Information Retrieval and Web Search Engines

© Leonidas Fegaras
University of Texas at Arlington

Information Retrieval (IR)

- Originally, used for document management systems
- Popular now due to web search engines
- IR has some similarities with traditional databases
  - very large data sets
  - use of indexes for fast access
- but IR has many differences from traditional databases
  - unstructured data (text documents)
  - keyword search queries
    eg, “windows and (glass or door) and not Microsoft”
  - read mostly -- but addition of new documents occasionally (off-line)
  - requires relevance ranking to retrieve the top-k results
  - imprecise semantics

Inverted File

- Also known as inverted index
- Maps words into document locations (URLs)
  - from each word, you get the set of documents that contain this word
- Relational schema
  - Document (id, URL)
  - Word (term, docID)
    key is combination of term/docID
- ... but IR systems do not use a RDBMS
  - they may use a B+tree or a hash index without a table
  - the index delivers the list of documents containing the word sorted by docID
- Keyword queries
  - the result of each query term is a list of documents sorted by docID
  - query1 and query2: list intersection (merging)
  - query1 or query2: list union
  - query1 and not query2: list subtraction

Keyword Queries in SQL

- Single-table selects plus UNION, INTERSECT, and EXCEPT
  - “windows and (glass or door) and not Microsoft”
  - ((select docID from Word where term=”windows”) intersect
    (select docID from Word where term=”glass” or term=”door”))
  - except
    (select docID from Word where term=”Microsoft”)
- Never done this way in IR!
  - they use special-purpose, optimized search engines
- Needs also relevance ranking
  - requires statistics
    - how often a term appears in a document?
    - how rare the term is among all documents?
  - not easy to calculate using RDBMS
**Better Schema**

- Need to include
  - number of documents containing the term
  - the term position in document (for checking term proximity)
    ```sql
    Document ( id, URL )
    Word ( termID, term, count )
    Posting ( termID, position, docID )
    ```
- Integrity constraint: for each term (tid,term,cnt) in Word
  ```sql
  cnt = count(select distinct docID from Posting where termID=tid)
  ```
- Keyword query: “computer and science”
  ```sql
  select distinct p1.docID
  from Word w1, Posting p1, Word w2, Posting p2
  where w1.term="computer" and w2.term="science"
  and w1.termID=p1.termID and w2.termID=p2.termID
  and p1.docID=p2.docID
  order by abs(p1.position-p2.position)
  ```

**The Vector Space Model**

- A model for estimating relevance ranking and document similarities
- Documents and queries are represented as vectors of floats
  - vector elements correspond to indexed terms (words)
  - vector values are term weights
  - highly sparse vector, usually implemented by inverted lists
- *Stop words* are considered irrelevant and are eliminated
  - e.g., certain words such as “the”, “a”, and HTML tags such as `<p>`
- Terms are usually *stems*
  - stemming: use language language-specific rules to convert words to their basic forms
  - e.g., “toys”, “toying”, are converted to “toy”

**Example**

- Document vectors can indicate frequency of terms in document
  ```
<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>science</th>
<th>engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>D4</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
  ```
- A query vector indicates the weight (ie, the importance) you give to a search term
  ```
<table>
<thead>
<tr>
<th>query</th>
<th>computer</th>
<th>science</th>
<th>engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
  ```
- If documents and the query are represented as points in a multidimensional space (one dimension per term), then relevance ranking is space proximity
  - the best match is the document closest to the query in the multidimensional space

**TFxIDF Weights**

- For a given document i and a term k we have:
  - the term frequency $tf_{ik}$ of term k in document i
  - the inverse document frequency $idf_k$ of term k, given by
    ```
    idf_k = \log(N/n_k)
    ```
    where N is the total number of documents and $n_k$ is the number of documents that contain the term k
  - The weight is $w_{ik} = tf_{ik} \cdot idf_k$
  - Normalization: force $w_{ik}$ to be between 0 and 1
    - that way, weights resemble probabilities
    ```
    w_{ik}' = \frac{w_{ik}}{\sum_j w_{ij}'}
    ```
    - Relevance of a query Q to a document $D_i$:
      ```
      \text{sim}(Q,D_i) = \sum_{i} q_i \cdot w_{qi}
      ```
Example

<table>
<thead>
<tr>
<th>tf</th>
<th>computer</th>
<th>science</th>
<th>engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>D4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i_d$</th>
<th>computer</th>
<th>science</th>
<th>engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>log(20) = 0.3</td>
<td>log(0.5) = 0.13</td>
<td>log(12) = 0.12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>query</th>
<th>computer</th>
<th>science</th>
<th>engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

$w_i$ = $i_d$ * $t_f$

$\text{so document D1 is the best match}$

Document Similarity

- Pairwise document similarity
  $\text{sim}(D_i, D_j) = \sum_{k=1}^{n} w_{i_k} * w_{j_k}$
- Normalization: divide by $\sqrt{\sum_{k=1}^{n} w_{i_k}^2}$ and by $\sqrt{\sum_{k=1}^{n} w_{j_k}^2}$
- Text clustering
  - finds overall similarities among groups of documents
- How to expand the search?
  - thesaurus expansion
  - relevance feedback

Web Search Engines

- These are IR systems for web-accessible HTML pages
- Inverse indexes are populated by web-crawlers off-line
  1. new index entries are created from the HTML documents
  2. then, the new entries are sorted
  3. finally, the results are merged with the existing index and a new index is created
- Relevance ranking goes beyond TFxIDF
  - page popularity
    - gives higher score to frequently visited web pages
  - based on the importance of other pages that refer to this page
    - if a page is referred to by an “important” page, then it is also “important”
  - Google's PageRank

Google

- The Indexer converts each crawled HTML document into a collection of “hits” and puts them into “barrels”
  - each barrel contains postings for a range of words
  - each barrel has one Lexicon with entries (word,wordID,docs,offset)
    - docs is the number of documents containing the word
    - offset points to the first entry in the Posting (the first hit)
    - the Lexicon is always in memory
  - a hit is (wordID,position,font,type). It is 2 bytes.
    - wordID is a reference to a word in the “lexicon”
    - position is the position of the word in the document
    - font indicates whether the word is inside <b><i></b>, <em><em>, etc
    - the type is a flag that indicates a fancy hit (word in title, URL, etc) or not
PageRank

- Assumption: if the pages pointing to a page are important, then the latter page is also important
- Let A1, A2, ..., An be the pages that point to the page A. Then the PageRank of A is
  \[ PR(A) = (1-d) + d \sum \frac{PR(A_i)}{C(A_i)} + \ldots + \frac{PR(A_n)}{C(A_n)} \]
  where C(Ai) is the number of outgoing links from Ai
- The PR vector is the principal eigenvector of the link matrix of the web
  - can be computed as the fixpoint of the above equation
  - in practice, it is computed incrementally
- Google computes the relevance of a page for a given search by first computing an TFxIDF relevance and then adjusting it by taking into account the PR of the top-ranked pages