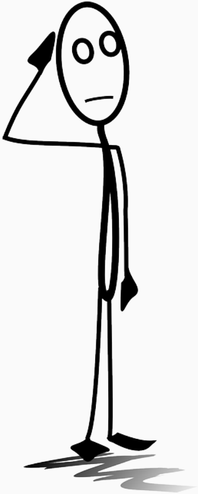
Big Data and Automated Content Analysis (6EC)
Week 6: »Unsupervised machine learning« Monday

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

May 8, 2023

UvA RM Communication Science

Today



Everything clear from previous weeks?
Questions?

This week, we will get a general overview of working with textual data. Due to a lack of time, I will introduce you to some of the basic concepts, point you to resources, and give you a practical, hands-on introduction.

Boumans and Trilling, 2016: Types of Automated Content Analysis

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

Bottom-up vs. top-down

Bottom-up

- Count most frequently occurring words
- Maybe better: Count combinations of words $\Rightarrow$ Which words co-occur together?

We *don't* specify what to look for in advance

Top-down

- Count frequencies of pre-defined words
- Maybe better: patterns instead of words

We *do* specify what to look for in advance

Bottom-up vs. top-down

Bottom-up

- Count most frequently occurring words
- Maybe better: Count combinations of words $\Rightarrow$ Which words co-occur together?

We *don't* specify what to look for in advance

Top-down

- Count frequencies of pre-defined words
- Maybe better: patterns instead of words

We *do* specify what to look for in advance

A simple bottom-up approach

```
from collections import Counter
texts = ["Communication in the Digital Society is a very very complex
↪  phenomenon", "I like to study it"]
bottom_up = []
for t in texts:
    bottom_up.append(Counter(t.lower().split()).most_common(3))
    print(bottom_up)
```

This results in:

```
[('very', 2), ('Communication', 1), ('in', 1)]
[('I', 1), ('like', 1), ('to', 1)]
```

Please note that you can also write this like:

```
bottom_up = [Counter(t.split()).most_common(3) for t  in texts]
```

A simple bottom-up approach

```python
from collections import Counter
texts = ["Communication in the Digital Society is a very very complex
↪  phenomenon", "I like to study it"]
bottom_up = []
for t in texts:
    bottom_up.append(Counter(t.lower().split()).most_common(3))
    print(bottom_up)
```

This results in:

```
[('very', 2), ('Communication', 1), ('in', 1)]
[('I', 1), ('like', 1), ('to', 1)]
```

Please note that you can also write this like:

```python
bottom_up = [Counter(t.split()).most_common(3) for t  in texts]
```

A simple bottom-up approach

```
from collections import Counter
texts = ["Communication in the Digital Society is a very very complex
↪  phenomenon", "I like to study it"]
bottom_up = []
for t in texts:
    bottom_up.append(Counter(t.lower().split()).most_common(3))
    print(bottom_up)
```

This results in:

```
[('very', 2), ('Communication', 1), ('in', 1)]
[('I', 1), ('like', 1), ('to', 1)]
```

Please note that you can also write this like:

```
bottom_up = [Counter(t.split()).most_common(3) for t  in texts]
```

A simple top-down approach

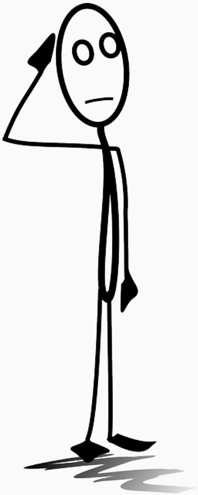
```python
texts = ["Communication in the Digital Society is a very very complex phenomenon", "I like to study it"]
features = ["communication", "digital", "study"]
for t in texts:
    print(f"\nAnalyzing '{t}':")
    for f in features:
        print(f"{f} occurs {t.lower().count(f)} times")
```

```
Analyzing 'Communication in the Digital Society is a very very complex phenomenon':
communication occurs 1 times
digital occurs 1 times
study occurs 0 times

Analyzing 'I like to study it':
communication occurs 0 times
digital occurs 0 times
study occurs 1 times
```

A simple top-down approach

```
texts = ["Communication in the Digital Society is a very very complex
↪  phenomenon", "I like to study it"]
features = ["communication", "digital", "study"]
for t in texts:
    print(f"\nAnalyzing '{t}':")
    for f in features:
        print(f"{f} occurs {t.lower().count(f)} times")
```

```
Analyzing 'Communication in the Digital Society is a very very complex phenomenon':
communication occurs 1 times
digital occurs 1 times
study occurs 0 times

Analyzing 'I like to study it':
communication occurs 0 times
digital occurs 0 times
study occurs 1 times
```



When would you use which approach?

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)
- You have a clear theoretical expectation? Bottom-up makes little sense.
- But in any case: you need to transform your text into something "countable".

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)
- You have a clear theoretical expectation? Bottom-up makes little sense.
- But in any case: you need to transform your text into something "countable".

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)
- You have a clear theoretical expectation? Bottom-up makes little sense.
- But in any case: you need to transform your text into something "countable".

Unsupervised Machine Learning for Text Classification

Supervised vs Unsupervised

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)
You might already know *some* techniques to figure out whether `x1`, `x2`,...`x_i` co-occur

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent Dirichlet Allocation)

Supervised vs Unsupervised

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)

You might already know *some* techniques to figure out whether `x1`, `x2`,...`x_i` co-occur

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent Dirichlet Allocation)

Supervised vs Unsupervised

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)

You might already know *some* techniques to figure out whether `x1`, `x2`,...`x_i` co-occur

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent Dirichlet Allocation)

Supervised vs Unsupervised

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)

You might already know *some* techniques to figure out whether `x1`, `x2,...x_i` co-occur

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent Dirichlet Allocation)

Supervised vs Unsupervised

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured x1, x2, x3 and you want to predict y, which you also measured

Unsupervised machine learning

You have no labels. (You did not measure `y`)

You might already know *some* techniques to figure out whether `x1`, `x2`,`...` `x_i` co-occur

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling (Non-negative matrix factorization and Latent Dirichlet Allocation)

Unsupervised Machine Learning for Text Classification

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 is a Bag-of-Words (BOW) approach

Doing a LDA in Python

We will use gensim (**Rehurek10softwareframework**) 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,

Doing a LDA in Python

We will use gensim (**Rehurek10softwareframework**) 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,

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
```

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
```

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
```

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
```

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

```python
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

```python
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

```
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)
```

Unsupervised Machine Learning for Text Classification

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()`)

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?

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?

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 pecularities 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 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"`)

```
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"`)

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

Unsupervised Machine Learning for Text Classification

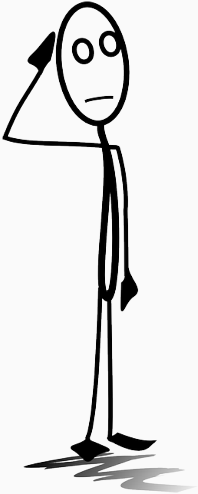
Using topic models

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).



Any questions?

Next steps

Take-home exam: you have time until Thursday 11th, end of the day

An example notebook with code for running LDA models can be found here:
https://github.com/uvacw/teaching-bdaca/blob/main/6ec-course/week06/exercises/

References

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant autmated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, *4*(1), 8–23. https://doi.org/10.1080/21670811.2015.1096598