Information Retrieval: Advanced Models for Text Representation

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.

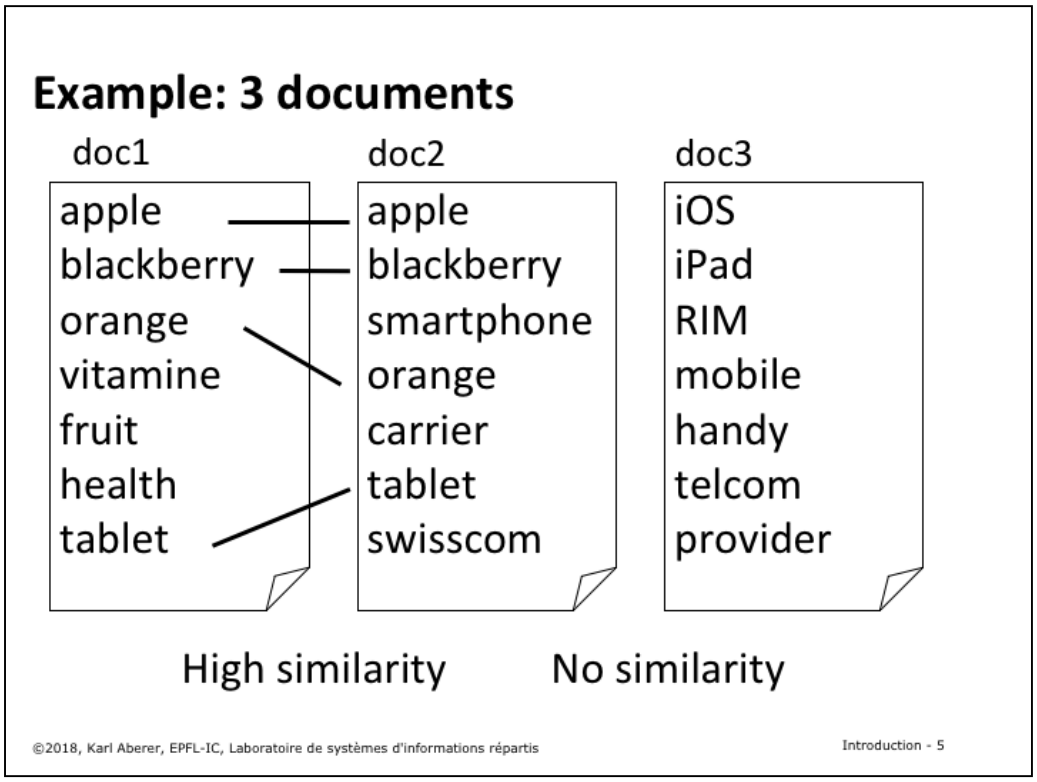
The Problem

Vector Space Retrieval handles poorly the following two situations

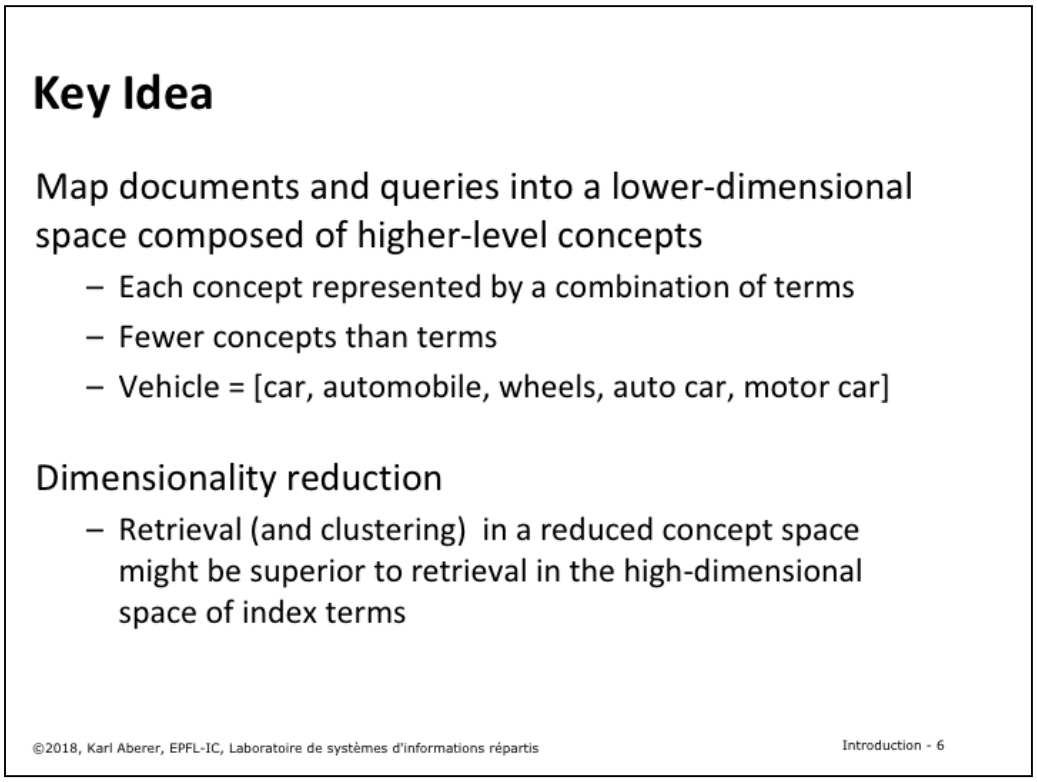
1. *Synonymy*: different terms refer to the same concept, e.g. car and automobile
   - Result: poor recall
2. *Homonymy*: the same term may have different meanings, e.g. apple, model, bank
   - Result: poor precision

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.

concept

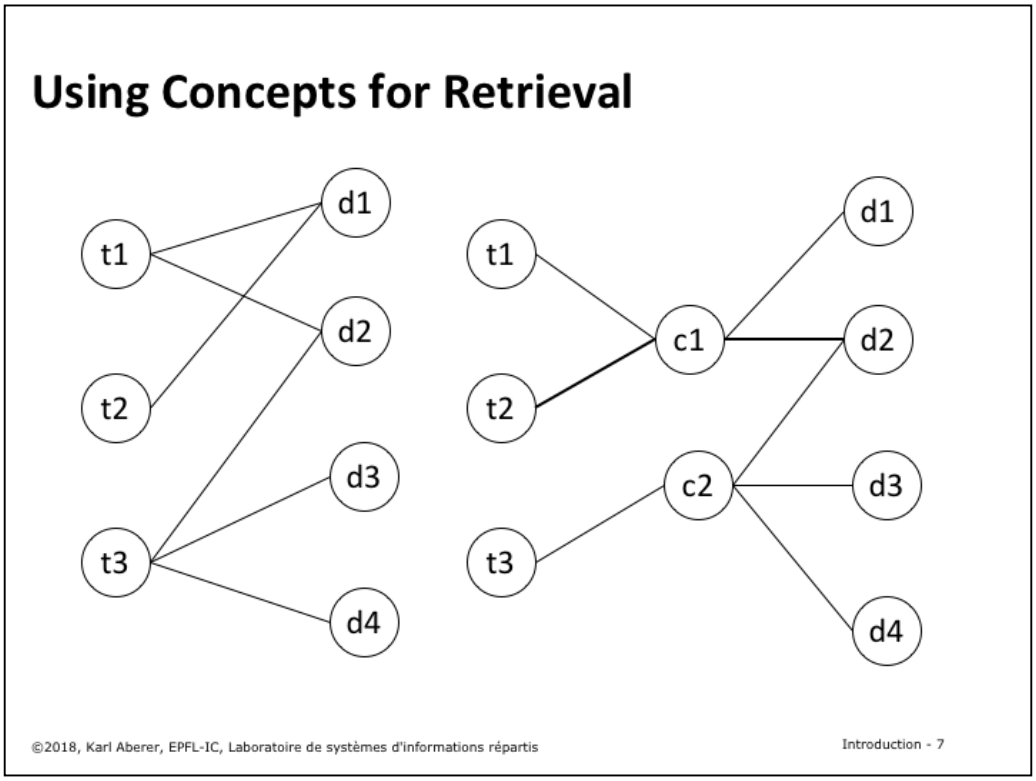


Let's illustrate the problem resulting from synomyms and homonyms by an example. Among these three documents at the level of terms doc1 and doc2 are (seem) highly related, whereas doc3 has no similarity with the other two documents. With human background knowledge it is however easy to see that in reality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

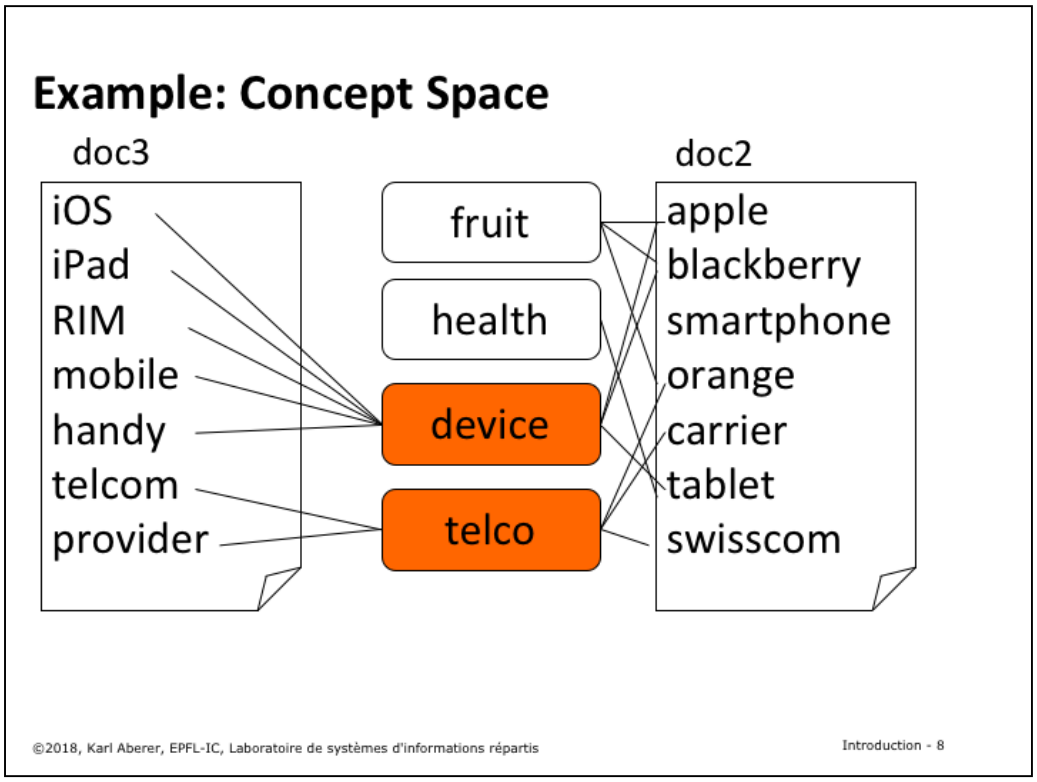


Thus it would be an interesting to base information retrieval methods on the use of concepts, instead of terms. To that end it is first necessary to define a "concept space" to which documents and queries are mapped, and compute similarity within that concept space. This idea is developed in the following. The concept space should ideally have much lower dimension than the term space, whose dimensionality is determined by the size of the vocabulary.

what is the term space in M ?



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.

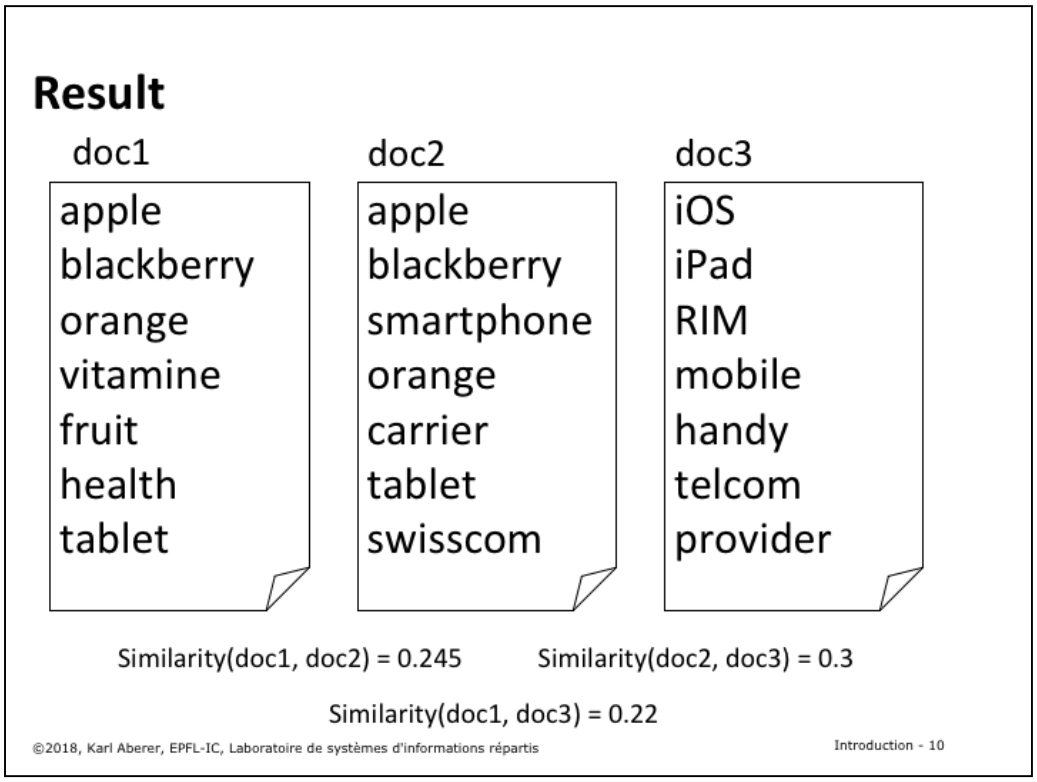
device = {iOS, iPad, RIM, mobile, handy, tablet, apple, blackberry}

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

doc3 = (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

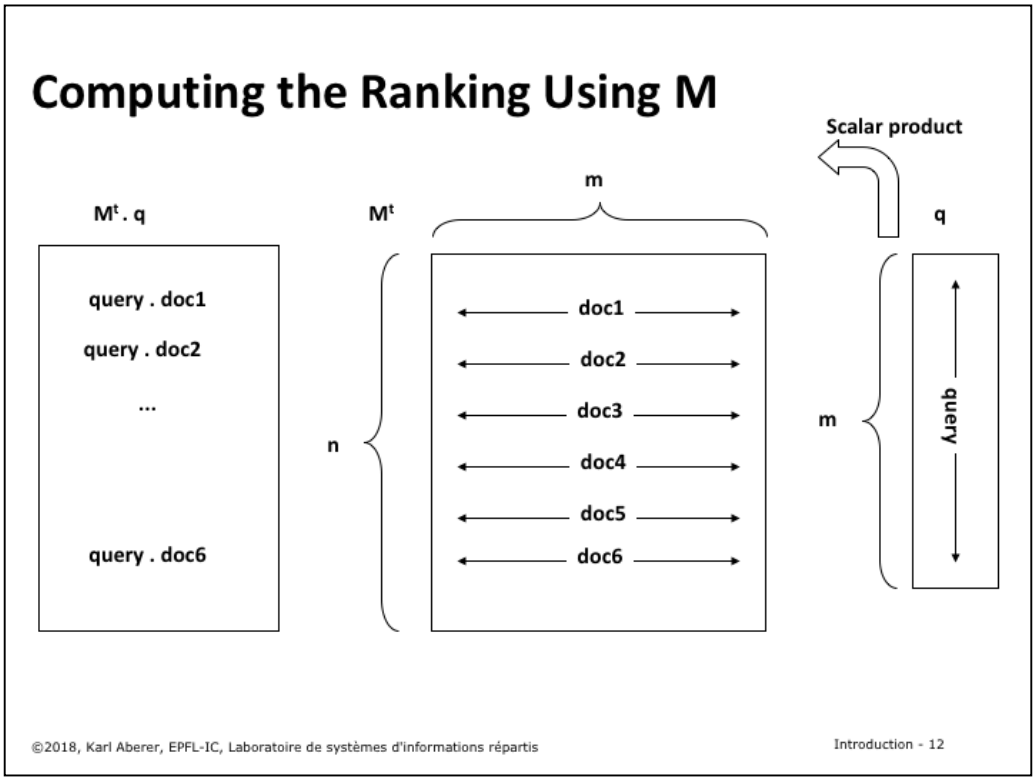
Problem: how to identify and compute "concepts" ?

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.

	doc1	doc2	doc3
term1			
term2			
term3			



We can understand the process of producing a retrieval results in vector space retrieval as a matric 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 matric M. We assume that all columns in M and 1 are normalized to 1.

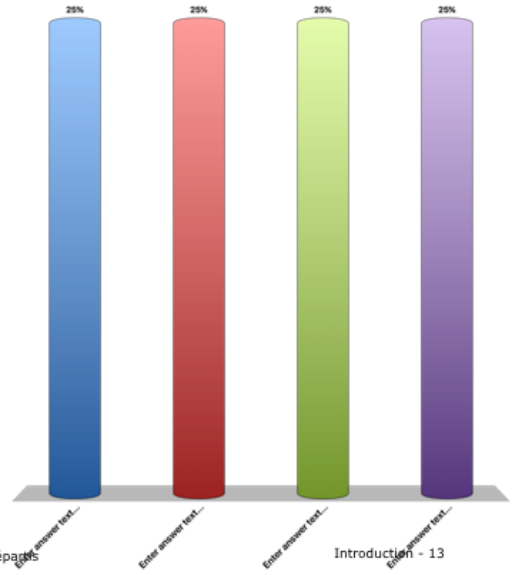
In vector space retrieval each row of the matrix M^t corresponds to

A. A document

B. A concept

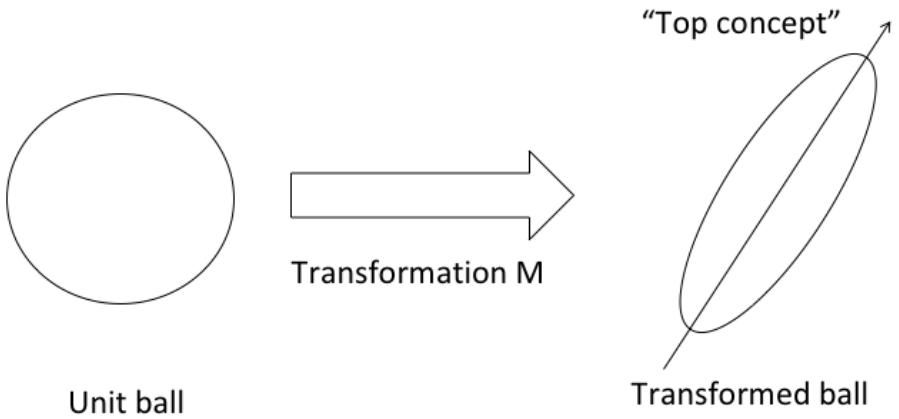
C. A query

D. A query result



Identifying Top Concepts

Key Idea: extract the essential features of M^t and approximate it by the most important ones



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.

M =

	doc1	doc2	doc3
term1			
term2			
term3			

	n1	n2	n3
m1			
m2			
m3			

recall face recognition:
- to find eigenvalues of SS^T
- gives us the best representation of faces with the smallest number of eigenvectors
- by representing it with a smaller no. of eigenvectors, we are uncovering its latent information

face1_x1 face1_x1 face1_x1		face1_x1 face1_x2 face1_x3
face1_x2 face1_x2 face1_x2		face2_x1 face2_x2 face2_x3
face1_x3 face1_x3 face1_x3		face3_x1 face3_x2 face3_x3

Therefore, the eigenvectors of MM^T gives us the latent information of the documents. And each element of the original document is a weighted combination of terms. READ CHEAP WAY OF COMPUTING EIGENVECTORS

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

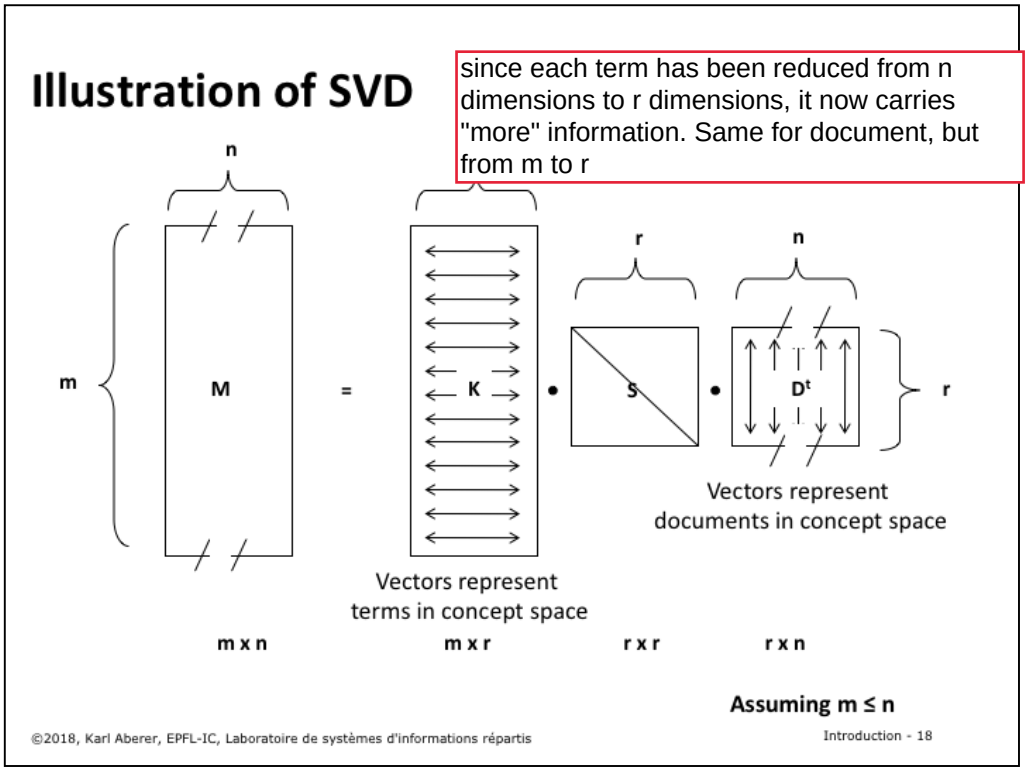
By taking only the largest ones we obtain a good "approximation" of M (least square approximation)

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.

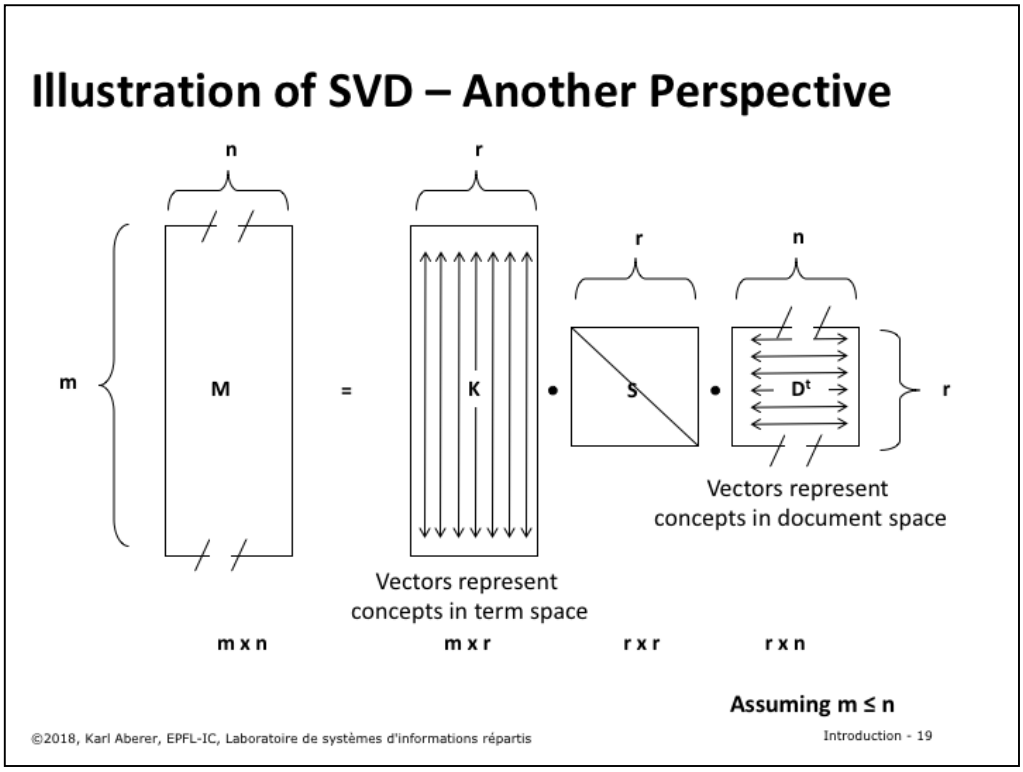
	doc1	doc2	doc3
term1			
term2			
term3			

	n1	n2	n3
m1			
m2			
m3			

r << n
each row of K is the representation of terms in the concept space

r << n
each row of D is the representation of terms in the document space

what is the max size of r ? read ML notes



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.

r is not document. r is concept

m : term
n : document
r : concept

In other words, each concept is a weighted vector of terms

Latent Semantic Indexing

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

- Keep the corresponding columns in K and D

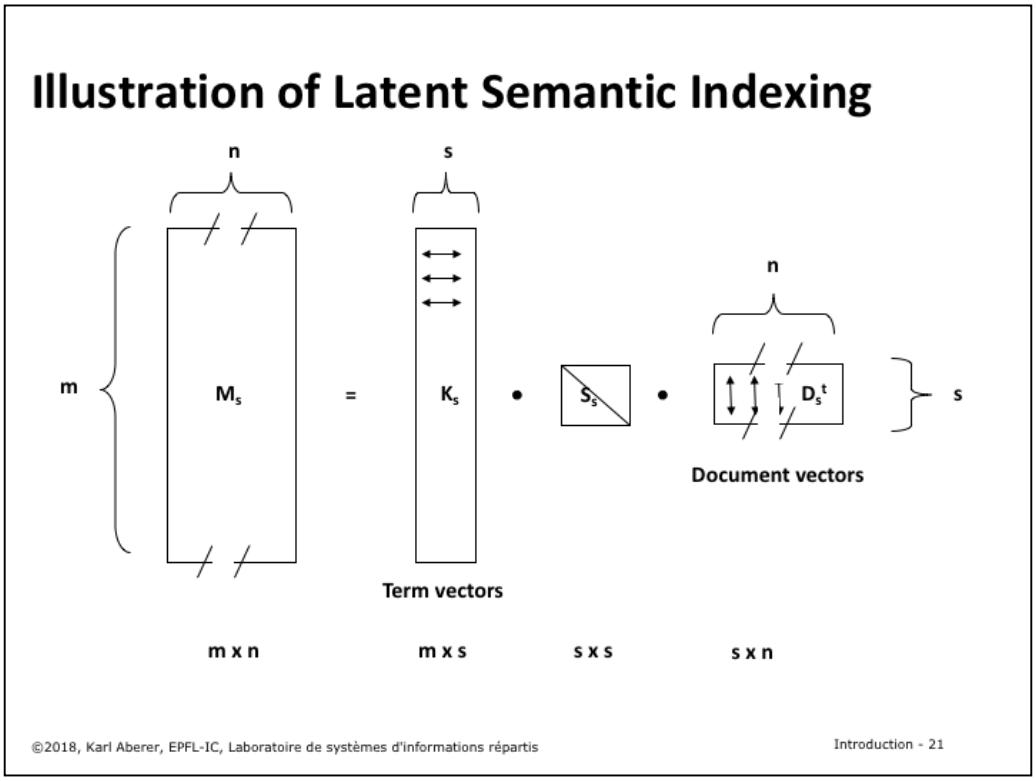
The resultant matrix is called M_s and is given by

- $M_s = K_s.S_s.D_s^t$ 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

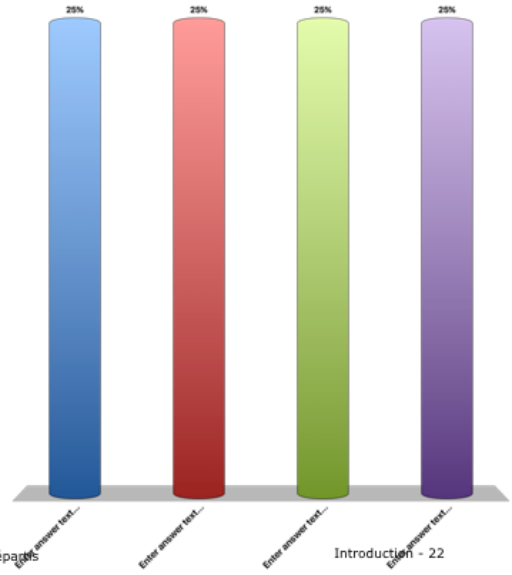
Using the singular value decomposition, we can now derive an "approximation" of M by taking only the s largest singular values in matrix S. The choice of s determines on how many of the "important concepts" the ranking will be based on. The assumption is that concepts with small singular value in S are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

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

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

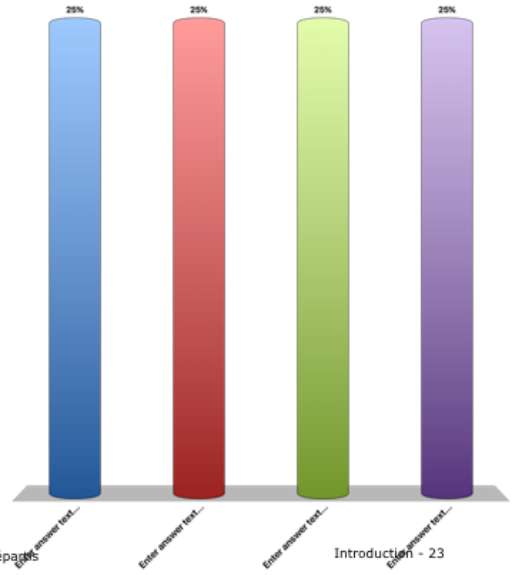


The number of term vectors in the SVD for LSI

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

C. Is larger than the number of rows in the matrix M



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 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:

$q^* = q^t.K_s.S_s^{-1}$ mapping query to the document space

Then compare transformed vector by using the standard cosine measure

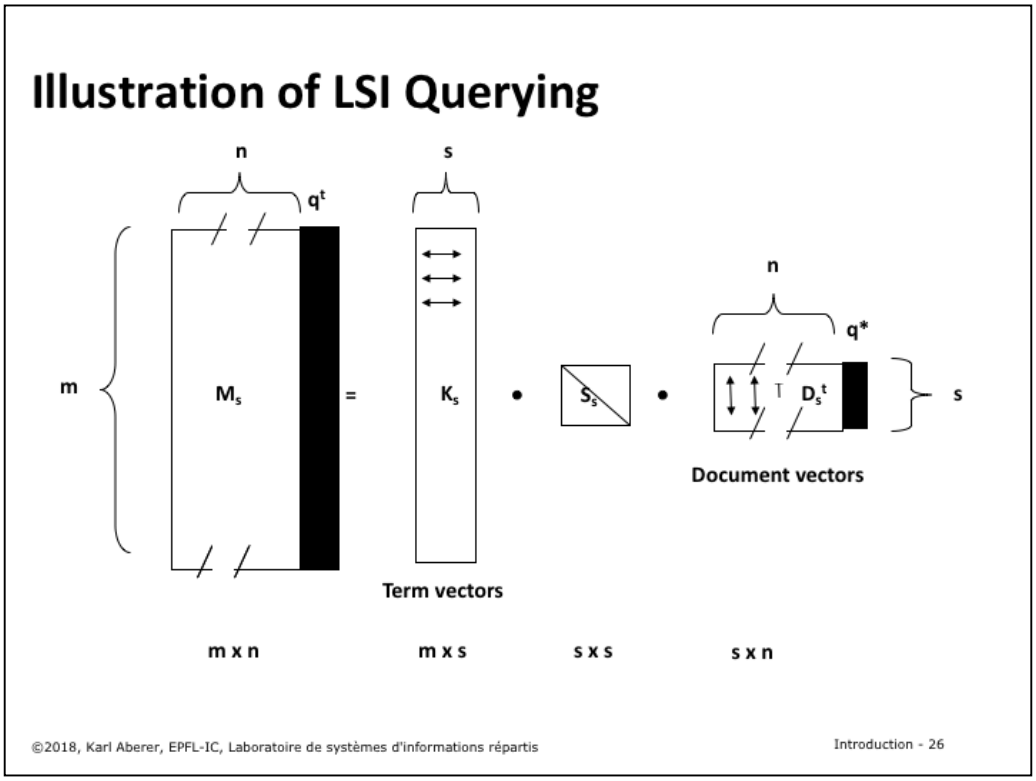
$$sim(q^*, d_i) = \frac{q^* \bullet (D_s^t)_i}{|q^*| |(D_s^t)_i|}$$

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)

truncate = PCA



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}

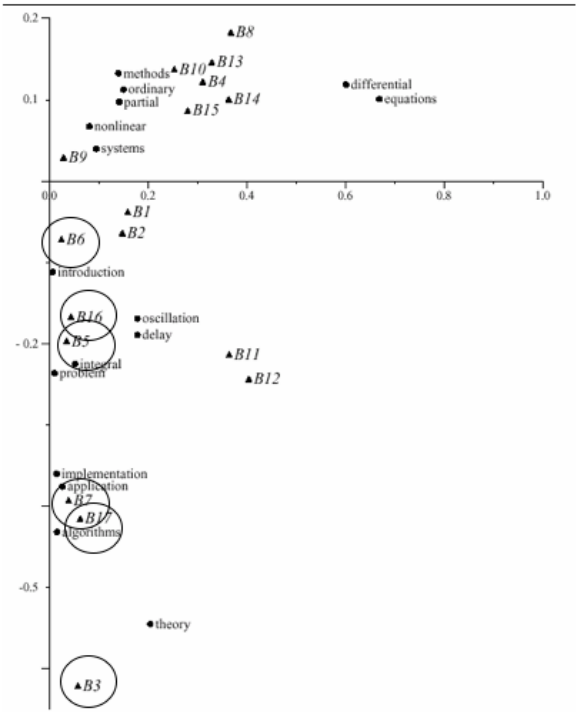
s=4

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.

why can produce negative result ?

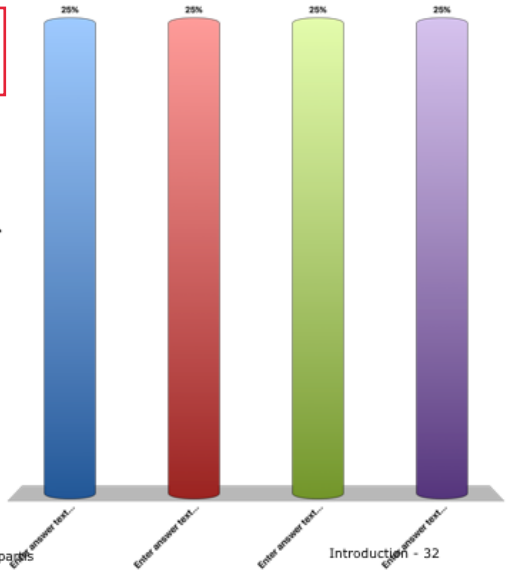
Plot of Terms and Documents in 2-d Concept Space



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.

A query transformed into the concept space for LSI has …

A. s components (number of singular values)
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

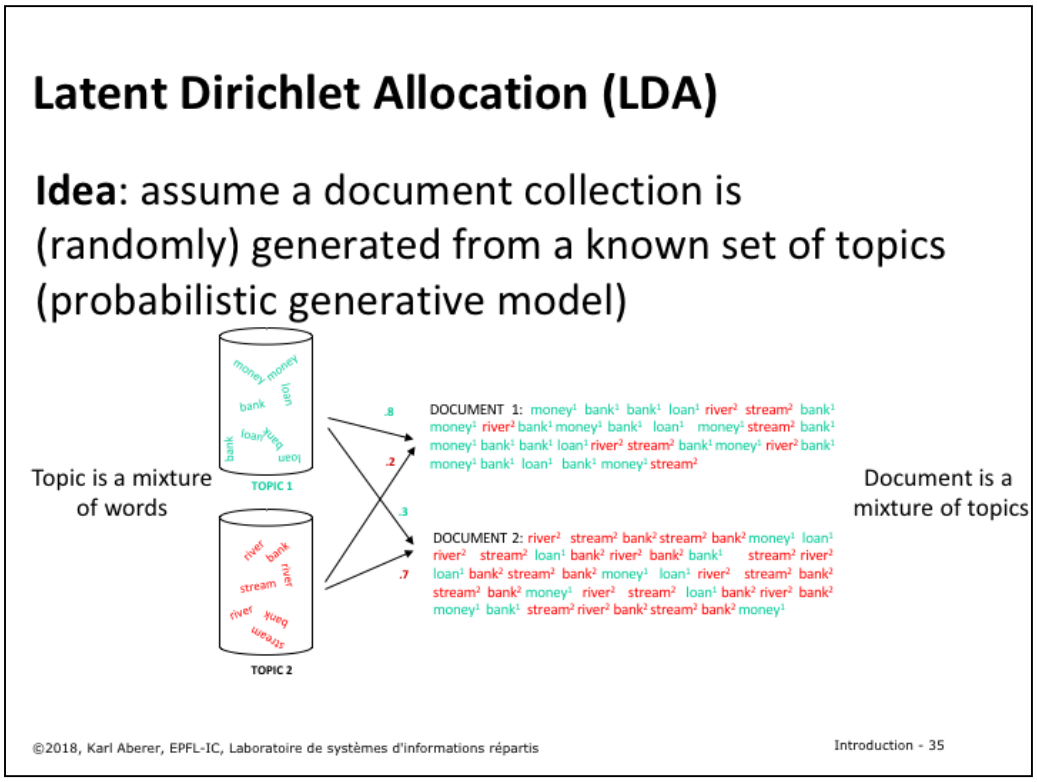
Latent Dirichlet Allocation

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

Same objective creating a 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.

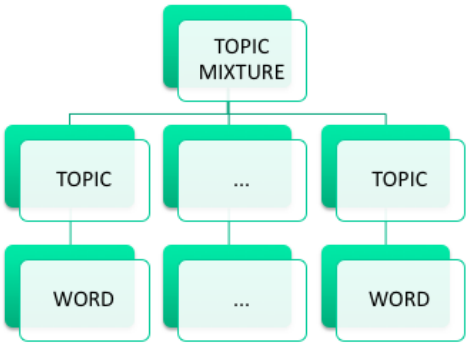
Similar to the probabilistic language model used in information retrieval, we have here another example of a probabilistic generative model.

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

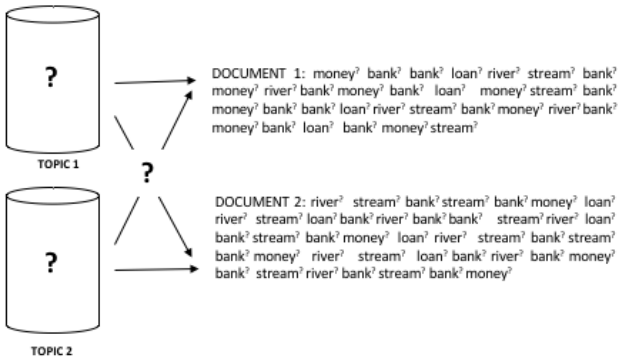
For every word position, sample a word from the chosen topic



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

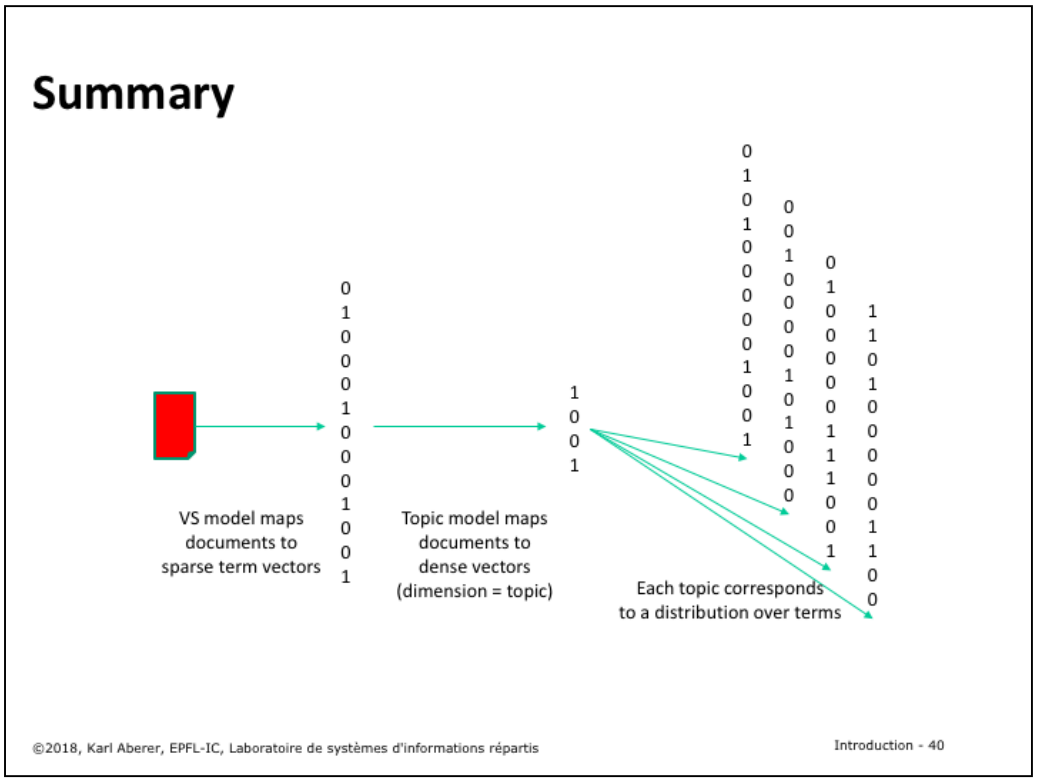
Unsupervised Learning of topics

- 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.

WORD EMBEDDINGS

Word Context

The neighborhood of a word expresses a lot about its meaning

– "You shall know a word by the company it keeps" (J. R. Firth 1957)

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Word: w = banking

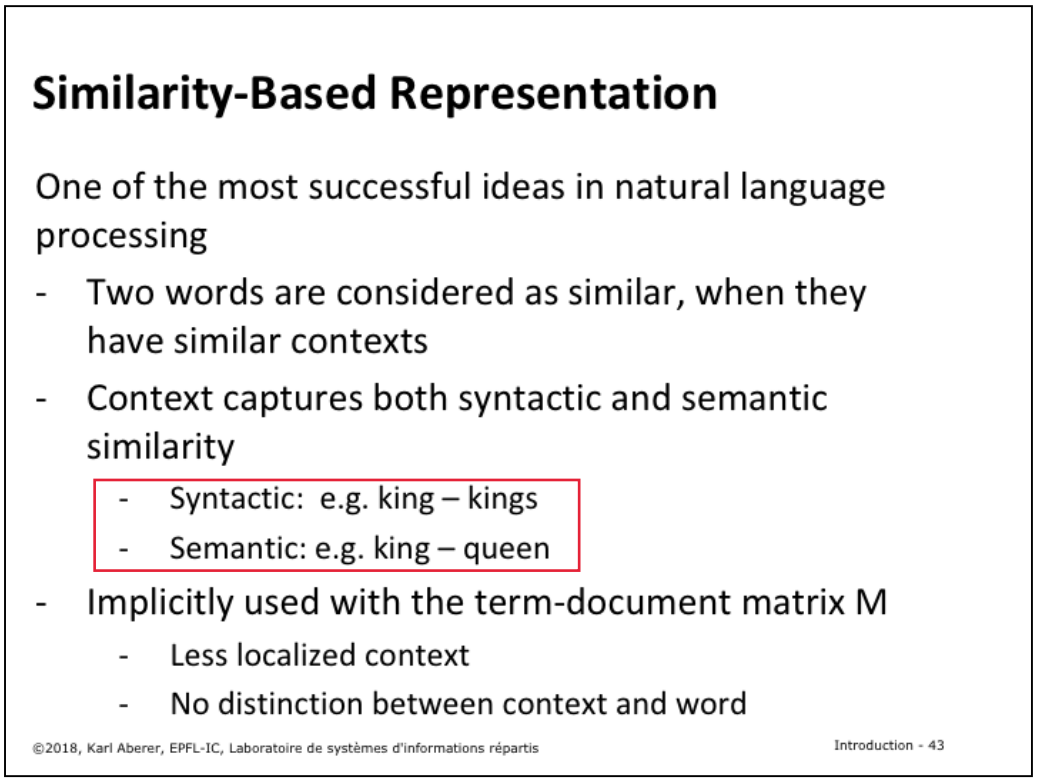
Context: C(w) = {government, debt, problems, turning, into, crises, as, has, happened, in}

Word-Context occurrence: (w, c), c ∈ C(w)

what about the other banking

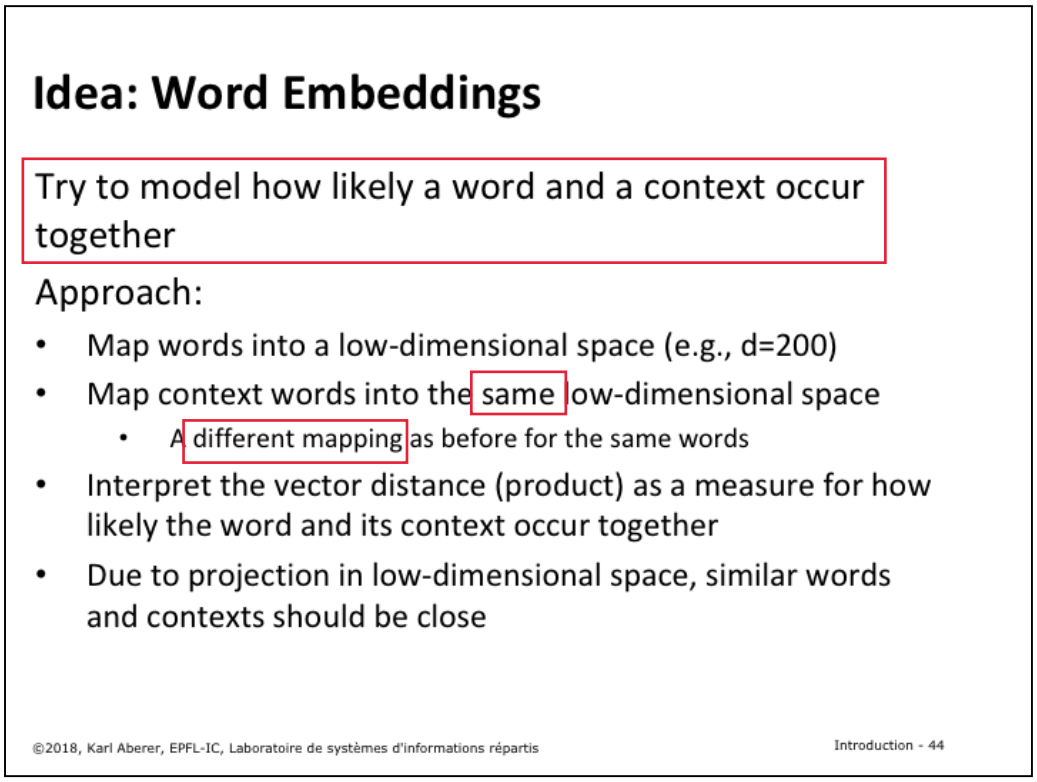
With latent semantic indexing we have exploited the idea that words that occur together in documents have a likelihood to have also some shared meaning. This idea is actually quite old, and has been already expressed, for example, by Firth in 1957. We will now exploit this idea in a slightly different way, by focusing on a much smaller context of a word than its document. We will just consider the local neighborhood of the work within a phrase. Typically we will consider all the neighborhoods that we can find in all documents of a large document collection.

For each word w we can identify its neighboring words, which will call its context and denote by C(W). The context can be determined by choosing a certain number of preceding and succeeding words, e.g. a typical number used is 5.



This idea is one of the main ideas used (successfully) in natural language processing. The neighborhood captures not only semantic relationships among words (as would also co-occurrence in the same document).It can at the same time also capture syntactic relationships. This is a main difference to methods based on document co-occurrence like LSI.

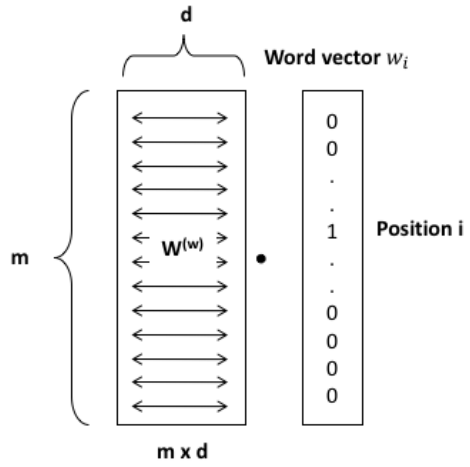
A second important difference is that methods based on this idea distinguish between a word occurring as the center of a context and as a member of context. For example, it is very unlikely the word “dog” would have in its context a second occurrence of the word “dog”. Thus dog as a “context word” needs to be treated differently from dog as the word of interest. This distinction is not made in purely co-occurrence based methods.



The basic idea for word embeddings is simple. Both words and context words are mapped into the same low-dimensional space. Their representations in that space should be similar, if the word and the context word occur often together. Thus we capture the information on word context in a low-dimensional representation. Through the dimensionality reduction the expectation is that words and contexts that play similar roles would also move together, and thus we could capture syntactic and semantic similarity.

Dimensionality reduction is helpful, since vocabularies can become very large, in particular if not only words but also short phrases are included (e.g. tens of millions of entries). Thus data would be very sparse in these spaces and it would be difficult to model similarity. The low-dimensional spaces have on the other hand typically a few hundred dimensions.

Overview of the Model



Mapping a word w

$V_w = W^{(w)} . w$

Mapping a context c

$V_c = W^{(c)} . c$

Parameters of model (θ)

All coefficients of $W^{(w)}$ and $W^{(c)}$

Rows represent words of the vocabulary in concept space of dimension d

We introduce the basic notations we will use in the following. Note that the size of the vocabulary m is typically very large (e.g. millions), whereas the size of the low-dimensional space d is typically in the hundreds. The model will be fully specified by the two matrices $W^{(w)}$ and $W^{(c)}$ used to map words and context words. For convenience we will denote in the following the coefficients of these two matrices by θ.

this is a dot product ?
so then V_w = vector
while V_c = matrix ?

A column of matrix Wc represents

A dimension in the concept space

How relevant word c is for each concept

How often a context words c co-occurs with all words

A representation of word c in concept space

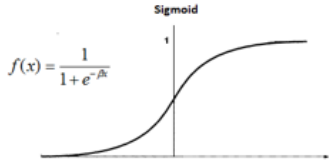
Learning the Model from the Data

Consider a pair (w, c)
Does it origin from the data?

Model this as a probability (sigmoid)

V_c and V_w are unknowns ? learn V_c and V_w such that P = 1 ? V_c and V_w correspond to page 45 ?

$$p(D = 1|w, c; \theta) = \frac{1}{1 + e^{-v_c \cdot v_w}} = \sigma(v_c \cdot v_w)$$



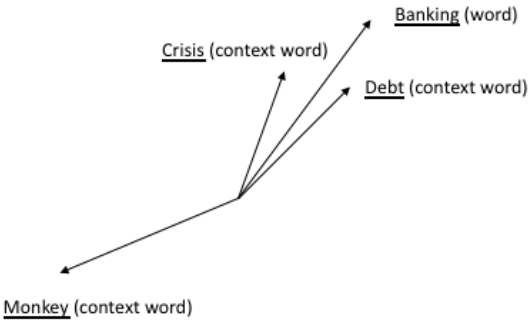
Intuition: convert similarity value in vector space into a probability

Differently to LSI, where the (parameters of the) model have been derived by a mathematical operation (SVD), the idea in word embeddings is to learn the parameters directly from the data. To start with, we consider a word-context pair (w,c) and ask the question whether it comes from the data. We would like that (ideally) the model gives us a probability of 1 if this is the case. In order to derive a probability from the model (the embedding into a low-dimensional space), we proceed as follows: we first take the scalar product of the embedded word and the context word vectors (this is where the parameters are hidden). This product will be large positive when the vectors are similar and large negative when they are opposite. In order to turn those values into a quantity that can be interpreted as probability (i.e. a value in [0,1]) we apply the sigmoid function It produces values close to 0 for large negative arguments, and values close to 1 for large positive arguments. The use of the sigmoid function in this model is for exactly the same reasons as it is for its use in binary logistic regression, converting a similarity value into a probability. Apart from mapping the values to [0,1] it also is insensitive to large outliers.

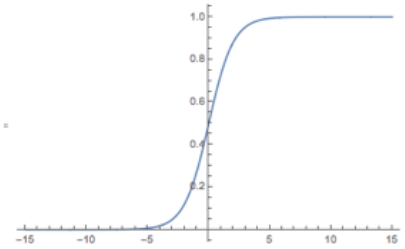
in LSI, the coefficients of the matrix are learnt through SVD.

in WE, the coefficients of the matrix are learnt through

Illustration of Word Embedding



Banking • Crisis = + 100 σ(Banking • Crisis)≈1
Banking • Debt= + 80 σ(Banking • Debt)≈1
Banking • Monkey = -150 σ(Banking • Monkey)≈0



Problem: finding θ

Here we illustrate the embedding of words and context words in the same low-dimensional space. The idea is that the context words that typically co-occur together with a word, have similar embedding vectors, whereas others (like monkey, which is rarely occurring together with bank) have very different ones. The sigmoid function then converts the similarity values obtained from the scalar products of the embedding vectors into the interval [0,1].

Finding Parameters

Formulate an optimization problem:

Assume we have positive examples D for (w, c), as well as negative examples $\widetilde{D}$ (not occurring in the document collection)

Find θ such that the overall probability is maximized

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta)$$

Now we assume that we do not only know positive examples of word-context occurrences, but also negative ones, i.e. pairs that do not occur together. We can similarly model this case as a probability $P(D=0|w,c,\theta)$. If we have now a (big) collection of positive and negative samples D and $\widetilde{D}$, the overall probability of such a collection to occur can be expressed as the product of all individual probabilities. Given this overall probability, the problem we would like to solve is to find parameters θ that maximize this probability. This is then the best model we can obtain under the assumptions that have been made.

positive examples: word and context word
negative examples: any pair that does not exist "together"

Maximizing the Probability

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta)$$

$$= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-v_c^T v_w)} + \sum_{(w,c)\in \tilde{D}} \log(1-\frac{1}{1+\exp(-v_c^T v_w)})$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-v_c^T v_w)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(v_c^T v_w)})$$

If we let $\sigma(x) = \frac{1}{1+e^{-x}}$ we get:

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log(\frac{1}{1+e^{v_c \cdot v_w}})$$

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c \cdot v_w)$$

We can transform the maximization objective using our model as shown above.

Loss Function

Define a loss function (to be minimized)

$$J(\theta) = \frac{1}{m}\sum_{t=1}^{m} J_t(\theta)$$

$$J_t(\theta) = -\log\left(\sigma(v_c \cdot v_w)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(v_{c_k} \cdot v_w\right)\right)$$

v_{c_k} are negative examples of context words taken from a set $P_n(w)$

why only 1 positive example ? Why dont we average over the negative examples ?

m = terms

minimize the negative of the likelihood

Following the previous derivation we can define a loss function J that optimizes the probabilities if minimized. (We transform the maximization problem into a minimization problem by defining a loss function $J(\theta)$) The loss function is defined over all terms in the vocabulary. For each term w it consists of a first component corresponding to a positive example, a (w,c) pair that occurs in the document collection, and a couple of negative examples for the same word. The question is how to obtain the negative examples.

How do you train since both vc and vw are variables ?

Obtaining Negative Samples

Negative samples are taken from

$$P_n(w) = V \setminus C(w)$$

Empirical approach

- If p_w is the probability of word w in collection, choose the word with probability $p_w^{3/4}$
- Less frequent words are sampled more often
- Practically: approximate the probability by sampling a few non-context words

The method for learning the model relies on having negative samples. How to obtain them? We can choose any word from the vocabulary V that is not contained itself in the context. In order to model the occurrences we do this by considering the frequency at which the words occur in the document collection. Doing so, experience shows that high frequency words (e.g. stopwords) would however be favored to much, reducing the quality of the model. Thus the probability of word occurrence used for sampling is moderated by an exponent (in practice ¾). Also for obtaining the statistics on word probabilities for practical purposes not the whole data collection is used, but just a small sample of words not occurring in the context.

Stochastic Gradient Descent

For every (w, c) in D, update θ

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$$

Updates only affect rows in $W^{(w)}$ and $W^{(c)}$ where w and c appear

Finally, given the loss function, finding the optimal parameters θ can then be done using standard methods from machine learning. Then θ can be determined using gradient descent, i.e. standard search for minima using derivatives. More precisely, we will apply stochastic gradient descent. We update the parameters for each sample, separatly. Standard gradient descent would update the values of data only after having seen all samples, which would take extremely long given the very large number of samples. When stochastic gradient descent is used, only rows that contain the word or some context word in the loss function need to be updated. This implies that the data has to be organized that these rows can be efficiently accessed (e.g. using hashing).

Result

Matrices $W^{(w)}$ and $W^{(c)}$ that capture information on word similarity

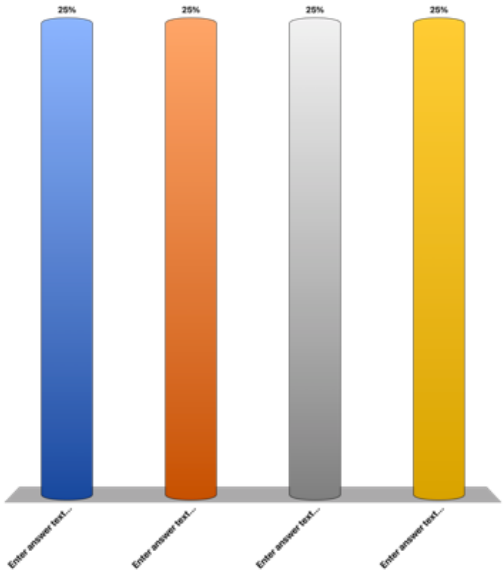
- Words appearing in similar contexts generate similar contexts and vice versa
- Hence, mapped to similar representations in lower dimensional space
- Use $W = W^{(w)} + W^{(c)}$ as the low-dimensional representation

As a result we obtain two models mapping words into a low dimensional space for different reasons. In practice, the final model is constructed as the sum of the two matrices. Though, there exists no theoretical insight, why exactly this models work well, practice shows that they are organizing words in interesting ways, capturing many semantic and syntactic relationships.

so then context and word will get mapped through the same matrix ?

A word embedding …

- depends only on the dimension d
- depends on the dimension d and number of iterations in gradient descent
- depends on the dimension d, number of iterations and chosen negative samples
- there are further factors on which it depends



A word embedding

Depends only on the dimension d

Depends on the dimension d and the number of iterations in gradient descent

Depends on the dimension d, the number of iterations and the chosen the negative samples

There are further factors on which it depends

Alternative Approaches

CBOW (Continuous Bag of Wods Model)

- Predict word from context

GLOVE: exploits the following observation

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Technically similar

The model we have discussed is one variant of a number of similar models that have been proposed recently. One interesting model is GLOVE, which is based on the observation that ratios of probabilities can be used to capture semantic relationships among terms. For example, the term solid is much more likely to co-occur with ice, than with water, and similarly steam co-occurs much more likely with gas than with ice. On the other hand, water co-occurs frequently with both. So the ratio of probabilities captures semantics on the meaning of solid and gas, beyond co-occurrence.

Technically most of these models follow similar approaches, first modeling some observation on how co-occurrence can capture semantic relationship, and then using gradient descent methods to learn the parameters.

Properties of Word Embeddings

1. Similar Terms are Clustered
2. Syntactic and Semantic Relationships encoded as Linear Mappings
3. Dimensions can capture Meanings

Word Embeddings have received recently a lot of attention as they exhibit fascinating capacity to capture different types of relationships among words. We will illustrate some of those in the following.

Glove: Nearest words to Frog

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

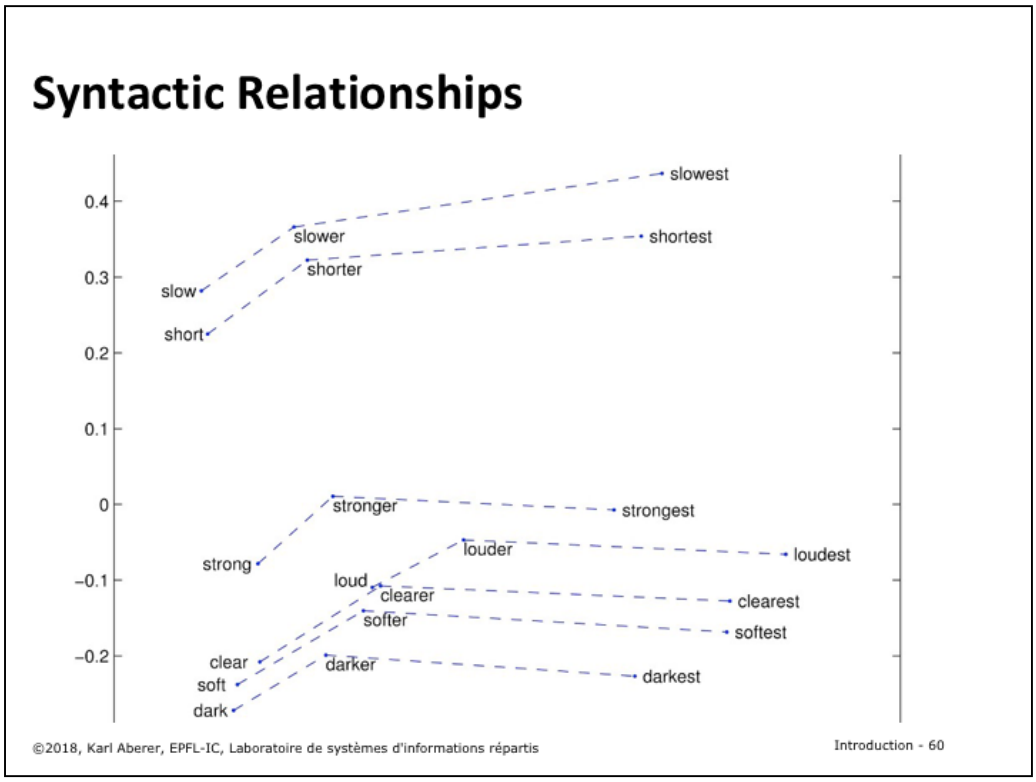
litoria

leptodactylidae

rana

eleutherodactylus

A first property is that similar words are grouped together. Here is a nice example, produced with Glove, that illustrates the point. (The underlying data is from Wikipedia).

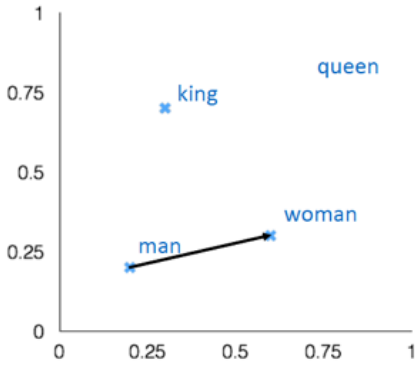


Word embeddings also capture syntactic relationships, like singular-plural or comparative and superlative, as shown here. This type of visualizations is obtained by projecting from the d-dimensional word embedding space (appropriately) into a 2-dimensional space.

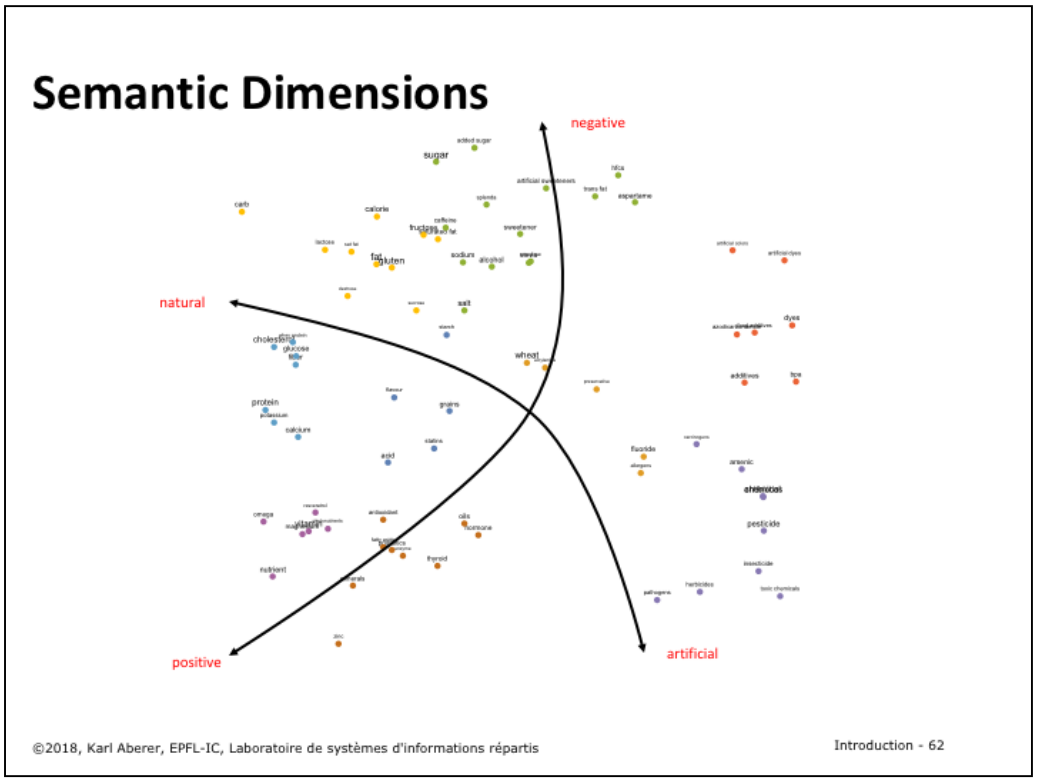
Word Analogies: semantic relationships

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Even more interestingly, word embeddings enable “computing” with relationships (this is called the word analogy task). Analogies translate into linear mappings!



Furthermore, word embeddings can capture semantic meaning in dimensions. In this example, data on nutrition has been analyzed and terms indicating qualities of foods are extracted. The word embedding clearly organizes it according to properties that are human understandable, e.g. natural vs. artificial ingredients.

Use of Word Embedding Models

Document Search
- Use word embedding vectors as document representation

how ?

Thesaurus construction and taxonomy induction
- Search engine for semantically analogous / related terms

Document classification
- Use of word embedding vectors of document terms as features

Impact

- Latent semantic indexing

Indexing by **latent semantic analysis**
S Deerwester, ST Dumais, GW Furnas… - Journal of the …, 1990 - search.proquest.com
Cited by 12027 Related articles All 87 versions Web of Science: 3525 Cite Save

- Latent Dirichlet allocation

Latent dirichlet allocation
DM Blei, AY Ng, MI Jordan - Journal of machine Learning research, 2003 - jmlr.org
Cited by 17791 Related articles All 123 versions Web of Science: 5278 Cite Save

- Word embeddings

Distributed representations of words and phrases and their compositionality
T Mikolov, I Sutskever, K Chen, GS Corrado… - Advances in neural …, 2013 - papers.nips.cc
Cited by 3370 Related articles All 23 versions Cite Save

References

Research Literature

- Deerwester, Scott, et al. "Indexing by latent semantic analysis." *Journal of the American society for information science* 41.6 (1990): 391.
- Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
- Goldberg, Yoav, and Omer Levy. "word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method." *arXiv preprint arXiv:1402.3722* (2014).

Course Materials inspired by

- http://videolectures.net/deeplearning2015_manning_language_vectors/