score

- TC – type consistency
- WC – table corpus consistency
- SC – syntax consistency
- DC – delimiters consistency

Should we remove delimiters consistency?
Does Refinement Help?

- Wlists: 10%-20% improvement
  - More significant in tables with high F-measure
- TDLLists: less than 5% improvement
Max_n_rep = 1 or 2 is not enough
Max_n_rep >= 3 gives similar performance

**Conclusion:** It’s enough to calculate a table with 3 rows.
Large Scale Table Extraction

- 100000 random English Web pages
  - 5-50 lines per list
  - Lines shorter than 100 letters
  - Total ~32000 lists
  - 11000 tables had more than one column

- Under conservative TE=0.6
  - 1.4% tables are extracted
What Next?

- Define columns headers
- Different columns order in different records
- Possible improvements to the algorithm
  - Example: to make a bounded split once the number of columns are known
Conclusions

- ListExtract (conservatively) able to extract well 1.4% of the given lists
  - Millions of lists from the Web
    - Assuming each Website has about one list
  - Data extraction advantages
- Self estimation which proves empirically to be rather correct
Thank You!