Indexing, Searching, and Ranking in Entity-Relationship Networks with Associated Text

Soumen Chakrabarti
IIT Bombay
http://www.cse.iitb.ac.in/~soumen
(In fewer words)

Ranking and Indexing for Semantic Search

Soumen Chakrabarti
IIT Bombay
http://www.cse.iitb.ac.in/~soumen
Working notion of “semantic search”

- Extractors and annotators associate **structured knowledge** with strings
- Neither complete nor perfect
- No complete schema (despite CYC and WordNet)
- Noisy structure must coexist with source text
- Must exploit structured info and uninterpreted strings in conjunction

**Figure:** Artist’s impression of “semantic search”
Annotated corpus and query examples

Name a **physicist** who searched for intelligent life in the cosmos
→ type=**physicist** NEAR “cosmos”...

**Where** was Sagan born?
→ type=**region** NEAR “Sagan”

**When** was Sagan born?
→ type=**time**
pattern=**isDDDD** NEAR “Sagan” “born”

Born in **New York** in **1934** , Sagan was a noted **astronomer** whose lifelong passion was searching for intelligent life in the cosmos.
Search in entity-relationship graphs
Talk outline

Ranking problems: what is “NEAR”? 
- Typed proximity search in text + is-a graphs 
- Proximity search in typed graphs 
- Discovering hidden favorite communities

Indexing and query processing problems 
- Typed proximity search in text + is-a graphs 
- Dynamic (query-sensitive) Pageranking on typed graphs
Learning proximity scores of token spans

```plaintext
type=person NEAR "television" "invent*"
```

- Rarity of selectors
- Distance from candidate position to selectors
- Many occurrences of one selector (closest)
- Combining scores from many selectors (sum)
Making up a feature vector $x$ for position 0

- Limit to $\pm W$ window
- $x(-4) = 0$;
  $x(-1) = \text{IDF}(\text{invent}^*)$;
  $x(-6) = \text{IDF}(\text{television})$
- $\text{IDF}(w) = \frac{\text{numDocs}}{\text{numDocsWith}(w)}$, or perhaps
  $\text{IDF}(w) = \log(1 + \frac{\text{numDocs}}{\text{numDocsWith}(w)})$
- Other features, e.g., is selector noun? candidate has digits?
- If in doubt, throw everything into the kitchen sink
The model vector $\beta$

- Score of candidate position is $\beta x$
- $\beta(j)$ is the value of the decay function at offset $j$
- TREC gives us correct $i$ and incorrect $j$ token spans
- $i \prec j$ means we want $\beta x_i + \text{margin} \leq \beta x_j$
- Want $\beta$ to be smooth with $\beta(-W - 1) = \beta(W + 1) = 0$

$$
\min_{\beta} \sum_{j=-W}^{W+1} (\beta_{j-1} - \beta_j)^2 + B \sum_{i \prec j} \text{SmoothLoss}(\beta x_i + 1 - \beta x_j)
$$

$$
\min_{\beta} \sum_{j=-W}^{W+1} (\beta_{j-1} - \beta_j)^2 + B \sum_{i \prec j} \log(1 + \exp(\beta x_i + 1 - \beta x_j))
$$
The model vector $\beta$

- Score of candidate position is $\beta x$
- $\beta(j)$ is the value of the decay function at offset $j$
- TREC gives us correct $i$ and incorrect $j$ token spans
- $i \prec j$ means we want $\beta x_i + \text{margin} \leq \beta x_j$
- Want $\beta$ to be smooth with $\beta(-W - 1) = \beta(W + 1) = 0$

\[
\begin{align*}
\min_{\beta} & \sum_{j=-W}^{W+1} (\beta_{j-1} - \beta_j)^2 + B \sum_{i \prec j} \text{SmoothLoss} (\beta x_i + 1 - \beta x_j) \\
\min_{\beta} & \sum_{j=-W}^{W+1} (\beta_{j-1} - \beta_j)^2 + B \sum_{i \prec j} \log \left( 1 + \exp (\beta x_i + 1 - \beta x_j) \right)
\end{align*}
\]
\( \beta \) fit to TREC QA

- Only positive offsets shown
- Unexpected decay shape!
- Smooth function slightly better than rough function
- Improves beyond flat scoring function with only IDF

Next: Extending beyond chain graphs

<table>
<thead>
<tr>
<th>Train</th>
<th>Test</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR 2001</td>
<td>2000</td>
<td>0.16</td>
</tr>
<tr>
<td>2001</td>
<td>2000</td>
<td>0.29</td>
</tr>
</tbody>
</table>

'2001' = 'TREC 2001'; 'MRR' = Mean reciprocal rank
Nodes have entity types: Person, Paper, Email, Company

Edges have relation types: wrote, sent, cited, in-reply-to

Edge \( e \) has type \( t(e) \in \{1, \ldots, T\} \)

Edge \((u, v)\) of type \( t(u, v) \) has weight \( \beta(t(u, v)) \) and conductance \( C(v, u) \)
Conductance, teleport, Pagerank

- Create dummy node $d$
- Create edges $(d, u)$ and $(u, d)$ for each $u$
- Let $0 < \alpha < 1$ be the teleport probability
- Let $r_v$ be the probability of teleport to $v$

$$C_{\alpha, \beta}(v, u) = \begin{cases} 
\alpha \frac{\beta(t(u,v))}{\sum_{(u,w) \in E} \beta(t(u,w))}, & u \neq d, v \neq d \\
1 - \alpha, & u \neq d, u \in V_o, v = d \\
1, & u \neq d, u \in V \setminus V_o, v = d \\
r_v, & u = d, v \neq d \\
0, & \text{otherwise}
\end{cases}$$

$C$ is a function of $\alpha$ and $\beta$
Dodging complicated constraints

\[
\min_{0 < \alpha < 1, \beta \geq 0, p} \text{ModelCost}(\beta) + B \sum_{i < j} \text{SmoothLoss}(p_u - p_v)
\]

subject to \( p = C\{\alpha, \beta\} \ p \)

- Both \( C \) and \( p \) are variables \( \Rightarrow \) complicated constraints
- Following power iterations, approximate \( p \approx C^H p_0 \) where \( H \) is a (possibly adaptive) horizon and \( p_0 \) is the initial Pagerank vector (say uniform)
- Also, \( C\{\alpha, \beta\} \) does not change if all \( \beta \) are scaled

\[
\min_{0 < \alpha < 1, \beta \geq 1} \text{ModelCost}(\beta) + B \sum_{i < j} \text{SmoothLoss}\left( (C^H p_0)_u - (C^H p_0)_v \right)
\]
ModelCost and SmoothLoss

Parsimonious model: all $\beta(t)$s equal

$$\text{ModelCost}(\beta) = \sum_{t \neq t'} \left( \beta(t) - \beta(t') \right)^2$$

If $\beta$ and $\kappa \beta$ (some multiple $\kappa > 1$) are both solutions, optimizer should prefer $\beta$

Use $\text{SmoothLoss}(z) = \text{huber}(z)$, where

$$\text{huber}(z) = \begin{cases} 
0, & z < 0 \\
\frac{z^2}{2W}, & z \in (0, W] \\
\frac{z - W}{2}, & z > W 
\end{cases}$$

Can compute approximate $\nabla_{\beta}, \partial / \partial \alpha$, and use gradient descent
Is SmoothLoss approximation good?

- Hinge and Huber essentially identical
- Empirically, wrt $\beta(t)$, true and Huber error have same minima
- Wrts $\alpha$ Huber has spurious minima, but basic grid search adequate
Learning rate and robustness

- 20000-node, 120000-edge graph
- 100 pairwise training preferences enough to cut down test error to 11 out of 2000
- 20% random reversal of train pairs leads to 5% increase in test error
- Model cost reduces as noise increases — makes sense
Accuracy of estimating $\beta$ and $\alpha$

- Assign hidden $\beta$, $\alpha$
- Compute weighted Pagerank and sample
- See if algorithm can recover hidden weights
- Upward (downward) pressure on small (large) $\beta$ thanks to
  \[\sum_{t, t'} (\beta(t) - \beta(t'))^2\]
  regularizer
- Large patches of $\beta$ lead to same Pagerank ordering
- $\alpha$ sensitive to $B$ setting
Learning a hidden favored community

- Unlike the random surfer, humans are very selective about following links.
- Links in some communities matter, other links do not.
- Also preferential teleport.
- $\prec$ expresses this indirectly, as in $u_1 \prec v_1$, $u_2 \prec v_2$ etc.
- Goal is to generalize and find the boundaries of favored community.
- Unlike global $\beta(t)$ here info is local; does not extend beyond the “teleport radius.”
A constrained flow formulation

- Directly estimate $p_{uv}$ instead of $C(v, u) = \Pr(v|u)$
- $q_{uv}$ is a “parsimonious” reference flow (unweighted Pagerank)

\[
\min_{\{0 \leq p_{uv} \leq 1\}} \{0 \leq s_{uv} : u \prec v\} \sum_{(u,v) \in E'} p_{uv} \log \frac{p_{uv}}{q_{uv}} + B \sum_{u \prec v} s_{uv} \quad \text{(SoftObj)}
\]

subject to
\[
\sum_{(u,v) \in E'} p_{uv} = 1 \quad \text{(Total)}
\]

\[
\forall v \in V' \sum_{(u,v) \in E'} p_{uv} = \sum_{(v,w) \in E'} p_{vw} \quad \text{(Balance)}
\]

\[
\forall v \in V_0 \quad (1 - \alpha) \sum_{(v,w) \in E} p_{vw} = \alpha p_{vd} \quad \text{(Teleport)}
\]

\[
\forall u \prec v \quad (1 + \epsilon) \sum_{(w,u) \in E'} p_{wu} \leq s_{uv} + \sum_{(w,v) \in E'} p_{wv} \quad \text{(SoftPref)}
\]
Experimental results

- We solve the dual via cutting-plane approach
- Time is linear in $|≺|$, $|V| + |E|$.
- Generalization improves with margin $1 + \epsilon$.
Talk outline

Ranking problems: what is “NEAR”?  
- Typed proximity search in text + is-a graphs  
- Proximity search in typed graphs  
- Discovering hidden favorite communities

Indexing and query processing problems  
- Typed proximity search in text + is-a graphs  
- Dynamic (query-sensitive) Pageranking on typed graphs
Indexing for is-a proximity search

“Which scientist studied whales?” →

type=scientist NEAR study|studied whale*

- Open-domain type hierarchies very large: 15000 internal and 80000 leaf types in WordNet (full set A)
- Runtime type expansion too expensive: even WordNet knows 650 scientist, 860 cities, ...

Pre-generalize

- Index a subset $R \subset A$
- Query atype $a \notin R$, want $k$ answers
- Probe index with $g$, ask for $k' > k$

...whales were studied by Cousteau...
Cost models

- How much space saved by indexing $R$ instead of $A$? (Cannot afford to try out many $R$s, need quick estimate)
- What is the average query time bloat owing to $a \rightarrow g$ pre-generalize and post-filter?

Post-filter

- Fetch $k'$ high-scoring spans $w$ for $g$
- Check if $w$ is-a $a$ as well (using forward and reachability index); if not, discard
- If fewer than $k$ survive, restart with larger $k'$ (expensive!)
Cost models

- How much space saved by indexing $R$ instead of $A$? (Cannot afford to try out many $R$s, need quick estimate)
- What is the average query time bloat owing to $a \rightarrow g$ pre-generalize and post-filter?

Post-filter

- Fetch $k'$ high-scoring spans $w$ for $g$
- Check if $w$ is-a $a$ as well (using forward and reachability index); if not, discard
- If fewer than $k$ survive, restart with larger $k'$ (expensive!)

...whales were studied by Cousteau...
Index space estimate

- Let corpusCount\(a\) be the count of tokens \(w\) in the corpus such that \(w\) is-a \(a\)
- One posting entry for each count of each type \(a\)
- Therefore our space estimate is (proportional to) \(\sum_{a \in R} \text{corpusCount}(a)\)
- Surprisingly accurate despite index compression
Characterizing a query workload

\[ \tilde{Pr}(a) = \frac{\text{queryLogCount}(a) + \lambda}{\sum_{a' \in A} (\text{queryLogCount}(a) + \lambda)} \]

- Heavy-tailed type distribution in queries
- Many test types never seen in training types and vice versa
  - \( \lambda = 0 \) would give these types zero probability
  - Danger of allocating no \( g \) close to these as
  - Build multinomial model over \( a \) with positive \( \lambda \)
  - Cross-validate likelihood of held-out test log
Query time bloat estimate

- $t_{\text{scan}}$ time to scan one candidate position while merging postings
- $t_{\text{filter}}$ time to check if $w$ is-a $a$
- If $R = A$ (all types indexed), query takes time roughly $t_{\text{scan}} \cdot \text{corpusCount}(a)$
- If $a \not\in R$, the price paid for generalization to $g$ consists of
  - Longer scans: $t_{\text{scan}} \cdot \text{corpusCount}(g)$
  - Post-filtering $k'$ responses: $k' \cdot t_{\text{filter}}$

\[
\text{expected bloat} = \sum_{a \in A} \tilde{\Pr}(a) \frac{t_{\text{scan}} \cdot \text{corpusCount}(g) + k' \cdot t_{\text{filter}}}{t_{\text{scan}} \cdot \text{corpusCount}(a)}
\]

Now we have a greedy cost-benefit analysis of every type $a$
Result of greedy knapsack

- Estimated query bloat reasonably accurate
- With only 520 MB index, only 1.9 average bloat
- Space comparable to inverted index on stems

<table>
<thead>
<tr>
<th>Corpus/Index</th>
<th>GBytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original corpus</td>
<td>5.72</td>
</tr>
<tr>
<td>Gzipped corpus</td>
<td>1.33</td>
</tr>
<tr>
<td>Stem index</td>
<td>0.91</td>
</tr>
<tr>
<td>Full type A index</td>
<td>4.30</td>
</tr>
<tr>
<td>Type subset R index</td>
<td>0.52</td>
</tr>
<tr>
<td>Query Bloat</td>
<td>1.90</td>
</tr>
<tr>
<td>Reachability index</td>
<td>0.01</td>
</tr>
<tr>
<td>Forward index</td>
<td>1.16</td>
</tr>
</tbody>
</table>
Searching a TypedWordGraph

- Attach query words $W$ to preloaded entities $N$
- Set teleport $r > 0$ only for word nodes
- Compute personalized Pagerank vector (PPV) $p_r$ — slow!
Options to date

Query-time Pagerank

- For $\sim 75000$ nodes, $\sim 200000$ edges, $\sim 11$ sec/query
- Impractical except for very small graphs

Combine per-word PPVs

- Proposed by ObjectRank
- For $\sim 75000$ nodes, $\sim 175000$ words, $\sim 526$ CPU-hours
- Full PPV index has size 102 GB
- Compare with text index: 56 MB
- Truncating down to 56 MB leads to serious loss of accuracy

RAG=Relative average goodness, Prec=Precision, KTau=Kendall’s Tau
HubRank query execution

- Input: query words $W$, abandon/trim threshold $\delta$
- Max priority queue, priority of node $u$ is estimate of path conductance from $d$ to $u$
- Grow active set $A$

$A$ is bordered with

- **Blockers**: Hub nodes with indexed (approx) PPVs
- **Losers $\ell$**: Conductance from $d$ to $\ell$ is too small to matter

- Load trimmed PPV for blockers, trivial PPV for losers
- Iteratively compute PPVs of all active nodes including $d$
- Sort PPV of $d$ and return results
HubRank results

- Indexing 10 CPU-hours (vs. 526, ObjectRank)
- 63 MB index (vs. 102 GB)
- $\delta$-trim cuts CPU, fill
- Negligible accuracy loss
- Query in 250 ms (vs. 11 s)
Conclusion

- Searching typed entity-relationship graphs
- Perhaps attached to mentions in unstructured text
- Many interesting challenges, surprisingly unexplored
- **Architecture for flexible space-time-accuracy tradeoffs**
- **Ranking**
  - Far from vector-space territory, no guidance
  - Learning linear proximity functions
  - Learning edge conductance parameters
- **Indexing and query processing**
  - Combining large is-a graphs with linear proximity
  - Dynamic personalized Pageranking
  - Anytime preprocessing, anytime query processing