Topics du jour

- Finish up web ranking
- Peer-to-peer search
- Search deployment models
  - Service vs. software
  - External vs. internal-facing search software
- Review of search topics

Tag/position heuristics

- Increase weights of terms in titles
- Increase weights of terms in `<h>` tags
- Increase weights of terms near the beginning of the doc, its chapters and sections - key phrases

Anchor text

The text in the vicinity of a hyperlink is descriptive of the page it points to.
Two uses of anchor text

- When indexing a page, also index the anchor text of links pointing to it.
- To weight links in the hubs/authorities algorithm from the last lecture.
- Anchor text usually taken to be a window of 6-8 words around a link anchor.

Indexing anchor text

- When indexing a document $D$, include anchor text from links pointing to $D$.

Indexing anchor text

- Can sometimes have unexpected side effects - e.g., evil empire.
- Can index anchor text with less weight.

Weighting links

- In hub/authority link analysis, can match anchor text to query, then weight link.

$$h(x) \leftarrow \sum_x a(y)$$

$$a(x) \leftarrow \sum_y h(y)$$

$$h(x) = \sum_x w(x,y) \cdot a(y)$$

$$a(x) = \sum_y w(x,y) \cdot h(y)$$
Weighting links

• What is \( w(x,y) \)?
• Should increase with the number of query terms in anchor text.
  – Say 1+ number of query terms.

Weighted hub/authority computation

• Recall basic algorithm:
  – Iteratively update all \( h(x), a(x) \);
  – After iteration, output pages with highest \( h() \) scores as top hubs; highest \( a() \) scores as top authorities.
• Now use weights in iteration.
• Raises scores of pages with “heavy” links.

Web sites, not pages

• Lots of pages in a site give varying aspects of information on the same topic.

Link neighborhoods

• Links on a page tend to point to the same topics as neighboring links.
  – Break pages down into pagelets (say separate by tags) and compute a hub/authority score for each pagelet.
Link neighborhoods

Ron Fagin’s links
• Logic links
  • Moshe Vardi’s logic page
  • International logic symposium
  • Paper on modal logic
• …
• My favorite football team
  • The 49ers
  • Why the Raiders suck
  • Steve’s homepage
  • The NFL homepage

Web vs. hypertext search

• The WWW is full of free-spirited opinion, annotation, authority conferral
• Most other forms of hypertext are far more structured
  – enterprise intranets are regimented and templated
  – very little free-form community formation
  – web-derived link ranking doesn’t quite work

Link analysis/search - summary

• Powerful new ideas
  – derived from sociology of web content creation
• Supplemented by other heuristics
• Less useful in intranets
• Challenges from dynamic html
• Application servers and web content management systems

Behavior-based ranking

• For each query $Q$, keep track of which docs in the results are clicked on
• On subsequent requests for $Q$, re-order docs in results based on click-throughs
• First due to DirectHit $\rightarrow$ AskJeeves
Query-doc popularity matrix $\mathbf{B}$

$$B_{qj} = \text{number of times doc } j \text{ clicked-through on query } q$$

When query $q$ issued again, order docs by $B_{qj}$ values.

Issues to consider

- Weighing/combining text- and click-based scores.
- What identifies a query?
  - Ferrari Mondial
  - Ferrari Mondial
  - Ferrari mondial
  - ferrari mondial
  - “Ferrari Mondial”
- Can use heuristics, but search parsing slowed

Vector space implementation

- Maintain a term-doc popularity matrix $\mathbf{C}$
  - as opposed to query-doc popularity
  - initialized to all zeros
- Each column represents a doc $j$
  - If doc $j$ clicked on query $q$, update $C_j \leftarrow C_j + \varepsilon q$
    (here $q$ is viewed as a vector).
- On a query $q'$, compute its cosine proximity to $C_j$ for all $j$.
- Combine this with the regular text score.

Issues

- Normalization of $C_j$ after updating.
- Boolean operators
- Why did the user click on the doc?
- Updating - live or batch?
- All votes count the same.
  - More on this in recommendation systems.
### Variants

- Time spent viewing page
  - Difficult session management
  - Inconclusive modeling so far.
- Does user back out of page?
- Does user stop searching?
- Does user transact?

### Peer-to-peer (P2P) search

- No central index
- Each node in a network builds and maintains own index
- Each node has “servent” software
  - On booting, servent pings ~4 other hosts
  - Connects to those that respond
  - Initiates, propagates and serves requests

### Which hosts to connect to?

- The ones you connected to last time
- Random hosts you know of
- Request suggestions from central (or hierarchical) nameservers

- All govern system’s shape and efficiency

### Serving P2P search requests

- Send your request to your neighbors
- They send it to their neighbors
  - decrement “time to live” for query
  - query dies when ttl = 0
- Send search matches back along requesting path
The promise of P2P

- Fresh content
  - no waiting for the next weekly indexing
- Dynamic content
  - results could be assembled from a database or other repository
  - live pricing/inventory information

P2P search issues

- Internet:
  - Query interpretation up to server
    - spamming potential
  - No co-ordination in network
    - fragmentation
- Enterprises:
  - security and access control
  - administration
  - distributed replication and caching

Search deployment

Intranet vs. extranet

Search deployment models

- As a service
  - public, e.g., web search
  - access-protected, e.g., proprietary newsfeeds and content
- As software
  - Outward-facing (Walmart, CDNow …)
  - Inward-facing within an enterprise
## Service deployment issues

<table>
<thead>
<tr>
<th>+ Ease of maintenance</th>
<th>+ Can tune to platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ software as well as indices</td>
<td></td>
</tr>
</tbody>
</table>

- To date, not much proprietary content
  - owners of valuable content don’t hand over custody

## Software deployment

- Inward vs. outward-facing
  - very different characteristics
    - corpus sizes
    - query rates
    - languages and localization
    - security
    - content management

## Outward-facing search software

- Relatively small corpora
  - typically under 1GB
- Sporadic query rates, high peak loads
- Fairly dynamic corpus
  - item prices in a catalog

## Typical eCommerce search setup

- Product database (RDBMS) w/product info
  - prices, descriptions
- Search engine - spiders DB, indexes structured+unstructured product info.
- Application server - content assembly, personalization + Web server
- Back-end inventory RDBMS
  - to complete the transaction.
Scaling search servers

Broker

Broker

Search servers

Partitioning the index

- By documents
  - Each server has a subset of the docs
  - Each has its own dictionary
  - Query sent out to “all” servers
- Broker ensures load-balancing, failover

Partitioning the index

- By terms
  - Each server has a subset of the lexicon
  - Query sent to server(s) with the query term(s)
  - Partition alphabetically → easy query dispatch
  - Partition by hashing → uniform spread
- Query optimization is hard
- Works best when query terms are uniformly spread across servers

Inward-facing search software

- Search within an intranet
- Enterprise portals

“Enterprise” doesn’t have to be a (for profit) company - government, academe, … any collaborative group with proprietary information.
Issues in enterprise search

- Scale - lots of docs, geographically distributed over non-uniform WAN
- Multiple languages and character sets
  - Locale modules for stemming, thesauri
- Multiple document repositories
  - Lotus, Exchange, Documentum, Filenet …
  - Materialized views of compound documents
- Multiple formats - pdf, MS office, …
  - multiple MIME-type attachments

Security and results lists

- Each doc has access permissions for groups
- User authenticated for membership in certain groups; can change with time
- Results of a search should only contain docs the user can view
  - Not sufficient to show a doc in results, then deny user attempting to access it
- Compound docs made up of pieces
  - each piece has own ACL’s

Bottom line

- Enterprise search - inside and outside - are quite different
- Each different from public web search service
- Inside enterprise search the most fragile
  - tremendous diversity
  - flexible, hard-to-administer software vs. expensive customization

Review of search topics
Inverted index

- Dictionary of terms
- Each term points to a series of postings entries
  - Postings for a term point to docs containing that term

Term storage in dictionary

- Store pointers to every \( k \)th on term string.
- Need to store term lengths (1 extra byte)

<table>
<thead>
<tr>
<th>Term</th>
<th>Postings ptr.</th>
<th>Term ptr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>33,47,154,159,202 ...</td>
<td></td>
</tr>
<tr>
<td>Suffices</td>
<td>33,14,107,5,43 ...</td>
<td></td>
</tr>
</tbody>
</table>

Postings file entry

- Store list of docs containing a term in increasing order of doc id.
  - \( \text{Brutus} \): 33,47,154,159,202 ...
- Suffices to store gaps.
  - 33,14,107,5,43 ...
- Gaps encoded with far fewer than 20 bits, using \( \gamma \) codes.

Total postings size

- Estimate using crude Zipf law analysis
  - Most frequent term occurs in \( n \) docs
  - Second most frequent term in \( n/2 \) docs
  - \( k \)th most frequent term in \( n/k \) docs, etc.
    - \( n/k \) gaps of \( k \) each - use \( \sim 2 \log_2 k \) bits for each gap using \( \gamma \) codes.
What gets indexed?

- Stemming - Porter’s.
- Case folding.
- Thesauri and soundex
- Spell correction

Query optimization

- Consider a query that is an AND of \( t \) terms.
- The idea: for each of the \( t \) terms, get its term-doc incidence from the postings, then AND together.
- Process in order of increasing freq:
  - start with smallest set, then keep cutting further.

Skip pointers

2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, ...

- At query time:
- As we walk the current candidate list, concurrently walk inverted file entry - can skip ahead
  - (e.g., 8, 21).
- Skip size: recommend about \( \sqrt{\text{list length}} \)

Wild-card queries

- \( \text{mon}^* \): find all docs containing any word beginning “mon”.
- Solution: index all \( k \)-grams occurring in any doc (any sequence of \( k \) chars).
- Query \( \text{mon}^* \) can now be run as
  - \( Sm \text{ AND mo AND on} \)
    - But we’d get a match on moon.
- Must post-filter these results against query.
**Phrase search**

- Search for “to be or not to be”
- No longer suffices to store only `<term:docs>` entries.
- Instead store, for each `term`, entries
  - `<number of docs containing term>`;
  - `doc1`: position1, position2 … ;
  - `doc2`: position1, position2 … ;
  - etc.

**Precision and recall**

- **Precision**: fraction of retrieved docs that are relevant
- **Recall**: fraction of relevant docs that are retrieved
- Both can be measured as functions of the number of docs retrieved

**Index construction**

- Parse and build postings entries one doc at a time
- To now turn this into a term-wise view, must sort postings entries by term (then by doc within each term)
- Block of postings records; can “easily” fit a couple into memory.
- Sort within blocks first, then merge.

**Fully dynamic updates**

- Inserting a (variable-length) record
  - a typical postings entry
- Maintain a pool of (say) 64KB *chunks*
- Chunk header maintains metadata on records in chunk, and its free space
Doc as vector

- Each doc $j$ can now be viewed as a vector of $tf \times idf$ values, one component for each term.
- So we have a vector space
  - terms are axes
  - docs live in this space
  - even with stemming, may have 10000+ dimensions

$tf \times idf$

$$w_{ij} = tf_{ij} \times \log\left(\frac{n}{n_i}\right)$$

$tf_{ij}$ = frequency of term $i$ in document $j$
$n = $ total number of documents
$n_i = $ the number of documents that contain term $i$
$idf_i = \log\left(\frac{n}{n_i}\right) = $ inverse document frequency of term $i$

Cosine similarity

- Distance between vectors $D1,D2$ captured by the cosine of the angle $x$ between them.
- Note - this is similarity, not distance.

The point of using vector spaces

- **Key:** A user’s query can be viewed as a (very) short document.
- Query becomes a vector in the same space as the docs.
- Can measure each doc’s proximity to it.
- Natural measure of scores/ranking - no longer Boolean.
Search using vector spaces

- Computing individual cosines
- Speeding up computations
  - Avoiding computing cosines to all docs
  - Dimensionality reduction
    - Random projection
    - LSI

Bayesian nets for text retrieval

Semi-structured search

- Structured search - search by restricting on attribute values, as in databases.
- Unstructured search - search in unstructured files, as in text.
- Semi-structured search: combine both.

Link analysis

- Two basic approaches
  - Universal, query-independent ordering on all web pages (based on link analysis)
    - Of two pages meeting a (text) query, one will always win over the other, regardless of the query
  - Query-specific ordering on web pages
    - Of two pages meeting a query, the relative ordering may vary from query to query
Ergodic Markov chains

- For any ergodic Markov chain, there is a unique long-term visit rate for each state.
  - Steady-state distribution.
- Over a long time-period, we visit each state in proportion to this rate.
- It doesn’t matter where we start.

Resources

- MIR 9.