Information Retrieval

WS 2017 / **2018**

Lecture 10, Tuesday January 9th, 2018
(Latent Semantic Indexing)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

Overview of this lecture

- Organizational
  - Your experiences with ES9 — Clustering
- Contents
  - Latent Semantic Indexing (LSI) — Motivation + examples
  - Singular Value Decomp. (SVD) — Definition + how to compute
  - Using LSI for retrieval — Three variants
  - ES10: another pencil and paper sheet, for understanding and practicing the linear algebra (in particular, the SVD)

In past years, we always had an implementation exercise for LSI. This was nice, because one could see the results. But most people coded just it without understanding how it really works.

Experiences with ES9

- Summary / excerpts
  - Many of you liked this type of exercise ("math with meaning" + the kind of task you also get in the exam)
  - Some of you don't like theory … there is no escape from it!
  - Several of you are unsure about how much to write and when it is enough … in case of doubt, ask on the forum!
  - Many of you were busy with your holidays and did not attend the last lecture
  - Minor bug in the assumptions for Exercise 3, which was quickly discussed in the forum and fixed though
  - "Cookie monster reminded me of missing out on the cookies in Lecture 7 due to watching the live stream"

Latent Semantic Indexing 1/8

- Motivation
  - Let's look at our example toy collection from L8 again:

    D_1 and D_2 and D_3 are "about" surfing the web

    D_5 and D_6 are "about" surfing on the beach

    internet and web are **synonyms**, surfing is a **polysem**
    = means different things in different context

	D_1	D_2	D_3	D_4	D_5	D_6
internet	1	1	0	1	0	0
web	1	0	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Latent Semantic Indexing 2/8

- Motivation
  - Let's look at the query web surfing again, using dot-product similarity as explained in L8
  - Then $sim(D_3, Q) > sim(D_2, Q) = sim(D_5, Q)$

    But D_2 seems just as relevant for the query as D_3, only that the word "internet" is used instead of "web"

	REL	REL	REL	(REL)	NOT	NOT	
	D_1	D_2	D_3	D_4	D_5	D_6	Q
internet	1	1	0	1	0	0	0
web	1	0	1	1	0	0	1
surfing	1	1	1	2	1	1	1
beach	0	0	0	1	1	1	0
	2	1	2	3	1	1	

Latent Semantic Indexing 3/8

- Conceptual solution

	REL D_1	REL D_2	REL D_3	(REL) D_4	NOT D_5	NOT D_6	Q
internet	1	1	**1**	1	0	0	0
web	1	**1**	1	1	0	0	1
surfing	1	1	1	2	1	1	1
beach	0	0	0	1	1	1	0
	2	2	2	3	1	1	

Add the missing synonyms to the documents

Then indeed: $\text{sim}(D_1, Q) = \text{sim}(D_2, Q) = \text{sim}(D_3, Q)$

The goal of LSI is to do something like this **automagically**

Latent Semantic Indexing 4/8

- A simple but powerful observation

	B_1	B_1	B_1	B_1+B_2	B_2	B_2
	D_1	**D_2**	**D_3**	**D_4**	**D_5**	**D_6**
internet	1	1	**1**	1	0	0
web	1	**1**	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

B_1	**B_2**
1	0
1	0
1	1
0	1

The modified matrix has **column rank 2**

That is, we can write each column as a (different) linear combination of the same two base columns (B_1 and B_2)

Note 1: the original matrix had column rank 4

Note 2: one can prove that **column rank = row rank**

Latent Semantic Indexing 5/8

- Matrix factorization

	D_1	D_2	D_3	D_4	D_5	D_6
1	1	1	**1**	1	0	0
2	1	**1**	1	1	0	0
3	1	1	1	2	1	1
4	0	0	0	1	1	1
	1	2	3	4	5	6

=

	B_1	B_2
1	1	0
2	1	0
3	1	1
4	0	1

•

B_1	B_1	B_1	B_1+B_2	B_2	B_2
D'_1	D'_2	D'_3	D'_4	D'_5	D'_6
1	1	1	1	0	0
0	0	0	1	1	1
1	2	3	4	5	6

Equivalently: the 4 x 6 term-document matrix can be written as a product of a 4 x 2 matrix with a 2 x 6 matrix

The base vectors B_1 and B_2 are the underlying **"concepts"**

The vectors D'_1, …, D'_6 are the representation of the documents in the (lower-dimensional) **"concept space"**

Latent Semantic Indexing 6/8

- The goal of LSI
  - Given an $m \times n$ term-document matrix A and $k < \text{rank}(A)$
  - Then find a matrix A' of (column) rank k such that the difference between A' and A is **as small as possible**
  - Formally: $A' = \text{argmin}_{A' \text{ m x n with rank k}} \|A - A'\|$
  - For the $\|\ldots\|$ we take the so-called **Frobenius-norm**

    This is defined as $\|D\| := \text{sqrt}(\Sigma_{ij} D_{ij}^2)$

    The reason for using this norm is purely technical: that way, the math on the next slides works out nicely

Latent Semantic Indexing 7/8

the columns of U are $L2$ normalized to 1 and orthogonal to each other

- How to compute such an approximation A'
  - We first compute the so-called **singular value decomposition (SVD)** of the given matrix A :

    the rows of V are L2-normalized and orthogonal to each other

  - Theorem: for any m x n matrix A of rank r, there exist U, S, V such that $\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}$, and where

    $\underset{m \times n}{\mathbf{A}} = \underset{m \times r}{\mathbf{U}} \cdot \underset{r \times r}{\mathbf{S}} \cdot \underset{r \times n}{\mathbf{V}}$

    U is an **m x r** matrix with $U^T \cdot U = I_r$ $\quad$ I_r = the r x r identity matrix

    S is an **r x r** matrix with non-zero entries only on its diag.

    V is an **r x n** matrix with $V \cdot V^T = I_r$
  - The decomposition is **unique** up to simultaneous permutation of the rows/columns of U, S, and V

    Standard form: diagonal entries of S positive and sorted

Latent Semantic Indexing 8/8

- Using the SVD, our task becomes easy
  - Let A = U · S · V be the SVD of A
  - For a given k < rank(A) let

    $\mathbf{U_k}$ = the first k columns of U   now an m x k matrix

    $\mathbf{S_k}$ = the upper k x k part of S   now a k x k matrix

    $\mathbf{V_k}$ = the first k rows of V   now a k x n matrix
  - Note that U_k is column-orthonormal just like U

    That is, $U_k^T \cdot U_k = I_k$ … and similarly, $V_k \cdot V_k^T = I_k$
  - Let $\overset{m\times n}{\mathbf{A_k}} = \overset{m\times k}{\mathbf{U_k}} \cdot \overset{k\times k}{\mathbf{S_k}} \cdot \overset{k\times n}{\mathbf{V_k}}$

    Then A_k is a matrix of rank k that minimizes $\|A - A_k\|$

Computing the SVD 1/4

iterative algorithm

- Overview
  - Can be computed from the **Eigenvector decomposition**

    See next slides for some of the mathematics behind
  - In practice, the more direct Lanczos method is used
  - This method has complexity **O(k · nnz)**, where **k** is the rank and **nnz** is the number of non-zero values in the matrix

    Note that for term-document matrices nnz ≪ n · m
  - When implementing this for large matrices in Python, one should use the method **scipy.sparse.linalg.svds**

    Using **numpy.linalg.svd** on large matrices takes forever

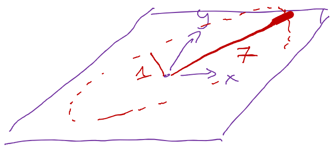
Computing the SVD 2/4

$$\overbrace{\begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}}^{B} = \overbrace{\frac{\sqrt{2}}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}}^{U} \overbrace{\begin{pmatrix} 7 & 0 \\ 0 & 1 \end{pmatrix}}^{diag(\ldots)} \overbrace{\frac{\sqrt{2}}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}}^{U^T}$$

- The Eigenvector Decomposition (EVD) $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \xrightarrow[\text{norm.}]{} \frac{1}{2}\sqrt{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
  - A real symmetric $n \times n$ matrix B has n pairwise orthogonal unit eigenvectors $u_1, \ldots, u_n$ (with eigenvalues $\lambda_1, \ldots, \lambda_n$)

    That is, $B \cdot u_i = \lambda_i \cdot u_i$ and $u_i \bullet u_j = 0$, for $i \neq j$, and $|u_i| = 1$

    Equivalently, $B = U \cdot \text{diag}(\lambda_1, \ldots, \lambda_n) \cdot U^T$ … and $U \cdot U^T = I$
  - To compute all E-values $\boldsymbol{\lambda}$, simply solve $\mathbf{det(B - \lambda \cdot I) = 0}$
  - Example: $B = \overbrace{\begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}}^{2\times 2}$



One Eigenvector : $x_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ Eigenvalue

$B \cdot x_1 = \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4+3 \\ 3+4 \end{pmatrix} = \begin{pmatrix} 7 \\ 7 \end{pmatrix} = 7 \cdot x_1$

Other Eigenvector : $x_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ Eigenvalue

$B \cdot x_2 = \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 4-3 \\ 3-4 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} = 1 \cdot x_2$

λ is Eigenvalue $\Leftrightarrow \exists x \neq 0 \quad B \cdot x = \lambda \cdot x$ $n \times n$ identity matrix

$\Leftrightarrow \exists x \neq 0 \quad (B - \lambda \cdot I) \cdot x = 0$

$\Leftrightarrow \det(B - \lambda \cdot I) = 0$

Computing the SVD 3/4

- From EVD to SVD, part 1
  - Let **A** be an arbitrary **m x n** matrix
  - Then the matrix $\mathbf{A} \cdot \mathbf{A}^\mathbf{T}$ is $\overset{m \times m}{\text{square}}$ and symmetric

    $\underbrace{(A \cdot A^T)^T}_{m \times m} = \overset{m \times n}{A^{T^T}} \cdot \overset{n \times m}{A^T} = A \cdot A^T$
  - Hence there exists an orthonormal matrix U and a diagonal matrix S_1 such that

    $A \cdot A^T = U \cdot S_1 \cdot U^T$
  - Analogously, the matrix $\underset{n \times m}{\mathbf{A}^\mathbf{T}} \cdot \underset{m \times n}{\mathbf{A}}$ is $\overset{n \times n}{\text{square}}$ and symmetric
  - Hence there also exists an orthonormal matrix V and a diagonal S_2 such that

    $A^T \cdot A = V \cdot S_2 \cdot V^T$

Computing the SVD 4/4

- From EVD to SVD, part 2
  - Let us assume that a decomposition $\overset{m\times n}{\mathbf{A}} = \overset{m\times r}{\mathbf{U}} \cdot \overset{r\times r}{\mathbf{S}} \cdot \overset{r\times n}{\mathbf{V}}$ exists
  - Then $\mathbf{A \cdot A^T = U \cdot S^2 \cdot U^T}$ and $\mathbf{A^T \cdot A = V^T \cdot S^2 \cdot V}$

    That is: the **columns of U** are the Eigenvectors of $A \cdot A^T$ and the **rows of V** are the Eigenvectors of $A^T \cdot A$

    And the singular values (diagonal values from S) are just the square roots of the eigenvalues (which are the same for $A \cdot A^T$ and for $A^T \cdot A$)

$$A \cdot A^T = (U \cdot S \cdot V) \cdot (U \cdot S \cdot V)^T = U \cdot S \cdot \overbrace{V \cdot V^T}^{= I_{r\times r}} \cdot \underset{}{\overset{S}{\overset{\shortparallel}{S^T}}} \cdot U^T = U \cdot S^2 \cdot U^T$$

$$A^T \cdot A = (U \cdot S \cdot V)^T \cdot (U \cdot S \cdot V) = V^T \cdot \underset{S}{\underset{\shortparallel}{S^T}} \cdot \underbrace{U^T \cdot U}_{= I_{r\times r}} \cdot S \cdot V = V^T \cdot S^2 \cdot V$$

Using LSI for better Retrieval 1/8

- **Variant 1:** work with A_k instead of A

	D_1	D_2	D_3	D_4	D_5	D_6
internet	1	1	0	1	0	0
web	1	0	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Our example A from the beginning

D'_1	D'_2	D'_3	D'_4	D'_5	D'_6
0.9	0.6	0.6	1.0	0.0	0.0
0.9	0.6	0.6	1.0	0.0	0.0
1.1	0.9	0.9	2.1	1.0	1.0
-0.1	0.1	0.1	0.9	1.0	1.0

best rank-2 approximation A_2

Using LSI for better Retrieval 2/8

- **Variant 1:** work with $\mathbf{A_k}$ instead of **A**
  - Problem: A_k is a dense matrix, that is, most / all of it's $m \cdot n$ entries will be non-zero

    Typically, both m and n will be very large, and then already storing this matrix is infeasible

    Example: $m = 1000$ and $n = 10M$ → $m \cdot n =$ **10 G**

Using LSI for better Retrieval 3/8

- **Variant 2:** work with $\mathbf{V_k}$ instead of with **A**
  - Recall: V_k gives the representation of the documents in the k-dimensional concept space

	D_1	D_2	D_3	D_4	D_5	D_6
internet	1	1	0	1	0	0
web	1	0	1	1	0	0
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Our example A from the beginning

D'_1	D'_2	D'_3	D'_4	D'_5	D'_6
0.4	0.3	0.3	0.7	0.3	0.3
0.5	0.2	0.2	0.0	-0.6	-0.6

V_2 from the SVD of A

Using LSI for better Retrieval 4/8

- **Variant 2:** work with $\mathbf{V_k}$ instead of with **A**
  - Observation: V_k is a dense matrix, that is, most or all of its $k \cdot n$ entries are non-zero
  - Note: the original matrix A has $m' \cdot n$ non-zero entries, where m' is the average number of words in a document
  - So storing V_k instead of A is ok if $k \approx m'$ or smaller

    No need for a sparse representation (inverted index)
  - This is a good variant to use in practice

Using LSI for better Retrieval 5/8

- **Variant 2:** work with $\mathbf{V_k}$ instead of with **A**
  - Problem 2: we need to map the query to concept space

    The dot-product similarity of query q with all documents is

    $$q^T \cdot A_k = q^T \cdot (U_k \cdot S_k \cdot V_k) = \underbrace{(\overset{1\times m}{q^T} \cdot \overset{m\times k}{U_k} \cdot \overset{k\times k}{S_k})}_{1\times k} \cdot \underbrace{V_k}_{k\times n}$$

    $q_k^T := \underbrace{q^T \cdot U_k \cdot S_k}_{1\times k}$ is the query mapped to "concept space"
  - The dot product $q_k^T \cdot V_k$ requires time **~ n · k** … since both q_k and V_k are dense
  - In comparison: computing the similarities of q with the original documents requires time **O(n · #query words)**

Using LSI for better Retrieval 6/8

- **Variant 3:** expand the original documents
  - In Variant 2, we have transformed both the query and the documents to concept space
  - LSI can also be viewed as doing **document expansion** in the original space + no change in the query
  - Namely, let $T_k = \underbrace{U_k}_{m \times k} \cdot \underbrace{U_k^T}_{k \times m}$ this is an m x m matrix
  - Then one can easily prove that $\underbrace{A_k}_{m \times n} = \underbrace{T_k}_{m \times m} \cdot \underbrace{A}_{m \times n}$

do it as an (optional) exercise !

Using LSI for better Retrieval 7/8

- Variant 3: expand the original documents
  - Here is some intuition for $\mathbf{T_k}$, assuming **0** or **1** entries

    In practice, we can get 0-1 entries by setting all entries in T above a certain threshold to 1, and all others to 0

this 1 says: if there is "internet" in D_i then add "web" to D_i'

	internet	web	surfing	beach			D_i			D'_i	
internet	1	1	0	0	1		1	1		1	1
web	1	1	0	0	2	•	0	2	=	1	2
surfing	0	0	1	0	3		1	3		1	3
beach	0	0	0	1	4		0	4		0	4

Using LSI for better Retrieval 8/8

- Linear combination with original scores
  - Experience: LSI adds some useful information to the term-document matrix, but also a lot of **noise**
  - In practice, one therefore uses a linear combination of the original scores and the LSI scores:

    Variant 1: $\text{scores} = \lambda \cdot q^T \cdot A + (1 - \lambda) \cdot q^T \cdot A_k$

    Variant 2: $\text{scores} = \lambda \cdot q^T \cdot A + (1 - \lambda) \cdot q_k{}^T \cdot V_k$

    Variant 3: $\text{scores} = \lambda \cdot q^T \cdot A + (1 - \lambda) \cdot q^T \cdot T_k \cdot A$

References

- Further reading
  - Textbook Chapter 18: Matrix decompositions & LSI

    http://nlp.stanford.edu/IR-book/pdf/18lsi.pdf
  - Deerwester, Dumais, Landauer, Furnas, Harshman

    Indexing by Latent Semantic Analysis, JASIS 41(6), 1990
- Web resources
  - http://en.wikipedia.org/wiki/Latent_semantic_indexing
  - http://en.wikipedia.org/wiki/Singular_value_decomposition
  - http://www.numpy.org/
  - http://www.scipy.org/