Overview of this lecture

- **Organizational**
  - Your experiences with ES9  
  - Clustering

- **Contents**
  - Latent Semantic Indexing (LSI)  
    - more linear algebra magic
  - Using LSI for retrieval  
    - three variants
  - ES10: modify your VSM code from ES8 to use LSI, evaluate the benchmark again, and use LSI to find related terms
Experiences with ES9 1/5

Summary / excerpts

- Interesting + very nice linear algebra
- Several of you skipped this sheet due to holidays
- For some it was fast, others took more time than expected
  
  "We expected this sheet to be very easy and fast ... however, debugging was very long and tedious"

  "This was the easiest sheet I ever failed to solve"

  "I'll start looking for the next Zen monastery to join"

- Seeing the centroid words was the best part!
- "Transpose here and there until the doctest ran through"
- "Redundancy in TIP file and Exercise Sheet is not good"
Results

- Many clusters on coherent "topics":
  
  thriller horror fiction crime action dead sci fi ... 
  kong hong wong chan lau cheung martial ... 

- Some clusters of a different nature 
  
  to, her, his, he, they, their, she, that, are, who ... 
  as, was, in, series, first, of, the, to, s, films, ... 

- Some clusters are mixes of "topics" and general words 
  
  short animated 2012 2011 2010 written 2007 drama ... 
  comedy fatty romance italian written harold lloyd ...
Reincarnation, excerpts from your thoughts

- Basic principle of reincarnation: Some soul/software is transferred from old to new body/hardware.
- If a "soul" exists, it's not: solid, liquid, gas (macroscopic), nor electric, magnetic, light, radiation (microscopic)
- Some say, they remember foreign countries ... But with LSD people see purple elephants → don't trust your brain
- Why is soul memoryless? (people who were reincarnated do not recall their previous life)
- Reincarnation seems another excuse to not deal with the difficulties of the current life (e.g. "caste system")
Experiences with ES9  4/5

- Reincarnation, naïve view
  - There is some invisible "spirit" or "ghost" inside of us
  - Which lives on when the body dies
  - And then somehow finds its way into the next body
  - And somehow forgets everything about the previous life
  - Maybe that is so ... maybe ... but probably not

However, some of you correctly pointed out that since we know nothing about the stuff that consciousness is made of, we cannot really know anything about this
Reincarnation, "interface" view

- A common phenomenon in our universe: very complex internal state ↔ very compact external manifestation
  
  Example: personality ↔ a piece of communication

  Example: software executable ↔ the algorithm behind

- Living beings are very good at (reverse) engineering the complex inner state from the compact representation

- With death, your complex internal state dies, but some external manifestations easily live on (e.g. a book written)

- A new "blank" organism with a suitable configuration can reverse-engineer the internal state (not exactly, of course)

- In a very concrete sense, something has "reincarnated" then
Latent Semantic Indexing  1/10

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

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
<th>$D_5$</th>
<th>$D_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>internet</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>web</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>surfing</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>beach</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Latent Semantic Indexing  

Motivation

- Let's look at the query **web surfing** again, using dot-product similarity as explained in L8

- Then \( \text{sim}(D_3, Q) > \text{sim}(D_2, Q) = \text{sim}(D_5, Q) \)

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

<table>
<thead>
<tr>
<th></th>
<th>( D_1 )</th>
<th>( D_2 )</th>
<th>( D_3 )</th>
<th>( D_4 )</th>
<th>( D_5 )</th>
<th>( D_6 )</th>
<th>( Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>internet</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>web</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>surfing</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>beach</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

2. 1 -> 2 3 1 1  
relevant and low score
Latent Semantic Indexing  3/10

- **Conceptual solution**

  \[
  \begin{array}{cccccc}
  \text{REL} & \text{REL} & \text{REL} & \text{REL} & \text{NOT} & \text{NOT} \\
  \hline
  D_1 & D_2 & D_3 & D_4 & D_5 & D_6 & Q \\
  \hline
  \text{internet} & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
  \text{web} & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
  \text{surfing} & 1 & 1 & 1 & 2 & 1 & 1 & 1 \\
  \text{beach} & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
  \end{array}
  \]

  \[
  2 \quad 2 \quad 2 \quad 3 \quad 1 \quad 1
  \]

  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**
A simple but powerful observation

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**
Matrix factorization

Equivalently: the $4 \times 6$ term-document matrix can be written as a product of a $4 \times 2$ matrix with a $2 \times 6$ 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".
The goal of LSI

- Given an m x n term-document matrix A and k < 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' = \arg\min_{A', m \times n} \text{with rank } k \| A - A' \|

For the $\| ... \|$ we take the so-called **Frobenius-norm**

This is defined as $\| D \| := \sqrt{\sum_{ij} D_{ij}^2}$

The reason for using this norm is purely technical: that way, the math on the next slides works out nicely
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 $m \times n$ matrix $A$ of rank $r$, there exist $U$, $S$, $V$ such that $A = U \cdot S \cdot V$, and where

- $U$ is an $m \times r$ matrix with $U^T \cdot U = I_r$ the $r \times r$ identity matrix
- $S$ is an $r \times r$ matrix with non-zero entries only on its diag.
- $V$ is an $r \times n$ 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 8/10

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

    \[
    U_k = \text{the first } k \text{ columns of } U \quad \text{now an } m \times k \text{ matrix}
    \]

    \[
    S_k = \text{the upper } k \times k \text{ part of } S \quad \text{now a } k \times k \text{ matrix}
    \]

    \[
    V_k = \text{the first } k \text{ rows of } V \quad \text{now a } k \times n \text{ matrix}
    \]

    Note: then \( U_k^T \cdot U_k = I_k \) and \( V_k \cdot V_k^T = I_k \)

  - Let \( A_k = U_k \cdot S_k \cdot V_k \)

    Then \( A_k \) is a matrix of rank \( k \) that minimizes \( \| A - A_k \| \)
How to compute the SVD

- Can be computed from the **Eigenvector decomposition**
  
  See next slide for some of the mathematics behind

- In practice, the more direct **Lanczos** method is used

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

  Note that for term-document matrices \text{nnz} $\ll n \cdot m$

For ES10, just use **svds** from **scipy.sparse.linalg**
Some of the mathematics behind the SVD

- 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 $u_i \cdot u_j = 0$, for $i \neq j$, and $|u_i| = 1$.

Equivalently, $B = U \cdot \text{diag}(\lambda_1, \ldots, \lambda_n) \cdot U^T$ ... and $U \cdot U^T = I$

- The matrices $A \cdot A^T$ and $A^T \cdot A$ are symmetric, hence there exist orthogonal $U$ and $V$ and diagonals $S_1$ and $S_2$ such that $A \cdot A^T = U \cdot S_1 \cdot U^T$ and $A^T \cdot A = V \cdot S_2 \cdot V^T$

- Let us assume that a decomposition $A = U \cdot S \cdot V$ exists:

$$A \cdot A^T = (U \cdot S \cdot V) \cdot (U \cdot S \cdot V)^T = U \cdot S \cdot V \cdot V^T \cdot S^T \cdot U^T = U \cdot S \cdot U^T$$

$$A^T \cdot A = \ldots = V^T \cdot S^2 \cdot V$$

$$\Rightarrow \quad S_1 = S_2$$
**Variant 1:** work with $A_k$ instead of $A$

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
<th>$D_5$</th>
<th>$D_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>internet</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>web</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>surfing</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>beach</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$D_1'$</th>
<th>$D_2'$</th>
<th>$D_3'$</th>
<th>$D_4'$</th>
<th>$D_5'$</th>
<th>$D_6'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>internet</td>
<td>0.9</td>
<td>0.6</td>
<td>0.6</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>web</td>
<td>0.9</td>
<td>0.6</td>
<td>0.6</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>surfing</td>
<td>1.1</td>
<td>0.9</td>
<td>0.9</td>
<td>2.1</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>beach</td>
<td>-0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.9</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Our example $A$ from the beginning

best rank-2 approximation $A_2$
Variant 1: work with $A_k$ 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 $n = 10M \rightarrow m \cdot n = 10 \text{ G}$
Using LSI for better Retrieval  3/8

**Variant 2:** work with $V_k$ instead of with $A$

- Recall: $V_k$ gives the representation of the documents in the $k$-dimensional concept space

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
<th>$D_5$</th>
<th>$D_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>internet</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>web</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>surfing</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>beach</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$D'_1$</th>
<th>$D'_2$</th>
<th>$D'_3$</th>
<th>$D'_4$</th>
<th>$D'_5$</th>
<th>$D'_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
<td>0.7</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.2</td>
<td>0.2</td>
<td>0.0</td>
<td>-0.6</td>
<td>-0.6</td>
</tr>
</tbody>
</table>

Our example $A$ from the beginning  $V_2$ from the SVD of $A$
Variant 2: work with $V_k$ 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 ES10.4
**Variant 2:** work with $V_k$ 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 $q$
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 $m \times m$ matrix

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

For ES10, simply compute $T_k$ from $U_k$ as shown, then compute the 100 term pairs with the largest entries in $T_k$
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.
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 ES10, take Variant 2 and experiment with a good \( \lambda \).
References

- **Further reading**
  - Textbook Chapter 18: Matrix decompositions & LSI
  - Deerwester, Dumais, Landauer, Furnas, Harshman
    Indexing by Latent Semantic Analysis, JASIS 41(6), 1990

- **Web resources**
  - http://www.numpy.org/
  - http://www.scipy.org/