Announcements (Thu. Dec. 6)

- Homework #4 sample solution will be emailed by this weekend
- Course evaluation is online and due by tomorrow
- Project demo of Mobile Pay today from Michael, Kevin, Derek
- Final exam 2-5pm Dec. 12
  - Open book, open notes; focus on the second half
  - Extended office hours
    - Tue.: 12pm-2pm; Wed.: 10-11:30am, 1-2pm

Outline

- Keyword search
  - A single search may return many pages
  - A user will not look at all result pages
  - Complete result may be unnecessary

Keywords × documents

<table>
<thead>
<tr>
<th>All documents</th>
<th>Document 1</th>
<th>Document 2</th>
<th>Document 3</th>
<th>Document 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>All keywords</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&quot;dog&quot;</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&quot;database&quot;</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&quot;search&quot;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Inverted lists: store the matrix by rows
- Signature files: store the matrix by columns

Inverted lists

- Store the matrix by rows
- For each keyword, store an inverted list
  - (keyword, doc-id-list)
  - ("database", {3, 7, 142, 857, ...})
  - ("search", {3, 9, 192, 512, ...})
  - It helps to sort doc-id-list (why?)
- Vocabulary index on keywords
  - B+-tree, hash-based, or trie (later)
- How large is an inverted list index?

Using inverted lists

- Documents containing "database"
  - Use the vocabulary index to find the inverted list for "database"
  - Return documents in the inverted list
- Documents containing "database" AND "search"
  - Return documents in the intersection of the two inverted lists
- OR? NOT?
  - Union and difference, respectively
What are “all” the keywords?

- All sequences of letters (up to a given length)?
  - … that actually appear in documents!
- All words in English?
- Plus all phrases?
  - Alternative: approximate phrase search by proximity
- Minus all stop words
  - They appear in nearly every document, e.g., a, of, the, it
  - Not useful in search
- Combine words with common stems
  - Example: database, databases
  - They can be treated as the same for the purpose of search

Frequency and proximity

- Frequency
  - \( \text{keyword}, \{ \text{doc-id}, \text{number-of-occurrences} \}, \text{...} \)\)
- Proximity (and frequency)
  - \( \text{keyword}, \{ \text{doc-id}, \{ \text{position-of-occurrence}_1, \text{position-of-occurrence}_2, \text{...} \} \}, \text{...} \)\)

Signature files

- Store the matrix by columns and compress them
- For each document, store a \( w \)-bit signature
- Each word is hashed into a \( w \)-bit value, with only \( s < w \) bits turned on
- Signature is computed by taking the bit-wise OR of the hash values of all words on the document

\[
\text{hash(“database”) = 0110} \quad \text{hash(“dog”) = 1100} \quad \text{hash(“cat”) = 0010}
\]

- Some false positives; no false negatives

Bit-sliced signature files

- Motivation
  - To check if a document contains a word, we only need to check the bits that are set in the word’s hash value
  - So why bother retrieving all \( w \) bits of the signature?
- Instead of storing \( n \) signature files, store \( w \) bit slices
- Only check the slices that correspond to the set bits in the word’s hash value
- Start from the sparse slices

Inverted lists versus signatures

- Inverted lists better for most purposes (TODS, 1998)
- Problems of signature files
  - False positives
  - Hard to use because \( s, w \), and the hash function need tuning to work well
  - Long documents will likely have mostly 1’s in signatures
  - Common words will create mostly 1’s for their slices
  - Difficult to extend with features such as frequency, proximity
- Saving grace of signature files
  - Sizes are tunable
  - Good for lots of search terms
  - Good for computing similarity of documents

Ranking result pages

Possible ranking criteria

- Based on content
  - Number of occurrences of the search terms
  - Similarity to the query text
- Based on link structure
  - Backlink count
  - PageRank
- And more…
Textual similarity
- Terms \( \{ t_1, \ldots, t_n \} \) and documents \( D = \{ d_1, d_2, \ldots \} \)
- IDF (Inverse Document Frequency) of \( t_i \):
  \[ \text{idf}_i = -\log\left( \frac{\# \text{of docs in } D \text{ containing } t_i}{\# \text{of docs}} \right) \]
- TF (Term Frequency) of \( t_i \) in \( d_j \):
  \[ \text{tf}_i,j = \frac{\# \text{of } t_i \text{ appearances in } d_j}{\# \text{of all terms in } d_j} \]
- TF-IDF weight vector of \( d_j \):
  \[ w_j = (\text{tf}_1,\text{idf}_1, \ldots, \text{tf}_n,\text{idf}_n) \]
- Textual similarity between two docs \( d_j \) and \( d_k \) can be measured by the normalized dot product of these vectors, i.e.:
  \[ \frac{\sum \text{tf}_i,j \text{idf}_i \text{tf}_i,k \text{idf}_i}{\sqrt{\sum \text{tf}_i,j^2 \text{idf}_i^2} \sqrt{\sum \text{tf}_i,k^2 \text{idf}_i^2}} \]

Why weigh significance by IDF?
- Without IDF weighting, the similarity measure would be dominated by the stop words
- "the" occurs frequently on the Web, so its occurrence on a particular page should be considered less significant
- "engine" occurs infrequently on the Web, so its occurrence on a particular page should be considered more significant

Problems with content-based ranking
- Many pages containing search terms may be of poor quality or irrelevant
  - Example: a page with just a line "search engine"
- Many high-quality or relevant pages do not even contain the search terms
  - Example: Google homepage
- Page containing more occurrences of the search terms are ranked higher; spamming is easy
  - Example: a page with line "search engine" repeated many times

Backlink
- A page with more backlinks is ranked higher
  - Intuition: Each backlink is a "vote" for the page's importance
  - Based on local link structure; still easy to spam
  - Create lots of pages that point to a particular page

Google's PageRank
- Main idea: Pages pointed by high-ranking pages are ranked higher
  - Definition is recursive by design
  - Based on global link structure; hard to spam
- Naïve PageRank
  - \( F(p) \): set of pages that page \( p \) points to
  - \( B(p) \): set of pages that point to \( p \)
  - \( \text{PageRank}(p) = \sum_{q \in B(p)} \frac{\text{PageRank}(q)}{|F(q)|} \)
  - Each page gets a boost from every page pointing to it
  - Each page distributes its importance evenly to pages it points to

Calculating naïve PageRank
- Initially, set all PageRank’s to 1; then evaluate
  \[ \text{PageRank}(p) = \sum_{q \in B(p)} \frac{\text{PageRank}(q)}{|F(q)|} \]
  repeatedly until the values converge (i.e. a fixed point is reached)

<table>
<thead>
<tr>
<th></th>
<th>Yahoo</th>
<th>m = 0.5</th>
<th>a = 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Amazon</th>
<th>m = 0.75</th>
<th>a = 0.75</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Microsoft</th>
<th>m = 0.6875</th>
<th>a = 0.6875</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>1</td>
<td>0.6875</td>
<td>1.2</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>0.6875</td>
<td>1.2</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0.6875</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Random surfer model

- A random surfer
  - Starts with a random page
  - Randomly selects a link on the page to visit next
  - Never uses the “back” button

- PageRank($p$) measures the probability that a random surfer visits page $p$

Problems with the naïve PageRank

- Dead end: a page with no outgoing links
  - A dead end causes all importance to “leak” eventually out of the Web

- Spider trap: a group of pages with no links out of the group
  - A spider trap will eventually accumulate all importance of the Web

Practical PageRank

- $d$: decay factor

- PageRank($p$) = 
  $d \cdot \sum_{q \in B(p)} \text{PageRank}(q)/|F(q)| + (1 - d)$

- Intuition in the random surfer model
  - A surfer occasionally gets bored and jump to a random page on the Web instead of following a random link on the current page

Google (1998)

- Inverted lists in practice contain a lot of context information

Summary

- Index documents for substring queries
  - Inverted lists, signature files—index “words”
  - Other approaches (not covered): suffix tree, Pat tree, suffix array—index all suffixes

- Web search and information retrieval go beyond substring queries
  - TF-IDF, PageRank, …