

# Computational Communication Science 2

## Week 4 - Lecture

### »Text similiarity and differences«

---

Anne Kroon

a.c.kroon@uva.nl, @annekroon

April, 2022

Digital Society Minor, University of Amsterdam

# Today

Cosine Similarity

Soft cosine similarity

Word embeddings

Implementation in Python

Topic modelling

An introduction to LDA

Choosing the best (or a good) topic model

Using topic models

Next steps

# The CCS toolbox

|  | Methodological approach  |  |  |
|--|--|--|--|
|  | <i>Counting and Dictionary</i>                                     | <i>Supervised Machine Learning</i>     | <i>Unsupervised Machine Learning</i>   |
| <b>Typical research interests and content features</b> | visibility analysis<br>sentiment analysis<br>subjectivity analysis | frames<br>topics<br>gender bias        | frames<br>topics   |
| <b>Common statistical procedures</b>                   | string comparisons<br>counting                                     | support vector machines<br>naive Bayes | principal component analysis<br>cluster analysis<br>latent dirichlet allocation<br>semantic network analysis |

deductive

inductive

Boumans and Trilling, 2016

# Cosine Similarity

---

# Cosine Similarity

## Cosine Similarity

- A measure that helps you determine how similar two documents are, irrespective of their size

## Applications in Communication Science

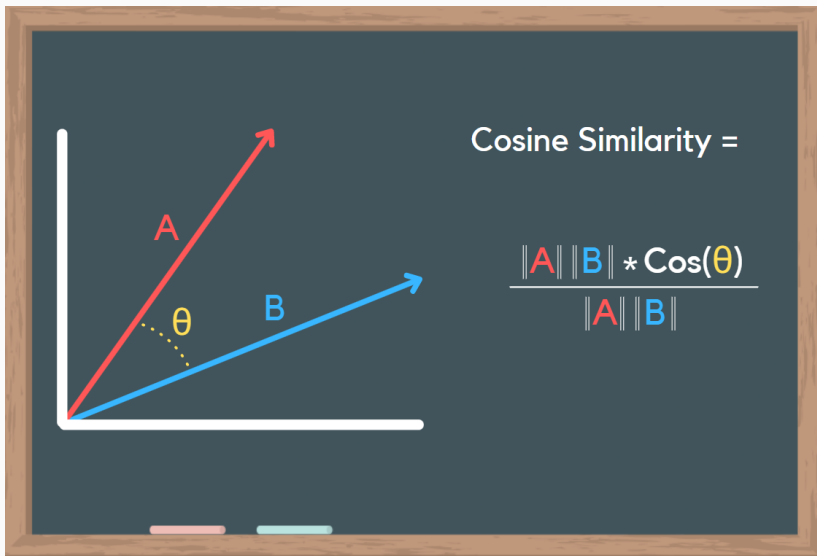
- For example, to map *linguistic alignment* of romantic couples over time (Brinberg and Ram, 2021)
- Or, in the political domain, agenda overlap between public opinion and political speech (Hager and Hilbig, 2020b)

# Cosine Similarity

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

- It measures the cosine of the angle between two vectors projected in a multi-dimensional space.
- 0 means orthogonal vectors (90 degrees); very dissimilar
- 1 means vectors are the same (0 degrees); similar

# Cosine Similarity



## how can we calculate this in python?

Let's review a practical application <sup>2</sup>.

---

<sup>2</sup><https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb>



# Implementation in Python

```

1  from sklearn.feature_extraction.text import CountVectorizer,
   ↪ TfIdfVectorizer
2  import pandas as pd
3
4  doc1 = "When I eat breakfast, I usually drink some tea".lower()
5  doc2 = "I like my tea with my breakfast".lower()
6  doc3 = "She likes cereal and coffee".lower()
7
8  vec = CountVectorizer(stop_words='english')
9  count_matrix = vec.fit_transform([doc1, doc2])
10
11 print(pd.DataFrame(count_matrix.A,
   ↪ columns=vec.get_feature_names()).to_string())

```

|   | breakfast | cereal | coffee | drink | eat | like | likes | tea | usually |
|---|-----------|--------|--------|-------|-----|------|-------|-----|---------|
| 0 | 1         | 0      | 0      | 1     | 1   | 0    | 0     | 1   | 1       |
| 1 | 1         | 0      | 0      | 0     | 0   | 1    | 0     | 1   | 0       |
| 2 | 0         | 1      | 1      | 0     | 0   | 0    | 1     | 0   | 0       |

# Implementation in Python

```

1 doc1_vector = pd.DataFrame(count_matrix.A,
  ↳ columns=vec.get_feature_names()).T[0].to_list()
2 doc2_vector = pd.DataFrame(count_matrix.A,
  ↳ columns=vec.get_feature_names()).T[1].to_list()
3
4 print(f"The vector belonging to doc1: {doc1_vector}")
5 print(f"The vector belonging to doc2: {doc2_vector}")

```

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

The vector belonging to doc2: [1, 0, 0, 0, 0, 1, 0, 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:

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

doc1: [1, 0, 0, 1, 1, 0, 0, 1, 1]

doc2: [1, 0, 0, 0, 0, 1, 0, 1, 0]

$dot\_product = (1 \cdot 1) + (0 \cdot 0) + (0 \cdot 0) + (1 \cdot 0) + (1 \cdot 0) + (0 \cdot 0) + (1 \cdot 1) + (1 \cdot 0)$

Or, using Python:

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

## Implementation in Python

2. Execute the part of the formula in the denominator. Take the cross product of the two vectors.

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

Calculate this by hand:

$$\text{doc1} = \sqrt{1^2 + 0^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2}$$

$$\text{doc1} = \sqrt{1^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2}$$

Implementation in Python:

```

1 import math
2 doc1_ = math.sqrt(sum( [i**2 for i in doc1_vector] ) )
3 doc2_ = math.sqrt(sum( [i**2 for i in doc2_vector] ) )
4

```

# Implementation in Python

## 3.Finally:

```

1 cos_sim = dot_product / (doc1_ * doc2_)
2 print(cos_sim)

```

0.5163977794943222

# Implementation in Python

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

```

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

```

```

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

```

## How to use this in practice

### What can you do with this?

- This is especially powerful if you want to compare different news articles, movies, song texts, etc.
- For example, which movies are most similar in terms of genre composition?

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

Identify movies that are similar in terms of genre <sup>3</sup>

<sup>3</sup><https://www.learndatasci.com/glossary/cosine-similarity/>



# Considering Cosine Similarity

## Things to consider

- What type of overlap are you interested in?
- What is the meaning of n-grams, stems, stopwords when considering your RQ? How you should preprocess, 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

```

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

```

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

## Implementation in Python

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

```
1 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.”<sup>4</sup>

---

<sup>4</sup>[https:](https://radimrehurek.com/gensim//auto_examples/tutorials/run_scm.html)

[//radimrehurek.com/gensim//auto\\_examples/tutorials/run\\_scm.html](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) <sup>5</sup>
- How can we capture 'semantic' meaning?

### How?

- Convert words to *word vectors* and then compute similarities <sup>6</sup>

<sup>5</sup><http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf>

<sup>6</sup><https://www.machinelearningplus.com/nlp/cosine-similarity/>



Soft cosine similarity <sup>7</sup>

<sup>7</sup><https://www.dimachemels.com/nlp/nlp/soft-cosine-similarity/>

<https://www.dimachemels.com/nlp/nlp/soft-cosine-similarity/>

# Word embeddings

## Word embeddings

- To use the SCM, you need word embeddings.



# Soft cosine similarity

---

Word embeddings

# Understanding SCM

SCM estimates extracts similarity from **word embeddings**.

## What are word embeddings?

- No technical details here, just the general idea
- 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)

# Soft cosine similarity

---

## Implementation 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 <sup>8</sup>. ...

```
import gensim.downloader as api

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

---

<sup>8</sup><https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb>

# Create a dictionary

Let's review our 3 documents:

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

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

```
1 from gensim.utils import simple_preprocess
2 from gensim.corpora import Dictionary
3 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:

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

True

## Create a bag-of-words representation

Next, let's represent each document by (token\_id, token\_count) tuples:

```
1 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:

```
1 from gensim.similarities import SparseTermSimilarityMatrix  
2 from gensim.similarities import WordEmbeddingSimilarityIndex  
3  
4 similarity_index = WordEmbeddingSimilarityIndex(fasttext_model300)  
5 similarity_matrix = SparseTermSimilarityMatrix(similarity_index,  
  ↪ dictionary)
```

## Inspect results

Get SCM using `.inner_product()`:

```
1  #between doc1 and doc2:
2  scm_doc1_doc2 = similarity_matrix.inner_product(bag_of_words_vectors[0],
↳ bag_of_words_vectors[1], normalized=(True, True))
3
4  #between doc1 and doc3:
5  scm_doc1_doc3 = similarity_matrix.inner_product(bag_of_words_vectors[0],
↳ bag_of_words_vectors[2], normalized=(True, True))
6
7  #between doc2 and doc3;
8  scm_doc2_doc3 = similarity_matrix.inner_product(bag_of_words_vectors[1],
↳ bag_of_words_vectors[2], normalized=(True, True))
9
10 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 generally involve some overtime dynamics.

### Trace convergence or agenda setting dynamics over time

- Beyond the scope this course to discuss it here, but if you are interested in how you can apply cosine and soft cosine in an overtime analysis, we have prepared a notebook that will help you do just that.
- [https://github.com/annekroon/CCS-2/blob/main/week04/exercises/OPTIONAL\\_overtime\\_similarity.ipynb](https://github.com/annekroon/CCS-2/blob/main/week04/exercises/OPTIONAL_overtime_similarity.ipynb)



# 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

```
1      w1,w2,w3,w4,w5,w6 ...  
2 text1, 2, 0, 0, 1, 2, 3 ...  
3 text2, 0, 0, 1, 2, 3, 4 ...  
4 text3, 9, 0, 1, 1, 0, 0 ...  
5 ...
```

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 occurred 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%  $T_3$ , ... 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 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:

1

```
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

```

1 texts_clean = [text.lower() for text in texts]
2 texts_clean=[" ".join(text.split()) for text in texts_clean] #remove dubble spaces
3 texts_clean = [" ".join([l for l in text if l not in punctuation]) for text in texts_clean]
  ↳ #remove punctuation
4 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
```

## Preprocessing

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

```
1  mystopwords = set(stopwords.words('english')) # use default NLTK stopwords list;  
   ↪ alternatively:  
2  # mystopwords = set(open('mystopwordfile.txt').readlines()) #read stopwords list from a  
   ↪ textfile with one stopwords per line  
3  texts_clean = [" ".join(word for word in text.split() if word not in mystopwords) for text in  
   ↪ texts_clean]  
4  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)

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

which looks something like:

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

# LDA implementation

## LDA implementation

```

1 raw_m1 = tokenized_texts_clean
2
3 # assign a token_id to each word
4 id2word_m1 = corpora.Dictionary(raw_m1)
5 # represent each text by (token_id, token_count) tuples
6 ldacorporus_m1 = [id2word_m1.doc2bow(text) for text in raw_m1]
7
8 #estimate the model
9 lda_m1 = models.LdaModel(ldacorporus_m1, id2word=id2word_m1, num_topics=10)
10 lda_m1.print_topics()

```

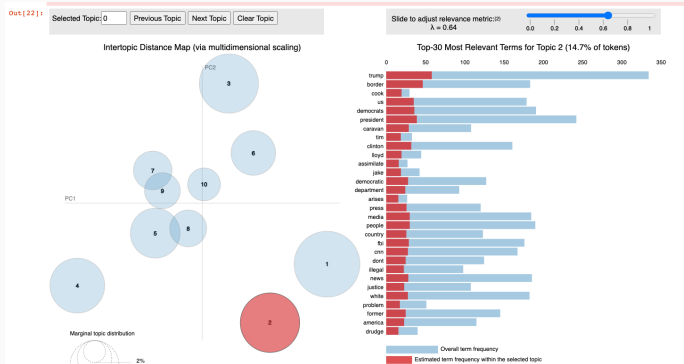
```

[(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" +

```

# Visualization with pyLDAvis

```
1 import pyLDAvis
2 import pyLDAvis.gensim_models as gensimvis
3 # first estimate gensim model, then:
4 vis_data = gensimvis.prepare(lda_m1,ldacorporus_m1,id2word_m1)
5 pyLDAvis.display(vis_data)
```



## Code examples

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

### qualitative: human judgement

Observation and interpretation based: observe the top N words in your topic, and evaluate the quality of the coherence of the topic. Can you identify words that do not belong to a topic?

### quantitative: 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?

- 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 peculiarities instead of finding general patterns
- There is no inherent ordering of topics
- 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 $\alpha$ : how sparse should the document-topic distribution $\theta$ be?

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

1

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

# 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).

## Next steps

---



## Exercise for this week's lab session

- Work through the example notebook on cosine and soft cosine:  
<https://github.com/annekroon/CCS-2/blob/main/week04/exercises/cosine-similarity-basics.ipynb>
- Work through the example notebook on LDA:  
<https://github.com/annekroon/CCS-2/blob/main/week04/exercises/topic-modelling.ipynb>
- *Optional:* overtime analysis of cosine or soft cosine:  
[https://github.com/annekroon/CCS-2/blob/main/week04/exercises/OPTIONAL\\_overtime\\_similarity.ipynb](https://github.com/annekroon/CCS-2/blob/main/week04/exercises/OPTIONAL_overtime_similarity.ipynb)

Cosine Similarity

oooooooooooooooooooo

Soft cosine similarity

ooooo  
oo  
oooooo

Topic modelling

ooooo  
ooooooooooooo  
ooooooo  
oo

Next steps

oo●

References

Have fun!

## References i

# References

---



Boumans, Jelle W. and Damian Trilling (2016). "Taking stock of the toolkit: An overview of relevant automated content analysis approaches and techniques for digital journalism scholars." In: *Digital Journalism* 4.1, pp. 8–23. ISSN: 2167-0811. DOI: [10.1080/21670811.2015.1096598](https://doi.org/10.1080/21670811.2015.1096598).



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](https://doi.org/10.1093/joc/jqab012).



Hager, Anselm and Hanno Hilbig (2020a). "Does Public Opinion Affect Political Speech?" In: *American Journal of Political Science* 64.4, pp. 921–937. DOI: <https://doi.org/10.1111/ajps.12516>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/ajps.12516>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/ajps.12516>.

## References ii



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



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](https://doi.org/10.13053/CyS-18-3-2043).