

Big Data and Automated Content Analysis (12EC)

Week 11: »Unsupervised Approaches to Text Analysis: Topic Models« Wednesday

Felicia Loecherbach
f.loecherbach@uva.nl

April 26, 2022

UvA RM Communication Science

Before we start: Questions from last week?

Today

Today: Unsupervised machine learning for
text


Unsupervised ML



Recap: Can you explain the difference between supervised and unsupervised machine learning?

Boumans2016: Types of Automated Content Analysis

	Methodological approach		
	<i>Counting and Dictionary</i>	<i>Supervised Machine Learning</i>	<i>Unsupervised Machine Learning</i>
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



Our goal is to identify topics in texts, but
we do not know the topics in advance.

If you do have theoretical expectations, use
classic SML (or fine-tune a Transformer,
maybe with few-/zero-shot learning.)
instead.

Unsupervised ML

A historical overview: PCA, *k*-means, LDA

Defining the problem

Remember our earlier distinction:

1. Finding similar variables (dimension reduction)
2. Finding similar cases (clustering)

Are we more interested in which features “belong together” or which cases “belong together”?

Conceptually, we want to know *both* which features (words) belong to each other (=form a topic), *and* which cases (documents) contain the same topics.

Defining the problem

Remember our earlier distinction:

1. Finding similar variables (dimension reduction)
2. Finding similar cases (clustering)

Are we more interested in which features “belong together” or which cases “belong together”?

Conceptually, we want to know *both* which features (words) belong to each other (=form a topic), *and* which cases (documents) contain the same topics.

Defining the problem

Remember our earlier distinction:

1. Finding similar variables (dimension reduction)
2. Finding similar cases (clustering)

Are we more interested in which features “belong together” or which cases “belong together”?

Conceptually, we want to know *both* which features (words) belong to each other (=form a topic), *and* which cases (documents) contain the same topics.

Defining the problem

We assume a BOW approach like this (as produced by scikit-learn vectorizer):

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 ...
```

raw counts or tf·idf scores

Defining the problem

We could then go via two routes:

1. We run a PCA/SVD to see which features (words) load on the same component; and *then* look at the component scores per document
2. We run a *k*-means cluster analysis to see which texts are similar; and *then* look at the most common words per cluster

Defining the problem

We could then go via two routes:

1. We run a PCA/SVD to see which features (words) load on the same component; and *then* look at the component scores per document
2. We run a k -means cluster analysis to see which texts are similar; and *then* look at the most common words per cluster

Some considerations

If we do PCA/SVD...

- Components are ordered (first explains most variance) \Rightarrow We assume that some topics are more important than others
- Components do *not* necessarily carry a meaningful interpretation \Rightarrow But maybe OK in practice?
- We assume that a word belongs to one (not multiple) topics
- We assume that a document has a score for each topic

Some considerations

If we do cluster analysis. . .

- We assume that (in the case of k -means) that topics (are roughly) similarly sized
- We assume that a *document* belongs to one (not multiple) topics
- We assume that a word *can* belong to multiple topics.

Both approaches have been used

- Both have different assumptions, implications, and constraints
- Both easy to do with scikit-learn
- Both used in research (for instance, PCA by Leydesdorff2017<empty citation> or Greussing2017<empty citation>; or k -means cluster analysis by burscher2016<empty citation>)
- Typically, PCA groups features, cluster analysis groups texts – (but you can then use the component scores to describe the texts, and the cluster centroids to describe the features)
- Still occasionally used, but in general considered outdated

You find some slides with code examples in the appendix.

Beyond PCA and k -means

PCA was invented in 1901 (!), and k -means is around since the 1950s/1960s.

There surely must be something newer!

There is: Latent Dirichlet Allocation (LDA) (Blei2003).

Beyond PCA and k -means

PCA was invented in 1901 (!), and k -means is around since the 1950s/1960s.

There surely must be something newer!

There is: Latent Dirichlet Allocation (LDA) (**Blei2003**).

LDA solves some problems

Actually, we have *two* things we want to model:

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

⇒ LDA does both simultaneously!

It also does not suffer from a few problems:

- does the goal of PCA, to find a solution in which one word loads on *one* component match real life, where a word can belong to several topics or frames?
- does the goal of cluster analysis, assigning each document to *one* cluster, match real life?

LDA solves some problems

LDA is a model that

1. estimates *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” (Maier2018a)

Let that last point sink in for a second!

LDA solves some problems

LDA is a model that

1. estimates *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” (Maier2018a)

Let that last point sink in for a second!

LDA solves some problems

LDA is a model that

1. estimates *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” (**Maier2018a**)

Let that last point sink in for a second!

LDA solves some problems

LDA is a model that

1. estimates *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” (**Maier2018a**)

Let that last point sink in for a second!

LDA solves some problems

LDA is a model that

1. estimates *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” (**Maier2018a**)

Let that last point sink in for a second!

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 (like PCA) is a Bag-of-Words (BOW) approach

Doing a LDA in Python

You can use gensim **Rehurek2010** for this.

Let us assume you have a list of lists of words (!) called texts:

```
1 articles=['The tax deficit is higher than expected. This said xxx ...',  
           'Germany won the World Cup. After a']  
2 texts=[[token for token in re.split(r"\W", art) if len(token)>0] for art  
         in articles]
```

which looks like this:

```
1 [['The', 'tax', 'deficit', 'is', 'higher', 'than', 'expected', 'This', 'said', 'xxx'],  
   ['Germany', 'won', 'the', 'World', 'Cup', 'After', 'a']]
```

(note that we of course could use a better tokenizer!)

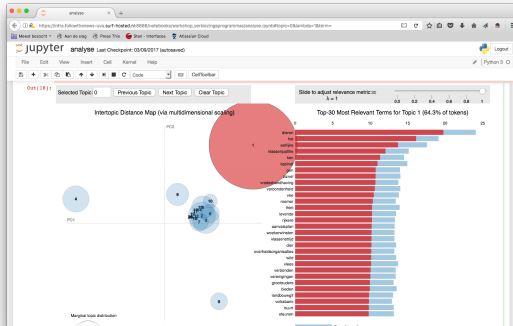
```
1 from gensim import corpora, models
2 import pandas as pd
3
4 NTOPICS = 100
5 LDAOUTPUTFILE="topicscores.tsv"
6
7 # Create a BOW representation of the texts
8 id2word = corpora.Dictionary(texts)
9 mm =[id2word.doc2bow(text) for text in texts]
10
11 # Train the LDA models.
12 mylda = models.ldamodel.LdaModel(corpus=mm, id2word=id2word, num_topics=
    NTOPICS, alpha="auto")
13
14 # Print the topics.
15 for top in mylda.print_topics(num_topics=NTOPICS, num_words=5):
16     print ("\n",top)
17
18 # the topic scores per document
19 topics = pd.DataFrame([dict(mylda.get_document_topics(doc,
    minimum_probability=0.0)) for doc in mm])
```

Output: Topics (below) & topic scores (next slide)

```
1  0.069*fusie + 0.058*brussel + 0.045*europesecommissie + 0.036*europese +  
   0.023*overname  
2  0.109*bank + 0.066*britse + 0.041*regering + 0.035*financien + 0.033*  
   minister  
3  0.114*nederlandse + 0.106*nederland + 0.070*bedrijven + 0.042*rusland +  
   0.038*russische  
4  0.093*nederlandsespoorwegen + 0.074*den + 0.036*jaar + 0.029*onderzoek +  
   0.027*raad  
5  0.099*banen + 0.045*jaar + 0.045*productie + 0.036*ton + 0.029*aantal  
6  0.041*grote + 0.038*bedrijven + 0.027*ondernemers + 0.023*goed + 0.015*  
   jaar  
7  0.108*werknemers + 0.037*jongeren + 0.035*werkgevers + 0.029*jaar +  
   0.025*werk  
8  0.171*bank + 0.122* + 0.041*klanten + 0.035*verzekeraar + 0.028*euro  
9  0.162*banken + 0.055*bank + 0.039*centrale + 0.027*leningen + 0.024*  
   financiële  
10 0.052*post + 0.042*media + 0.038*nieuwe + 0.034*netwerk + 0.025*  
    personeel  
11 ...
```


Visualization with pyldavis

```
1 import pyLDAvis
2 import pyLDAvis.gensim_models as gensimvis
3 # first estimate gensim model, then:
4 vis_data = gensimvis.prepare(mylda,mm,id2word)
5 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” (Maier2018a)

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 peculiarities 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 α from the data, using multiple passes:

```
1 mylda LdaModel(corpus=tfidfcorpus[ldacorporus], 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 α from the data, using multiple passes:

```
1 mylda LdaModel(corpus=tfidfcorpus[ldacorporus], 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.

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: **Tsur2015**

Unsupervised ML

Should one still use LDA?

The popularity of LDA

In the last decade, LDA has become *extremely* popular in the social sciences due to

- easy-to-use R and Python packages
- its promise to not require (a) manual (qual or quant) analysis; (b) annotations for SML; (c) creation of dictionaries etc.
- a bit of a “cool new technique” image

The popularity of LDA

But there is no silver bullet!

Unfortunately,

- validating topic models is hard – and many (most) studies don't do it (well);
- there are so many choices and parameters, in combination with no simple and definite evaluation metric, that it is very hard to justify why a particular model is chosen;
- experience shows that it often “doesn't work” \Rightarrow it's quite normal to have many uninterpretable or ambiguous topics;
- The smaller the dataset, the less likely it is to work
- LDA tends to also pick up peculiarities that don't matter and outliers

Solutions?

There are some extensions on classical LDA, in particular:

- Author-topic models
- Structural topic models (STM) (**Roberts2014**)
- Dynamic topic models (**Blei2006**)

These allow covariates (e.g., add info on who wrote a text) to improve the model, or allow to account for the changing use of words and topics over time.

Also, there are techniques for validation available (e.g., topic intrusion and/or word intrusion tasks).

Solutions?

But some we can't solve everything.

- It's still BOW.
- We cannot incorporate any language knowledge from larger, pre-trained datasets (e.g., via embeddings)

⇒ If we think of the performance leap that we observe with Transformers in other areas, we have all reason to assume that we can do better.

Unsupervised ML

State-of-the-art approaches to topic modelling

Let's bring in embeddings and Transformers!

Using embeddings and transformers for topic modelling

For example:

- top2vec ([angelov2020top2vec](#)), which embeds *topic vectors* in the same space as document vectors and word vectors
- Contextualized Topic models ([bianchi-etal-2021-pre](#); [bianchi-etal-2021-cross](#)), with a lot of code examples at <https://contextualized-topic-models.readthedocs.io/en/latest/introduction.html>
- ...

Using embeddings and transformers for topic modelling

For example:

- top2vec (**angelov2020top2vec**), which embeds *topic vectors* in the same space as document vectors and word vectors
- Contextualized Topic models (**bianchi-etal-2021-pre**; **bianchi-etal-2021-cross**), with a lot of code examples at <https://contextualized-topic-models.readthedocs.io/en/latest/introduction.html>
- ...

Using embeddings and transformers for topic modelling

For example:

- top2vec (**angelov2020top2vec**), which embeds *topic vectors* in the same space as document vectors and word vectors
- Contextualized Topic models (**bianchi-etal-2021-pre**; **bianchi-etal-2021-cross**), with a lot of code examples at <https://contextualized-topic-models.readthedocs.io/en/latest/introduction.html>
- ...

BERTopic (Grootendorst2022)

“In this paper, we introduce BERTopic, a topic model that leverages clustering techniques and a class-based variation of TF-IDF to generate coherent topic representations. More specifically, we first create document embeddings using a pretrained language model to obtain document-level information. Second, we first reduce the dimensionality of document embeddings before creating semantically similar clusters of documents that each represent a distinct topic. Third, to overcome the centroid-based perspective, we develop a classbased version of TF-IDF to extract the topic representation from each topic. These three independent steps allow for a flexible topic model that can be used in a variety of use-cases, such as dynamic topic modeling.”

(for details, read the paper)

BERTopic (Grootendorst2022)

Let's look at specific examples, for instance: https://maartengr.github.io/BERTopic/getting_started/quickstart/quickstart.html

but also the visualization capabilities:

https://maartengr.github.io/BERTopic/getting_started/visualization/visualization.html#visualize-topics-per-class

Much more coherent topics than LDA!

	20 NewsGroups		BBC News		Trump	
	TC	TD	TC	TD	TC	TD
LDA	.058	.749	.014	.577	-.011	.502
NMF	.089	.663	.012	.549	.009	.379
T2V-MPNET	.068	.718	-.027	.540	-.213	.698
T2V-Doc2Vec	.192	.823	.171	.792	-.169	.658
CTM	.096	.886	.094	.819	.009	.855
BERTopic-MPNET	.166	.851	.167	.794	.066	.663

Table 1: Ranging from 10 to 50 topics with steps of 10, topic coherence (TC) and topic diversity (TD) were calculated at each step for each topic model. All results were averaged across 3 runs for each step. Thus, each score is the average of 15 separate runs.

(And no need to set k ! And there is a dedicated “outlier topic” called -1 !)

Are there downsides?

Of course!

- By definition, much more “black-box”-y than BOW approaches
- Risk of biases introduced by LLM
- Much more resource-hungry (you probably want to do this with a GPU (e.g., on CoLab))

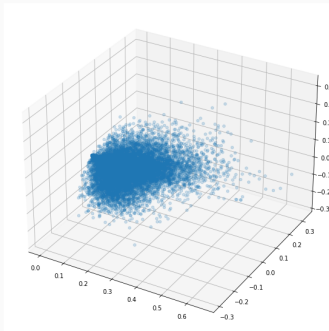
To conclude: PCA, k -means, LDA are interesting starting points – but if I were to start an unsupervised topic analysis model now, I'd go for BERTopic.

Appendix

```
1 from sklearn import datasets
2 from sklearn.decomposition import TruncatedSVD
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 from sklearn.pipeline import make_pipeline
5
6 texts = datasets.fetch_20newsgroups(data_home='rec.autos',
   ↪  remove=('headers', 'footers', 'quotes'), subset='train')['data']
7
8 myvec = TfidfVectorizer(max_df=.5, min_df=5,
   ↪  token_pattern='(?u)\\b[a-zA-Z][a-zA-Z]+\\b')
9 mysvd = TruncatedSVD(n_components=3)
10 mypipe = make_pipeline(myvec, mysvd)
11 r = mypipe.fit_transform(texts)
```

Plotting the texts according to their component scores

```
1 import matplotlib.pyplot as plt
2
3 fig = plt.figure(figsize=(10,10))
4 ax = fig.add_subplot(projection='3d')
5 ax.scatter([e[0] for e in r], [e[1] for e in r], [e[2] for e in r],alpha
            =.2)
```



Using the scores

```
1 import pandas as pd
2 textscores= pd.DataFrame(r)
3 featurescores = pd.DataFrame(mysvd.components_.T, index = myvec.
    get_feature_names())
```

featurescores

	0	1	2
aa	0.001019	0.001665	-0.001053
aaa	0.001799	0.003467	-0.002502
aaron	0.000583	0.001025	-0.000948
ab	0.000994	0.001040	-0.002188
abandon	0.000719	0.000626	0.000212
...
zur	0.000075	0.000344	-0.000342
zv	0.000066	-0.000134	-0.000123
zx	0.001135	-0.000953	0.000787
zy	0.000021	-0.000080	-0.000075
zz	0.000110	-0.000156	-0.000034

16150 rows × 3 columns

textscores

	0	1	2
0	0.252318	-0.048142	-0.100314
1	0.142955	-0.074158	0.006181
2	0.320513	-0.104095	-0.088318
3	0.179523	-0.086744	0.088419
4	0.156641	-0.003298	0.030814
...
11309	0.205703	-0.002241	0.030386
11310	0.183100	-0.096423	-0.070144
11311	0.129721	-0.011929	-0.031808
11312	0.159569	-0.016293	0.039790
11313	0.086385	-0.041932	-0.034792

11314 rows × 3 columns

Grouping features vs grouping cases

We have a corpus of a many texts.

- We used SVD to figure out relationships between features
- We could now look at the most important features per component (“topic”, “frame”?) by sorting featurescores
- We could see which texts are most representative for each “topic” or “frame” by sorting textscores

⇒ Alternative: Choose the opposite approach and first find out which cases are most similar, *then* describe what features characterize each group of cases

Grouping features vs grouping cases

We have a corpus of a many texts.

- We used SVD to figure out relationships between features
- We could now look at the most important features per component (“topic”, “frame”?) by sorting featurescores
- We could see which texts are most representative for each “topic” or “frame” by sorting textscores

⇒ Alternative: Choose the opposite approach and first find out which cases are most similar, *then* describe what features characterize each group of cases

```
1 from sklearn.cluster import KMeans
2
3 k = 5
4
5 mykm = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1)
6 myvec = TfidfVectorizer(max_df=.5, min_df=5,
7 ↪ token_pattern='(?u)\\b[a-zA-Z][a-zA-Z]+\\b')
8 mypipe = make_pipeline(myvec, mykm)
9
10 predictions = mypipe.fit_transform(texts)
11 # potentially: textscores = pd.DataFrame(predictions)
```

Of course, you need to determine the appropriate k ... (see earlier lecture)

Let's get the terms closest to the centroids

```
1 order_centroids = mykm.cluster_centers_.argsort()[:, :-1]
2 terms = myvec.get_feature_names()
3
4 print("Top terms per cluster:")
5
6 for i in range(k):
7     print("Cluster {}: ".format(i), end='')
8     for ind in order_centroids[i, :10]:
9         print("{} ".format(terms[ind]), end='')
10    print()
```

returns something like:

```
1 Top terms per cluster:
2 Cluster 0: windows file dos window with on you this have files
3 Cluster 1: you on this was with are have be not they
4 Cluster 2: thanks any me have anyone or please if on this
5 Cluster 3: he was his him not as this but on god
6 Cluster 4: you are be not they as this have if on
```

(of course, we could do sth similar with pandas as well)

Let's get the terms closest to the centroids

```
1 order_centroids = mykm.cluster_centers_.argsort()[:, :-1]
2 terms = myvec.get_feature_names()
3
4 print("Top terms per cluster:")
5
6 for i in range(k):
7     print("Cluster {}: ".format(i), end='')
8     for ind in order_centroids[i, :10]:
9         print("{} ".format(terms[ind]), end='')
10    print()
```

returns something like:

```
1 Top terms per cluster:
2 Cluster 0: windows file dos window with on you this have files
3 Cluster 1: you on this was with are have be not they
4 Cluster 2: thanks any me have anyone or please if on this
5 Cluster 3: he was his him not as this but on god
6 Cluster 4: you are be not they as this have if on
```

(of course, we could do sth similar with pandas as well)

Final project

Exercising with unsupervised machine learning for text

Two notebooks in this week's folder: LDA and BERTopic.

But in particular, look at the BERTopic website for more examples!

It's time to think about your final projects!

- Let's look at the course manual!
- Talk to me about your ideas!
- Main point: It needs to be sth you like, and it needs to cover techniques from both parts of the course!

