

Basic Strategies

Retrieval Strategies

- Manual Systems
 - Boolean, Fuzzy Set, Inference Networks
- Automatic Systems
 - Vector Space Model
 - Latent Semantic Indexing
- Adaptive
 - Probabilistic, Genetic Algorithms , Neural Networks

Manual Systems

- Boolean Retrieval
- Extended Boolean Retrieval
- Fuzzy Set Retrieval
- Vector Space Model

Manual

- For many years, most commercial systems were only Boolean.
- Most old library systems and Lexis/Nexis have a long history of Boolean retrieval.
- Users who are experts at a complex query language can find what they are looking for.
- Ex: (t1 AND t2) OR (t3 AND t3) WITHIN 2 Sentences of (t4 AND t5) NOT (t9 OR t10).

Boolean Retrieval

- *Expression*:=
 - term
 - (*expr*)
 - NOT *expr*
 - *expr* AND *expr*
 - *expr* OR *expr*
- (cost OR price) AND paper AND NOT article

Extended (Weighted) Boolean Retrieval

- Ranking by term frequency (Sony Search Engine)
 - x AND y: $tf_x \times tf_y$
 - x OR y: $tf_x + tf_y$
 - NOT x: 0 if $tf_x > 0$, 1 if $tf_x = 0$
- User may assign term weights
 - cost and +paper

Summary of Boolean Retrieval

- Pro
 - Can use very restrictive search
 - Makes experienced users happy
- Con
 - Complex query language, confusing to end users

Retrieval Strategies

- Given a query of t terms, find a measure of relevance between the documents and the query. Rank documents based on this measure.
- Most commonly used strategy is the vector space model (proposed by Salton in 1975)

Vector Space Model

- Documents are mapped into term vector space.
- Each dimension represents tf-idf for one term.
- Queries are treated like documents.
- Documents are ranked by closeness to the query. Closeness is determined by a similarity score calculation.

VSM Example

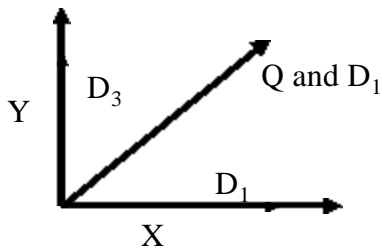
- Consider a two term vocabulary, D and I

Query: A I

D_1 - A I

D_2 - A

D_3 - I



Cosine Measure

- If X and Y are two n -dimensional vectors $\langle x_i \rangle$ and $\langle y_i \rangle$, the angle q between them satisfies:
 - $X \cdot Y = |X| |Y| \cos q$
 - $\cos q = X \cdot Y / (|X||Y|)$

Similarity Measures

- Inner Product
 - $SC(Q, D_i) = (D_i)(Q)$
- Cosine
 - $SC(Q, D_i) = (D_i)(Q) / (|D_i||Q|)$
- Dice
 - $SC(Q, D_i) = 2(D_i)(Q) / (D_i^2 + Q^2)$
- Jaccard
 - $SC(Q, D_i) = (D_i Q) / ((D_i)^2 + Q^2 - |D_i||Q|)$

VSM Similarity Measure

- Similarity is computed as the Euclidean distance from the query to each document:
- $SC(Q, D_2) = (Q \cdot D_2) / |Q| |D_2|$

$$Q = \langle 1, 1 \rangle$$

$$D_2 = \langle 1, 0 \rangle$$

$$Q \cdot D_2 = (1)(1) + (1)(0) = 1$$

$$|Q| = (1^2 + 1^2)^{1/2} = 2^{1/2} = 1.414$$

$$|D_2| = (1^2 + 0^2)^{1/2} = 1$$

$$SC(Q, D_2) = (1.414)(1) / (1.414)(1) = 1.0$$

Vector Example

- Q: “gold silver truck”
- D₁: “Shipment of gold delivered in a fire”
- D₂: “Delivery of silver arrived in a silver truck”
- D₃: “Shipment of gold arrived in a truck”
- Number of documents a term appears in is called the *document frequency*.
 - Document Frequency of the *j*th term (df_j)
- Inverse Document Frequency (idf) = $\log_{10}(n / df_j)$

Id	Term	df	idf
1	a	3	0
2	arrived	2	0.176
3	damaged	1	0.477
4	delivery	1	0.477
5	fire	1	0.477
6	gold	1	0.176
7	in	3	0
8	of	3	0
9	silver	1	0.477
10	shipment	2	0.176
11	truck	2	0.176

Incorporating Term Frequency

- Term Frequency is the number of times a term i appears in document j (tf_{ij})

<i>doc</i>	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
D_1	0	0	.477	0	.477	.176	0	0	0	.176	0
D_2	0	.176	0	.477	0	0	0	0	.954	0	.176
D_3	0	.176	0	0	0	.176	0	0	0	.176	.176
Q	0	0	0	0	0	.176	0	0	.477	0	.176

- $SC(Q, D_1) = (0)(0) + (0)(0) + (0)(0.477) + (0)(0) + (0)(0.477) + (0.176)(0.176) + (0)(0) + (0)(0)$
- This SC uses the inner product.

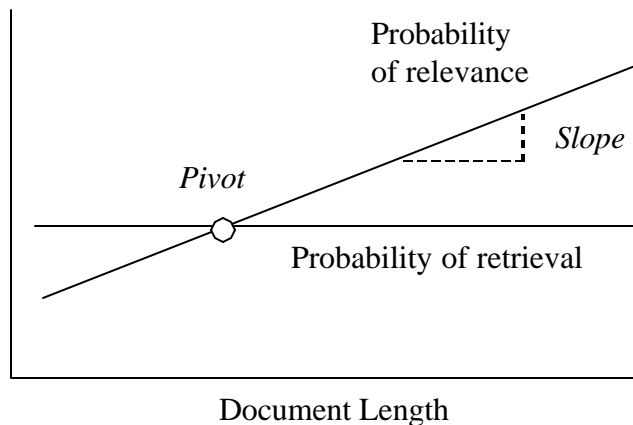
Weights for Term Components

- Classic thing to do is use $tf \times idf$
- Incorporate idf in the query and the document, one or the other or neither.
- Scale the idf with a log or even a double log.
- Augment the weight with some constant (e.g.; $w = (w)(0.5)$)

Incorporating Document Length

- Inner Product
 - Longer documents will score very high because they have more chances to match query words
- Cosine
 - Longer documents are somewhat penalized because the weight of the document $|D|$ may be very high.
- Singhal found that longer documents tend to be more relevant than shorter ones so he proposed an adjusted *document weight*.

Document Length Normalization



Summary: Vector Space Model

- Pros
 - Fairly cheap to compute
 - Yields decent effectiveness
 - Very popular -- SMART is most commonly used academic prototype
- Cons
 - No theoretical foundation
 - Weights in the vectors are very arbitrary
 - Assumes term independence

Latent Semantic Indexing

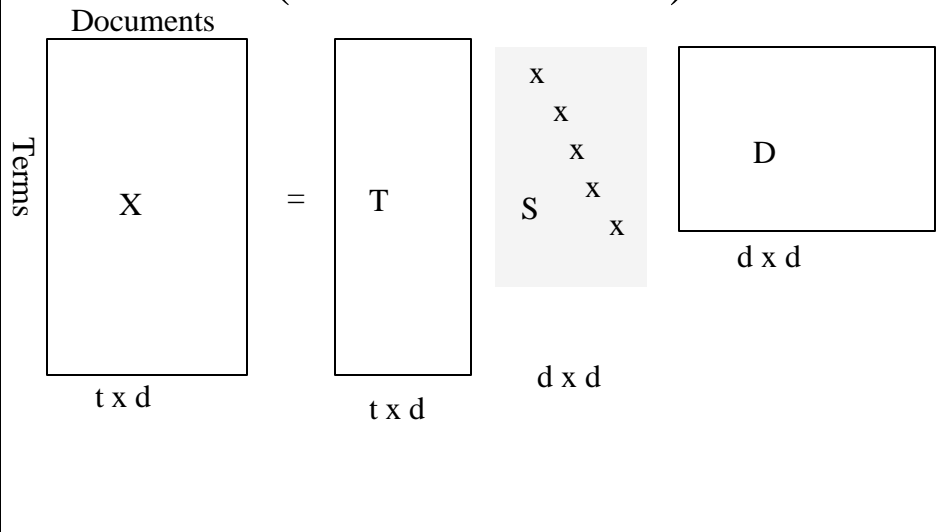
Latent Semantic Indexing

- Goal is to compute query document similarity by matching on “concepts” instead of terms
- Matrix manipulation is used to identify key “concepts”
 - “Singular value decomposition (SVD) is used to allow the arrangement of the term-document space to reflect the major associative patterns in the data, and ignore the smaller, less important differences. (Deerwester, JASIS, 1990).”

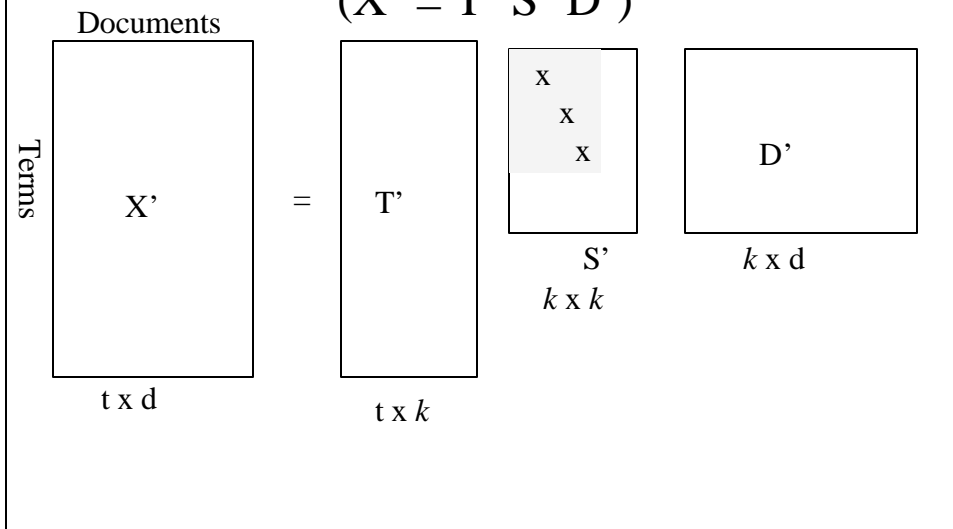
Latent Semantic Indexing

- Compute term-document matrix, X
- Compute the SVD for X
 - This will yield three matrices
 - $X = T S D'$
 - S , contains the $|D|$ singular values where $|D|$ is the number of documents.
 - Choose the top k of them to form S' . This is the interesting part -- choosing all $|D|$ gives you the original matrix, choosing fewer values gives you a “fuzzier” matrix.
 - Delete the corresponding columns of T and D to form T' and D' .
 - Now compute $X' = T' S' D'$

Matrix Structures (basic $X = T S D$)

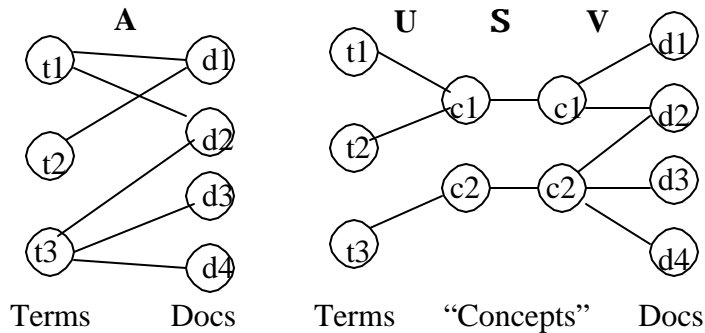


Matrix Structures After S has been reduced to S' ($X' = T' S' D'$)



Latent Semantic Indexing

$A = USV^t$ by singular value decomposition (SVD),
where S is diagonal.



Using the new X'

- Compare two terms
 - Take dot product of two row vectors of X'
- Compare two documents
 - Take dot product of two column vectors of X'
- Comparing a query to a document
 - Need to make the query look like another document. Can do this by mapping the query into 2-space: $(q^T)(T')(S')^{-1}$ Note: Taking the inverse of the singular value matrix can be done in $O(k)$ time.

LSI Summary

- Pros
 - Doesn't just match on terms, tries to match on concepts
- Cons
 - Computationally expensive, its not cheap to compute singular values, but its getting easier.
 - Choice of k is somewhat arbitrary, done by experimentation.