Basic Tokenizing, Indexing, and Implementation of Vector-Space Retrieval
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 $s_j = \cosSim(d_j, q)$

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 (cont’d)

• Many data structures are appropriate for fast access
  – B-trees, sparse 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 \( \rightarrow \) TF
  – For each such entry, keep the total number of documents where the word occurred:
    • \( \rightarrow \) IDF
### Step 2 (cont’d)

#### Index terms

<table>
<thead>
<tr>
<th>Index terms</th>
<th>( df )</th>
<th>( D_i, tf_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>3</td>
<td>D_7, 4</td>
</tr>
<tr>
<td>database</td>
<td>2</td>
<td>D_1, 3</td>
</tr>
<tr>
<td>science</td>
<td>4</td>
<td>D_2, 4</td>
</tr>
<tr>
<td>system</td>
<td>1</td>
<td>D_5, 2</td>
</tr>
</tbody>
</table>
Step 2 (cont’d)

- Term frequencies and DF for each token can be computed in one pass
- Cosine similarity also requires the lengths of the document vectors.
- Might need a second pass (through document collection or the inverted index) to compute document vector lengths.
Step 2 (cont’d)

– 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.
– Remember that the length of a document vector is the square-root of sum of the squares of the weights of its tokens.

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

- Complexity of creating vector and indexing a document of $n$ tokens is $O(n)$.
- So indexing $m$ such documents is $O(m \cdot n)$.
- Computing token IDF$s$ can be done during the same first pass.
- Computing vector lengths is also $O(m \cdot n)$.
- Complete process is $O(m \cdot n)$, 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, where the document id is the key, and the partial accumulated score is the value.
Step 3 (cont’d)

- **Input**: Query and Inverted Index (from Step2)
- **Output**: Similarity values between query and documents
Step 4: Ranking

• Sort the hashtable 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

Starts with a CONTINGENCY table for each query

<table>
<thead>
<tr>
<th></th>
<th>retrieved</th>
<th>not retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>relevant</td>
<td>$TP$</td>
<td>$FN$</td>
</tr>
<tr>
<td>not relevant</td>
<td>$FP$</td>
<td>$TN$</td>
</tr>
</tbody>
</table>

$n_1 = TP + FN$

$n_2 = TP + FP$

$N$
Precision and Recall

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

\[
\text{Recall: } \frac{TP}{TP + FN}
\]

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

\[
\text{Precision: } \frac{TP}{TP + FP}
\]