8. PROBABILISTIC INFORMATION RETRIEVAL

Probabilistic Information Retrieval

The notion of similarity in the vector space model does not directly imply relevance

- The similarity values have no interpretation, they are just used to rank
- An information retrieval model deals with uncertainty on the users information needs
- Probability theory provides a principled approach to reason about this uncertainty

Probabilistic IR models attempt to directly model relevance as a probability

One of the key drawbacks of the vector space retrieval model is the lack of interpretability of the similarity values. This gave rise to the development of probabilistic retrieval models, that attempt to “compute” relevance as a probability.

Query Likelihood Model

Given query q, determine the probability $P(d|q)$ that document d is relevant to query q

Bayes Rule $P(d|q) = \frac{P(q|d)P(d)}{P(q)}$

Assumptions

- $P(d)$, the probability of a document occurring is uniform across a collection
- $P(q)$ is the same for all queries

Thus: $P(d|q)$ can be derived from $P(q|d)$, the query likelyhood

The problem of retrieval can be understood in a probabilistic setting as the problem of determining the probability of a document d being relevant, given a query q. We observe that the probability of a document to occur in a collection is constant (which makes sense assuming all documents are different), and the probability of a query to occur is the same for all documents. Thus, using Bayes rule, the problem of determining whether a document is relevant for a query is equivalent to the problem of determining whether a query is relevant to a document. The latter probability P(q|d) is also called the query likelihood.

Language Modeling

Query likelihood: determine $P(q|d)$

Assume each document d is generated by a Language Model M_d

- a language model is a mechanism that generates the words of the language

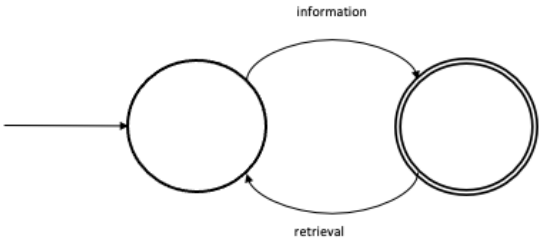
Then $P(q|d)$ can be interpreted as the probability that the query q was generated by the language model M_d

The notion of query likelihood gives now rise to the following approach to model relevance. We assume that documents are the result of language model. A language model is a (in general probabilistic) process that produces text, and a given document d is assumed to be produced by its specific language model M_d. Then the problem of retrieval can be viewed in the following way: if a query is relevant to a document, it should have been produced by the same language model as the document. Using this argument, the query likelihood corresponds to the probability that the query has been produced by the same language model as the document.

Let's have now a more detailed look in what a language model is and how we use it implement this intuitive model practically.

What is a Language Model?

Deterministic language model = automaton = grammar



This model can produce:

"information retrieval"

"information retrieval information retrieval"

It cannot produce:

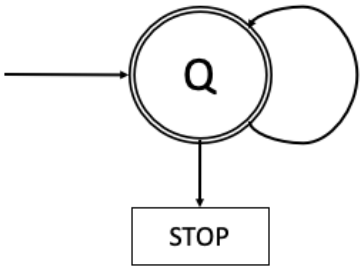
"retrieval information"

In the simplest case a language model is a deterministic automaton. In theoretical computer science deterministic automatons are those that can recognize or produce regular languages.

Probabilistic Language Model

Unigram model: assign a probability to each term to appear

- More complex models can be used, e.g., bigrams



Model M_1		Model M_2	
STOP	0.2	STOP	0.25
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.03	frog	0.0002
toad	0.03	toad	0.0001
said	0.02	said	0.01
likes	0.015	likes	0.01
dog	0.01	dog	0.04

Two different language models derived from 2 documents

Instead of using a deterministic automaton, we can also use a probabilistic state automaton, in other words, a Markov process. In the simplest case the automaton has a single state, and every state transition emits with a certain probability one term out of a vocabulary. In addition, the automaton can stop with a certain probability. The table captures the transition probabilities of two possible models M1 and M2. In the two models, the probability to stop is given as P(STOP|Q) = 0.2.

Probability to Create a Query

What is the probability that a query q has been generated by model M?

Example: q = the frog said dog STOP

$P(q|M_1)$ = 0.2 * 0.03 * 0.02 * 0.01 * 0.2 = 0.000 000 24

$P(q|M_2)$ = 0.15 * 0.0002 * 0.01 * 0.04 * 0.25 = 0.000 000 003

So retrieval becomes the problem of computing for a query q the probability $P(q|M_d)$ for all the documents d

Given a language model for the generation of documents, we can now compute within that model the probability that a given query q has been generated by the model of a document d. We give one example showing such a computation. With this approach we are now ready to compute query likelihood for all documents of a document collection.

Learning and Using the Model

Learning the model: Maximum Likelihood Estimation (MLE) of probabilities under Unigram Model

$$\hat{P}_{mle}(t|M_d) = \frac{tf_{t,d}}{L_d}$$

where

- $tf_{t,d}$ is the number of occurrences of t in d (term frequency)
- L_d is the number of terms in the document (document length)

Using the model (independence assumption)

$$\hat{P}(q|M_d) = \prod_{t \epsilon q} \hat{P}_{mle}(t|M_d)$$

For applying the probabilistic retrieval method described before, we need first to learn the language model of each document. The learning is performed using Maximum Likelihood Estimation (MLE). In the case of the unigram model, this is a straightforward task. We just estimate the term probabilities by counting the document frequencies and normalizing by document length. When using the model for a query q, we then use those estimates, to estimate the relevance of a query for the document, as illustrated before.

Consider the document:
"Information retrieval is the task of finding the documents satisfying the information needs of the user"

Using MLE to estimate the unigram probability model, what is $P(\text{the}|M_d)$ and $P(\text{information}|M_d)$?

1. 1/16 and 1/16
2. 1/12 and 1/12
3. 1/4 and 1/8 Correct
4. 1/3 and 1/6

Consider the following document

d = “information retrieval and search”

1. P(information search | M_d) > P(information | M_d)
2. P(information search | M_d) = P(information | M_d)
3. P(information search | M_d) < P(information | M_d) Correct

Issues with MLE

Problem 1: if the query contains a term not occurring in the document then $\hat{P}(q|M_d) = 0$!

Problem 2: this is an estimation! A term that occurs once, might have been "lucky", whereas another one with same probability to occur is not contained in the document

- need to give non-zero probability to unseen terms!

Applying the afore mentioned approach to estimate relevance of a document to a query has a practical problem: if the query contains a term not occurring in the document the estimated probability will be unavoidably zero, since one of the factors of the product computing that probability will be zero. In other words, the query cannot be generated by the document model, thus the document is not relevant to the query. This is not only impractical, but also not meaningful from a more theoretical perspective. Since we used MLE to generate the model, we were using the statistics of one specific document, that has been generated by a potentially complex model, that may contain other terms that just were not generated for this document.

Smoothing

Idea: add a small weight for non-occurring terms in a document, that is smaller than the normalized collection frequency

$$\hat{P}(t|M_c) \leq cf_t/T$$

where

- cf_t = number of times term t occurs in collection
- T = total number of terms in collection

Smoothed estimate

$$\hat{P}(t|d) = \lambda \hat{P}_{mle}(t|M_d) + (1-\lambda)\hat{P}_{mle}(t|M_c)$$

M_c = language model of the whole collection

λ = tuning parameter

To fix the aforementioned problem an approach called smoothing is applied. The basic idea is to assume that in fact every term potentially could occur in the document generated by its document model, including those that are not part of the actual document; only that the probability of terms not seen in the document is presumably less likely to occur as it would be expected to occur in the overall document collection. The smoothed estimate then combines the estimated likelihood to occur in the document according to the model generated from the document, with the estimated likelihood of a term occurring in the general document collection, modeled as a generic language model using the statistics from the document collection.

Probabilistic Retrieval

With smoothing the relevance is computed as

$$P(d|q) \propto P(d) \prod_{t \epsilon q} ((1-\lambda)P(t|M_c) + \lambda P(t|M_d))$$

From a technical perspective the probabilities are computed using term and document frequencies

- thus the same data is used as in vector space retrieval

Probabilistically motivated models show generally better performance

- But parameter tuning (λ) is critical
- λ can be query-dependent, e.g., query size

Here we summarize the approach for probabilistic retrieval. From a more technical perspective computational cost of probabilistic retrieval is not very different from vector space retrieval. The computation of the likelihoods for the document models requires determination of term frequencies, so in that sense it is equivalent. For the collection models the global term frequencies need to be computed, which again is similar to computing inverse document frequencies in a document collection.

In practice, the fine tuning of the model parameters (in that case λ) is essential for that the model performs well. Different methods have been devised for that. It is also possible to make the parameters dependent on the query, in particular on the query size.

Example

Collection consisting of d_1 and d_2

d_1: Einstein was one of the greatest scientists

d_2: Albert Einstein received the Nobel prize

Query q: Albert Einstein

Using λ=1/2:

$P(q|d_1) = ½ * (0/7 + 1/13) * ½ * (1/7 + 2/13) \approx 0.0057$

$P(q|d_2) = ½ * (1/6 + 1/13) * ½ * (1/6 + 2/13) \approx 0.0195$

Albert Einstein

This is a simple example illustrating the use of probabilistic retrieval. Notate that the document lengths of d1 and d2 are 7 and 6, and that the collection length is 13.

Example: Comparing VS and PR

	Precision			
Rec.	tf-idf	LM	%chg	
0.0	0.7439	0.7590	+2.0	
0.1	0.4521	0.4910	+8.6	
0.2	0.3514	0.4045	+15.1	*
0.3	0.2761	0.3342	+21.0	*
0.4	0.2093	0.2572	+22.9	*
0.5	0.1558	0.2061	+32.3	*
0.6	0.1024	0.1405	+37.1	*
0.7	0.0451	0.0760	+68.7	*
0.8	0.0160	0.0432	+169.6	*
0.9	0.0033	0.0063	+89.3	
1.0	0.0028	0.0050	+76.9	
Ave	0.1868	0.2233	+19.55	*

Ponte & Croft, 1998

This is a result reported from comparing vector space retrieval with probabilistic retrieval. It shows that in this experiment probabilistic retrieval improves precision significantly, in particular for higher values of recall. (LM = language model).

Overview of Retrieval Model Properties

	Vector Space Model	Language Model	BM25 (another prob. Model)
Model	geometric	probabilistic	probabilistic
Length normalization	Requires extensions (pivot normalization)	Inherent to model	Tuning parameters
Inverse document frequency	Used directly	Smoothing and collection frequency has similar effect	Used directly
Multiple term occurrences	Taken into account	Taken into account	Ignored
Simplicity	No tuning required	Tuning essential	Tuning essential

Here we compare the characteristics of the vector space model with the probabilistic retrieval model based on language models, and BM25 another model based on a probabilistic approach, that is today considered as one of the most performant retrieval models.

One aspect that is taken implicitly care off in the probabilistic retrieval model based on language models is normalization for document length. For vector space retrieval specific extensions have been developed, that modify the weighting parameters with the document length. For collections with widely varying document lengths this proved to be a useful improvement. In general, the vector space model is preferred when a quick and simple solution is sought. For probabilistic models better performance can be achieved, but this depends on careful parameter tuning which requires specialized expertise.

9. LATENT SEMANTIC INDEXING

Latent Semantic Indexing

Vector space retrieval is vague and noisy

- Based on index terms
- Unrelated documents might be included in the answer set
  - apple (company) vs. apple (fruit)
- Relevant documents that do not contain at least one index term are not retrieved
  - car vs. automobile

Observation

- The user information need is more related to concepts and ideas than to index terms

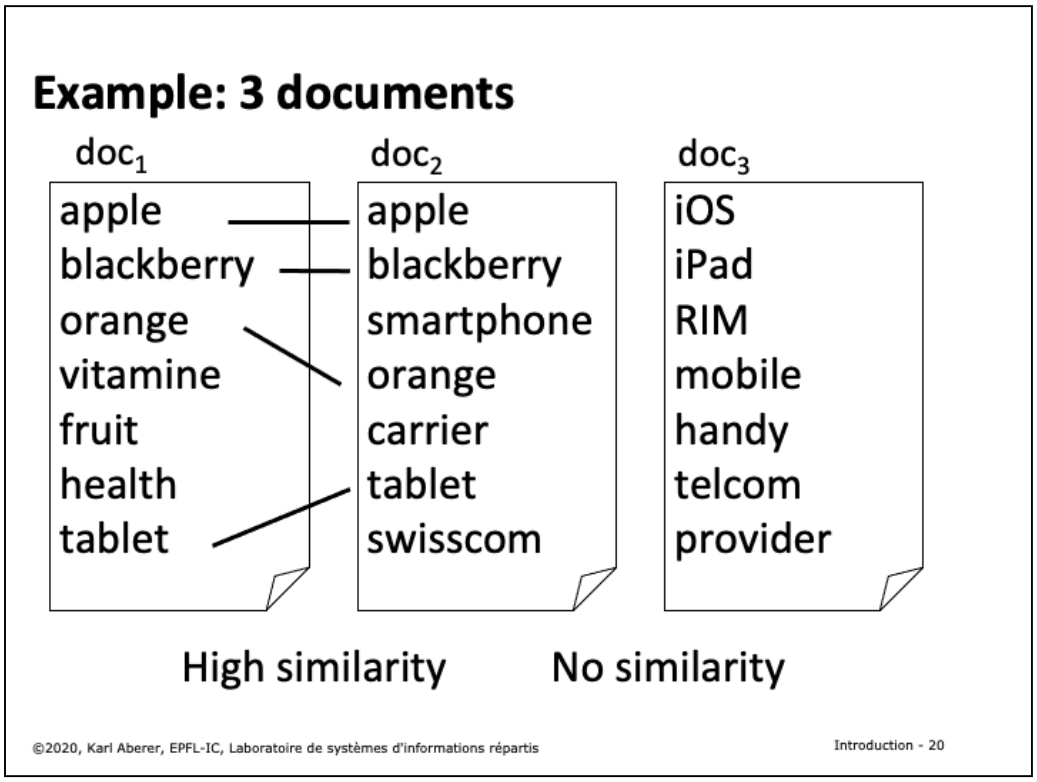
Despite its success and widespread use the vector space retrieval model suffers from some problems. The important insight is that terms may indicate a concept a user is interested in, but there does not necessarily exist a one to one correspondence between terms and concepts. As a consequence retrieval results may contain irrelevant documents, and relevant documents may be missed.

The Problem

Vector Space Retrieval handles poorly the following two situations

1. *Synonymy*: different terms refer to the same concept, e.g. car and automobile
   - Result: poor recall
2. *Homonymy*: the same term may have different meanings, e.g. apple, model, bank
   - Result: poor precision

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.



Let's illustrate the problem resulting from synomyms and homonyms by an example. Among these three documents at the level of terms doc1 and doc2 are (seem) highly related, whereas doc3 has no similarity with the other two documents. With human background knowledge it is however easy to see that in reality doc2 and doc3 are closely related, as they talk about mobile communications, whereas doc1 is completely unrelated to the others as it is about health and nutrition.

Key Idea

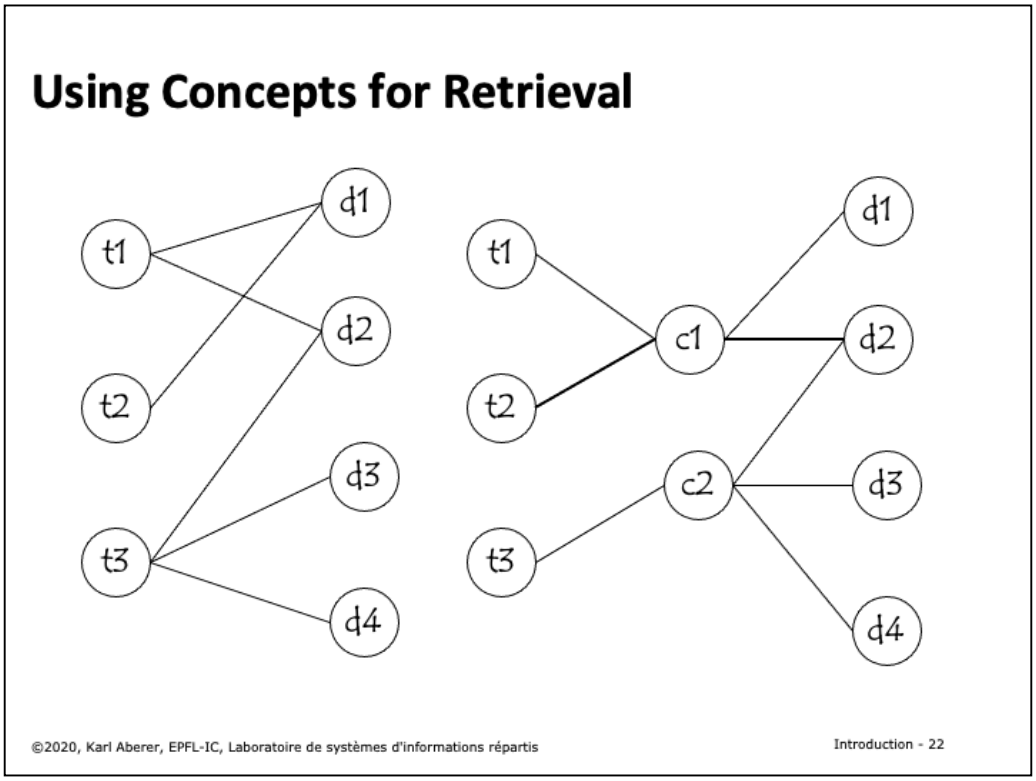
Map documents and queries into a lower-dimensional space composed of higher-level concepts

- Each concept represented by a combination of terms
- Fewer concepts than terms
- e.g. vehicle = {car, automobile, wheels, auto car, motor car}

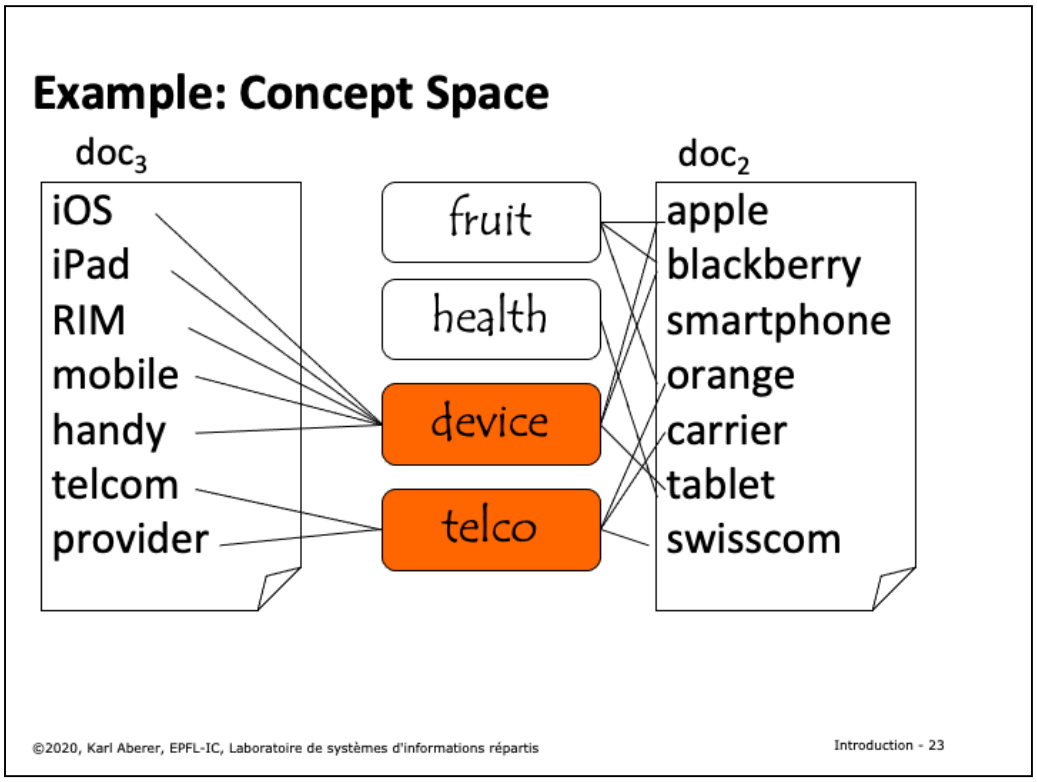
Dimensionality reduction

- Retrieval (and clustering) in a reduced concept space might be superior to retrieval in the high-dimensional space of index terms

Thus it would be an interesting to base information retrieval methods on the use of concepts, 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. The concept space should ideally have much lower dimension than the term space, whose dimensionality is determined by the size of the vocabulary.



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.



Applying this idea to our example before, we can imagine to have a concept space consisting of four concepts, two related to health, two related to mobile communication. When we consider now doc 2 and doc3 we can already recognize much better the close conceptual relationship the two documents have.

Similarity Computation in Concept Space

Concept represented by terms, e.g.

device = {iOS, iPad, RIM, mobile, handy, tablet, apple, blackberry}

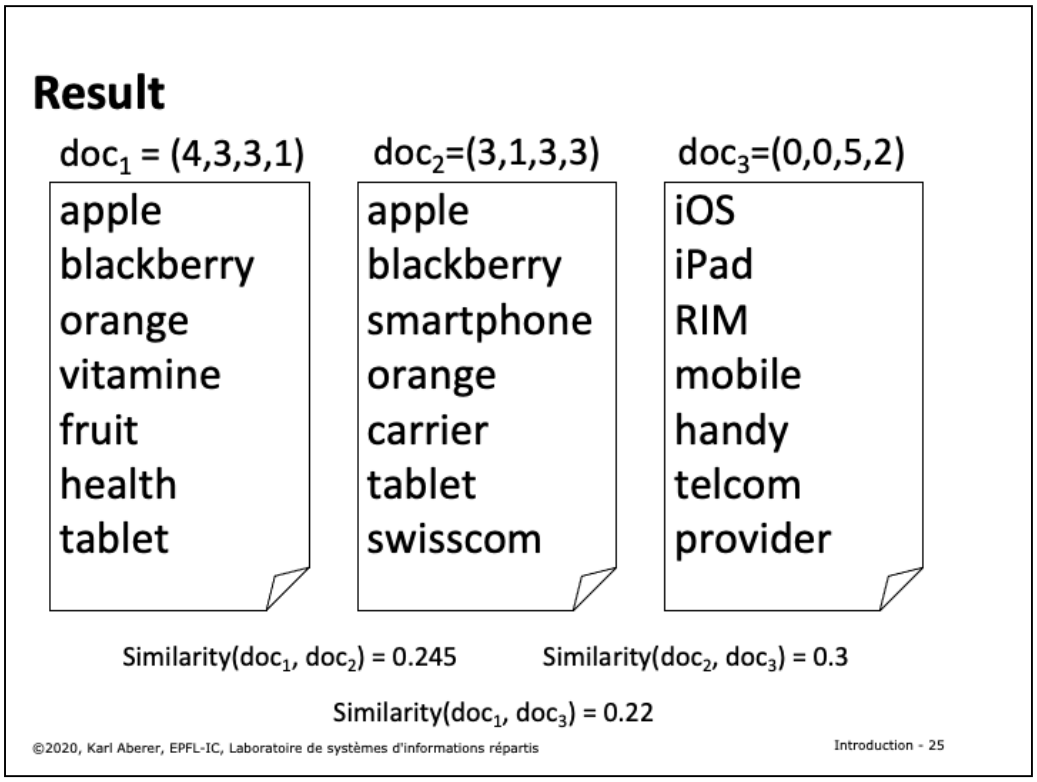
Document represented by concept vector, counting number of concept terms, e.g.

$doc_1 = (4, 3, 3, 1)$

$doc_3 = (0, 0, 5, 2)$

Similarity computed by scalar product of normalized concept vectors

We may consider concepts as being represented by sets of terms, and documents by concept vectors that count how many concept terms occur in the document. Using this approach we would obtain the non-normalized concept vectors doc1 = (4,3,3,1), doc2=(3,1,3,3) and doc3=(0,0,5,2).



After normalizing these vectors we compute the cosine similarities among the resulting concept vectors and obtain:

Sim (2,3) = 0.3

Sim (1,2) = 0.245

Sim (1,3) = 0.22

This result shows that indeed documents 2 and 3 are the more related ones, though still some confusion remains due to the high number of synonyms occurring in these documents.

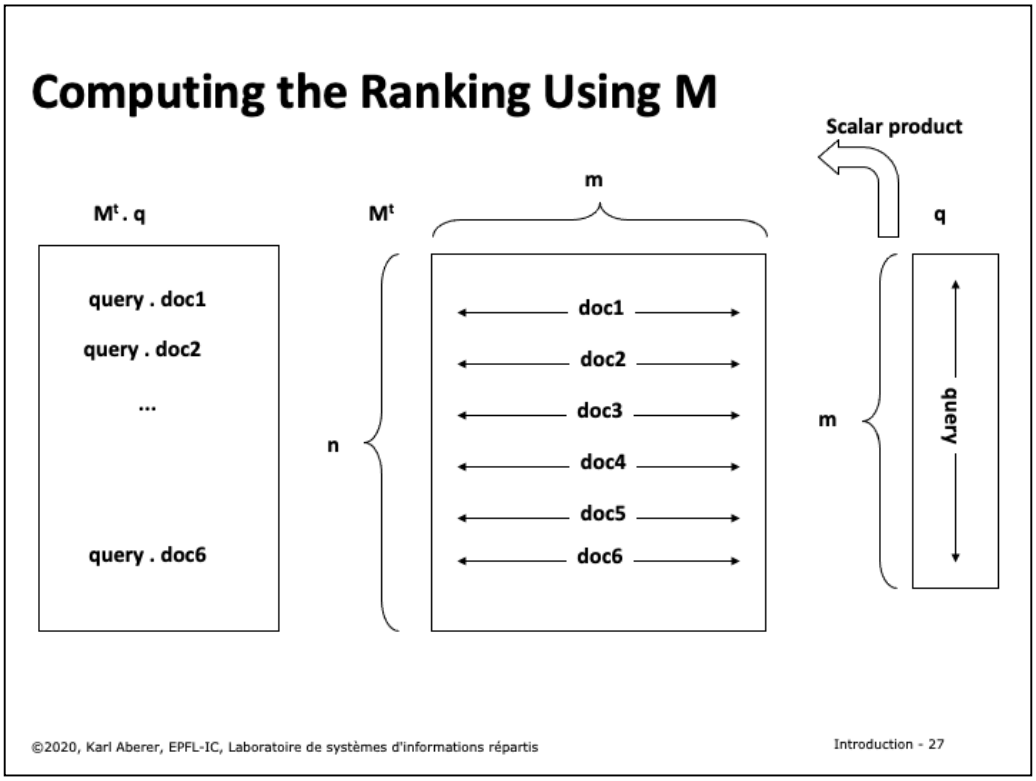
Basic Definitions

Problem: how to identify and compute "concepts" ?

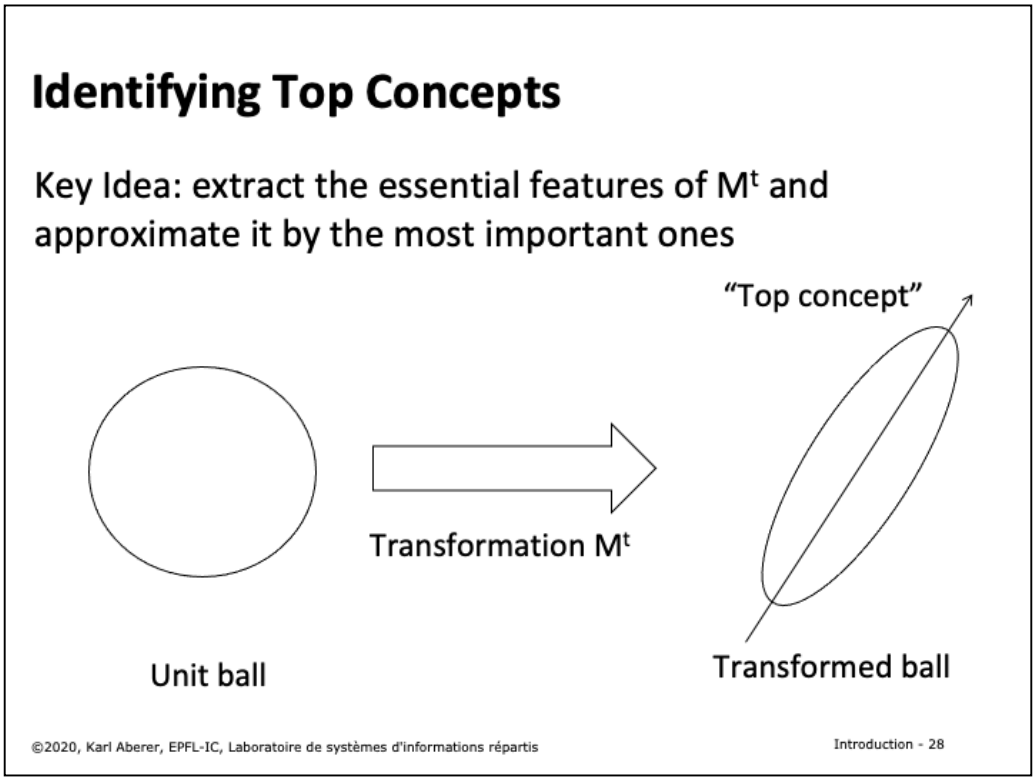
Consider the term-document matrix

- 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 t_i and d_j
- The weight w_{ij} can be based on a tf-idf weighting scheme

The problem is to identify a method that identifies and characterizes important concepts in document collections. One approach would be to perform this task manually, e.g. by using a predefined ontology and let users annotate documents using terms of the ontology. This is an approach that his been used in libraries, but is labor intensive. Thus we will now present a method that performs the task of concept identification and document classification by concepts automatically. Starting point for the method is the term-document matrix that we have introduced for vector space retrieval with weigths based on a tf-idf weighting scheme.



We can understand the process of producing a retrieval results in vector space retrieval as a matrix operation. This is illustrated in this figure. The ranking is the result of computing the product of a query vector q with the term-document matrix M. We assume that all columns in M and q are normalized to 1.



One way to understand of how concepts can be extracted from a document collection is to consider the effect of the term-document matric on data. If we apply this matrix to a (high-dimensional) unit ball it will distort this ball into an ellipsoid. This ellipsoid will have one direction with the strongest distortion. We may think of this direction as corresponding to a particularly important concept of the document collection.

Singular Value Decomposition (SVD)

Represent Matrix M as $M = K.S.D^t$

- K and D are matrices with orthonormal columns

$$K.K^t = I = D.D^t$$

- 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)

To extract this particularly important directions of a matrix mapping, a standard mathematical construction from linear algebra is used, the singular value decomposition (SVD). SVD 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.

Construction of SVD

K is the matrix of eigenvectors derived from $M.M^t$

D is the matrix of eigenvectors derived from $M^t.M$

Algorithms for constructing the SVD of a m x n matrix have complexity $O(n^3)$ if $m \leq n$

Formally the SVD can be computed by constructing eigenvectors of matrices derived from the original matrix M. This computationg can be performed 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.

Interpretation of SVD

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

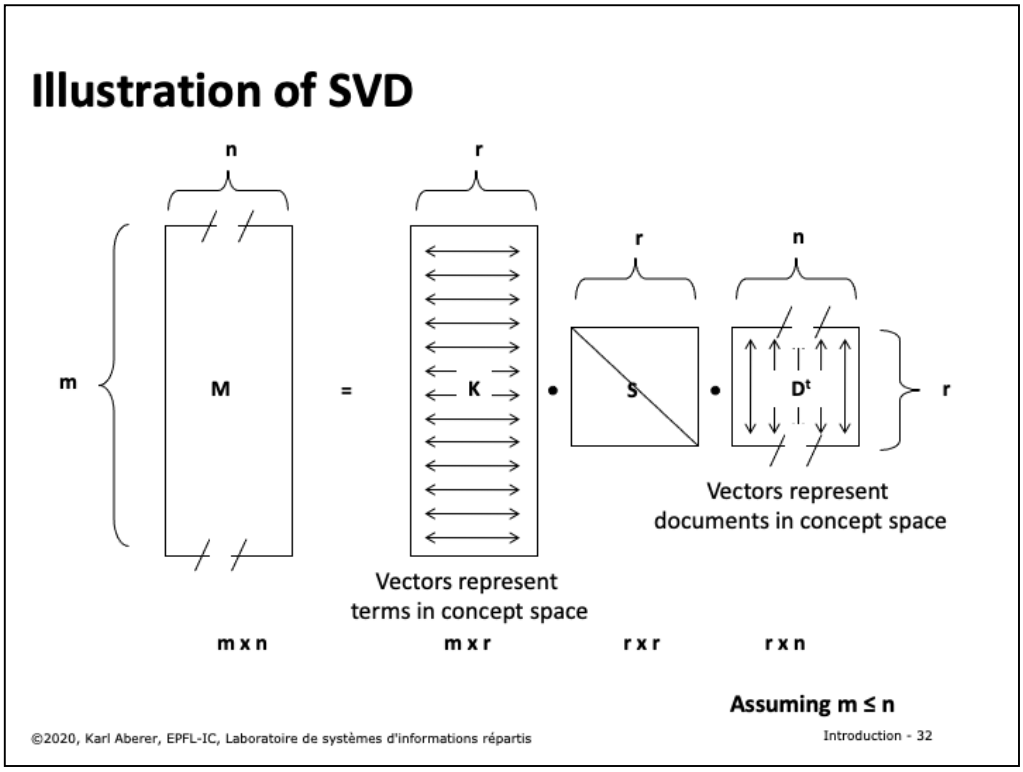
By taking only the largest ones we obtain a «good» approximation of M (least square approximation)

The singular values s_i are the lengths of the semi-axes of the hyperellipsoid E defined by

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

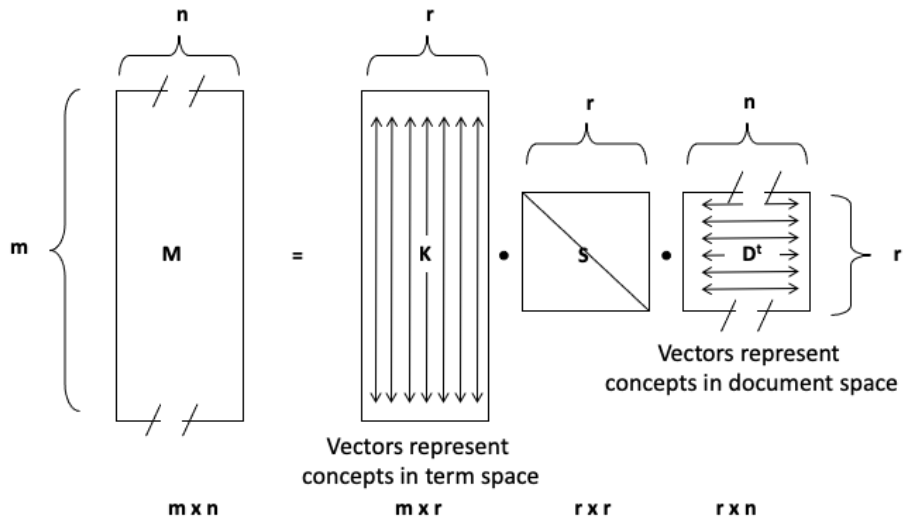
One way to understand of how the SVD extracts the important concepts from the term-document matrix is the following: the decomposition can be used to rewrite the original matrix as the sum of components that are weighted by the singular values. Thus we can obtain approximations of the matrix by only considering the larger singular values. The SVD after eliminating less important dimensions (smaller singular values) can be interpreted as a least square approximation to the original matrix. The symbol ⊗ denotes the **outer product** of two vectors, d_i is the i-th row of D.

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.



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. The rows in K can then be interpreted as the representation of terms in the concept space, and the rows in D as the representation of documents in the concept space.

Illustration of SVD – Another Perspective



We can also see of how the concepts are represented by terms respectively documents. In particular, each concepts is now described as a weighted vector of terms.

Latent Semantic Indexing (LSI)

In the matrix S, select only the s largest singular values

- Keep the corresponding columns in K and D

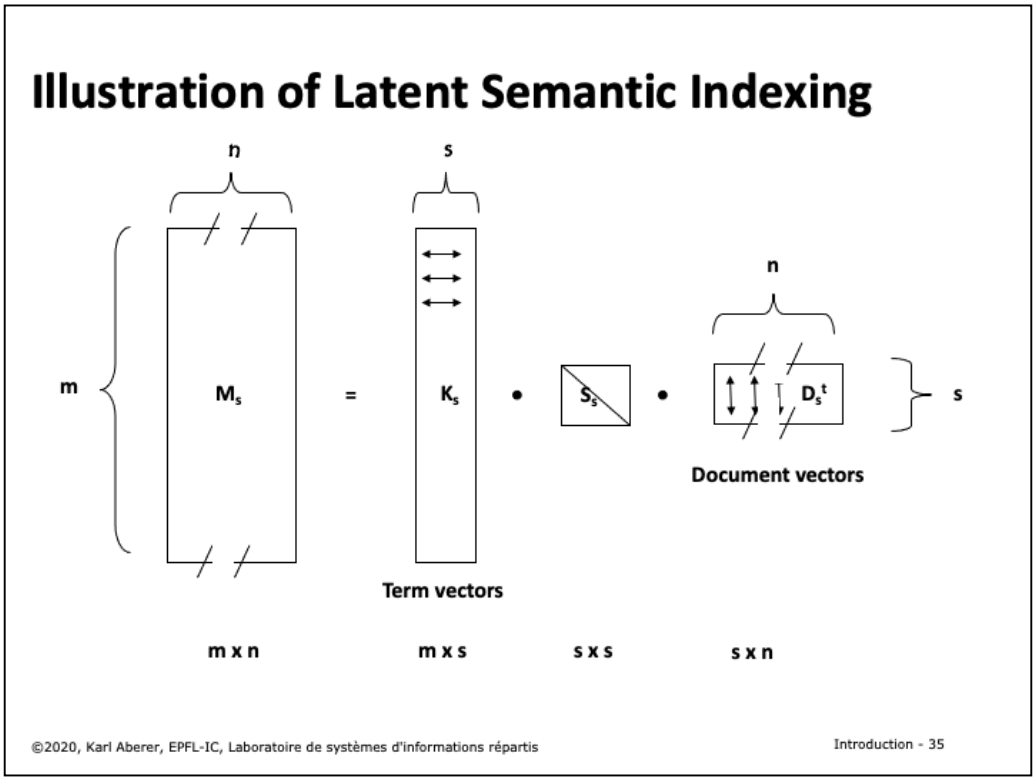
The resultant matrix is called M_s and is given by

- $M_s = K_s.S_s.D_s^t$ 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. The resulting method is called Latent Semantic Indexing.



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 rows 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 documents can be computed.

Answering Queries

Documents can be compared by computing cosine similarity in the concept space, i.e., comparing their columns $(D_s^t)_i$ and $(D_s^t)_j$ in matrix D_s^t

A query q is treated like one further document

- it is added as an additional column to matrix M
- the same transformation is applied to this column as for mapping M to D

After performing the SVD the similarity of different documents can be determined by computing the cosine similarity measure among their representation in the concept space (the columns of matrix D_s^t). Queries are considered like documents that are added to the document collection. Answering queries then corresponds then to computing the similarity between the query considered as a document and the documents in the collection.

Mapping Queries

Mapping of M to D

$M = K.S.D^t$

$S^{-1}.K^t.M = D^t$ (since $K.K^t = 1$)

$D = M^t.K.S^{-1}$

Apply same transformation to q:

$q^* = q^t.K_s.S_s^{-1}$

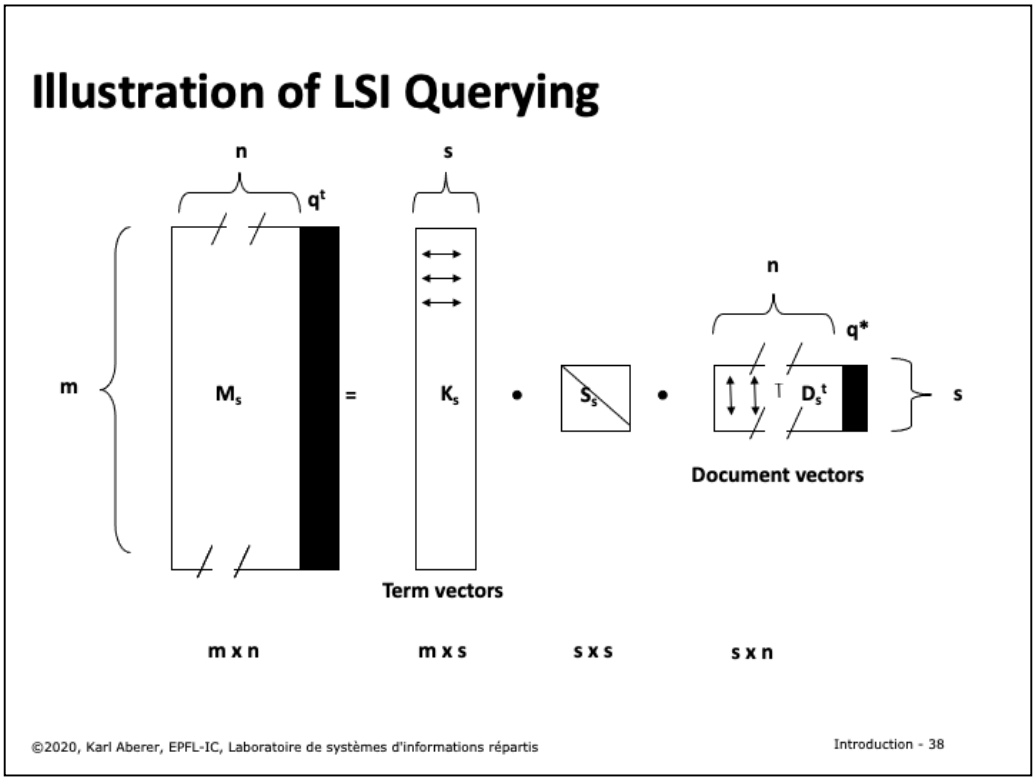
Then compare transformed vector by using the standard cosine measure

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

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, in order to produce the corresponding column in the matrix D_s^t, representing documents in the concept space. We exploit the fact that $K_s^t.K_s = 1$.

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

This is the transformation that 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: Documents

B1 A Course on Integral Equations
B2 Attractors for Semigroups and Evolution Equations
B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4 Geometrical Aspects of Partial Differential Equations
B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6 Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7 Knapsack Problems: Algorithms and Computer Implementations
B8 Methods of Solving Singular Systems of Ordinary Differential Equations
B9 Nonlinear Systems
B10 Ordinary Differential Equations
B11 Oscillation Theory for Neutral Differential Equations with Delay
B12 Oscillation Theory of Delay Differential Equations
B13 Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14 Sinc Methods for Quadrature and Differential Equations
B15 Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16 The Boundary Integral Approach to Static and Dynamic Contact Problems
B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

This is an example of a (simple) document collection that we will use in the following as running example.

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

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

In the following 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 Concept 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 compute the query vector in the concept space.

Ranked Result

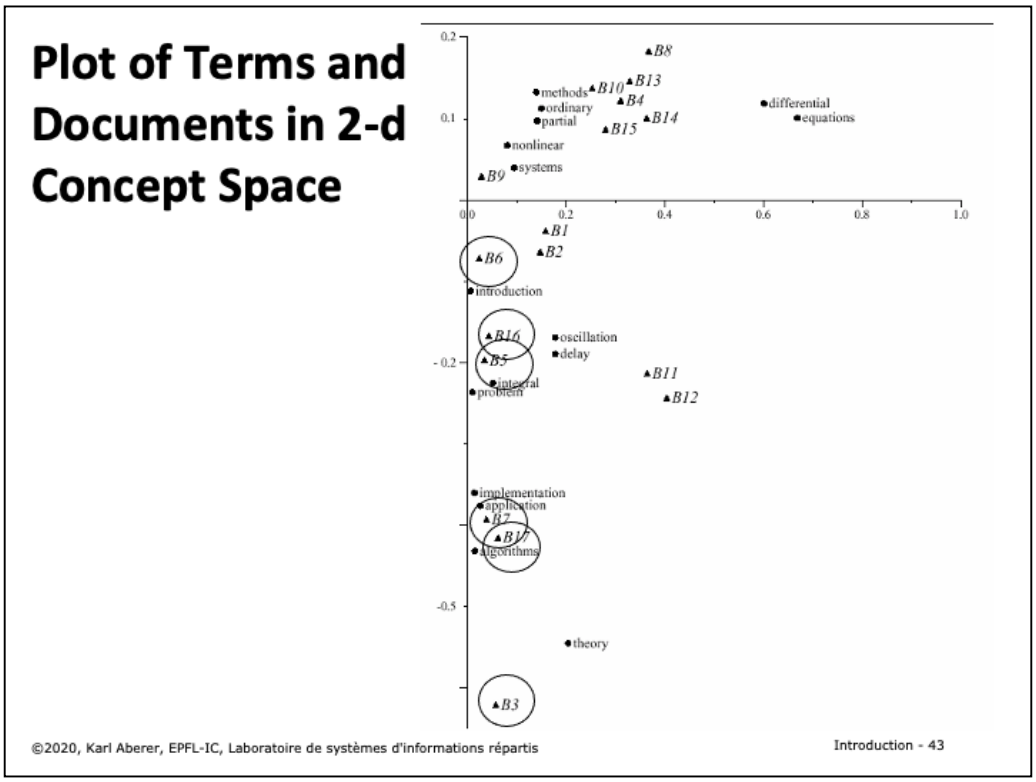
s=2

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.



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.

In vector space retrieval each row of the matrix M corresponds to

A. A document
B. A concept
C. A query
D. A term Correct

Applying SVD to a term-document matrix M. Each concept is represented in K

A. as a singular value
B. as a linear combination of terms of the vocabulary Correct
C. as a linear combination of documents in the document collection
D. as a least squares approximation of the matrix M

The number of term vectors in the matrix K_s used for LSI

A. Is smaller than the number of rows in the matrix M
B. Is the same as the number of rows in the matrix M Correct
C. Is larger than the number of rows in the matrix M

A query transformed into the concept space for LSI has …

A. s components (number of singular values) Correct
B. m components (size of vocabulary)
C. 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 underlying concept representation
- Facilitates interfacing with the user

Disadvantages

- Computationally expensive
- Poor statistical explanation

From a modelling perspective LSI suffers from poor explanatory power. For example, the least squares approximation concept is related to the assumption that term frequencies are normally distributed, which is in contradiction with the observation that term frequencies are power law distributed.

Alternative Techniques

Probabilistic Latent Semantic Analysis

- Based on Bayesian Networks

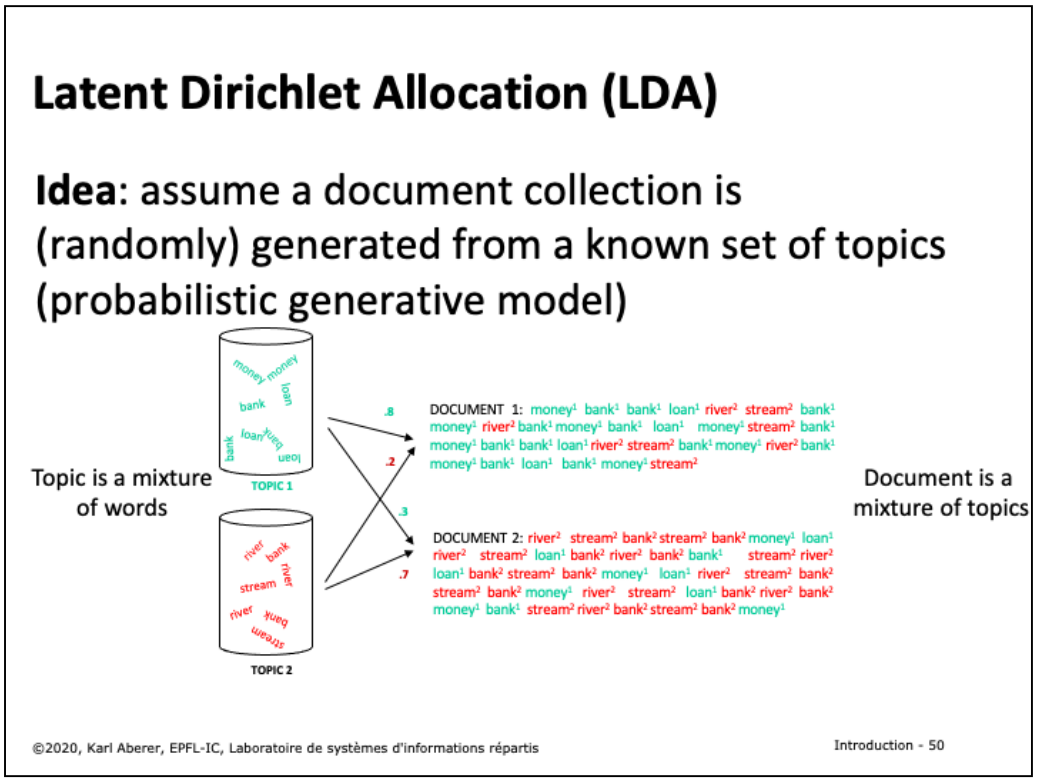
Latent Dirichlet Allocation

- Based on Dirichlet Distribution
- State-of-the-art method for concept extraction

Same objective of creating a lower-dimensional concept space based on the term-document matrix

- Better explained mathematical foundation
- Better experimental results

The conceptual problems of LSI have triggered significant efforts in developing better suited models for concept extraction. The most recent and successful result of this has been the development of a method based on the use of Dirichlet distributions. Like LSI it produces a concept space, where concepts are represented as term vectors. The method has better theoretical foundations and empirically it produces better results, and is nowadays considered as the golden standard. The approach is mathematically more involved than LSI, and therefore we will not be able to develop this method in this course.



This illustration shows the basic idea underlying LDA: it is assumed that there exist topics that are represented as mixtures of words. In the example we see words that are related to two meanings of “bank”, the financial institution and the river bank, and are represented by different related works. Then, documents are supposed to be generated from a mixture of topics, where the mixture is defined by some weights, as indicated in the figure. As a result each document contains terms form its associated topics in the right proportion.

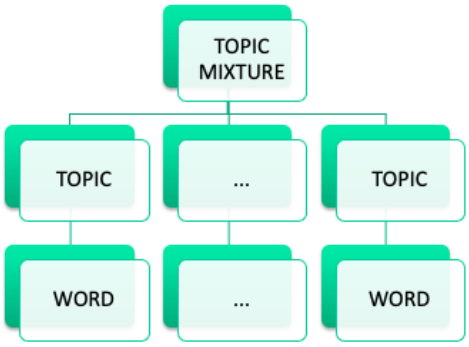
Similar to the probabilistic language model used in information retrieval, we have here another example of a probabilistic generative model.

Document Generation using a Probabilistic Process

For each document, choose a mixture of topics

For every word position, sample a topic from the topic mixture

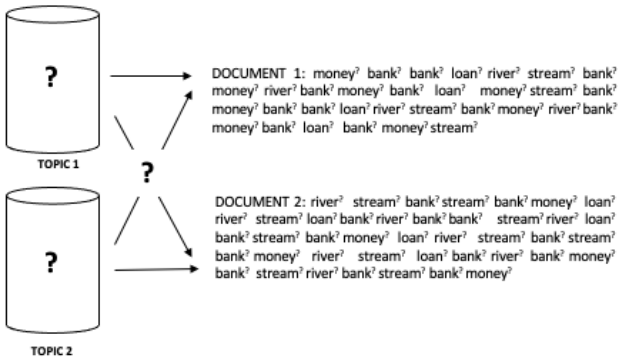
For every word position, sample a word from the chosen topic



Given the underlying model we can define a probabilistic process for generating documents.

LDA: Topic Identification

Approach: Inverting the process: given a document collection, reconstruct the topic model



The challenge is to determine the probabilistic model. The situation we consider in practice is that a set of documents is given, and we intend to reconstruct the (most likely) probabilistic process that has generated this document collection. This is a difficult problem to solve.

Latent Dirichlet Allocation

Topics are **interpretable** unlike the arbitrary dimensions of LSI

Distributions follow a Dirichlet distribution

Construction of topic model is mathematically involved, but computationally feasible

Considered as the state-of-the art method for topic identification

We do not present the details of LDA here, as the method is mathematically fairly involved. However, it is important to note that it is considered today as the state-of-the-art method of topic detection.

Use of Topic Models

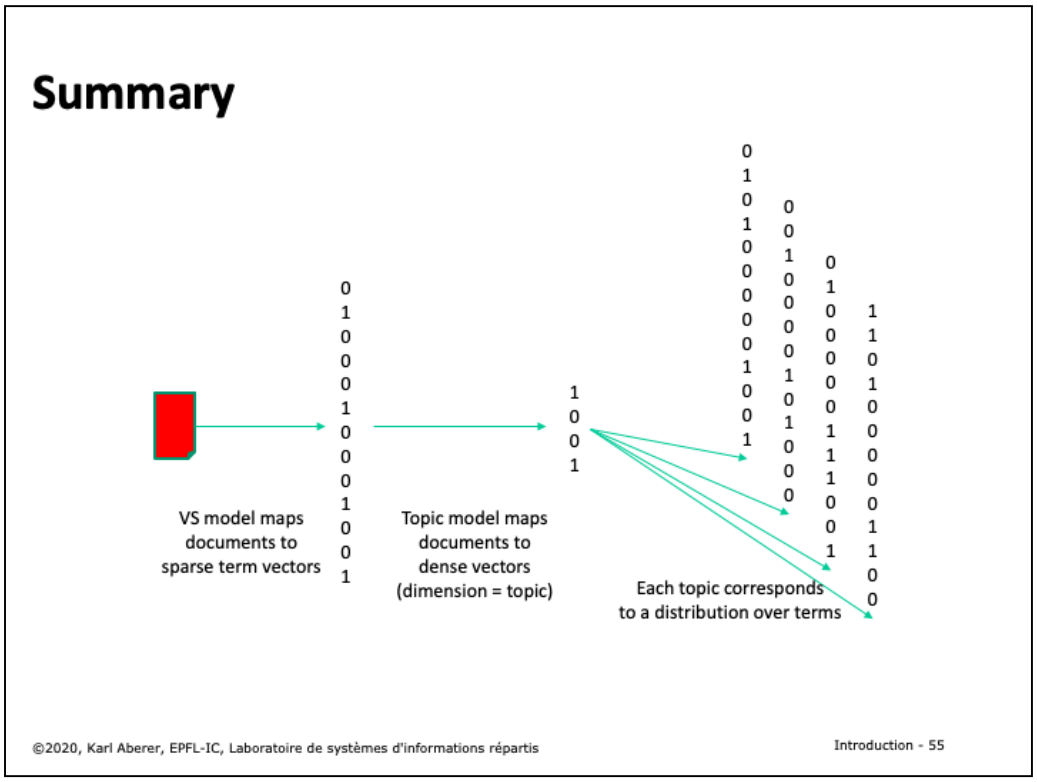
Unsupervised Learning of topics

- Understanding main topics of a topic collection
- Organizing the document collection

Use for document retrieval: use topic vectors instead of term vectors to represent documents and queries

Document classification (Supervised Learning): use topics as features

Topic models provide a representation of documents at a conceptual level. Therefore they are applied in many different contexts, including document retrieval and classification.



This illustration summarizes the connection among the vector space model, which was introduced for information retrieval and topic models that produce low-dimensional (dense) representation of documents.