Contents

1 From Boolean to Statistical Models

1.1 Problems with Boolean Models

- difficult to capture user information request and document content
- difficult to control the number of documents retrieved
- difficult to rank output
- difficult to perform automatic relevance feedback

1.2 Boolean Model

- Evaluation by Blair and Maron [CACM, March 1985]
  - 40,000 legal documents.
  - System: STAIRS with ranking by number of matches found in documents.
  - Original queries were given by lawyers.
  - Boolean queries were formulated by paralegals with expertise in STAIRS
  - Interaction between lawyers and paralegals were allowed.
  - Iterations continued until the lawyers were satisfied.

Average Precision and Recall = 20% and 80%

1.3 Statistical Models

- user specifies a set of desired terms with optional weights

$$Q =< \text{database } 0.5; \text{text } 0.8; \text{information } 0.2 >$$

- retrieval based on similarity between query and documents
- outputs are ranked according to similarity
- automatic relevance feedback

1.4 Problems to Solve in Statistical Models

- How to determine important words in a document?
- How to determine the degree of importance of a word *within a document* and *within the entire collection*?
- How to determine the degree of similarity with a document and the query?

1.5 Term-frequency Considerations

- Elimination of function (common) words.
- Convert every word to its root form (only if you want to ignore syntactic categories in retrieval).
- Compute the weights of terms based on frequency information.
- Choose a threshold and assign to each document all terms with weights greater than the threshold.

  "The frequency of occurrence of nonfunction words may actually be used to indicate term importance for content representation." [Salton p. 279]

By indexing single words only, the indexing process ignores compound nouns or concepts embodied in more than one word. Also grammatical information such as syntactic categories is ignored. Consider synonyms: *lead* (verb) vs *lead* (noun), and *lead* (noun) as a kind of metal vs *lead* (noun) as the core of a pencil.

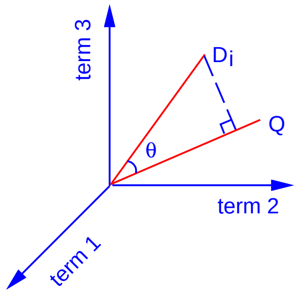
1.6 Document Representation in Vector-Space Model

- $$\begin{array}{ccccccc} D_i & = & \langle t_{i,1}, & t_{i,2}, & \ldots, & t_{i,n} \rangle \\ Q & = & \langle q_1, & q_2, & \ldots & q_n \rangle \end{array}$$
- Example:

$$\begin{array}{ccccccccc} & & \text{retrieval} & \text{database} & \text{architecture} & \text{computer} & \text{text} & \text{management} & \text{information} \\ D_i & = & 1, & 4, & 0, & 4, & 4, & 0, & 9 \\ Q & = & 0, & 0.5, & 0, & 0, & 0.8, & 0, & 0.2 \end{array}$$

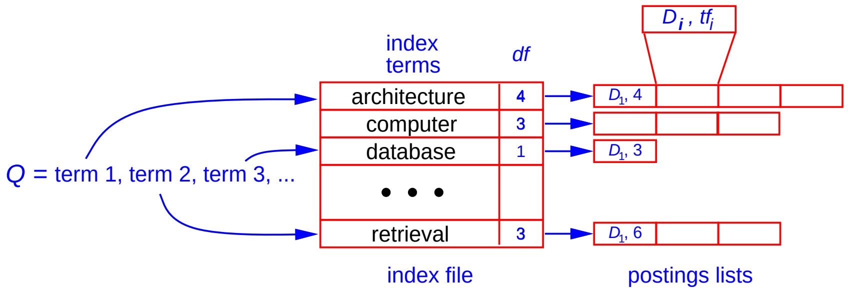
- Documents are ranked according to a similarity measure:

$$sim(Q, D_i) = \frac{\sum_{j=1}^{n} q_j \times t_{i,j}}{\sqrt{\sum_{j=1}^{n} q_j^2 \times \sum_{j=1}^{n} t_{i,j}^2}} = cos\theta$$



1.7 Implementation based on Inverted Files

- In practice, document vectors are not stored directly; an inverted organization provides much better access speed.
- The index file can be implemented as a hash file, a sorted list, or a B-tree.



1.8 Document and Query Term Weights

- $t_{i,j} = tf_{i,j} \times idf_j$

$$\begin{aligned} tf_{i,j} &= \text{frequency of term } j \text{ in document } i \\ idf_j &= \text{inverse document frequency of term } j \\ &= \log_2 \frac{\text{number of documents}}{df_j} \\ df_j &= \text{document frequency of term } j \\ &= \text{number of documents containing term } j \end{aligned}$$

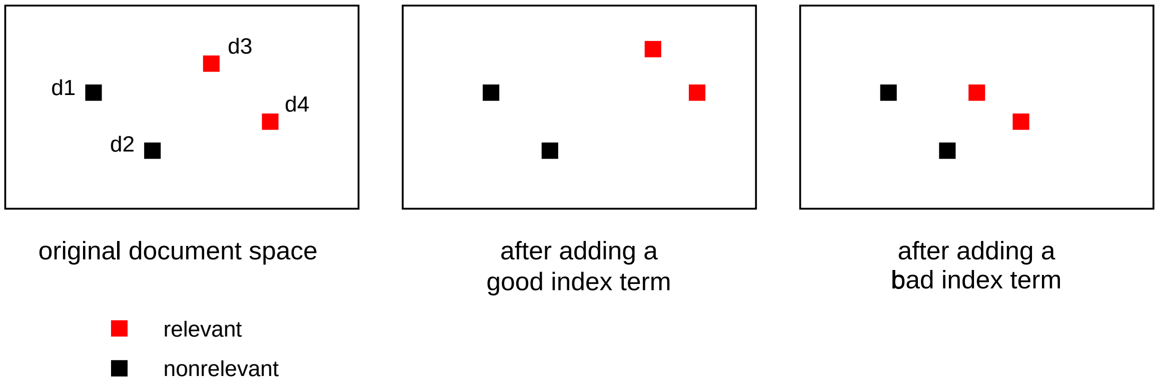
- a term has high weight in a document if it appears frequently in that document but infrequently in other documents.
- $q_j = \{0, 1\}$
- A natural language query can be regarded as a document: $q_j = tf_{q,j} \times idf_j$

Discussions:

- Comparing documents to documents: treating queries as documents or vice versa.
- Paragraph based retrieval: treating paragraphs as documents to retrieve relevant paragraphs from long documents; can also be used to formulate new ranking algorithms based on "paragraph hits".
- Comparing the similarity of documents structurally: comparing paragraphs from two documents pairwisely.

1.9 Term-discrimination Value

- The purpose of an index term is to distinguish a document from another.
- If documents are visualized as existing in a document space (of high dimensions), a good index term should "spread out" the documents as much as possible.



Example of good index terms and bad index terms:

	all terms	indexed	bad	good
d_1	(a, b, c, d, r)	(b, c, d)	(a, b, c, d)	(b, c, d, r)
d_2	(a, b, n, d, r)	(b, n, d)	(a, b, n, d)	(b, n, d, r)
d_3	(a, m, p, q)	(m, p, q)	(a, m, p, q)	(m, p, q)
d_4	(a, x, p, q)	(x, p, q)	(a, x, p, q)	(x, p, q)

- Including a will make similarity for every pair of documents greater than zero, which is bad from term discrimination point of view.
- Including r will bring d_1 and d_2 closer together while separating them farther away from d_3 and d_4.

1.10 Term-discrimination Value — Continue

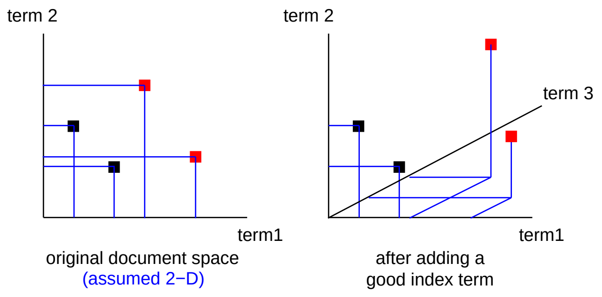
- Suppose two documents are similar if the distance between them in the document space is short. Therefore, a good index term should *decrease* the overall similarity between the document pairs.

- If Q is the total similarity value of the existing document space and Q_j is the total similarity value after adding index term j, term j is a good index term if dv_j is positive.

$$dv_j = Q - Q_j$$

$$Q = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{k=1 \\ i \neq k}}^{N} sim(D_i, D_k)$$

- Adding a term is like adding one more dimension to the vector-space.
- Documents not containing the term will not change.
- Document containing the term will be "dragged" apart from those not containing the term:



- The best case is to consider the distance between the sets of relevant and irrelevant documents, but this entails *a priori* knowledge on the relevance of documents.
- As the document frequency increases, the term discrimination value increases and then decreases.
- Computation of Q is expensive:
  - Require computing the similarity between $N(N-1)/2$ document pairs for each term considered.
  - Require recomputation after document updates.
- Compromise is to:
  - Find the centroid of the document space
  - Maximize the distance between the documents from the centroid.