Evaluating retrieval effectiveness

$$\text{precision} \equiv \frac{\text{retrieved and relevant}}{\text{retrieved}}$$

$$\text{recall} \equiv \frac{\text{retrieved and relevant}}{\text{relevant}}$$

Vector Space Model

Model each document as a vector in which each term has a weight.

Query also modeled as a vector.

Uses:

- Ranked keyword searches
- Relevance feedback
- Semantic indexing

Often used in conjunction with clustering to speed up search.

Selecting Weights

$$\begin{aligned} f_{d,t} &= \text{number of times term } t \text{ appears in document } d \\ w_{d,t} &= \text{weight of } t \text{ in } dt \end{aligned}$$

Accounting for frequency within a document

$$\begin{aligned} w_{d,t} &= f_{d,t} && \text{frequency} \\ w_{d,t} &= \log(1 + f_{d,t}) && \text{log frequency} \end{aligned}$$

Accounting for "information content" of a term

$$w_t = \log N / f_t$$

giving: $w_{d,t} = \log(N/f_t) \log(1 + f_{d,t})$

Similarity between vectors

Dot product: $w_q \cdot w_d$

Inverse Euclidean distance: $1/\|w_q - w_d\|$

Problem: both weigh longer documents more heavily

Cosine measure: $\dfrac{w_q \cdot w_d}{\|w_q\|\|w_d\|}$

Based on $X \cdot Y = \|X\|\|Y\| \cos\theta$

Querying using the Cosine Measure and Inverted Lists

Directly calculating cosine measure from query to every document is too expensive.

Use inverted file: keep (document,frequency) pairs in the posting lists

$\langle t; w_t; [(d_{t,1}, f_{d_{t,1},t}), (d_{t,2}, f_{d_{t,2},t}), \ldots, (d_{t,m}, f_{d_{t,m},t})] \rangle$

e.g. $\langle \text{apple}; .8; [(3,3), (6,1), (29,6)] \rangle$

Frequency counts can typically be compressed at least as well as distances.

Searching

Algorithm: Search(Q)

$A = \emptyset$ (Accumulators)

For each term $t \in Q$

$\langle t; w_t; P_t \rangle$ = Search lexicon for t

P_t = uncompress(P_t)

for each $(d, f_{d,t})$ in P_t

if $a_d \in A$

$a_d = a_d + w_t f_{d,t}$

else

$a_d = w_t f_{d,t}$

$A = A + \{a_d\}$

For each $a_d \in A$

$a_d = a_d / W_d$

Select k documents from A with largest a_d

Since $|A|$ can get large it can be pruned so only top $K \geq k$ documents are kept during search.

Relevance Feedback

Consider a sequence of queries $Q_1, Q_2, \ldots, Q_m$ in which R_i, I_i are the relevant and irrelevant documents returned by query i (typically marked by the user).

Dec Hi strategy:

$$Q_{i+1} = Q_i + \sum_{d \in R_i} D_d - D_n$$

More generally:

$$Q_{i+1} = \pi Q_0 + \omega Q_i + \alpha \sum_{d \in R_i} D_d + \beta \sum_{d \in I_i} D_d$$

For efficiency implementations can bound the number of nonzero entries in each iteration Q_i, or use clustering for the searches.

Clustering

Goals:

1. Speed up searches for complicated queries.
2. Find documents which are similar

There are many techniques for clustering, as well as many other applications.
We will talk about them next week.

Latent Semantic Indexing (LSI)

Goal: find documents that have similar content but not necessarily the same keywords. (*e.g.* query with *automobile* will pick up documents with *car* or perhaps *driver*)

Building Index:

1. Uses Singular Value Decomposition (SVD) on the matrix formed by all the term vectors.
2. Drop all but highest k rows of SVD.

Effectively reduces dimensionality of space to k dimensions.

Query:

1. Transform query to k-dimensional space using precomputed matrix (from SVD).
2. Find closest match to documents in reduced space, typically using the cosine measure.

Does not explicitly build clusters.

Singular value decomposition (SVD)

Given an $n \times m$ matrix A with $m \geq n$, SVD(A) is:

$$A = U\Sigma V^T$$

where $UU^T = VV^T = I$ and $\Sigma = \mathrm{diag}(\sigma_1, \cdots \sigma_n)$.

If organized such that $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_n$, then SVD is unique (except for equal σ).

Key property of SVD

Theorem: Let $A = U\Sigma V^T$ be the SVD of A. Then the matrix

$$A_k = \sum_{i=0}^{k} u_i \cdot \sigma_i \cdot v_i^T$$

is the closest matrix of rank k to A (measured as summed squared differences of the elements).

Some other properties of the SVD

$$A = U\Sigma V^T$$

- The columns of U are the orthonormal eigenvectors of AA^T and are referred to as the *left singular vectors.*
- The columns of V are the orthonormal eigenvectors of A^TA and are referred to as the *right singular vectors.*
- The diagonal values of Σ are the nonnegative square roots of the eigenvalues of AA_T, and are referred to as the *singular values.*

Calculating the SVD: Similar techniques as for calculating eigenvectors. For dense matrices will take $O(n^3)$ time.
For sparse matrices use Lanczos Algorithm.

SVD for Indexing

Each column of A is a document and each row is a term.

Generate U, Σ and V using SVD.

Drop all but the first k terms of each.

Documents now described in a k-dimensional instead of n dimensional space.

Queries: $\hat{q} = q^T U_k \Sigma_k^{-1}$
where q is the query and $\hat{q}$ is the query in transformed to the lower dimensional space.

Applications of LSI

Traditional indexing and searching

Cross language retrieval: Group words that have similar meaning in different languages.

- Include in the "training" databases documents which are composed of the same text in multiple languages appended together.
- Has been used for English-French, English-Japanese Kanji, and English-Greek .

Modeling Human memory: Trained on an encyclopedia and tested on the synonym test of the TOEFL exam. Did as well as average student.

Matching people: Used to assign reviewers to papers. Use papers written by potential reviewers for training.

Performance of LSI

TREC database

- Over 1,000,000 documents (AP News wire, Wall Street Journal, journal abstracts, U.S. Patents, ...)
- 200 queries with relevance judgments. Average of 300 relevant documents per query. Queries tend to be long (10-100 keywords). Relevance judgments were hand checked.

LSI Compared to best keyword searches

- 31% better on filtering tasks
- 16% better on retrieval tasks

Because of cost of SVD, index was built with a sample of around 70,000 documents instead of the full database.