Information Retrieval and Data Mining
Part 1 – Information Retrieval

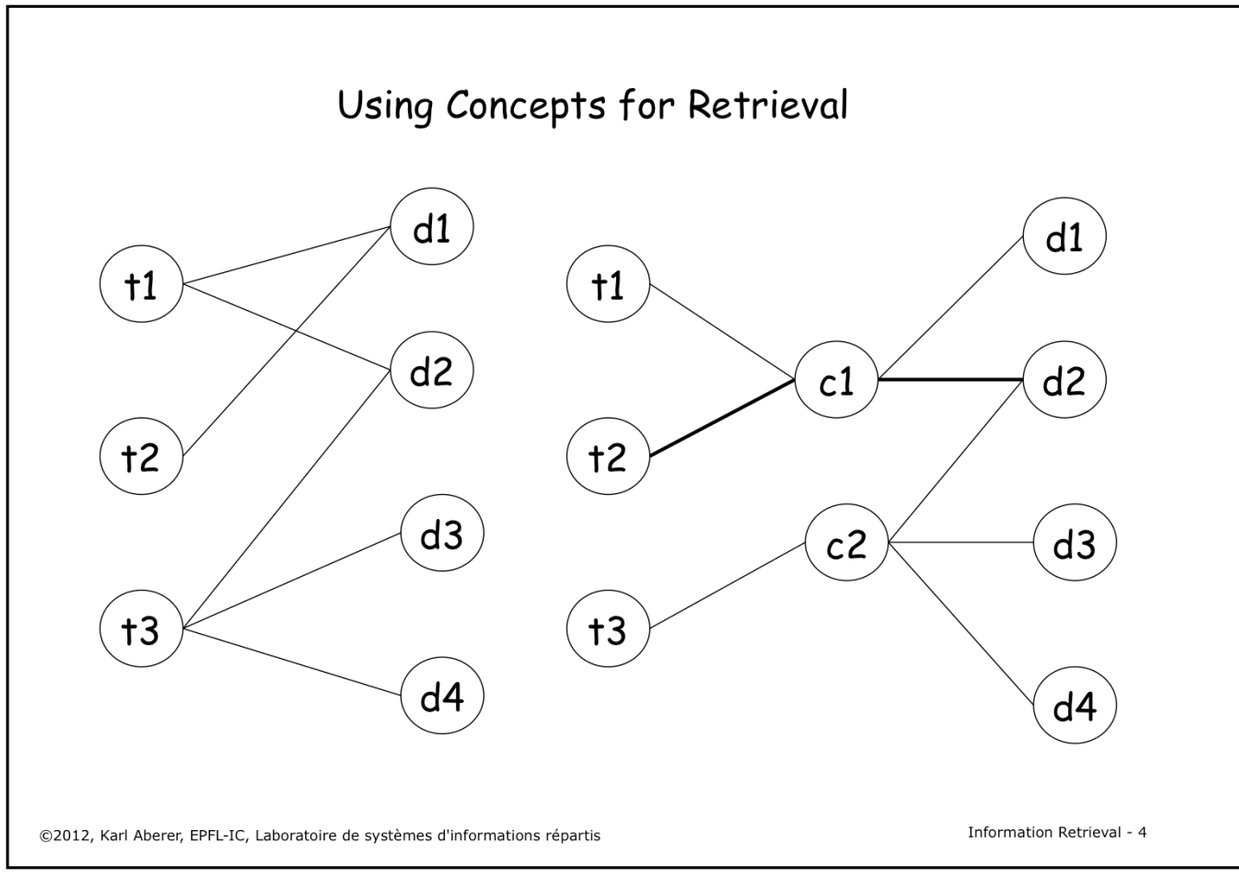
Today's Question

1. Information Retrieval
2. Text Retrieval Models
3. Latent Semantic Indexing
4. User Relevance Feedback
5. Inverted Files
6. Web Information Retrieval

3. Latent Semantic Indexing

- Vector Space Retrieval might lead to poor retrieval
  - Unrelated documents might be included in the answer set
  - Relevant documents that do not contain at least one index term are not retrieved
- Observation
  - retrieval based on index terms is vague and noisy
  - The user information need is more related to concepts and ideas than to index terms
- Key Idea
  - map documents and queries into a lower dimensional space composed of higher level concepts which are fewer in number than the index terms
- Dimensionality reduction
  - Retrieval (and clustering) in a reduced concept space might be superior to retrieval in the high-dimensional space of index terms

Despite its success and wide application the vector space retrieval model suffers some problems. On the one hand the presence of a specific term in a document not necessarily indicates it's relevance to a given query, on the other hand documents not containing any query term might nevertheless be related to the information need expressed by a query. These problems are related to the fact that the same concepts can be expressed through many different terms (synonyms) and that the same term may have multiple meanings (homonyms). Studies show that different users use the same keywords for expressing the same concepts only 20% of the time. Thus it would be an interesting idea to find methods using concepts for retrieval, instead of terms. To that end it is first necessary to define a "concept space" to which documents and queries are mapped, and compute similarity within that concept space. This idea is developed in the following.



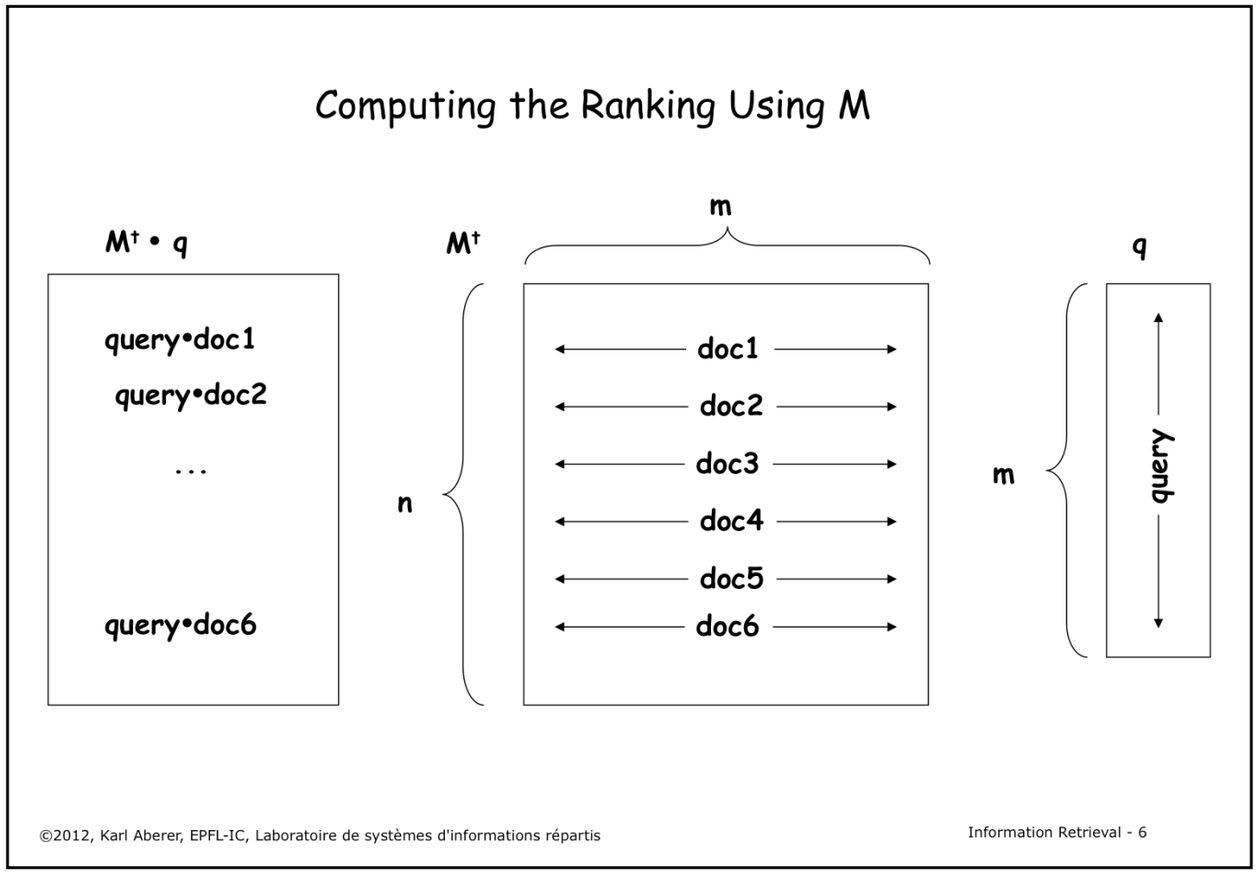
This figure illustrates the approach: rather than directly relating documents d and terms t, as in vector space retrieval, there exists an intermediate layer of concepts c to which both queries and documents are mapped. The concept space can be of a smaller dimension than the term space. In this small example we can imagine that the terms t1 and t2 are synonyms and thus related to the same concept c1. If now a query t2 is posed, in the standard vector space retrieval model only the document d1 would be returned, as it contains the term t2. By using the intermediate concept layer the query t2 would also return the document d2.

The question we want to explore in the following is of how such a concept space could be constructed. Possibilities are manually construction (similar to an ontology) or analyzing properties of natural language. Such methods are however not very practical. Rather we will show an approach that exploits the structure of the term-document matrix, which implicitly encodes the concepts and how to make these concepts explicit by applying matrix operations.

Basic Definitions

- Problem: how to identify and compute "concepts" ?
- Consider the term-document matrix
  - Let M_{ij} be a term-document matrix with m rows (terms) and n columns (documents)
  - To each element of this matrix is assigned a weight w_{ij} associated with k_i and d_j
  - The weight w_{ij} can be based on a tf-idf weighting scheme

The approach is based on the term-document matrix representation we already introduced for vector space retrieval. The weights in the matrix are generated according to the approach we have introduced for vector space retrieval.



This figure illustrates how the term-document matrix M is used to compute the ranking of the documents with respect to a query q when using the vector space retrieval model. The columns in M and q are normalized to 1.

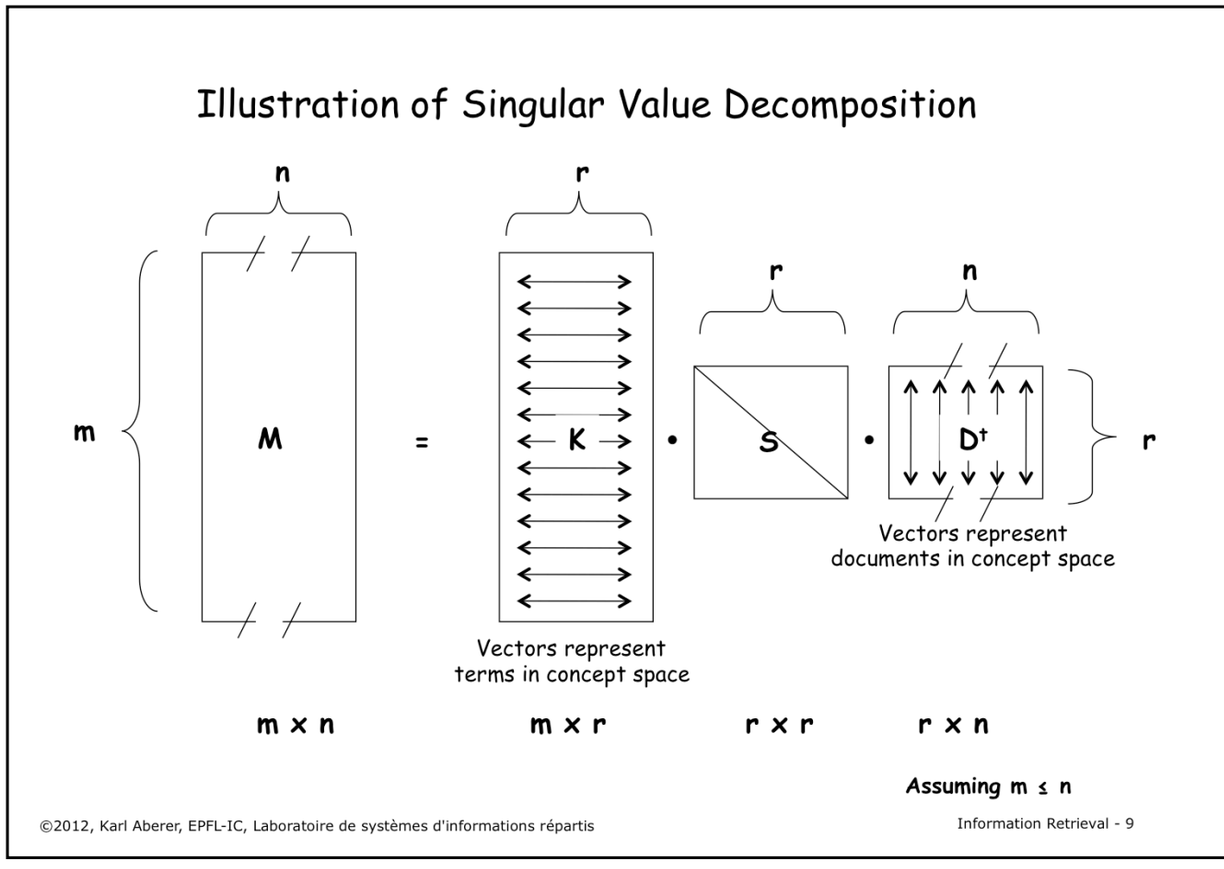
Question

- In vector space retrieval each row of the matrix M corresponds to
  1. A document
  2. A concept
  3. A query
  4. A query result

Singular Value Decomposition

- Key Idea: extract the essential features of M^t and approximate it by the most important ones
- Singular Value Decomposition (SVD)
  - $M = K \bullet S \bullet D^t$
  - K and D are matrices with orthonormal columns
  - S is an r x r diagonal matrix of the singular values sorted in decreasing order where r = min(m, n), i.e. the rank of M
  - Such a decomposition always exists and is unique (up to sign)
- Construction of SVD
  - K is the matrix of eigenvectors derived from $M \bullet M^t$
  - D^t is the matrix of eigenvectors derived from $M^t \bullet M$
  - Algorithms for constructing the SVD of a m× n matrix have complexity $O(n^3)$ if $m \approx n$

For extracting "concepts" a standard mathematical construction from linear algebra is used, the singular value decomposition (SVD). It decomposes a matrix into the product of three matrices. The middle matrix S is a diagonal matrix, where the elements of this matrix are the singular values of the matrix M. This decomposition can always be constructed in O(n^3). Note that the complexity is considerable, which makes the approach computationally expensive. There exist however also approximation techniques to perform this decomposition more efficiently.



This figure illustrates the structure of the matrices generated by the SVD. In general, m <=n, i.e. the number of documents may be larger than the size of the vocabulary (which is illustrated by th broken box).

Interpretation of SVD

- **First interpretation**: we can write M as sum of outer vector products

$$M = \sum_{i=1}^{r} s_i k_i \otimes d_i^t$$

- The s_i are ordered in decreasing size, thus by taking only the largest ones we obtain a good "approximation" of M
- **Second interpretation**: the singular values s_i are the lengths of the semi-axes of the hyperellipsoid E defined by

$$E = \left\{ Mx \mid \|x\|_2 = 1 \right\}$$

- Each value s_i corresponds to a dimension of a "concept space"
- **Third interpretation**: the SVD is a least square approximation

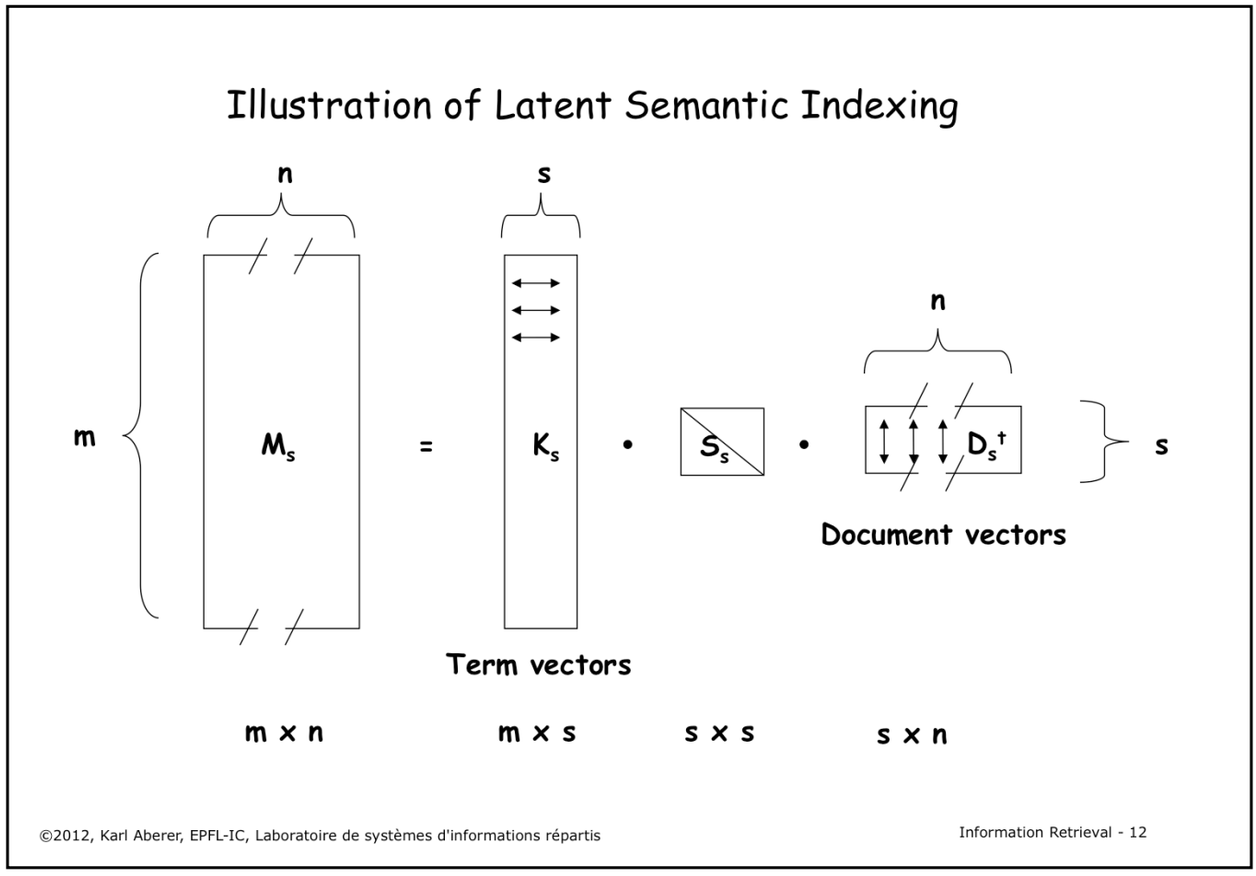
The singular value decomposition is a useful construction to reveal properties of a matrix. There exist several interpretations of which properties an SVD reveals:

1. We can write the original matrix M as the sum of components weighted by the singular values. Thus we can obtain approximations of the matrix by only considering the larger singular values. The symbol ⊗ denotes the **outer product** of two vectors.
2. The singular values have also a geometrical interpretation, as they tell us how a unit ball (||x||=1) is distorted when the linear transformation defined by the matrix M is applied to it. We can interpret the axes of the hyperellipsoid E as the dimensions of the concept space.
3. The SVD after eliminating less important dimensions (smaller singular values) can be interpreted as a least square approximation to the original matrix.

Latent Semantic Indexing

- In the matrix S, select only the s largest singular values
  - Keep the corresponding columns in K and D
- The resultant matrix is called M_s and is given by
  - $M_s = K_s \bullet S_s \bullet D_s^t$
    where s, $s < r$, is the dimensionality of the concept space
- The parameter s should be
  - large enough to allow fitting the characteristics of the data
  - small enough to filter out the non-relevant representational details

Using the singular value decomposition, we can now derive an "approximation" of M by taking only the s largest singular values in matrix S. The choice of s determines on how many of the "important concepts" the ranking will be based on. The assumption is that concepts with small singular value in S are rather to be considered as "noise" and thus can be neglected.



This figure illustrates the structure of the matrices after reducing the dimensionality of the concept space to s, when only the first s singular values are kept for the computation of the ranking. The columns in matrix K_s correspond to term vectors, whereas the columns in matrix D_s^t correspond to document vectors. By using the cosine similarity measure between columns of matrix D_s^t the similarity of the documents can be computed.

Questions

- Applying SVD to a term-document matrix M. Each concept is represented
  1. As a singular value
  2. As a linear combination of terms of the vocabulary
  3. As a linear combination of documents in the document collection
  4. As a least square approximation of the matrix M

- The number of term vectors in the SVD for LSI
  1. Is smaller than the number of rows in the matrix M
  2. Is the same as the number of rows in the matrix M
  3. Is larger than the number of rows in the matrix M

Answering Queries

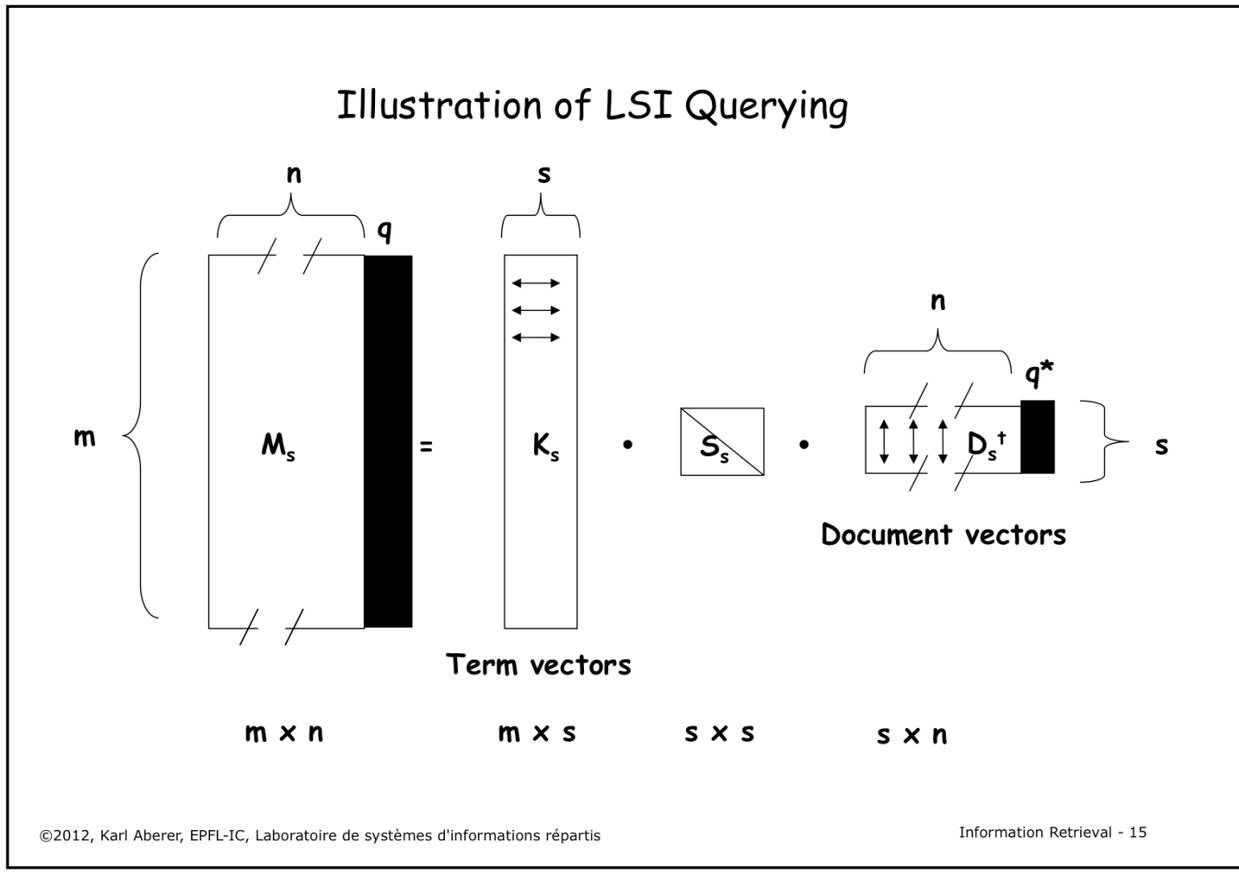
- Documents can be compared by computing cosine similarity in the "document space", i.e., comparing their rows d_i and d_j in matrix D_s^t
- A query q is treated like one further document
  - it is added like an additional column to matrix M
  - the same transformation is applied as for mapping M to D
- Mapping of M to D
  - $M = K \bullet S \bullet D^t \Rightarrow S^{-1} \bullet K^t \bullet M = D^t$ (since $K \bullet K^t = 1$) $\Rightarrow D = M^t \bullet K \bullet S^{-1}$
- Apply same transformation to q: $q^* = q^t \bullet K_s \bullet S_s^{-1}$
- Then compare transformed vector by using the standard cosine measure

$$sim(q^*, d_i) = \frac{q^* \bullet (D_s^t)_i}{|q^*| |(D_s^t)_i|}$$

After performing the SVD the similarity of different documents can be determined by taking the standard cosine similarity measure among their representation in the document representation in the concept space (the columns of matrix D_s^t). For answering queries, they are considered like documents that are added to the document collection. This construction works as follows: when a new column (the query) is added to M, we have to apply the same transformation to this new column, as to the other columns of M, to produce the corresponding column in matrix D_s^t, representing documents in the concept space. We can use the fact that $K_s^t \bullet K_s = 1$

Since $M_s = K_s \bullet S_s \bullet D_s^t$ we obtain $S_s^{-1} \bullet K_s^t \bullet M_s = D_s^t$ or $D_s = M_s^t \bullet K_s \bullet S_s^{-1}$

This is the transformation is applied to the query vector q to obtain a query vector q* in the concept space. After that step, the similarity of the query to the documents in the concept space can be computed. ($(D_s^t)_i$ denotes the i-th column of matrix D_s^t)



This figure illustrates of how a query vector is treated like an additional document vector.

Example (SVD, s=2)

K_s

$$\begin{pmatrix} -0.0154227 & -0.422647 \\ -0.0242622 & -0.382996 \\ -0.178994 & -0.196573 \\ -0.603612 & 0.0992276 \\ -0.668904 & 0.135237 \\ -0.0143585 & -0.354329 \\ -0.0123052 & -0.174656 \\ -0.0063823 & -0.0905044 \\ -0.150975 & 0.119194 \\ -0.0816699 & 0.0728611 \\ -0.150975 & 0.119194 \\ -0.178994 & -0.196573 \\ -0.142064 & 0.0983069 \\ -0.00711902 & -0.144923 \\ -0.0954645 & 0.0687813 \\ -0.203256 & -0.579569 \end{pmatrix}$$

S_s

$$\begin{pmatrix} 4.52655 & 0 \\ 0 & 2.74066 \end{pmatrix}$$

D_s

$$\begin{pmatrix} -0.147774 & 0.0493447 \\ -0.147774 & 0.0493447 \\ -0.0568424 & -0.634717 \\ -0.312508 & 0.12142 \\ -0.00481714 & -0.187237 \\ -0.0240726 & -0.0608051 \\ -0.00815196 & -0.336379 \\ -0.36892 & 0.197629 \\ -0.0391324 & 0.0516819 \\ -0.314476 & 0.129042 \\ -0.405113 & -0.26937 \\ -0.405113 & -0.26937 \\ -0.33055 & 0.148005 \\ -0.314476 & 0.129042 \\ -0.281123 & 0.0855504 \\ -0.00271846 & -0.0637277 \\ -0.0529817 & -0.414944 \end{pmatrix}$$

On the following pages we give a complete illustration of computing LSI for our running example. This is SVD for Term-Document Matrix from our running example.

Mapping of Query Vector into Document Space

q^* q^t K_s S_s^{-1}

$$\begin{pmatrix} -0.076442 & -0.605047 \end{pmatrix} = \begin{pmatrix} 0 \\ 2.14007 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1.44692 \end{pmatrix} \begin{pmatrix} -0.0154227 & -0.422647 \\ -0.0242622 & -0.382996 \\ -0.178994 & -0.196573 \\ -0.603612 & 0.0992276 \\ -0.668904 & 0.135237 \\ -0.0143585 & -0.354329 \\ -0.0123052 & -0.174656 \\ -0.0063823 & -0.0905044 \\ -0.150975 & 0.119194 \\ -0.0816699 & 0.0728611 \\ -0.150975 & 0.119194 \\ -0.178994 & -0.196573 \\ -0.142064 & 0.0983069 \\ -0.00711902 & -0.144923 \\ -0.0954645 & 0.0687813 \\ -0.203256 & -0.579569 \end{pmatrix} \begin{pmatrix} 0.220919 & 0. \\ 0. & 0.364876 \end{pmatrix}$$

(query "application theory")

In this step we comput the query vector on the concept space.

Ranked Result

s=2

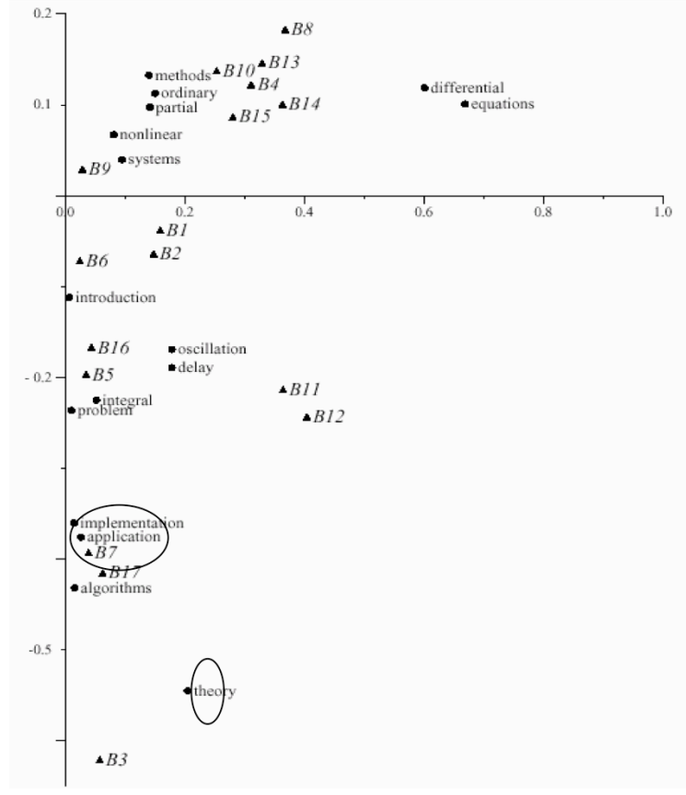
0.999999	{17}
0.999339	{3}
0.996554	{16}
0.995009	{5}
0.994859	{7}
0.968592	{6}
0.653706	{12}
0.653706	{11}
-0.168923	{15}
-0.19534	{1}

s=4

0.992173	{17}
0.970698	{16}
0.837632	{3}
0.537269	{12}
0.537269	{11}
0.434723	{7}
0.348928	{5}
-0.101838	{6}
-0.125203	{15}
-0.131291	{4}

This is the ranking produced for the query for different values of s.

Plot of Terms and Documents in 2-d Space



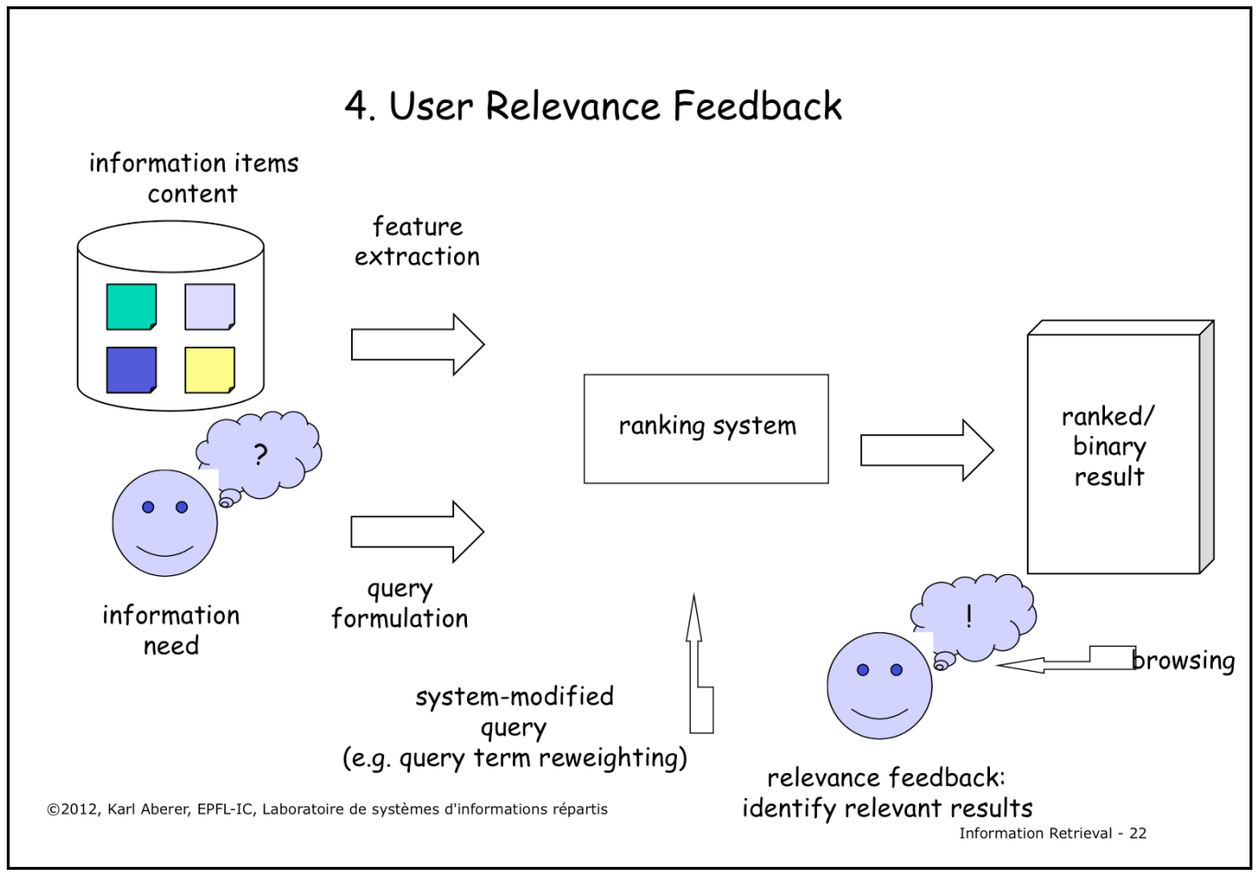
Since in our example the concept space has two dimensions, we can plot both the documents and the terms in this 2-dimensional space. It is interesting to observe of how semantically "close" terms and documents cluster in the same regions. This illustrates very well the power of latent semantic indexing in revealing the « essential concepts » in document collections.

Question

- A query transformed into the concept space for LSI has
  1. s components (number of singular values)
  2. m components (size of vocabulary)
  3. n components (number of documents)

Discussion of Latent Semantic Indexing

- Latent semantic indexing provides an interesting conceptualization of the IR problem
- Advantages
  - It allows reducing the complexity of the underline representational framework
  - For instance, with the purpose of interfacing with the user
- Disadvantages
  - Computationally expensive
  - Assumes normal distribution of terms (least squares), whereas term frequencies are a count



In general, a user does not necessarily know what his information need is and how to appropriately formulate a query. BUT usually a user can well identify relevant documents. Therefore the idea of user relevance feedback is to reformulate a query by taking into account feedback of the user on the relevance of already retrieved documents.

The advantages of such an approach are the following:

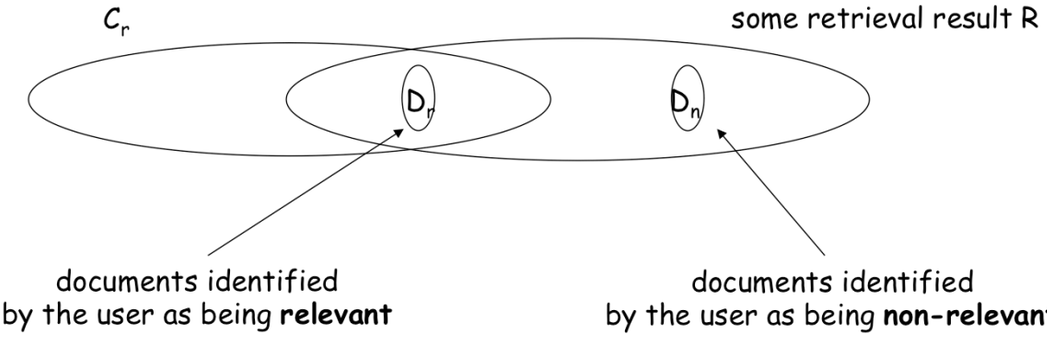
- The user is not involved in query formulation, but just points to interesting data items.
- The search task can be split up in smaller steps.
- The search task becomes a process converging to the desired result.

Identifying Relevant Documents

- If C_r is the set of relevant documents, then the optimal query vector is

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{n - |C_r|} \sum_{\vec{d}_j \notin C_r} \vec{d}_j$$

- Idea: approximate this vector by letting the user identify relevant and non-relevant documents



If we would know the exact set of relevant documents, the optimal query vector can be expressed in terms of the document vectors as shown above. Intuitively this expression can be well understood: it gives, proportional to the frequency of the documents, positive weights to document vectors in the set of relevant documents and negative weights for the others. The problem for practical retrieval is of course that we do not know the set of relevant documents C_r. However, with user relevance feedback we have the possibility to **approximate** this optimal query vector.

Expanded Query for Relevance Feedback

- If users identify some relevant documents D_r from the result set R of a retrieval query q
  - Assume all elements in $R \setminus D_r$ are not relevant, i.e. $D_n = R \setminus D_r$
  - Modify the query to approximate the theoretically optimal query (Rochio)

$$\vec{q}_{approx} = \alpha\vec{q} + \frac{\beta}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \frac{\gamma}{|R \setminus D_r|}\sum_{\vec{d}_j \notin D_r}\vec{d}_j$$

α, β, γ are tuning parameters

The best approximation for C_r that we can obtain, is by considering the set of documents D_r that the user indicated to be relevant, as the set of relevant documents. This set is used to modify the original query vector in a way that tries to approximate the optimal query vector. The scheme for doing this shown here is called Rochio scheme. There exist several variations of this scheme that however follow all the same principle. The tuning parameters are used to set the relative importance of

- keeping the original vector
- increasing the weight of vectors from D_r
- decreasing the weights from vectors of the complement of D_r

Example

$$\vec{q}_{approx} = \alpha\vec{q} + \frac{\beta}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \frac{\gamma}{|R \setminus D_r|}\sum_{\vec{d}_j \notin D_r}\vec{d}_j$$

- Query q= "application theory"
- Result

0.77: B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory
0.68: B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
0.23: B11 Oscillation Theory for Neutral Differential Equations with Delay
0.23: B12 Oscillation Theory of Delay Differential Equations

- Query reformulation

$$\vec{q}_{approx} = \frac{1}{4}\vec{q} + \frac{1}{4}\vec{d}_3 - \frac{1}{12}(\vec{d}_{17} + \vec{d}_{12} + \vec{d}_{11}), \alpha = \beta = \gamma = \frac{1}{4}$$

- Result for reformulated query

0.87: B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
0.61: B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory
0.29: B7 Knapsack Problems: Algorithms and Computer Implementations
0.23: B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra

This example shows how the query reformulation works. By identifying document B3 as being relevant and modifying the query vector it turns out that new documents (B5 and B7) become relevant. The reason is that those new documents share terms with document B3, and these terms are newly considered in the reformulated query.

Question

- Can documents which do not contain any keywords of the original query receive a positive similarity coefficient after relevance feedback ?
  1. No
  2. Yes, independent of the values β and γ
  3. Yes, but only if β > 0
  4. Yes, but only if γ > 0