ML for natural and physical scientists 2023 4

clustering

dr.federica bianco | *fbb.space* |  *fedhere* |  *fedhere*

this slide deck:

https://slides.com/federicabianco/mlpns23_4

- **Machine Learning basic concepts**
  - interpretability
  - parameters vs hyperparameters
  - supervised/unsupervised
- **Clustering : how it works**
  - **Partitioning**
    - **Hard clustering**
    - **Soft Clustering**
  - **Hirarchical**
    - **agglomerative**
    - **devisive**
  - **also:**
    - **Density based**
    - **Grid based**
    - **Model based**

A really nice medium post by Kayu Jan Wong came out just as I was preparing this lecture!

https://towardsdatascience.com/6-types-of-clustering-methods-an-overview-7522dba026ca

Kay Jan Wong Follow

Mar 24 · 8 min read · Member-only · Listen

6 Types of Clustering Methods — An Overview

Types of clustering methods and algorithms and when to use them

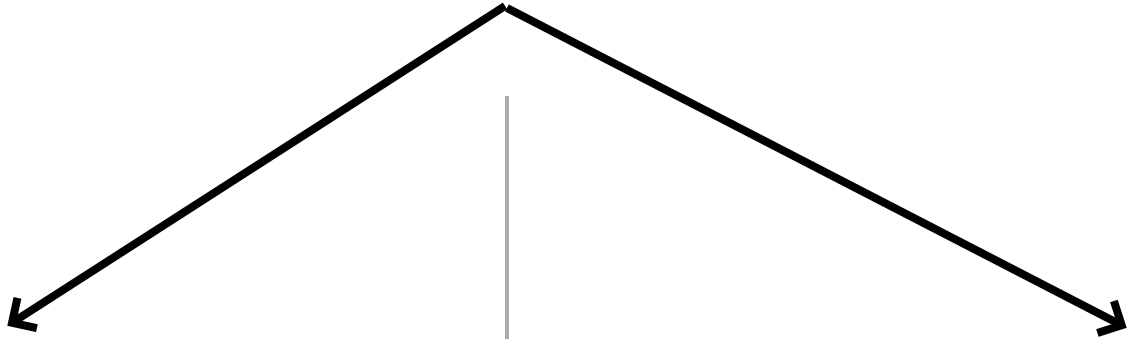
Table of Contents

- Centroid-based / Partitioning (*K-means*)
- Connectivity-based (*Hierarchical Clustering*)
- Density-based (*DBSCAN*)
- Graph-based (*Affinity Propagation*)
- Distribution-based (*Gaussian Mixture Model*)
- Compression-based (*Spectral Clustering, BIRCH*)

spectral clustering and affinity propagation are ***not*** covered in these slides

what is machine learning

what is machine learning?



supervised learning

classification

prediction

feature selection

unsupervised learning

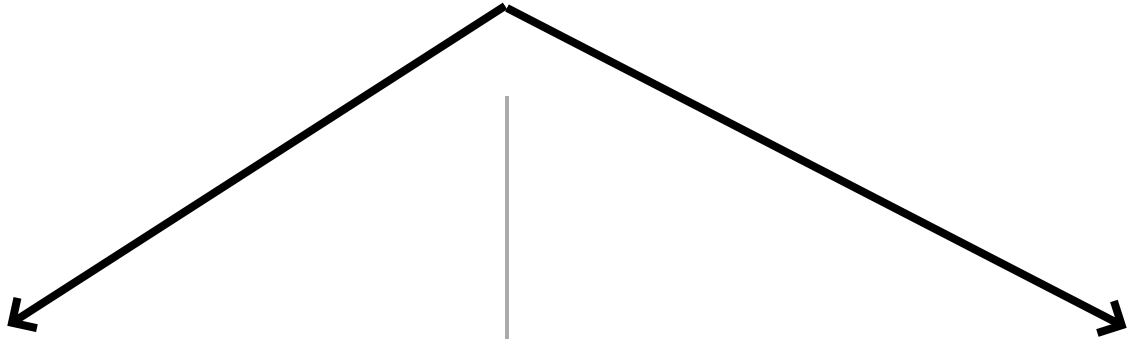
understanding structure

organizing/compressing data

anomaly detection

dimensionality reduction

what is machine learning?



supervised learning

k-Nearest Neighbors

~~**Regression**~~

Support Vector Machines

Classification/Regression Trees

Neural networks

classification

prediction

feature selection

unsupervised learning

understanding structure

organizing/compressing data

anomaly detection

dimensionality reduction

clustering

PCA

Apriori

general ML points

used to:

understand structure of feature space

classify based on examples,

regression (classification with infinitely small classes)

general ML points

ML model have *parameters* and *hyperparameters*

parameters: the model optimizes based on the data

hyperparameters: chosen by the model author, could be based on domain knowledge, other data, guessed (?!).

e.g. the shape of the polynomial

general ML points

ML model have *parameters* and *hyperparameters*

parameters:

hyperparameters:

```python
import numpy as np
import emcee

def log_prob(x, ivar):
    return -0.5 * np.sum(ivar * x ** 2)

ndim, nwalkers = 5, 100
ivar = 1. / np.random.rand(ndim)
p0 = np.random.randn(nwalkers, ndim)

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_prob, args=[ivar])
sampler.run_mcmc(p0, 10000)
```

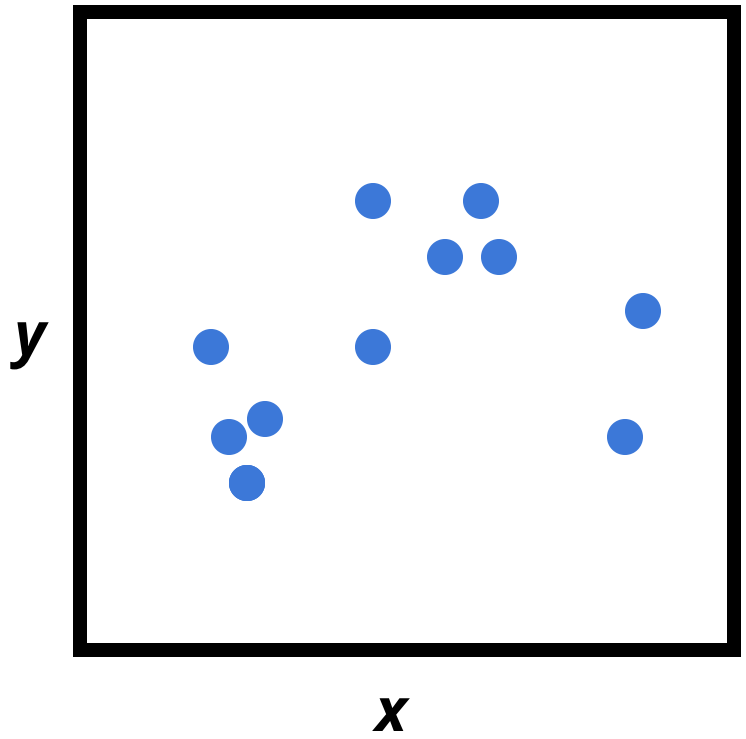
1 clustering

clustering is an unsupervised learning method

GOAL: partitioning data in *maximally homogeneous, maximally distinguished* subsets.

observed **features:**

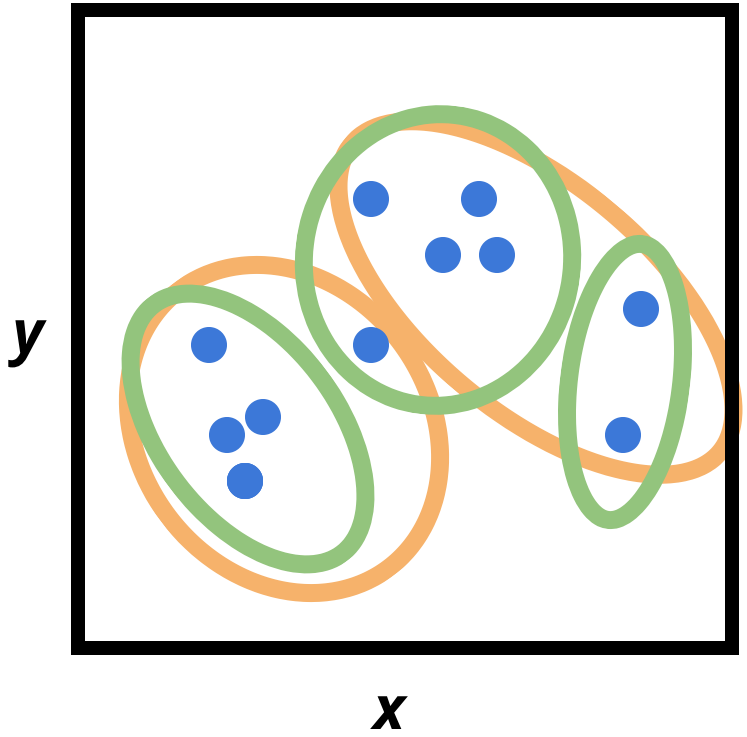
$(\vec{x}, \vec{y})$



clustering is an unsupervised learning method

all features are observed for all objects in the sample

$(\vec{x}, \vec{y})$



how should I group the observations in this feature space?

e.g.: how many groups should I make?

optimal clustering

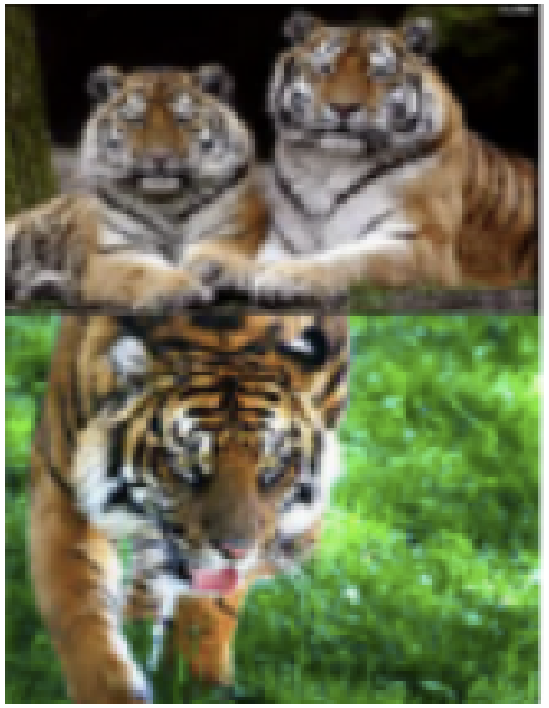
http://www-bcf.usc.edu/~soltanol/Applications.html

internal criterion:

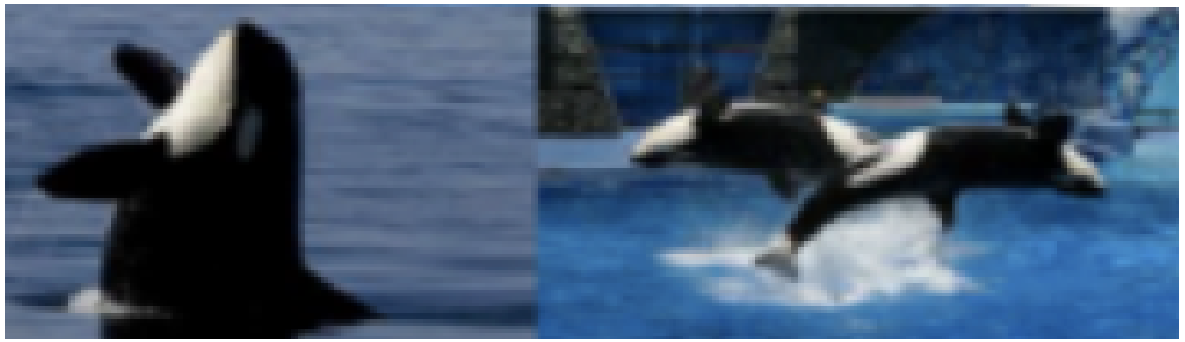
members of the cluster should be similar to each other (intra cluster compactness)

external criterion:

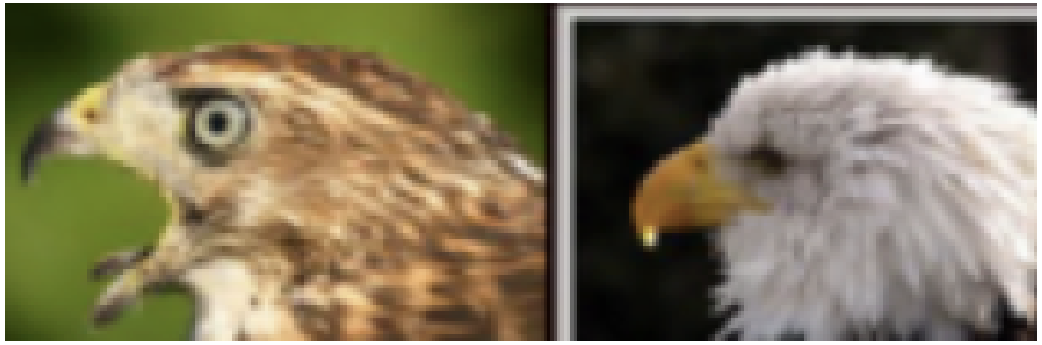
objects outside the cluster should be dissimilar from the objects inside the cluster



tigers



wales



raptors

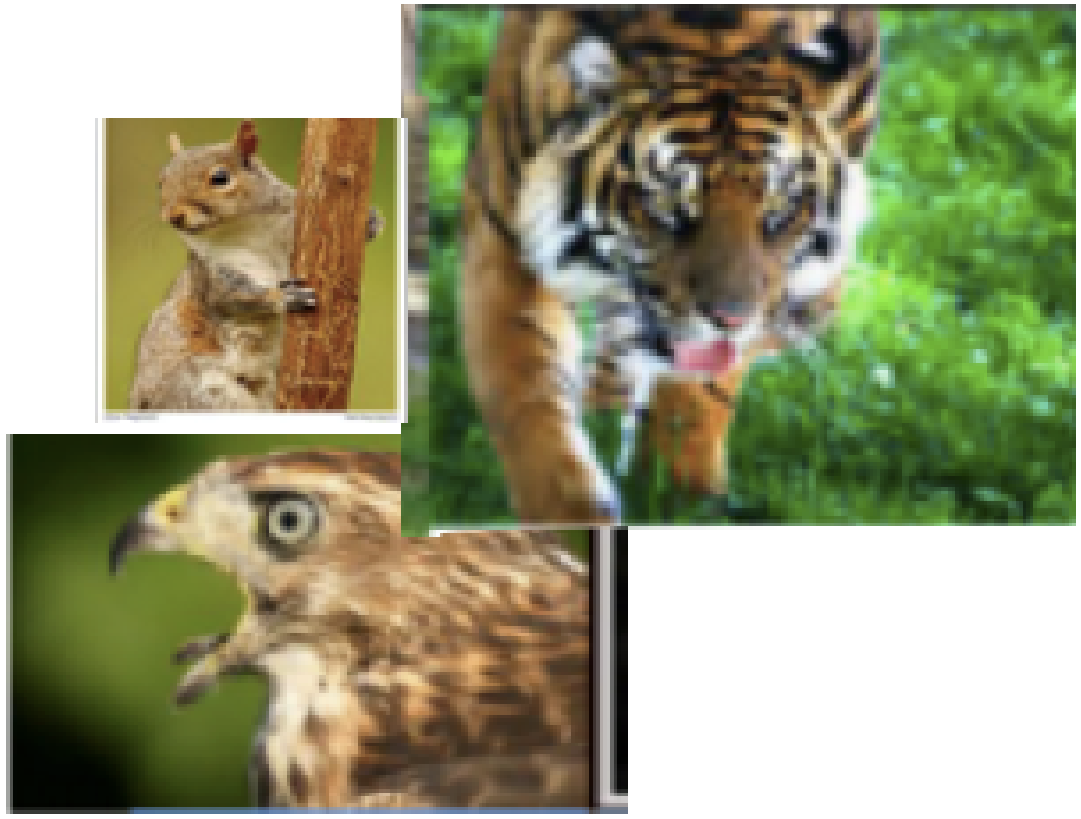
zoologist's clusters

internal criterion:

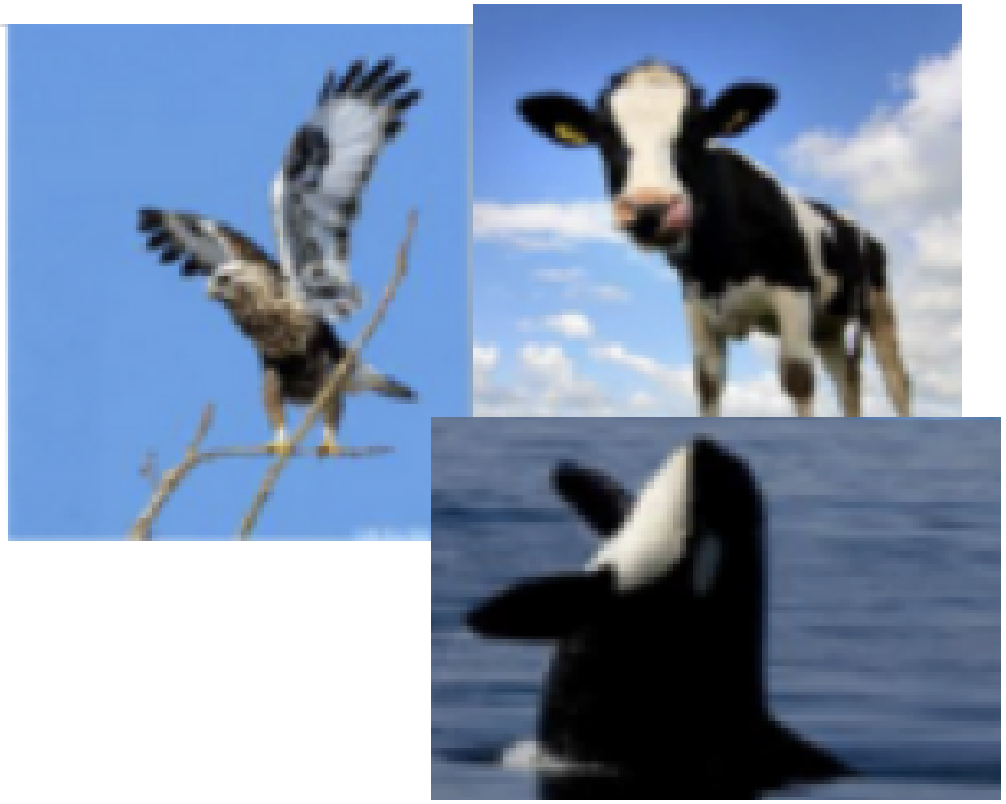
members of the cluster should be similar to each other (intra cluster compactness)

external criterion:

objects outside the cluster should be dissimilar from the objects inside the cluster



orange/green



black/white/blue

photographer's clusters

internal criterion:

members of the cluster should be similar to each other (intra cluster compactness)

external criterion:

objects outside the cluster should be dissimilar from the objects inside the cluster

the optimal clustering depends on:

- how you define similarity/distance
- the purpose of the clustering

the ideal clustering algorithm should have:

- Scalability (naive algorithms are Np hard)
- Ability to deal with different types of attributes
- Discovery of clusters with arbitrary shapes
- Minimal requirement for domain knowledge
- Deals with noise and outliers
- Insensitive to order
- Allows incorporation of constraints
- Interpretable

distance metrics

2

distance metrics
continuous variables

Minkowski family of distances

$$D(i,j) = \sqrt[p]{|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \ldots + |x_{iN} - x_{jN}|^p}$$

distance metrics continuous variables

Minkowski family of distances

$$D(i,j) = \sqrt[p]{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^p}$$

N features (dimensions)

distance metrics

continuous variables

Minkowski family of distances

$$D(i,j) = \sqrt[p]{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^p}$$

N features (dimensions)

properties:

$$D(i,j) > 0$$

$$D(i,i) = 0$$

$$D(i,j) = D(j,i)$$

$$D(i,j) <= D(i,k) + D(k,j)$$

distance metrics

continuous variables

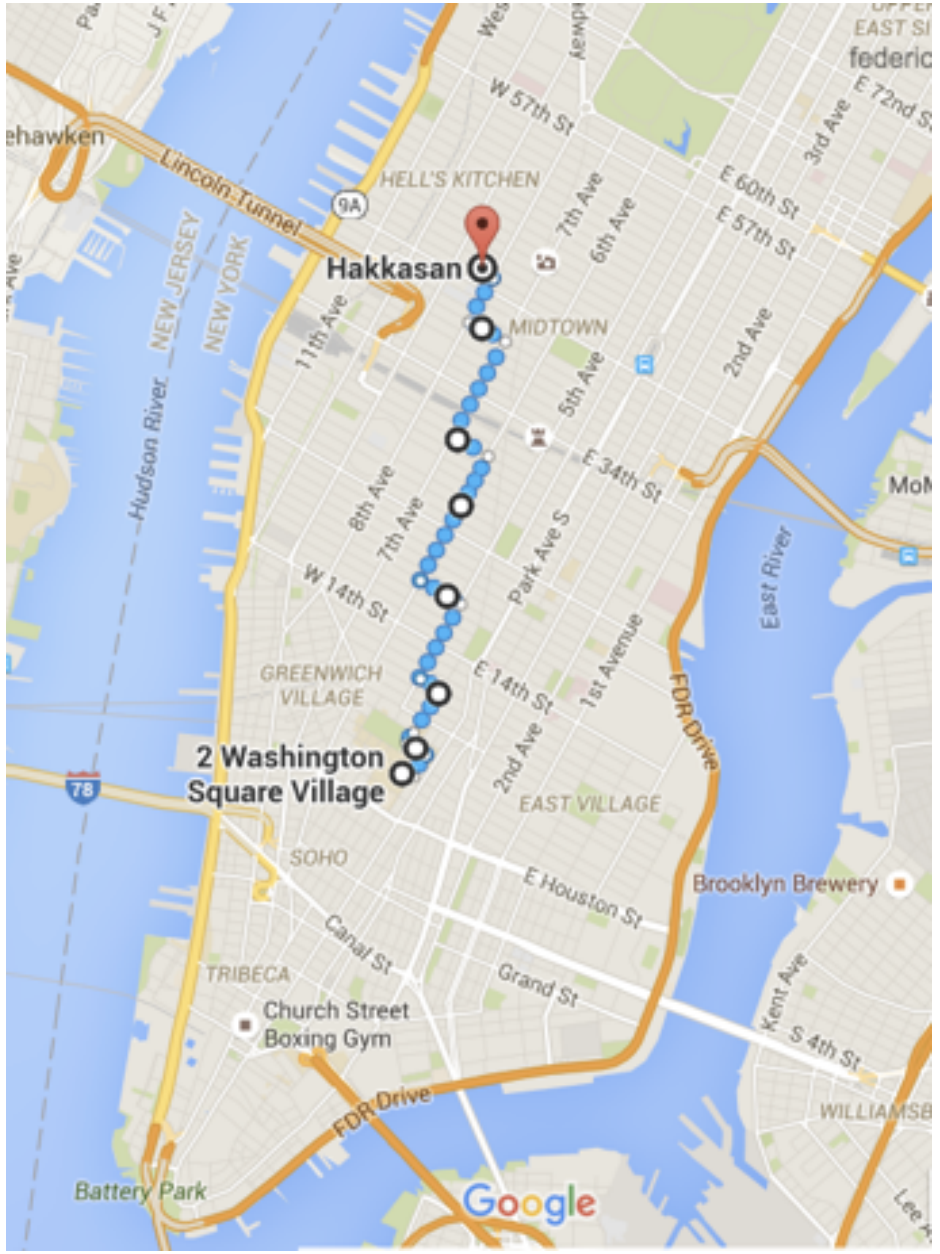
Minkowski family of distances

$$D(i,j) = \sqrt[p]{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^p}$$

Manhattan: p=1

$$D_{Man}(i,j) = \sum_{k=1}^{N} |x_{ik} - x_{jk}|$$

features: x, y



distance metrics continuous variables

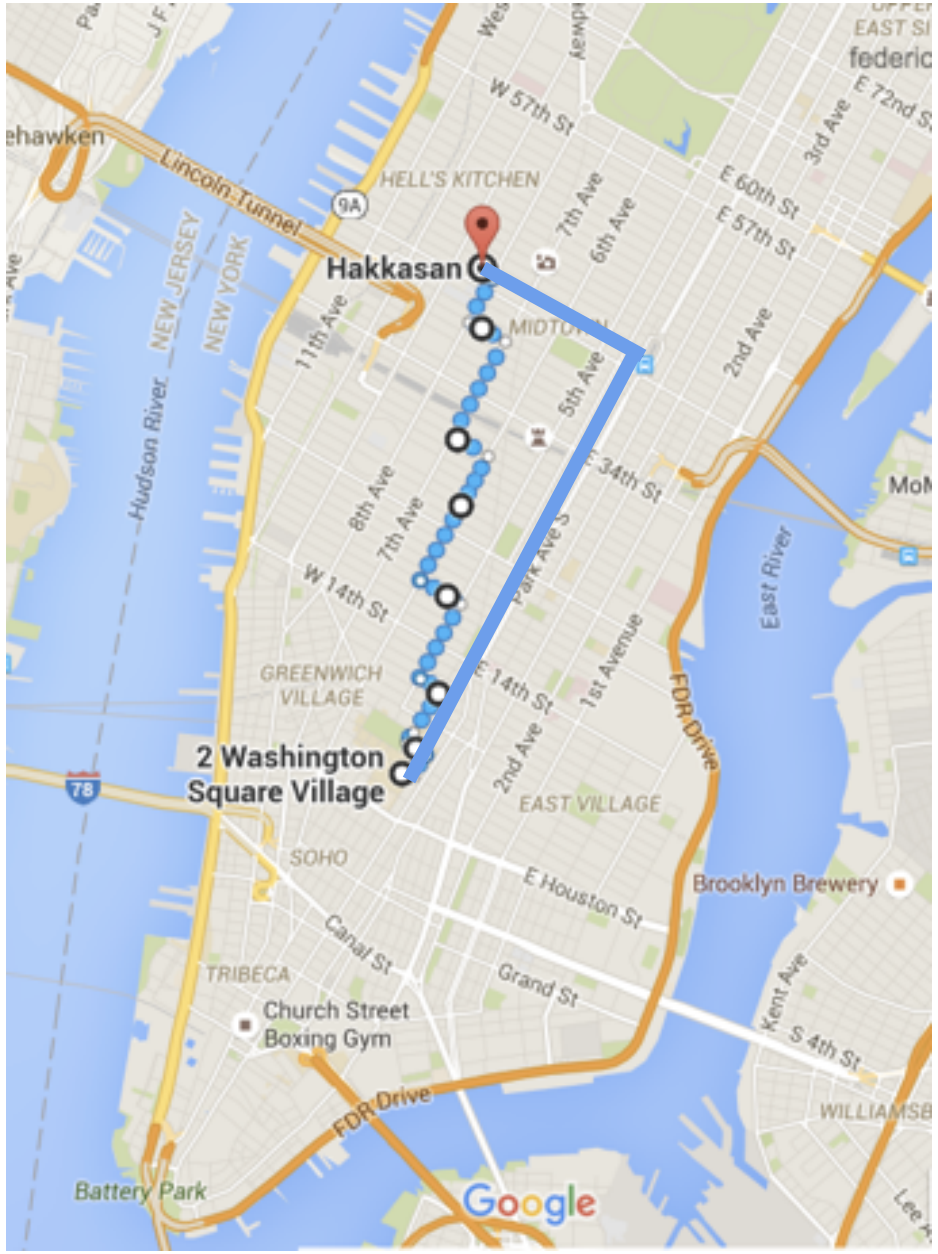
Minkowski family of distances

$$D(i,j) = \sqrt[p]{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^p}$$

Manhattan: p=1

$$D_{Man}(i,j) = \sum_{k=1}^{N} |x_{ik} - x_{jk}|$$

features: x, y



distance metrics

continuous variables

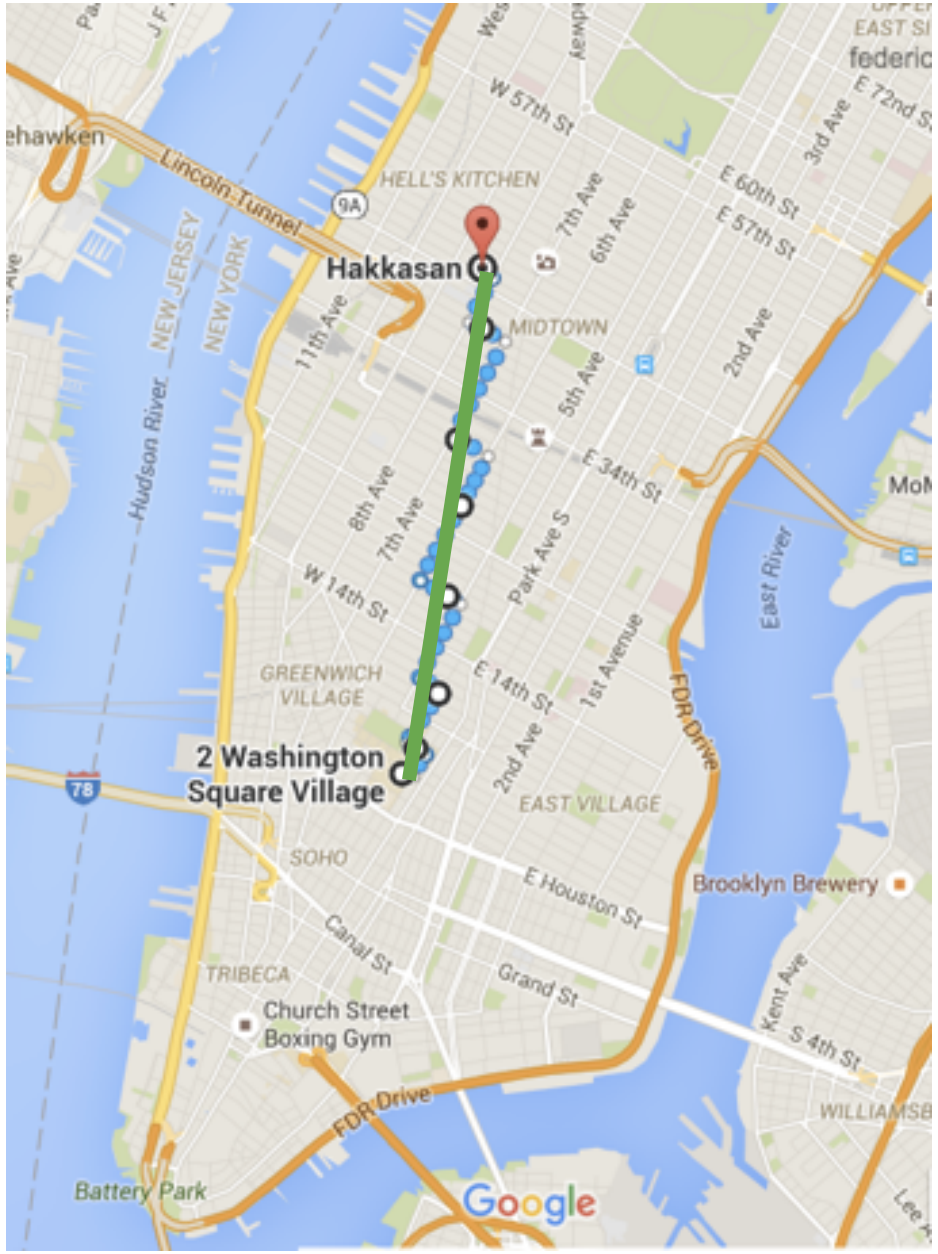
Minkowski family of distances

$$D(i,j) = \sqrt[p]{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^p}$$

Euclidean: p=2

$$D_{Euc}(i,j) = \sqrt{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^2}$$

features: x, y



distance metrics

continuous variables

features

latitude and longitude

$\phi_i, \lambda_i, \phi_j, \lambda_j$

Minkowski family of distances

$$D(i,j) = \sqrt[p]{\sum_{k=1}^{N} |x_{ik} - x_{jk}|^p}$$

Great Circle distance

$$D(i,j) = R \arccos(\sin\phi_i \cdot \sin\phi_j + \cos\phi_i \cdot \cos\phi_j \cdot \cos\Delta\lambda)$$

distance metrics

categorical variables: *binary*

observation *i*

observation *j*

	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

$M_{i=0,j=0}$: number of features in neither

$M_{i=1,j=1}$: number of features in both

$M_{i=1,j=0}$: number of features in *i* but not *j*

$M_{i=0,j=1}$: number of features in *j* but not *i*

Uses presence/absence of features in data

distance metrics

categorical variables: *binary*

observation *i*

observation *j*

	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

Simple Matching Coefficient or Rand similarity

$$SMC(i,j) = \frac{M_{i=0,j=0}+M_{i=1,j=1}}{M_{i=0,j=0}+M_{i=1,j=0}+M_{i=0,j=1}+M_{i=1,j=1}}$$

Simple Matching Distance

$$SMD(i,j) = 1 - SMC(i,j)$$

$M_{i=0,j=0}$: number of features in neither

$M_{i=1,j=1}$: number of features in both

$M_{i=1,j=0}$: number of features in *i* but not *j*

$M_{i=0,j=1}$: number of features in *j* but not *i*

Uses presence/absence of features in data

distance metrics

categorical variables: *binary*

observation *i*

observation *j*

	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

Simple Matching Coefficient or Rand similarity

$$SMC(i,j) = \frac{M_{i=0,j=0}+M_{i=1,j=1}}{M_{i=0,j=0}+M_{i=1,j=0}+M_{i=0,j=1}+M_{i=1,j=1}}$$

Simple Matching Distance

$$SMD(i,j) = 1 - SMC(i,j)$$

$M_{i=0,j=0}$: number of features in neither

$M_{i=1,j=1}$: number of features in both

$M_{i=1,j=0}$: number of features in *i* but not *j*

$M_{i=0,j=1}$: number of features in *j* but not *i*

Uses presence/absence of features in data

distance metrics

categorical variables: *binary*

observation *i*

observation *j*

	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

Simple Matching Coefficient or Rand similarity

$$SMC(i,j) = \frac{M_{i=0,j=0}+M_{i=1,j=1}}{M_{i=0,j=0}+M_{i=1,j=0}+M_{i=0,j=1}+M_{i=1,j=1}}$$

Simple Matching Distance

$$SMD(i,j) = 1 - SMC(i,j)$$

$M_{i=0,j=0}$: number of features in neither

$M_{i=1,j=1}$: number of features in both

$M_{i=1,j=0}$: number of features in *i* but not *j*

$M_{i=0,j=1}$: number of features in *j* but not *i*

Uses presence/absence of features in data

distance metrics

categorical variables: *binary*

observation *i*

observation *j*

	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

Jaccard similarity

$$J(i,j) = \frac{M_{i=1,j=1}}{M_{i=0,j=1}+M_{i=1,j=0}+M_{i=0,j=0}}$$

Jaccard distance

$$D(i,j) = 1 - J(i,j)$$

distance metrics

categorical variables: *binary*

observation *i*

observation *j*

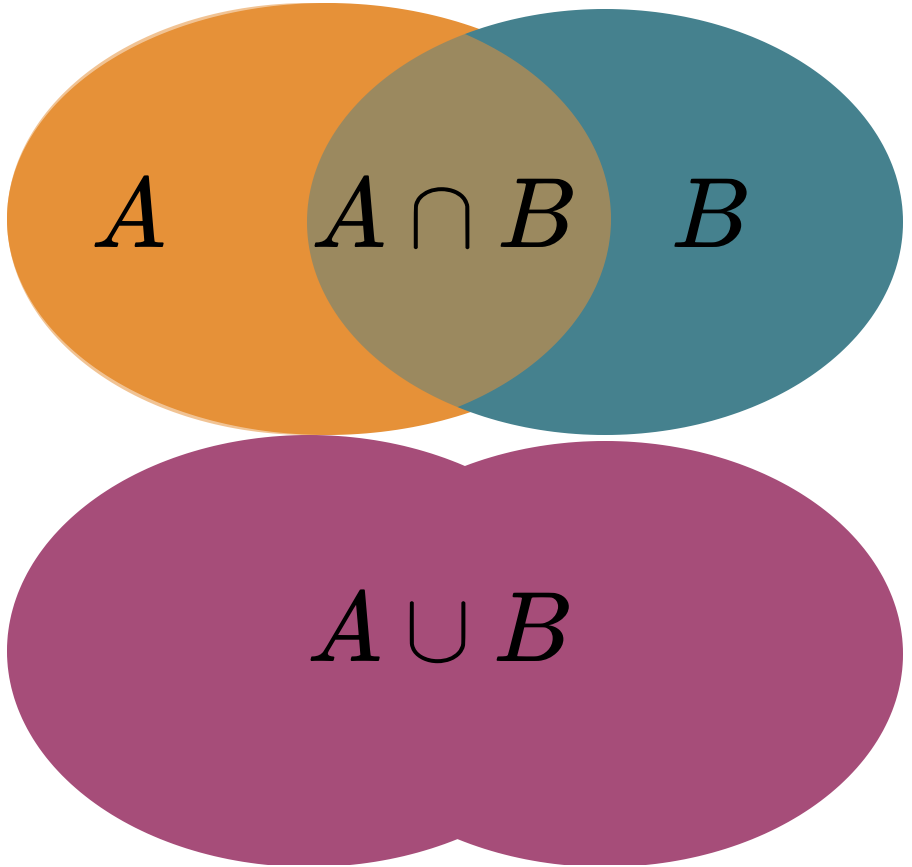
	1	**0**	**sum**
1	M11	M10	M11+M10
0	M01	M00	M01+M00
sum	M11+M01	M10+M00	M11+M00+ M01+ M10

Jaccard similarity

$$J(i,j) = \frac{A \cap B}{A \cup B}$$

Jaccard distance

$$D(i,j) = 1 - J(i,j)$$



distance metrics

categorical variables: *binary*

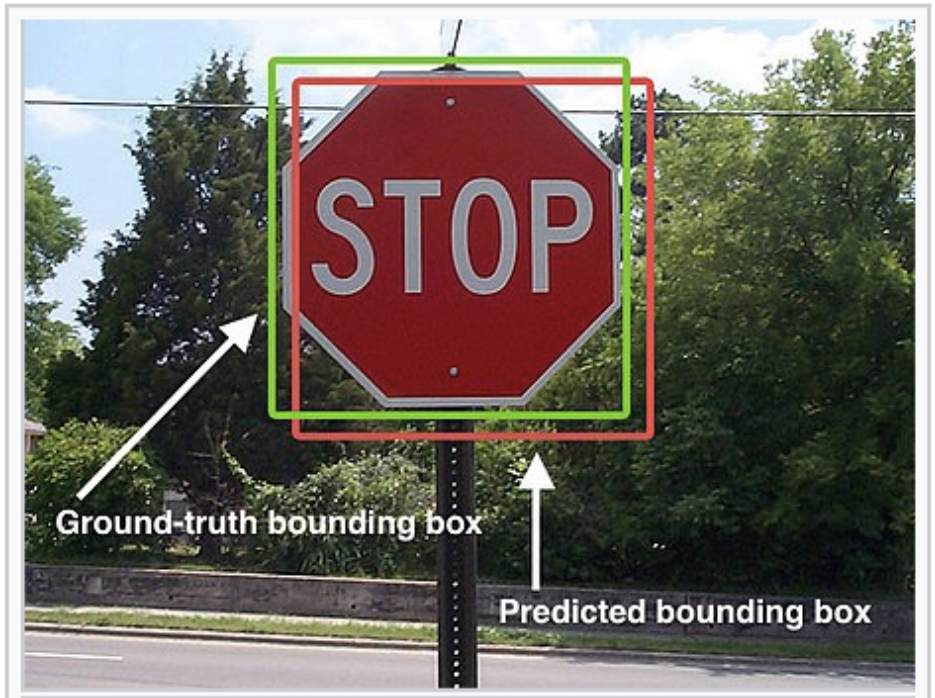
Jaccard similarity

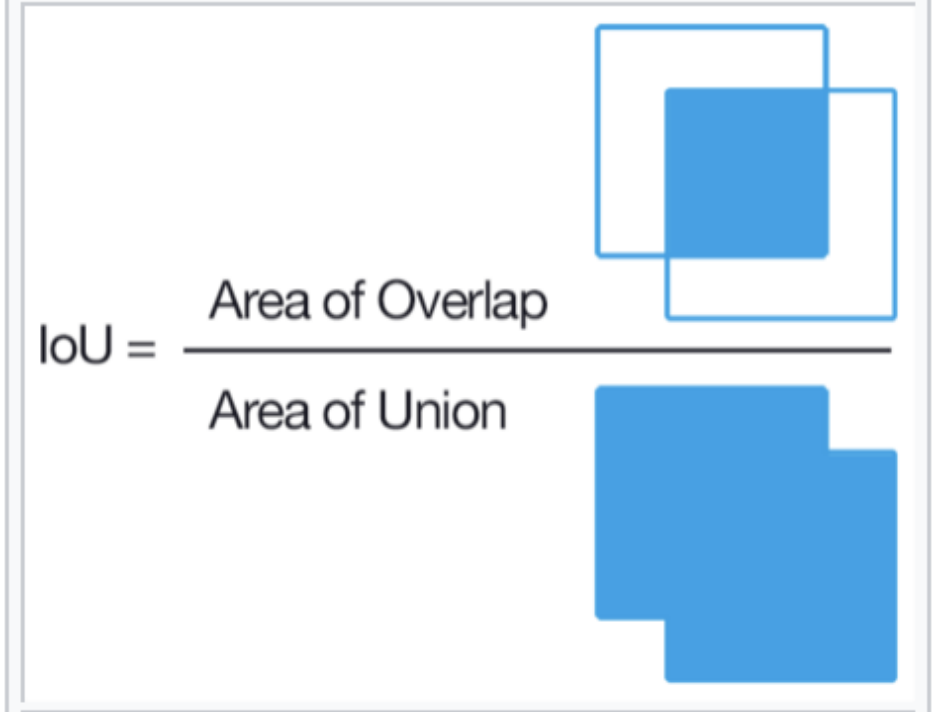
$$J(i,j) = \frac{A \cap B}{A \cup B}$$

Application to Deep Learning for image recognition

Convolutional Neural Nets

https://en.wikipedia.org/wiki/Jaccard_index





distance metrics

https://docs.scipy.org/doc/scipy/reference/spatial.distance.html

SciPy.org

SciPy.org | Docs | SciPy v1.3.1 Reference Guide

Spatial algorithms and data structures (scipy.spatial)

index | modules | next | previous

Distance computations (scipy.spatial.distance)

Function Reference

Distance matrix computation from a collection of raw observation vectors stored in a rectangular array.

pdist(X[, metric])	Pairwise distances between observations in n-dimensional space.

Table of Contents

- Distance computations (scipy.spatial.distar
  - Function Reference

Previous topic

scipy.spatial.procruste

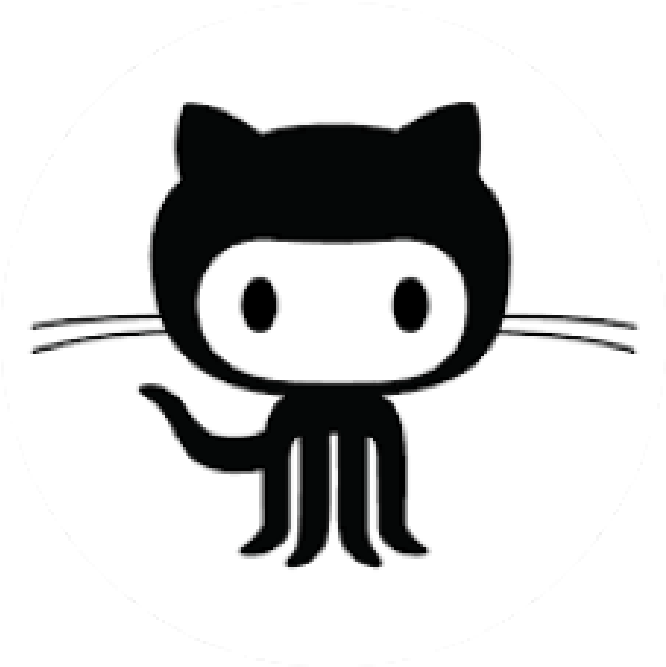
Next topic

Distance functions between two boolean vectors (representing sets) `u` and `v`. As in the case of numerical vectors, `pdist` is more efficient for computing the distances between all pairs.

dice(u, v[, w])	Compute the Dice dissimilarity between two boolean 1-D arrays.
hamming(u, v[, w])	Compute the Hamming distance between two 1-D arrays.
jaccard(u, v[, w])	Compute the Jaccard-Needham dissimilarity between two boolean 1-D arrays.

Distance functions between two numeric vectors `u` and `v`. Computing distances over a large collection of vectors is inefficient for these functions. Use `pdist` for this purpose.

braycurtis(u, v[, w])	Compute the Bray-Curtis distance between two 1-D arrays.
canberra(u, v[, w])	Compute the Canberra distance between two 1-D arrays.
chebyshev(u, v[, w])	Compute the Chebyshev distance.
cityblock(u, v[, w])	Compute the City Block (Manhattan) distance.



https://github.com/fedhere/MLPNS_FBianco/blob/main/clustering/imageProcessingKmeans.ipynb

whitening

3

The term "whitening" refers to white noise, i.e. noise with the same power at all frequencies"

Data can have covariance (and it almost always does!)

axis 0 -> observations

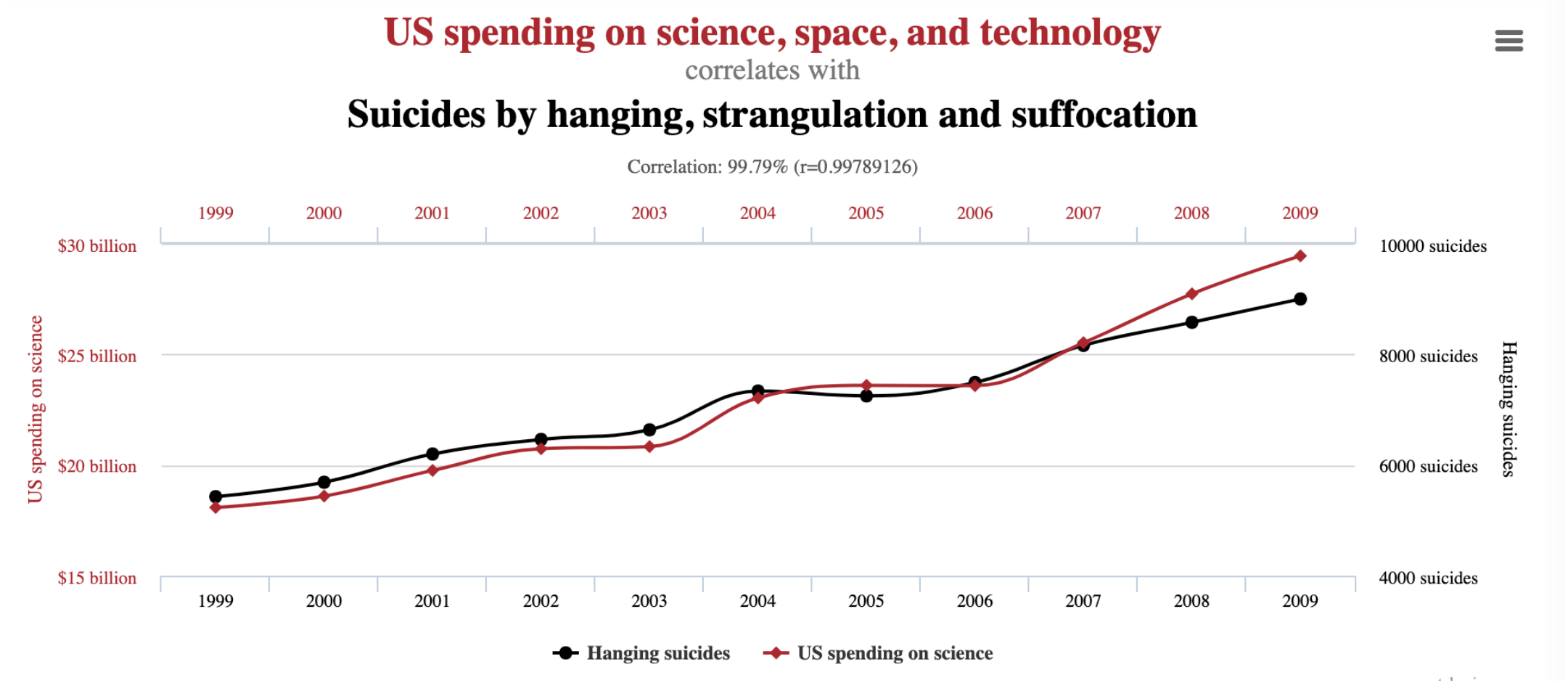
Clicca qui Meteo Italia oggi mercoledì 18 settembre

18/09/2019	TEMPO	PROB.PRECIP.	TEMP.(°C)	UMIDITÀ(%)	VENTO(KM/H)	RAFFICHE(KM/H)
02:00		-	21	57	7	20
05:00		-	20	57	7	16
08:00		-	20	61	5	14
11:00		-	23	55	7	16
14:00		50%	26	58	7	20
17:00		50%	22	62	22	50
20:00		10%	18	77	16	45
23:00		-	17	75	12	32

Bacheca – Parma domani mercoledì 18 settembre – meteoweek.com

axis 1 -> features

Data can have covariance (and it almost always does!)

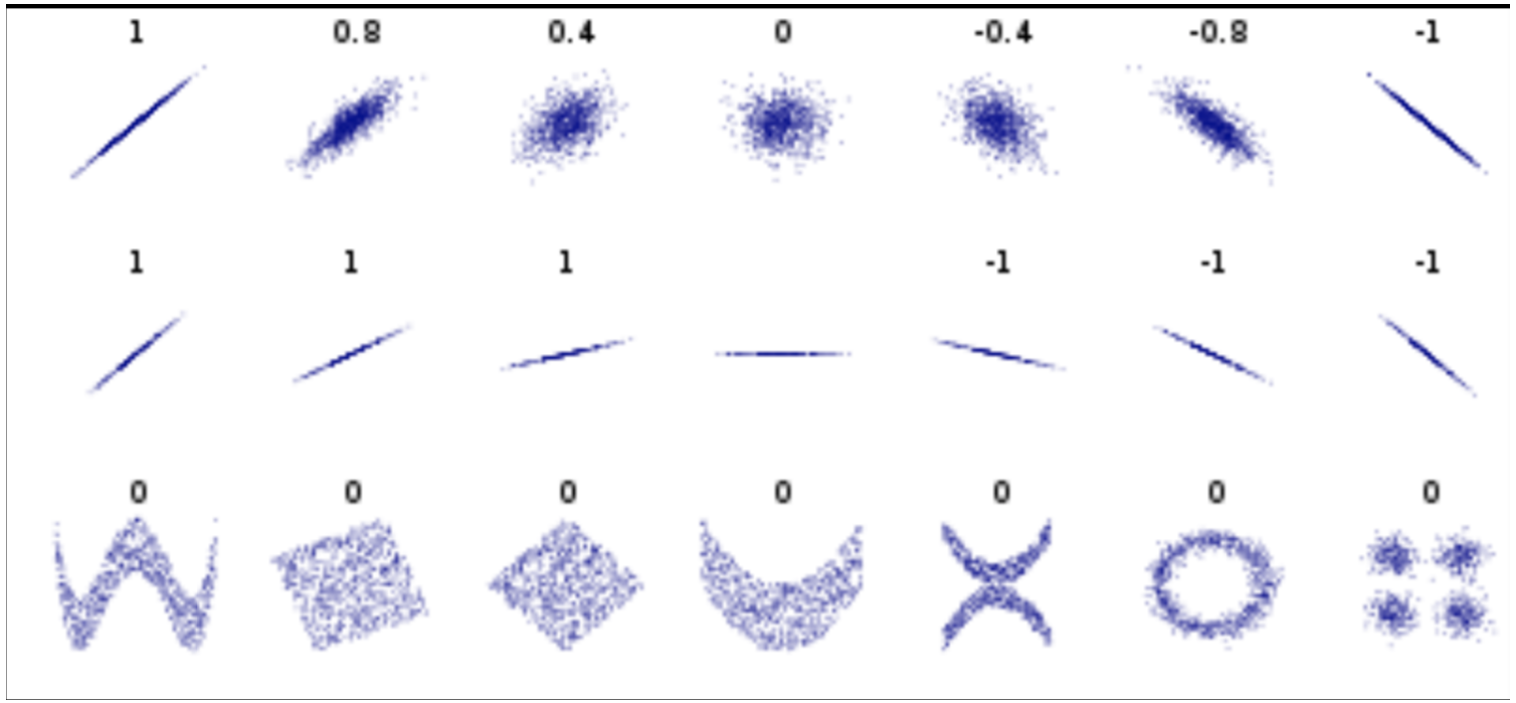


https://www.tylervigen.com/spurious-correlations

Data can have covariance (and it almost always does!)

Pearson's correlation (linear correlation)

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$



correlation = correlation / variance

Data can have covariance (and it almost always does!)

PLUTO Manhattan data (42,000 x 15) correlation matrix

axis 0 -> observations

$$\Sigma = \begin{bmatrix} \mathrm{E}[(X_1 - \mu_1)(X_1 - \mu_1)] & \mathrm{E}[(X_1 - \mu_1)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_1 - \mu_1)(X_n - \mu_n)] \\ \mathrm{E}[(X_2 - \mu_2)(X_1 - \mu_1)] & \mathrm{E}[(X_2 - \mu_2)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_2 - \mu_2)(X_n - \mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n - \mu_n)(X_1 - \mu_1)] & \mathrm{E}[(X_n - \mu_n)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_n - \mu_n)(X_n - \mu_n)] \end{bmatrix}.$$

axis 1 -> features

Data can have covariance (and it almost always does!)

PLUTO Manhattan data (42,000 x 15) correlation matrix

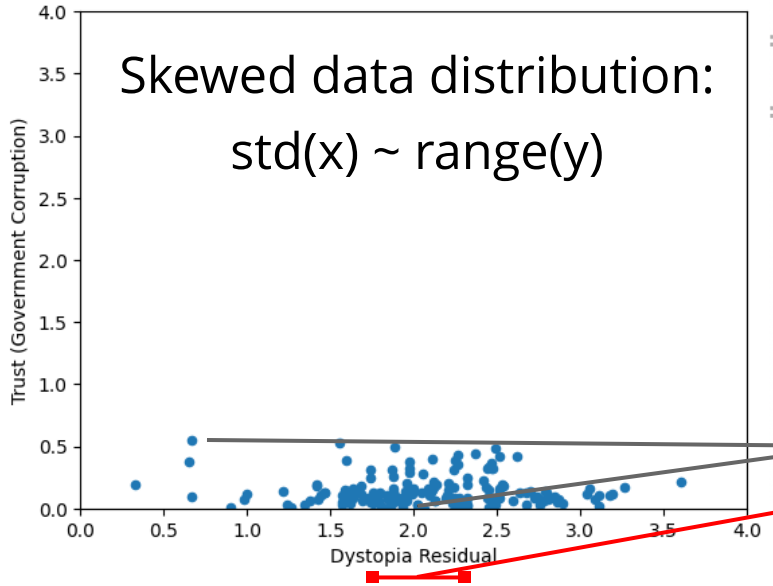
$$\Sigma = \begin{bmatrix} \mathrm{E}[(X_1 - \mu_1)(X_1 - \mu_1)] & 0 & \cdots & 0 \\ 0 & \mathrm{E}[(X_2 - \mu_2)(X_2 - \mu_2)] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & \mathrm{E}[(X_n - \mu_n)(X_n - \mu_n)] \end{bmatrix}.$$

A covariance matrix is diagonal if the data has no correlation

Generic preprocessing... WHY??

Worldbank Happyness Dataset https://github.com/fedhere/MLPNS_FBianco/blob/main/clustering/happiness_solution.ipynb

	Country	Region	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual	year
0	Switzerland	Western Europe	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738	2015
1	Iceland	Western Europe	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201	2015
2	Denmark	Western Europe	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204	2015
3	Norway	Western Europe	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531	2015
4	Canada	North America	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176	2015



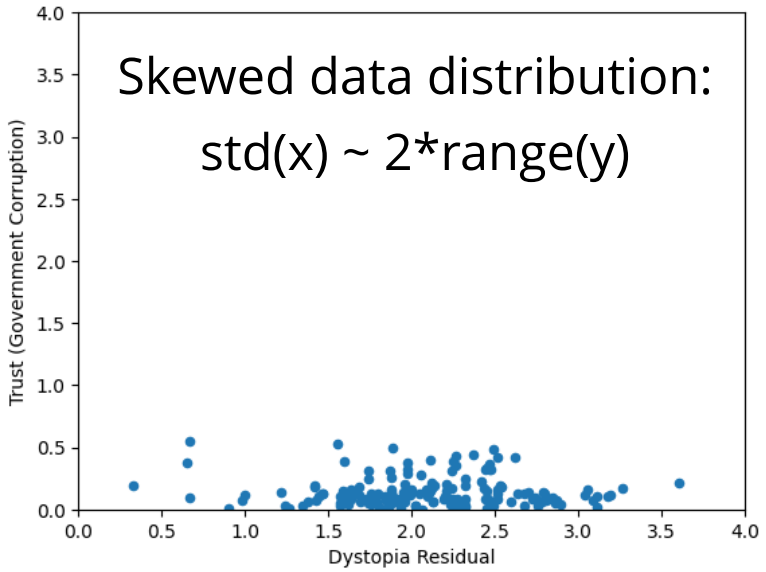

```
happiness15.describe()
```

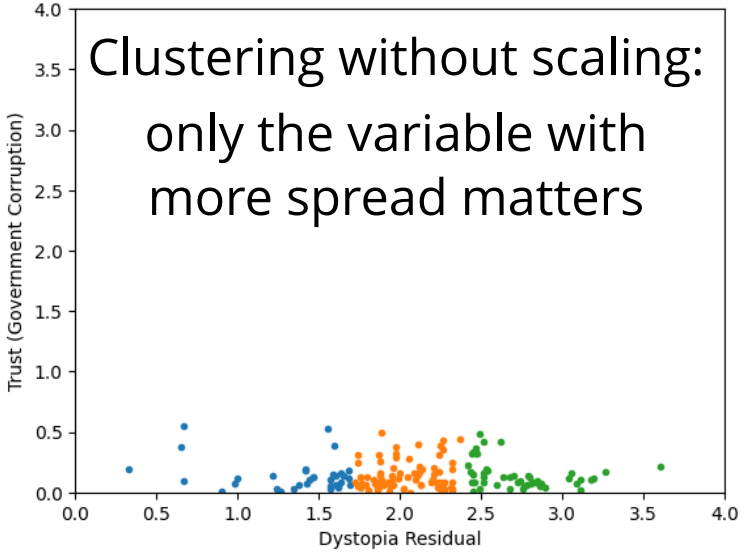
	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual	year
count	160.000000	158.000000	160.000000	158.000000	160.000000	160.000000	160.000000	160.000000	158.000000	160.000000
mean	5.365756	0.047885	0.842979	0.991046	0.628037	0.428151	0.143023	0.236448	2.098977	2015.050000
std	1.141280	0.017146	0.402840	0.272369	0.246332	0.149803	0.119492	0.126605	0.553550	0.445805
min	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.328580	2015.000000
25%	4.517750	0.037268	0.539453	0.856823	0.437857	0.328630	0.061067	0.148800	1.759410	2015.000000
50%	5.203000	0.043940	0.901085	1.029510	0.695745	0.434635	0.107220	0.216130	2.095415	2015.000000
75%	6.193250	0.052300	1.155523	1.214405	0.809837	0.547057	0.179565	0.307547	2.462415	2015.000000
max	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795880	3.602140	2019.000000

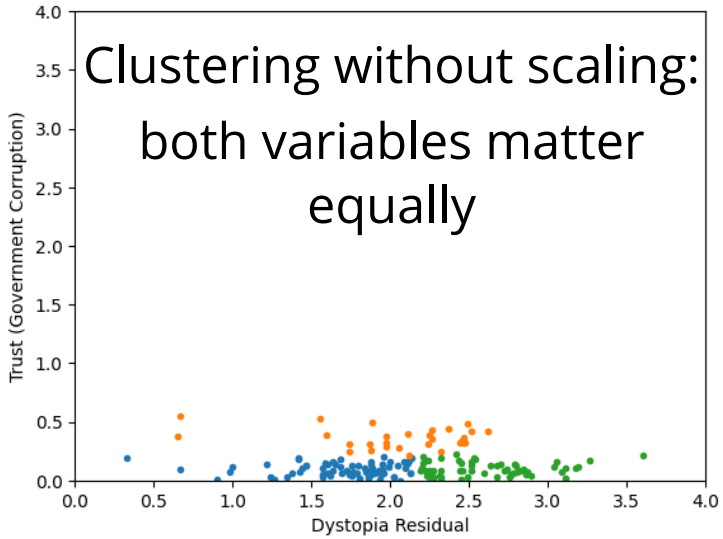
Generic preprocessing... WHY??

Worldbank Happyness Dataset https://github.com/fedhere/MLPNS_FBianco/blob/main/clustering/happiness_solution.ipynb

	Country	Region	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual	year
0	Switzerland	Western Europe	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738	2015
1	Iceland	Western Europe	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201	2015
2	Denmark	Western Europe	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204	2015
3	Norway	Western Europe	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531	2015
4	Canada	North America	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176	2015





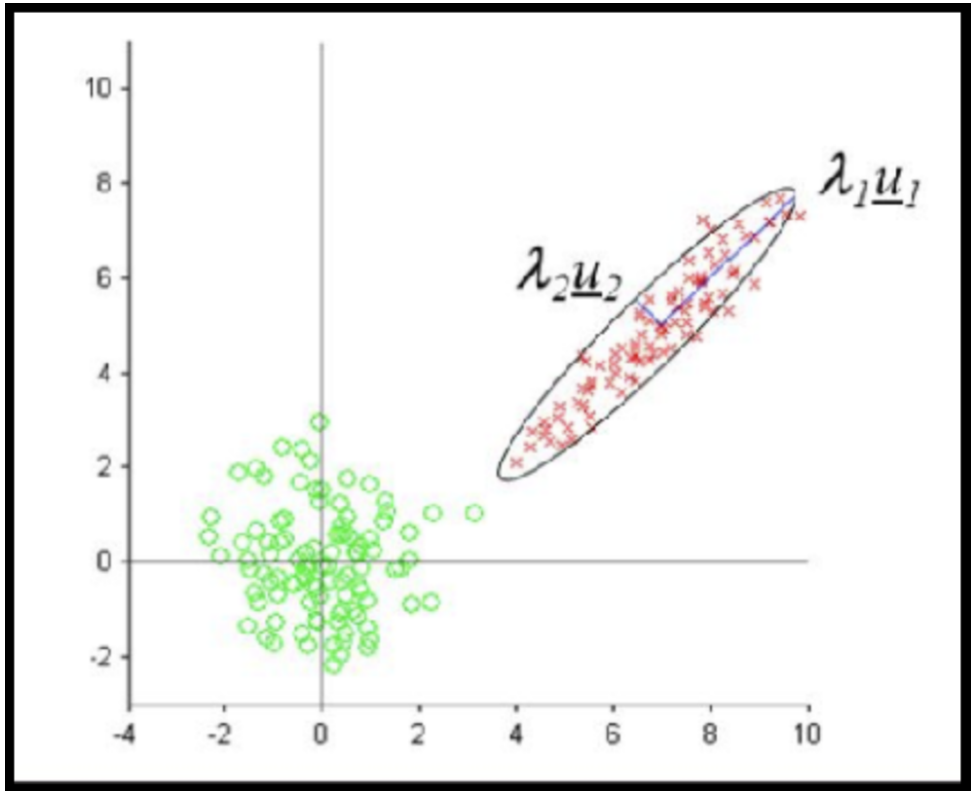


Generic preprocessing

Data can have covariance (and it almost always does!)

ORIGINAL DATA

STANDARDIZED DATA



Data that is not correlated appear as a sphere in the Ndimensional feature space

Generic preprocessing: most commonly, we will just correct for the spread and centroid

for each feature: divide by standard deviation and subtract mean

```
X = preprocessing.scale(X, axis=0)
```

Last executed 2018-12-12 09:35:39 in 46ms

```
X.mean(axis=0)
```

Last executed 2018-12-12 09:35:40 in 13ms

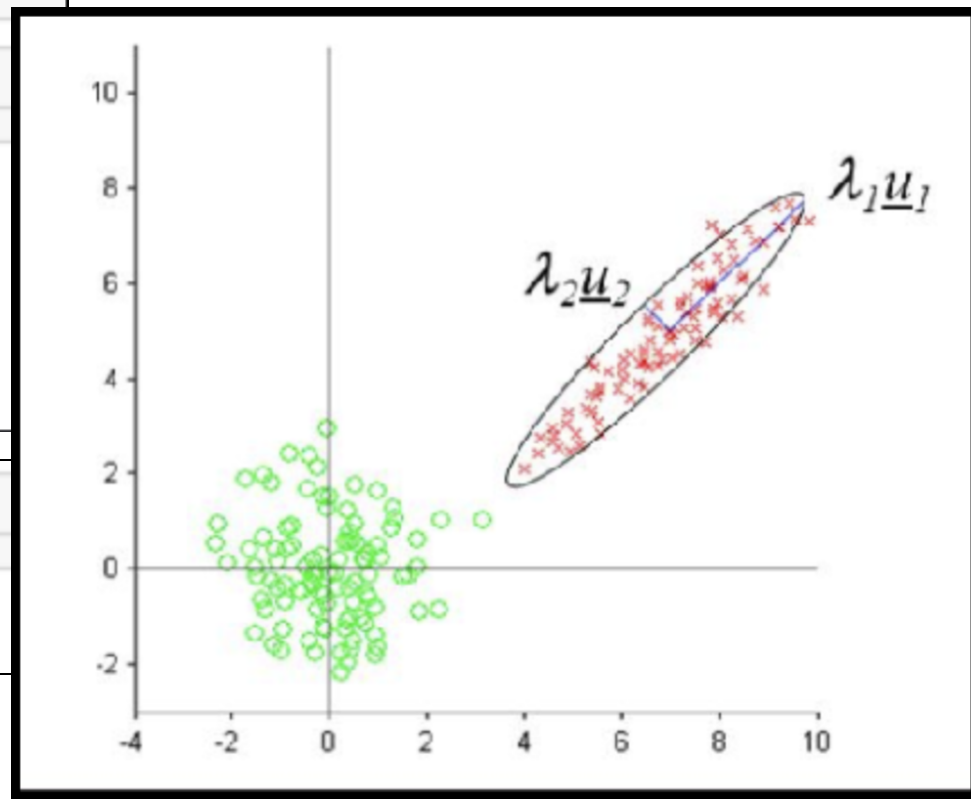
```
array([ 3.85590369e-16, -6.93196168e-17, -5.90549813e-16, -5.95882091e-16,
       -8.49165306e-16, -1.57568821e-15, -8.00508267e-16,  5.55890004e-16,
       -5.16564452e-16,  1.09378357e-15,  3.46598084e-16,  2.31954102e-16,
        2.78611537e-16, -2.51283611e-16,  8.66495210e-18,  3.03939858e-16,
       -3.66594127e-17, -9.27149875e-16, -6.39873386e-16,  2.93275302e-17,
        9.19817992e-17,  6.33208038e-18, -1.99960433e-17,  9.55144336e-16,
       -2.20623011e-16,  6.93196168e-17, -9.46479383e-17,  2.26621824e-16,
        6.93196168e-17,  2.32953905e-16])
```

```
X.std(axis=0)
```

Last executed 2018-12-12 09:36:28 in 19ms

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

mean of each feature should be 0, standard deviation of each feature should be 1



Full On Whitening

: remove covariance by diagonalizing the transforming the data with a matrix that diagonalizes the covariance matrix

find the matrix W that diagonalized Σ

```
from zca import ZCA
import numpy as np
X = np.random.random((10000, 15)) # data
array
trf = ZCA().fit(X)
X_whitened = trf.transform(X)
X_reconstructed =
trf.inverse_transform(X_whitened)
assert(np.allclose(X, X_reconstructed))
```

this is at best hard, in some cases impossible even numerically on large datasets

Generic preprocessing: other common schemes

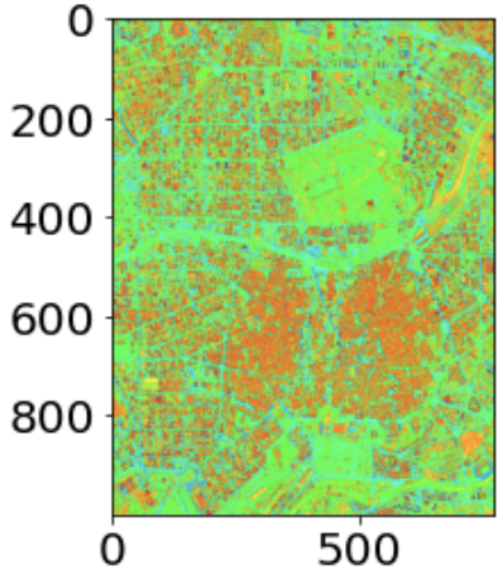
for image processing (e.g. segmentation) often you need to mimmax preprocess

```
1 from sklearn import preprocessing
2 Xopscaled = preprocessing.minmax_scale(op.reshape(op.shape[1] * op.shape[0], 3).astype(float), axis=1)
3 Xopscaled.reshape(op.shape)[200, 700]
```

```
1 array([0.        , 1.        , 0.46428571])
```

```
pl.imshow(Xopscaled.reshape(op.shape));
```

```
Clipping input data to the valid range for imshow with RGB data
```



how clustring works

4

- **Partitioning**
  - **Hard clustering**
  - **Soft Clustering**

K-means (McQueen ’67)

K-medoids (Kaufman & Rausseeuw ’87)

Expectation Maximization (Dempster,Laird,Rubin ’77)

- **Hirarchical**
  - **agglomerative**
  - **devisive**

- **also:**
  - **Density based**
  - **Grid based**
  - **Model based**

5 clustering by partitioning

k-means:

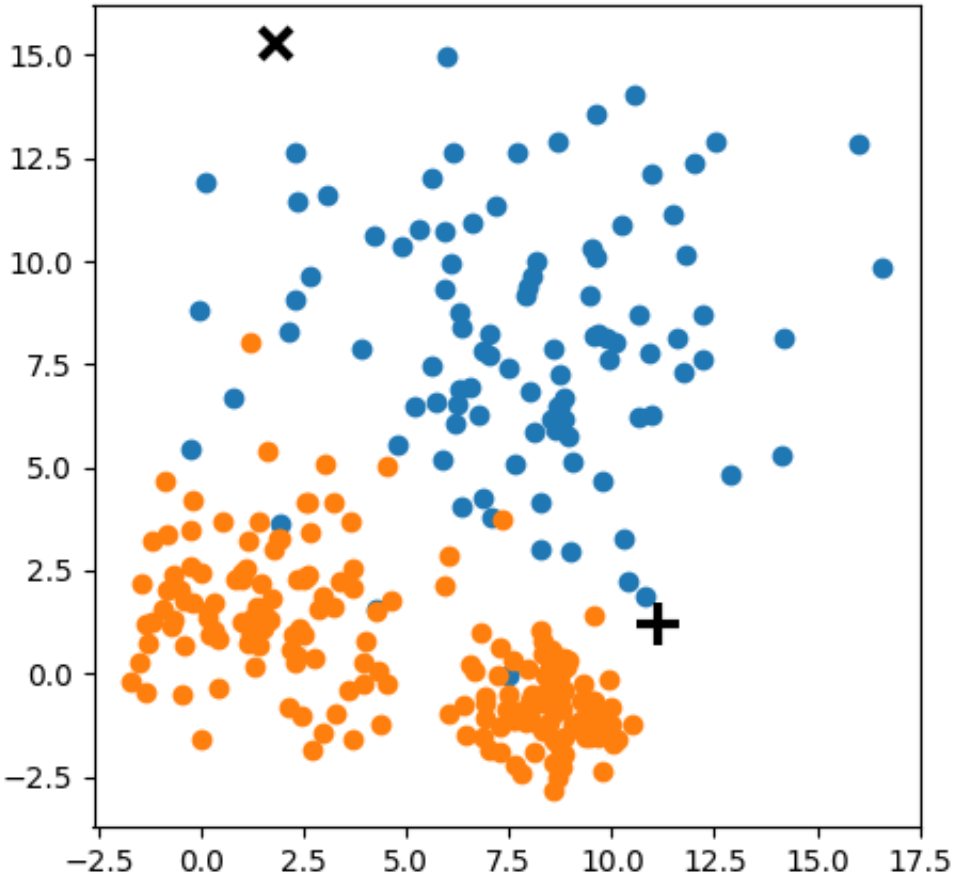
Hard partitioning cluster method

5.1

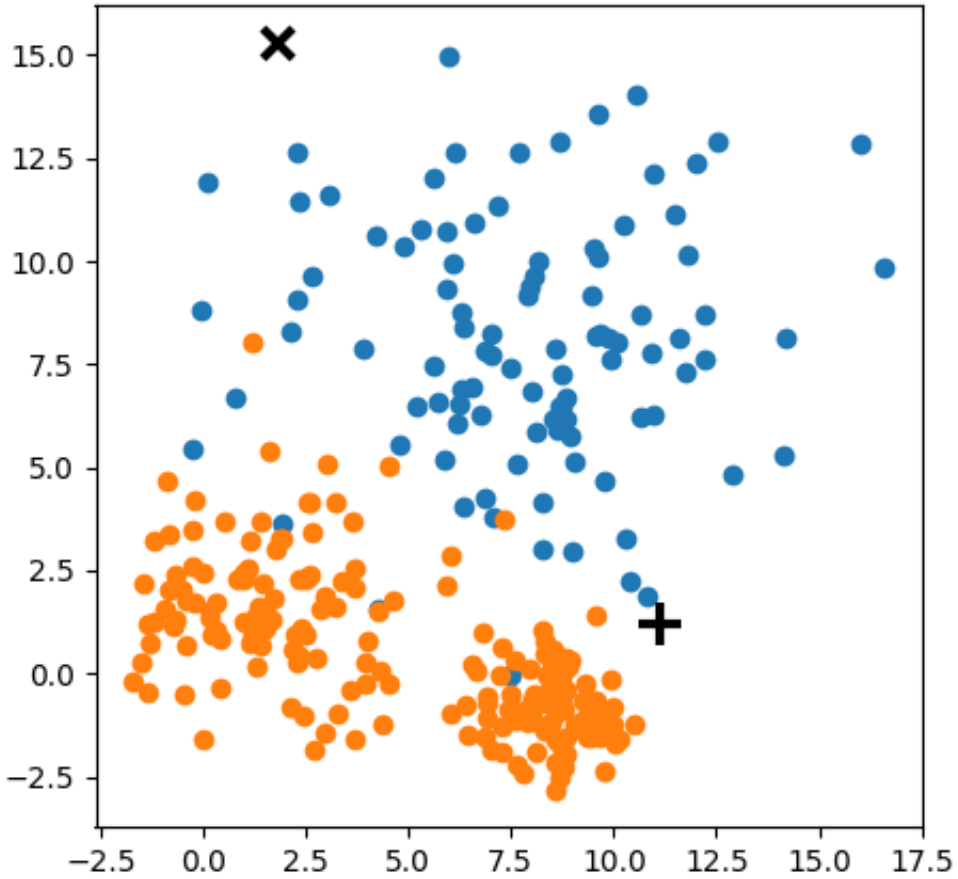
K-means: *the algorithm*

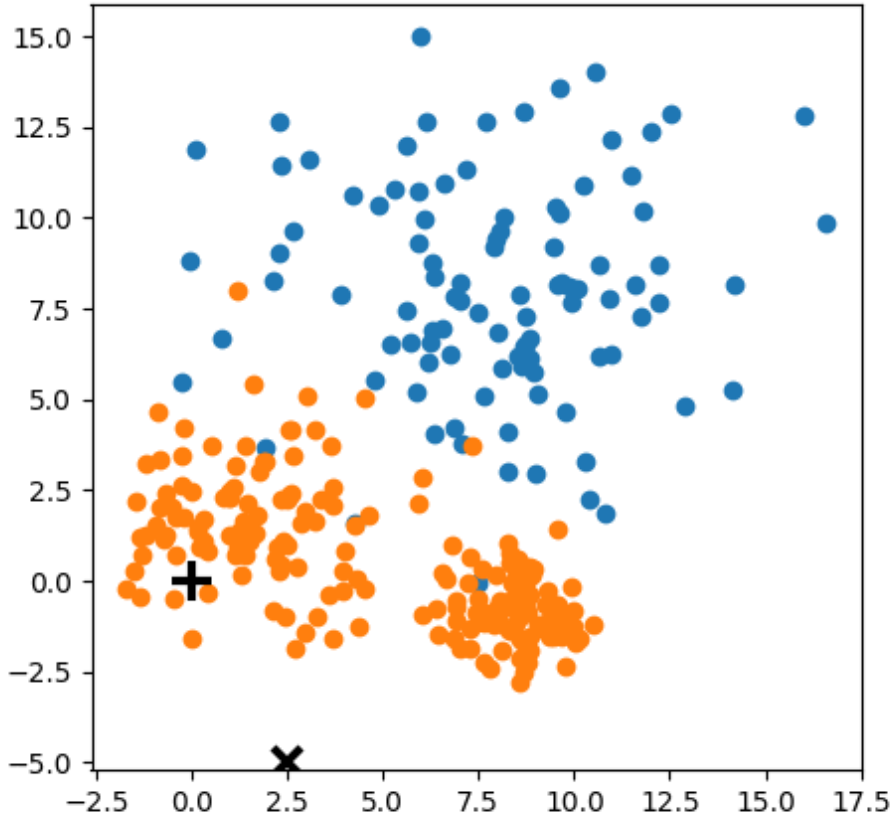
```
Choose N “centers” guesses: random points in the feature
space
repeat:
  Calculate distance between each point and each center

  Assign each point to the closest center

  Calculate the new cluster centers

untill (convergence):
  when clusters no longer change
```

K-means: *the algorithm*



K-means: *the algorithm*





K-means:

Objective: minimizing the aggregate distance within the cluster.

Order: #clusters #dimensions #iterations #datapoints ***O(KdN)***

CONs:

Its non-deterministic: the result depends on the (random) starting point

It only works where the mean is defined: alternative is K-medoids which represents the cluster by its central member (median), rather than by the mean

Must declare the number of clusters upfront (how would you know it?)

PROs:

Scales linearly with *d* and *N*

K-means: *the objective function*

Objective: minimizing the aggregate distance within the cluster.

Order: #clusters #dimensions #iterations #datapoints ***O(KdN)***

O(KdN): complexity scales linearly with

-*d* number of dimensions

-*N* number of datapoints

-*K* number of clusters

K-means: *the objective function*

Objective: minimizing the aggregate distance within the cluster.

total intra-cluster variance $\sum_k \sum_{i \in k} (\vec{x}_i - \vec{\mu}_k)^2$

Order: #clusters #dimensions #iterations #datapoints ***O(KdN)***

Must declare the number of clusters

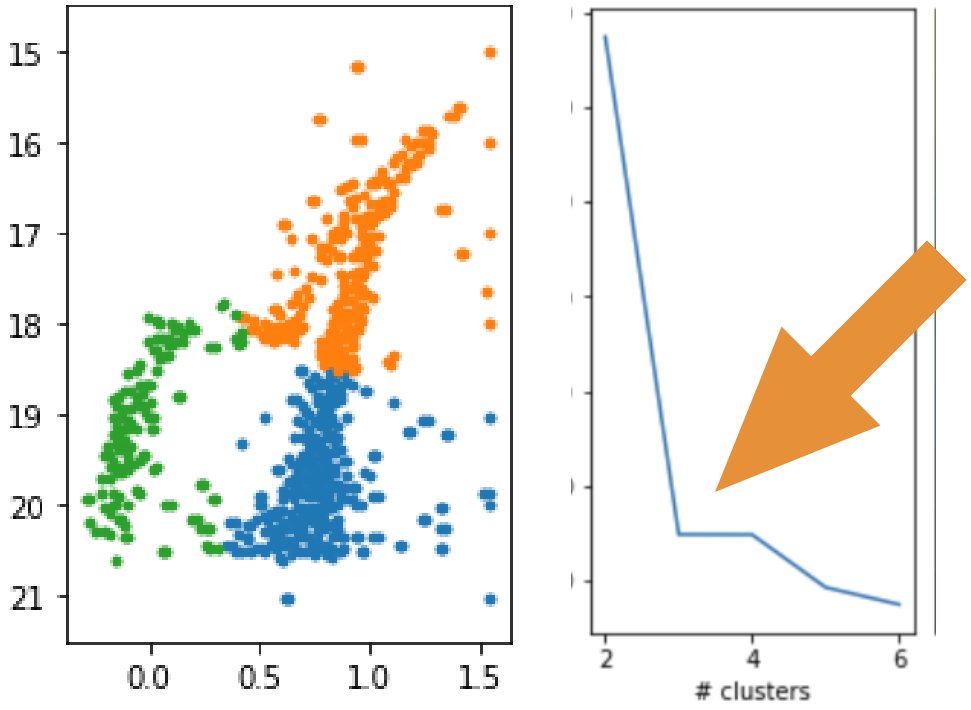
either you know it because of domain knowledge

or

you choose it after the fact: "*elbow method*"

https://github.com/fedhere/DSPS/blob/master/lab10/StellarPopClustersLab.ipynb

```
for i in range(1, nmaxc):
  c = KMeans(n_clusters=i, random_state=302).fit(X2)
  nc.append(c.inertia_)
  print("i.c. variance with {} clusters {:.2f}".format(i, c.inertia_))
pl.plot(range(1, nmaxc), nc)
```



K-means: *the objective function*

Objective: minimizing the aggregate distance within the cluster.

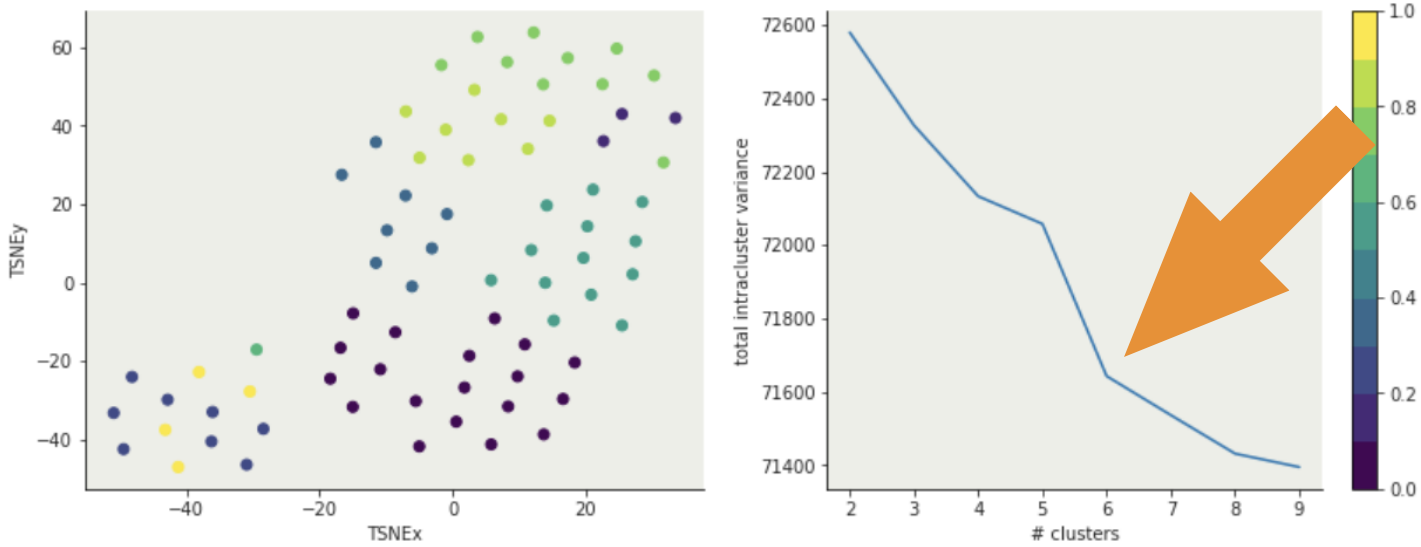
total intra-cluster variance $\sum_k \sum_{i \in k} (\vec{x}_i - \vec{\mu}_k)^2$

Order: #clusters #dimensions #iterations #datapoints ***O(KdN)***

Must declare the number of clusters upfront (how would you know it?)

either domain knowledge or

after the fact: "elbow method"



K-means: *hyperparameters*

class `sklearn.cluster.` **`KMeans`** (*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=None, algorithm='auto'*) ¶

[source]

- *n_clusters* : number of clusters
- *init* : the initial centers or a scheme to choose the center

  'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.

  'random': choose k observations (rows) at random from data for the initial centroids.

  If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.
- *n_init* : if >1 it is effectively an ensamble method: runs n times with different initializations
- *random_state* : for reproducibility

expectation-maximization

Soft partitioning cluster method

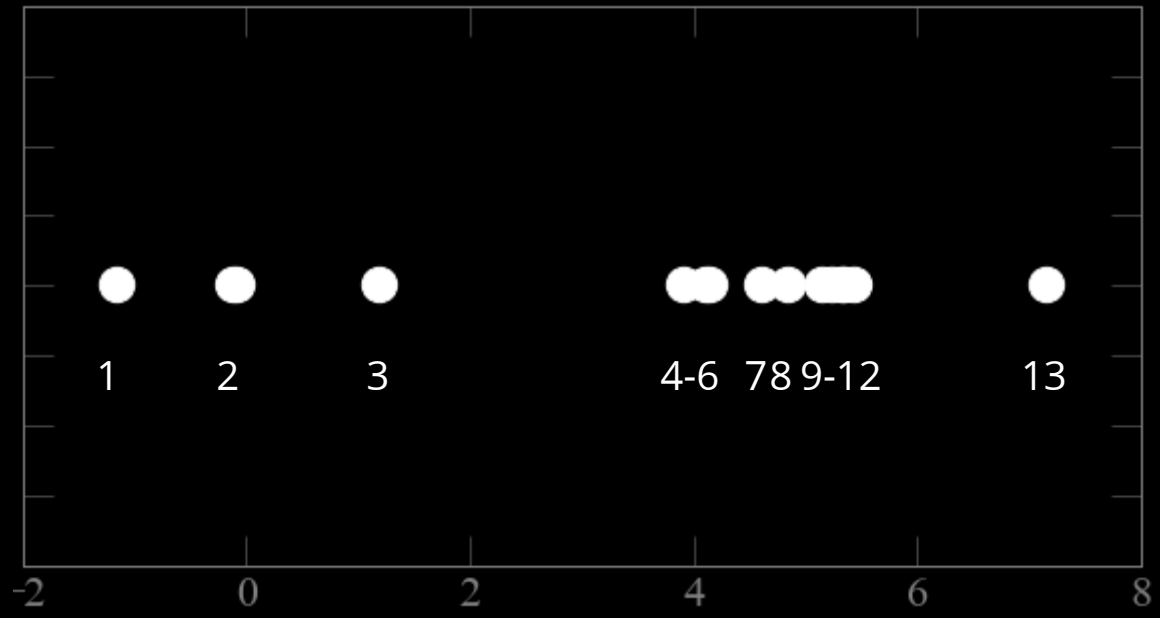
52

Hard clustering : each object in the sample belongs to only 1 cluster

Soft clustering : to each object in the sample we assign a degree of belief that it belongs to a cluster

Soft = probabilistic

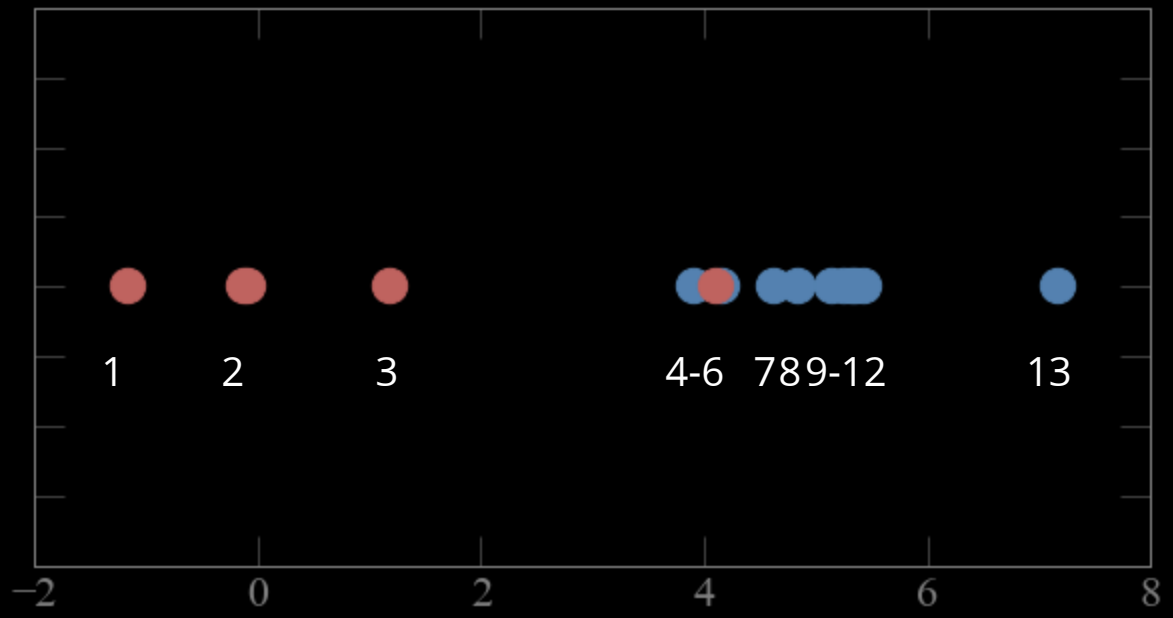
Mixture models



these points come from 2 gaussian distribution.

which point comes from which gaussian?

Mixture models



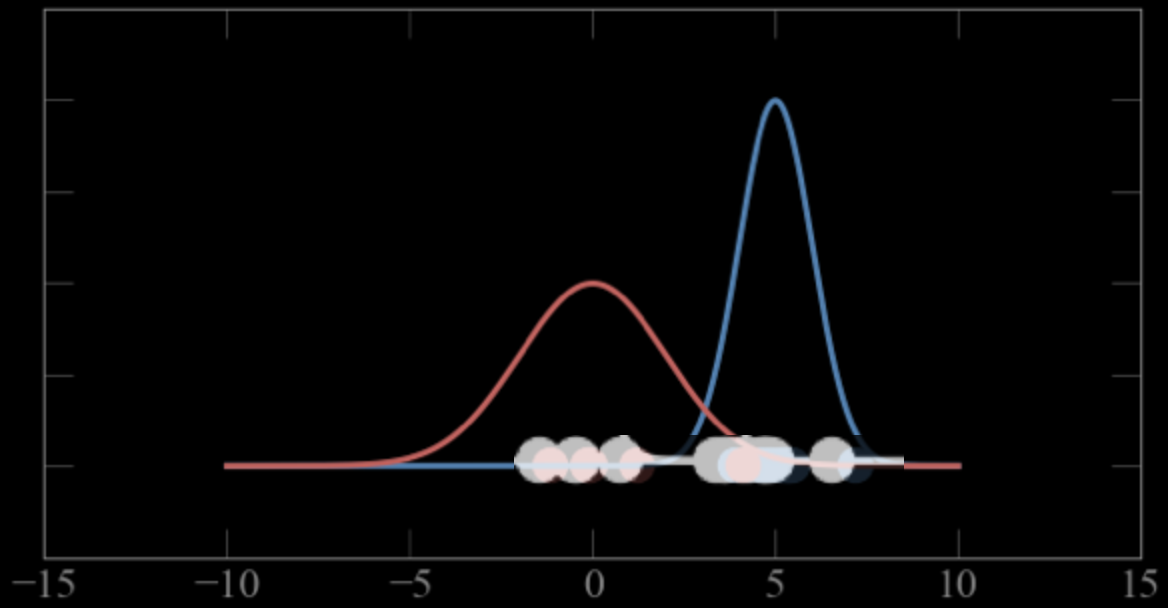
CASE 1:

if i know which point comes from which gaussian

i can solve for the parameters of the gaussian

(e.g. maximizing likelihood)

Mixture models

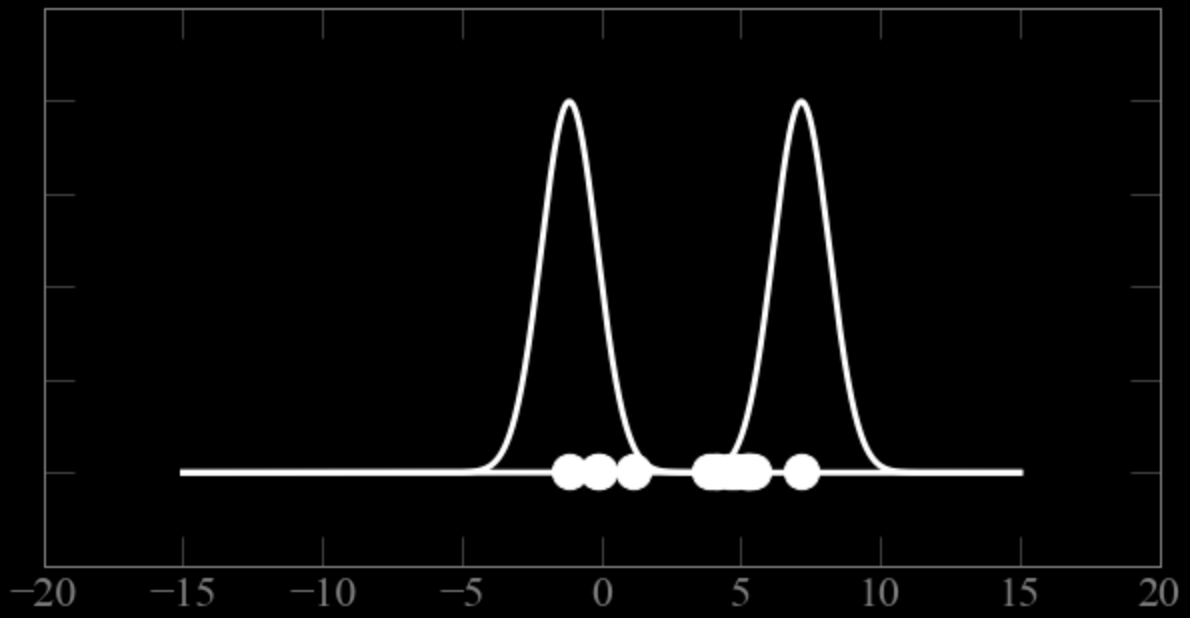


CASE 2:

if i know which the parameters (μ,σ) of the gaussians

i can figure out which gaussian each point is most likely to come from (calculate probability)

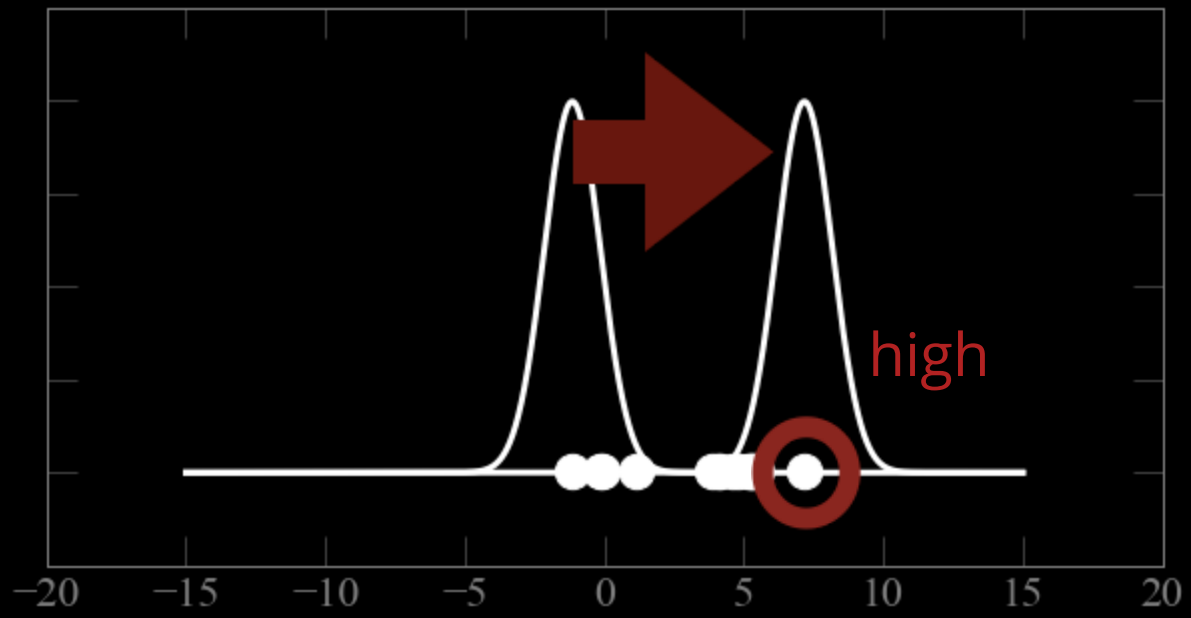
Mixture models:
Expectation maximization



Guess parameters $g = (\mu,\sigma)$ for 2 Gaussian distributions A and B

calculate the probability of each point to belong to A and B

Mixture models:
Expectation maximization

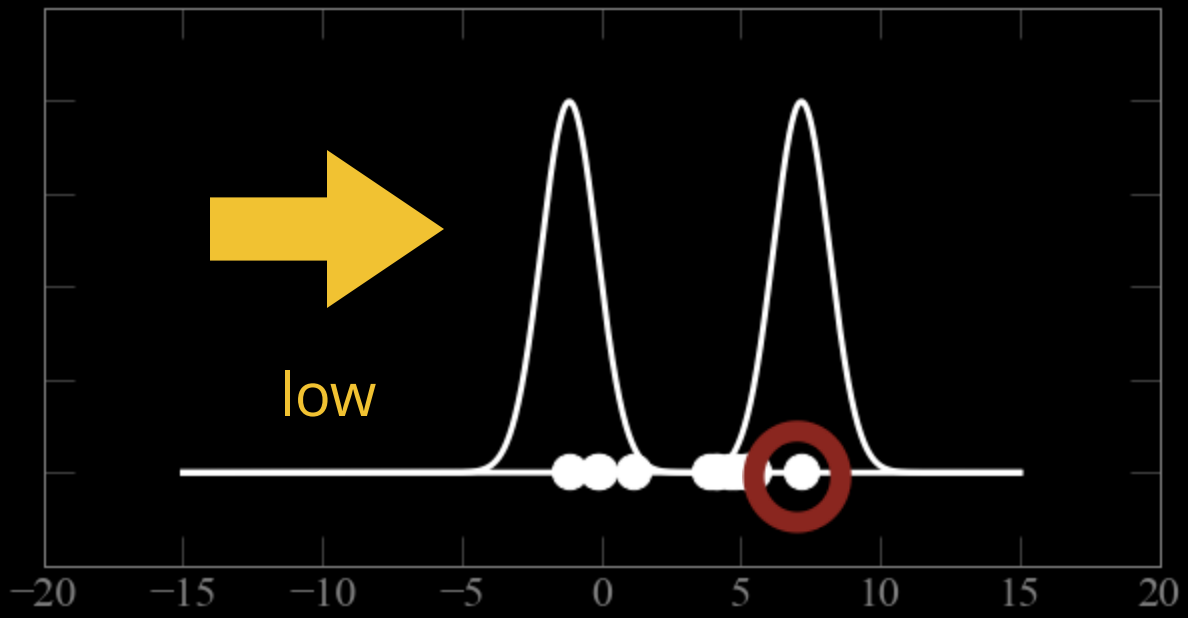


$$P(x_i|\mu_j,\sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{x_i-\mu_j}{2\sigma_j^2}\right)$$

Guess parameters *g*= *(μ,σ)* for 2 Gaussian distributions A and B

calculate the probability of each point to belong to A and B

Mixture models:
Expectation maximization

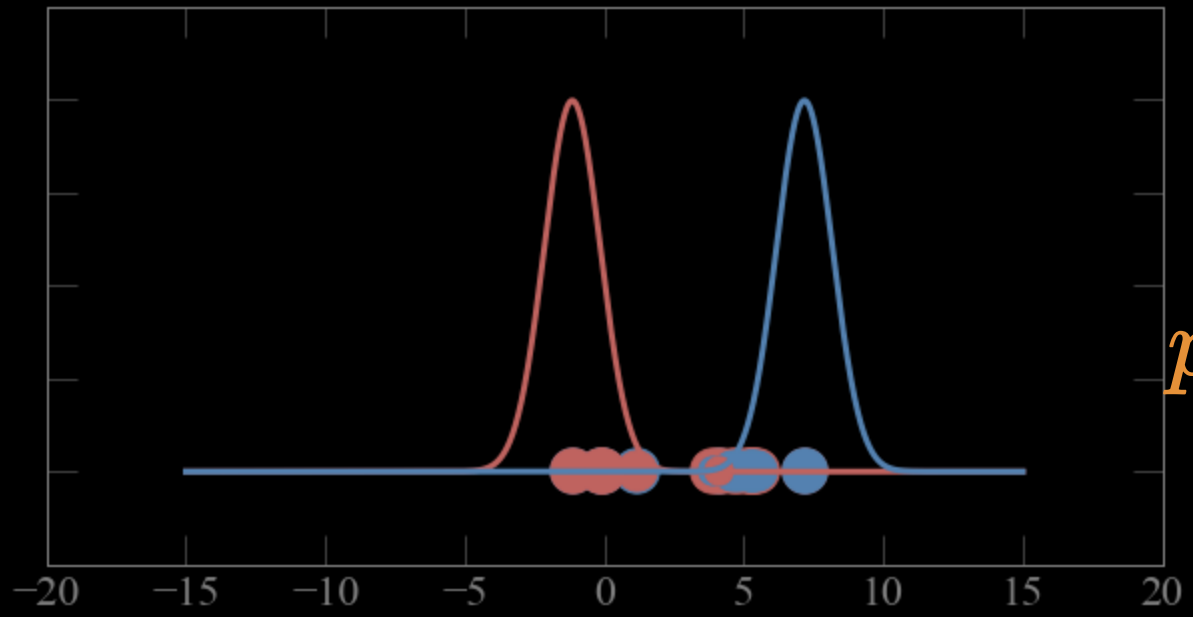


$$P(x_i|\mu_j,\sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{x_i-\mu_j}{2\sigma_j^2}\right)$$

Guess parameters *g*= *(μ,σ)* for 2 Gaussian distributions A and B

calculate the probability of each point to belong to A and B

Mixture models:
Expectation maximization



$$P(x_i|\mu_j,\sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{x_i-\mu_j}{2\sigma_j^2}\right)$$

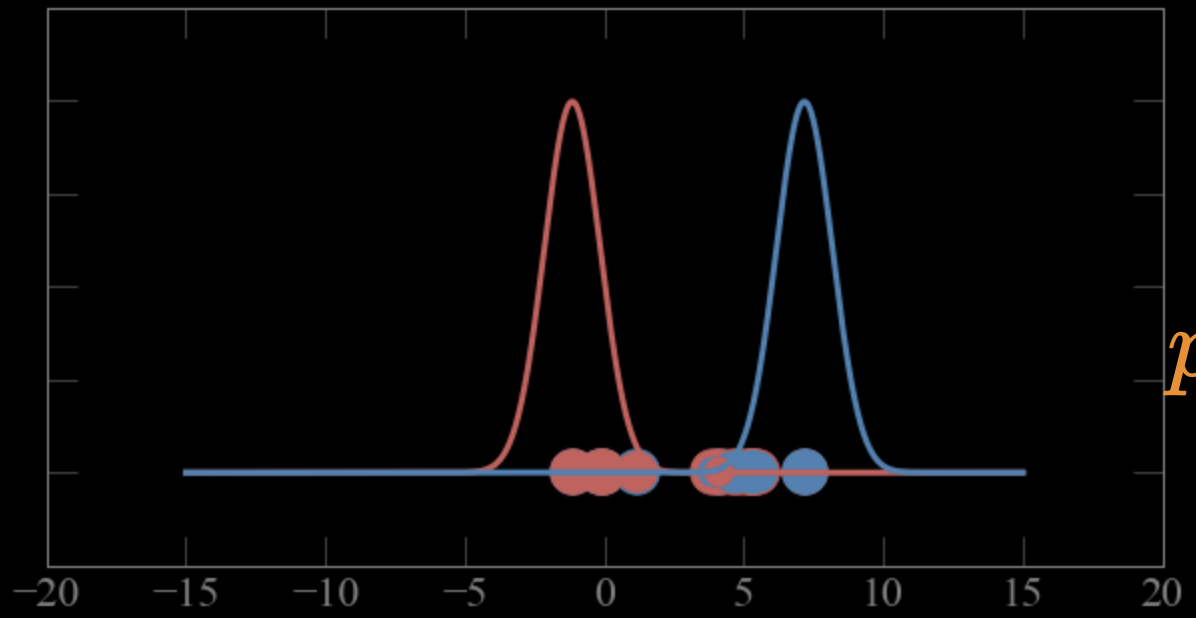
Bayes theorem: P(A|B) = P(B|A) P(A) / P(B)

$$p_{ij} = P(g_1|x_i) = \frac{P(x_i|g_1)|P(g_1)}{P(x_i|g_1)P(g_1)+P(x_i|g_2)P(g_2)}$$

Guess parameters *g*= *(μ,σ)* for 2 Gaussian distributions A and B

1- calculate the probability ***p_ji*** of each point to belong to gaussian ***j***

Mixture models:
Expectation maximization



$$P(x_i|\mu_j,\sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{x_i-\mu_j}{2\sigma_j^2}\right)$$

Bayes theorem: P(A|B) = P(B|A) P(A) / P(B)

$$p_{ij} = P(g_1|x_i) = \frac{P(x_i|g_1)|P(g_1)}{P(x_i|g_1)P(g_1)+P(x_i|g_2)P(g_2)}$$

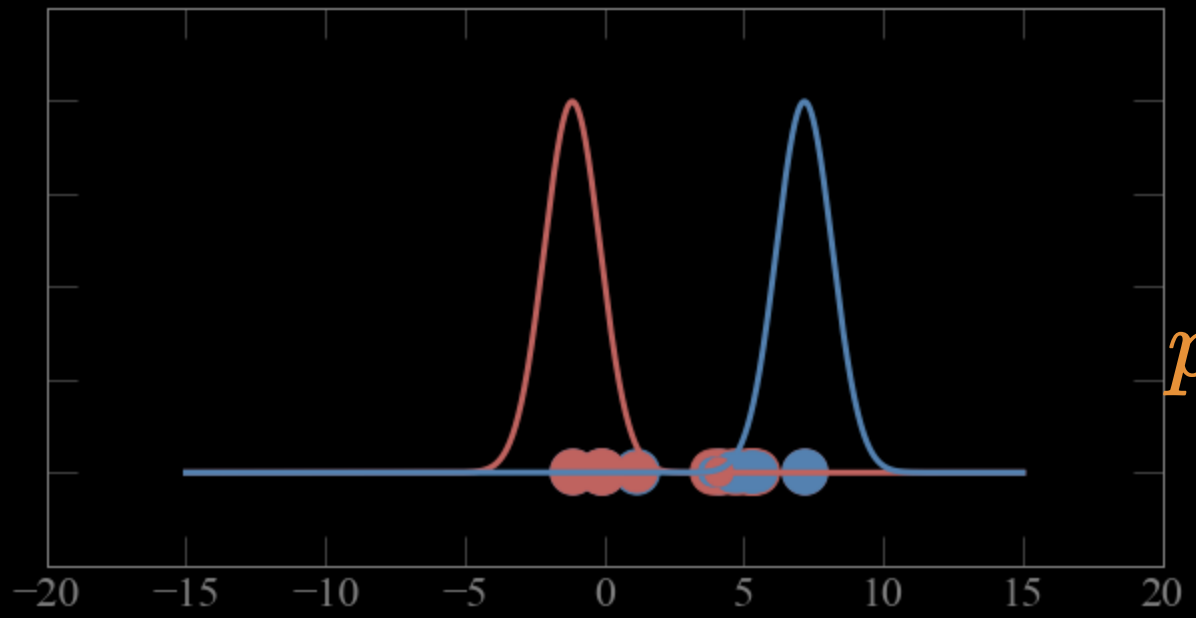
$$\mu_i = \frac{\sum_j P(g_i|x_j)x_j}{\sum_j P(g_i|x_j)} \qquad \sigma_j = \frac{\sum_i P(g_j|x_i)(x_i-\mu_j)^2}{\sum_j P(g_j|x_i)}$$

Guess parameters *g*= *(μ,σ)* for 2 Gaussian distributions A and B

1- calculate the probability ***p_ji*** of each point to belong to gaussian ***j***

2a - calculate the weighted ***mean*** of the cluster, weighted by the *p_ji*

Mixture models:
Expectation maximization



$$P(x_i|\mu_j,\sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{x_i-\mu_j}{2\sigma_j^2}\right)$$

Bayes theorem: P(A|B) = P(B|A) P(A) / P(B)

$$p_{ij} = P(g_1|x_i) = \frac{P(x_i|g_1)|P(g_1)}{P(x_i|g_1)P(g_1)+P(x_i|g_2)P(g_2)}$$

$$\mu_i = \frac{\sum_j P(g_i|x_j)x_j}{\sum_j P(g_i|x_j)} \qquad \sigma_j = \frac{\sum_i P(g_j|x_i)(x_i-\mu_j)^2}{\sum_j P(g_j|x_i)}$$

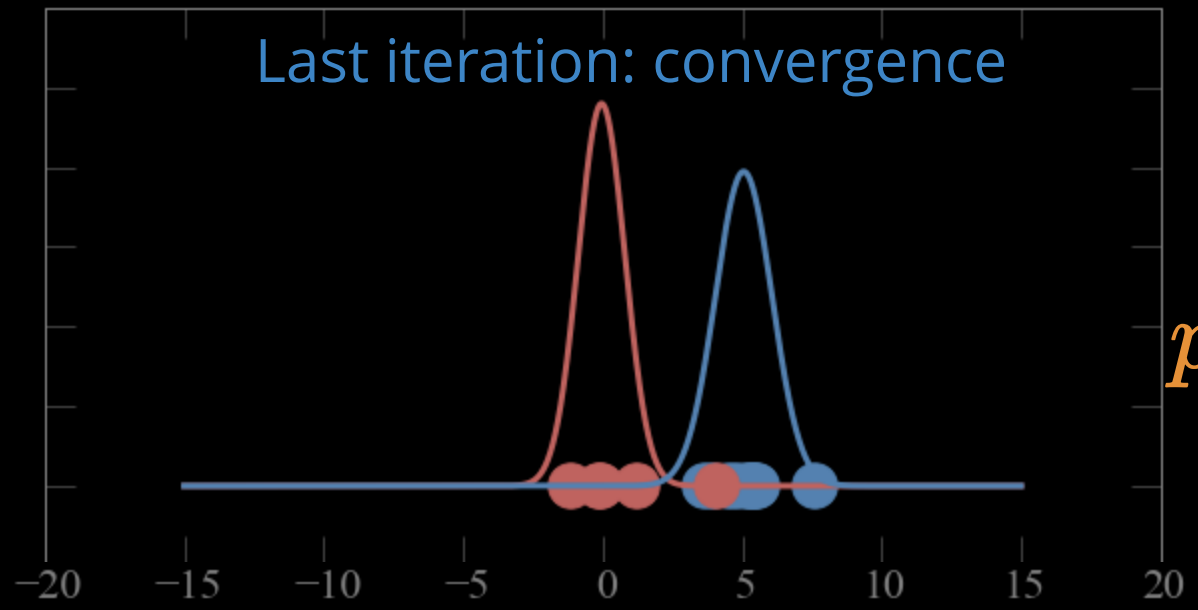
Guess parameters *g*= *(μ,σ)* for 2 Gaussian distributions A and B

1- calculate the probability ***p_ji*** of each point to belong to gaussian ***j***

2a - calculate the weighted ***mean*** of the cluster, weighted by the *p_ji*

2b - calculate the weighted ***sigma*** of the cluster, weighted by the *p_ji*

Mixture models:
Expectation maximization



$$P(x_i|\mu_j,\sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{x_i-\mu_j}{2\sigma_j^2}\right)$$

Bayes theorem: P(A|B) = P(B|A) P(A) / P(B)

$$p_{ij} = P(g_1|x_i) = \frac{P(x_i|g_1)|P(g_1)}{P(x_i|g_1)P(g_1)+P(x_i|g_2)P(g_2)}$$

$$\mu_i = \frac{\sum_j P(g_i|x_j)x_j}{\sum_j P(g_i|x_j)} \qquad \sigma_j = \frac{\sum_i P(g_j|x_i)(x_i-\mu_j)^2}{\sum_j P(g_j|x_i)}$$

Alternate expectation and maximization step till convergence

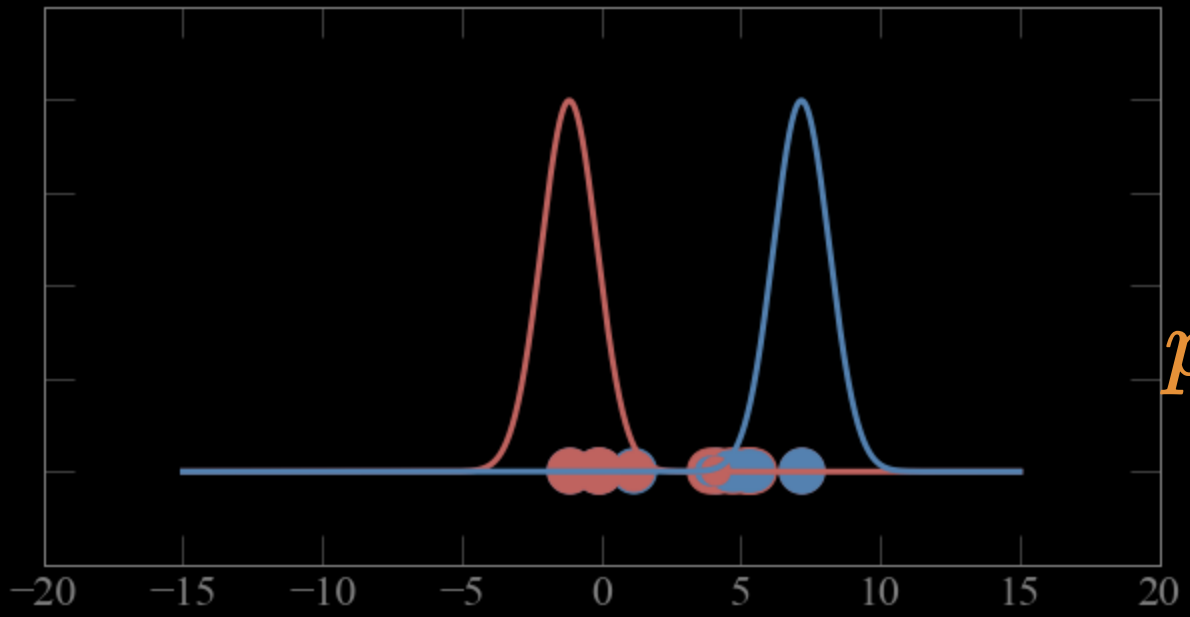
1- calculate the probability ***p_ji*** of each point to belong to gaussian ***j***

2a - calculate the weighted ***mean*** of the cluster, weighted by the *p_ji*

2b - calculate the weighted ***sigma*** of the cluster, weighted by the *p_ji*

} *expectation step* ↓ *maximization step* ↻

Mixture models:
Expectation maximization



$$P(x_i|\mu_j,\sigma_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{x_i-\mu_j}{2\sigma_j^2}\right)$$

Bayes theorem: P(A|B) = P(B|A) P(A) / P(B)

$$p_{ij} = P(g_1|x_i) = \frac{P(x_i|g_1)|P(g_1)}{P(x_i|g_1)P(g_1)+P(x_i|g_2)P(g_2)}$$

$$\mu_i = \frac{\sum_j P(g_i|x_j)x_j}{\sum_j P(g_i|x_j)} \qquad \sigma_j = \frac{\sum_i P(g_j|x_i)(x_i-\mu_j)^2}{\sum_j P(g_j|x_i)}$$

Alternate expectation and maximization step till convergence

1- calculate the probability ***p_ji*** of each point to belong to gaussian ***j***

2a - calculate the weighted ***mean*** of the cluster, weighted by the *p_ji*

2b - calculate the weighted ***sigma*** of the cluster, weighted by the *p_ji*

} *maximization step*

expectation step ↓

↻

EM: *the algorithm*

Choose N "centers" guesses (like in K-means)

repeat

Expectation step: Calculate the probability of each distribution given the points

Maximization step: Calculate the new centers and variances as weighted averages of the datapoints, weighted by the probabilities

untill (convergence)

Expectatin Maximization:

Order: #clusters #dimensions #iterations #datapoints #parameters

O(KdNp) (>K-means)

based on Bayes theorem

Its non-deterministic: the result depends on the (random) starting point (like K-mean)

It only works where a probability distribution for the data points can be defines (or equivalently a likelihood) (like K-mean)

Must declare the number of clusters and the shape of the pdf upfront (like K-mean)

Convergence Criteria

General

Any time you have an objective function (or loss function) you need to set up a tolerance : if your objective function did not change by more than ε since the last step you have reached convergence (i.e. you are satisfied)

ε is your tolerance

For clustering:

convergence can be reached if

no more than n data point changed cluster

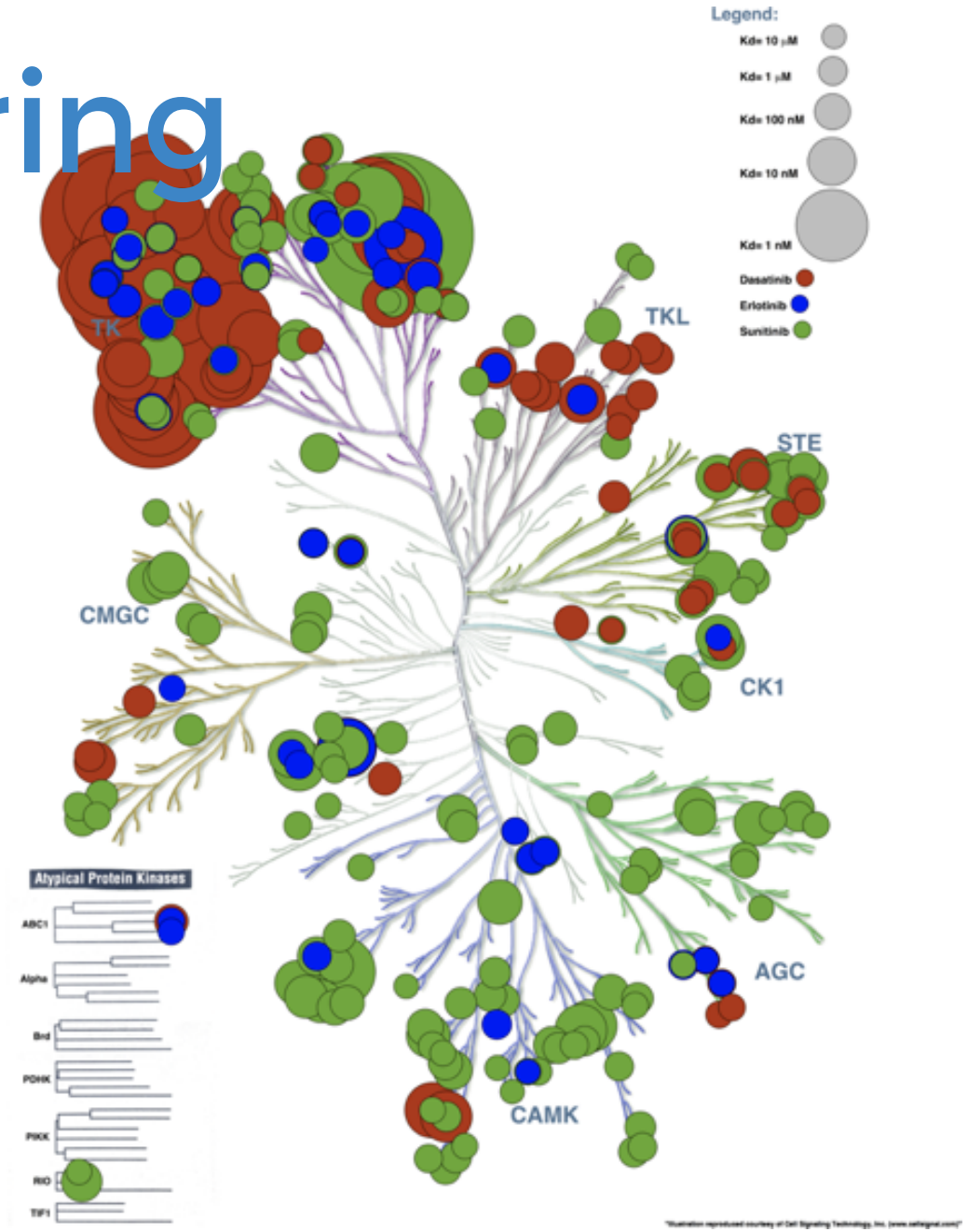
n is your tolerance

Hierarchical clustering

6

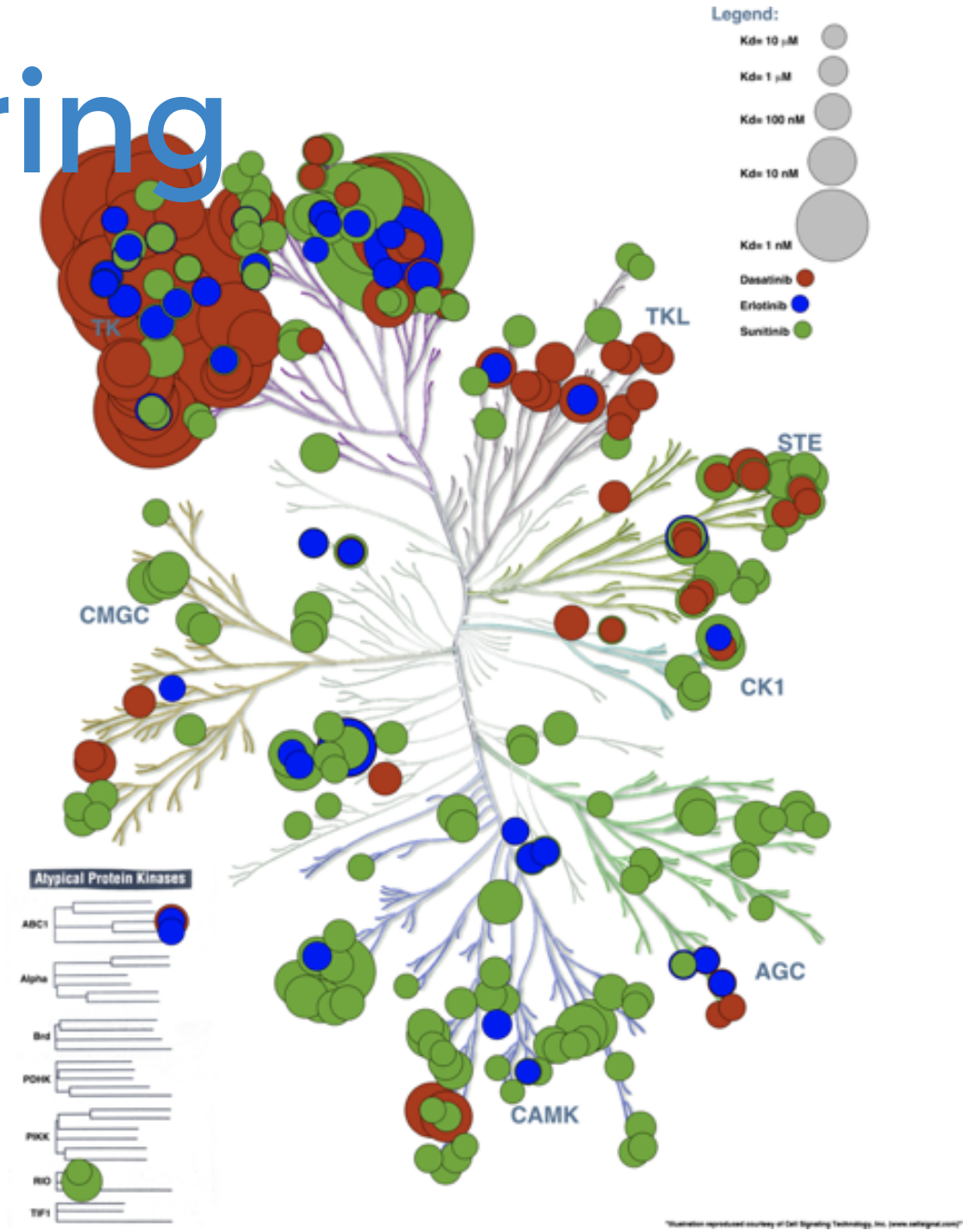
Hierarchical clustering

removes the issue of deciding K (number of clusters)



Hierarchical clustering

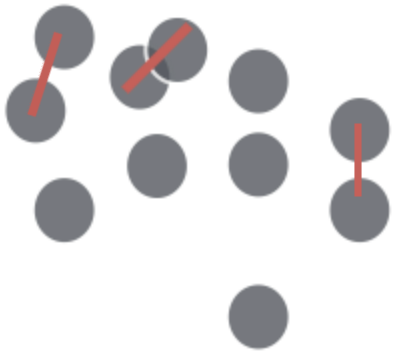
it calculates distance between clusters and single points: *linkage*



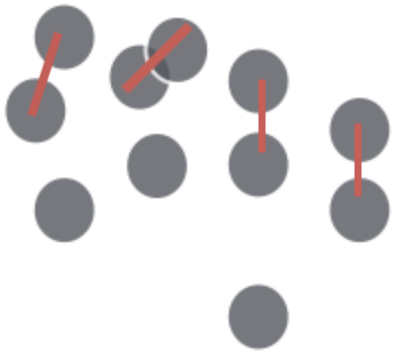
6.1 Agglomerative hierarchical clustering

Hierarchical clustering
agglomerative (bottom up)

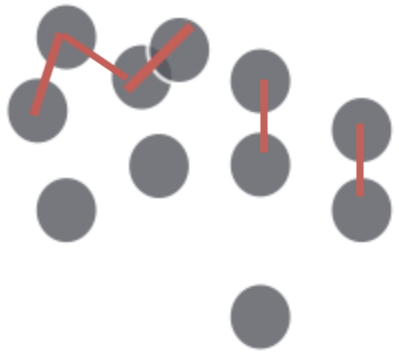
Hierarchical clustering
agglomerative (bottom up)

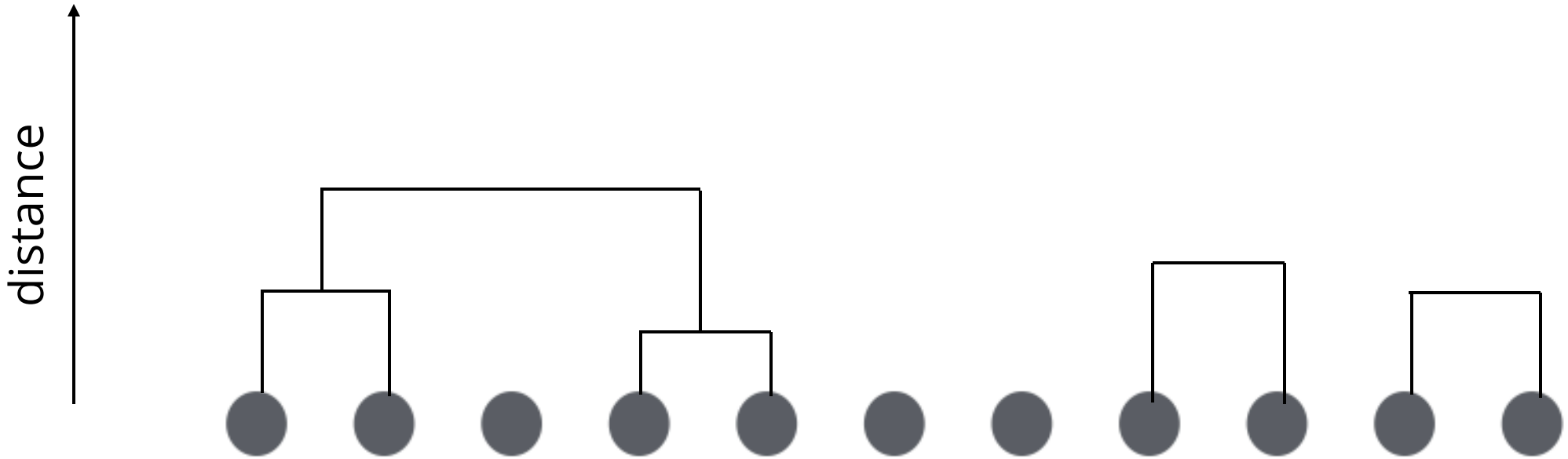


Hierarchical clustering
agglomerative (bottom up)



Hierarchical clustering
agglomerative (bottom up)



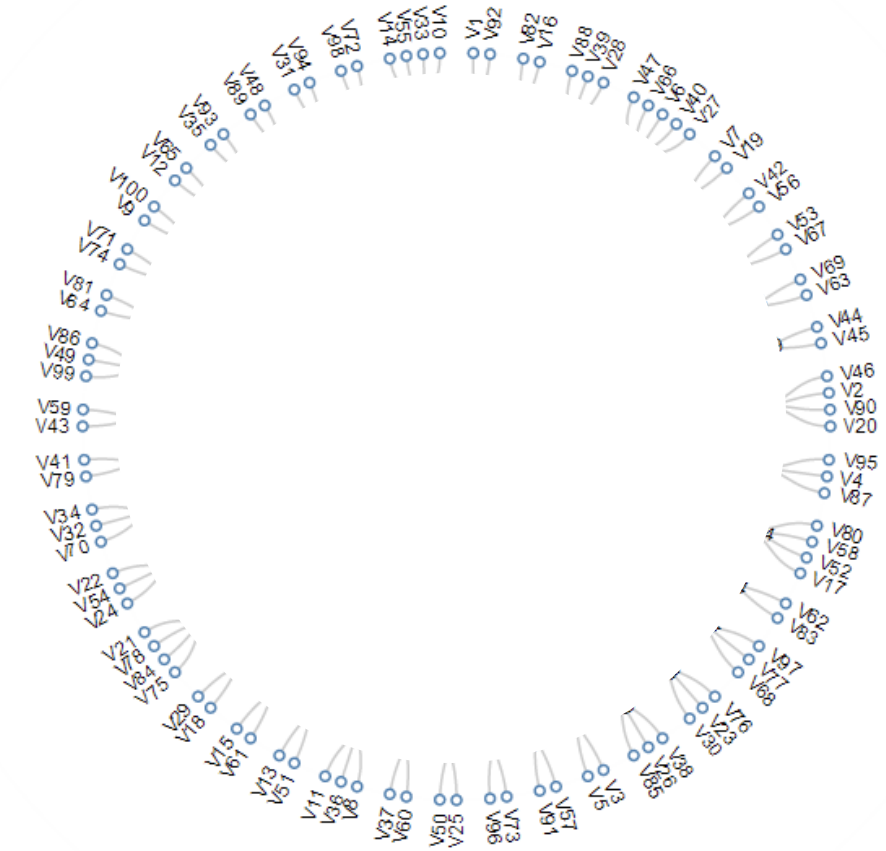


Agglomerative clustering: *the algorithm*

```
compute the distance matrix
each data point is a singleton cluster
repeat
    merge the 2 cluster with minimum distance
    update the distance matrix
untill
    only a single (n) cluster(s) remains
```

Hierarchical clustering agglomerative (bottom up)

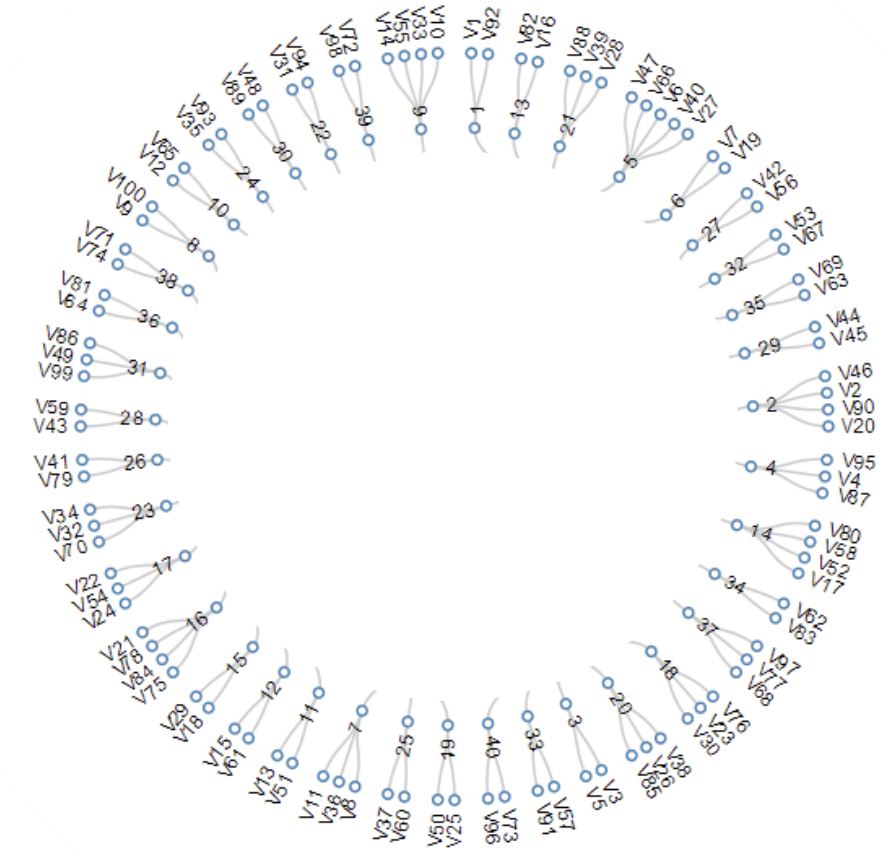
it's deterministic!



Hierarchical clustering agglomerative (bottom up)

it's deterministic!

computationally intense because every ***cluster pair*** distance has to be calculate



Hierarchical clustering agglomerative (bottom up)

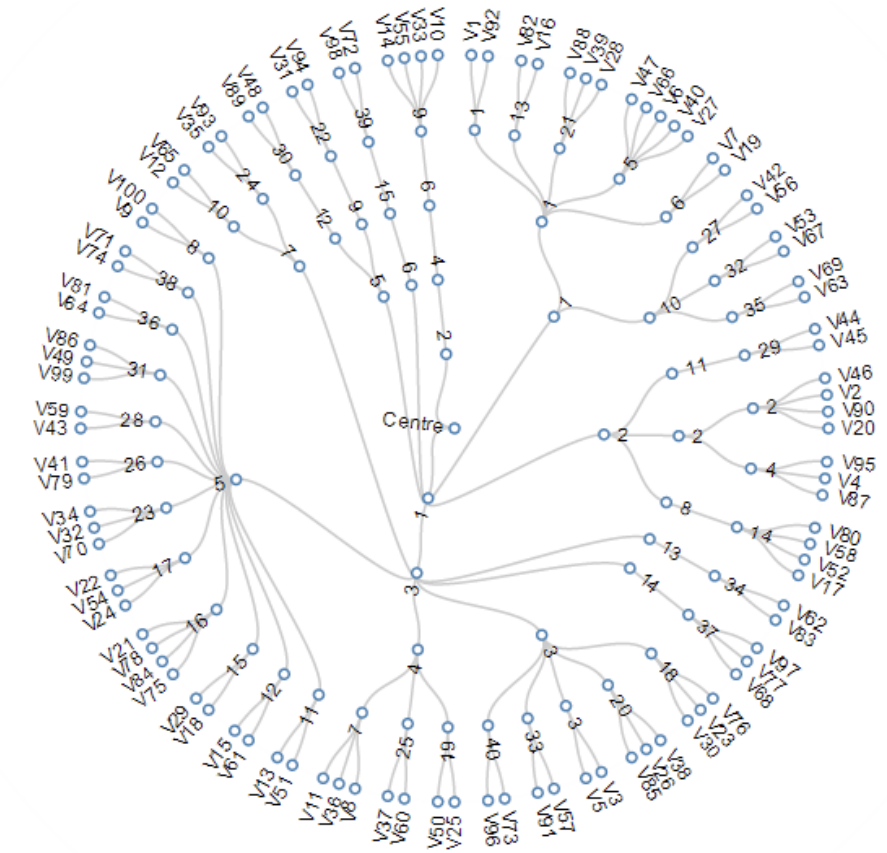
it's deterministic!

computationally intense because every ***cluster pair*** distance has to be calculate

it is slow, though it can be optimize:

complexity

$$O(N^2 d + N^3)$$



Agglomerative clustering:

Order: $O(N^2 d + N^3)$

PROs

It's deterministic

CONs

It's greedy (optimization is done step by step and agglomeration decisions cannot be undone)

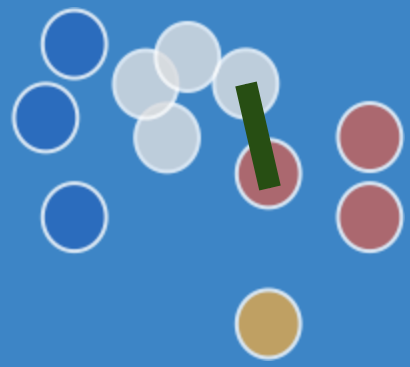
It's computationally expensive

Agglomerative clustering: *hyperparameters*

class `sklearn.cluster.` **`AgglomerativeClustering`** (*n_clusters=2, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', pooling_func='deprecated', distance_threshold=None*) ¶ [source]

- *n_clusters* : number of clusters
- *affinity* : the distance/similarity definition
- *linkage* : the scheme to measure distance to a cluster
- *random_state* : for reproducibility

linkage:

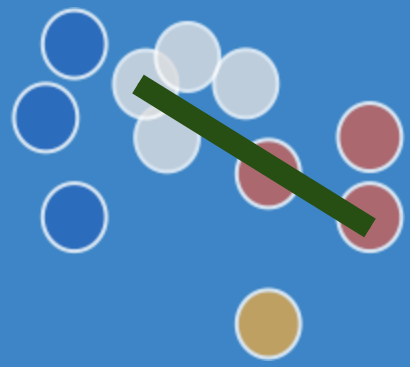


distance between a point and a cluster:

single link distance

D(c1,c2) = min(D(xc1, xc2))

linkage:



distance between a point and a cluster:

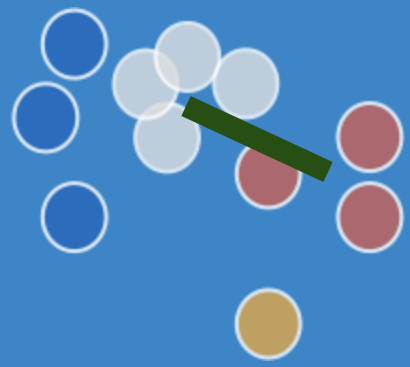
single link distance

D(c1,c2) = min(D(xc1, xc2))

complete link distance

D(c1,c2) = max(D(xc1, xc2))

linkage:



distance between a point and a cluster:

single link distance

$D(c1,c2) = \min(D(x_{c1}, x_{c2}))$

complete link distance

$D(c1,c2) = \max(D(x_{c1}, x_{c2}))$

centroid link distance

$D(c1,c2) = \text{mean}(D(x_{c1}, x_{c2}))$

linkage:

distance between a point and a cluster:

single link distance

D(c1,c2) = min(D(x_{c1}, x_{c2}))

complete link distance

D(c1,c2) = max(D(x_{c1}, x_{c2}))

centroid link distance

D(c1,c2) = mean(D(x_{c1}, x_{c2}))

Ward distance (global measure)

$$D_{tot} = \sum_j \sum_{i,x \in C_j} (x_i - \mu_j)^2$$

6.2 Divisive hierarchical clustering

Hierarchical clustering
divisive (top down)

Hierarchical clustering
divisive (top down)

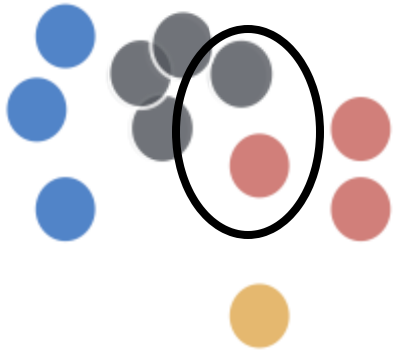
Hierarchical clustering
divisive (top down)

it is
non-deterministic
(like k-mean)

Hierarchical clustering
divisive (top down)

it is
non-deterministic
(like k-mean)

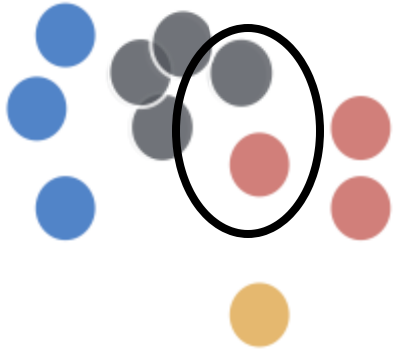
it is greedy -
just as k-means
two nearby points
may end up in
separate clusters



Hierarchical clustering
divisive (top down)

it is
non-deterministic
(like k-mean)

it is greedy -
just as k-means
two nearby points
may end up in
separate clusters



it is high complexity for
exhaustive search
$O(2^N)$
But can be reduced (~k-means)
$O(2Nk)$ or $O(N^2)$

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5751574/

Divisive clustering:
the algorithm

```
Calculate clustering criterion for all subgroups, e.g. min
intracluster variance
repeat
split the best cluster based on criterion above
untill
each data is in its own singleton cluster
```

Divisive clustering:

Order: $O(N^2)$ (w K-means procedure)

It's non-deterministic: the result depends on the (random) starting point (like K-mean) unless its exaustive (but that is $O(2^N)$)

or

It's greedy (optimization is done step by step)

Density Based

DBSCAN

7

DBSCAN

Density-based spatial clustering of applications with noise

DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature

DBSCAN

defines cluster membership based on local density: based on Nearest Neighbors algorithm.

DBSCAN:

the algorithm

- A point *p* is a ***core point*** if at least minPts points are within distance *ε* (including *p*).
- A point *q* is ***directly reachable*** from *p* if *q* is within distance *ε* from *p and p is a* core point.
- A point *q* is ***reachable*** from *p* if there is a path *p*1, ..., *pn* with *p*1 = *p* and *pn* = *q*, where each *pi*+1 is directly reachable from *pi* (a chain of directly reachable points).
- All points not reachable from any other point are ***outliers* or *noise.***

DBSCAN:

the algorithm

class `sklearn.cluster.` **`DBSCAN`** (*eps=0.5, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

- ***ε*** : maximum distance to join points
- **min_sample** : minimum number of points in a cluster, otherwise they are labeled outliers.
- **metric** : the distance metric
- **p** : float, optional The power of the Minkowski metric

DBSCAN:

the algorithm

class `sklearn.cluster.` **`DBSCAN`** (*eps=0.5, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

- ***ε*** : maximum distance to join points
- **min_sample** : minimum number of points in a cluster, otherwise they are labeled outliers.
- **metric** : the distance metric
- **p** : float, optional The power of the Minkowski metric

its extremely sensitive to these parameters!

DBSCAN:

the algorithm

An automatic method to calculate heart rate from zebrafish larval cardiac videos

Springer

May 2018 · BMC Bioinformatics 19(1)

DOI: 10.1186/s12859-018-2166-6

Chia-Pin Kang · Hung-Chi Tu · Tzu-Fun Fu · Show all 6 authors · Darby Tien-Hao Chang

```
DBSCAN(D, eps, MinPts)
   C = 0
   for each unvisited point P in dataset D
      mark P as visited
      NeighborPts = regionQuery(P, eps)
      if sizeof(NeighborPts) < MinPts
         mark P as NOISE
      else
         C = next cluster
         expandCluster(P, NeighborPts, C, eps, MinPts)

expandCluster(P, NeighborPts, C, eps, MinPts)
   add P to cluster C
   for each point P' in NeighborPts
      if P' is not visited
         mark P' as visited
         NeighborPts' = regionQuery(P', eps)
         if sizeof(NeighborPts') >= MinPts
            NeighborPts = NeighborPts joined with NeighborPts'
      if P' is not yet member of any cluster
         add P' to cluster C

regionQuery(P, eps)
   return all points within P's eps-neighborhood (including P)
```

Fig. 4 Pseudocode of the DBSCAN algorithm

DBSCAN clustering:

Order: $O(N^2)$

PROs

Deterministic.

Deals with noise and outliers

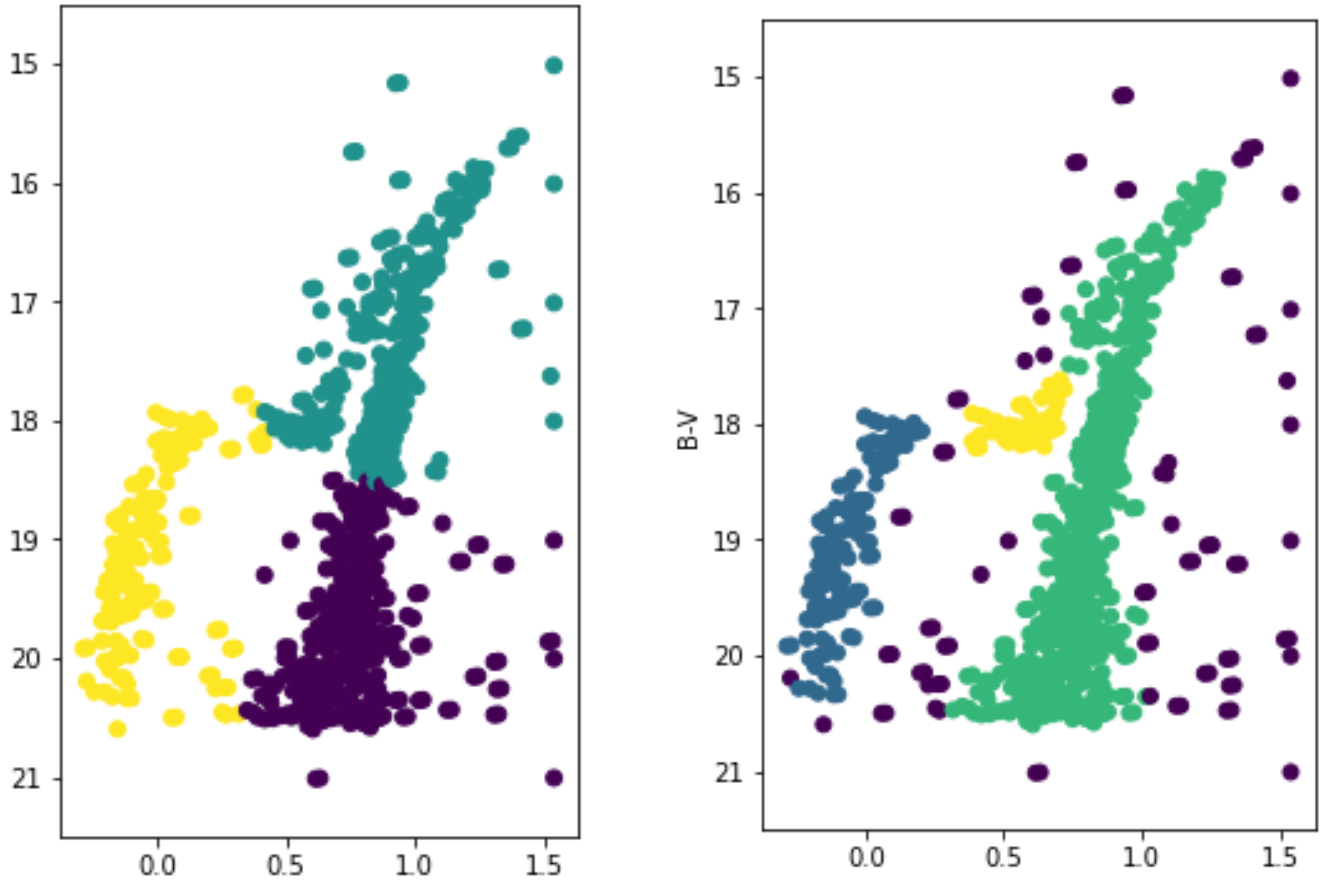
Can be used with any definition of distance or similarity

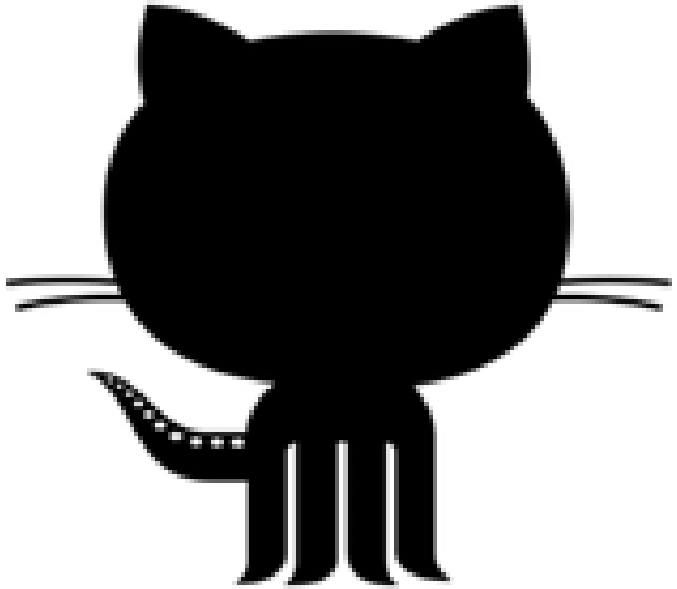
PROs

Not entirely deterministic.

Only works in a constant density field

different results for k-means and DBScan





https://github.com/fedhere/DSPS/blob/master/lab10/StellarPopClustersLab.ipynb

Clustering : unsupervised learning where all features are observed for all datapoints. The goal is to partition the space into maximally homogeneous maximally distinguished groups

clustering is easy, but interpreting results is tricky

Distance : A definition of distance is required to group observations/ partition the space.

Common distances over continuous variables

- Minkowski (inlcudes Euclidian = Minkowski(2)
- Great Circle (for coordinated on a sphere, e.g. earth or sky)

Common distances over categorical variables:

- Simple Distance Matrix
- Jaccard Distance

Whitening

Models assume that the data is not correlated. If your data is correlated the model results may be invalid. And your data always has correlations.

- whiten the data by using the matrix that diagonalizes the covariance matrix. This is ideal but computationally expensive if possible at all

Solution:

- scale your data so that each feature is mean=0 stdev=2 and at least they have the same weight on the result

Partition clustering:

Hard: K-means *O(KdN)* , needs to decide the number of clusters, non deterministic simple efficient implementation but the need to select the number of clusters is a significant flaw

Soft: Expectation Maximization *O(KdNp)* , needs to decide the number of clusters, need to decide a likelihood function (parametric), non deterministic

Hierarchical:

Divisive: Exhaustive $O(2^N)$; $O(N^2)$ at least non deterministic

Agglomerative: $O(N^2d + N^3)$**,** deterministic, greedy. Can be run through and explore the best stopping point. Does not require to choose the number of clusters a priori

Density based

DBSCAN: Density based clustering method that can identify outliers, which means it can be used in the presence of noise. Complexity $O(N^2)$. Most common (cited) clustering method in the natural sciences.

encoding categorical variables:

variables have to be encoded as numbers for computers to understand them. You can encode categorical variables with integers or floating point but you implicitly impart an order. The standard is to ***one-hot-encode*** which means creating a binary (True/False) feature (column) for each category of a categorical variables but this *increases the feature space and generated covariance.*

a comprehensive review of clustering methods

Data Clustering: A Review, Jain, Mutry, Flynn 1999

https://www.cs.rutgers.edu/~mlittman/courses/lightai03/jain99data.pdf

a blog post on how to generate and interpret a scipy dendrogram by Jörn Hees

https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/

resources

your data aint that big...

https://www.chrisstucchio.com/blog/2013/hadoop_hatred.html