Computational Communication Science 2
Week 4 - Lecture
»Text similiarity and differences«

Marthe Möller
Anne Kroon

A.M.Moller@uva.nl, @MartheMoller
a.c.kroon@uva.nl, @annekroon

April, 2022

Digital Society Minor, University of Amsterdam

Today

Cosine Similarity

Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- -1 means the opposite (180 degrees)
- 0 means orthogonal vectors (90 degrees)
- 1 means vectors are the same (0 degrees)

Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathrm{A} \cdot \mathrm{B}}{\|\mathrm{A}\|\|\mathrm{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- -1 means the opposite (180 degrees)
- 0 means orthogonal vectors (90 degrees)
- 1 means vectors are the same (0 degrees)

title		Avatar	Pirates of the Caribbean: At World's End	Spectre	The Dark Knight Rises	John Carter
	genre	action adventure fantasy science fiction	adventure fantasy action	action adventure crime	action crime drama thriller	action adventure science fiction
title	genres					
Avatar	action adventure fantasy science fiction	1.000000	0.691870	0.315126	0.084696	0.859850
Pirates of the Caribbean: At World's End	adventure fantasy action	0.691870	1.000000	0.455470	0.122417	0.366490
Spectre	action adventure crime	0.315126	0.455470	1.000000	0.473354	0.366490
The Dark Knight Rises	action crime drama thriller	0.084696	0.122417	0.473354	1.000000	0.098501
John Carter	action adventure science fiction	0.859850	0.366490	0.366490	0.098501	1.000000

Interpretation of cosine similarity [1]

[1] https://www.learndatasci.com/glossary/cosine-similarity/

how can we calculate this in python?

Let's review a practical application [2].

[2] https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb

Implementation in Python

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import pandas as pd

doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()

vec = CountVectorizer(stop_words='english')
count_matrix = vec.fit_transform([doc1, doc2])

print(pd.DataFrame(count_matrix.A, columns=vec.get_feature_names()).to_string())
```

	breakfast	drink	eat	like	tea	usually
0	1	1	1	0	1	1
1	1	0	0	1	1	0

Implementation in Python

```
from sklearn.feature_extraction.text import CountVectorizer,
↪ TfidfVectorizer
import pandas as pd

doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()

vec = CountVectorizer(stop_words='english')
count_matrix = vec.fit_transform([doc1, doc2])

print(pd.DataFrame(count_matrix.A,
↪ columns=vec.get_feature_names()).to_string())
```

	breakfast	drink	eat	like	tea	usually
0	1	1	1	0	1	1
1	1	0	0	1	1	0

Implementation in Python

```python
doc1_vector = pd.DataFrame(count_matrix.A, columns=vec.get_feature_names()).T[0].to_list()
doc2_vector = pd.DataFrame(count_matrix.A, columns=vec.get_feature_names()).T[1].to_list()

print(f"The vector belonging to doc1: {doc1_vector}")
print(f"The vector belonging to doc2: {doc2_vector}")
```

```
The vector belonging to doc1: [1, 1, 1, 0, 1, 1]
The vector belonging to doc2: [1, 0, 0, 1, 1, 0]
```

Implementation in Python

```python
doc1_vector = pd.DataFrame(count_matrix.A,
↪   columns=vec.get_feature_names()).T[0].to_list()
doc2_vector = pd.DataFrame(count_matrix.A,
↪   columns=vec.get_feature_names()).T[1].to_list()

print(f"The vector belonging to doc1: {doc1_vector}")
print(f"The vector belonging to doc2: {doc2_vector}")
```

```
The vector belonging to doc1: [1, 1, 1, 0, 1, 1]
The vector belonging to doc2: [1, 0, 0, 1, 1, 0]
```

Implementation in Python

Now, lets populate the formula. 1.Execute the part of the formula in the numerator. Specifically, take the dot product of the vectors:

$$\sum_{i=1}^{n} A_i B_i$$

```python
dot_product = sum([num1 * num2 for num1, num2 in zip(doc1_vector, doc2_vector)])
print(dot_product)
```

Implementation in Python

Now, lets populate the formula. 1.Execute the part of the formula in the numerator. Specifically, take the dot product of the vectors:

$$\sum_{i=1}^{n} A_i B_i$$

```
dot_product = sum([num1 * num2 for num1, num2 in zip(doc1_vector, doc2_vector)])
print(dot_product)
```

2

Implementation in Python

2.Execute the part of the formula in the denumerator. Take the product of the lengths of the vectors.

$$\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}$$

```
import math
doc1_ = math.sqrt(sum( [i**2 for i in doc1_vector]) )
doc2_ = math.sqrt(sum( [i**2 for i in doc2_vector]) )

doc1_ * doc2_
```

3.872983346207417

Implementation in Python

2.Execute the part of the formula in the denumerator. Take the product of the lengths of the vectors.

$$\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}$$

```
import math
doc1_ = math.sqrt(sum( [i**2 for i in doc1_vector]) )
doc2_ = math.sqrt(sum( [i**2 for i in doc2_vector]) )

doc1_ * doc2_
```

3.872983346207417

Implementation in Python

3.Finally:

```
cos_sim = dot_product / (doc1_ * doc2_)
print(cos_sim)
```

```
0.5163977794943222
```

Implementation in Python

3.Finally:

```
cos_sim = dot_product / (doc1_ * doc2_)
print(cos_sim)
```

```
0.5163977794943222
```

Implementation in Python

We can, however, do this much faster using sklearn's cosine_similarity.

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity([doc1_vector, doc2_vector])
```

```
array([[1.        , 0.51639778],
[0.51639778, 1.        ]])
```

Implementation in Python

We can, however, do this much faster using sklearn's cosine_similarity.

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity([doc1_vector, doc2_vector])
```

```
array([[1.        , 0.51639778],
[0.51639778, 1.        ]])
```

Considering Cosine Similarity

Things to consider

- What type of overlap are you interested in?
- What is the meaning of n-grams, stems, entities, stopwords when considering your RQ? How you should preproces, depends on your RQ and aim.
- Computationally cheap and fast; works well in e.g., recommender systems (week 6!)

Drawbacks

- An *exact* match in terms of words is needed. Is that realistic?

Considering Cosine Similarity

Things to consider

- What type of overlap are you interested in?
- What is the meaning of n-grams, stems, entities, stopwords when considering your RQ? How you should preproces, depends on your RQ and aim.
- Computationally cheap and fast; works well in e.g., recommender systems (week 6!)

Drawbacks

- An *exact* match in terms of words is needed. Is that realistic?

Considering Cosine Similarity

Things to consider

- What type of overlap are you interested in?
- What is the meaning of n-grams, stems, entities, stopwords when considering your RQ? How you should preproces, depends on your RQ and aim.
- Computationally cheap and fast; works well in e.g., recommender systems (week 6!)

Drawbacks

- An *exact* match in terms of words is needed. Is that realistic?

Considering Cosine Similarity

Things to consider

- What type of overlap are you interested in?
- What is the meaning of n-grams, stems, entities, stopwords when considering your RQ? How you should preproces, depends on your RQ and aim.
- Computationally cheap and fast; works well in e.g., recommender systems (week 6!)

Drawbacks

- An *exact* match in terms of words is needed. Is that realistic?

Implementation in Python

```python
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

What do you expect here? Should there be some level of overlap?

Implementation in Python

```python
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

```python
print(cosine_similarity([doc1_vector, doc2_vector, doc3_vector]))
```

```
[[1.         0.51639778 0.        ]
 [0.51639778 1.         0.        ]
 [0.         0.         1.        ]]
```

Zero overlap between doc3 and the other documents. Is that correct?

Implementation in Python

```python
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

```python
print(cosine_similarity([doc1_vector, doc2_vector, doc3_vector]))
```

```
[[1.         0.51639778 0.        ]
[0.51639778 1.         0.        ]
[0.         0.         1.        ]]
```

Zero overlap between doc3 and the other documents. Is that correct?

Implementation in Python

```
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

```
print(cosine_similarity([doc1_vector, doc2_vector, doc3_vector]))
```

```
[[1.         0.51639778 0.        ]
 [0.51639778 1.         0.        ]
 [0.         0.         1.        ]]
```

Zero overlap between doc3 and the other documents. Is that correct?

Soft cosine similarity

Soft cosine similarity

...enter soft cosine similarty (Sidorov et al., 2014)

"Soft Cosine Measure (SCM) is a method that allows us to assess the similarity between two documents in a meaningful way, even when they have no words in common. It uses a measure of similarity between words, which can be derived using [word2vec] vector embeddings of words." [3]

[3] https://radimrehurek.com/gensim//auto_examples/tutorials/run_scm.html

Soft cosine similarity

...enter soft cosine similarty (Sidorov et al., 2014)

"Soft Cosine Measure (SCM) is a method that allows us to assess the similarity between two documents in a meaningful way, even when they have no words in common. It uses a measure of similarity between words, which can be derived using [word2vec] vector embeddings of words." [3]

[3] https://radimrehurek.com/gensim//auto_examples/tutorials/run_scm.html

Soft Cosine Measure (SCM)

SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words 'play' and 'game' are different but related (Sidorov et al., 2014) [4]
- How can we capture 'semantic' meaning?

How?

- Convert words to *word vectors* and then compute similarities [5]

[4] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf
[5] https://www.machinelearningplus.com/nlp/cosine-similarity/

Soft Cosine Measure (SCM)

SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words ‘play’ and ‘game’ are different but related (Sidorov et al., 2014) [4]
- How can we capture ‘semantic’ meaning?

How?

- Convert words to *word vectors* and then compute similarities [5]

[4] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf

[5] https://www.machinelearningplus.com/nlp/cosine-similarity/

Soft Cosine Measure (SCM)

SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words ‘play’ and ‘game’ are different but related (Sidorov et al., 2014) [4]
- How can we capture ‘semantic’ meaning?

How?

- Convert words to *word vectors* and then compute similarities [5]

[4] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf
[5] https://www.machinelearningplus.com/nlp/cosine-similarity/

Soft Cosine Measure (SCM)

SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words 'play' and 'game' are different but related (Sidorov et al., 2014) [4]
- How can we capture 'semantic' meaning?

How?

- Convert words to *word vectors* and then compute similarities [5]

[4] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf

[5] https://www.machinelearningplus.com/nlp/cosine-similarity/

Soft Cosine Measure (SCM)

SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words 'play' and 'game' are different but related [6]
- How can we capture 'semantic' meaning?

How?

- Convert words to *word vectors* and then compute similarities

[6] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf

Soft Cosine Measure (SCM)

SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words 'play' and 'game' are different but related [6]
- How can we capture 'semantic' meaning?

How?

- Convert words to *word vectors* and then compute similarities

[6] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf

Soft Cosine Measure (SCM)

SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words ‘play’ and ‘game’ are different but related [6]
- How can we capture ‘semantic’ meaning?

How?

- Convert words to *word vectors* and then compute similarities

[6] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf

Soft Cosine Measure (SCM)

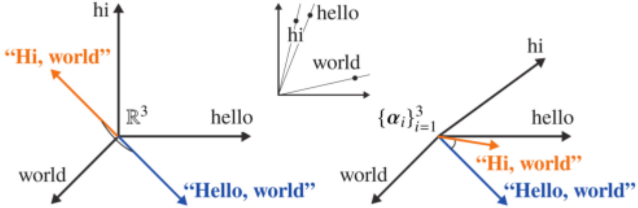
SCM

- Even if two sentences do not share the same words, we can calculate similarity by modelling *synonym*
- For example, the words ‘play’ and ‘game’ are different but related [6]
- How can we capture ‘semantic’ meaning?

How?

- Convert words to *word vectors* and then compute similarities

[6] http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf



Soft cosine similarity [7]

[7] https://radimrehurek.com/gensim//auto_examples/tutorials/run_scm.html

Word embeddings

Word embeddings

- To use the SCM, you need embeddings.
- Word embeddings

Word embeddings

Word embeddings

- To use the SCM, you need embeddings.
- Word embeddings

Soft cosine similarity

Word embeddings

Understanding SCM

SCM estimates extracts similarity from word embeddings.

what are word embeddings?

- Word embeddings help capturing the meaning of text
- Word embeddings are low-dimensional vector representations that capture semantic meaning
- State-of-the-art in NLP...
- *"...a word is characterized by the company it keeps..."* (Firth, 1957)

Understanding SCM

SCM estimates extracts similarity from word embeddings.

what are word embeddings?

- Word embeddings help capturing the meaning of text
- Word embeddings are low-dimensional vector representations that capture semantic meaning
- State-of-the-art in NLP...
- *"...a word is characterized by the company it keeps..."* (Firth, 1957)

Understanding SCM

SCM estimates extracts similarity from word embeddings.

what are word embeddings?

- Word embeddings help capturing the meaning of text
- Word embeddings are low-dimensional vector representations that capture semantic meaning
- State-of-the-art in NLP...
- *"...a word is characterized by the company it keeps..."* (Firth, 1957)

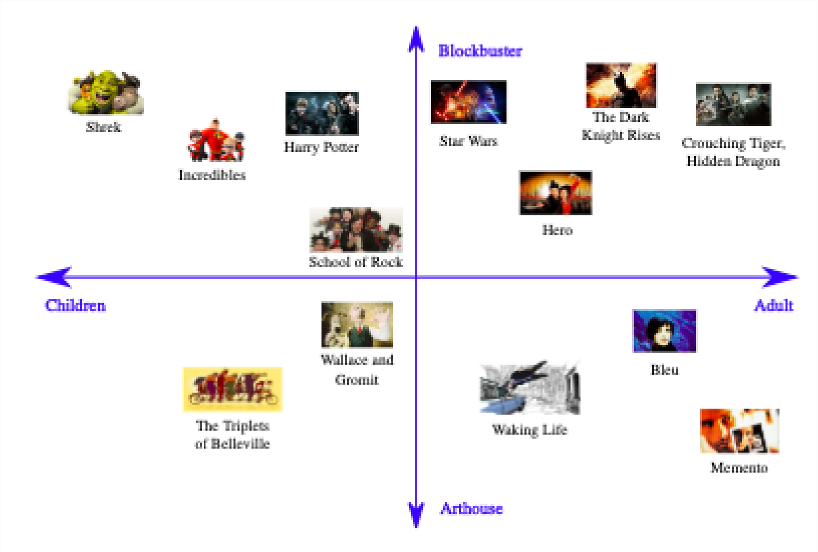
Understanding SCM

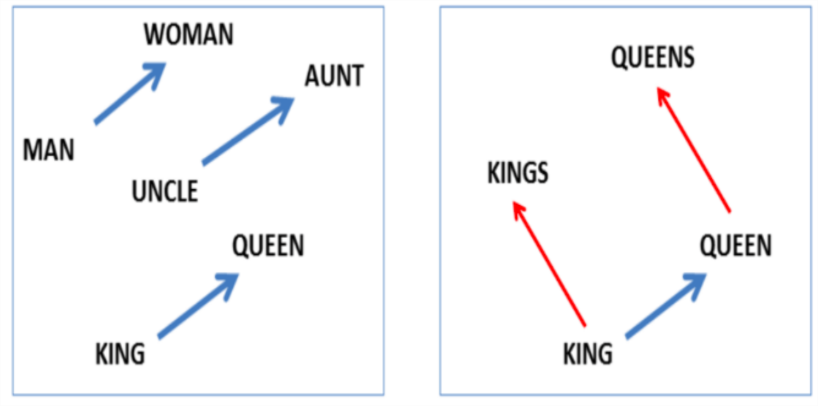
SCM estimates extracts similarity from word embeddings.

what are word embeddings?

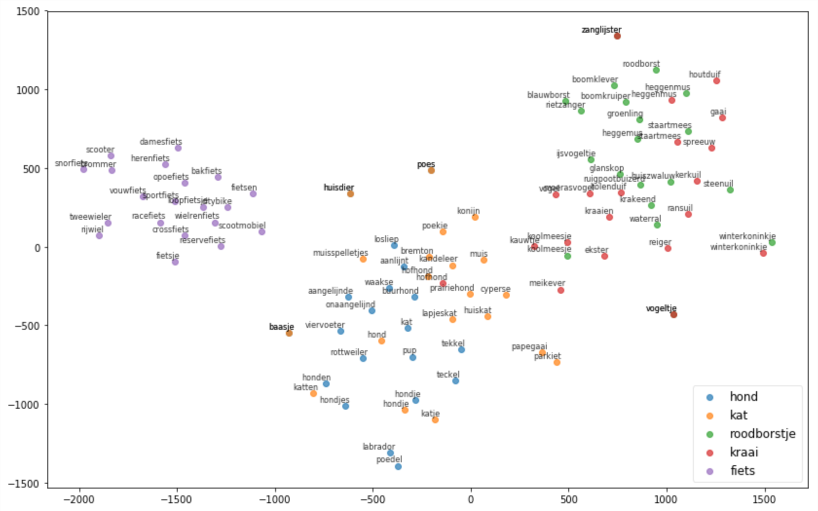
- Word embeddings help capturing the meaning of text
- Word embeddings are low-dimensional vector representations that capture semantic meaning
- State-of-the-art in NLP...
- *“...a word is characterized by the company it keeps...”* (Firth, 1957)

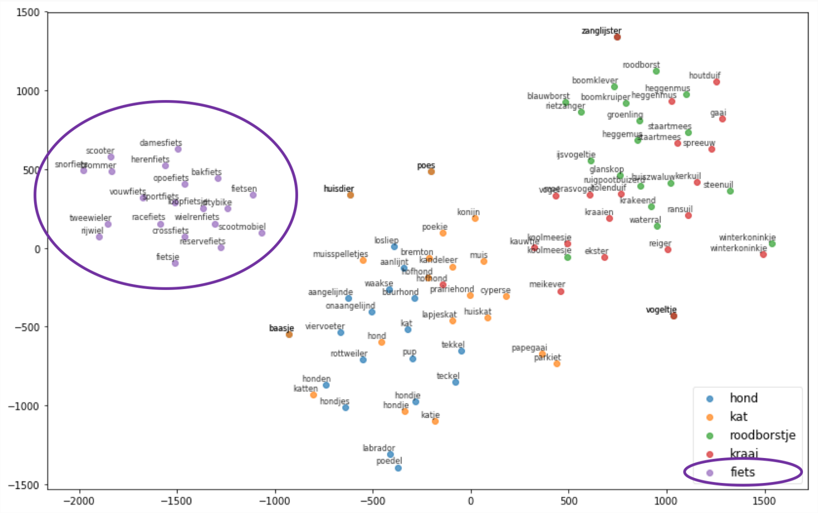

Shrek
Incredibles
The Triplets of Belleville
Harry Potter
Star Wars
Bleu
The Dark Knight Rises
Memento

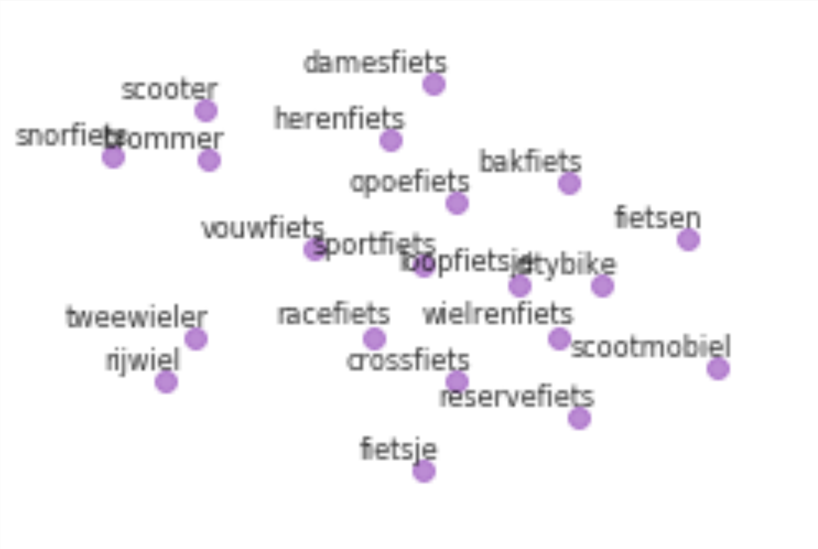

Blockbuster
Shrek
Incredibles
Harry Potter
Star Wars
The Dark Knight Rises
Crouching Tiger, Hidden Dragon
Hero
School of Rock
Children
Adult
Wallace and Gromit
Bleu
The Triplets of Belleville
Waking Life
Memento
Arthouse

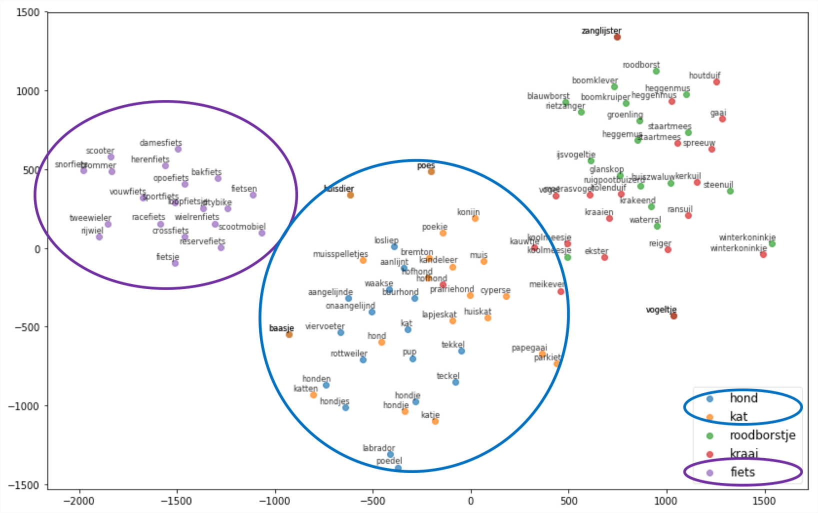


$king - man + woman = ?$

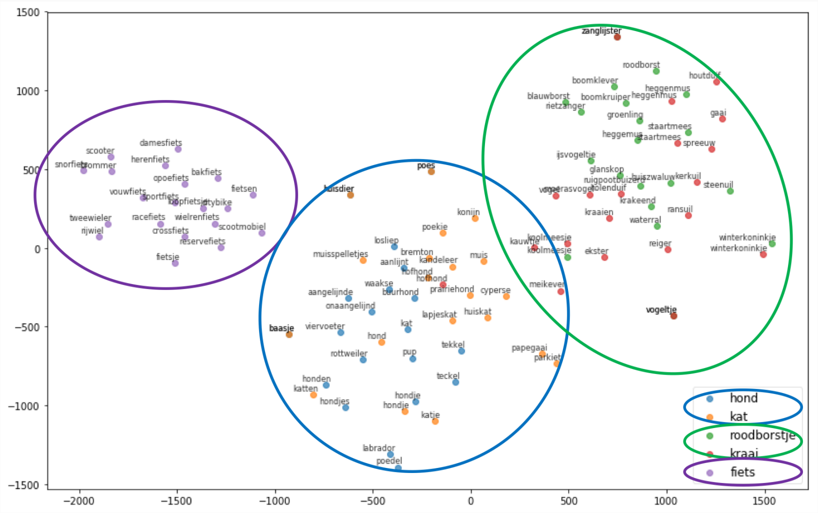

zanglijster
roodborst
houtduif
boomklever
heggenmus
blauwborst
boomkruiper
rietzanger
groenling
gaai
staartmees
heggemus
spreeuw
ijsvogeltje
damesfiets
scooter
herenfiets
snorfiets
brommer
bakfiets
poes
opoefiets
glanskop
ruigpootbuizerd
huiszwaluw
kerkuil
steenuil
fietsen
huisdier
vouwfiets
sportfiets
loopfiets
citybike
krakeend
kraaien
ransuil
konijn
tweewieler
racefiets
wielrenfiets
waterral
rijwiel
crossfiets
scootmobiel
poekje
reservefiets
koolmeesje
kauwtje
winterkoninkje
losliep
reiger
fietsje
muisspelletjes
bremton
ekster
muis
aanlijnt
kandeleer
hofhond
waakse
prairiehond
cyperse
meikever
aangelijnde
buurhond
onaangelijnd
huiskat
lapjeskat
vogeltje
kat
baasje
viervoeter
hond
tekkel
papegaai
parkiet
rottweiler
pup
teckel
honden
katten
hondje
hondjes
katje
labrador
poedel
hond
kat
roodborstje
kraai
fiets
1500
1000
500
0
−500
−1000
−1500
−2000
−1500
−1000
−500
0
500
1000
1500


1500
1000
500
0
−500
−1000
−1500
−2000
1500
zanglijster
roodborst
houtduif
boomklever
heggenmus
blauwborst
boomkruiper
rietzanger
groenling
gaai
staartmees
heggemus
spreeuw
ijsvogeltje
glanskop
poes
huisdier
steenuil
ransuil
konijn
kraaien
waterral
poekie
winterkoninkje
losliep
muisspelletjes
muis
bremton
aanlijnt
kandeleer
eekster
reiger
meikever
waakse
prairiehond
cyperse
aangelijnde
onaangelijnd
vogeltje
lapjeskat
huiskat
baasje
viervoeter
kat
hond
tekkel
papegaai
parkiet
rottweiler
pup
teckel
honden
katten
hondjes
hondje
katje
labrador
poedel
damesfiets
scooter
snorfiets
brommer
herenfiets
bakfiets
opoefiets
fietsen
vouwfiets
sportfiets
loopfiets
citybike
tweewieler
racefiets
wielrenfiets
scootmobiel
rijwiel
crossfiets
reservefiets
fietsje
hond
kat
roodborstje
kraai
fiets


damesfiets
scooter
herenfiets
bakfiets
opoefiets
fietsen
vouwfiets
sportfiets
tweewieler
racefiets
wielrenfiets
scootmobiel
rijwiel
crossfiets
reservefiets
fietsje



hond
kat
roodborstje
kraai
fiets
zanglijster
roodborst
houtduif
boomklever
heggenmus
blauwborst
boomkruiper
rietzanger
groenling
gaai
staartmees
heggemus
spreeuw
ijsvogeltje
glanskop
huiszwaluw
kerkuil
steenuil
krakeend
ransuil
kraaien
waterral
winterkoninkje
reiger
koolmeesje
kauwtje
ekster
meikever
vogeltje
damesfiets
scooter
snorfiets
brommer
herenfiets
bakfiets
opoefiets
fietsen
vouwfiets
sportfiets
citybike
tweewieler
racefiets
wielrenfiets
scootmobiel
rijwiel
crossfiets
reservefiets
fietsje
poes
huisdier
konijn
poekie
losliep
muisspelletjes
bremton
muis
aanlijnt
kandeleer
hofhond
waakse
buurhond
prairiehond
cyperse
aangelijnde
onaangelijnd
lapjeskat
huiskat
baasje
viervoeter
kat
hond
tekkel
papegaai
parkiet
rottweiler
pup
teckel
honden
katten
hondje
hondjes
katje
labrador
poedel
1500
1000
500
0
−500
−1000
−1500
−2000


1500
1000
500
0
−500
−1000
−1500
−2000
−1500
−1000
−500
0
500
1000
1500
hond
kat
roodborstje
kraai
fiets
damesfiets
scooter
snorfiets
brommer
herenfiets
bakfiets
opoefiets
fietsen
vouwfiets
sportfiets
loopfiets
citybike
tweewieler
racefiets
wielrenfiets
scootmobiel
rijwiel
crossfiets
reservefiets
fietsje
poes
huisdier
konijn
poekie
losliep
muisspelletjes
bremton
aanlijnt
kandeleer
muis
hofhond
waakse
buurhond
prairiehond
cyperse
aangelijnde
onaangelijnd
lapjeskat
huiskat
kat
baasje
viervoeter
hond
tekkel
papegaai
parkiet
rottweiler
pup
teckel
honden
katten
hondjes
hondje
katje
labrador
poedel
zanglijster
roodborst
houtduif
boomklever
heggenmus
blauwborst
boomkruiper
rietzanger
groenling
gaai
staartmees
heggemus
spreeuw
ijsvogeltje
glanskop
ruigpootbuizerd
huiszwaluw
kerkuil
steenuil
holenduif
krakeend
kraaien
ransuil
waterral
koolmeesje
kauwtje
winterkoninkje
reiger
ekster
meikever
vogeltje

Soft cosine similarity

Implemention in Python

SCM in Python

Calculating Soft Cosine Measure

- To use the SCM, you need embeddings.
- We *can* train embeddings on our own corpus (if we had a lot of data) ...
- But for now we will use pre-trained models [8]. ...

```
import gensim.downloader as api

fasttext_model300 = api.load('fasttext-wiki-news-subwords-300')
```

[8] https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb

SCM in Python

Calculating Soft Cosine Measure

- To use the SCM, you need embeddings.
- We *can* train embeddings on our own corpus (if we had a lot of data) . . .
- But for now we will use pre-trained models [8]. . . .

```
import gensim.downloader as api

fasttext_model300 = api.load('fasttext-wiki-news-subwords-300')
```

[8] https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb

SCM in Python

Calculating Soft Cosine Measure

- To use the SCM, you need embeddings.
- We *can* train embeddings on our own corpus (if we had a lot of data) . . .
- But for now we will use pre-trained models [8]. . . .

```
import gensim.downloader as api

fasttext_model300 = api.load('fasttext-wiki-news-subwords-300')
```

[8] https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb

SCM in Python

Calculating Soft Cosine Measure

- To use the SCM, you need embeddings.
- We *can* train embeddings on our own corpus (if we had a lot of data) ...
- But for now we will use pre-trained models [8]. ...

```
import gensim.downloader as api

fasttext_model300 = api.load('fasttext-wiki-news-subwords-300')
```

[8] https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb

Create a dictionary

Let's review our 3 documents:

```
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

Initialize a Dictionary. This step assigns a token_id to each word:

```
from gensim.utils import simple_preprocess
from gensim.corpora import Dictionary
dictionary = corpora.Dictionary([simple_preprocess(doc) for doc in
↪ [doc1, doc2, doc3]])
```

Now, let's check whether a specific word–for example coffee–is in our dictionary:

```
'coffee' in dictionary.token2id
```

True

Create a dictionary

Let's review our 3 documents:

```
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

Initialize a Dictionary. This step assigns a `token_id` to each word:

```
from gensim.utils import simple_preprocess
from gensim.corpora import Dictionary
dictionary = corpora.Dictionary([simple_preprocess(doc) for doc in
↪  [doc1, doc2, doc3]])
```

Now, let's check whether a specific word–for example `coffee`–is in our `dictionary`:

```
'coffee' in dictionary.token2id
```

```
True
```

Create a dictionary

Let's review our 3 documents:

```
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

Initialize a Dictionary. This step assigns a `token_id` to each word:

```
from gensim.utils import simple_preprocess
from gensim.corpora import Dictionary
dictionary = corpora.Dictionary([simple_preprocess(doc) for doc in
↪  [doc1, doc2, doc3]])
```

Now, let's check whether a specific word–for example `coffee`–is in our `dictionary`:

```
'coffee' in dictionary.token2id
```

```
True
```

Create a dictionary

Let's review our 3 documents:

```
doc1 = "When I eat breakfast, I usually drink some tea".lower()
doc2 = "I like my tea with my breakfast".lower()
doc3 = "She likes cereal and coffee".lower()
```

Initialize a Dictionary. This step assigns a `token_id` to each word:

```
from gensim.utils import simple_preprocess
from gensim.corpora import Dictionary
dictionary = corpora.Dictionary([simple_preprocess(doc) for doc in
↪  [doc1, doc2, doc3]])
```

Now, let's check whether a specific word–for example `coffee`–is in our `dictionary`:

```
'coffee' in dictionary.token2id
```



```
True
```

Create a bag-of-words representation

Next, let's represent each document by `(token_id, token_count)` tuples:

```
bag_of_words_vectors = [ dictionary.doc2bow(simple_preprocess(doc)) for doc in [doc1, doc2, doc3]]
```

Build a term similarity matrix and compute a sparse term similarity matrix:

```
from gensim.similarities import SparseTermSimilarityMatrix
from gensim.similarities import WordEmbeddingSimilarityIndex

similarity_index = WordEmbeddingSimilarityIndex(fasttext_model300)
similarity_matrix = SparseTermSimilarityMatrix(similarity_index, dictionary)
```

Create a bag-of-words representation

Next, let's represent each document by (token_id, token_count) tuples:

```
bag_of_words_vectors = [ dictionary.doc2bow(simple_preprocess(doc))
↪  for doc in [doc1, doc2, doc3]]
```

Build a term similarity matrix and compute a sparse term similarity matrix:

```
from gensim.similarities import SparseTermSimilarityMatrix
from gensim.similarities import WordEmbeddingSimilarityIndex

similarity_index = WordEmbeddingSimilarityIndex(fasttext_model300)
similarity_matrix = SparseTermSimilarityMatrix(similarity_index,
↪  dictionary)
```

Inspect results

Get SCM using `.inner_product()`: :

```
#between doc1 and doc2:
scm_doc1_doc2 = similarity_matrix.inner_product(bag_of_words_vectors[0], bag_of_words_vectors[1], normalized=(True, True))

#between doc1 and doc3:
scm_doc1_doc3 = similarity_matrix.inner_product(bag_of_words_vectors[0], bag_of_words_vectors[2], normalized=(True, True))

#between doc2 and doc3;
scm_doc2_doc3 = similarity_matrix.inner_product(bag_of_words_vectors[1], bag_of_words_vectors[2], normalized=(True, True))

print(f"SCM between:\ndoc1 <-> doc2: {scm_doc1_doc2:.2f}\ndoc1 <-> doc3: {scm_doc1_doc3:.2f}\ndoc2 <-> doc3: {scm_doc2_doc3:.2f}")
```

Do the results make more sense?:

```
SCM between:
doc1 <-> doc2: 0.29
doc1 <-> doc3: 0.15
doc2 <-> doc3: 0.28
```

Inspect results

Get SCM using `.inner_product():` :

```
#between doc1 and doc2:
scm_doc1_doc2 = similarity_matrix.inner_product(bag_of_words_vectors[0],
↪   bag_of_words_vectors[1], normalized=(True, True))

#between doc1 and doc3:
scm_doc1_doc3 = similarity_matrix.inner_product(bag_of_words_vectors[0],
↪   bag_of_words_vectors[2], normalized=(True, True))

#between doc2 and doc3;
scm_doc2_doc3 = similarity_matrix.inner_product(bag_of_words_vectors[1],
↪   bag_of_words_vectors[2], normalized=(True, True))

print(f"SCM between:\ndoc1 <-> doc2: {scm_doc1_doc2:.2f}\ndoc1 <-> doc3:
↪   {scm_doc1_doc3:.2f}\ndoc2 <-> doc3: {scm_doc2_doc3:.2f}")
```

Do the results make more sense?:

```
SCM between:
doc1 <-> doc2: 0.29
doc1 <-> doc3: 0.15
doc2 <-> doc3: 0.28
```

Applications of cosine and soft cosine similarity

Applications of *cosine* and *soft cosine* in the field of Communication Science

Trace convergence or agenda setting dynamics over time

- For example, to map *linguistic alignment* of romantic couples over time (Brinberg and Ram, 2021)
- Or, in the political domain, agenda overlap between public opinion and political speech (Hager and Hilbig, 2020)

You will also look into overtime dynamics during this week's lab session!

Applications of cosine and soft cosine similarity

Applications of *cosine* and *soft cosine* in the field of Communication Science

Trace convergence or agenda setting dynamics over time

- For example, to map *linguistic alignment* of romantic couples over time (Brinberg and Ram, 2021)
- Or, in the political domain, agenda overlap between public opinion and political speech (Hager and Hilbig, 2020)

You will also look into overtime dynamics during this week's lab session!

Topic modelling

Let's assume you want to know a bit more about the *content* you are investigating

Cosine and soft cosine do *not* inform us about substantive issues present in text.

1. Which topics can we extract from the corpus?
2. How present is each of these topics in each text in the corpus?

Recap: Document-term matrix

Document-term matrix

```
     w1,w2,w3,w4,w5,w6 ...
text1, 2, 0, 0, 1, 2, 3 ...
text2, 0, 0, 1, 2, 3, 4 ...
text3, 9, 0, 1, 1, 0, 0 ...
...
```

These can be simple counts, but also more advanced metrics, like tf-idf scores (where you weigh the frequency by the number of documents in which it occurs), cosine distances, etc.

- given a term-document matrix, easy to do with any tool
- probably extremely skewed distributions

Recap: Document-term matrix

Document-term matrix

```
     w1,w2,w3,w4,w5,w6 ...
text1, 2, 0, 0, 1, 2, 3 ...
text2, 0, 0, 1, 2, 3, 4 ...
text3, 9, 0, 1, 1, 0, 0 ...
...
```

These can be simple counts, but also more advanced metrics, like tf-idf scores (where you weigh the frequency by the number of documents in which it occurs), cosine distances, etc.

- given a term-document matrix, easy to do with any tool
- probably extremely skewed distributions

Recap: clustering

- given a term-document matrix, we can easily find clusters of documents that resemble each other

We need other models to

1. model *simultaneously* (a) which topics we find in the whole corpus, and (b) which of these topics are present in which document; while at the same time
2. allowing (a) words to be part of multiple topics, and (b) multiple topics to be present in one document; and
3. being able to make connections between words "even if they never actually occured in a document together" (Maier et al., 2018)[p. 96]

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., . . . Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures, 12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

We need other models to

1. model *simultaneously* (a) which topics we find in the whole corpus, and (b) which of these topics are present in which document; while at the same time
2. allowing (a) words to be part of multiple topics, and (b) multiple topics to be present in one document; and
3. being able to make connections between words "even if they never actually occured in a document together" (Maier et al., 2018)[p. 96]

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., . . . Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures*, *12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

We need other models to

1. model *simultaneously* (a) which topics we find in the whole corpus, and (b) which of these topics are present in which document; while at the same time
2. allowing (a) words to be part of multiple topics, and (b) multiple topics to be present in one document; and
3. being able to make connections between words "even if they never actually occured in a document together" (Maier et al., 2018)[p. 96]

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., . . . Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures, 12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

Topic modelling

An introduction to LDA

Enter **topic modeling with Latent Dirichlet Allocation (LDA)**

LDA, what's that?

No mathematical details here, but the general idea

- There are k topics, $T_1 \dots T_k$
- Each document D_i consists of a mixture of these topics, e.g.$80\% T_1, 15\% T_2, 0\% T_3, \dots 5\% T_k$
- On the next level, each topic consists of a specific probability distribution of words
- Thus, based on the frequencies of words in D_i, one can infer its distribution of topics
- Note that LDA (like PCA) is a Bag-of-Words (BOW) approach

Doing a LDA in Python

We will use gensim (Rehurek and Sojka, 2010) for this (make sure you have version >4.0)

Let us assume you have a list of lists of documents called `texts`:

```
print(texts[0][:115])
```

which looks something like:

```
'Stop the presses: CNN covered some actual news yesterday when it reported on the story of
↪  medical kidnapping victim Alyssa Gilderhus at the Mayo Clinic. But was it actually InfoWars
↪  and FreeMartyG which publicly shamed CNN into doing this real journalism? Cue the Mission
↪  Impossible theme music for this one...\n\nThis mission, as we accepted it, began more than a
↪  year ago during the baby Charlie Gard medical kidnapping scandal in the UK and we thought
↪  that it had ended with an apparently unsuccessf'
```

Řehůřek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50. Valletta, Malta: ELRA.

Doing a LDA in Python

We will use gensim (Rehurek and Sojka, 2010) for this (make sure you have version >4.0)

Let us assume you have a list of lists of documents called `texts`:

```
print(texts[0][:115])
```

which looks something like:

```
'Stop the presses: CNN covered some actual news yesterday when it reported on the story of
↪ medical kidnapping victim Alyssa Gilderhus at the Mayo Clinic. But was it actually InfoWars
↪ and FreeMartyG which publicly shamed CNN into doing this real journalism? Cue the Mission
↪ Impossible theme music for this one...\n\nThis mission, as we accepted it, began more than a
↪ year ago during the baby Charlie Gard medical kidnapping scandal in the UK and we thought
↪ that it had ended with an apparently unsuccessf'
```

Řehůřek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50. Valletta, Malta: ELRA.

Preprocessing

Your preprocessing steps and feature engineering decisions *largely* affect your topics

1. You can apply 'manual' preprocessing steps ...
2. ...In isolation or combination with for example `tfidf` transformations

```
texts_clean = [text.lower() for text in texts]
texts_clean=[" ".join(text.split()) for text in texts_clean]  #remove dubble spaces
texts_clean = ["".join([l for l in text if l not in punctuation]) for text in texts_clean]
↪  #remove punctuaction
texts_clean[0][:500]
```

which looks something like:

```
'stop the presses cnn covered some actual news yesterday when it reported on the story of
↪  medical kidnapping victim alyssa gilderhus at the mayo clinic but was it actually infowars
↪  and freemartyg which publicly shamed cnn into doing this real journalism cue the mission
↪  impossible theme music for this one this mission as we accepted it began more than a year
↪  ago during the baby charlie gard medical kidnapping scandal in the uk and we thought that it
↪  had ended with an apparently unsuccessful april '
```

Preprocessing

Your preprocessing steps and feature engineering decisions *largely* affect your topics

1. You can apply 'manual' preprocessing steps ...
2. ... In isolation or combination with for example `tfidf` transformations

```
texts_clean = [text.lower() for text in texts]
texts_clean=[" ".join(text.split()) for text in texts_clean]  #remove dubble spaces
texts_clean = ["".join([l for l in text if l not in punctuation]) for text in texts_clean]  #remove punctuaction
texts_clean[0][:500]
```

which looks something like:

```
'stop the presses cnn covered some actual news yesterday when it reported on the story of medical kidnapping victim alyssa gilderhus at the mayo clinic but was it actually infowars and freemartyg which publicly shamed cnn into doing this real journalism cue the mission impossible theme music for this one this mission as we accepted it began more than a year ago during the baby charlie gard medical kidnapping scandal in the uk and we thought that it had ended with an apparently unsuccessful april '
```

Preprocessing

Your preprocessing steps and feature engineering decisions *largely* affect your topics

1. You can apply 'manual' preprocessing steps ...
2. ... In isolation or combination with for example `tfidf` transformations

```
texts_clean = [text.lower() for text in texts]
texts_clean=[" ".join(text.split()) for text in texts_clean]  #remove dubble spaces
texts_clean = ["".join([l for l in text if l not in punctuation]) for text in texts_clean]  #remove punctuaction
texts_clean[0][:500]
```

which looks something like:

```
'stop the presses cnn covered some actual news yesterday when it reported on the story of medical kidnapping victim alyssa gilderhus at the mayo clinic but was it actually infowars and freemartyg which publicly shamed cnn into doing this real journalism cue the mission impossible theme music for this one this mission as we accepted it began more than a year ago during the baby charlie gard medical kidnapping scandal in the uk and we thought that it had ended with an apparently unsuccessful april '
```

Preprocessing

Your preprocessing steps and feature engineering decisions *largely* affect your topics

1. You can apply 'manual' preprocessing steps ...
2. ... In isolation or combination with for example `tfidf` transformations

```
texts_clean = [text.lower() for text in texts]
texts_clean=[" ".join(text.split()) for text in texts_clean]  #remove dubble spaces
texts_clean = ["".join([l for l in text if l not in punctuation]) for text in texts_clean]  #remove punctuaction
texts_clean[0][:500]
```

which looks something like:

```
'stop the presses cnn covered some actual news yesterday when it reported on the story of medical kidnapping victim alyssa gilderhus at the mayo clinic but was it actually infowars and freemartyg which publicly shamed cnn into doing this real journalism cue the mission impossible theme music for this one this mission as we accepted it began more than a year ago during the baby charlie gard medical kidnapping scandal in the uk and we thought that it had ended with an apparently unsuccessful april '
```

Preprocessing

Without *stopword removal*, *tfidf* transformation and/or *pruning*, you topics will not be very informative.

```
mystopwords = set(stopwords.words('english')) # use default NLTK stopword list;
↪  alternatively:
# mystopwords = set(open('mystopwordfile.txt').readlines())  #read stopword list from a
↪  textfile with one stopword per line
texts_clean = [" ".join(word for word in text.split() if word not in mystopwords) for text in
↪  texts_clean]
texts_clean[0][:500]
```

which looks something like:

```
'stop presses cnn covered actual news yesterday reported story medical kidnapping victim alyssa
↪  gilderhus mayo clinic actually infowars freemartyg publicly shamed cnn real journalism cue
↪  mission impossible theme music one mission accepted began year ago baby charlie gard medical
↪  kidnapping scandal uk thought ended apparently unsuccessful april fools joke cnn sure many
↪  recall charlie gard infant rare form otherwise notsorare condition mitochondrial disease
↪  story went viral made international news '
```

Preprocessing

Without *stopword removal*, *tfidf* transformation and/or *pruning*, you topics will not be very informative.

```
mystopwords = set(stopwords.words('english')) # use default NLTK stopword list; alternatively:
# mystopwords = set(open('mystopwordfile.txt').readlines())  #read stopword list from a textfile with one stopword per line
texts_clean = [" ".join(word for word in text.split() if word not in mystopwords) for text in texts_clean]
texts_clean[0][:500]
```

which looks something like:

```
'stop presses cnn covered actual news yesterday reported story medical kidnapping victim alyssa gilderhus mayo clinic actually infowars freemartyg publicly shamed cnn real journalism cue mission impossible theme music one mission accepted began year ago baby charlie gard medical kidnapping scandal uk thought ended apparently unsuccessful april fools joke cnn sure many recall charlie gard infant rare form otherwise notsorare condition mitochondrial disease story went viral made international news '
```

Preprocessing

Without *stopword removal*, *tfidf* transformation and/or *pruning*, you topics will not be very informative.

```
mystopwords = set(stopwords.words('english')) # use default NLTK stopword list; alternatively:
# mystopwords = set(open('mystopwordfile.txt').readlines())  #read stopword list from a textfile with one stopword per line
texts_clean = [" ".join(word for word in text.split() if word not in mystopwords) for text in texts_clean]
texts_clean[0][:500]
```

which looks something like:

```
'stop presses cnn covered actual news yesterday reported story medical kidnapping victim alyssa gilderhus mayo clinic actually infowars freemartyg publicly shamed cnn real journalism cue mission impossible theme music one mission accepted began year ago baby charlie gard medical kidnapping scandal uk thought ended apparently unsuccessful april fools joke cnn sure many recall charlie gard infant rare form otherwise notsorare condition mitochondrial disease story went viral made international news '
```

Tokenization

`gensim` expects a list of words (hence: `tokenize` your corpus)

```
tokenized_texts_clean = [TreebankWordTokenizer().tokenize(text) for text in texts_clean ] #
↪  tokenize texts; convert all strings to a list of tokens
tokenized_texts_clean[0][:500]
```

which looks something like:

```
['stop',
'presses',
'cnn',
'covered',
'actual',
'news',
'yesterday',
'reported',
'story',
..
```

Tokenization

gensim expects a list of words (hence: `tokenize` your corpus)

```
tokenized_texts_clean = [TreebankWordTokenizer().tokenize(text) for text in texts_clean ] #
↪  tokenize texts; convert all strings to a list of tokens
tokenized_texts_clean[0][:500]
```

which looks something like:

```
['stop',
'presses',
'cnn',
'covered',
'actual',
'news',
'yesterday',
'reported',
'story',
..
```

Tokenization

gensim expects a list of words (hence: tokenize your corpus)

```
tokenized_texts_clean = [TreebankWordTokenizer().tokenize(text) for text in texts_clean ] # tokenize texts; convert all strings to a list of tokens
tokenized_texts_clean[0][:500]
```

which looks something like:

```
['stop',
'presses',
'cnn',
'covered',
'actual',
'news',
'yesterday',
'reported',
'story',
..
```

LDA implementation

LDA implementation

```
raw_m1 = tokenized_texts_clean

# assign a token_id to each word
id2word_m1 = corpora.Dictionary(raw_m1)
# represent each text by (token_id, token_count) tuples
ldacorpus_m1 = [id2word_m1.doc2bow(text) for text in raw_m1]

#estimate the model
lda_m1 = models.LdaModel(ldacorpus_m1, id2word=id2word_m1, num_topics=10)
lda_m1.print_topics()
```

```
[(0,
'0.015*"trump" + 0.012*"said" + 0.006*"president" + 0.006*"people" + 0.004*"cnn" + 0.004*"us" +
↪  0.004*"house" + 0.004*"news" + 0.003*"also" + 0.003*"twitter"'),
(1,
'0.010*"said" + 0.008*"trump" + 0.004*"one" + 0.004*"people" + 0.004*"us" + 0.004*"president" +
↪  0.004*"would" + 0.003*"media" + 0.003*"also" + 0.003*"new"'),
(2,
'0.011*"trump" + 0.009*"said" + 0.007*"president" + 0.005*"would" + 0.004*"people" + 0.004*"us"
↪  + 0.003*"also" + 0.003*"like" + 0.003*"news" + 0.003*"state"'),
(3,
'0.010*"trump" + 0.006*"president" + 0.005*"said" + 0.004*"would" + 0.004*"us" + 0.003*"also" +
↪  0.003*"people" + 0.003*"media" + 0.003*"news" + 0.003*"one"'),
```

LDA implementation

LDA implementation

```
raw_m1 = tokenized_texts_clean

# assign a token_id to each word
id2word_m1 = corpora.Dictionary(raw_m1)
# represent each text by (token_id, token_count) tuples
ldacorpus_m1 = [id2word_m1.doc2bow(text) for text in raw_m1]

#estimate the model
lda_m1 = models.LdaModel(ldacorpus_m1, id2word=id2word_m1, num_topics=10)
lda_m1.print_topics()
```

```
[(0,
'0.015*"trump" + 0.012*"said" + 0.006*"president" + 0.006*"people" + 0.004*"cnn" + 0.004*"us" +
↪  0.004*"house" + 0.004*"news" + 0.003*"also" + 0.003*"twitter"'),
(1,
'0.010*"said" + 0.008*"trump" + 0.004*"one" + 0.004*"people" + 0.004*"us" + 0.004*"president" +
↪  0.004*"would" + 0.003*"media" + 0.003*"also" + 0.003*"new"'),
(2,
'0.011*"trump" + 0.009*"said" + 0.007*"president" + 0.005*"would" + 0.004*"people" + 0.004*"us"
↪  + 0.003*"also" + 0.003*"like" + 0.003*"news" + 0.003*"state"'),
(3,
'0.010*"trump" + 0.006*"president" + 0.005*"said" + 0.004*"would" + 0.004*"us" + 0.003*"also" +
↪  0.003*"people" + 0.003*"media" + 0.003*"news" + 0.003*"one"'),
```

LDA implementation

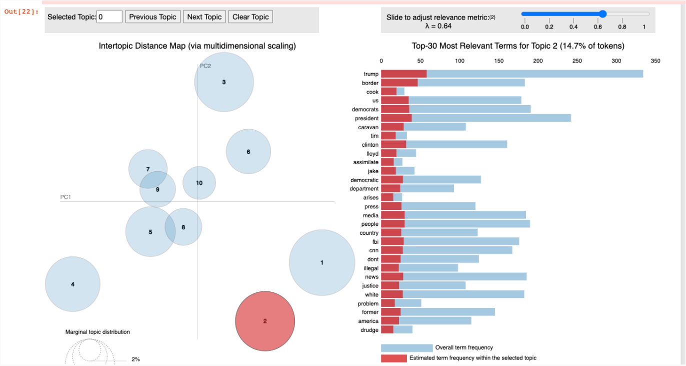
LDA implementation

```
raw_m1 = tokenized_texts_clean

# assign a token_id to each word
id2word_m1 = corpora.Dictionary(raw_m1)
# represent each text by (token_id, token_count) tuples
ldacorpus_m1 = [id2word_m1.doc2bow(text) for text in raw_m1]

#estimate the model
lda_m1 = models.LdaModel(ldacorpus_m1, id2word=id2word_m1, num_topics=10)
lda_m1.print_topics()
```

```
[(0,
'0.015*"trump" + 0.012*"said" + 0.006*"president" + 0.006*"people" + 0.004*"cnn" + 0.004*"us" +
↪  0.004*"house" + 0.004*"news" + 0.003*"also" + 0.003*"twitter"'),
(1,
'0.010*"said" + 0.008*"trump" + 0.004*"one" + 0.004*"people" + 0.004*"us" + 0.004*"president" +
↪  0.004*"would" + 0.003*"media" + 0.003*"also" + 0.003*"new"'),
(2,
'0.011*"trump" + 0.009*"said" + 0.007*"president" + 0.005*"would" + 0.004*"people" + 0.004*"us"
↪  + 0.003*"also" + 0.003*"like" + 0.003*"news" + 0.003*"state"'),
(3,
'0.010*"trump" + 0.006*"president" + 0.005*"said" + 0.004*"would" + 0.004*"us" + 0.003*"also" +
↪  0.003*"people" + 0.003*"media" + 0.003*"news" + 0.003*"one"'),
```

Visualization with pyldavis

```python
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
# first estiate gensim model, then:
vis_data = gensimvis.prepare(lda_m1,ldacorpus_m1,id2word_m1)
pyLDAvis.display(vis_data)
```



Visualization with pyldavis

Short note about the λ setting:

It influences the ordering of the words in pyldavis.

> *"For $\lambda = 1$, the ordering of the top words is equal to the ordering of the standard conditional word probabilities. For λ close to zero, the most specific words of the topic will lead the list of top words. In their case study, Sievert and Shirley (2014, p. 67) found the best interpretability of topics using a λ-value close to .6, which we adopted for our own case" (Maier et al., 2018, p. 107)*

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., . . . Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures*, *12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

Code examples

https://github.com/annekroon/gesis-machine-learning/blob/main/day3/excercise-afternoon/lda.ipynb

Topic modelling

Choosing the best (or a good) topic model

Choosing the best (or a good) topic model

- There is no single best solution (e.g., do you want more coarse of fine-grained topics?)
- Non-deterministic
- Very sensitive to preprocessing choices
- Interplay of both metrics and (qualitative) interpretability

See for more elaborate guidance:

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., . . . Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures, 12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

Evaluation metrics (closer to zero is better)

perplexity

A goodness-of-fit measure, answering the question: If we do a train-test split, how well does the trained model fit the test data?

coherence

- mean coherence of the whole model: attempts to quantify the interpretability
- coherence per topic: allows to get topics that are most likely to be coherently interpreted (`.top_topics()`)

Evaluation metrics (closer to zero is better)

perplexity

A goodness-of-fit measure, answering the question: If we do a train-test split, how well does the trained model fit the test data?

coherence

- mean coherence of the whole model: attempts to quantify the interpretability
- coherence per topic: allows to get topics that are most likely to be coherently interpreted (`.top_topics()`)

So, how do we do this?

- Basically, similar to the idea behind our grid search from two weeks ago: estimate multiple models, store the metrics for each model, and then compare them (numerically, or by plotting)
- Idea: We select some candidate models, and then look whether they can be interpreted.
- But what can we tune?

So, how do we do this?

- Basically, similar to the idea behind our grid search from two weeks ago: estimate multiple models, store the metrics for each model, and then compare them (numerically, or by plotting)
- Idea: We select some candidate models, and then look whether they can be interpreted.
- But what can we tune?

So, how do we do this?

- Basically, similar to the idea behind our grid search from two weeks ago: estimate multiple models, store the metrics for each model, and then compare them (numerically, or by plotting)
- Idea: We select some candidate models, and then look whether they can be interpreted.
- But what can we tune?

Choosing k: How many topics do we want?

- Typical values: $10 < k < 200$
- Too low: losing nuance, so broad it becomes meaningless
- Too high: picks up tiny pecularities instead of finding general patterns
- There is no inherent ordering of topics (unlike PCA!)
- We can throw away or merge topics later, so if out of $k = 50$ topics 5 are not interpretable and a couple of others overlap, it still may be a good model

Choosing α: how sparse should the document-topic distribution θ be?

- The higher α, the more topics per document
- Default: $1/k$
- But: We can explicitly change it, or – really cool – even learn α from the data (`alpha = "auto"`)

Takeaway: It takes longer, but you probably want to learn alpha from the data, using multiple passes:

```
mylda LdaModel(corpus=tfidfcorpus[ldacorpus], id2word=
    id2word, num_topics=50, alpha='auto', passes=10)
```

Choosing α: how sparse should the document-topic distribution θ be?

- The higher α, the more topics per document
- Default: $1/k$
- But: We can explicitly change it, or – really cool – even learn α from the data (`alpha = "auto"`)

Takeaway: It takes longer, but you probably want to learn alpha from the data, using multiple passes:

```
            mylda LdaModel(corpus=tfidfcorpus[ldacorpus], id2word=
                id2word, num_topics=50, alpha='auto', passes=10)
```

Choosing η: how sparse should the topic-word distribution λ be?

- Can be used to boost specific words
- Can also be learned from the data

Takeaway: Even though you can do `eta="auto"`, this usually does not help you much.

Choosing η: how sparse should the topic-word distribution λ be?

- Can be used to boost specific words
- Can also be learned from the data

Takeaway: Even though you can do `eta="auto"`, this usually does not help you much.

Topic modelling

Using topic models

Using topic models

You got your model – what now?

1. Assign topic scores to documents
2. Label topics
3. Merge topics, throw away boilerplate topics and similar (manually, or aided by cluster analysis)
4. Compare topics between, e.g., outlets
5. or do some time-series analysis.

Example: Tsur, O., Calacci, D., & Lazer, D. (2015). A Frame of Mind: Using Statistical Models for Detection of Framing and Agenda Setting Campaigns. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (pp. 1629–1638).

Topic modelling

Other forms of topic models

Other forms of topic models

- Author-topic models
- Structural topic models
- Non-negative matrix factorization
- . . .

Next steps

Exercise for this afternoon

- Work through the example notebook on LDA:
  https://github.com/annekroon/gesis-machine-learning/blob/main/day3/excercise-afternoon/lda.ipynb
- But most importantly: **Use a dataset of your choice** and find a suitable topic model. You can also try to compare multiple approaches (e.g., clustering vs LDA). Possible inspiration:
  https://github.com/annekroon/gesis-ml-learning/blob/master/03wednesday/livecoding.ipynb

Good luck!

References i

References

Brinberg, Miriam and Nilam Ram (2021). "Do new romantic couples use more similar language over time? Evidence from intensive longitudinal text messages." In: *Journal of Communication* 71.3, pp. 454–477. ISSN: 0021-9916. DOI: 10.1093/joc/jqab012.

Hager, Anselm and Hanno Hilbig (2020). "Does public opinion affect political speech?" In: *American Journal of Political Science* 64.4, pp. 921–937. ISSN: 15405907. DOI: 10.1111/ajps.12516.

Maier, Daniel et al. (2018). "Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology." In: *Communication Methods and Measures* 12.2-3, pp. 93–118. ISSN: 19312466. DOI: 10.1080/19312458.2018.1430754. URL: https://doi.org/10.1080/19312458.2018.1430754.

Rehurek, Radim and Petr Sojka (2010). "Software Framework for Topic Modelling with Large Corpora." In: *IN PROCEEDINGS OF THE LREC 2010 WORKSHOP ON NEW CHALLENGES FOR NLP FRAMEWORKS*, pp. 45–50.

References ii

Sidorov, Grigori et al. (2014). "Soft similarity and soft cosine measure: Similarity of features in vector space model." In: *Computacion y Sistemas* 18.3, pp. 491–504. ISSN: 20079737. DOI: 10.13053/CyS-18-3-2043.