# 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

September 29, 2021

Gesis

# Today

Types of Automated Content Analysis

Finding similar variables

- An introduction to dimensionality reduction
- Principal Component Analysis and Singular Value Decomposition

Finding similar cases

- k-means clustering
- Hierarchical clustering

Important notes

# Automated Content Analysis

# Recap: Types of Automated Content Analysis

# Automated Content Analysis

## Top-down vs. bottom-up

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

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


Boumans and Trilling, 2016

The same logic applies to non-textual data!

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


Boumans and Trilling, 2016

The same logic applies to non-textual data!

# 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`)

**You might already be familiar with 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)

# 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`)
You might already be familiar with 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)

# 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`)

**You might already be familiar with 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)

# 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)

**You might already be familiar with 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)

# 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)

**You might already be familiar with 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)

## Let's distinguish four use cases...

1. Finding similar variables (dimensionality reduction) – unsupervised
2. Finding similar cases (clustering) – unsupervised
3. Predicting a continous variable (regression) – supervised
4. Predicting group membership (classification) – supervised

| | x1 | x2 | x3 | x4 | x5 | y |
|---|---|---|---|---|---|---|
| case1 | 110 | 110 | 110 | 110 | 110 | 110 |
| case2 | 110 | 110 | 110 | 110 | 110 | 110 |
| case3 | 110 | 110 | 110 | 110 | 110 | 110 |
| case4 | 110 | 110 | 110 | 110 | 110 | 110 |

| | x1 | x2 | x3 | x4 | x5 | (y) |
|---|---|---|---|---|---|---|
| case1 | 110 | 110 | 110 | 110 | 110 | (110) |
| case2 | 110 | 110 | 110 | 110 | 110 | (110) |
| case3 | 110 | 110 | 110 | 110 | 110 | (110) |
| case4 | 110 | 110 | 110 | 110 | 110 | (110) |

Dimensionality reduction: finding similar variables (features)

| | x1 | x2 | x3 | x4 | x5 | (y) |
|---|---|---|---|---|---|---|
| case1 | 110 | 110 | 110 | 110 | 110 | (110) |
| case2 | 110 | 110 | 110 | 110 | 110 | (110) |
| case3 | 110 | 110 | 110 | 110 | 110 | (110) |
| case4 | 110 | 110 | 110 | 110 | 110 | (110) |

Clustering: finding similar cases

| | x1 | x2 | x3 | x4 | x5 | | y |
|---|---|---|---|---|---|---|---|
| case1 | 110 | 110 | 110 | 110 | 110 | $\rightarrow$ | 110 |
| case2 | 110 | 110 | 110 | 110 | 110 | $\rightarrow$ | 110 |
| case3 | 110 | 110 | 110 | 110 | 110 | $\rightarrow$ | 110 |
| case4 | 110 | 110 | 110 | 110 | 110 | $\rightarrow$ | 110 |
| | | | | | | | |
| new case | 110 | 110 | 110 | 110 | 110 | $\rightarrow$ | ? |

Regression and classification: learn how to predict $y$.

**Note, again, that the 110 signs can be *anything*.** For us, often word counts or $tf \cdot$ idf scores ($x$) and, for supervised approaches, a topic, a sentiment, or similar ($y$).

But it could also be pixel colors or clicks on links or anything else.

| | x1 | x2 | x3 | x4 | x5 | y |
|---|---|---|---|---|---|---|
| case1 | 110 | 110 | 110 | 110 | 110 | 110 |
| case2 | 110 | 110 | 110 | 110 | 110 | 110 |
| case3 | 110 | 110 | 110 | 110 | 110 | 110 |
| case4 | 110 | 110 | 110 | 110 | 110 | 110 |

# 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

# Finding similar variables

## An introduction to dimensionality reduction

## 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:

```
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 → PCA to reduce to 50 components → 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 and Singular Value Decomposition

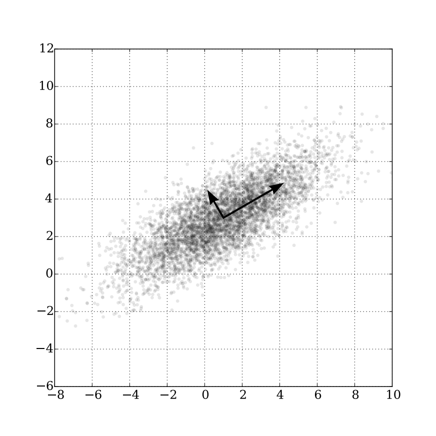
## Finding similar variables

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

```
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import FunctionTransformer
import matplotlib.pyplot as plt
%matplotlib inline

autotexts = datasets.fetch_20newsgroups('rec.autos', remove=('headers',
     'footers', 'quotes'), subset='train')['data']
religiontexts = datasets.fetch_20newsgroups('soc.religion.christian',
     remove=('headers', 'footers', 'quotes'), subset='train')['data']

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*.

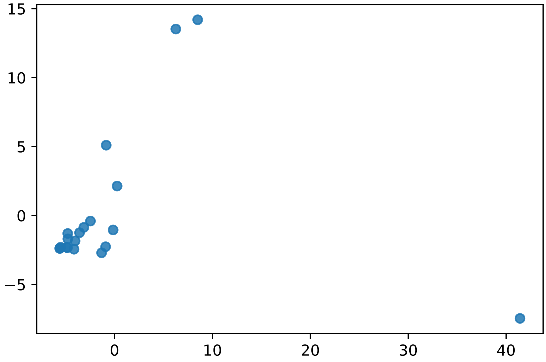
```
myvec = CountVectorizer(texts, max_df=.5, min_df=5)
mypca = PCA(n_components=2)

mypipe = make_pipeline(myvec, FunctionTransformer(lambda x: x.todense(),
      accept_sparse=True), mypca)

r = mypipe.fit_transform(texts)
```

# Plotting the result

```
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 cases

# Finding similar cases

## k-means clustering

## 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**

# 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**

## 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

## 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

## 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**

## 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

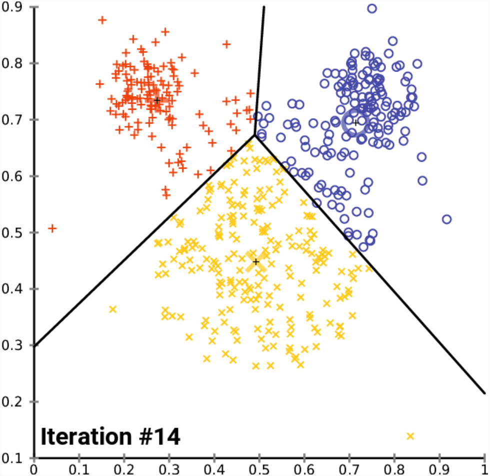
## k-means clustering

- 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.

```python
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

```
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

## 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

⇒ 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

# 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

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!

This afternoon we will discuss one of the most popular unsupervised methods of the moment – topic modeling.