Blocking Schemes for Record Linkage

Matthew Michelson
CSCI 548
2006
# Record Linkage – Finding Matches

## Census Data

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Phone</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matt</td>
<td>Michelson</td>
<td>555-5555</td>
<td>12345</td>
</tr>
<tr>
<td>Jane</td>
<td>Jones</td>
<td>555-1111</td>
<td>12345</td>
</tr>
<tr>
<td>Joe</td>
<td>Smith</td>
<td>555-0011</td>
<td>12345</td>
</tr>
</tbody>
</table>

## A.I. Researchers

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Phone</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matthew</td>
<td>Michelson</td>
<td>555-5555</td>
<td>12345</td>
</tr>
<tr>
<td>Jim</td>
<td>Jones</td>
<td>555-1111</td>
<td>12345</td>
</tr>
<tr>
<td>Joe</td>
<td>Smeth</td>
<td>555-0011</td>
<td>12345</td>
</tr>
</tbody>
</table>
Record Linkage – Finding Matches

- Can’t compare all records!
  - Just 5,000 to 5,000 $\rightarrow$ 25,000,000 comparisons!
  - At 0.01s/comparison $\rightarrow$ 250,000 s $\rightarrow$ ~3 days!
- Need to use a subset of comparisons
  - “Candidate matches”
  - Want to cover true matches
  - Want to throw away non-matches
## Blocking – Generating Candidates

\[(\text{token, last name}) \text{ AND } (1^{\text{st}} \text{ letter, first name}) = \text{block-key}\]

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matt</td>
<td>Michelso</td>
</tr>
<tr>
<td>Jane</td>
<td>Jones</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matthew</td>
<td>Michelso</td>
</tr>
<tr>
<td>Jim</td>
<td>Jones</td>
</tr>
</tbody>
</table>

\[(\text{token, zip})\]

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matt</td>
<td>Michelson</td>
<td>12345</td>
</tr>
<tr>
<td>Matt</td>
<td>Michelson</td>
<td>12345</td>
</tr>
<tr>
<td>Matt</td>
<td>Michelson</td>
<td>12345</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matthew</td>
<td>Michelson</td>
<td>12345</td>
</tr>
<tr>
<td>Jim</td>
<td>Jones</td>
<td>12345</td>
</tr>
<tr>
<td>Joe</td>
<td>Smeth</td>
<td>12345</td>
</tr>
</tbody>
</table>

...
Blocking - Intuition

<table>
<thead>
<tr>
<th>First</th>
<th>Last Name</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matt</td>
<td>Michelso</td>
<td>12345</td>
</tr>
<tr>
<td>Jane</td>
<td>Jones</td>
<td>12345</td>
</tr>
<tr>
<td>Joe</td>
<td>Smith</td>
<td>12345</td>
</tr>
</tbody>
</table>

Census Data

1 Block of 12345 Zips

→ Compare to the “block-key”

Group & Check to reduce Checks
Bi-Gram Indexing

- Can we use a better blocking key than tokens?
  - What about “fuzzy” tokens?
    - Matt → Matthew, William → Bill? (similarity)
    - Michael → Mychael (spelling)

- Bi-Gram Indexing
Bi-Gram indexing

- **Step 1**: Take token and break it into bigrams
  - Token: matt
  - (‘ma,’ ‘at,’ ‘tt,’)

- **Step 2**: Generate all sub-lists
  - (# bigrams) x (threshold) = sub-list length
    - 3 x .7 = 2

- **Step 3**: Sort sub-lists and put them into inverted index
  - (‘at’ ‘ma’) (‘at’ ‘tt’) (‘ma’ ‘tt’) \(\rightarrow\) record w/ matt
    - Block key
BiGram Indexing: Properties

- Threshold properties
  - lower = shorter sub-lists → more lists
  - higher = longer sub-lists → less lists, less matches

- Now we can find spelling mistakes, close matches, etc…
Blocking – Multi-pass

- Sort neighborhoods on block keys
- Multiple independent runs using keys
  - runs capture different match candidates
- Attributed to (Hernandez & Stolfo, 1998)
- E.g.) 1\textsuperscript{st} \rightarrow (token, last name)
  2\textsuperscript{nd} \rightarrow (token, first name) &
  (token, phone)
Blocking – Multi-pass

- Can we make blocks without sorting?
  - Yes! We can cluster…
Blocking – Canopies Method

McCallum, Nigam, Ungar, Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching, 2000, KDD

Idea: form clusters around certain key values, within some threshold value
Blocking – Canopies Method

1. Start with 2 threshold values, T1 and T2, s.t. T1 > T2
   - based on similarity function, hand picked or learned thresholds
2. Select a random record from list of records and calculate it’s similarity to all other records
   - Very cheap in some cases: inverted index
3. Create “Canopy” for all records where similarity less than T1
4. Remove all records from the list of records where similarity less than T2
5. Repeat 1-4 until your list is empty
Blocking – Canopies Method

- Sim. function = abs. zip distance, $T_1 = 6$, $T_2 = 3$

List of records: 90001, 90002, 90006, 88181, 90292, 90293
Blocking – Multi-pass

- Back to the world of multi-pass…
- Terminology:
  - Each pass is a “conjunction”
    - (token, first) AND (token, phone)
  - Combine passes to form “disjunction”
    - [(token, last)] OR [(token, first) AND (token, phone)]
  - Disjunctive Normal Form rules
    - form “Blocking Schemes”
Blocking Effectiveness

- Determined by rules
  - Determined by choices for attributes and methods
    - (token, zip) captures all matches, but all pairs too
    - (token, first) AND (token, phone) gets half the matches, and only 1 candidate generated
    - Which is better? Why?
  - How to quantify??
Blocking Effectiveness

Reduction Ratio (RR) = 1 − ||C|| / (||S|| * ||T||)

S, T are data sets; C is the set of candidates

Pairs Completeness (PC) [Recall] = \( S_m / N_m \)

S_m = # true matches in candidates,
N_m = # true matches between S and T

Examples:
(token, last name) AND (1st letter, first name)

RR = 1 − 2/9 ≈ 0.78
PC = 1 / 2 = 0.50

(token, zip)

RR = 1 − 9/9 = 0.0
PC = 2 / 2 = 1.0
Multi-Pass Blocking Schemes

Old Techniques: Ad-hoc rules
New Techniques: Learn rules!

*Learned rules justified by quantitative effectiveness*

How to choose methods and attributes?

- **Blocking Goals:**
  - Small number of candidates (High RR)
  - Don’t leave any true matches behind! (High PC)

- **Previous approaches:**
  - Ad-hoc by researchers or domain experts

- **New Approach:**
  - Blocking Scheme Learner (BSL) – modified Sequential Covering Algorithm
Learning Schemes – Intuition

- Learn restrictive conjunctions
  - partition the space $\rightarrow$ minimize False Positives

- Union restrictive conjunctions
  - Cover all training matches
  - Since minimized FPs, conjunctions should not contribute many FPs to the disjunction
Example to clear things up!

Space of training examples

Rule 1 :- \((\text{zip|token}) \& (\text{first|token})\)

Final Rule :- \([(\text{zip|token}) \& (\text{first|token})] \cup [(\text{last|1}\text{\textsuperscript{st} Letter}) \& (\text{first|1}\text{\textsuperscript{st} Letter})]\)

- = Not match
- = Match
SCA: propositional rules

- Multi-pass blocking = disjunction of conjunctions
- Learn conjunctions and union them together!
- Cover all training matches to maximize PC

**SEQUENTIAL-COVERING**( class, attributes, examples, threshold)

| LearnedRules ← {} |
| Rule ← LEARN-ONE-RULE(class, attributes, examples) |
| While examples left to cover, do |
|   LearnedRules ← LearnedRules U Rule |
|   Examples ← Examples – {Examples covered by Rule} |
|   Rule ← LEARN-ONE-RULE(class, attributes, examples) |
| If Rule contains any previously learned rules, remove them |
| Return LearnedRules |
SCA: propositional rules

- LEARN-ONE-RULE is greedy
  - rule containment as you go, instead of comparison afterward
  - Ex) rule: \((\text{token|zip}) \& (\text{token|first})\)
    
    \((\text{token|zip}) \text{ CONTAINS } (\text{token|zip}) \& (\text{token|first})\)
  - Guarantee later rule is less restrictive – If not how are there examples left to cover?
Learn-One-Rule

- Learn conjunction that maximizes RR
- General-to-specific beam search
  - Keep adding/intersecting (attribute, method) pairs
    - Until can’t improve RR
    - Must satisfy minimum PC
Experiments

HFM = \{(\text{token, make}) \cap \{\text{token, year}\} \cap \{\text{token, trim}\}\)
\bigcup \{(\text{1st letter, make}) \cap \{\text{1st letter, year}\} \cap \{\text{1st letter, trim}\}\)
\bigcup \{\text{synonym, trim}\}

BSL = \{(\text{token, model}) \cap \{\text{token, year}\} \cap \{\text{token, trim}\}\)
\bigcup \{\text{token, model}\} \cap \{\text{token, year}\} \cap \{\text{synonym, trim}\}\)

<table>
<thead>
<tr>
<th>Cars</th>
<th>RR</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>HFM</td>
<td>47.92</td>
<td>99.97</td>
</tr>
<tr>
<td>BSL</td>
<td>99.86</td>
<td>99.92</td>
</tr>
<tr>
<td>BSL (10%)</td>
<td>99.87</td>
<td>99.88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Census</th>
<th>RR</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best 5 Winkler</td>
<td>99.52</td>
<td>99.16</td>
</tr>
<tr>
<td>Adaptive</td>
<td>99.9</td>
<td>92.7</td>
</tr>
<tr>
<td>BSL</td>
<td>98.12</td>
<td>99.85</td>
</tr>
<tr>
<td>BSL (10%)</td>
<td>99.50</td>
<td>99.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurants</th>
<th>RR</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marlin</td>
<td>55.35</td>
<td>100.00</td>
</tr>
<tr>
<td>BSL</td>
<td>99.26</td>
<td>98.16</td>
</tr>
<tr>
<td>BSL (10%)</td>
<td>99.57</td>
<td>93.48</td>
</tr>
</tbody>
</table>
Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Attr,</th>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canopie Ad-hoc</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Bi-Gram Ad-hoc</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>BSL Learn</td>
<td>SCA</td>
<td></td>
</tr>
</tbody>
</table>

**Tradeoffs:** Learning vs. Non
- Need to label (but already labeled for RL!), but get well justified, productive blocking

**Choice:** Choose a learning method!
- Maybe use canopies within a learning method!
Conclusions

- Automatic Blocking Schemes using Machine Learning
  - Not created by hand
    - cheaper
    - easily justified
  - Better than non-experts ad-hoc and comparable to domain expert’s rules
    - Nice reductions – scalable record linkage
    - High coverage – don’t hinder record linkage