Information Retrieval Models

Advanced Topics

Using n-grams

N-grams

- Until now we have considered a document as a bag of independent words

- Sometimes, words appearing in sequence bring a specific meaning than single words

- e.g., video game, credit card, open source

- Counting them as (additional) terms in our model could improve the performances of our IR engine

HOWTO

- Using the scikit-learn vectorizers (CountVectorizer, TfIdfVectorizer, HashingVectorizer) just add a further parameter ngram_range=tuple

- E.g. to consider bigrams

- vectorizer=TfidfVectorizer(tokenizer=my_tokenizer, ngram_range=(1,2))

Example

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import snowball
import re

def my_tokenizer(text):
        sw=stopwords.words('english')
        stemmer=snowball.SnowballStemmer(language="english")
        tokens=word_tokenize(text)
        pruned=[stemmer.stem(t) for t in tokens if re.search(r"^[a-zA-Z]",t) and not t in sw]
        return pruned

#initialize the vectorizer considering bigrams
vectorizer=TfidfVectorizer(tokenizer=my_tokenizer,ngram_range=(1,2))

#Add the files in the corpus, a document on each line
corpus=["This document describes racing cars",
        "This document is about videos of table games",
        "This is a nice racing video game",
         "Video killed the radio star"]

#creates the model
model=vectorizer.fit_transform(corpus)
#print the dictionary
print(vectorizer.get_feature_names_out())
```

Dictionary being printed…

['car' 'describ' 'describ race' 'document' 'document describ'

'document video' 'game' 'kill' 'kill radio' 'nice' 'nice race' 'race'

'race car' 'race video' 'radio' 'radio star' 'star' 'tabl' 'tabl game'

'video' 'video game' 'video kill' 'video tabl']

Computing similarity…

```
#adds a query to the model
query=vectorizer.transform(["video game"])

cos=cosine_similarity(query,model)
print(cos)
```

Similarity: bigrams vs. unigrams

- Result would be:

```
[[0.          0.30389824 0.59923094 0.11299246]]
```

- With unigrams it would have been:

```
[[0.          0.62306963 0.62306963 0.21757626]]
```

  That is, the second and third document would be equally relevant!

Query Reformulation

Query Modification

Problem: How can we reformulate the query to help a user who is trying several searches to get at the same information?

- Thesaurus expansion:
  - Suggest terms similar to query terms
- Relevance feedback:
  - Suggest terms (and documents) similar to retrieved documents that have been judged to be relevant

Relevance Feedback

- Main Idea:
  - Modify existing query based on relevance judgements
    - Extract terms from relevant documents and add them to the query
    - AND/OR re-weight the terms already in the query
- There are many variations:
  - Usually positive weights for terms from relevant docs
  - Sometimes negative weights for terms from non-relevant docs
- Users, or the system, guide this process by selecting terms from an automatically-generated list

Rocchio Method

Rocchio automatically:

- Re-weights terms
- Adds in new terms (from relevant docs)
  - have to be careful when using negative terms
  - Rocchio is **not** a machine learning algorithm

Rocchio Method

$$Q_1 = \alpha\ Q_0 + \frac{\beta}{n_1}\sum_{i=1}^{n_1} R_i - \frac{\gamma}{n_2}\sum_{i=1}^{n_2} S_i$$

where

Q_0 = the vector for the initial query

R_i = the vector for the relevant document i

S_i = the vector for the non - relevant document i

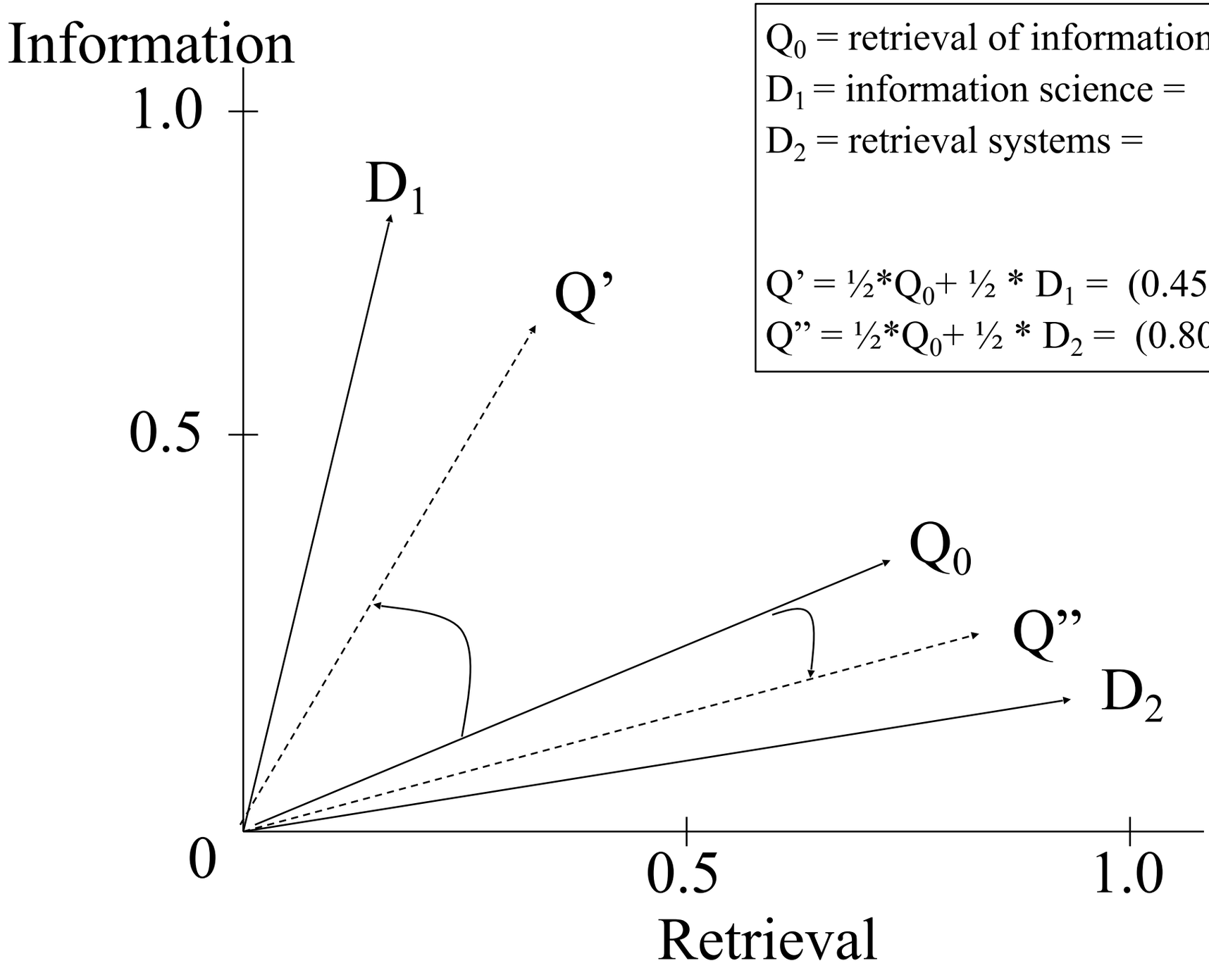
n_1 = the number of relevant documents chosen

n_2 = the number of non - relevant documents chosen

α, β and γ tune the importance of relevant and nonrelevant terms

(in some studies best to set β to 0.75 and γ to 0.25)

Rocchio/Vector Illustration



Q_0 = retrieval of information = (0.7,0.3)
D_1 = information science = (0.2,0.8)
D_2 = retrieval systems = (0.9,0.1)

$Q' = ½*Q_0 + ½ * D_1 = (0.45,0.55)$
$Q'' = ½*Q_0 + ½ * D_2 = (0.80,0.20)$

Latent Semantic Indexing

…beyond Vector Space Models…

Latent Semantic Indexing

- vector-space similarity approach works only if the terms of the query are explicitly present in the relevant documents
- rich expressive power of natural language
  - often queries contain terms that express concepts related to text to be retrieved

Synonymy

- The same concept can be expressed using different sets of terms
  - e.g. bandit, brigand, thief
- Negatively affects recall
- Possible solution:
  - we can use a thesaurus to bring back synonymous to the same form
  - but this would affect the precision ☹

Polysemy

- Identical terms can be used in very different semantic contexts
  - e.g. bank, chip
    - repository where important material is saved
    - the slope beside a body of water
- Negatively affects precision
- Possible solution:
  - controlled dictionary and consistent use of terms
  - .. but expensive and difficult to check

Word independence

- With the vector space model, we are assuming independence among terms in a document
- … however we know this is not true!!
- The word “card” after “credit” assumes a specific meaning
- Possible solution:
  - We could use more complex probabilistic models accounting for bigrams
  - We could build models explicitly taking into account complex relationships between terms and documents
  - Too complex and expensive…

Thus…

- It looks like words do not represent documents in the best possible way
- From a semantic point of view, a document is not a collection of words
- …it rather provides a set of concepts
- We should find a way to move from words to concepts

Latent Semantic Indexing

- Developed at Bellcore in 1988
  - Patented in 1989
  - http://lsi.argreenhouse.com/lsi/LSI.html
- Some references
  - Dumais, S. T., Furnas, G. W., Landauer, T. K. and Deerwester, S. (1988), "Using latent semantic analysis to improve information retrieval." In Proceedings of CHI'88: Conference on Human Factors in Computing, New York: ACM, 281-285.
  - Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R.A. (1990) "Indexing by latent semantic analysis." Journal of the Society for Information Science, 41(6), 391-407.
  - Foltz, P. W. (1990) "Using Latent Semantic Indexing for Information Filtering". In R. B. Allen (Ed.) Proceedings of the Conference on Office Information Systems, Cambridge, MA, 40-47.

Idea

"We would like a representation in which a set of terms, which by itself is incomplete and unreliable evidence of the relevance of a given document, is replaced by some other set of entities which are more reliable indicants. We take advantage of the implicit higher-order (or latent) structure in the association of terms and documents to reveal such relationships."

(Deerwester et al):

Using SVD

- Data-driven
- LSI uses linear algebra technique called singular value decomposition (SVD)
  - attempts to estimate the hidden structure
  - discovers the most important associative patterns between words and concepts
- In other words…
  - The analysis is moved from the space of terms to the space of concepts

What is SVD?

- Given a term to document matrix X with n terms and m documents
- SVD decomposes a matrix into three matrices

$$X = U\Sigma V^T$$

- Σ is a k × k diagonal matrix containing singular values
  - where k is the rank of X

- U (m × k) and V (k × n) contains eigenvectors, i.e., linearly independent vectors

Basically…

- Instead of representing documents as a set of correlated factors (terms), we represent documents as set of uncorrelated factors (concepts)
- Some of these factors in the orthonormal matrices U and V are very small
- we can ignore them putting them to zero

Basic steps

- Let **X** denote a term-document matrix

- $\mathbf{X} = [\mathbf{x}_1 \ldots \mathbf{x}_n]^T$
  - each row is the vector-space representation of a document
  - each column contains occurrences of a term in each document in the dataset

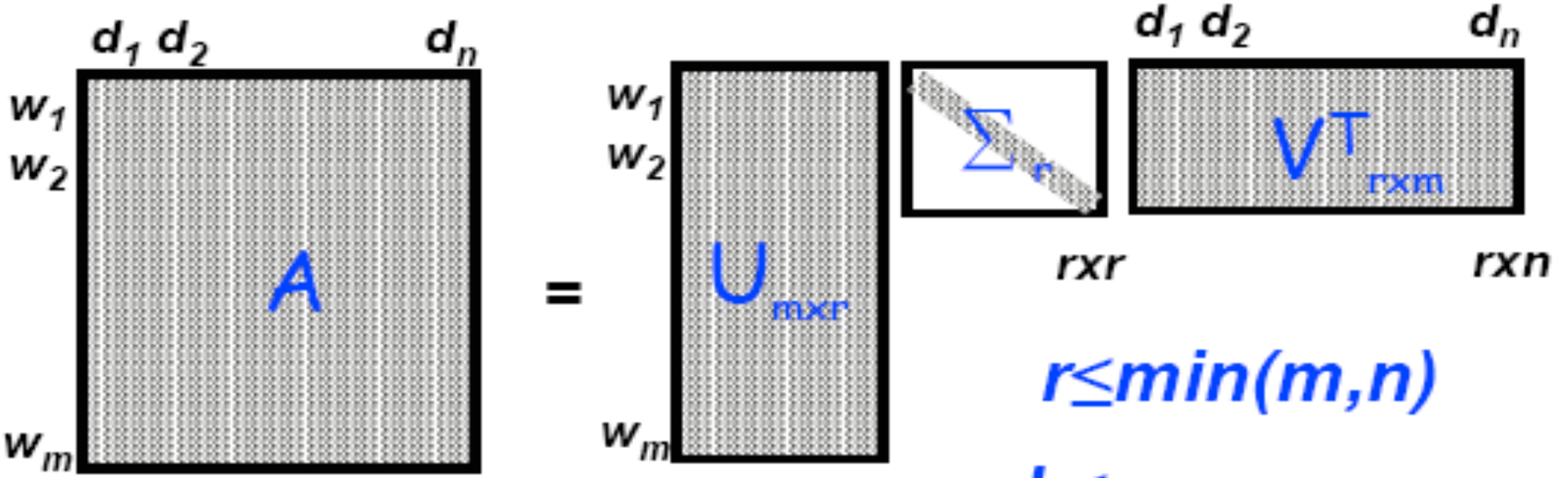
- Latent semantic indexing
  - compute the SVD of **X**:

$$X = U\Sigma V^T$$

Σ - singular value matrix

  - set to zero all but largest K singular values - $\hat{\Sigma}$
  - obtain the reconstruction of **X** as: $\hat{X} = U\hat{\Sigma}V^T$

SVD: Dimensionality Reduction



Both U and V has orthonormal column vectors

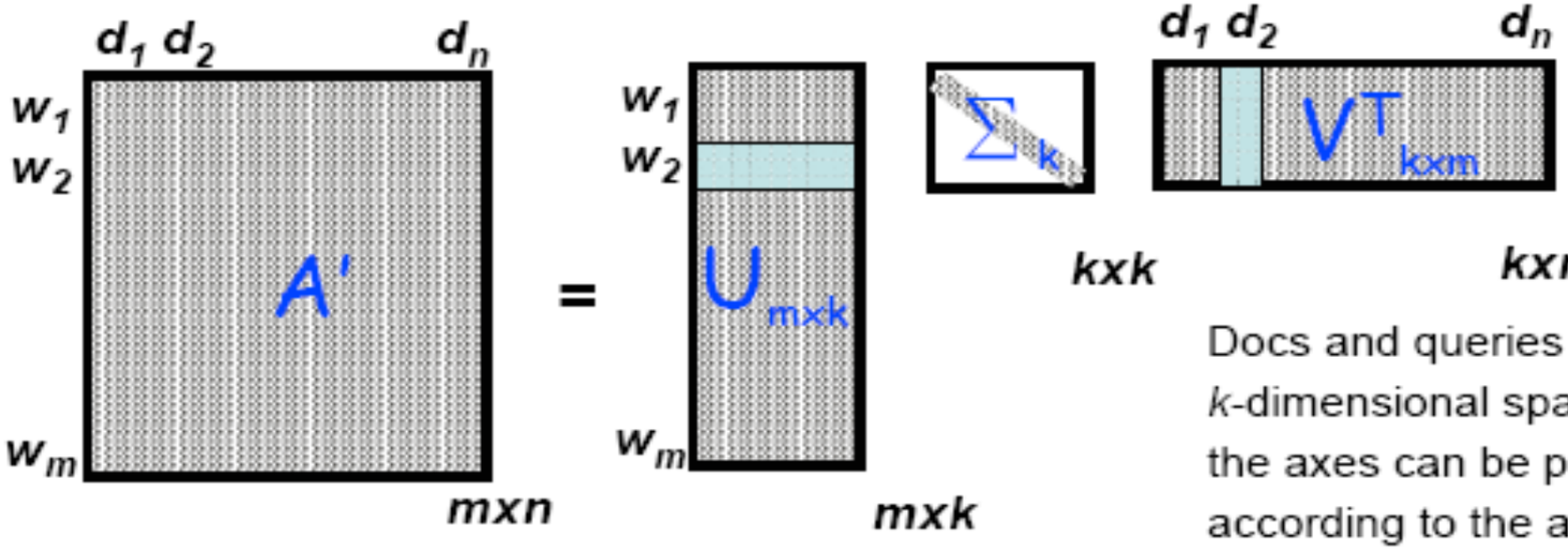
$U^TU=I_{kXk}$

$V^TV=I_{kXk}$

$r \leq min(m,n)$

$k \leq r$

$||A||_F^2 \geq ||A'||_F^2$



Docs and queries are represented in a *k*-dimensional space. The quantities of the axes can be properly weighted according to the associated diagonal values of Σ_k

LSI Example

Consider a collection of documents:

d1: Indian government goes for open-source software
d2: Debian 3.0 Woody released
d3: Wine 2.0 released with fixes for Gentoo 1.4 and Debian 3.0
d4: gnuPOD released: iPOD on Linux… with GPLed software
d5: Gentoo servers running at open-source mySQL database
d6: Dolly the sheep not totally identical clone
d7: DNA news: introduced low-cost human genome DNA chip
d8: Malaria-parasite genome database on the Web
d9: UK sets up genome bank to protect rare sheep breeds
d10: Dolly's DNA damaged

LSI Example: term-documents matrix

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
open-source	1	0	0	0	1	0	0	0	0	0
software	1	0	0	1	0	0	0	0	0	0
Linux	0	0	0	1	0	0	0	0	0	0
released	0	1	1	1	0	0	0	0	0	0
Debian	0	1	1	0	0	0	0	0	0	0
Gentoo	0	0	1	0	1	0	0	0	0	0
database	0	0	0	0	1	0	0	1	0	0
Dolly	0	0	0	0	0	1	0	0	0	1
sheep	0	0	0	0	0	1	0	0	0	0
genome	0	0	0	0	0	0	1	1	1	0
DNA	0	0	0	0	0	0	2	0	0	1

Some cosine similarities

sim(d1,d3)=0

sim(d1,d4)=0.3

sim(d1,d5)=0.7

sim(d7,d10)=0.63

sim(d8,d10)=0

…However d3 is about (open source) Linuxes, and d8 about (DNA) manipulation

U (term matrix)

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
open-source	-0.05	0.225	0.516	-0.32	-0.16	0.311	-0.4	0.08	-0.08	0.536941
software	-0.03	0.287	0.027	-0.68	0.208	-0.02	-0.19	-0.01	0.117	-0.5984
Linux	-0.01	0.198	-0.12	-0.34	0.189	-0.18	0.509	-0.16	-0.04	0.381802
released	-0.05	0.651	-0.35	0.041	0.107	-0.13	0.107	0.102	-0.09	0.240537
Debian	-0.04	0.454	-0.23	0.378	-0.08	0.051	-0.4	0.263	-0.05	-0.14127
Gentoo	-0.06	0.397	0.294	0.259	-0.27	0.249	0.253	-0.61	0.264	-2.07E-01
database	-0.14	0.159	0.582	0.135	-0.04	-0.27	0.396	0.501	-0.25	-2.34E-01
Dolly	-0.19	-0.03	-0.15	-0.2	-0.69	-0.33	0.029	0.215	0.513	7.56E-02
sheep	-0.03	-0.01	-0.06	-0.11	-0.41	-0.39	-0.18	-0.39	-0.69	-8.06E-02
genome	-0.51	-0.03	0.21	0.181	0.396	-0.52	-0.32	-0.24	0.242	1.25E-01
DNA	-0.82	-0.12	-0.21	-0.08	-0.04	0.434	0.146	0.06	-0.21	-6.58E-02

The first k columns represent correlations between terms and concepts

$$\Sigma$$

2.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	1.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	1.70	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	1.63	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	1.36	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.92	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.67	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.25

Diagonal matrix referred as concept matrix

V (document matrix)

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
d1	-0.03	0.213	0.28	-0.59	0.031	0.21105	-0.63898	0.1	0.078	-0.24544
d2	-0.03	0.461	-0.3	0.246	0.015	-0.05682	-0.32204	0.543	-0.28	0.396452
d3	-0.06	0.627	-0.15	0.398	-0.15	0.12629	-0.04561	-0.37	0.25	-0.43182
d4	-0.03	0.474	-0.23	-0.57	0.309	-0.24411	0.46553	-0.11	-0.02	0.095603
d5	-0.1	0.326	0.719	0.044	-0.29	0.21263	0.27717	-0.05	-0.12	0.379892
d6	-0.09	-0.02	-0.11	-0.18	-0.68	-0.52451	-0.16444	-0.26	-0.35	-2.02E-02
d7	-0.84	-0.11	-0.11	0.017	0.189	0.25642	-0.02886	-0.18	-0.35	-2.78E-02
d8	-0.26	0.055	0.409	0.185	0.215	-0.58169	0.08525	0.382	-0.01	-0.43859
d9	-0.2	-0.01	0.108	0.106	0.242	-0.38229	-0.3474	-0.36	0.482	0.497578
d10	-0.4	-0.06	-0.19	-0.16	-0.45	7.82E-02	1.91E-01	0.408	0.603	3.91E-02

The first k columns represent correlations between documents and concepts

Let’s recompute the cosines

Only considering the first 2 columns of V (k=2)

sim(d1,d2)=0.99
sim(d1,d3)=0.99
sim(d7,d10)=0.99
sim(d8,d10)=0.99
sim(d1,d10)=-0.06

(well d1 and d10 are not really related)

- similarity between documents belonging to the same cluster increased
- similarity between documents belonging to different clusters decreased

U’ (k=2)

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
open-source	-0.05	0.225	0	0	0	0	0	0	0	0
software	-0.03	0.287	0	0	0	0	0	0	0	0
Linux	-0.01	0.198	0	0	0	0	0	0	0	0
released	-0.05	0.651	0	0	0	0	0	0	0	0
Debian	-0.04	0.454	0	0	0	0	0	0	0	0
Gentoo	-0.06	0.397	0	0	0	0	0	0	0	0
database	-0.14	0.159	0	0	0	0	0	0	0	0
Dolly	-0.19	-0.03	0	0	0	0	0	0	0	0
sheep	-0.03	-0.01	0	0	0	0	0	0	0	0
genome	-0.51	-0.03	0	0	0	0	0	0	0	0
DNA	-0.82	-0.12	0	0	0	0	0	0	0	0

Σ' (k=2)

2.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

V’ (k=2)

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
d1	-0.03	0.213	0	0	0	0	0	0	0	0
d2	-0.03	0.461	0	0	0	0	0	0	0	0
d3	-0.06	0.627	0	0	0	0	0	0	0	0
d4	-0.03	0.474	0	0	0	0	0	0	0	0
d5	-0.1	0.326	0	0	0	0	0	0	0	0
d6	-0.09	-0.02	0	0	0	0	0	0	0	0
d7	-0.84	-0.11	0	0	0	0	0	0	0	0
d8	-0.26	0.055	0	0	0	0	0	0	0	0
d9	-0.2	-0.01	0	0	0	0	0	0	0	0
d10	-0.4	-0.06	0	0	0	0	0	0	0	0

Reconstructed term-document matrix (k=2)

$$X' = U' * \Sigma' * V'^{T}$$

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
open-source	0.119	0.253	0.346	0.26	0.188	0.002	0.05	0.062	0.02	0.017
software	0.149	0.319	0.435	0.328	0.23	-0.01	-0.02	0.054	0.005	-0.02
Linux	0.102	0.22	0.299	0.226	0.158	-0	-0.02	0.035	0.002	-0.02
released	0.337	0.724	0.986	0.744	0.522	-0.01	-0.06	0.118	0.008	-0.05
Debian	0.235	0.505	0.687	0.519	0.364	-0.01	-0.04	0.083	0.006	-0.03
Gentoo	0.208	0.445	0.606	0.457	0.326	-0	0.028	0.092	0.021	0.002
database	0.092	0.187	0.259	0.193	0.159	0.025	0.258	0.111	0.067	0.117
Dolly	-0	-0.02	-0.02	-0.02	0.022	0.044	0.419	0.12	0.098	0.198
sheep	-0	-0	-0.01	-0	0.003	0.008	0.077	0.022	0.018	0.036
genome	0.025	0.014	0.034	0.014	0.107	0.116	1.107	0.329	0.262	0.521
DNA	0.002	-0.06	-0.06	-0.06	0.114	0.19	1.795	0.518	0.422	0.846

How to choose k?

- Finding optimal dimension for semantic space
  - precision-recall improve as dimension is increased until hits optimal, then slowly decreases until it hits standard vector model
  - run SVD once with big dimension, say k = 1000, then can test dimensions ≤ k
  - in many tasks 150-350 works well, still room for research
- A lot depends on the application
- There are also procedures to automatically choose k

LSI: Pros and cons

Pros:

- Able to deal with synonymy and homonymy
- Stemming could be avoided
  - However it works better with stemming!
- Increases similarity between documents of the same cluster
- Decreases similarity between documents of different clusters

Cons:

- More expensive than traditional Vector Space Models (SVD computation)
- Difficult to add new documents
- Determining the optimal k is a crucial issue
- Often needs a large document corpus

LSI in Python

Using gensim…

```
from gensim import corpora
from gensim import models
from gensim import similarities
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import snowball
import re

def my_tokenizer(text):
    """tokenization function"""
    sw=stopwords.words('english')
    stemmer=snowball.SnowballStemmer(language="english")
    tokens=word_tokenize(text)
    pruned=[stemmer.stem(t.lower()) for t in tokens \
            if re.search(r"^[a-zA-Z]",t) and not t.lower() in sw]
    return pruned

documents=["Indian government goes for open source software",
"Debian 3.0 Woody released",
"Wine 2.0 released with fixes for Gentoo 1.4 and Debian 3.0",
"gnuPOD released: iPOD on Linux… with GPLed software",
"Gentoo servers running at open source mySQL database",
"Dolly the sheep not totally identical clone",
"DNA news: introduced low-cost human genome DNA chip",
"Malaria-parasite genome database on the Web",
"UK sets up genome bank to protect rare sheep breeds",
"Dolly's DNA damaged"]
```

Creating LSI model and similarities

```
texts=[]
for d in documents:
    # creates an array of tokenized documents
    texts.append(my_tokenizer(d))

#creates the dictionary for the document corpus
dictionary = corpora.Dictionary(texts)
#creates a bag of word corpus
bow_corpus=[dictionary.doc2bow(text) for text in texts]

#creates a tf-idf model from the bag of word corpus
tfidf = models.TfidfModel(bow_corpus)

#extracts the tf-idf corpus
corpus_tfidf = tfidf[bow_corpus]

#creates the LSI model, with 2 topics
lsi_model=models.LsiModel(corpus_tfidf,num_topics=2,id2word=dictionary)

#creates an index that facilitates the computation of similarities
index = similarities.MatrixSimilarity(lsi_model[corpus_tfidf])
```

Steps

- First we create a dictionary for the document corpus
- Then, we extract a bag of word corpus
- Then, we create a tf-idf model and extract its corpus
- After, on top of the tf-idf corpus we can create the LSI model, specifying the number of topics
- Finally, we can create the similarity matrix

Let’s print some similarities…

```
print(list(index)[0])
print(list(index)[5])
```

```
[1.0000001  0.97902197 0.9843578  0.9853968  0.99798703 0.10885489
 0.13524207 0.68249047 0.19551672 0.11114562]
[ 0.10885489 -0.09597263 -0.06798209 -0.06199703  0.17167735  1.
  0.9996466   0.8008437   0.9961556   0.9999973 ]
```

- Document 1 is very similar to the first 5 documents (though it has some similarity with the sixth one)
- Document 6 is similar to the last 5 documents

Running a query…

```
#tokenizes the query
query_document = my_tokenizer("DNA")

#indexes the query using the documents' dictionary
query_bow = dictionary.doc2bow(query_document)
query_lsi = lsi_model[query_bow]  # convert the query to LSI space


#computes the similarity between the query and the documents
sims = index[query_lsi]
print(list(enumerate(sims)))
```

Note

The instruction

```
lsi_model[query_bow]
```

converts the query into the LSI space

Result

```
[(0, 0.118368536), (1, -0.08643689), (2, -0.05842559),
(3, -0.05243706), (4, 0.18110284), (5, 0.9999541), (6,
0.99985534), (7, 0.8065415), (8, 0.9969488), (9,
0.99997354)]
```

Unsurprisingly, the query is very similar to the last 5 documents, even if not all of them contain the term “DNA”

Getting the LSI model

Useful when we want to use it for machine learning, see later…

```
corpus_lsi=lsi_model[corpus_tfidf]
for doc in corpus_lsi:
    scores=[topic_score for (topic_id, topic_score) in doc]
    print(scores)
```

Result

[0.322585702702323, -0.011813322545723082]

[0.6534290583018245, 0.1112154310261141 7]

[0.7148643126316745, 0.1011054806767825 4]

[0.3631170170144971, 0.04913716990076994 6]

[0.437157635919749 1, -0.04389094154631413 6]

[0.0338480926985675 3, -0.4662671631018059 4]

[0.0647395470107032 8, -0.6514432585710758]

[0.1601939884848437 3, -0.1846639499143924 7]

[0.0395612516649797 6, -0.2448641560016971 7]

[0.0564688980827505 8, -0.7538129187682796]

Note

- We no longer have a term-document matrix
- We have a concept-document matrix
- Note that sometimes high correlations are negatives (see topic 2 for the second half of the documents)
- This is because the weighting of the words in the topics is negative too, so the product becomes positive (see next slides)

Printing topics

```
print(lsi_model.show_topic(0))
```

```
 [('debian', 0.4434854694998996),
 ('releas', 0.3938718465810023),
 ('woodi', 0.35142421095730686),
 ('gentoo', 0.29585355292326354),
 ('fix', 0.28306005410237245),
 ('wine', 0.2830600541023724),
 ('open', 0.17417437157754478),
 ('sourc', 0.17417437157754465),
 ('softwar', 0.15920294729105 03),
 ('databas', 0.14610110507158 3)]
```

Printing topics

```
print(lsi_model.show_topic(1))
```

```
 [('dna', -0.5359729615542284),
 ('dolli', -0.4042679632122831),
 ('damag', -0.4027363894775456),
 ('human', -0.18203378118963037),
 ('chip', -0.18203378118963037),
 ('news', -0.18203378118963037),
 ('introduc', -0.18203378118963037),
 ('low-cost', -0.18203378118963037),
 ('genom', -0.17581616785376525),
 ('total', -0.17564030867034489)]
```

Notes

- As you have seen, the second topic has negative coefficients for its relevant words

- This explains the high negative correlations for the relevant documents in the document/topic matrix

Computing the model coherence

- Can be useful to determine the optimal number of topics

- You should pick either the maximum or the “knee” of the coherence by varying the number of the topics

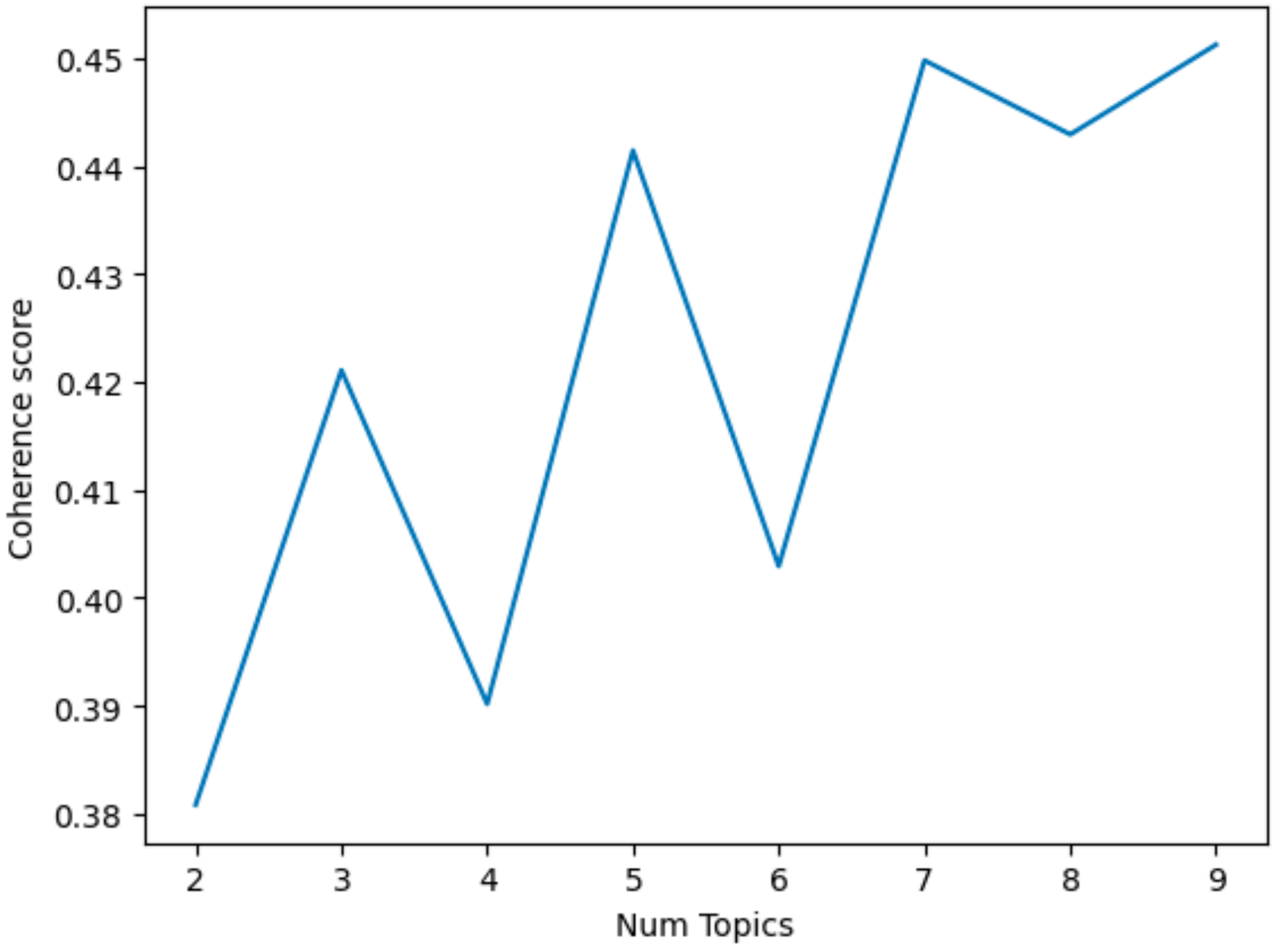
Plotting coherence...

```
from gensim.models import CoherenceModel
import matplotlib.pyplot as plt

coherence_values = []
model_list = []
for num_topics in range(2, 10):
    model = models.LsiModel(corpus_tfidf, num_topics=num_topics, id2word=dictionary)
    model_list.append(model)
    coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
    coherence_values.append(coherencemodel.get_coherence())

plt.plot(range(2, 10), coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
```

Result



We can get better results by increasing the number of topics
Be careful about excessive fragmentation

Other topic models

Latent Dirichlet Allocation

- The Latent Dirichlet Allocation (LDA) model is relatively similar to LSI
- Reference: D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” The Journal of Machine Learning Research, vol. 3, pp. 993–1022, 2003.
- However, it is based on probability distributions
  - A topic is defined as a distribution of words (i.e., different words will have more or less importance in a topic)
  - A document is a distribution of topics (i.e., different topics will have more or less importance in a document
- Again, implementation available in gensim