Latent Semantic Indexing

Fabian Drach & James Min

23.01.2020

Agenda

- Overview
- Goal and mathematical implementation
- Advantages / Disadvantages
- Uses of LSI Systems
- Sources

Overview

- **L**atent **S**emantic **I**ndexing (also Latent Semantic Analysis) is a method in Information Retrieval
- LSI is based on assumption there is underlying latent semantic structure in data which is corrupted by variety of words used
- This semantic structure can be discovered and enhanced by projecting the data onto a lower-dimensional space (using SVD)

Example

	D1	D2	D3	D4	D5	D6
internet	1	1	0	1	0	0
web	1	0	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Information Retrieval WS17/18, Lecture 10, University of Freiburg, Germany

D1, D2 and D3 are about surfing the web

D5 and D6 are about surfing at the beach

internet & web are synonyms, surfing is a polysem (different meaning in different context)

Example

	D1	D2	D3	D4	D5	D6
internet	1	1	0	1	0	0
web	1	0	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Information Retrieval WS17/18, Lecture 10, University of Freiburg, Germany

Query: web surfing

Q
0
1
1
0

Example

	D1	D2	D3	D4	D5	D6
internet	1	1	0	1	0	0
web	1	0	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1
DPS	2	1	2	3	1	1

Q
0
1
1
0

Information Retrieval WS17/18, Lecture 10, University of Freiburg, Germany

Query: web surfing

computating similarity using dot product similarity

Example

	D1	D2	D3	D4	D5	D6
internet	1	1	0	1	0	0
web	1	0	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1
DPS	2	1	2	3	1	1

Q
0
1
1
0

Conceptual solution: adding missing synonyms to the documents (web == internet)

so that sim(D1,Q) = sim(D2,Q) = sim(D3,Q)

Example

	D1	D2	D3	D4	D5	D6
internet	1	1	**1**	1	0	0
web	1	**1**	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1
DPS	2	**2**	2	3	1	1

Q
0
1
1
0

Information Retrieval WS17/18, Lecture 10, University of Freiburg, Germany

Conceptual solution: adding missing synonyms to the documents (web == internet)

so that sim(D1,Q) = sim(D2,Q) = sim(D3,Q)

Goal: to do something like this automatically

Example

	D1	D2	D3	D4	D5	D6
internet	1	1	**1**	1	0	0
web	1	**1**	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1
DPS	2	**2**	2	3	1	1

Matrix can be shortened to two base columns

base vectors b1 & b2 are underlying **concepts**

b1	b2
1	0
1	0
1	1
0	1

- D1 = b1
- D2 = b1
- D3 = b1
- D4 = b1 + b2
- D5 = b2
- D6 = b2

Example

	D1	D2	D3	D4	D5	D6
internet	1	1	**1**	1	0	0
web	1	**1**	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1
DPS	2	**2**	2	3	1	1

Also:

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

b1	b2
1	0
1	0
1	1
0	1

x

vectors D'1, … , D'6 are representation in the concept space

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

Goal of LSI

Given a m x n term-document matrix **A** and
k < rank(A)
then find a matrix **A'** of column rank k such that the diffenrence between A' and A is as small as possible

Computation

For any m x n matrix A of rank r there exists U, S, V such that A = U * S * V

U is a m x r matrix with $U * U^T = I_r$, with I_r is the r x r identity matrix

S is an r x r matrix with non-zero entries only on its diagonal

V is an r x n matrix with $V * V^T = I_r$

U^T: columns of U are normalized to 1 and orthogonal to each other

V^T: same for rows

Computation

U is a m x r matrix with $U * U^T = I_r$, with I_r is the r x r identity matrix

S is an r x r matrix with non-zero entries only on its diagonal

V is an r x n matrix with $V * V^T = I_r$

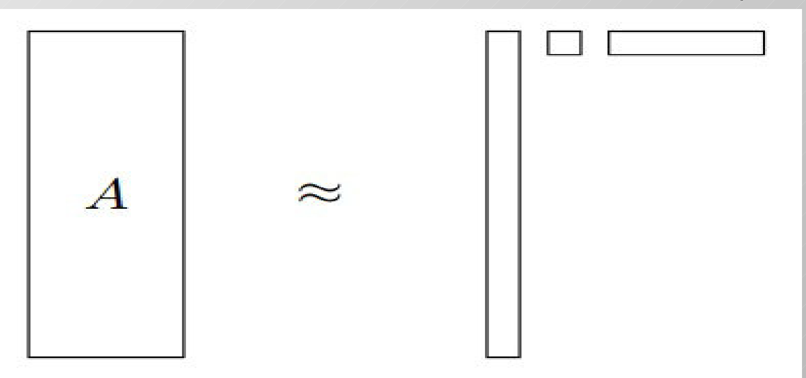
- The decomposition is unique up to simultaneous permutations of the rows / columns of U, S and V
- Standard form: diagonal entries of S are positiv and sorted

Computation

U is a m x r matrix with $U * U^T = I_r$, with I_r is the r x r identity matrix

S is an r x r matrix with non-zero entries only on its diagonal

V is an r x n matrix with $V * V^T = I_r$



Lars Eldén, Matrix Methods in Data Mining and Pattern Recognition, S. 135

Computation

Using SVD this task becomes easier

Let A = U * S * V be the SVD of A

for a given k < rank(A) let

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

S_k = the upper k * 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: U_k is column-orthonormal just like U

Computation

Let $A_k = U_k * S_k * V_k$

then A_k is a matrix of rank k that minimizes

$|| A - A_k ||$

Disadvantages

Working with A_k instead of A

	D1	D2	D3	D4	D5	D6
internet	1	1	0	1	0	0
web	1	0	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

Information Retrieval WS17/18, Lecture 10, University of Freiburg, Germany

	D1	D2	D3	D4	D5	D6
internet	0.9	0.6	0.6	1.0	0.0	0.0
web	0.9	0.6	0.6	1.0	0.0	0.0
surfing	1.1	0.9	0.9	2.1	1.0	1.0
beach	-0.1	0.1	0.1	0.9	1.0	1.0

Disadvantages

Working with A_k instead of A:

Problem: A_k is a dense matrix

Typically, both m and n will be very large

Solution: working with V_k instead of A (concept-based matrix)

	D1	D2	D3	D4	D5	D6
internet	1	1	0	1	0	0
web	1	0	1	1	0	1
surfing	1	1	1	2	1	1
beach	0	0	0	1	1	1

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

Solution: working with V_k instead of A

V_k is a dense matrix, but much smaller then A_k

Problem: query needs to be mapped to concept space

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

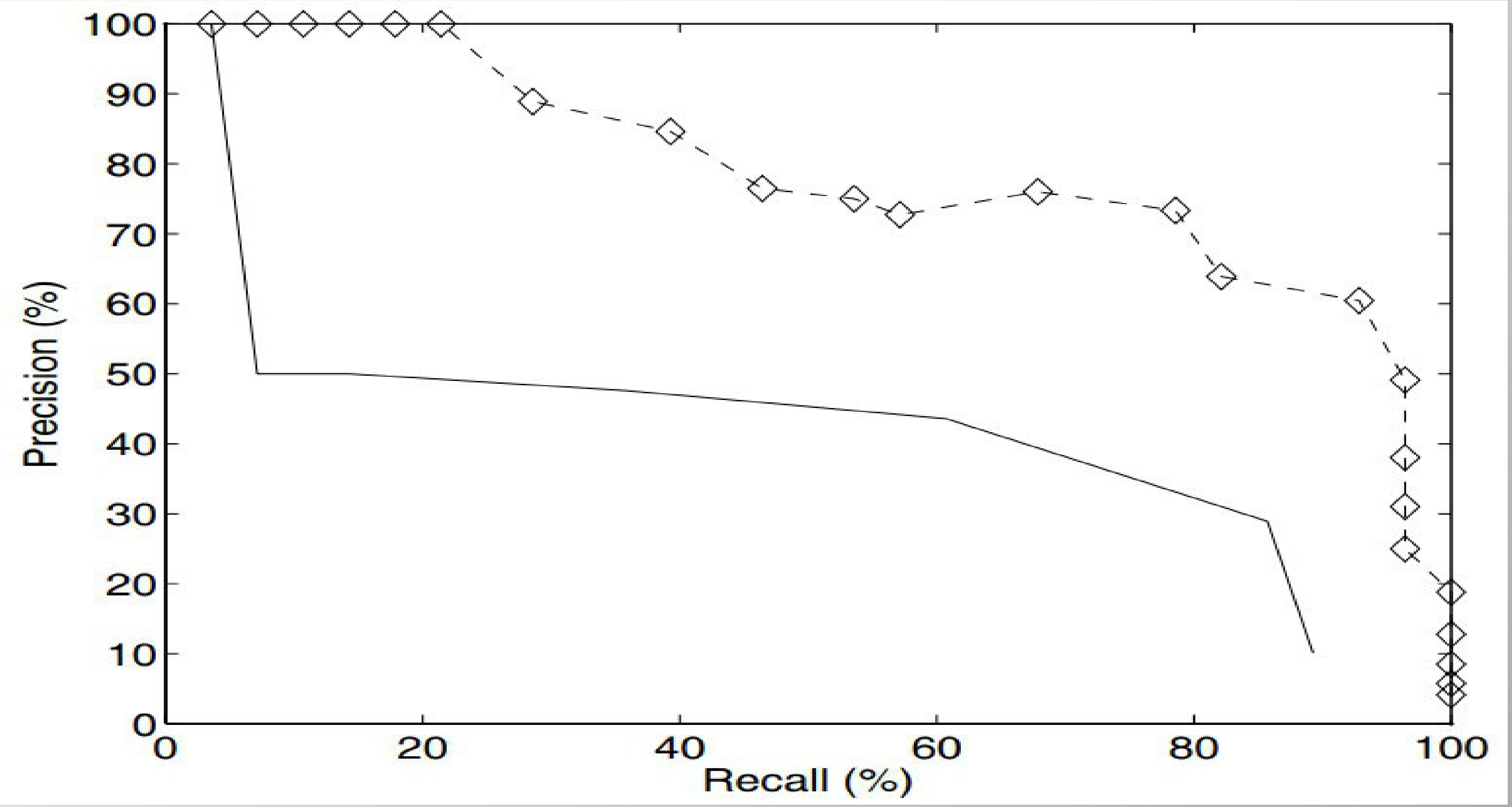
Advantages

LSI works better than older approaches to IR like boolean retrieval or vector space model by increasing recall

Synonyms are a huge challenge for boolean retrieval, because the query often returns irrelevant results and misses the important ones

Advantages

LSI (dotted line) vs. vector space model



Lars Eldén, Matrix Methods in Data Mining and Pattern Recognition, S. 136

Advantages

LSI is also used for automated document categorization

In several occasion it has been shown, the way a human and a LSI system categorizes text are very similar

Advantages

The mathematic approach to LSI grants the system language independency

A query can be made in a language (e.g. German) and the results can be in a different language (e.g. English)

This is possible by reducing words to concepts and the purely semantic approach to information

Advantages

One could even go further and eliminate the (natural) language aspect completely

A study with the MEDLINE abstracts has shown that LSI is able to classify documents with other means (in this example clasify genes and biological information)

LSI can also be used to understand software code

Uses of LSI Systems

LSI can be used to achieve:

- Information discovery
- Automated document classification
- Text summarization
- Automatic keyword annotation (e.g. images)
- Essay Scoring
- Spam filtering

Sources and further reading

University of Freiburg, Germany

Information Retrieval WS17/18, Lecture 10

https://www.youtube.com/watch?v=CwBn0voJDaw

Lars Eldén, Matrix Methods in Data Mining and Pattern Recognition

Any questions left?

Thank you for your attention!

Basic definitions for the presentation

1. Document: a collection of words-The instance of “rows” of our dataset
2. Body: a collection of documents- our entire data set
3. Dictionary: The set of all words that appear in at least one document in our body
4. Topic: a collection of words that co-occur

Latent: Features that are “hidden” in the data which can not be directly measured. These features are essential to the data, but are not original features of the data set.

3 Steps to LSI

1. Construct a weighted term-document matrix
2. Apply SVD on the matrix
3. Use the result to identify the concepts contained in the text

Document-Term Matrix

a basic idea of a Document-Term Matrix is that documents can be represented as points in Euclidean space or **vectors**

	big	blue	bus	car	orange	small	the	tiny	yellow
"the big yellow bus"	1	0	1	0	0	0	1	0	0
"the small yellow car"	0	0	0	1	0	1	1	0	1
the big blue car"	1	1	0	1	0	0	1	0	0
the tiny orange bus"	0	0	1	0	1	0	1	1	0

In [1]:
```python
from sklearn.feature_extraction.text import CountVectorizer



body = ["the big yellow bus",
        "the small yellow car",
        "the big blue car",
        "the tiny orange bus"]

vectorizer = CountVectorizer()
bag_of_words = vectorizer.fit_transform(body)
bag_of_words.todense()
```

Out[1]:
```
matrix([[1, 0, 1, 0, 0, 0, 1, 0, 1],
        [0, 0, 0, 1, 0, 1, 1, 0, 1],
        [1, 1, 0, 1, 0, 0, 1, 0, 0],
        [0, 0, 1, 0, 1, 0, 1, 1, 0]])
```

In [8]:

```
from sklearn.decomposition import TruncatedSVD
import pandas as pd

svd = TruncatedSVD(n_components=2)
lsa = svd.fit_transform(bag_of_words)


topic_encoded_df = pd.DataFrame(lsa, columns= ["topic_1", "topic_2"])
topic_encoded_df["body"]=body
display(topic_encoded_df[["body","topic_1", "topic_2" ]])
```

	body	topic_1	topic_2
0	the big yellow bus	1.694905	0.299524
1	the small yellow car	1.515851	-0.769110
2	the big blue car	1.515851	-0.769110
3	the tiny orange bus	1.266186	1.440585

In [3]:

```
dictionary = vectorizer.get_feature_names()
dictionary
```

Out[3]:

```
['big', 'blue', 'bus', 'car', 'orange', 'small', 'the', 'tiny', 'yello
w']
```

In [4]:

```
encoding_matrix = pd.DataFrame(svd.components_,
                               index = ['topic_1', 'topic_2'],
                               columns=dictionary).T
encoding_matrix
```

Out[4]:

	topic_1	topic_2
big	0.353937	-0.140256
blue	0.167100	-0.229718
bus	0.326416	0.519736
car	0.334199	-0.459436
orange	0.139578	0.430274
small	0.167100	-0.229718
the	0.660615	0.060300
tiny	0.139578	0.430274
yellow	0.353937	-0.140256

In [11]:

```
import numpy as np
encoding_matrix['abs_topic_1'] = np.abs(encoding_matrix['topic_1'])
encoding_matrix['abs_topic_2'] = np.abs(encoding_matrix['topic_2'])
#encoding_matrix.sort_values('abs_topic_1', ascending=False)
encoding_matrix.sort_values('abs_topic_2', ascending=False)
```

Out[11]:

	topic_1	topic_2	abs_topic_1	abs_topic_2
bus	0.326416	0.519736	0.326416	0.519736
car	0.334199	-0.459436	0.334199	0.459436
orange	0.139578	0.430274	0.139578	0.430274
tiny	0.139578	0.430274	0.139578	0.430274
small	0.167100	-0.229718	0.167100	0.229718
blue	0.167100	-0.229718	0.167100	0.229718
yellow	0.353937	-0.140256	0.353937	0.140256
big	0.353937	-0.140256	0.353937	0.140256
the	0.660615	0.060300	0.660615	0.060300

In [12]:

```
import numpy as np
encoding_matrix['abs_topic_1'] = np.abs(encoding_matrix['topic_1'])
encoding_matrix['abs_topic_2'] = np.abs(encoding_matrix['topic_2'])
encoding_matrix.sort_values('abs_topic_1', ascending=False)
#encoding_matrix.sort_values('abs_topic_2', ascending=False)
```

Out[12]:

	topic_1	topic_2	abs_topic_1	abs_topic_2
the	0.660615	0.060300	0.660615	0.060300
big	0.353937	-0.140256	0.353937	0.140256
yellow	0.353937	-0.140256	0.353937	0.140256
car	0.334199	-0.459436	0.334199	0.459436
bus	0.326416	0.519736	0.326416	0.519736
blue	0.167100	-0.229718	0.167100	0.229718
small	0.167100	-0.229718	0.167100	0.229718
orange	0.139578	0.430274	0.139578	0.430274
tiny	0.139578	0.430274	0.139578	0.430274

In [ ]:

Literaturverzeichnis

- Eldén, Lars. Matrix Methods in Data Mining and Pattern Recognitio
  Linköping University, Linköping, Sweden. 2007.
- Cohen, William. Latent semantic indexing. Carnegie Mellon Univers
  Information Extraction Fall 2010
  (http://curtis.ml.cmu.edu/w/courses/index.php/Latent_semantic_inc
- Introduction to Latent Semantic Analysis.
  https://www.youtube.com/watch?v=hB51kkus-Rc