G. Salton, A. Wong, C.S. Yang
Cornell University

Information Retrieval and Language Processing 1975

A Vector Space Model for Automatic Indexing

Presentation by Patrick Roos
Some slides taken from a presentation by **T.K.Prasad**

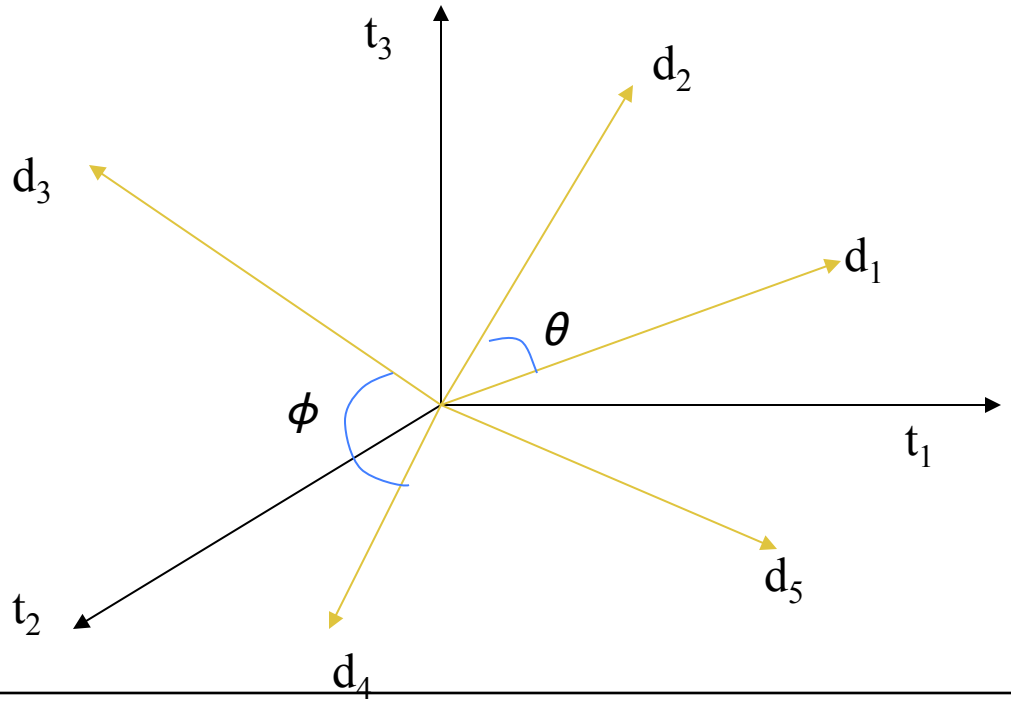
Document Space

- Documents D_i
- Each D_i is identified by one or more terms T_j
- Each term can be weighted according to their importance to the document
- D_i is a t-dimensional vector
  $D_i = (d_{i1}, d_{i2}, d_{i3}, \ldots, d_{it})$
- "Document" can be any set of entities identified by weighted property vectors (e.g. SIB-data)

Documents are vectors

- Each D_i can be viewed as a vector of weights, one component for each term
  - Terms are axes of vector space
  - Docs are points in this vector space
- Given the index vectors for two documents we can compute a similarity coefficient $s(D_i, D_j)$

Intuition



Postulate: Documents that are "close together" in the vector space talk about the same things.

Desiderata for proximity

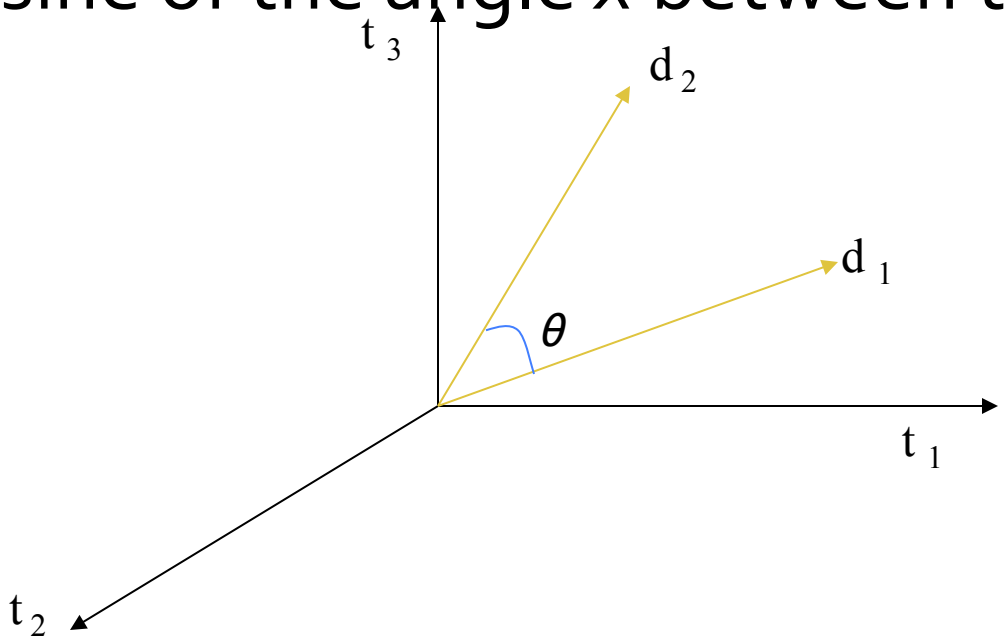
- If d_1 is near d_2, then d_2 is near d_1.
- If d_1 near d_2, and d_2 near d_3, then d_1 is not far from d_3.
- No doc is closer to d than d itself.

First cut

- *Idea*: Distance between d_1 and d_2 is the length of the vector $|d_1 - d_2|$.
  - Euclidean distance
- Why is this not a great idea?
- We still haven't dealt with the issue of length normalization
  - Short documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at *angles* instead
  - "Proportional content"

Cosine similarity

- Distance between vectors d_1 and d_2 *captured* by the cosine of the angle *x* between them.



Cosine similarity

- A vector can be *normalized* (given a length of 1) by dividing each of its components by its length – here we use the L_2 norm $\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$
- This maps vectors onto the unit sphere:
- Then, $\left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}} = 1$
- Longer documents don't get more weight

Cosine similarity

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors.

Normalization

Queries in the vector space model

Central idea: the query as a vector

- We regard the query as short document
  - Note that d_q is very sparse!
- We return the documents ranked by the closeness of their vectors to the query, also represented as a vector.

$$sim(d_j, d_q) = \frac{\vec{d}_j \cdot \vec{d}_q}{\left|\vec{d}_j\right|\left|\vec{d}_q\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,q}^2}}$$

Computing a similarity score

Say we have query vector $Q = (0.4, 0.8)$

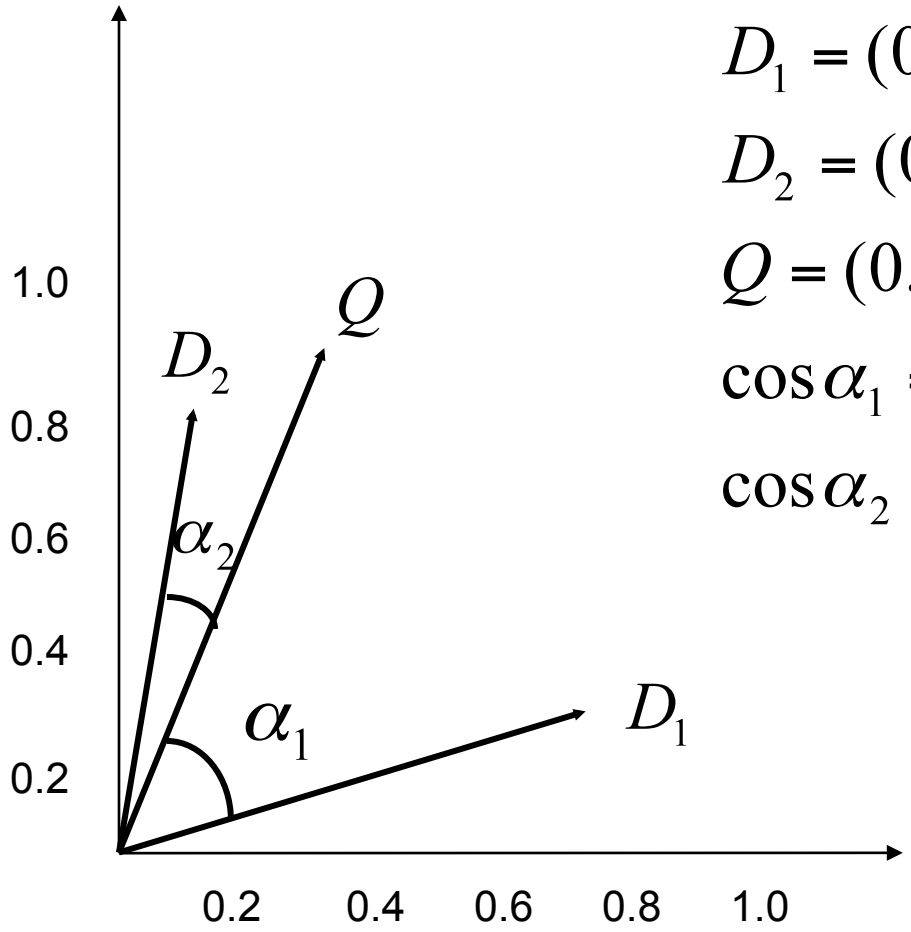
Also, document $D_2 = (0.2, 0.7)$

What does their similarity comparison yield?

$$sim(Q, D_2) = \frac{(0.4 * 0.2) + (0.8 * 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] * [(0.2)^2 + (0.7)^2]}}$$

$$= \frac{0.64}{\sqrt{0.42}} = 0.98$$

Computing Similarity Scores



$$D_1 = (0.8, 0.3)$$

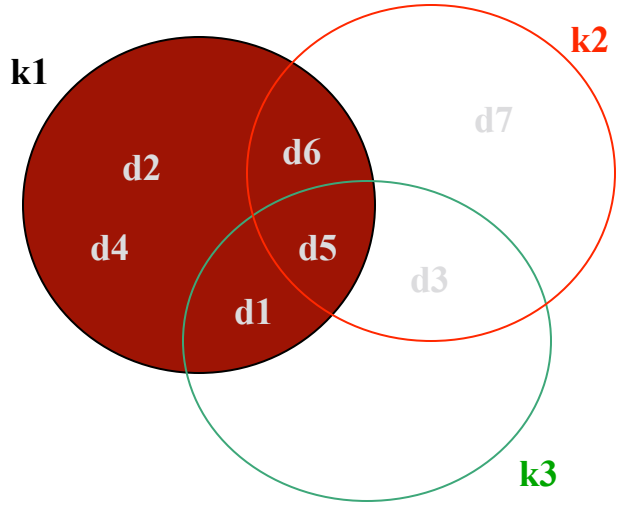
$$D_2 = (0.2, 0.7)$$

$$Q = (0.4, 0.8)$$

$$\cos\alpha_1 = 0.74$$

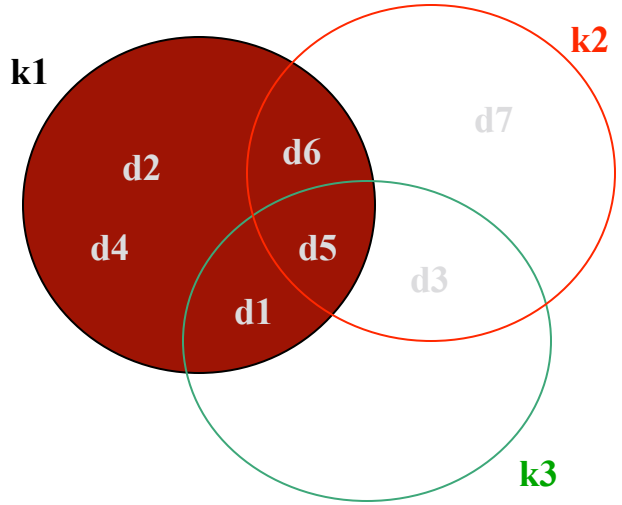
$$\cos\alpha_2 = 0.98$$

The Vector Model: Example I



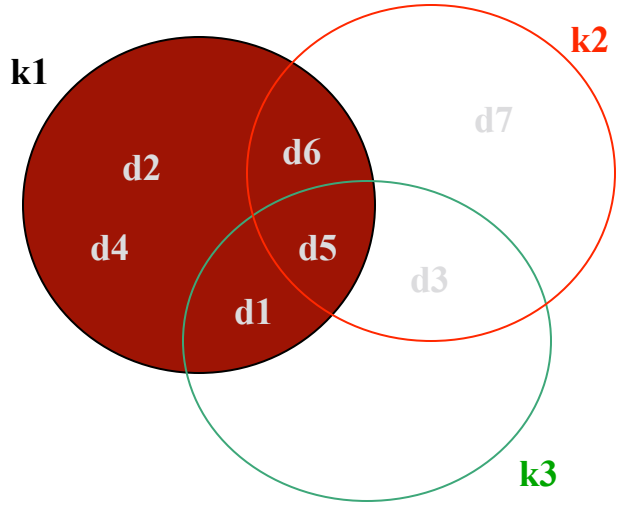
	k1	k2	k3	q • dj
d1	1	0	1	2
d2	1	0	0	1
d3	0	1	1	2
d4	1	0	0	1
d5	1	1	1	3
d6	1	1	0	2
d7	0	1	0	1
q	1	1	1	

The Vector Model: Example II



	k1	k2	k3	q • dj
d1	1	0	1	4
d2	1	0	0	1
d3	0	1	1	5
d4	1	0	0	1
d5	1	1	1	6
d6	1	1	0	3
d7	0	1	0	2
q	1	2	3	

The Vector Model: Example III



	k1	k2	k3	q • dj
d1	2	0	1	5
d2	1	0	0	1
d3	0	1	3	11
d4	2	0	0	2
d5	1	2	4	17
d6	1	2	0	5
d7	0	5	0	10
q	1	2	3	

What's the point of using vector spaces?

- A well-formed algebraic space for retrieval
- Query becomes a vector in the same space as the docs.
- Can measure each doc's proximity to it.
- Natural measure of scores/ranking
  - Documents and queries are expressed as ***bags of words***

Vector Model

- Non-binary (numeric) term weights used to compute *degree of similarity* between a query and each of the documents.
- Enables
  - *partial matches*
    - to deal with incompleteness
  - *answer set ranking*
    - to deal with information overload

Assigning Weights

- *tf* - term frequency
  - Count of times term occurs in document.
  - The more times a term *t* occurs in document *d* the more likely it is that *t* is relevant to the document.
  - Used alone, favors common words, long documents.
- *df*- document frequency
  - The number of documents in the collection to which the term is assigned
  - The more a term *t* occurs throughout all documents, the more poorly *t* discriminates between documents

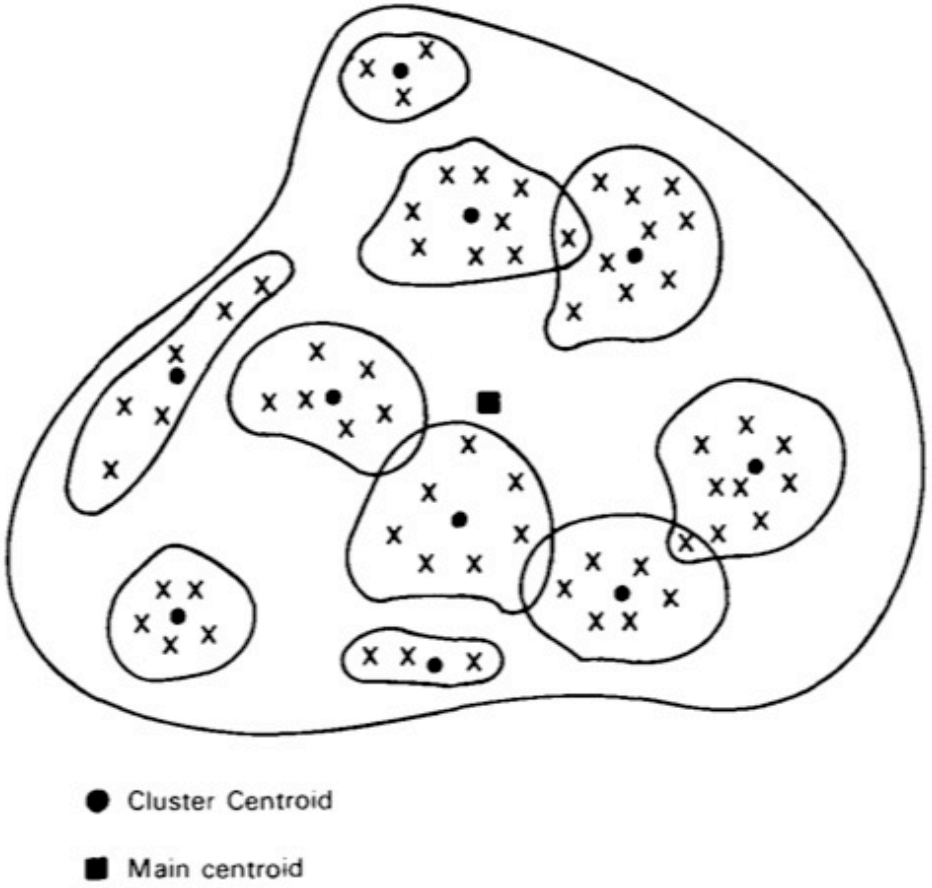
Assigning Weights – Improved

- *idf* - inverse document frequency
  - Factor inversely related to the document frequency of term, e.g.: $idf_t = \log(n/df_t)$
- ***tf * idf*** measure
- A term weighting system proportional to (*tf * idf*) will assign the largest weight to those terms which arise with high frequency in individual documents, but are at the same time relatively rare in the collection as a whole.

tf * idf

- 14 % average improvement in recall and precision (average precision improvement at the ten fixed recall points (.1 - 1)) using *tf***idf* over just *tf*.
- *tf***idf* produces lower density measures in the document space
- Improved recall-precision performance seems associated with decrease density

Clustered Document Space



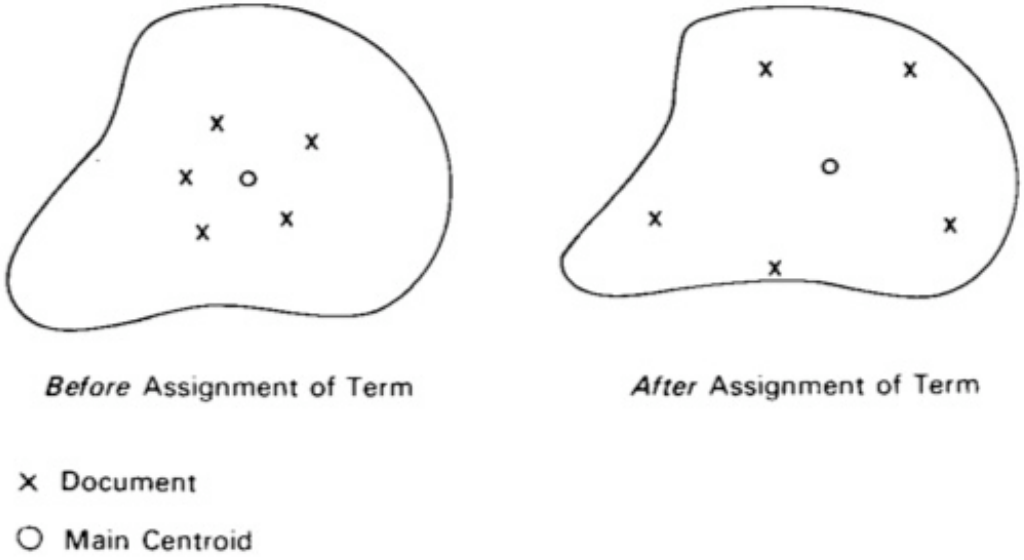
Performance and Space Density

	(a) Effect of performance improvement on space density				(b) Effect of performance deterioration on space density			
	Cluster organization A (155 clusters; 2.1 overlap)		Cluster organization B (83 clusters; 1.3 overlap)		Cluster organization A (155 clusters; 2.1 overlap)		Cluster organization B (83 clusters; 1.3 overlap)	
Type of indexing	Standard term frequency weights (f_i^k)	Term frequency with inverse doc. freq. ($f_i^k \cdot IDF_k$)	Standard term frequency weights (f_i^k)	Term frequency with inverse doc. freq. ($f_i^k \cdot IDF_k$)	Standard term frequency weights (f_i^k)	Term frequency with document frequency ($f_i^k \cdot DF_k$)	Standard term frequency weights (f_i^k)	Term frequency with document frequency ($f_i^k \cdot DF_k$)
Recall-precision output*	—	+14%	—	+14%	—	−10.1%	—	−10.1%
Average similarity between documents and corresponding cluster centeroids (x)	.712	.668 (−.044)	.650	.589 (−.061)	.712	.741 (+.029)	.650	.696 (+.046)
Average similarity between cluster centroids and main centroid	.500	.454 (−.046)	.537	.492 (−.045)	.500	.555 (+.055)	.537	.574 (+.037)
Average similarity between pairs of cluster centroids (y)	.273	.209 (−.046)	.315	.252 (−.063)	.273	.329 (+.056)	.315	.362 (+.047)
Ratio y/x	.273/.712 = .383	.209/.668 = .318 (−19%)	.315/.650 = .485	.252/.589 = .428 (−12%)	.273/.712 = .383	.329/.741 = .444 (+16%)	.315/.650 = .485	.362/.696 = .520 (+7%)

* From [2].

Discrimination Value

- Different approach similar to IDF
- A good index term decreases the similarity between documents when assigned to a collec

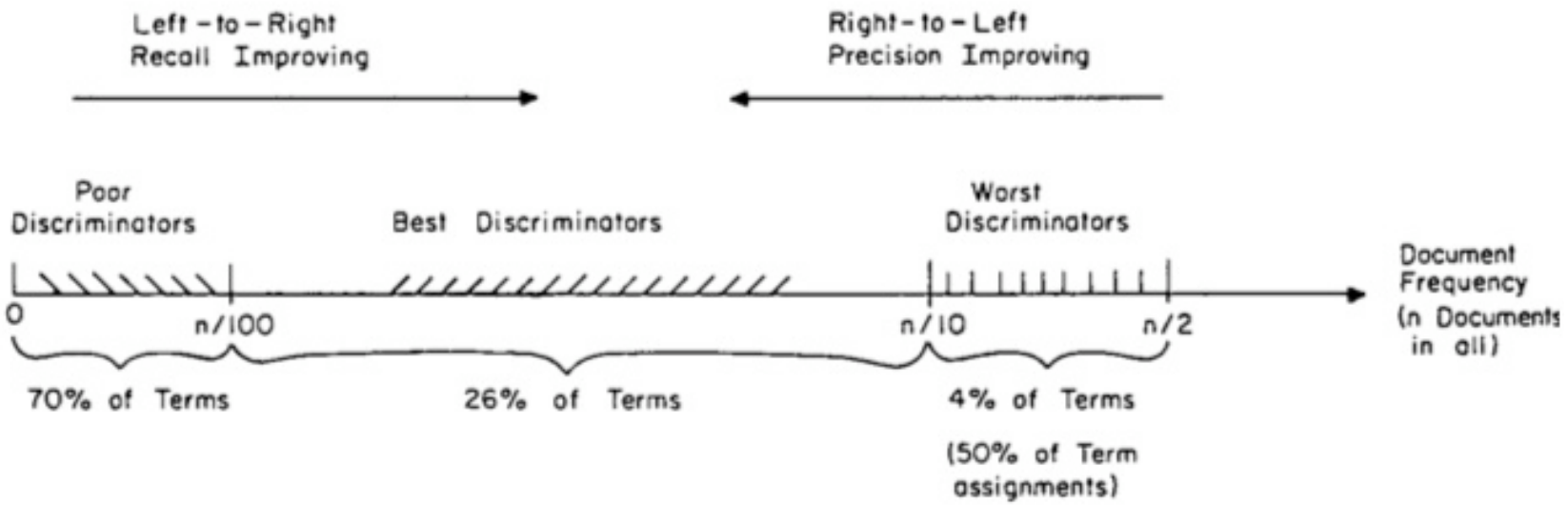


Terms' Discrimination Value

(1963 *Time* Magazine)

Good terms	Poor terms
1. Buddhist	7560. Work
2. Diem	7561. Lead
3. Lao	7562. Red
4. Arab	7563. Minister
5. Viet	7564. Nation
6. Kurd	7565. Party
7. Wilson	7566. Commune
8. Baath	7567. U.S.
9. Park	7568. Govern
10. Nenni	7569. New

Discrimination Value and Term Frequencies



Indexing Strategy

- Terms with medium document frequency should be used for content identification directly, without further transformation.
- Document Frequency Terms with very high document frequency be moved to the left on the document frequency spectrum by transforming them into entities of lower frequency; the best way of doing this is by taking high- frequency terms and using them as components of indexing *phrases--a phrase such as "programming language"* will necessarily exhibit lower document frequency than either "program", or "language" alone.
- Terms with very low document frequency should be moved to the right on the document frequency spectrum by being transformed into entities of higher frequency; one way of doing this is by collecting several low frequency terms that appear semantically similar and including them in a common term (thesaurus) class.

Vector Space Model

- Advantages:
  - Term-weighting improves answer set quality
  - Partial matching allows retrieval of docs that approximate the query conditions
  - Cosine ranking formula sorts documents according to degree of similarity to the query
- Disadvantages:
  - Assumes independence of index terms; not clear that this is bad though