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

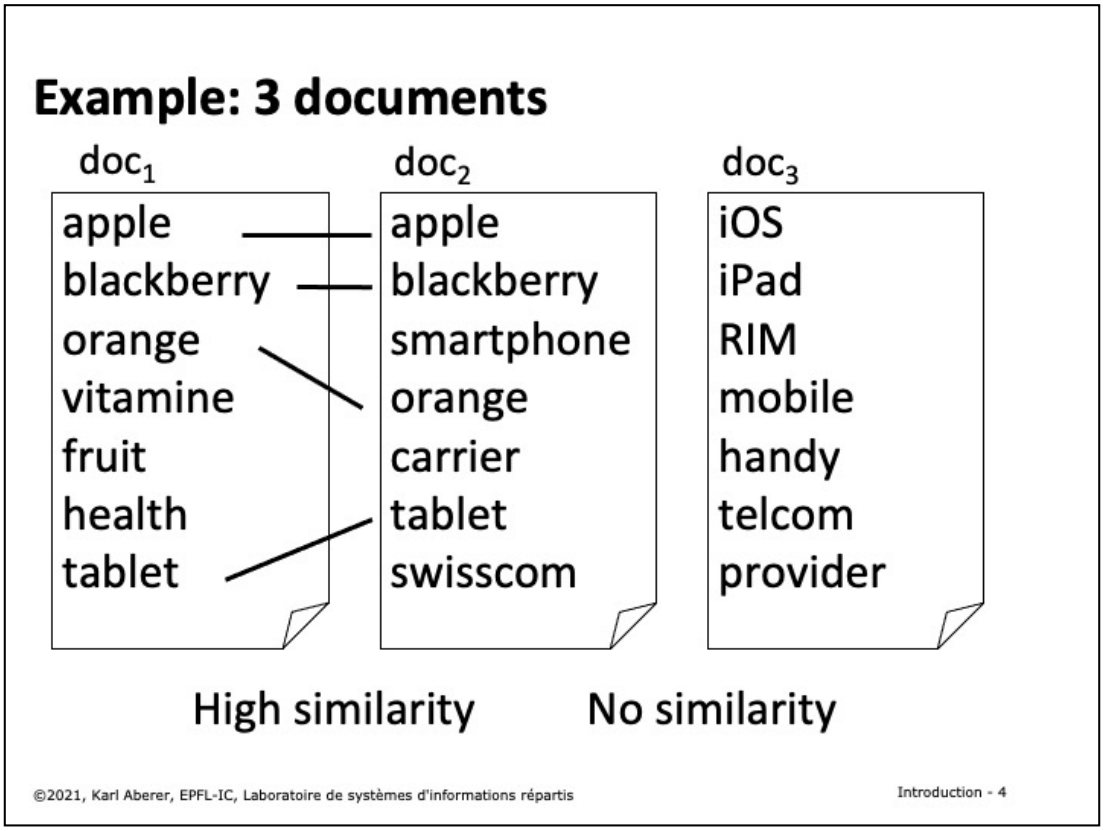
Despite its success and widespread use the vector space retrieval model suffers from some problems. The important insight is that terms may indicate a concept a user is interested in, but there does not necessarily exist a one to one correspondence between terms and concepts. As a consequence retrieval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

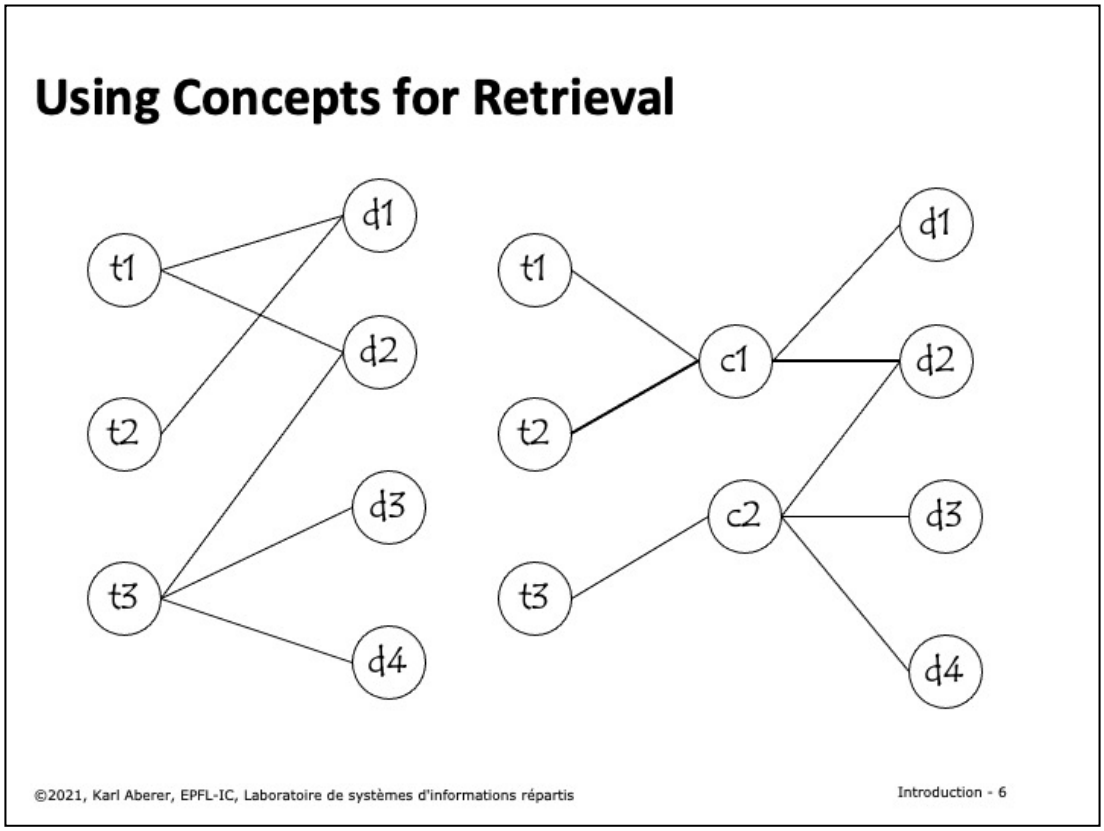
Map documents and queries into a lower-dimensional space composed of higher-level concepts

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

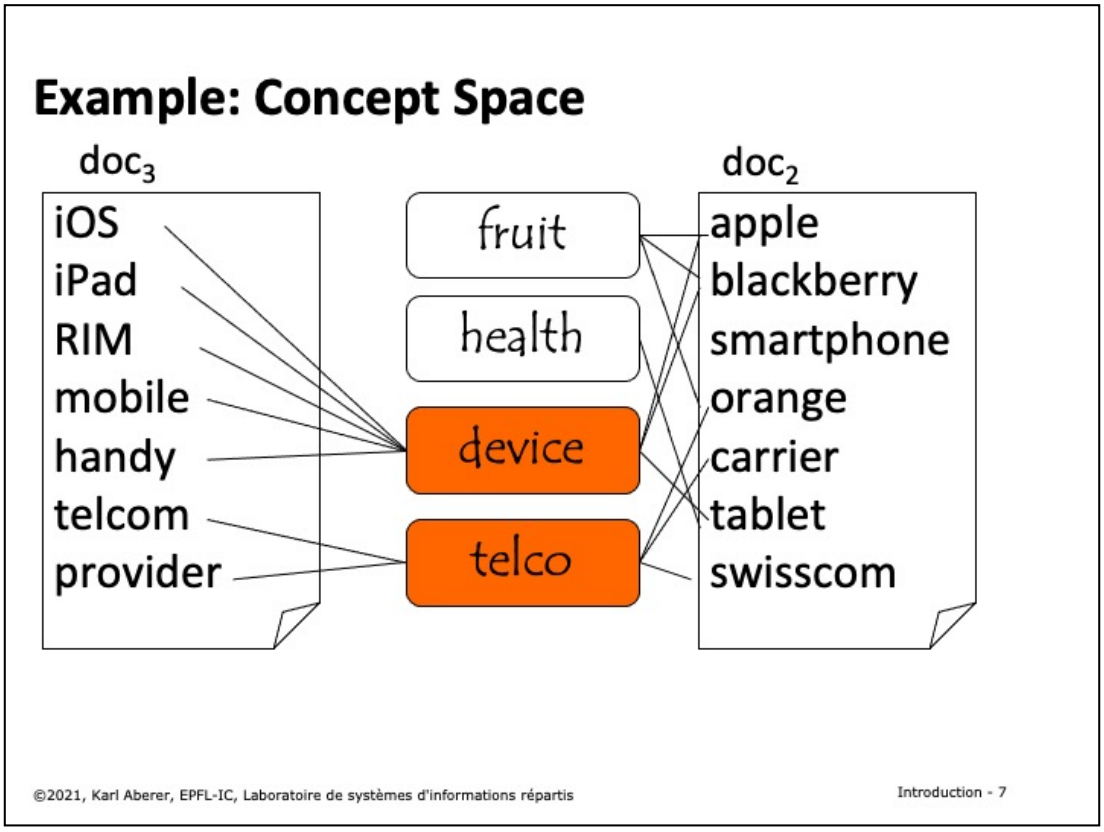
Dimensionality reduction

- 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}

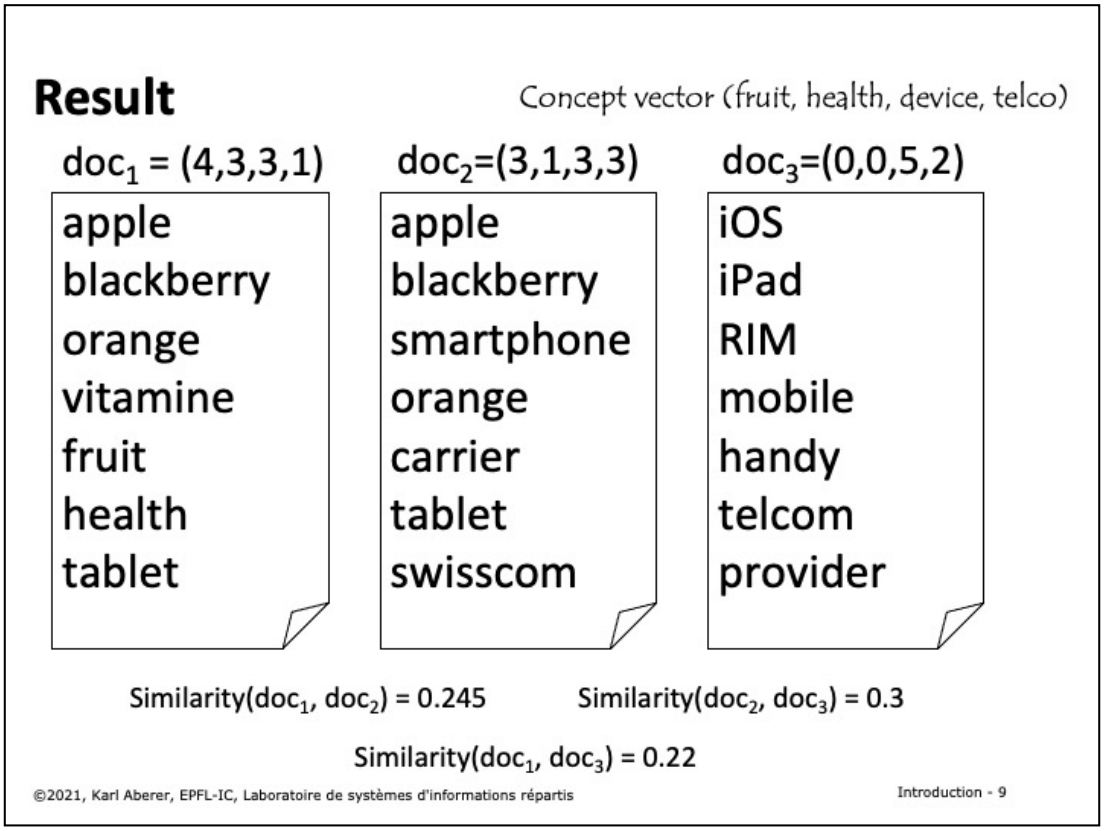
Document represented by concept vector, counting number of concept terms, e.g.

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

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

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 homonyms occurring in these documents.

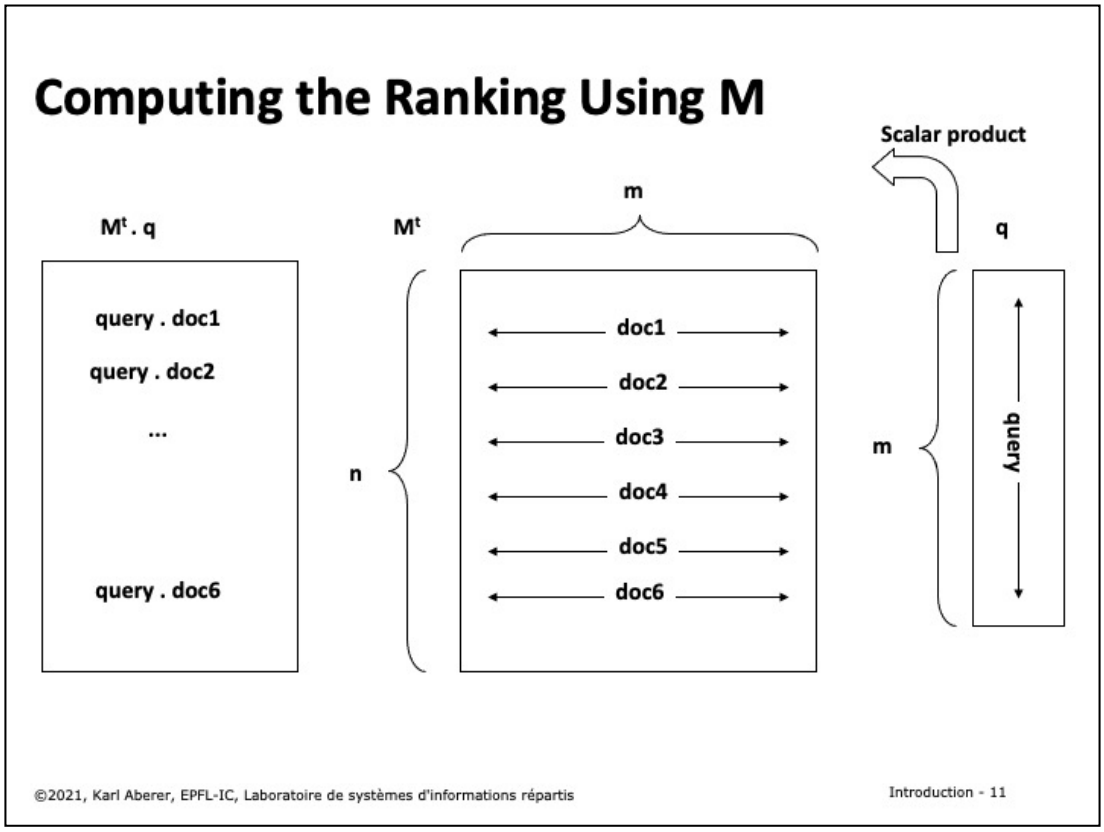
Basic Definitions

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.



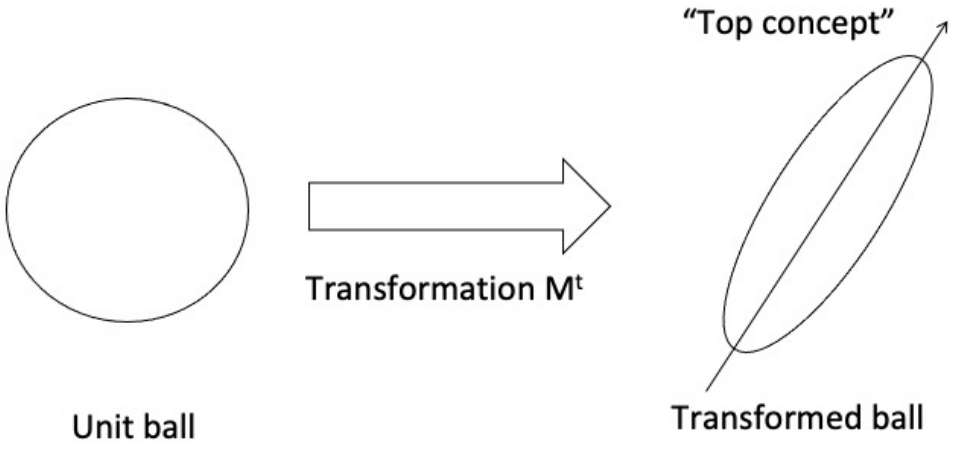
We can understand the process of producing a retrieval results in vector space retrieval as a matrix operation. This is illustrated in this figure. The ranking is the result of computing the product of a query vector q with the term-document matrix M. We assume that all columns in M and q are normalized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

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

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

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 computation can be performed in O(n^3). Note that the complexity is considerable, which makes the approach computationally expensive. There exist however also approximation and randomized techniques to perform this decomposition more efficiently (e.g. https://research.fb.com/blog/2014/09/fast-randomized-svd/)

Interpretation of SVD

We can write M = K.S.D^t 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)

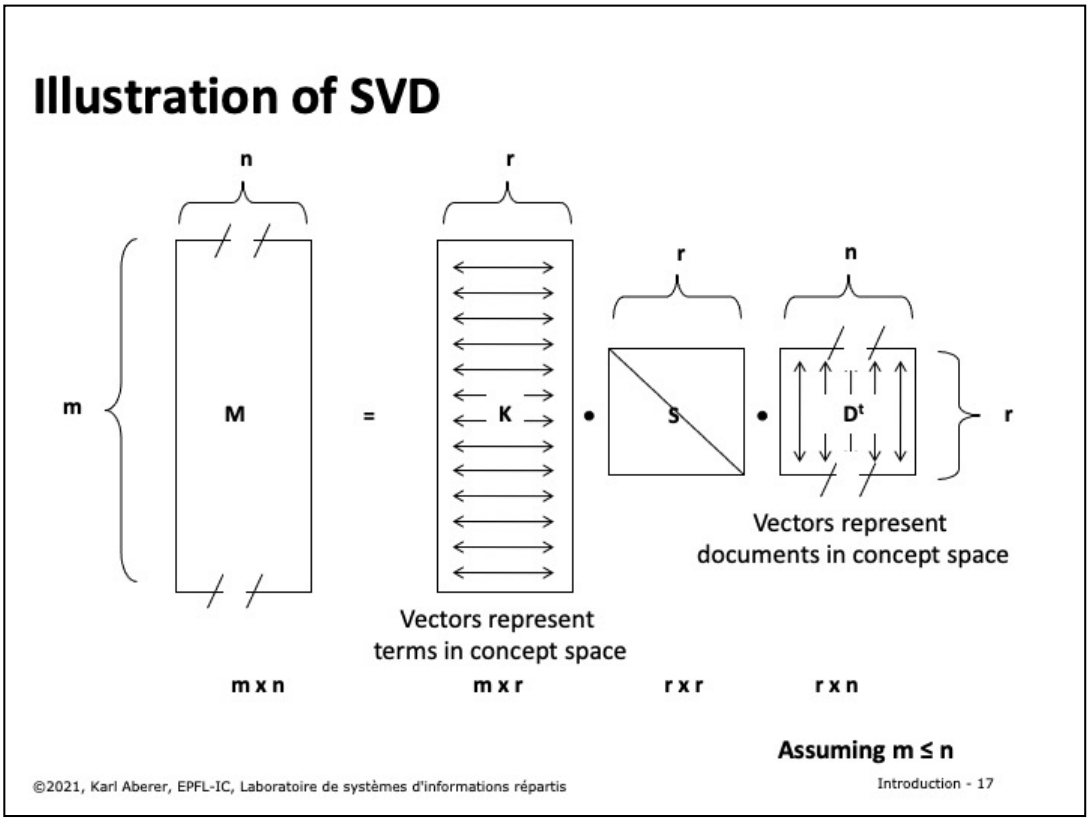
The singular values s_i are the lengths of the semi-axes of the hyperellipsoid E defined by

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

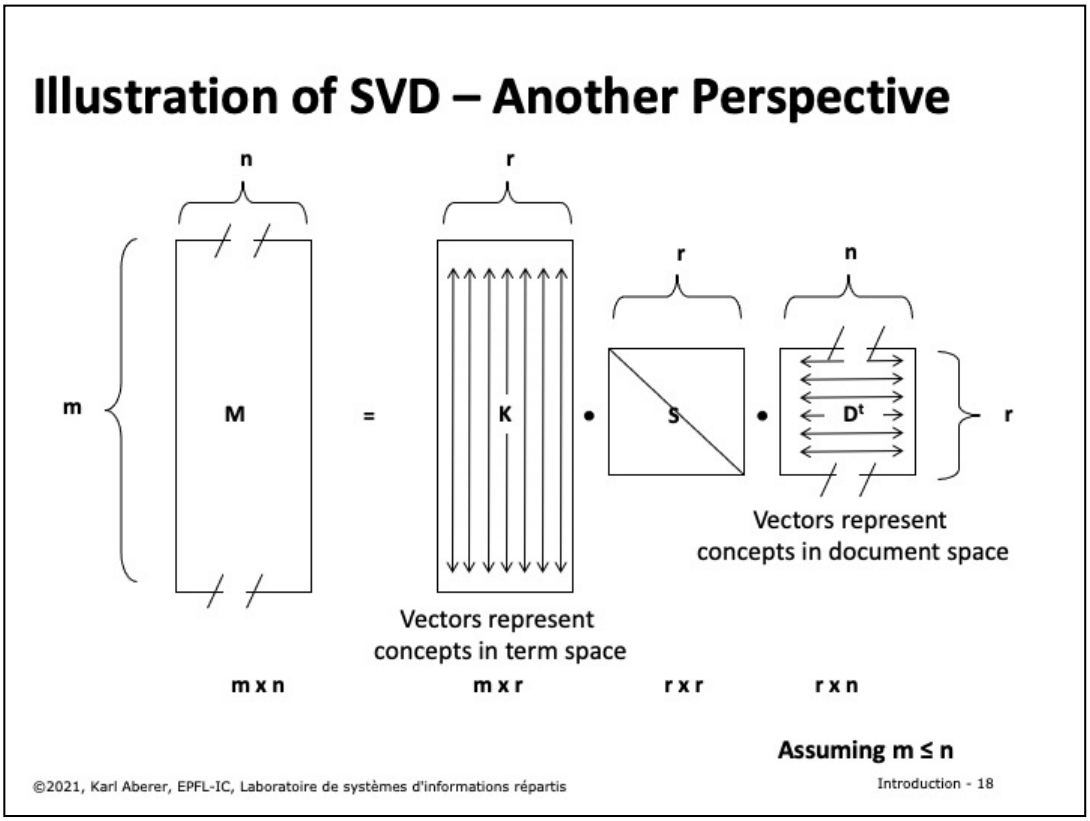
One way to understand of how the SVD extracts the important concepts from the term-document matrix is the following: the decomposition can be used to rewrite the original matrix as the sum of components that are weighted by the singular values. Thus we can obtain approximations of the matrix by only 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 outer product of vectors u =.(u_1,…,u_m) and v = (v_1,…,v_n) is given by the mxn matrix m with entries m_ij = u_i v_j.

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.



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

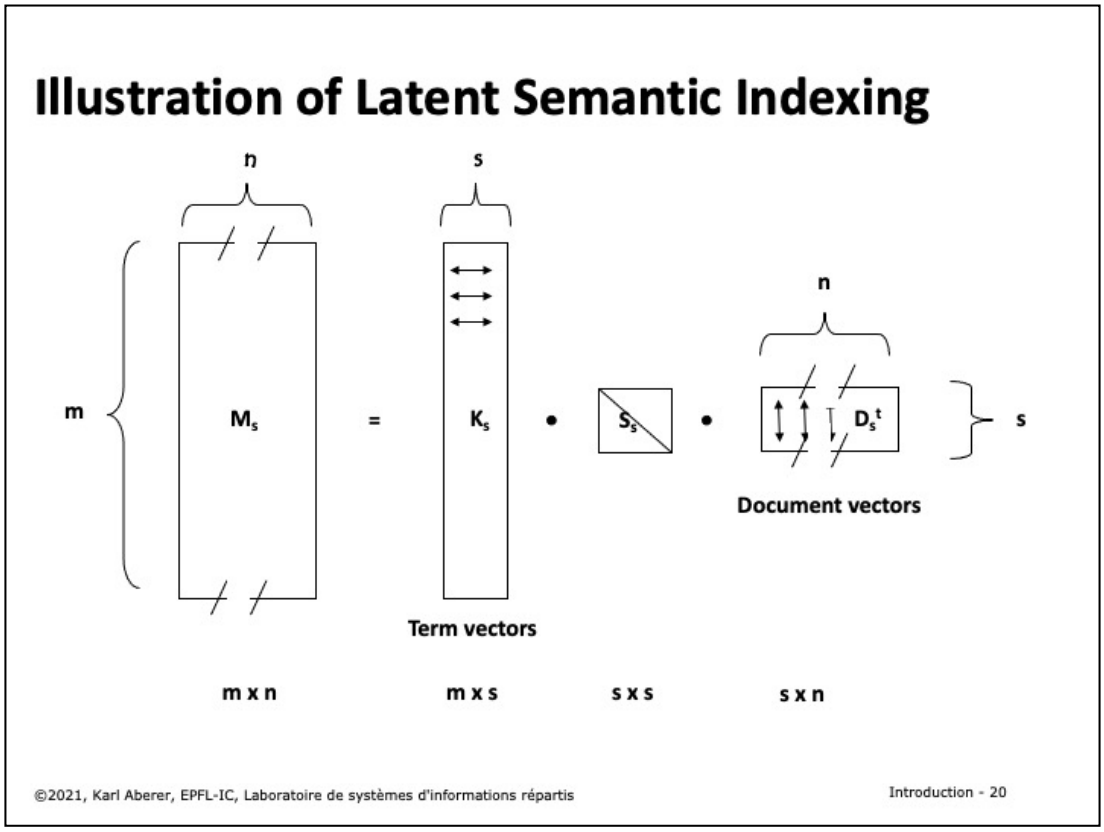
The resultant matrix is called M_s and is given by

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

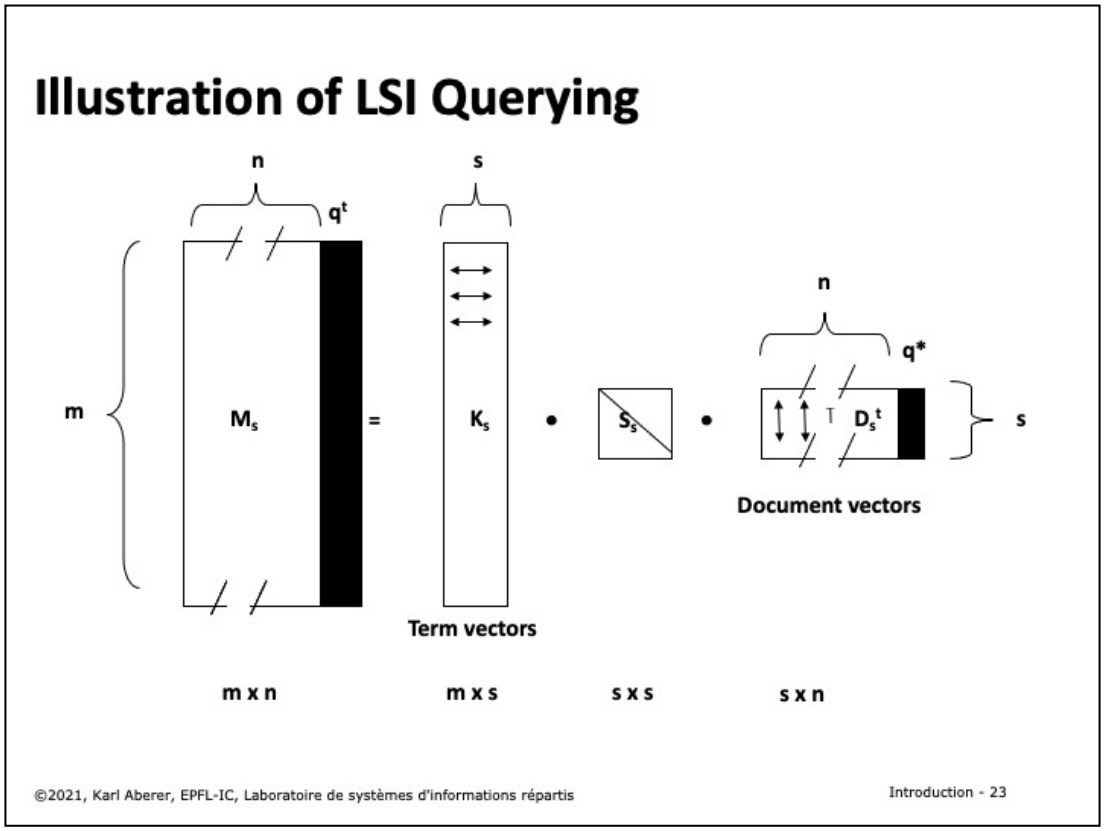
$q^* = q^t.K_s.S_s^{-1}$

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.

Implementation in Python

```
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(analyzer='word', ngram_range=(1,1), min_df = 2, stop_words = 'english')
features = tf.fit_transform(titles)
M = np.transpose(np.array(features.todense()))
```

```
# compute SVD
K, S, Dt = np.linalg.svd(M, full_matrices=False)

# LSI select dimensions
K_sel = K[:,0:2]
S_sel = np.diag(S)[0:2,0:2]
Dt_sel = Dt[0:2,:]
```

Results (s=2)

```
K_sel
```

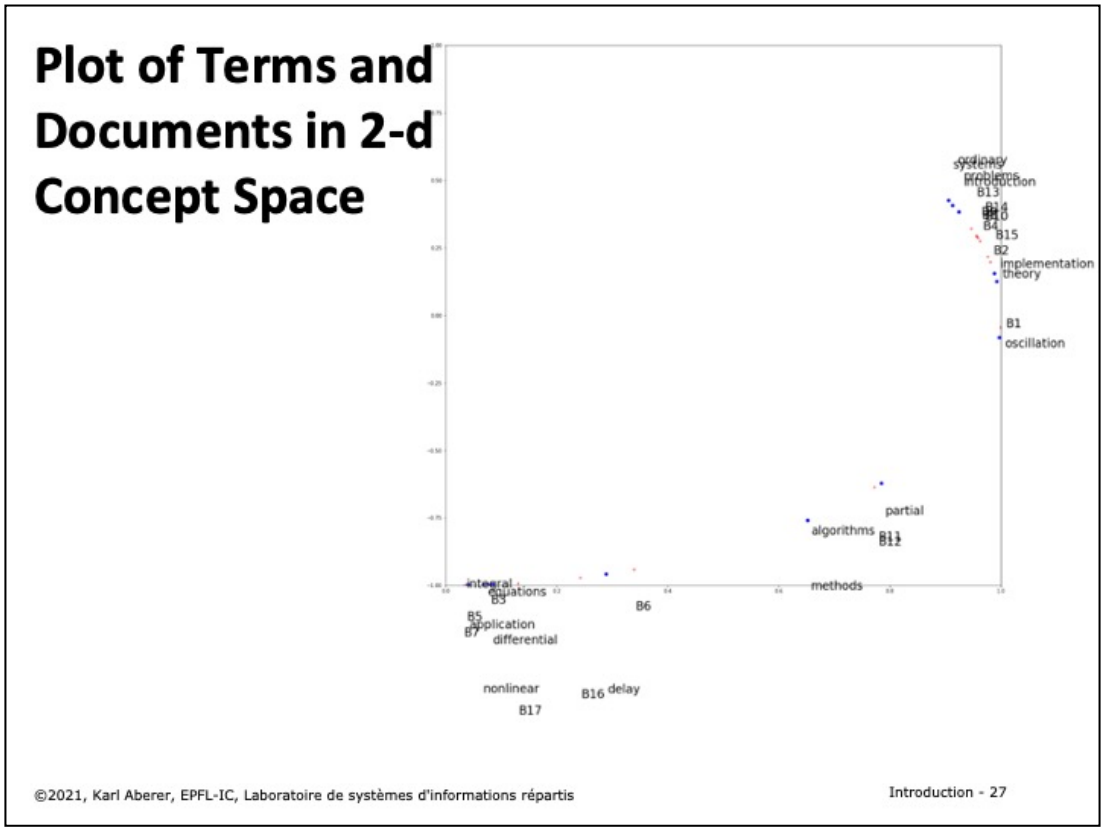
```
array([[ 0.01781272, -0.4729881 ],
       [ 0.03264057, -0.43230378],
       [ 0.15088442, -0.17568951],
       [ 0.55589867,  0.07082109],
       [ 0.6843092 ,  0.1075997 ],
       [ 0.01570413, -0.37133288],
       [ 0.09073864, -0.07173948],
       [ 0.01775573, -0.20943739],
       [ 0.19758761,  0.08201858],
       [ 0.11060875,  0.05205271],
       [ 0.19758761,  0.08201858],
       [ 0.15088442, -0.17568951],
       [ 0.20802226,  0.09313466],
       [ 0.01555703, -0.22913745],
       [ 0.10330872, -0.00853892],
       [ 0.14994428, -0.49674497]])
```

```
S_sel
```

```
array([[2.12109044, 0.        ],
       [0.        , 1.50037763]])
```

```
np.transpose(Dt_sel)
```

```
array([[ 0.18982901, -0.0083562 ],
       [ 0.32262142,  0.07171508],
       [ 0.04727092, -0.58437076],
       [ 0.33399497,  0.1003478 ],
       [ 0.01183895, -0.31440669],
       [ 0.03875274, -0.10771362],
       [ 0.01329859, -0.40782546],
       [ 0.3018997 ,  0.09205811],
       [ 0.07134057,  0.02202618],
       [ 0.33016936,  0.09458633],
       [ 0.29162009, -0.24019296],
       [ 0.29162009, -0.24019296],
       [ 0.29566823,  0.10051203],
       [ 0.33016936,  0.09458633],
       [ 0.40993831,  0.08283115],
       [ 0.03543573, -0.14179904],
       [ 0.05664377, -0.43260133]])
```



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.

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
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
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)
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

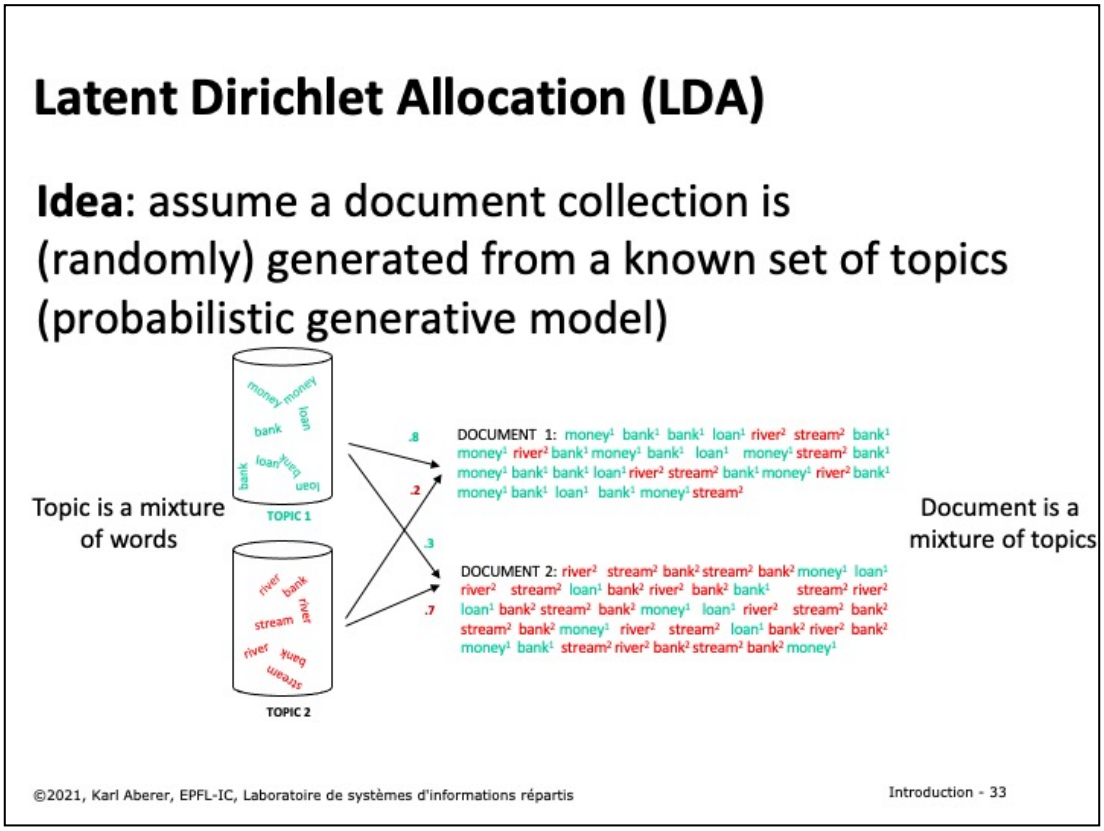
Latent Dirichlet Allocation

- 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.

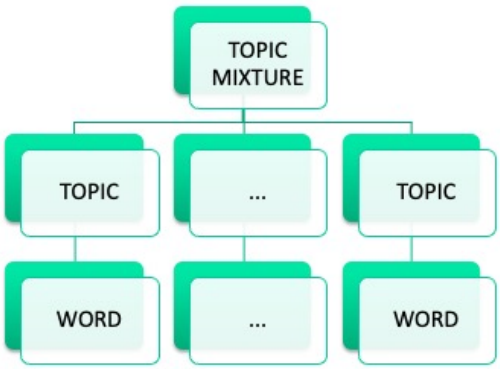
Similar to the probabilistic language model used in information retrieval, we have here another example of a probabilistic generative model.

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

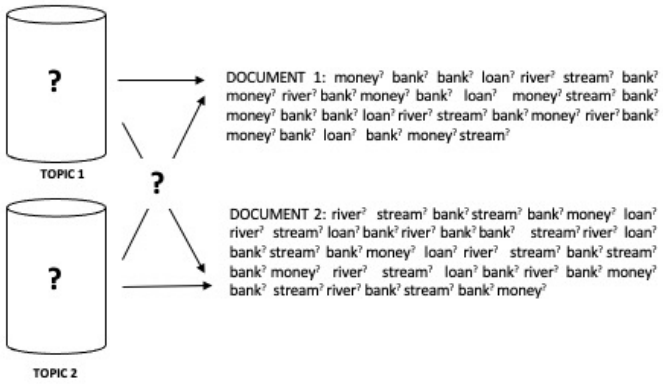
For every word position, sample a word from the chosen topic



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

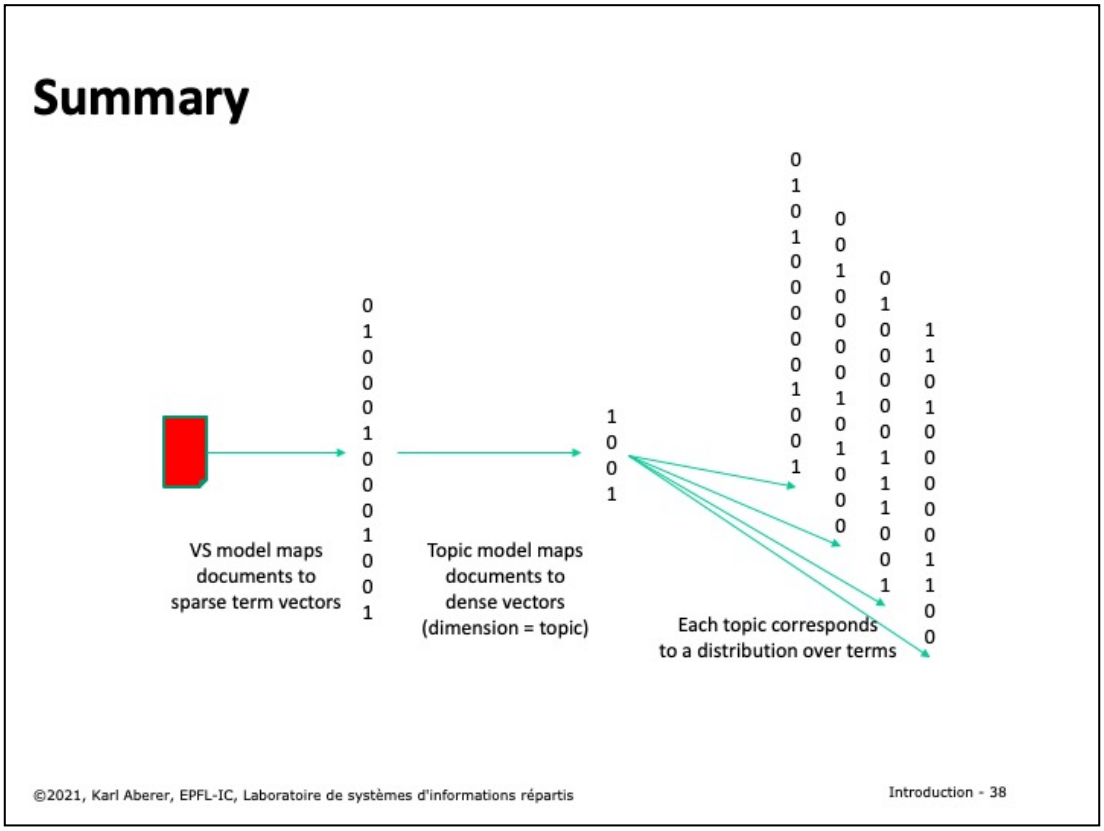
Unsupervised Learning of topics

- 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.



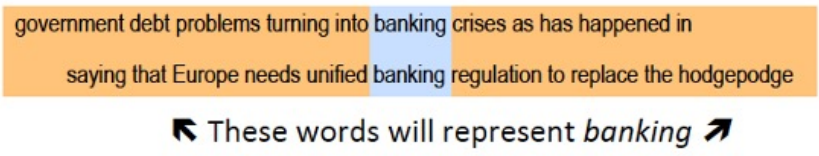
This illustration summarizes the connection among the vector space model, which was introduced for information retrieval and topic models that produce low-dimensional (dense) representation of documents.

10. 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)



↖ These words will represent *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.

Word Context

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* ↗

Context: words in a window of size n, e.g., n = 2

Example:

- Word: w = banking
- Context 1: $C_1(w)$ = {turning, into, crises, as}
- Context 2: $C_2(w)$ = {needs, unified, regulation, to}

Word-Context occurrence: $(w, c), c \in C_i(w)$

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.

Similarity-Based Representation

One of the most successful ideas in natural language processing

Two words are considered as similar,
when they have similar contexts

- Context captures both syntactic and semantic similarity
  - Syntactic: e.g. king – kings
  - Semantic: e.g. king – queen
- Implicitly used with the term-document matrix M
  - Less localized context
  - No distinction between roles of context and word

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.

Idea: Word Embeddings

Model how likely a word and a context occur together

Approach:

- Map words into a low-dimensional space (e.g., d = 200)
- Map context words into the same low-dimensional space
  - A different mapping for the same words
- Interpret the vector distance (product) as a measure for how likely the word and its context occur together

Due to projection in low-dimensional space, similar words and contexts should be close

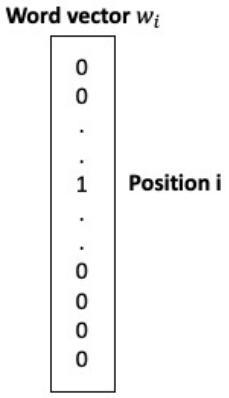
The basic idea for word embeddings is simple. Both words and context words are mapped into the same low-dimensional space. Their representations in that space should be similar, if the word and the context word occur often together. Thus we capture the information on word context in a low-dimensional 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.

Mapping to Concept Space

Vocabulary T of size m

Words $w_i \in T$ are encoded as "1-hot vectors" of length m, i.e., the component at position i is 1 all others are 0



We introduce now the representations of words and context words in an m-dimensional space. The representation where the i-th component of a vector is set to 1 for the i-th word in the vocabulary is called often the 1-hot encoding.

Model Parameters

Dimension of concept space d

The mapping to concept space is represented by matrices $W^{(w)}$ and $W^{(c)}$ of dimension $d \times m$

Mapping a word w_i

$$\boldsymbol{w}_i = W^{(w)} w_i \qquad \boldsymbol{w}_i \text{ is the column } i \text{ in } W^{(w)}$$

Mapping a context word c_i

$$\boldsymbol{c}_i = W^{(c)} w_i \qquad \boldsymbol{c}_i \text{ is the column } i \text{ in } W^{(c)}$$

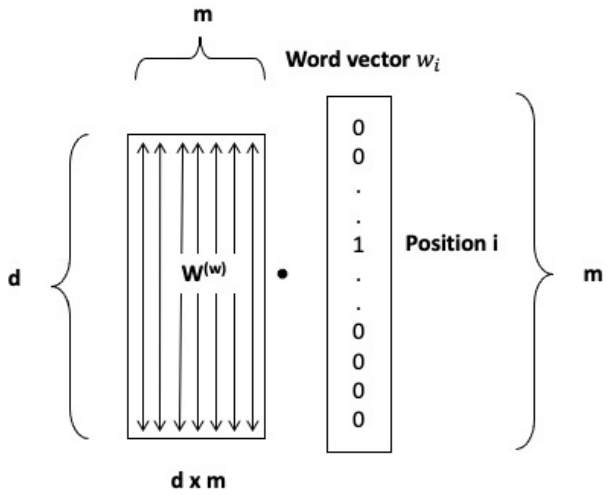
Parameters of model θ : All coefficients of $W^{(w)}$ and $W^{(c)}$

We introduce now the parameters of the word embedding model. First, we choose the dimension of concept space. 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 is specified by the two matrices $W^{(w)}$ and $W^{(c)}$ used to map words and context words into the concept space. Note that the two mappings are different, i.e. the same word will have a different representation in its role as word and context word.

For convenience we will denote in the following the coefficients of these two matrices by θ.

Illustration



Columns represent words of the vocabulary in concept space

Here we illustrate the mapping of words into concept space. Since the word vectors use the 1-hot encoding, we observe that the columns of the matrix $W^{(w)}$ in fact are exactly the representation of the words in the concept space. Similarly, we can also view the mapping of context words.

Learning the Model from Data

Given a document collection, how to learn the parameters θ?

Consider as prediction problem:

- Given a word, predict whether another word is a context word (skipgram)
- Given a set of context words, predict whether a word relates to this context (CBOW)

The problem we face now, is how to determine the model parameters, given a document collection. To do that we have to formulate a problem that can be solved using an optimization approach.

Since we are interested in the relation between words and context words, we can formulate a prediction problem. The model that we develop should be able to predict for a given word, which are the most likely context word (resulting in what we will be calling the skipgram model), or conversely, given a set of context words, what are the most likely words.

Statistical Learning

We aim to learn a function

$$f: S \rightarrow T$$

We have training data:

samples $f(s_1), f(s_2), \ldots, f(s_t)$

We define a parametrized function (model)

$$f_\theta: S \rightarrow T$$

Learning is obtaining good parameters θ

The prediction problem we have formulated corresponds to learning a function f.

We can learn an approximation f_θ of this function from the samples that we can obtain from the existing document collection.

For our problem this function will be the likelihood that a given word occurs together with some context words, or vice versa.

Loss Function

A function that measures how well the model fits the training data

Example:

$$J(\theta) = \sum_{t=1}^{s} \left(f(s_t) - f_\theta(s_t) \right)^2$$

Learning is applying an algorithm that minimizes this loss function

In order to determine whether a function f_θ approximates well the function from the training data, a loss function is introduced. It measures of how much the approximation function is deviating from the function applied to the training data.

Regularization

Regularization: limit model complexity

- In order to avoid overfitting
- In order the model to generalize well to new data

Possible approaches

- Limit the complexity of the model (e.g. linear)
- Add a regularization term to the loss function

Example:

$$J(\theta) = \sum_{t=1}^{s} \left(f(s_t) - f_\theta(s_t)\right)^2 + \gamma|\theta|_2$$

One key problem in statistical learning (or machine learning) is overfitting. If we provide a large number of parameters θ we can easily learn a good, or even perfect approximation of the function applied to the training data, but the function will perform poorly on other data. Therefore the idea is to “limit” the amount of information that can be stored in the model parameters, by minimizing some measure on them by including it into the loss function. This is called regularization.

Representation Learning

A learning algorithm that creates for some input data a new representation with desirable properties:

- Most often the representation is in some vector space
- For example, represent words of a vocabulary as vectors

Idea: create the word embeddings using representation learning

In general, a statistical learning algorithm can learn any kind of function, like predicting the co-occurrence of words and context-words, or predicting fraud in bank transactions, evolution of stock markets etc. while the parameters of the model θ are considered as “hidden parameters” without further use.

In representation learning, the model parameters are associated with some input data, and used to create an alternative representation for it. So no more the prediction itself is the main objective of learning, but obtaining an alternative representation of the training data. The prediction is introduced to formulate an optimization problem of which the solution will generate this representation.

Learning Probabilities

The function we would like to learn, is the probability $P_\theta(w_i|c)$ for the word w_i to appear with the context word c

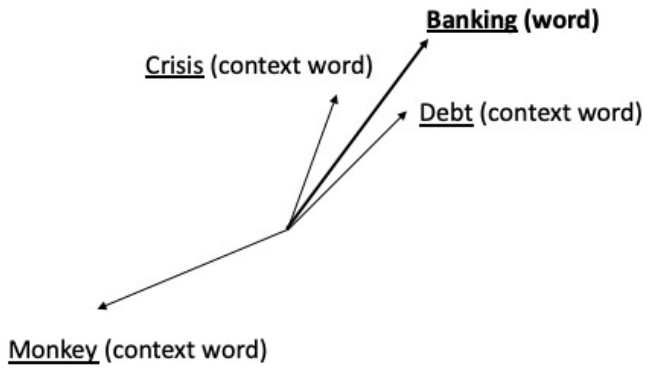
Deriving the probability from the representations

- Consider a word w and a context word c
- We require that if w and c are likely to co-occur their representations $\boldsymbol{w}$ and $\boldsymbol{c}$ should be similar, i.e., the scalar product $\boldsymbol{w} \cdot \boldsymbol{c}$ should be large

For learning representations we formulate a learning problem that requires to optimize a function to be learnt. For our purposes we require to learn the probability of words and context words to co-occur. The question is of how to derive such a probability from the model parameters θ, i.e. the vector representations of the words.

One consideration we can make is the following: if the words and context words have large probability of co-occurring we can require that their representations are similar, that means their cosine similarity is large. Then we could use this value to determine the probability, which also should be large, from cosine similarity.

Illustration



Banking • Crisis = + 100
Banking • Debt= + 80
Banking • Monkey = -150

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].

Softmax Function

Normalized scalar product?

- Produces values in [-1,1]
- $\sum_{i=1}^{m} P_\theta(w_i|c)$ does not add up to 1

Standard approach to convert a set of values into probabilities: softmax function

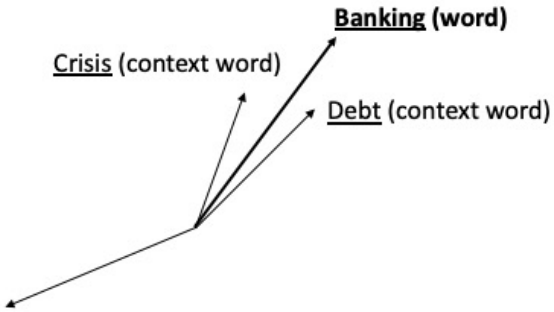
Values: $\boldsymbol{w_i} \cdot \boldsymbol{c}$ for $i = 1, \ldots, m$

$$P_\theta(w_i|c) = \frac{e^{\boldsymbol{w_i} \cdot \boldsymbol{c}}}{\sum_{j=1}^{m} e^{\boldsymbol{w_j} \cdot \boldsymbol{c}}}$$

The problem is of how to derive from the scalar product a probability. If we use the normalized scalar product the issue is that it does neither produce values in the interval [0,1] that could be interpreted as probability values, nor do probabilities add up to 1. E.g. the probability of a given context word to occur together with some word of the vocabulary should add up to 1.

To address this issue, there exists a standard approach to convert an (arbitrary) set of values into a probability distribution. First, applying the exponential function to the set of values (in our case the scalar products), produces positive values only, and second normalizing by the some of the exponentials makes sure that the values fall into the interval [0,1].

Illustration



Banking • Crisis = +3
Banking • Debt= +2
Banking • Monkey = -1

P(Banking, Crisis) = 0.72
P(Banking, Debt= 0.27
P(Banking, Monkey) = 0.01

```
from scipy.special import softmax
arr = [3,2,-1]
softmax(arr, axis=0)
```

```
array([0.72139918, 0.26538793, 0.01321289])
```

Word vectors that are similar, produce values close to 1

Applying the softmax function to the scalar products, produces now a probability distribution for a context word, co-occurring with some word of the vocabulary.

Loss Function for the Model

Given a text, maximize the probability for the observed context words

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{w \epsilon T} \prod_{c \in C(w)} P_\theta(w|c) =$$

$$\underset{\theta}{\operatorname{argmax}} \sum_{w \epsilon T} \sum_{c \in C(w)} \log P_\theta(w|c)$$

Issue: computing $P_\theta(w|c)$ is expensive (iterate over the complete vocabulary)

Based on these considerations we can now formulate a loss function that enables us to learn representations for the words and context words. The loss function aims to optimize the model parameters such that the product of the probability of all word-context word co-occurrences in the training data is maximized. A standard step to reformulate the optimization problem is to apply a logarithm to the loss function, which does not change the maximization problem, but converts the produt into a sum, which is more stable from the perspective of numerical processing.

This gives a first formulation of what is called usually the skipgram Model. One issue with the loss function is that it is expensive to compute, since for each word we have to iterate over the complete vocabulary to compute the probabilities.

Approximation

Learn a function that models the probability of a (w, c) pair to occur in the document collection

$$P_\theta : T \times T \rightarrow [0,1]$$

s.t. $P_\theta(w, c)$ close to 1 if (w, c) occurs in the document collection, and close to 0 otherwise

Using the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$

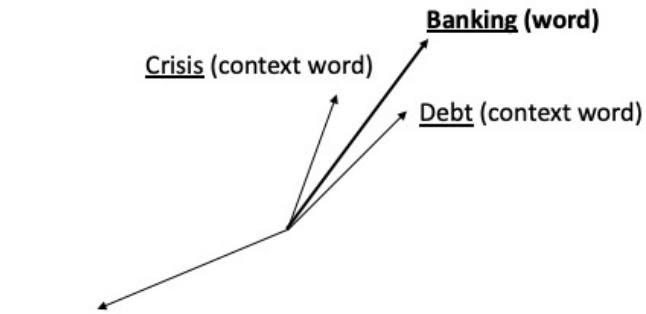
$$P_\theta(w, c) = \frac{1}{1 + e^{-\boldsymbol{c} \cdot \boldsymbol{w}}} = \sigma(\boldsymbol{c} \cdot \boldsymbol{w})$$

In order to arrive at a more efficient way to compute the representations, a simpler function can be learnt, which models directly the probability of a word and context word to co-occur, instead of introducing the probability distributions for all context words to co-occur with a specific word. This probability can be derived from the scalar products by applying the sigmoid function, which can be considered as the simpler version of the softmax functions when only a single data value is considered.

Performing this approximation avoids to iterate over the complete vocabulary for each word to compute the conditional probabilities, but rather consider some selected pairs.

Doing this approximation essentially introduces independence assumptions on the occurrences of word-context word pairs. It also changes the problem from a prediction problem (which the words that appear in a context) to a classification problem (which word, context pairs to occur)

Illustration



Banking • Crisis = +3
Banking • Debt= +2
Banking • Monkey = -1

P(Banking, Crisis) = 0.95
P(Banking, Debt= 0.88
P(Banking, Monkey) = 0.27

```
import numpy as np
def sigmoid(z):
    return 1/(1 + np.exp(-z))
[sigmoid(3),sigmoid(2),sigmoid(-1)]
```

```
[0.9525741268224334, 0.8807970779778823, 0.2689414213699951]
```


Here we illustrate the embedding of words and context words using the simpler version of the model. Now the probabilities can be independently computed for word-context word pair. Comparing to the previous example, where we used the softmax function, we see that the order of probability values did not change, but the absolute values are different.

Optimizing the Parameters

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

Maximizing the overall probabilities

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\epsilon D} P_\theta(w, c) \prod_{(w,c)\epsilon \widetilde{D}} \left(1 - P_\theta(w, c)\right) =$$

$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\epsilon D} \log \sigma(\boldsymbol{c} \cdot \boldsymbol{w}) + \sum_{(w,c)\epsilon \widetilde{D}} \log \sigma(-\boldsymbol{c} \cdot \boldsymbol{w})$$

Note: since the model is linear (simple), no regularization is used

Since the problem is now formulated as classification problem, it is necessary to consider also negative examples, i.e. word-context pairs that do not occur in the document collection. Assuming we have such negative examples, we can reformulate the overall optimization problem to obtain the model parameters.

Loss Function

We optimize the parameters by minimizing the following loss function

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

$$J_t(\theta) = -\log\big(\sigma(\boldsymbol{c_t} \cdot \boldsymbol{w_t})\big) - \sum_{k=1}^{K} \log(\, \sigma(-\boldsymbol{c_k} \cdot \boldsymbol{w_t}))$$

- $(\boldsymbol{w_t}, \boldsymbol{c_t})$ is a word-context pair
- c_k are negative examples of context words taken from a set $P_n(w_t)$

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 iterates over all word-context pairs in the document collection. For each term w it consists of a first component corresponding to a positive example,the specific (w,c) pair that occurs in the document collection, and a couple of negative examples for the same word. Note that for simplicity of notation we are omitting an index t for the words $\boldsymbol{c}_k$

An open question remains of how to obtain the negative examples.

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.

Solving the learning problem

Gradient Descent:

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

Stochastic Gradient Descent (SGD):

For every $(w_t, c_t) \in D$, update θ separately

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

Finally, given the loss function, finding the optimal parameters θ can then be done using standard methods from machine learning. θ can be determined using gradient descent, i.e. standard search for minima of a function using derivatives.

To make the problem computationally tractable, we apply stochastic gradient descent. We update the parameters for each sample of a word-context pair, separately. That implies that we have to consider only the corresponding component of the loss function.

Computing derivatives

$$J_t(\theta) = -\log\left(\sigma(\boldsymbol{c}_t \cdot \boldsymbol{w}_t)\right) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{c}_k \cdot \boldsymbol{w}_t))$$

with $\theta = \boldsymbol{w}_1, \ldots, \boldsymbol{w}_m, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$

therefore

$$\nabla_\theta J_t(\theta) = \left(\frac{\partial}{\partial \boldsymbol{w}_1} J_t, \ldots \frac{\partial}{\partial \boldsymbol{w}_m} J_t, \frac{\partial}{\partial \boldsymbol{c}_1} J_t, \ldots \frac{\partial}{\partial \boldsymbol{c}_m} J_t\right)$$

Then $\frac{\partial}{\partial \boldsymbol{w}_i} J_t = 0$ and $\frac{\partial}{\partial \boldsymbol{c}_i} J_t = 0$, if $\boldsymbol{w}_i \neq \boldsymbol{w}_t$ and $\boldsymbol{c}_i \neq \boldsymbol{c}_t$

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).

Updating model parameters

For J_t updates only affect rows in $W^{(w)}$ and $W^{(c)}$ that correspond to $\boldsymbol{w}_t$, $\boldsymbol{c}_t$and $\boldsymbol{c}_k$,

And the updates are according to the corresponding partial derivative

e.g.,

$$\boldsymbol{w}_t^{new} = \boldsymbol{w}_t^{old} - \alpha \frac{\partial}{\partial \boldsymbol{w}_t} J_t(\boldsymbol{w}_t^{old}, \boldsymbol{c}_t^{old}, \boldsymbol{c}_1^{old}, \dots, \boldsymbol{c}_K^{old})$$

Computing the Derivative

Some notation

$$x = \mathbf{c} \cdot \boldsymbol{w}, \mathrm{p} = \sigma(x), z_k = -\mathbf{c}_k \cdot \boldsymbol{w}, q_k = \sigma(z_k)$$

Then $$J_t(\boldsymbol{w}, \mathbf{c}, \mathbf{c}_1, \ldots, \mathbf{c}_k) = -\log(p) - \sum_{k=1}^{K} \log q_k$$

$$\frac{\partial J_t}{\partial \boldsymbol{w}} = \frac{\partial J_t}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial \boldsymbol{w}} + \sum_{k=1}^{K} \frac{\partial J_t}{\partial q_k}\frac{\partial q_k}{\partial z_k}\frac{\partial z_k}{\partial \boldsymbol{w}}$$

$$\frac{\partial J_t}{\partial p} = -\frac{1}{p}, \frac{\partial p}{\partial x} = \mathrm{p}(1-\mathrm{p}), \frac{\partial x}{\partial \boldsymbol{w}} = \mathbf{c}$$

Finally $$\frac{\partial J_t}{\partial \boldsymbol{w}} = -(1-\mathrm{p})\mathbf{c} + \sum_{k=1}^{K}(1-q_k)\,\mathbf{c}_k$$

Computing the derivative of the loss function, for updating the model parameters, requires evaluating derivatives along the operators that compose the loss function. This step is part of what is in machine learning commonly called backpropagation. We illustrate the essential steps of this operation for computing $\frac{\partial J_t}{\partial \boldsymbol{w}}$. To that end it is helpful to identify the different components that constitute the loss function and compose then the derivatives of those components using the chain rule. This reveals that the updates to the parameters result in simple operations. As an exercise one can complete the computation for the other derivatives.

.

Observations

1. The function P_θ can be learnt from a document corpus
2. We can apply statistical learning to learn the function
3. As we learn the parameters θ, implicitly by learning the function we learn a representation for words (the word embedding vectors)

CBOW Model

Consider a pair $(w, \{c_1, \ldots, c_n\})$

Does it origin from the data?

Compute $\bar{\boldsymbol{c}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{c_i}$

$$P_\theta(w, \{c_1, \ldots, c_n\}) = \frac{1}{1 + e^{-\bar{\boldsymbol{c}} \cdot \boldsymbol{w}}} = \sigma(\bar{\boldsymbol{c}} \cdot \boldsymbol{w})$$

In the CBOW model, the idea is to predict from a complete context a corresponding word. We can formulate this similarly as we did for the skipgram model, only that we use a set of context words. To take into account the set of context wrods, we take the average vector of their representation, to determine the similarity of the context with the target word w. The rest of the processing proceeds as for the skipgram model.

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.

Comments

- There exists an alternative approach to learning using hierarchical softmax (different loss functions)
  - Uses the original formulation of the approach
  - Negative sampling is faster
- CBOW produces better results for frequent terms, skipgram for rare terms

The approaches that we presented avoided the use of the softmax functions to generate probability distributions. There exist implementations of the approach as we introduced initially, based on a method called hierarchical softmax. This method is based on a hierarchical aggregation of the softmax values, which makes the problem computationally more tractable. Still negative sampling is the faster approach.

Emperically, CBOW produces better representations for frequent terms, whereas skipgram works better for rare terms.

A column of matrix $W^{(c)}$ represents

1. How relevant word c is for each concept
2. How often a context word c co-occurs with all words
3. A representation of word c in concept space

A word embedding for given corpus …

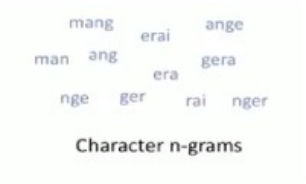
1. depends only on the dimension d
2. depends on the dimension d and number of iterations in gradient descent
3. depends on the dimension d, number of iterations and chosen negative samples
4. there are further factors on which it depends

Fasttext

Fasttext introduces the idea of using word n-grams (phrases) and **subword embeddings**

- Build embeddings for character n-grams
- Enable processing of unseen words

Example (in French): mangerai



The word embedding models we have presented where based on the assumption that the vocabulary consists of the words found in the text corpus. Fasttext is a variant of word embeddings that has been developed by Facebook research. It incorporates in particular two important ideas:

- The use of word n-grams, respective phrases: often phrases carry a different meaning than their constituent words. Considering phrases (that are sufficiently frequent) can help to produce better representations of text and capture meaning that cannot be assigned to the constituents of the phrase. As an extreme example one might consider the phrase “The Who” which designates a rock band, whereas the constituent words have no meaning that would be related to music in any way.
- The use of character n-grams: many languages, e.g. latin languages, have complex grammatical systems with similar word variations. Often some of them might even not occur in a training corpus. By using so-called subword embeddings the meaning identified for seen words, can so be transferred also to unseen works. This also works across languages, as often languages inherit expressions from foreign languages, or words are same or similar across different languages.

Subword Embeddings

For a given word create representations for all of its subwords S_w, including the word itself

- Add special characters at the start and end of the word
- Create all n-grams

Example: S_w for word "mangerai", n=3

- {<mangerai>, <ma, man, ang, nge, ger, era, rai, ai>}

Representation of w: $\boldsymbol{w} = \sum_{s \in S_w} \boldsymbol{s}$

When using subword embeddings, the representation of a word is aggregated of the representation of all of its constituent n-grams, by adding up the n-gram representations.

Byte Pair Encoding (BPE)

Subword embeddings potentially generate large numbers of possible tokens, many of which are not useful

- Large vocabulary implies training the model becomes expensive

BPE allows to create smaller vocabularies, while capturing essential subword information

- Statistically analyze text for frequent character sequences
- Replace frequent sequences by tokens

When using for subword embeddings character n-grams one faces the problem that the number of tokens, i.e. the size of the vocabulary, may become very large, which in turn makes training the models very expensive. In addition, many of the tokens are rare and potentially not very useful. In order to address this problem a method called byte pair encoding is used in order to obtain smaller vocabularies and select only frequent subwords. For this method, first the text needs to be analysed to determine the frequent tokens.

Example BPE

"Do Do! Do! Do! Do? Do! Done Done! Done? Done."

All subwords of size n=2 (character 2-grams)

{Do, o!, o?, o., on, ne, e!, e?, e.}

Better solution

{Do, Do!, Done, !, ?, .}

Tokenized text

[Do, Do!, Do!, Do!, Do, ?, Do!, ., Done, Done, !, Done, ?, Done, .]

This example illustrates the idea of BPW. Instead of blindly selecting all character n-grams, a set of frequent n-grams is chosen and used to tokenize the text.

BPE Algorithm

Add word boundaries

"Do</w>Do!</w>Do!</w>Do!</w>Do?</w>Do!</w>Done</w>Done!</w>Done?</w>Done."

```
Initial Vocabulary V = all characters
while i < max_iterations
     determine most frequent pair of tokens
     add pair to V as a new token
     update token frequencies
     i = i+1
```


For running the BPW algorithm first a special token for word boundaries is added (in this case </w>).

The BPE algorithm proceeds iteratively.

It determines in every iteration the most frequent pair of tokens and groups to create a new token. Then token frequencies are updated with the new set of tokens. The algorithm terminates after a predetermined number of tokens has been created.

In order to determine token frequencies, the algorithm can first scan the document corpus to determine all n-gram frequencies (up to a given bound).

BPE Example

"Do</w>Do!</w>Do!</w>Do!</w>Do?</w>Do!</w>Done</w>Done!</w>Done?</w>Done."

V = {</w>,D,o,!,?,n,e,.}

i = 1, most frequent pair: D,o: 9

V = {</w>,D,o,!,?,n,e,.,v0 = Do}

New text:

v0</w>v0!</w>v0!</w>v0!</w>v0?</w>v0ne</w>v0ne!</w>v0ne?</w>v0ne.

Tokenized text:

[Do, Do, !, Do, !, Do, !, Do,?, Do, !, n, e, Do, n, e , !, Do, n, e, ?, Do, n, e, .]

i = 2: most frequent pair: !, </w>: 5

V = {</w>,!</w>, D,o,!,?,n,e,.,v0 = Do, v1 = !</w>}

New text:

v0</w>v0v1v0v1v0v1v0?</w>v0v1v0ne</w>v0nev1v0ne?</w>v0ne.]

Tokenized text:

[Do, Do, !, Do, !, Do, !, Do,?, Do, !, n, e, Do, n, e , !, Do, n, e, ?, Do, n, e, .]

Here we illustrate the first two steps of the algorithm. Note that in the second step the tokenization does not change, as we introduce a token corresponding to !</w>. Having tokens that combine a character n-gram with a word boundary is important to identify character sequences that would typically occur at the end (or beginning) of a word and to distinguish them from character sequences that occur in the middle of a word.

For example, the sequence "ed" at the end of of an English word (like "worked") implies a different meaning of this character sequences at the beginning of a word (like "education") or in the middle of a word (like "medicine")

Alternative Approaches

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

"solid" is related to "ice" but unrelated to "steam":
we expect a larger ratio of co-occurrence probabilities

"gas" is related to "steam" but unrelated to "ice":
we expect a smaller ratio of co-occurrence probabilities

"water" is related to both "ice" and "steam":
we expect a ratio of co-occurrence probabilities that is close to 1 (both large)

A random word that is unrelated to both "ice" and "steam":
we expect a ratio of co-occurrence probabilities that is close to 1 (both small)

The skipgram and CBOW models are one variant among a number of similar models that have been proposed recently to create word representations that capture semantics.

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. In the Glove paper it is stated: "Compared to the raw probabilities, the ratio is better able to distinguish relevant words (solid and gas) from irrelevant words (water and fashion) and it is also better able to discriminate between the two relevant words" This may be interpreted as follows: the ratio of probabilities captures the semantics of the meaning of solid and gas, without referring to specific case in which this meaning occurs, namely the context of water.

Global Co-occurrence Count

Word embeddings with global vectors (GloVe)

- Denote by x_{ik} the global co-occurrence count of words w_i and w_k in the vocabulary, where w_i is the center word and w_k is the context word
- Note: $x_{ik} = x_{ki}$
- Denote by x_i the total number of context words occurring for w_i

Then

$$P(w_i|w_k) = \frac{x_{ik}}{x_i}$$

In order to consider the property illustrated before, Glove uses the global occurrence counts among words. In this way it incorporates into the model also global statistics on the co-occurrence of words, different to skipgram/CBOW which exploit local information. Using the global co-occurrence counts we can determine the co-occurrence probability.

Modeling Ratios

Find a function f such that for three words w_i, w_j, w_k

$$f(\boldsymbol{w}_i, \boldsymbol{w}_j, \boldsymbol{w}_k) = \frac{P(w_i|w_k)}{P(w_j|w_k)}$$

If we choose

$$f(\boldsymbol{w}_i, \boldsymbol{w}_j, \boldsymbol{w}_k) = e^{(\boldsymbol{w}_i - \boldsymbol{w}_j)\cdot \boldsymbol{w}_k} = \frac{e^{\boldsymbol{w}_i \cdot \boldsymbol{w}_k}}{e^{\boldsymbol{w}_j \cdot \boldsymbol{w}_k}}$$

we get

$$e^{\boldsymbol{w}_i \cdot \boldsymbol{w}_k} = P(w_i|w_k) = \frac{x_{ik}}{x_i}$$

In order to formalize the observation that we have described, we need to design a function that captures rations between co-occurrence probability. We are free to choose any function f for that purpose. For technical consideration in Glove this function has been chosen to be of a very specific form, as the exponential of the scalar product of the difference of the representation of two (context) words with the representation of a a given word. When using this form of function, it follows that the conditional probability of co-occurrence is modelled as the exponential for the scalar product of the representations of the two words.

Glove Loss Function

With word w_i and context words w_k

Assume

$$e^{\boldsymbol{w}_i \cdot \boldsymbol{w}_k} = P(w_i|w_k) = \frac{x_{ik}}{x_i}$$

taking the logarithm

$$\boldsymbol{w}_i \cdot \boldsymbol{w}_k = \log x_{ik} - \log x_i$$

Adding additional bias terms for w_k

$$\boldsymbol{w}_i \cdot \boldsymbol{w}_k + \boldsymbol{b}_i + \boldsymbol{b}_k \approx \log x_{ik}$$

Once we have defined the condition that the representation of words has to satisfy, we can derive a loss function. By taking the logarithm we obtain a linear relationship between the scalar product and the statistics on co-occurrence. Apart from the co-occurrence frequencies, also a term for each word is introduced in the loss function (this is design decision that has been taken by the authors of Glove).

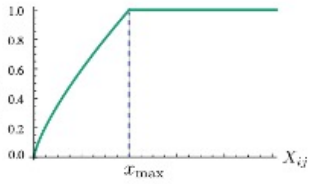
Glove Loss Function

Squared error loss with weights

$$\sum_{i=1}^{m}\sum_{j=1}^{m} h(x_{ij})(\boldsymbol{w}_i \cdot \boldsymbol{w}_j + \boldsymbol{b}_i + \boldsymbol{b}_j - \log x_{ij})^2$$

Weight function

$$h(x) = \min\left(\left(\frac{x}{x_{max}}\right)^{\beta}, 1\right)$$



With $\beta = 0.75$ and $x_{max} = 100$

Finally the loss function is enhanced by introducing a weight function h. It aims at resolving two problems:

- If x_ij equals zero, the loss function would be ill-defined. The limit lim_x->0 f(x) log^2(x) is finite, making the function well-defined
- Small values of x_ij carry in general less information and should thus have lower weight.

Properties of Word Embeddings

1. Similar terms are clustered
2. Syntactic and semantic relationships encoded as linear mappings
3. Dimensions can capture meaning

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

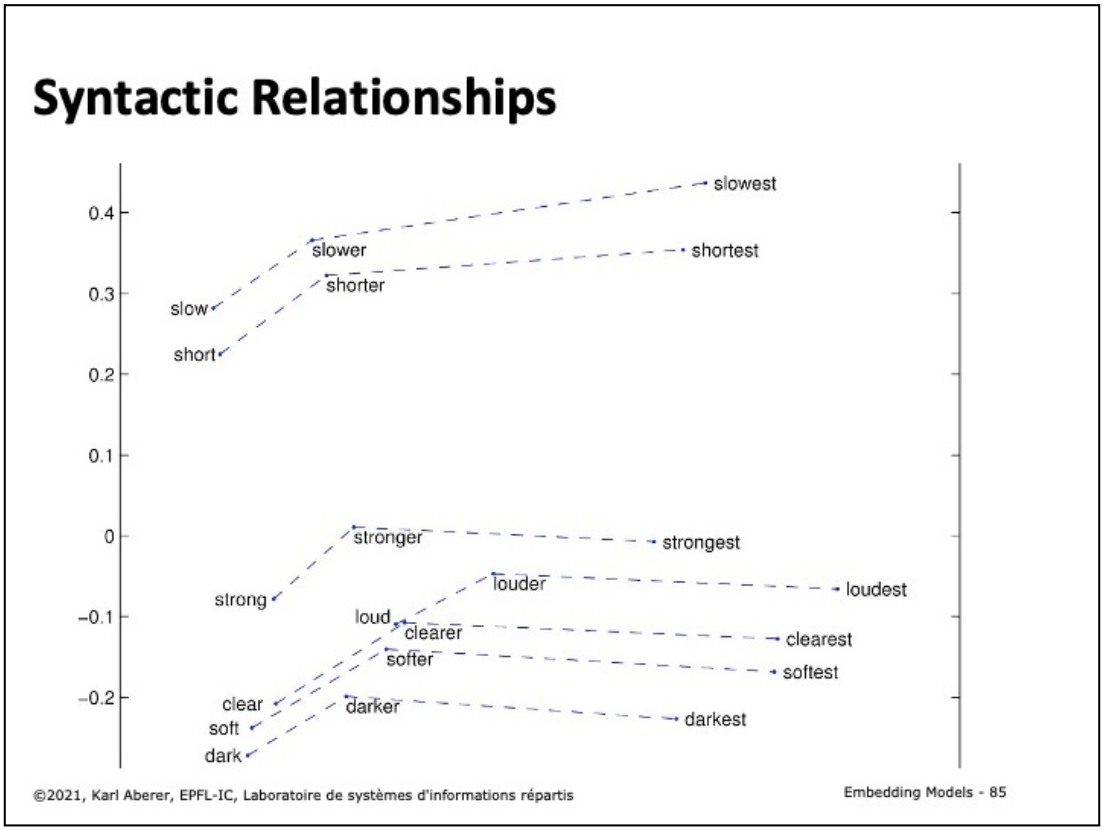
litoria

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).

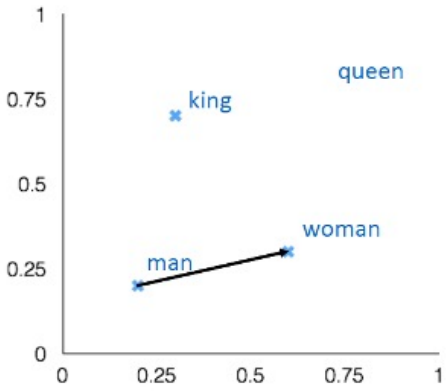


Word embeddings also capture syntactic relationships, like singular-plural or comparative and superlative, as shown here. This type of visualizations is obtained by projecting from the d-dimensional word embedding space (appropriately) into a 2-dimensional space.

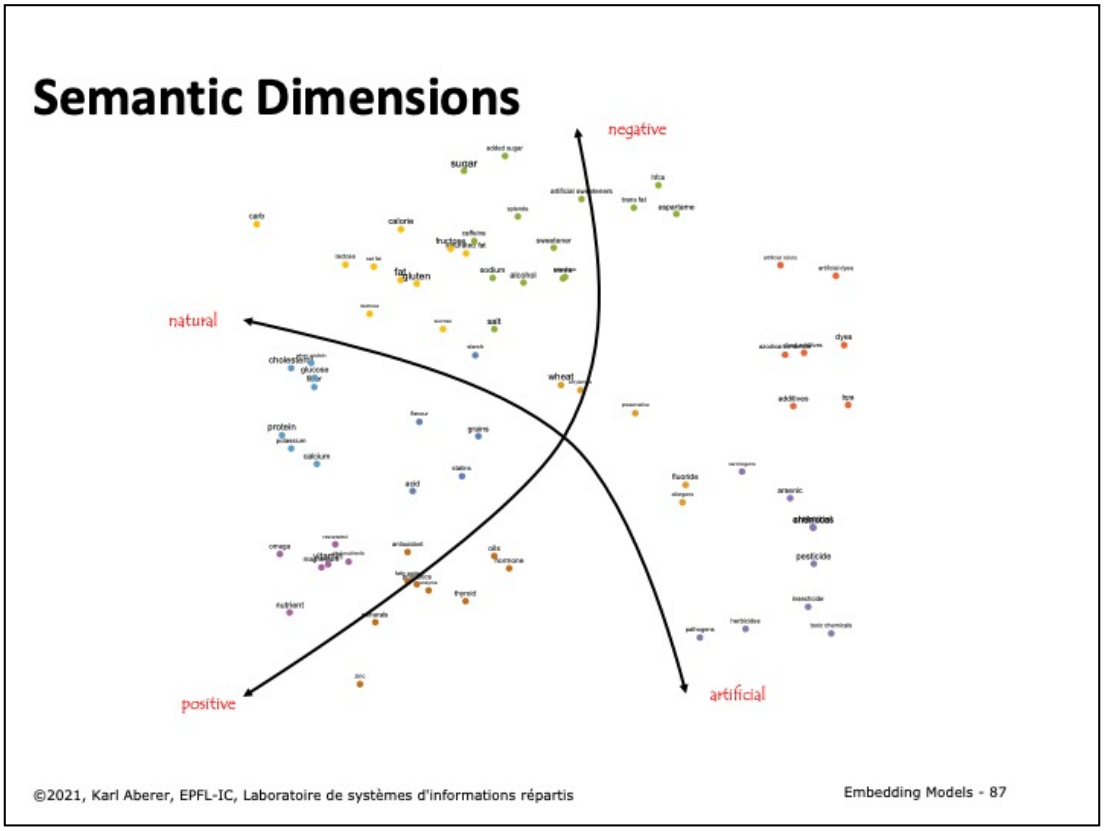
Word Analogies: semantic relationships

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

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

+2000

- 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

+8000

- 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

+8000