Information Retrieval

WS 2015 / 2016

Lecture 8, Wednesday December 8th, 2015
(Vector Space Model, 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 ES7 — cookies, UTF-8
- Contents
  - Synonyms — motivation + examples
  - Vector Space Model (VSM) — documents as vectors
  - Latent Semantic Indexing (LSI) — find synonyms automagically
  - Using LSI for retrieval — three variants + another
  - Exercise Sheet 8: re-implement your code from ES2 using the VSM, and re-evaluate benchmark from ES2 using LSI

Experiences with ES7

- Summary / excerpts
  - "This exercise sheet was annoying"

    Sorry … but a perfect summary of the typical developer experience with encoding issues, in particular UTF-8
  - "Very useful … I will not struggle anymore with encodings"

    Thanks, that was exactly the intention of the lecture !
  - Quite some bit fiddling needed

    Some were not up to the low-level details and defaulted to the built-in functions

Synonyms 1/4

- Motivation
  - We have already seen fuzzy (prefix) search

    Search uniwercity find university

  - Today we want to find synonyms = others word meaning the same thing as a given word

    Search university find college

    Search bringdienst find lieferservice

    Search cookie find biscuit

    Note: typically no **lexical** similarity whatsoever, the similarity is only in the **meaning**

Synonyms 2/4

- Solution 1: Maintain a thesaurus
  - For each word, manually compile a list of synonyms

    university: uni, academy, college, ...

    bringdienst: lieferservice, heimservice, pizzaservice, ...

    cookie: biscuit, confection, wafer, ...

  - Problem 1: laborious, and still notoriously out of date
  - Problem 2: it depends on the context, which synonyms are appropriate … for example:

    university award ≠ academy award

    http cookie ≠ http biscuit

Synonyms 3/4

- Solution 2: Track user behavior
  - Investigate whole **search sessions**

    Track sessions with, guess what: COOKIES
  - For example, many users searching for either of

    pizza freiburg

    bringdient freiburg

    then click on

    Lieferservice Freiburg im Breisgau

    This provides a hint that pizza and bringdienst and lieferservice are related

Synonyms 4/4

- Solution 3: Automatic methods
  - The text itself also tells us which words are related
  - For example, consider (German) **delivery webpages**

    some mention Bringdienst, others say Lieferservice

    but apart from that they use the same words a lot, like:
    pizza, mozzarella, käse, nudeln, vegetarisch, …

    Can we find out (automatically) that two words are related, based on the similar context they appear in ?

    This is the topic of today's lecture !

Vector Space Model 1/8

- Motivation
  - For this lecture, it will be useful to represent documents as **vectors** … here is our running example for today:

	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

  - Each row corresponds to a word, each column to a document
  - Non-zero entries: score for that word in that document

    In the lecture, we use tf scores … for ES8, use BM25 scores

Vector Space Model 2/8

- Terminology
  - Often referred to as the **Vector Space Model (VSM)**
  - In the VSM, words are traditionally referred to as **terms**
  - Putting the vectors from all documents from a given corpus side by side gives us the so-called **term-document matrix**

	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

Vector Space Model 3/8

- Retrieval
  - A query can also be represented as a vector … we take 1 for a term used in the query, and 0 for all other terms
  - We measure the relevance of a document to the query by taking the **dot product** of the two vectors

  Note: this is exactly the same score as in Lecture 2

	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	

Vector Space Model 4/8

- Algebra
  - More formally, let us write A for the term-document matrix and q for the query vector
  - Then the matrix-vector product $q^T \cdot A$ gives us a vector with the relevance scores of all the documents

  Let us implement this together now

	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

Vector Space Model 5/8

- Basic linear algebra in Python
  - For standard linear algebra, we can use **numpy**

    sudo apt-get install python3-numpy

```
import numpy
A = numpy.array([[1, 1, 0, 1, 0, 0], …])
q = numpy.array([0, 1, 1, 0])
scores = q.dot(A)
print(scores)
```

    Use **numpy.array** and **dot** for multiplication, not *

    q is a row vector above = q^T from the previous slide

    See the code from the lecture for more example usage

Vector Space Model 6/8

- Sparse matrices
  - Most entries in a term-document matrix are **zero**

    Storing all entries explicitly infeasible for large matrices
  - Sparse-matrix representation: store only the non-zero entries (together with their row and column index)

(1, 0, 0), (1, 0, 1), (1, 0, 3), …, (2, 2, 3), …

	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

Vector Space Model 7/8

- Sparse matrices
  - Two principle ways to store the list of non-zero values

    row-major: store row by row (sort by row index first)

    column-major: store col by col (sort by col index first)

  - Note: the sparse row-major representation of a term-document matrix is equivalent to an inverted index

(1, 0, 0), (1, 0, 1), (1, 0, 3)	inverted list for term 0
(1, 1, 0), (1, 1, 2), (1, 1, 3)	inverted list for term 1
(1, 2, 0), (1, 2, 1), (1, 2, 1), …	inverted list for term 2
(1, 3, 3), (1, 3, 4), (1, 3, 5)	inverted list for term 3

(non-zero score, row index = term id, col index = doc id)

Vector Space Model 8/8

CSR = compressed sparse row

UNI FREIBURG

- Sparse matrices in Python
  - Not included in numpy, we have to use **scipy**

```
sudo apt-get install python3-scipy
```

```
import scipy.sparse
nz_vals    = [1, 1, 1, 1, 1, 1, ...]
row_inds = [0, 0, 0, 1, 1, 1, ...]
col_inds   = [0, 1, 3, 0, 2, 3, ...]
A = scipy.sparse.csr_matrix((nz_vals, (row_inds, col_inds)))
q = scipy.sparse.csr_matrix([0, 1, 1, 0])
scores = q.dot(A)
print(scores)
```

See the code from the lecture for more example usage

Latent Semantic Indexing 1/9

- Motivation
  - Let's look at our example 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 2/9

- Motivation
  - Let's look at the query web surfing again, using dot-product similarity as explained on slide 10
  - 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	X	X	
	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 3/9

- Conceptual solution

	REL	REL	REL	REL	×	×		
	D_1	D_2	D_3	D_4	D_5	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: $sim(D_1, Q) = sim(D_2, Q) = sim(D_3, Q)$

The goal of LSI is to do something like this automatically

Latent Semantic Indexing 4/9

- 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 5/9

- Matrix factorization

	D1	D2	D3	D4	D5	D6
	1	1	1	1	0	0
	1	1	1	1	0	0
	1	1	1	2	1	1
	0	0	0	1	1	1

=

B1	B2
1	0
1	0
1	1
0	1

•

D'1	D'2	D'3	D'4	D'5	D'6
1	1	1	1	0	0
0	0	0	1	1	1

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 6/9

- 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: $A' = \text{argmin}_{A' \, m \times n \text{ with rank } k} \| A - A' \|$

    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 7/9

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

    Theorem: for any m x n matrix A of rank r, there exist U, S, V such that **A = U · S · V** , and where

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

    S is a r x r matrix with entries only on its diagonal

    V is an r x n matrix with $V^T \cdot V = I_n$ the n x n identify matrix

    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 8/9

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

    U_k = the first k columns of U now an m x k matrix

    S_k = the upper k x k part of S now a k x k matrix

    V_k = the first k rows of V now a k x n matrix

    Note: then still $U_k \cdot U_k^T = I_m$ and $V_k \cdot V_k^T = I_n$

    Let $\mathbf{A_k = U_k \cdot S_k \cdot V_k^T}$

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

Latent Semantic Indexing 9/9

- How to compute the SVD
  - Easy to compute from the **Eigenvector decomposition**

    Namely of the quadratic matrices $A \cdot A^T$ and $A^T \cdot A$
  - In practice, the more direct Lanczos method is used

    This has complexity $O(k \cdot nnz)$, where k is the rank and nnz is the number of non-zero values in the matrix

    Note that for term-document matrices $nnz << n \cdot m$

    For ES8, just use **svds** from **scipy.sparse.linalg**

    See the code from the lecture for a usage example

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

    E.g. if $m = 1000$ 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

    Note: no need for a sparse representation / an inverted index when storing / using V_k

    This is the variant you should use for ES8.3

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

    $q^T \cdot A_k = q^T \cdot (U_k \cdot S_k \cdot V_k) = (q^T \cdot U_k \cdot S_k) \cdot V_k$

    Then $q_k^T := q^T \cdot U_k \cdot S_k$ is query mapped to concept space
  - The dot product $q_k^T \cdot V_k$ requires time $\sim n \cdot k$ … since both q_k and V_k are dense

    In comparison: computing the similarities of q with the original documents requires time $O(n \cdot \#q)$ and less

    where $\#q$ = number of query words in q

Using LSI for better Retrieval 6/8

$U = (| \; | \; | \; \cdots \; |)$ column-orthogonal

- **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 $T_k = U_k \cdot U_k^T$ this is an m x m matrix

    Then one can easily prove that $A_k = T_k \cdot A$

$r = \text{rank}(A)$

$$\underset{m \times m}{T_k} \cdot \underset{m \times n}{A} = \underset{m \times k}{U_k} \cdot \underbrace{U_k^T \cdot U}_{= \underset{k \times r}{[I_k \; 0]}} \cdot \underset{r \times r}{S} \cdot \underset{r \times n}{V} = U_k \cdot \underbrace{([I_k \; 0] \cdot S \cdot V)}_{= S_k \cdot V_k} = A_k$$

For ES8, simply compute T_k from U_k as shown, then compute the 50 term pairs with the largest entries in T_k

Using LSI for better Retrieval 7/8

- Variant 3: expand the original documents
  - Here is some intuition for 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

says that "web" and "internet" are related

"adds" web to D_i if internet is present

T

	internet	web	surfing	beach
internet	1	1	0	0
web	1	1	0	0
surfing	0	0	1	0
beach	0	0	0	1

•

D_i
1
0
1
0

=

D'_i
1
1
1
0

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

  For ES9, take Variant 2 and experiment with a good λ

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/