A Practical Introduction to Machine Learning in Python

Day 3 - Wednesday

»Unsupervised Machine Learning«

Damian Trilling
Anne Kroon

d.c.trilling@uva.nl, @damian0604
a.c.kroon@uva.nl, @annekroon

Gesis

March 11, 2020

Today

1. Recap: Types of Automated Content Analysis
2. Finding similar variables
   - An introduction to dimensionality reduction
   - Principal Component Analysis and Singular Value Decomposition
   - Multidimensional scaling
3. Finding similar cases
   - k-means clustering
   - Hierarchical clustering
4. Important notes
5. Recap: PCA and Clustering
6. LDA Topic models
   - An introduction to LDA
   - Choosing the best (or a good) topic model
   - Using topic models
   - Other forms of topic models

Recap: 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

Some terminology

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`)

Again, you already know some techniques to find out how `x1`, `x2`,...`x_i` co-occur from other courses:

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

Some terminology

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`)

Again, you already know some techniques to find out how `x1`, `x2`,...`x_i` co-occur from other courses:

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

Some terminology

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`)
Again, you already know some techniques to find out how `x1`, `x2`,...`x_i` co-occur from other courses:

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

Some terminology

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`)

Again, you already know some techniques to find out how `x1`, `x2`,...`x_i` co-occur from other courses:

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

Some terminology

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)

Again, you already know some techniques to find out how `x1`, `x2`,...`x_i` co-occur from other courses:

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

A lot of applications and use cases, . . .

. . . but we'll distinguish two today:

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"?

There are many other techniques than those presented today, and vice versa, those presented today can also be used for other purposes

A lot of applications and use cases, . . .

. . . but we'll distinguish two today:

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"?
There are many other techniques than those presented today, and vice versa, those presented today can also be used for other purposes

Finding similar variables

An introduction to dimensionality reduction

Dimensionality reduction

dimensionality = the number of features we have

(1) Explorative data analysis and visualization

- No good way to visualize 10,000 dimensions (or even 4)

(2) The curse of dimensionality

More features means more data (good!), but:

- Too many features can lead to unfeasible computation times
- We need more training cases to increase the likelihood that the possible combinations actually occur

Dimensionality reduction

dimensionality = the number of features we have

(1) Explorative data analysis and visualization

- No good way to visualize 10,000 dimensions (or even 4)

(2) The curse of dimensionality

More features means more data (good!), but:

- Too many features can lead to unfeasible computation times
- We need more training cases to increase the likelihood that the possible combinations actually occur

Dimensionality reduction

First approach: feature selection

- Only choose the features that are really relevant

Example: Exclude all terms that occur in more than 50% of the documents, or in less than $n = 5$ documents:

```
1 vec = CountVectorizer(max_df=0.5, min_df=5)
```

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

Dimensionality reduction

Second approach: feature extraction

- Create a smaller set of features
- E.g.: 1,000 features $\rightarrow$ PCA to reduce to 50 components $\rightarrow$ SML with these 50 component scores as features

Dimensionality reduction

So, we can use unsuvised ML as a dimension reduction step in a supervised ML pipeline. But it can also be a goal in itself, to

understand the data better or to visualize them.

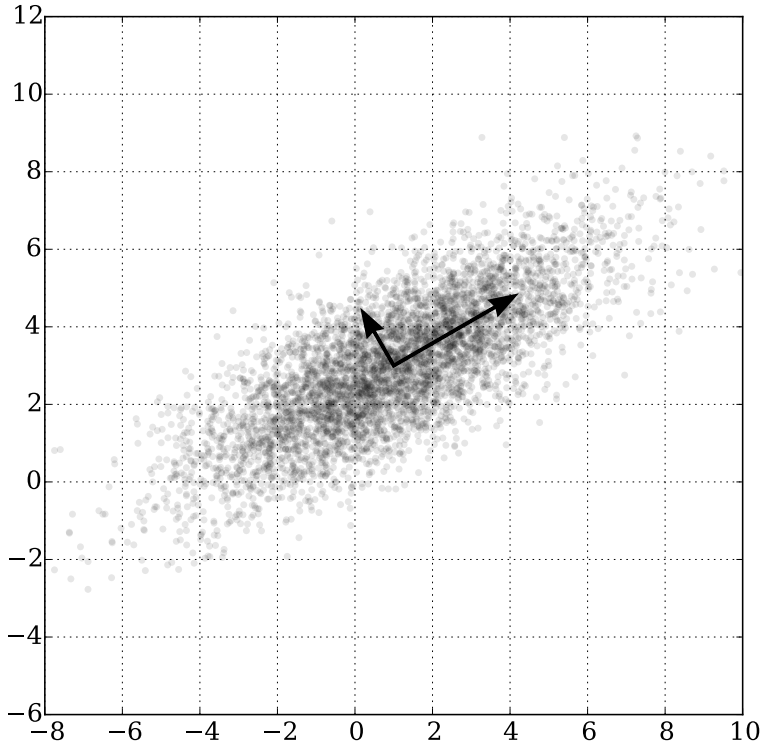
Finding similar variables

Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)

PCA

- related to and often confused with Factor Analysis (same menu item in SPSS – many people who believe they run FA actually run PCA!)
- Components are ordered (first explains most variance)
- Components do *not* necessarily carry a meaningful interpretation

PCA



https://upload.wikimedia.org/wikipedia/commons/f/f5/GaussianScatterPCA.svg

Preparation: Import modules and get some texts

```
1  from sklearn import datasets
2  from sklearn.decomposition import PCA
3  from sklearn.decomposition import TruncatedSVD
4  from sklearn.feature_extraction.text import CountVectorizer
5  from sklearn.pipeline import make_pipeline
6  from sklearn.preprocessing import FunctionTransformer
7  import matplotlib.pyplot as plt
8  %matplotlib inline
9
10 autotexts = datasets.fetch_20newsgroups('rec.autos', remove=('headers',
       'footers', 'quotes'), subset='train')['data']
11 religiontexts = datasets.fetch_20newsgroups('soc.religion.christian',
       remove=('headers', 'footers', 'quotes'), subset='train')['data']
12
13 texts = autotexts[:20] + religiontexts[:20]
```

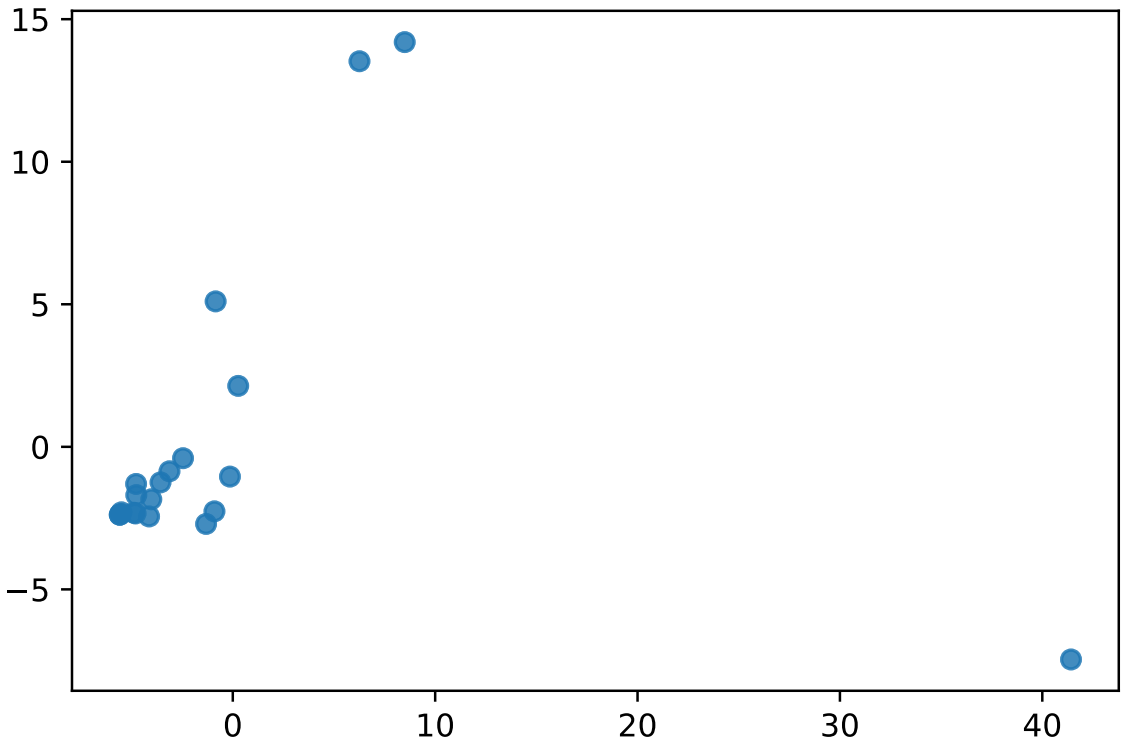
Running PCA

PCA does not accept a *sparse matrix* as input (but the CountVectorizer gives one as output), so we need to transform it into a *dense matrix*.

```
1  myvec = CountVectorizer(texts, max_df=.5, min_df=5)
2  mypca = PCA(n_components=2)
3
4  mypipe = make_pipeline(myvec, FunctionTransformer(lambda x: x.todense(),
       accept_sparse=True), mypca)
5
6  r = mypipe.fit_transform(texts)
```

Plotting the result

```
1  plt.scatter([e[0] for e in r], [e[1] for e in r], alpha=.6)
```



Singular value decomposition

The need to use a dense matrix is *really* a problem for large feature sets (which we have in NLP).

We therefore can better use SVD, which is essentially* the same and very simple to use:

```
mysvd = TruncatedSVD(n_components=2)
mypipe = make_pipeline(myvec, mysvd)
r = mypipe.fit_transform(texts)
```

(In this specific case, we even get exactly the same plot. . .)

* It's mathematically different, but you can SVD is even used "under the hood" by several PCA modules to solve PCA problems.

More info and background: https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8

Singular value decomposition

The need to use a dense matrix is *really* a problem for large feature sets (which we have in NLP).
We therefore can better use SVD, which is essentially* the same and very simple to use:

```
mysvd = TruncatedSVD(n_components=2)
mypipe = make_pipeline(myvec, mysvd)
r = mypipe.fit_transform(texts)
```

(In this specific case, we even get exactly the same plot. . .)
* It's mathematically different, but you can SVD is even used "under the hood" by several PCA modules to solve PCA problems.
More info and background: `https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8`

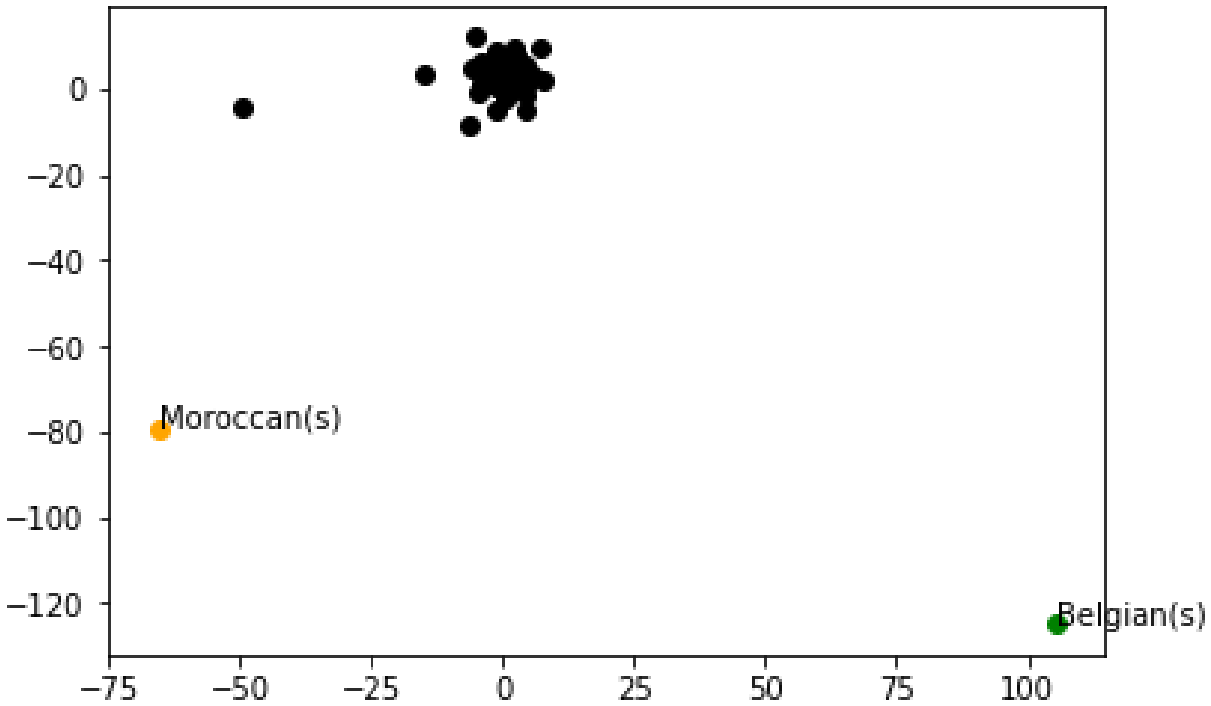
Finding similar variables

Multidimensional Scaling (MDS)

Multidimensional scaling

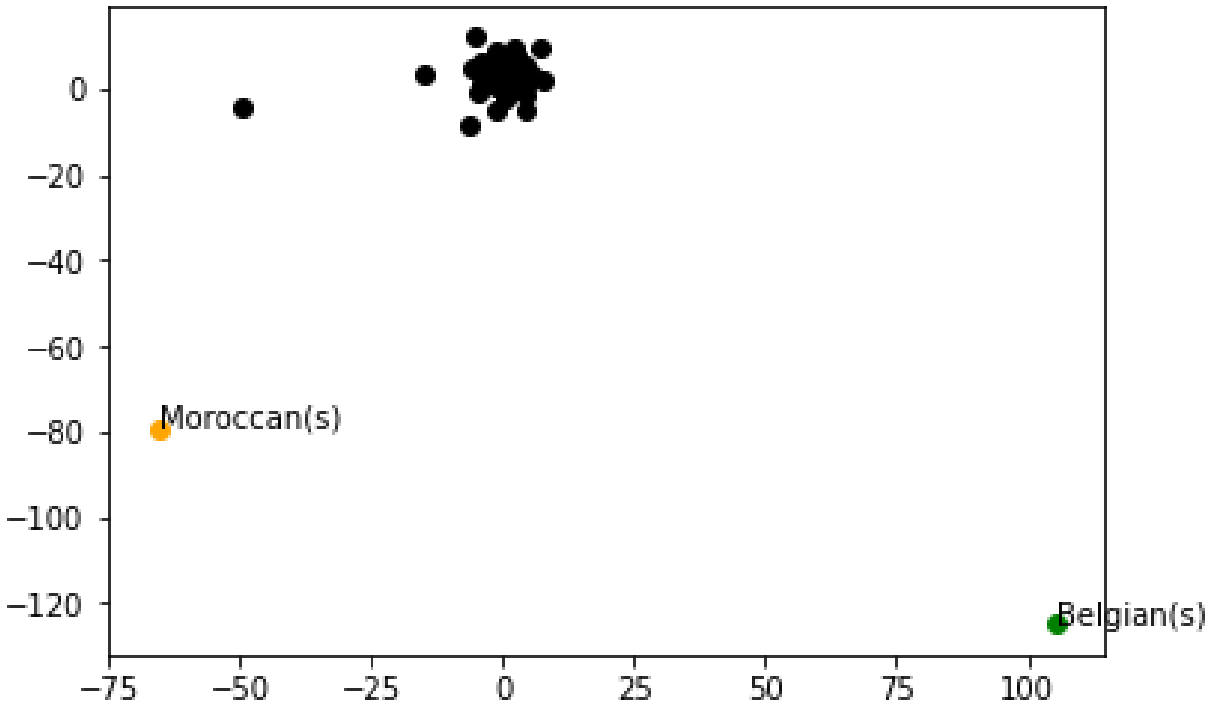
Assume we have a $n \times n$ matrix in which in which the cell entries indicate the distance between ech of our n features to each other feature based on their co-occurrence (= a dissimilarity matrix).

MDS helps us visualising this in a two- or threedimensional space.



Multidimensional scaling

Assume we have a $n \times n$ matrix in which in which the cell entries indicate the distance between ech of our n features to each other feature based on their co-occurrence (= a dissimilarity matrix). MDS helps us visualising this in a two- or threedimensional space.



Multidimensiponal scaling

- Low-dimensional representation of the data in which the *distances* respect well the distances in the original high-dimensional space
- With $D = 2$ and $D = 3$ often used for visualization (e.g., in political science)

Finding similar cases

k-means clustering

Grouping features vs grouping cases

Let's consider a corpus of several thousand user comments.
We could use SVD, MDS, or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

⇒ **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

Let's consider a corpus of several thousand user comments.
We could use SVD, MDS, or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

⇒ **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

Let's consider a corpus of several thousand user comments.
We could use SVD, MDS, or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

⇒ **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

Let's consider a corpus of several thousand user comments.
We could use SVD, MDS, or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

⇒ **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

Let's consider a corpus of several thousand user comments.
We could use SVD, MDS, or similar techniques to

- figure out relationships between features
- see which features stand out
- get a first sense what topics are in the corpus.

But:

- We do not learn anything about *which* texts (cases) belong to which topic
- We could use the component scores returned by `.fit_transform()` to then group our cases

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

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

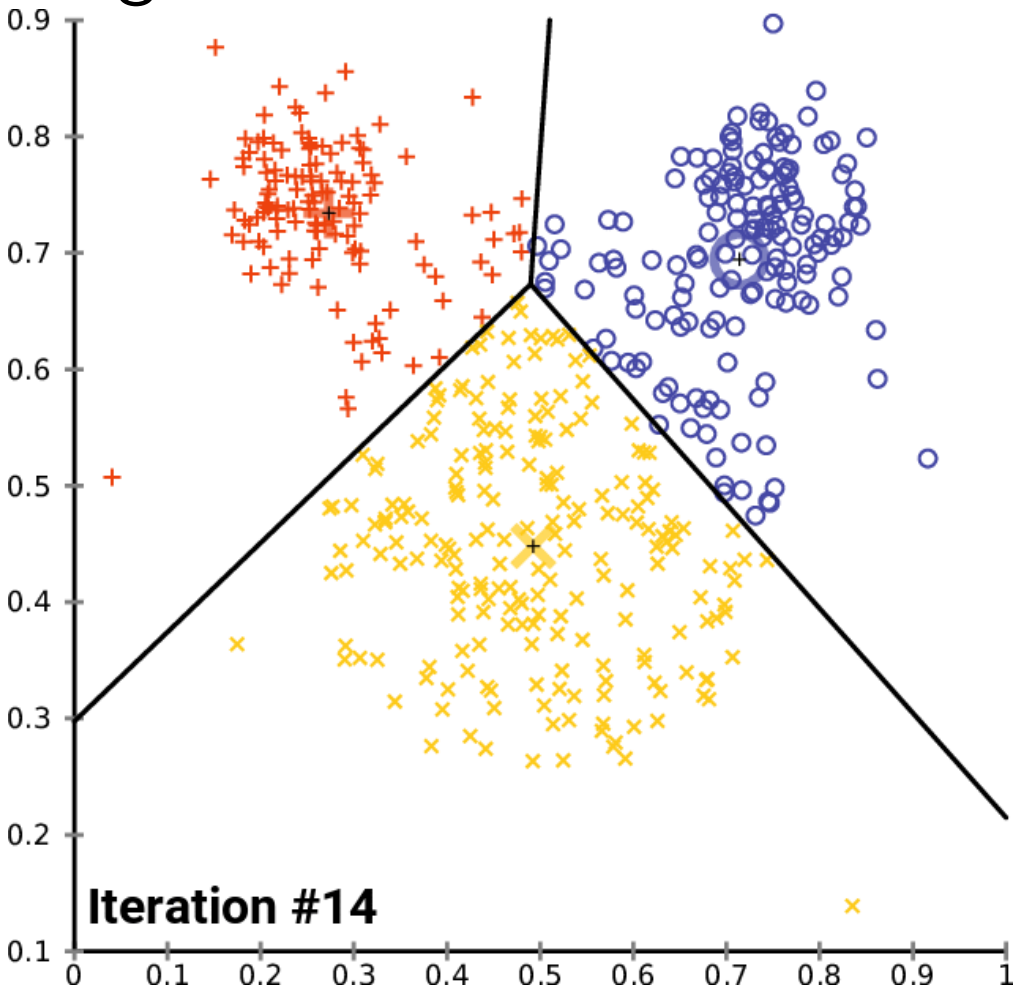
k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering

- Goal: group cases into k clusters
- k is set in advance
- Algorithm to determine k centroids (points in the middle of the cases that belong to it) such that the distances between the cases and their centroids are minimized
- non-deterministic: starts with a randomly choosen centroids (there are other versions)
- Cheap to compute: works even with large number of cases
- We can run PCA first to reduce the number of features if we want/need to

k-means clustering



https://upload.wikimedia.org/wikipedia/commons/e/ea/K-means_convergence.gif

Notice the big symbols indicating the centroids.

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
2  from sklearn.cluster import KMeans
3
4  k = 5
5  texts = ['text1 ejkh ek ekh', 'ekyerykel'] # a list of texts
6
7  vec = TfidfVectorizer(min_df=5, max_df=.4)
8  features = vec.fit_transform(texts)
9  km = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1)
10 predictions = km.fit_predict(features)
```

That's it!

- `predictions` is a list of integers indicated the predicted cluster number. We can thus use `zip(predictions, texts)` to put them together.
- We could also use `.fit()` and `.transform()` sperately and use our `km` to predict clusters for additional cases we have not used to train the model

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans

k = 5
texts = ['text1 ejkh ek ekh', 'ekyerykel'] # a list of texts

vec = TfidfVectorizer(min_df=5, max_df=.4)
features = vec.fit_transform(texts)
km = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1)
predictions = km.fit_predict(features)
```

That's it!

- `predictions` is a list of integers indicated the predicted cluster number. We can thus use `zip(predictions, texts)` to put them together.
- We could also use `.fit()` and `.transform()` sperately and use our `km` to predict clusters for additional cases we have not used to train the model

Let's get the terms closest to the centroids

```
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
terms = vec.get_feature_names()

print("Top terms per cluster:")

for i in range(k):
print("Cluster {}: ".format(i), end='')
for ind in order_centroids[i, :10]:
print("{} ".format(terms[ind]), end='')
print()
```

returns something like:

```
Top terms per cluster:
Cluster 0: heard could if opinions info day how really just around
Cluster 1: systems would ken pc am if as care summary ibm
Cluster 2: year car years was my no one higher single than
Cluster 3: which like seen 1000 few easily based personal work used
Cluster 4: as was he if they my all will get has
```

Let's get the terms closest to the centroids

```
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
terms = vec.get_feature_names()

print("Top terms per cluster:")

for i in range(k):
print("Cluster {}: ".format(i), end='')
for ind in order_centroids[i, :10]:
print("{} ".format(terms[ind]), end='')
print()
```

returns something like:

```
Top terms per cluster:
Cluster 0: heard could if opinions info day how really just around
Cluster 1: systems would ken pc am if as care summary ibm
Cluster 2: year car years was my no one higher single than
Cluster 3: which like seen 1000 few easily based personal work used
Cluster 4: as was he if they my all will get has
```

Using k-means clustering...

- we get the cluster membership for each text; and
- we get the terms that are most characteristic for the documents in each cluster.

Finding the optimal *k*

- The only way to find *k* is to estimate multiple models with different *k*s
- No single best solution; finding a balance between error within clusters (distances from centroid) and low number of clusters.
- An elbow plot can be helpful (see example in Burscher et al, 2016)

Code-example for creating an elbow plot:
`https://pythonprogramminglanguage.com/kmeans-elbow-method/`
(Don't forget to insert `%matplotlib inline` to actually see the plot)

Burscher, B., Vliegenthart, R., & de Vreese, C. H. (2016). Frames beyond words: Applying cluster and sentiment analysis to news coverage of the nuclear power issue. *Social Science Computer Review, 34*(5), 530-545. doi:10.1177/0894439315596385

Finding the optimal *k*

- The only way to find *k* is to estimate multiple models with different *k*s
- No single best solution; finding a balance between error within clusters (distances from centroid) and low number of clusters.
- An elbow plot can be helpful (see example in Burscher et al, 2016)

Code-example for creating an elbow plot:
`https://pythonprogramminglanguage.com/kmeans-elbow-method/`
(Don't forget to insert `%matplotlib inline` to actually see the plot)

Burscher, B., Vliegenthart, R., & de Vreese, C. H. (2016). Frames beyond words: Applying cluster and sentiment analysis to news coverage of the nuclear power issue. *Social Science Computer Review, 34*(5), 530-545. doi:10.1177/0894439315596385

Finding similar cases

Hierarchical clustering

Downsides of k-means clustering

k-means is fast, but has problems:

- k can only be determined by fitting multiple models and comparing them
- bad results if the wrong k is chosen
- bad results if the (real) clusters are non-spherical
- bad results if the (real) clusters are not evenly sized

Hiearchical clusttering

General idea

- To start, each case has its own cluster
- Merge the two clusters that are most similar
- Repeat until desired number of clusters is reached

Different options

- Stopping criterion: based on numerical statistic (e.g., Duda-Hart) or dendrogram
- Linkage: how to determine which two clusters should be merged?

Hiearchical clusttering

General idea

- To start, each case has its own cluster
- Merge the two clusters that are most similar
- Repeat until desired number of clusters is reached

Different options

- Stopping criterion: based on numerical statistic (e.g., Duda-Hart) or dendrogram
- Linkage: how to determine which two clusters should be merged?

Let's look into some options

`https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering`

$\Rightarrow$ Ward's linkage is a good default all-rounder choice, especially if you encounter the problem that other linkages lead to almost all cases ending up in one cluster.

Hierarchical clustering takeaway

- The main reason *not* to use hierarchical methods (but k-means) is their computational cost: when clustering survey data of media users, never use *k*-means!
- But for NLP/ML, costs may be too high (if not used carefully)
- Very much worth considering, though, if you are really into grouping cases!

Important notes for all types of clustering

Important notes

Consider the scales of measurement

Clustering is based on distances – if your features are not measured on the same scale, or if it is not meaningful to calculate a numerical distance, it won't produce meaningful results! Consider standardizing/whitening your features!

Pay attention outliers/extreme cases

Extreme cases or outliers can have a strong influence.

Do proper pre-processing

To reduce the number of features, but also to have *meaningful* features (dimensions on which you expect high distances between the clusters).

Important notes

Consider the scales of measurement

Clustering is based on distances – if your features are not measured on the same scale, or if it is not meaningful to calculate a numerical distance, it won't produce meaningful results! Consider standardizing/whitening your features!

Pay attention outliers/extreme cases

Extreme cases or outliers can have a strong influence.

Do proper pre-processing

To reduce the number of features, but also to have *meaningful* features (dimensions on which you expect high distances between the clusters).

Important notes

Consider the scales of measurement

Clustering is based on distances – if your features are not measured on the same scale, or if it is not meaningful to calculate a numerical distance, it won't produce meaningful results! Consider standardizing/whitening your features!

Pay attention outliers/extreme cases

Extreme cases or outliers can have a strong influence.

Do proper pre-processing

To reduce the number of features, but also to have *meaningful* features (dimensions on which you expect high distances between the clusters).

Exercise for this afternoon

1. Go to `https://figshare.com/articles/News-Processed-Dataset/5296357` and download `WSJ_20170607_to_20170726_10AmTo4Pm.json` (the small file of 9 MB)
2. You can read the file as follows:

```
import json
data = []
with  open('/home/damian/Downloads/WSJ_20170607_to_20170726_10AmTo4Pm.
    json') as f:
  for line in f:
    data.append(json.loads(line))
    texts = [e['content'] for e in data]
```

3. Use unsupervised machine learning techniques (and/or other techniques) to draw inferences about topics of (groups of) texts!

Let's get to one of the most popular unsupervised methods of the moment – topic modeling.

Recap: PCA and Clustering

Let's assume we want to find out the topics in a large corpus of documents

We could either

- use PCA to find out related features (and interpret those as topics)
- or use clustering to find similar documents (and then look at the words they share to interpret as topics)

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?

Let's assume we want to find out the topics in a large corpus of documents

We could either

- use PCA to find out related features (and interpret those as topics)
- or use clustering to find similar documents (and then look at the words they share to interpret as topics)

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?

Recap: PCA

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
- some problematic assumptions: 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?

Recap: PCA

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
- some problematic assumptions: 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?

Recap: clustering

- given a term-document matrix, we can easily find clusters of documents that resemble each other
- but also here does the goal of cluster analysis, assigning each document to *one* cluster, match real life?

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

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

LDA, what's that?

No mathematical details here, but the general idea

- There are k topics, $T_1 \ldots T_k$
- Each document D_i consists of a mixture of these topics, e.g. $80\% T_1, 15\% T_2, 0\% T_3, \ldots 5\% T_k$
- 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 (Řehůřek & Sojka, 2010) for this.
Let us assume you have a list of lists of words (!) called `texts`:

```
articles=['The tax deficit is higher than expected. This said xxx ...',
    'Germany won the World Cup. After a']
texts=[art.split() for art in articles]
```

which looks like this:

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

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.

```
from gensim import corpora, models

NTOPICS = 100
LDAOUTPUTFILE="topicscores.tsv"

# Create a BOW represenation of the texts
id2word = corpora.Dictionary(texts)
mm =[id2word.doc2bow(text) for text in texts]

# Train the LDA models.
mylda = models.ldamodel.LdaModel(corpus=mm, id2word=id2word, num_topics=
    NTOPICS, alpha="auto")

# Print the topics.
for top in mylda.print_topics(num_topics=NTOPICS, num_words=5):
print ("\n",top)

print ("\nFor further analysis, a dataset with the topic score for each
    document is saved to",LDAOUTPUTFILE)

scoresperdoc=mylda.inference(mm)

with open(LDAOUTPUTFILE,"w",encoding="utf-8") as fo:
for row in scoresperdoc[0]:
fo.write("\t".join(["{:0.3f}".format(score) for score in row]))
fo.write("\n")
```

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*
       financiele
10 0.052*post + 0.042*media + 0.038*nieuwe + 0.034*netwerk + 0.025*
       personeel
11 ...
```

Data Editor (Browse) – topicscores.dta

Edit | Browse | Filter | Variables | Properties | Snapshots

topic4[2] .019

	source2	firstwords	polarity	subjectivity	pubdate_day	pubdate_mo~h	pubdate_year	pubdate_da~k	topic1	topic2	topic3	topic4	topic5
1	nrc handelsblad	palingsound schinke	-.0086207	.6069971	31	12	2011	zaterdag	.018	.019	3.587	.019	.019
2	nrc handelsblad	groep investeerders	-.1041667	.3129167	31	12	2011	zaterdag	.018	.019	.019	.019	.019
3	nrc handelsblad	abnamro debacles ij	.0082292	.4895443	31	12	2011	zaterdag	.018	27.71	.019	.019	.019
4	nrc handelsblad	abnamro financiv´le	-.0179617	.5706419	31	12	2011	zaterdag	.018	15.1	.019	2.646	.019
5	nrc handelsblad	crisis verhouding k	.0758049	.5448864	31	12	2011	zaterdag	.018	.019	9.008	.019	.019
6	nrc handelsblad	snel vakantie vrije	-.016315	.5118808	31	12	2011	zaterdag	.018	.019	.019	.019	.019
7	nrc handelsblad	herinnering doos le	.18875	.6208333	31	12	2011	zaterdag	.018	.019	.019	.019	.019
8	nrc handelsblad	hackers publiceren	.1454545	.4545455	31	12	2011	zaterdag	.018	.019	.019	.019	.019
9	nrc handelsblad	waterballet montevi	-.2333333	.4333333	31	12	2011	zaterdag	.018	.019	.019	.019	.019
10	nrc handelsblad	bouw dupe ambities	.0925417	.5939167	5	11	2010	vrijdag	.018	.019	.078	2.442	.019
11	nrc handelsblad	eindelijk wint nuch	.1755093	.48125	5	11	2010	vrijdag	.018	.019	8.302	.019	.019
12	nrc handelsblad	oud nieuws tv bbcst	.02	.4322222	5	11	2010	vrijdag	.018	10.053	.019	.019	.019
13	nrc handelsblad	tmg hyves krantenbe	.0425203	.5420412	5	11	2010	vrijdag	.018	.019	.019	.019	.019
14	nrc handelsblad	getuigenis rechter	.0858929	.5770833	5	11	2010	vrijdag	.018	.019	.019	11.621	.019
15	nrc handelsblad	akzonobel philips g	.0220455	.4381818	5	11	2010	vrijdag	.018	.019	.019	.019	.019
16	nrc handelsblad	mondiaal kritiek be	-.038172	.3094624	5	11	2010	vrijdag	.018	19.957	.019	.019	.019
17	nrc handelsblad	export diamant fiat	.0628571	.4438095	5	11	2010	vrijdag	.018	4.745	.019	.019	.019
18	nrc handelsblad	canada bod potash r	.0252924	.4795322	5	11	2010	vrijdag	.018	26.741	.019	.019	.019
19	nrc handelsblad	zwakke bouwsector c	.0171	.4736333	14	3	2009	NA	.018	.019	.019	.019	4.806
20	nrc handelsblad	pensioenconflict wa	.028114	.4636842	14	3	2009	NA	.018	.019	.019	.019	.019
21	nrc handelsblad	rechter allin loon	.1318182	.3939394	14	3	2009	NA	.018	.019	.019	.019	.019
22	nrc handelsblad	bad bank remedie da	.0891026	.550641	14	3	2009	NA	.018	10.235	.019	.019	.019
23	nrc handelsblad	bescheiden salaris	-.075	.56	14	3	2009	NA	.018	.019	.019	.019	.019
24	nrc handelsblad	generalmotors autos	.0138889	.4388889	14	3	2009	NA	.018	.019	.019	.019	.019
25	nrc handelsblad	rusland rozen tuinb	.0314141	.5643051	14	3	2009	NA	.018	.019	24.595	.019	.019
26	nrc handelsblad	cynisme oplossing k	.0100833	.6511667	14	3	2009	NA	.018	.019	.019	.019	.019
27	nrc handelsblad	the good bed ugly l	.0265504	.5298449	13	3	2009	NA	.018	.019	.019	.019	.019
28	nrc handelsblad	kerk stroom nietswe	-.0087719	.6149123	13	3	2009	NA	.018	.019	.019	.019	.019
29	nrc handelsblad	kerk stroom goud ac	0	0	13	3	2009	NA	.018	.019	.019	.019	.019
30	nrc handelsblad	supersnelle koekenp	0	0	13	3	2009	NA	.018	.019	.019	.019	.019
31	nrc handelsblad	dalailama chinese e	0	0	13	3	2009	NA	.018	.019	.019	.019	.019
32	nrc handelsblad	bezuinigen hulpgeld	.0894192	.4560606	13	3	2009	NA	.018	.019	.019	.019	.019
33	nrc handelsblad	vaders arbeidsethos	.0160985	.5575758	13	3	2009	NA	.018	.019	.019	.019	.019
34	nrc handelsblad	varkens lux winnaar	.040873	.6218254	4	10	2008	NA	.018	.019	.019	.019	.019
35	nrc handelsblad	liberale kinderopva	.1179095	.5297055	4	10	2008	NA	.018	.019	.019	.019	1.03
36	nrc handelsblad	banken verzinsels k	.068521	.6308389	4	10	2008	NA	8.232	.019	.019	.019	.019
37	nrc handelsblad	rabobanktopman bert	0	0	4	10	2008	NA	.018	.019	.019	.019	.019
38	nrc handelsblad	kinderopvang bril v	0	0	4	10	2008	NA	.018	.019	.019	.019	.019
39	nrc handelsblad	tassen gevoel verli	0	0	4	10	2008	NA	.018	.019	.019	.019	.019
40	nrc handelsblad	abnamro winkelend p	.0876761	.62277	4	10	2008	NA	.018	.019	6.904	.019	5.511
41	nrc handelsblad	abnamro belgiv´ mole	.0439506	.4976852	4	10	2008	NA	.018	.019	.019	.019	.019
42	nrc handelsblad	abnamro handen deut	.1838401	.5264302	4	10	2008	NA	.018	.019	1.854	.019	.019
43	nrc handelsblad	abnamro fortis bank	.0842391	.494058	4	10	2008	NA	4.939	.019	14.39	.019	.019
44	nrc handelsblad	abnamro fortis spra	.0540715	.6290807	4	10	2008	NA	.018	.019	.019	.019	.019
45	nrc handelsblad	abnamro fortis jaar	.0297297	.4960135	4	10	2008	NA	.018	11.041	.019	.019	.019
46	nrc handelsblad	abnamro nederland s	.1006944	.6030555	4	10	2008	NA	.018	.019	.019	.019	.019
47	nrc handelsblad	abnamro belgiv´ mole	.0405952	.5004464	4	10	2008	NA	.018	.019	.019	.019	.019
48	nrc handelsblad	arbeidsmarkt vs sle	.0166667	.4	4	10	2008	NA	7.103	.019	.019	.019	12.682

Variables

Enter filter text here

Name	Label
byline	
section	

Properties

Variables

Name	section
Label	
Type	str26
Format	%26s
Value Label	
Notes	

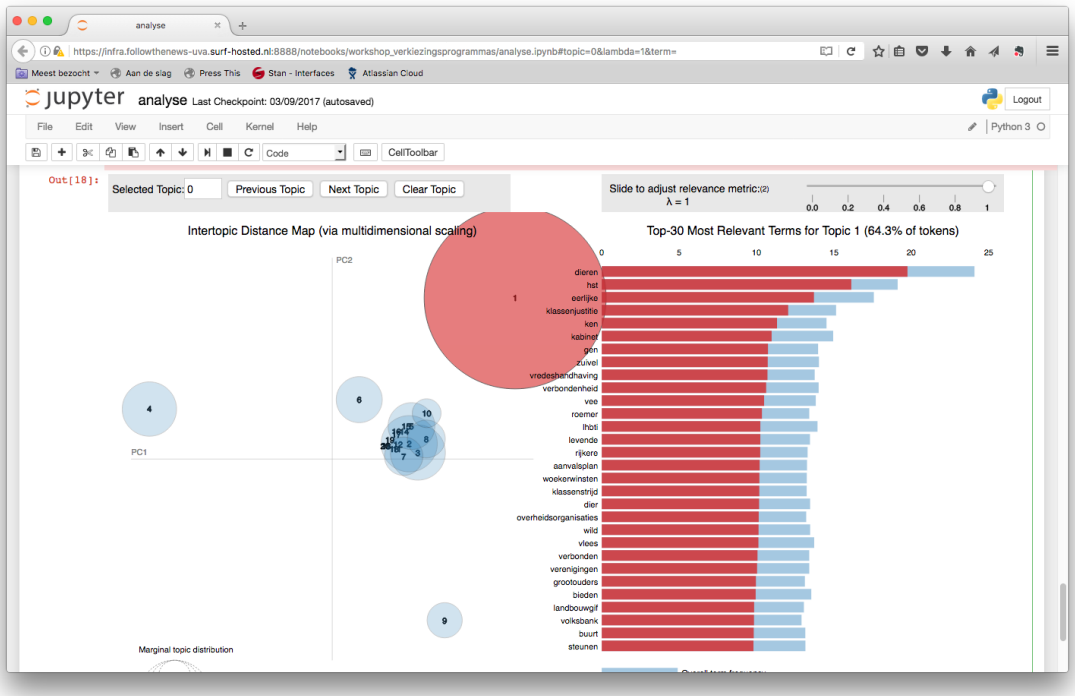
Data

Filename	topicscores.
Label	
Notes	
Variables	164
Observations	28,406
Size	14.06M
Memory	64M

Vars: 158 of 164 | Obs: 28,406 | Filter: Off

Visualization with pyldavis

```
import pyLDAvis
import pyLDAvis.gensim
# first estiate gensim model, then:
vis_data = pyLDAvis.gensim.prepare(mylda,mm,id2word)
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/damian0604/bdaca/blob/master/rm-course-2/week12/lda.ipynb

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.

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).

Other forms of topic models

Other forms of topic models

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