Information Retrieval and Web Search

IR models: Vector Space Model

Instructor: Rada Mihalcea

[Note: Some slides in this set were adapted from an IR course taught by Ray Mooney at UT Austin (who in turn adapted them from Joydeep Ghosh), and from an IR course taught by Chris Manning at Stanford]
Naïve Implementation

Convert all documents in collection D to tf-idf weighted vectors, $d_j$, for keyword vocabulary V.

Convert query to a tf-idf-weighted vector $q$.

For each $d_j$ in D do

Compute score $s_j = \text{cosSim}(d_j, q)$

Sort documents by decreasing score.

Present top ranked documents to the user.

Time complexity: $O(|V| \cdot |D|)$  Bad for large $V$ & $D$!

$|V| = 10,000; \ |D| = 100,000; \ |V| \cdot |D| = 1,000,000,000$
Practical Implementation

- Based on the observation that documents containing none of the query keywords do not affect the final ranking
- Try to identify only those documents that contain at least one query keyword
- Actual implementation of an inverted index
Step 1: Preprocessing

- Implement the preprocessing functions:
  - For tokenization
  - For stop word removal
  - For stemming

- **Input:** Documents that are read one by one from the collection

- **Output:** Tokens to be added to the index
  - No punctuation, no stop-words, stemmed
Step 2: Indexing

• Build an inverted index, with an entry for each word in the vocabulary

• **Input**: Tokens obtained from the preprocessing module
• **Output**: An inverted index for fast access
Step 2 (continued)

- Many data structures are appropriate for fast access
  - B-trees, skipped lists, hashtables

- We need:
  - One entry for each word in the vocabulary
  - For each such entry:
    - Keep a list of all the documents where it appears together with the corresponding frequency $\rightarrow \text{tf}$
    - For each such entry, keep the total number of documents in each the word occurs:
      - $\rightarrow \text{idf}$
**Step 2 (continued)**

<table>
<thead>
<tr>
<th>Index terms</th>
<th>$df$</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>3</td>
</tr>
<tr>
<td>database</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>science</td>
<td>4</td>
</tr>
<tr>
<td>system</td>
<td>1</td>
</tr>
</tbody>
</table>

Index file

- $D_7$, 4
- $D_1$, 3
- $D_2$, 4
- $D_5$, 2

Lists

$D_j$, $tf_j$
Step 2 (continued)

- tf and idf for each token can be computed in one pass
- Cosine similarity also requires document lengths
- Need a second pass to compute document vector lengths
  - Remember that the length of a document vector is the square-root of sum of the squares of the weights of its tokens.
  - Remember the weight of a token is: tf-idf
  - Therefore, must wait until idf-s are known (and therefore until all documents are indexed) before document lengths can be determined.

- Do a second pass over all documents: keep a list or hashtable with all document id-s, and for each document determine its length.
Time Complexity of Indexing

- Complexity of creating vector and indexing a document of $n$ tokens is $O(n)$.
- So indexing $|D|$ such documents is $O(|D| \cdot n)$.
- Computing token idf-s can be done during the same first pass
- Computing vector lengths is also $O(|D| \cdot n)$.
- Complete process is $O(|D| \cdot n)$, which is also the complexity of just reading in the corpus.
Step 3: Retrieval

- Use inverted index (from step 2) to find the limited set of documents that contain at least one of the query words.
- Incrementally compute cosine similarity of each indexed document as query words are processed one by one.
- To accumulate a total score for each retrieved document, store retrieved documents in a hashtable (or another search data structure), where the document id is the key, and the partial accumulated score is the value.

- **Input**: Query and Inverted Index (from Step 2)
- **Output**: Similarity values between query and documents
Step 4: Ranking

- Sort the search structure including the retrieved documents based on the value of cosine similarity
- Return the documents in descending order of their relevance

- **Input**: Similarity values between query and documents
- **Output**: Ranked list of documented in reversed order of their relevance
What Weighting Methods?

- Weights applied to both document terms and query terms
- Direct impact on the final ranking
- Direct impact on the results
- Direct impact on the quality of IR system
### Standard Evaluation Measures

Start with a CONTINGENCY table

<table>
<thead>
<tr>
<th></th>
<th>retrieved</th>
<th>not retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>relevant</td>
<td>$w$</td>
<td>$x$</td>
</tr>
<tr>
<td>not relevant</td>
<td>$y$</td>
<td>$z$</td>
</tr>
</tbody>
</table>

$n_1 = w + x$

$n_2 = w + y$

$N$
Precision and Recall

From all the documents that are relevant out there, how many did the IR system retrieve?

Recall: \[
\frac{W}{W+X}
\]

From all the documents that are retrieved by the IR system how many are relevant?

Precision: \[
\frac{W}{W+y}
\]
Precision and Recall for a Set of Queries

- For each query, determine the retrieved documents retrieved and the relevant documents

- Calculate
  - Macro-average: average the P/R/F calculated for the individual queries
  - Micro-average: sum all/relevant documents for individual queries, and calculate P/R/F only once