A Search Engine for Natural Language Applications
AND
Relational Web Search: A Preview

Michael J. Cafarella
(joint work with Michele Banko, Doug Downey, Oren Etzioni, Stephen Soderland)

CSE454
University of Washington
November 15, 2005

Outline

- NLP applications and the web corpus
- The Bindings Engine
  - Query language
  - Neighbor Index
- Experiments
  - >300x performance gain
  - ~4x space penalty
- Novel applications
- Relational Web Search: Preview

NLP Applications: An Example

KnowItAll (Etzioni et al, WWW04, AIJ05): unsupervised web-scale info extraction
- Generate candidate fact-extractions from web
- Use extraction frequencies to assess probability that a candidate extraction is true

Simplified algorithm:
- Search web for various hypernym-phrases (e.g., "cities such as X" where X is a noun)
- Count hits for each unique X
- Use hit counts as inputs to trained classifier
- Sort Xs by classifier probability, then threshold

Pointwise Mutual Information
Information Retrieval (PMI-IR)

Turney uses PMI-IR to find semantic orientation (ACL02)
- Estimate co-occur probs using hitcounts

\[
SOM(phrase) = \log \left( \frac{hit(phrase \ AND \ "excellent") \ AND \ "poor" \ hits \ phrase}}{hit(phrase \ AND \ "good") \ AND \ "poor" \ hits \ phrase}} \right)
\]

- Anything more than NEAR (e.g., breaking at sentences) requires original text
- If there are 10k phrases and 14 reference words, we need 140k search queries

Search engine inefficiencies

- Search engines ill-suited to the task
  - System must download and parse many docs for each query; many end up as useless
  - Worst of all, each downloaded document likely requires a disk seek
  - Certain apps require #queries = #candidates x #phrases
- Problems common to many NLP apps
Bindings Engine

- Bindings Engine (BE) is search engine where:
  - No downloads during query processing
  - Disk seeks constant in corpus size
  - \#queries = \#phrases
- BE’s approach:
  - "Variabilized" search query language
  - Pre-processes all documents before query-time
  - Integrates variable/type data with inverted index, minimizing query seeks

Query language

- cities such as \(<\text{NounPhrase}>\>
- President Bush \(<\text{Verb}>\>
- \(<\text{NounPhrase}>\) is the capital of \(<\text{NounPhrase}>\)
- reach me at \(<\text{phone-number}>\)

- Any sequence of concrete terms and typed variables
- (some limitations posed by current index)
- NEAR is insufficient
- Paper also discusses functions, which modify variable bindings (e.g., “\text{head}(\text{NounPhrase})”)
**Neighbor Index**

- At each position in the index, store “neighbor text” that might be useful
- Let’s index `<NounPhrase>` and `<Adj-Term>`

“In love cities such as Seattle.”

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdjT: “such”</td>
<td>AdjT: “Seattle”</td>
</tr>
<tr>
<td>NP: “cities”</td>
<td>NP: “Seattle”</td>
</tr>
</tbody>
</table>

**Neighbor Index**

Query: “cities such as `<NounPhrase>`”

“cities such as `<NounPhrase>`”

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdjT: “such”</td>
<td>AdjT: “Seattle”</td>
</tr>
<tr>
<td>NP: “cities”</td>
<td>NP: “Seattle”</td>
</tr>
</tbody>
</table>

**Asymptotic analysis**

- `k` concrete terms in query
- `B` bindings found for query
- `N` documents in corpus
- `T` indexed types in corpus

<table>
<thead>
<tr>
<th></th>
<th>Query Time</th>
<th>Index Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE</td>
<td><code>O(k)</code></td>
<td><code>O(N * T)</code></td>
</tr>
<tr>
<td>Std Model</td>
<td><code>O(k + B)</code></td>
<td><code>O(N)</code></td>
</tr>
</tbody>
</table>

*`B` and `N` scale together; `k` often small; `T` often exclusive*
Experimental details

- 20 machine cluster, 50m page corpus
- BE types: `<NounPhrase> & <Adj-term>`
- Experiment 1: 150 assorted queries with 1 variable, 2-3 concrete terms. BE vs Nutch-based “standard implementation”
- Experiment 2: KnowItAll system test

Experiment 1: Processing speed

Experiment 2: KnowItAll on BE

<table>
<thead>
<tr>
<th>Num Extractions</th>
<th>Std Imp/Google</th>
<th>BE</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>5,976s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50k</td>
<td>29,880s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150k</td>
<td>89,641s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Costs: Index size**

**Costs: Index construction time**
Novel applications: Interactive Information Extraction

- BE is fast enough to allow new interaction models
- KnowItAll is a batch process
- KnowItNow is interactive; approximates core of KnowItAll using a few BE queries

Relational Web Search

- Consider that last slide for a sec:
- It looks a lot like a database table
- Can web-style search generate structured output, instead of just a list of docs?

Relational Web Search (2)

- Modern search works treats docs as bags of words; no internal structure
- Instead, we use the corpus to assemble a huge entity-relation graph

Relational Web Search (3)

- We automatically extract it from the doc; it’s called the extraction graph
- All searches are done over the E.G., not the original document set
- Lets us perform various queries:
  - Qualified-list (“west coast liberal arts colleges”)
  - Unnamed-item (“tallest inactive volcano in Africa”)
  - Relationship (describe relation between Bill Clinton and Justice Ginsberg)
  - Tabular (database table of cities and capitals)
The Extraction Graph
- "Is-A" edges come from KnowItAll
- Predicate edges found by looking for certain linguistic patterns
- From 90m docs, we have:
  - 652m object-relation-object triples
  - 227m nodes
  - 544m edges
  - ~71.7% of "Is-A" correct
  - ~44% of predicate edges correct

Searching
- Perform "spreading activation" search on graph
- Each edge has a "decay factor" that retards spread

Results: qualified-list

Results: tabular query

Questions?
- Thanks
- We're hiring!
- Comments to mjc@cs.washington.edu