Queries and Indexes

CISC489/689-010, Lecture #7
Wednesday, March 4
Ben Carterette

Project Notes

• Next worksheet:
  – Inverted lists for terms in the wiki000 documents.
  – For each term, store:
    • The list of document numbers it occurs in.
    • The term frequencies in those documents.
    • The document frequency (total number of documents it occurs in).
  – If inclined, you may store other information:
    • Term positions, field information, etc.
Project Notes

• Inverted list compression:
  – You should compress the inverted lists.
  – Use d-gaps for document numbers.
  – Compress integers using one of the methods discussed in class.

• Store everything in memory.
  – Writing to disk will be the next part of the project.
  – I strongly recommend using ir.cis to run your code.
    • It has a total of 128Gb of RAM (8 nodes, 16Gb per node).

Indexing Process

Documents
  (E-mails, web pages, Word docs, news articles, ...)

Text acquisition
  (Crawler, feeds, filter, ...)

Corpus
  Accessible data store

Text transformation
  (Parsing, stopping, stemming, extraction, ...)

Index creation
  (Document/term stats, weighting, inversion, ...)

Server(s)
Query Process

• We have an index stored on disk.
  — Inverted file, vocabulary, collection.
  — Contains features of terms and documents:
    • Term frequencies in documents, document frequencies,
      term positions, link-graph features, ...

• User inputs a query.

• Engine computes features of the query.

• Engine accesses index to respond to query.
  — Matches query features to document/term features in
    index to score each document.

• Returns a ranked list of documents.
Abstract Model of Ranking

More Concrete Model

$$R(Q, D) = \sum_i g_i(Q) f_i(D)$$

- $f_i$ is a document feature function
- $g_i$ is a query feature function

[Floof's Tropical Fish Shop is the best place to find tropical fish at low prices. Whether you're looking for a little fish or a big fish, we've got what you need. We even have fake seaweed for your fish tank (and little surfboards too).]

9.7 fish
4.2 tropical
22.1 tropical fish
8.2 seaweed
4.2 surfboards
14 incoming links
3 days since last update

24.5 Document Score

303.01 Document Score
Example “Collection”

$S_1$ Tropical fish include fish found in tropical environments around the world, including both freshwater and saltwater species.

$S_2$ Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

$S_3$ Tropical fish are popular aquarium fish, due to their often bright coloration.

$S_4$ In freshwater fish, this coloration typically derives from iridescence, while saltwater fish are generally pigmented.

Four sentences from the Wikipedia entry for tropical fish
\[ R(Q, D) = \sum_i g_i(Q) f_i(D) \]

- \( g_i(Q) \) = \# of occurrences of \( i \) in \( Q \)
- \( f_i(D) \) = \# of occurrences of \( i \) in \( D \)

**Query:**
- tropical fish
- pigmented fish
- saltwater species bright coloration

<table>
<thead>
<tr>
<th>Term</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>aquarium</td>
<td>3:3</td>
</tr>
<tr>
<td>are</td>
<td>3:1</td>
</tr>
<tr>
<td>around</td>
<td>3:1</td>
</tr>
<tr>
<td>as</td>
<td>2:1</td>
</tr>
<tr>
<td>bright</td>
<td>1:1</td>
</tr>
<tr>
<td>coloration</td>
<td>3:1</td>
</tr>
<tr>
<td>derive</td>
<td>4:1</td>
</tr>
<tr>
<td>due</td>
<td>3:1</td>
</tr>
<tr>
<td>environments</td>
<td>1:1</td>
</tr>
<tr>
<td>fish</td>
<td>1:2</td>
</tr>
<tr>
<td>fishkeepers</td>
<td>2:3</td>
</tr>
<tr>
<td>found</td>
<td>2:3</td>
</tr>
<tr>
<td>fresh</td>
<td>2:3</td>
</tr>
<tr>
<td>freshwater</td>
<td>1:4</td>
</tr>
<tr>
<td>from</td>
<td>1:1</td>
</tr>
<tr>
<td>generally</td>
<td>1:1</td>
</tr>
<tr>
<td>include</td>
<td>1:1</td>
</tr>
<tr>
<td>including</td>
<td>1:1</td>
</tr>
<tr>
<td>iridescence</td>
<td>4:2</td>
</tr>
<tr>
<td>marine</td>
<td>2:3</td>
</tr>
<tr>
<td>often</td>
<td>2:3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>only</td>
<td>2:1</td>
</tr>
<tr>
<td>pigmented</td>
<td>3:1</td>
</tr>
<tr>
<td>popular</td>
<td>4:1</td>
</tr>
<tr>
<td>refer</td>
<td>2:1</td>
</tr>
<tr>
<td>referred</td>
<td>2:1</td>
</tr>
<tr>
<td>requiring</td>
<td>2:1</td>
</tr>
<tr>
<td>salt</td>
<td>1:1</td>
</tr>
<tr>
<td>saltwater</td>
<td>4:1</td>
</tr>
<tr>
<td>species</td>
<td>1:1</td>
</tr>
<tr>
<td>term</td>
<td>2:1</td>
</tr>
<tr>
<td>the</td>
<td>1:1</td>
</tr>
<tr>
<td>their</td>
<td>2:1</td>
</tr>
<tr>
<td>this</td>
<td>2:1</td>
</tr>
<tr>
<td>these</td>
<td>2:1</td>
</tr>
<tr>
<td>to</td>
<td>2:2</td>
</tr>
<tr>
<td>tropical</td>
<td>3:1</td>
</tr>
<tr>
<td>typically</td>
<td>1:2</td>
</tr>
<tr>
<td>use</td>
<td>2:1</td>
</tr>
<tr>
<td>water</td>
<td>2:1</td>
</tr>
<tr>
<td>while</td>
<td>2:1</td>
</tr>
<tr>
<td>with</td>
<td>2:1</td>
</tr>
<tr>
<td>world</td>
<td>1:1</td>
</tr>
</tbody>
</table>

**Query Processing**

- **Term-at-a-time**
  - Accumulates scores for documents by processing term lists one at a time
- **Document-at-a-time**
  - Calculates complete scores for documents by processing all term lists, one document at a time
- **Both approaches have optimization techniques that significantly reduce time required to generate scores**
Term-At-A-Time

\[ \text{salt} \quad 1:1 \quad 4:1 \]
\[ \text{partial scores} \quad 1:1 \quad 4:1 \]
\[ \text{old partial scores} \quad 1:1 \quad 4:1 \]
\[ \text{water} \quad 1:1 \quad 2:1 \quad 4:1 \]
\[ \text{new partial scores} \quad 1:2 \quad 2:1 \quad 4:2 \]
\[ \text{old partial scores} \quad 1:2 \quad 2:1 \quad 4:2 \]
\[ \text{tropical} \quad 1:2 \quad 2:2 \quad 3:1 \]
\[ \text{final scores} \quad 1:4 \quad 2:3 \quad 2:2 \quad 4:2 \]

Term-At-A-Time

\textbf{procedure} \textsc{TermAtATimeretrieval}(Q, I_j, f, g, k)
\begin{align*}
A & \leftarrow \text{HashTable}() \\
L & \leftarrow \text{Array}() \\
R & \leftarrow \text{PriorityQueue}(k) \\
\text{for all terms } w_i \text{ in } Q \text{ do} \\
    l_i & \leftarrow \text{InvertedList}(w_i, I) \\
    L.add(l_i) \\
\text{end for} \\
\text{for all lists } l_i \in L \text{ do} \\
    \text{while } l_i \text{ is not finished do} \\
        d & \leftarrow l_i.get\text{CurrentDocument}() \\
        A_d & \leftarrow A_d + \frac{g_j(f_i)}{f_i} \\
        l_i & \text{moveToNextDocument}() \\
    \text{end while} \\
\text{end for} \\
\text{for all accumulators } A_d \text{ in } A \text{ do} \\
    s_D & \leftarrow A_d \quad \triangleright \text{Accumulator contains the document score} \\
    R.add(s_D, D) \\
\text{end for} \\
\text{return the top } k \text{ results from } R \\
\text{end procedure}
\end{align*}
Document-At-A-Time

\[\text{salt} \quad 1:1 \quad \text{4:1} \quad \text{water} \quad 1:1 \quad 2:1 \quad 4:1 \quad \text{tropical} \quad 1:2 \quad 2:2 \quad 3:1 \quad \text{score} \quad 1:4 \quad 2:3 \quad 3:1 \quad 4:2\]

Document-At-A-Time

\begin{verbatim}
procedure DocumentAtATimeRetrieval(Q, I, f, g, k)
    // Initialize variables
    L ← Array()
    R ← PriorityQueue(k)
    for all terms wi in Q do
        li ← InvertedList(wi, I)
        L.add(li)
    end for
    for all documents d ∈ I do
        for all inverted lists li in L do
            if li points to d then
                \( s_D = s_D + g(Q)f(L_i) \)  \( \triangleright \) Update the document score
                li.movePastDocument(d)
            end if
        end for
        R.add(\( s_D, D \))
    end for
    return the top k results from R
end procedure
\end{verbatim}
Optimization Techniques

• Inverted lists can be very long
  – Decompression time + processing time can add up fast
• Optimizations are used to speed up processing time
• Two classes of optimization
  – Read less data from inverted lists
    • e.g., skip lists
    • better for simple feature functions
  – Calculate scores for fewer documents
    • e.g., conjunctive processing
    • better for complex feature functions

```java
1: procedure CONJUNCTIVETERMATEATRETRIVAL(Q, I, f, g, k)
2: A ← HashTable()
3: L ← Array()
4: R ← PriorityQueue(k)
5: for all terms w_i in Q do
6:   l_i ← InvertedList(w_i, I)
7:   L.add(l_i)
8: end for
9: for all lists l_i ∈ L do
10:   while l_i is not finished do
11:     if i = 0 then
12:       d ← l_i.getcurrentDocument()
13:       A_d ← A_d + g_i(Q) f(l_i)
14:     else
15:       d ← l_i.getcurrentDocument()
16:       d ← A_d.getnextDocumentAfter(d)
17:       l_i.skipToNextDocument(d)
18:       if l_i.getcurrentDocument() = d then
19:         A_e ← A_e + g_i(Q) f(l_i)
20:       else
21:         A_d.remove(d)
22:       end if
23:     end if
24:   end while
25: end for
26: for all accumulators A_d in A do
27:   s_D ← A_d  // Accumulator contains the document score
28: R.add( s_D, D )
29: end for
30: return the top k results from R
31: end procedure
```
Threshold Methods

• Threshold methods use number of top-ranked documents needed \((k)\) to optimize query processing
  — for most applications, \(k\) is small

• For any query, there is a minimum score that each document needs to reach before it can be shown to the user
  — score of the \(k\)th-highest scoring document
  — gives threshold score \(\tau\)
  — optimization methods estimate \(\tau'\) to ignore documents
Threshold Methods

• For document-at-a-time processing, use score of lowest-ranked document so far for $\tau'$
  – for term-at-a-time, have to use $k_w$-largest score in the accumulator table

• MaxScore method compares the maximum score that remaining documents could have to $\tau'$
  – safe optimization in that ranking will be the same without optimization

MaxScore Example

<table>
<thead>
<tr>
<th>eucalyptus</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tree</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Indexer computes $\mu_{tree}$
  – maximum score for any document containing just “tree”
• Assume $k=3$, $\tau'$ is lowest score after first three docs
• Likely that $\tau' > \mu_{tree}$
  – $\tau'$ is the score of a document that contains both query terms
• Can safely skip over all gray postings
Other Approaches

• Early termination of query processing
  – ignore high-frequency word lists in term-at-a-time
  – ignore documents at end of lists in doc-at-a-time
  – unsafe optimization

• List ordering
  – order inverted lists by quality metric (e.g., PageRank) or by partial score
  – makes unsafe (and fast) optimizations more likely to produce good documents

Review

• Query processing:
  – Document-at-a-time
  – Term-at-a-time
  – Optimizations:
    • Conjunctive processing
    • Thresholding

• How do you define $f_i$ and $g_i$ in the scoring function?
  – What is the actual goal?
Information Needs

- An information need is the underlying cause of the query that a person submits to a search engine
  - sometimes called information problem to emphasize that information need is generally related to a task
- Categorized using variety of dimensions
  - e.g., number of relevant documents being sought
  - type of information that is needed
  - type of task that led to the requirement for information

Queries and Information Needs

- A query can represent very different information needs
  - May require different search techniques and ranking algorithms to produce the best rankings
- A query can be a poor representation of the information need
  - User may find it difficult to express the information need
  - User is encouraged to enter short queries both by the search engine interface, and by the fact that long queries don’t work
Retrieval Models

• Provide a mathematical framework for defining the search process
  – includes explanation of assumptions
  – basis of many ranking algorithms
  – can be implicit
• Theories about relevance

Relevance

• Complex concept that has been studied for some time
  – Many factors to consider
  – People often disagree when making relevance judgments
• Retrieval models make various assumptions about relevance to simplify problem
  – e.g., topical vs. user relevance
  – e.g., binary vs. multi-valued relevance
# Retrieval Model Overview

- **Older models**
  - Boolean retrieval
  - Vector Space model
- **Probabilistic Models**
  - BM25
  - Language models
- **Combining evidence**
  - Inference networks
  - Learning to Rank

## Boolean Retrieval

- **Two possible outcomes for query processing**
  - TRUE and FALSE
  - “exact-match” retrieval
  - simplest form of ranking
- **Query usually specified using Boolean operators**
  - AND, OR, NOT
  - proximity operators also used
Boolean Retrieval

• Advantages
  – Results are predictable, relatively easy to explain
  – Many different features can be incorporated
  – Efficient processing since many documents can be eliminated from search

• Disadvantages
  – Effectiveness depends entirely on user
  – Simple queries usually don’t work well
  – Complex queries are difficult

Searching by Numbers

• Sequence of queries driven by number of retrieved documents
  – e.g. “lincoln” search of news articles
  – president AND lincoln
  – president AND lincoln AND NOT (automobile OR car)
  – president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
  – president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)
Vector Space Model

- Documents and queries represented as vectors in V-dimensional space.
  - Vector coefficients are term weights.
    
    \[ D_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \ldots & w_{1,V} \end{bmatrix} \]
    
    \[ D_2 = \begin{bmatrix} w_{21} & w_{22} & w_{23} & \ldots & w_{2,V} \end{bmatrix} \]
    
    \[ \ldots \]
    
    \[ D_N = \begin{bmatrix} w_{N,1} & w_{N,2} & w_{N,3} & \ldots & w_{N,V} \end{bmatrix} \]
    
    \[ Q = \begin{bmatrix} w_{Q,1} & w_{Q,2} & w_{Q,3} & \ldots & w_{Q,V} \end{bmatrix} \]
Vector Space Model

• Visualization:

Term Weighting

• What features are useful in term weights?
• Term frequency (tf):
  – term occurs often in a document $\rightarrow$ document more likely to be relevant.
• Inverse document frequency (idf):
  – term appears in many documents $\rightarrow$ less discriminating $\rightarrow$ documents it appears in less likely to be relevant.
• Document length:
  – Very long document $\rightarrow$ each term occurrence less important $\rightarrow$ document less likely to be relevant.
• How can we combine these into a weight $w_{ik}$?
Term Weighting

• There are many different ways to weight terms.
• tf-idf weighting is one of the most common.
  – Term frequency of term k in document i: $tf_{ik} = \frac{f_{ik}}{l_i}$
  – Inverse document frequency of term k: $idf_k = \log \frac{N}{n_k}$
  – Weight of term k in document i = tf*idf: $d_{ik} = \frac{f_{ik}}{l_i} \log \frac{N}{n_k}$

Vector Space Model

• Documents ranked by distance between vectors representing query and documents
  – e.g. Cosine correlation

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{ij}^2 \cdot \sum_{j=1}^{t} q_j^2}}$$
Similarity

Consider two documents $D_1, D_2$ and a query $Q$

- $D_1 = (0.5, 0.8, 0.3), D_2 = (0.9, 0.4, 0.2), Q = (1.5, 1.0, 0)$

\[
\text{Cosine}(D_1, Q) = \frac{\sum_{j=1}^{3} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{3} d_{ij}^2 \cdot \sum_{j=1}^{3} q_j^2}}
\]

\[
\text{Cosine}(D_1, Q) = \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}}
\]

\[
= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87
\]

\[
\text{Cosine}(D_2, Q) = \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}}
\]

\[
= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97
\]