Information Retrieval

WS 2013 / 2014

Lecture 8, Wednesday December 10th, 2013
(Synonyms, 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 ES#7 (cookies, UTF-8)
- Synonyms
  - Yet another form of fuzzy search, but this time with no syntactic similarity whatsoever:

    search pizza, find lieferservice
  - We will look at a fancy, fully automatic approach:

    Latent Semantic Indexing (LSI)
  - Exercise Sheet 8: use LSI to compute pairs of most related terms from our example collection from ES1

    You will learn a new tool for that today: **Octave**

Experiences with ES#7 (Cookies, UTF-8)

- Summary / excerpts   last checked December 10, 15:00
  - Nice course / topic, despite aversion against UTF-8
  - Encoding stuff still confusing, but get's better bit by bit :-)
  - Java is too intelligent for this sheet … well, and slow
  - First 100 lines of code, then 10 lines of code … always ask!
  - "Maybe I should watch the recording" … maybe, yes
  - Does Prof. Bast use vim also for "real" coding … YES
  - JavaScript / web server etc. interesting, but not really IR

    I respectfully disagree, web apps are at the core of IR
  - First C++ experiences → segmentation fault
  - Some of you don't read the feedback you get, please do!

Synonyms 1/4

- Motivation
  - We have already seen **wildcard search**

    Search uni* ... find university
  - And we have seen **error-tolerant search**

    Search uniwercity ... find university
  - Today we want to look at **synonym search**

    Synonym = another word meaning the same thing

    Search university ... find college

    Search bringdienst ... find lieferservice

    Search cookie ... find biscuit

    Note: typically no **syntactic** similarity whatsoever

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, ...
  - Two major problems with this approach:

    1. This is laborious, and still notoriously out of date

    2. Depends on context, which synonyms are appropriate

    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

    bringdienst 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 **pizza delivery webpages**

    They have similar contents (and style)

    Some use the word Bringdienst, others use Lieferservice

    Can we find out that these two words are related, based on the similar context they appear in ?
  - **Latent Semantic Indexing** (LSI) tries to do exactly that, and it does it fully unsupervised / automatically

    This is the topic of today's lecture !

Latent Semantic Indexing 1/9

- Our running example for this lecture

	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

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

D_5 and D_6 are "about" surfing on the beach

The words internet and web are **synonyms** here

The word surfing is **polysemous** here = it means different things in different context

Latent Semantic Indexing 2/9

- Problems with standard retrieval

	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		

Consider the query web surfing on that matrix

Let us use dot-product similarity, as in Lecture 2

Then e.g. $\text{sim}(D_3, Q) > \text{sim}(D_2, Q) = \text{sim}(D_5, Q)$

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

Latent Semantic Indexing 3/9

- Conceptual solution

	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: $\mathrm{sim}(D_1, Q) = \mathrm{sim}(D_2, Q) = \mathrm{sim}(D_3, Q)$

The goal of LSI is to do something like this automatically

Latent Semantic Indexing 4/9

- A simple but powerful observation

	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

4 x 6

=

B1	B2
1	0
1	0
1	1
0	1

4 x 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

2 x 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 $D'_1, \ldots, D'_6$ 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
  - Given an integer $k < \text{rank}(A)$
  - 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 $\| \dots \|$ 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 $\mathbf{A = U \cdot S \cdot V^T}$, and where

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

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

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

    Note: we can always choose S such that the diagonal entries are positive and sorted (largest entry at 1, 1)

    Then the decomposition is unique

Latent Semantic Indexing 8/9

- Using the SVD our task becomes easy
  - Let $A = U \cdot S \cdot V^T$ be the SVD of A
  - For a given $k < \text{rank}(A)$ 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 columns of V — now an n x k matrix

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

    Let $\mathbf{A' = U_k \cdot S_k \cdot V_k^T}$

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

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 $A^T \cdot T$
  - 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 built-in **svds** from Octave, see slide 27

Using LSI for better Retrieval 1/7

- **Variant 1:** work with **A'** 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	0.9	1.1	-0.1	0.6	0.6
0.9	0.1	0.6	0.6	0.9	0.1
1.0	1.0	2.1	0.9	0.0	0.0
1.0	1.0	0.0	0.0	1.0	1.0

best rank-2 approximation A'

Using LSI for better Retrieval 2/7

- **Variant 1:** work with **A'** instead of A
  - Problem: A' 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

    For ES8, $m = 1000$ and $n = 8.2M$ → $m \cdot n =$ **8.2B**

Using LSI for better Retrieval 3/7

- **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	-0.5	0.3	-0.2	0.3	-0.2
0.7	0.0	0.3	0.6	0.3	0.6

V_2^T from the SVD of A

Using LSI for better Retrieval 4/7

- **Variant 2:** work with $\mathbf{V_k}$ instead of with A
  - Problem 1: V_k is also 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 $k \leq m'$ or $k \approx m'$

    Note: no need for an inverted index then

Using LSI for better Retrieval 5/7

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

    Let q be the query … note: $|q|$ = #keywords typ. very small

    Similarity (dot-product) of q with all documents is

    $q^T \cdot A' = (q^T \cdot U_k \cdot S_k \cdot V_k^T) = (S_k \cdot U_k \cdot q)^T \cdot V_k^T =: \mathbf{q'}^T \cdot V_k^T$

    This $q' = S_k \cdot U_k \cdot q$ is the query mapped to concept space

    Then we need to compute dot-product with all docs in V_k^T

    Since q' and V_k^T are dense, this requires time $\sim n \cdot k$

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

Using LSI for better Retrieval 6/7

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

    Then, $A' = T \cdot A$

PROOF : ... maybe in the end ...

Using LSI for better Retrieval   7/7

- Variant 3: expand the original documents
  - Here is some intuition for T, assuming 0 or 1 entries

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

    For ES8, output the 100 term pairs with the largest values

effect of this entry: if „internet" in D_i: add „web" to D_i

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

T $m \times m$

$m = \#$ terms

Octave 1/6

- A script language for numerical computation
  - GNU's open source version of the proprietary **Matlab**
  - Makes numerical computations easy, which would otherwise be a pain to use in Java / C++

    In particular: computations with matrices and vectors
  - Also comes with an interactive shell, see next slide
  - Language has C-like commands (printf, fopen, ...)
  - Still it's a **script language**, and correspondingly slow
  - The built-in functions (like svds) are quite fast though
  - Download and Doc.: http://www.gnu.org/software/octave

Octave 2/6

- The Octave interactive shell + help
  - Use pretty much like a Bash shell, in particular:

    Arrow ↑ : previous command

    Arrow ↓ : next command

    CTRL+R : search in history

    CTRL+A : go to beginning of line

    CTRL+E : go to end of line

    CTRL+K : delete from cursor position to end of line
  - Interactive help with help <function name>
  - Google for Matlab, not Octave, the basic stuff is identical

    matlab read sparse matrix

Octave 3/6

- File handling
  - Open a file with fopen just like in C, e.g.

```
input_file = fopen("input.txt", "r");
output_file = fopen("output.txt", "w");
```

  - Read from text file with load or textscan, e.g.

```
tmp1 = load("stupid.matrix");
tmp2 = textscan(input_file, "%s");
```

  - Write text file with fdisp, e.g.

```
fdisp(output_file, "stupid result");
```

Octave 4/6

- Sparse matrices
  - Use spconvert to convert from explicit sparse-matrix format (Ex. 8.1) to the internal sparse-matrix format

```
tmp = load("stupid.matrix"));
A = spconvert(tmp);
clear tmp;
```

  - Compute the k-truncated SVD for a sparse matrix:

```
[U, S, V] = svds(A, k);
```

    Note: the running time of this is proportional to k

  - In comparison, for the full SVD for a dense matrix:

```
[U, S, V] = svd(A);
```

Octave 5/6

- Some more useful commands
  - Manually create the matrix from our running example

    ```
    A = [1 1 0 1 0 0; 1 0 1 1 0 0; 1 1 1 2 1 1; 0 0 0 1 1 1];
    ```

    Note: if you omit the semicolon in the end or write a comma, the result will be printed on the screen
  - Flatten a matrix to a vector and then sort it:

    ```
    V = reshape(A, size(A)(1) * size(A)(2));
    Vs = sort(V, 'descend');
    ```
  - Get indices + value of all entries with a certain property:

    ```
    [I, J, V] = find(A == 1);
    ```

Octave 6/6

- And yet more useful commands
  - Get a portion of a matrix or vector

    ```
    Uk = U(:, 1:k);   // First k columns of U.
    ```

    Note: matrix / vector indices in Octave start at **1**, not 0
  - Multiply a matrix with its transpose

    ```
    T = Uk * Uk';
    ```

    Note: for the transpose use ' and not ^T or sth like that

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
- Wikipedia
  - http://en.wikipedia.org/wiki/Latent_semantic_indexing
  - http://en.wikipedia.org/wiki/Singular_value_decomposition
  - http://www.gnu.org/software/octave
  - http://en.wikipedia.org/wiki/GNU_Octave