PHYS465: Statistical Data Analysis in Physics

Week 3: Clustering and Classification

Mathew Smith and Brooke Simmons

mat.smith@lancaster.ac.uk

Physics Building; C46

These notes lean heavily on notes originally created by Prof. Jessi Cisewski Kehe, https://www.jjckehe.com/home

Clustering “versus” Classification

Clustering

Find subtypes or groups that are not defined *a priori* based on measurements

→ “Unsupervised learning” or “Learning without labels”

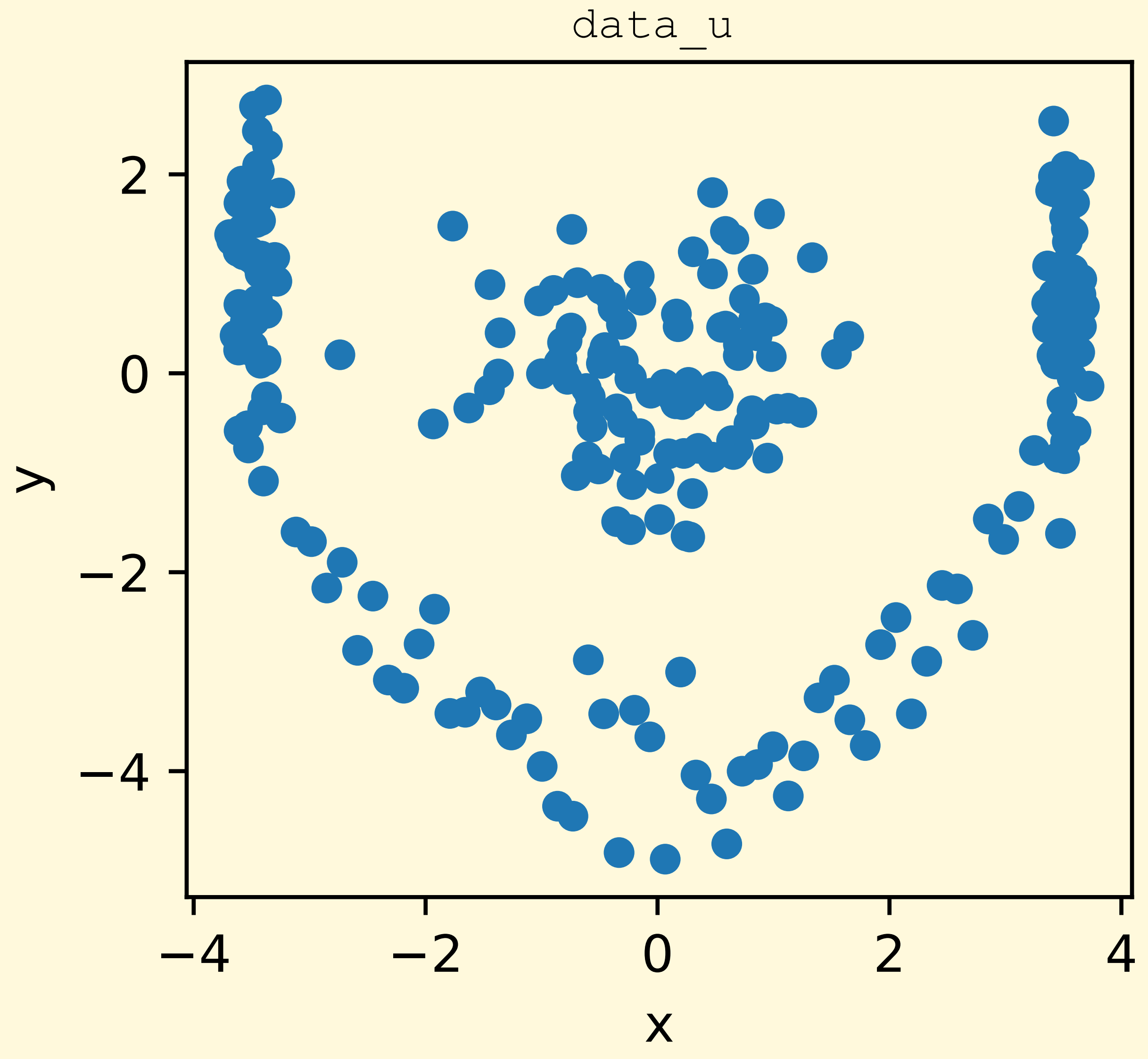
Classification

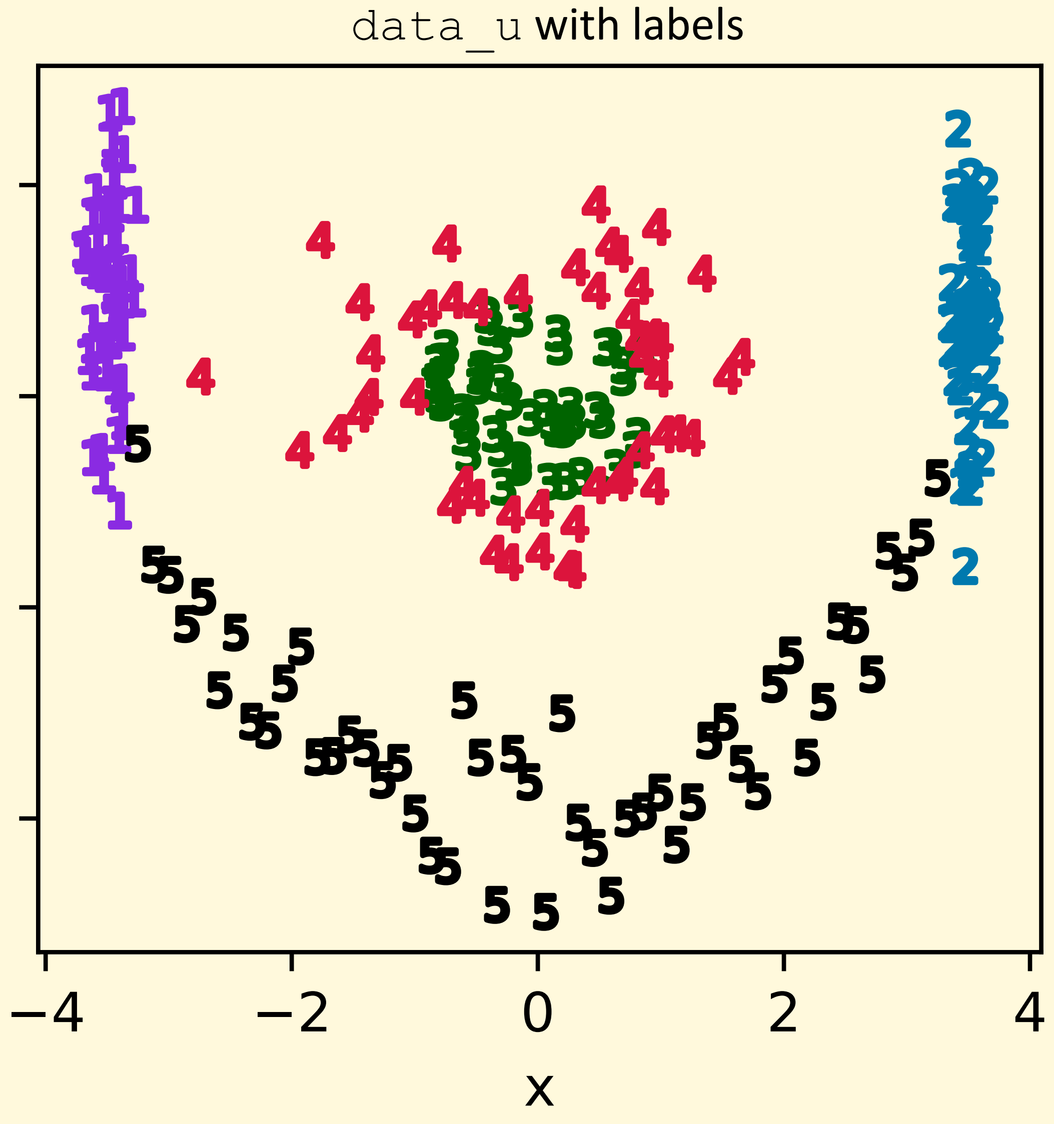
Use *a priori* group labels in analysis to assign new observations to a particular group or class

→ Sometimes applied once some type of clustering has been applied; can be separate

→ “Supervised learning” or “Learning with labels”

So, how do we sort and label data?



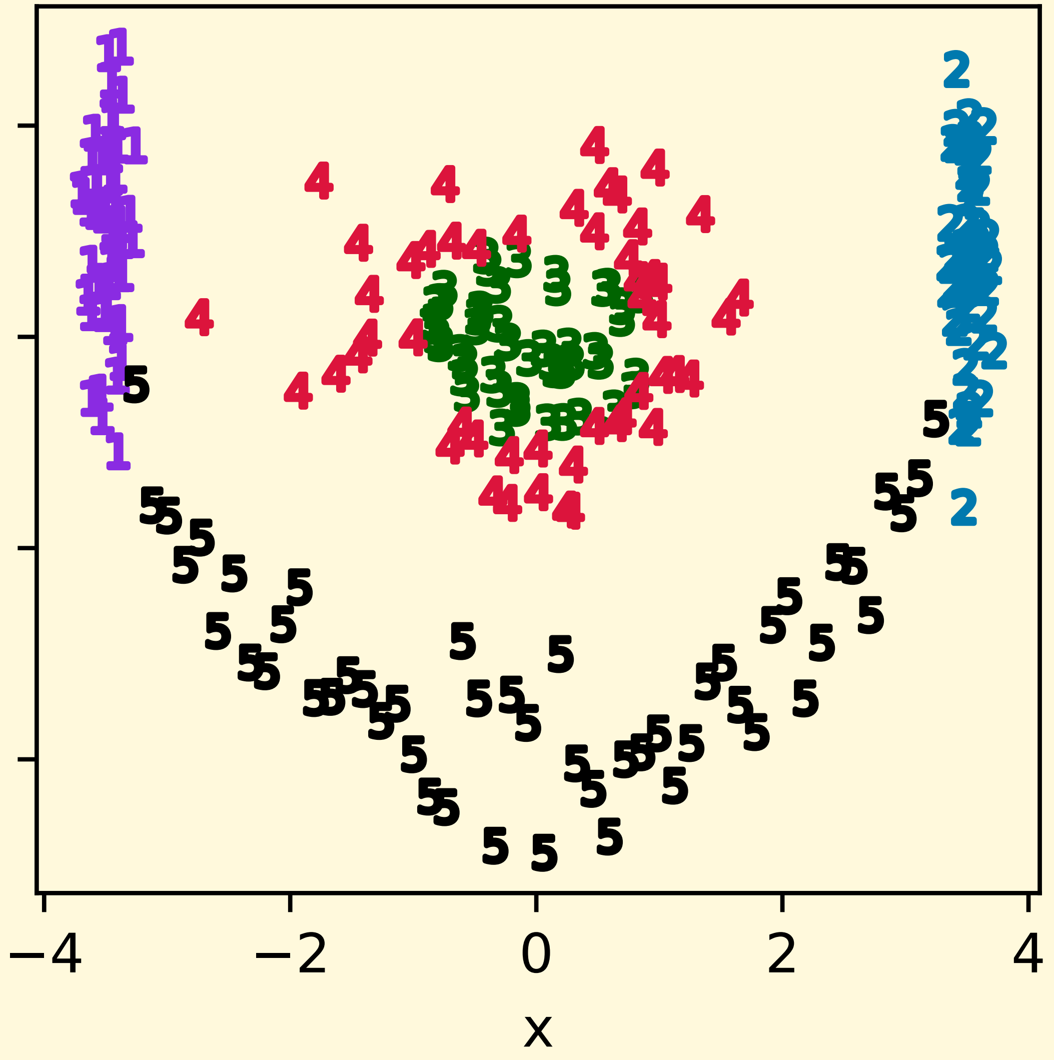


A test dataset with known labels

```
# make 5 data subsets of size 50 points each
x1     = np.random.normal(-3.5, 0.1, size=n_pts)
y1     = np.random.normal( 1., 1.0, size=n_pts)
x2     = np.random.normal( 3.5, 0.1, size=n_pts)
y2     = np.random.normal( 1., 1.0, size=n_pts)
# datasets 3 and 4 are concentric circles in overall distribution shape
xtemp = np.random.normal( 0., 0.75, size=n_pts*2)
ytemp = np.random.normal( 0., 0.75, size=n_pts*2)
rad1  = np.sqrt(xtemp**2 + ytemp**2)
i_r   = np.argsort(rad1)
# it's not necessary to separate these into datasets 3 and 4 right here;
# this is now sorted so I could just do it with labelling later
# but it may help others follow along to be explicit that datasets
# 3 and 4 are quite artificially separated in this example
x3     = xtemp[i_r[:n_pts]]
y3     = ytemp[i_r[:n_pts]]
x4     = xtemp[i_r[n_pts:]]
y4     = ytemp[i_r[n_pts:]]
x5     = np.linspace(-3.25, 3.25, n_pts)
y5     = (x5/2.)**2 - 4 + np.random.normal(0., 0.5, n_pts)

x_all = np.append(x1, [x2, x3, x4, x5])
y_all = np.append(y1, [y2, y3, y4, y5])

label_ones = np.ones_like(x1).astype(int)
data_u = np.array([x_all, y_all])
labels = np.append(label_ones, [label_ones+1, label_ones+2,
                                label_ones+3, label_ones+4])
```



Clustering setup

Notation:

Given vectors $\boldsymbol{X} = \{\boldsymbol{X}_1, \boldsymbol{X}_2, \dots, \boldsymbol{X}_n\} \in \mathbb{R}^p$

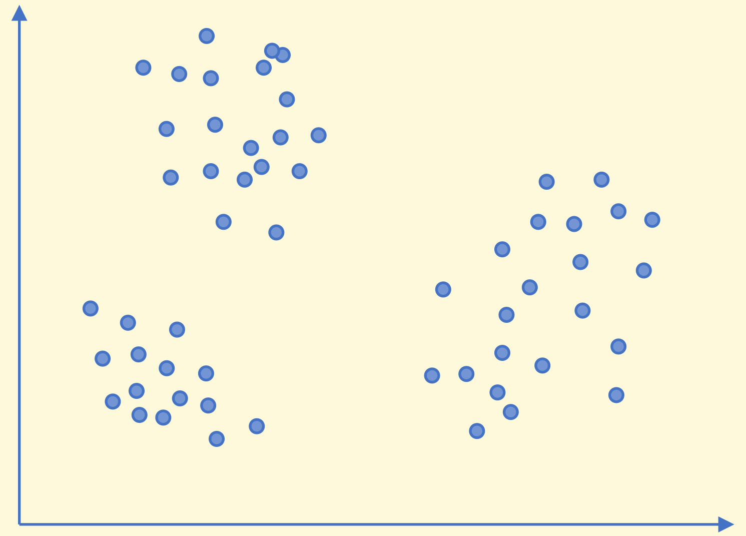
→ n observations in p-dimensional space

→ variables/features/attributes indexed by $j = 1, \dots, p$: j^{th} variable is $\boldsymbol{X}_j$

→ observations indexed by $i = 1, \dots, n$: i^{th} variable is $\boldsymbol{X}_i$

Goals and Limitations:

- Want to learn properties about the joint distribution $P(\boldsymbol{X})$ of these vectors: organise, summarise, categorise, explain
- No direct measure of success (*e.g.,* no notion of a misclassification rate) → successful *if* true structure is captured



General goals of clustering

Partition observations such that:

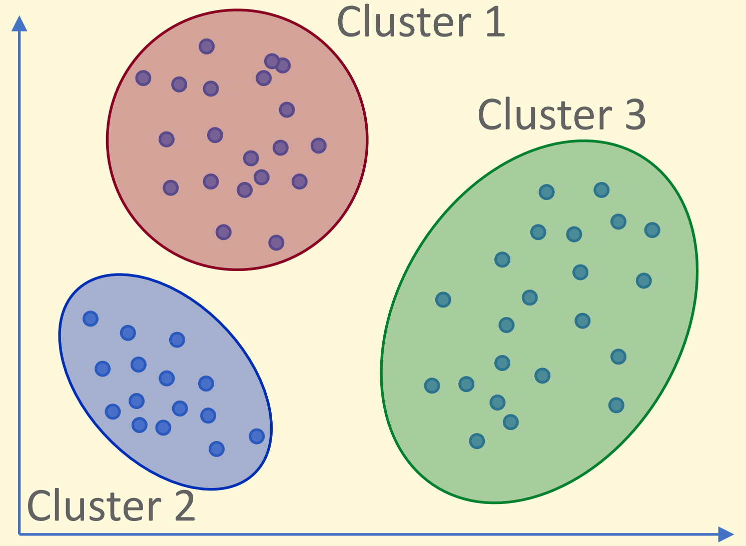
Observations within a cluster are similar

→ "Compactness" property

Observations in different clusters are non-similar

→ "Closeness" property

Typically want compact clusters
that are well-separated



Dissimilarity Measure: within-cluster variation

→ Characterises degree of "closeness"

Dissimilarity matrix $\boldsymbol{D} = \{d_{ii'}\}$ such that $d_{ii} = 0$ and

$$d_{ii'}^{j} = d\left(x_{ij}, x_{i'j}\right)$$

Some examples of $d_{ii'}^{j}$ are $\left(x_{ij} - x_{i'j}\right)^2$ or $\left|x_{ij} - x_{i'j}\right|$

$D_{ii'} = D\left(\boldsymbol{X}_i, \boldsymbol{X}_i'\right) = \sum_{j=1}^{p} w_j \cdot d_{ii'}^{j}$ where w_j are weights and $\sum_{j=1}^{p} w_j = 1$

Dissimilarity Measure: within-cluster variation

Total cluster variability $= \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n} D_{ii'}$

$$= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\left(\sum_{C(i')=k} D_{ii'} + \sum_{C(i')\neq k} D_{ii'}\right)$$

where $C(i) = k$ is the assignment of observation i to cluster k

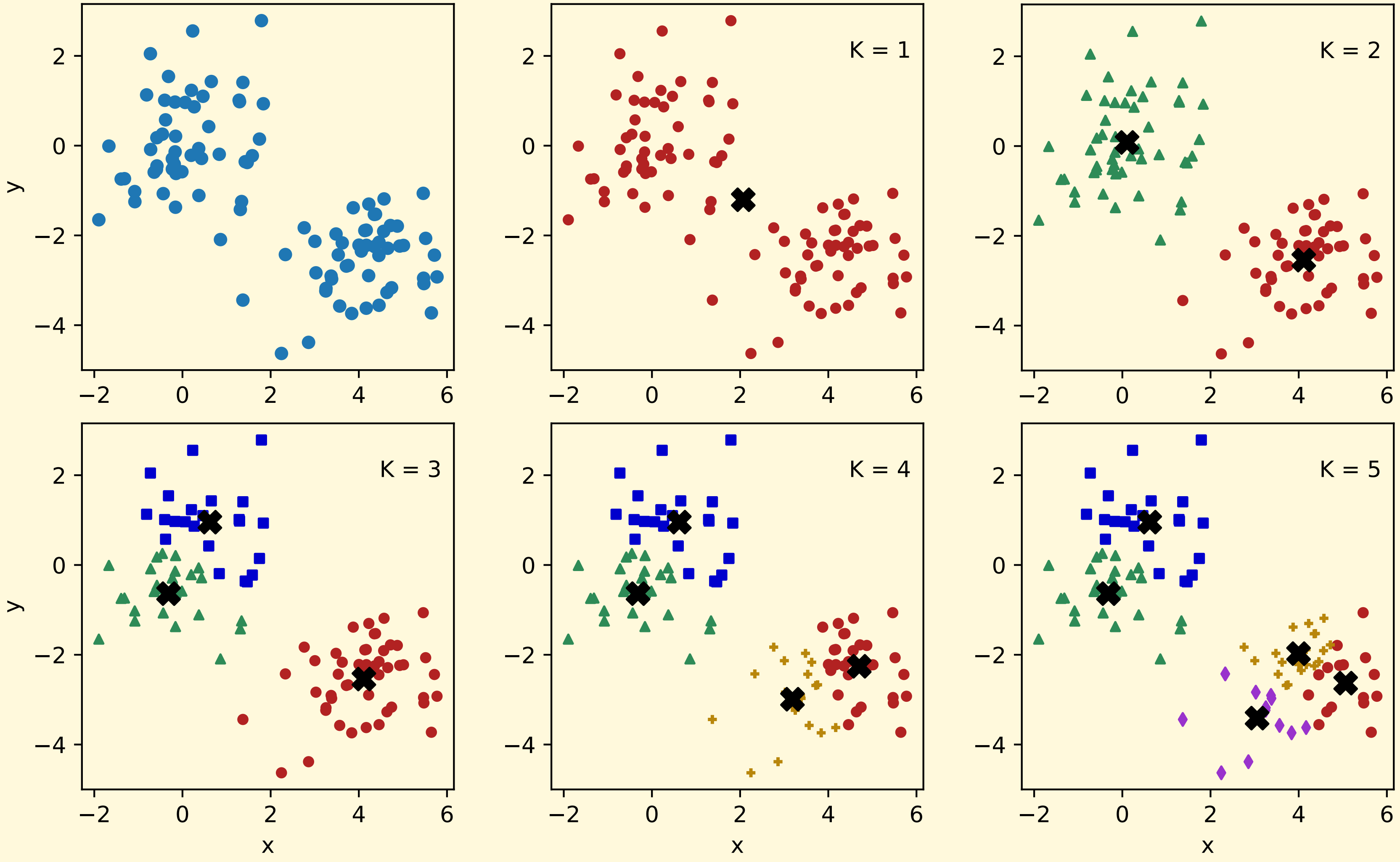
Total within-cluster variability: $\frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} D_{ii'}$

Total between-cluster variability: $\frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')\neq k} D_{ii'}$

K-Means Clustering

Main idea: partition observations in K separate clusters that do not overlap.

- Each observation is assigned to 1 cluster.
- No notion of strength of membership in the cluster.
- Must specify K in advance, and will always find K clusters.



K-Means Clustering: procedure

Goal: minimise total within-cluster scatter using

$$D_{ii'} = \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2 = \|\boldsymbol{X}_i - \boldsymbol{X}_{i'}\|^2$$

Note: K-means always uses this dissimilarity measure (no variations).

Then the within-cluster scatter is written as

$$\frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} \|\boldsymbol{X}_i - \boldsymbol{X}_{i'}\|^2 = \sum_{k=1}^{K} \left|C_k\right| \sum_{C(i)=k} \|\boldsymbol{X}_i - \bar{\boldsymbol{X}}_k\|^2$$

where $\left|C_k\right|$ is the number of observations in cluster C_k ,

with means $\bar{\boldsymbol{X}}_k = \left(\bar{X}_1^k, \ldots, \bar{X}_p^k\right)$

K-Means Clustering: recipe

1. Pick K, the number of clusters.
2. Select K cluster centres (multiple ways to initiate these).
3. Iterate until members no longer switch clusters:
    a. Assign each observation (data point) to its closest cluster centre.
    b. Re-calculate cluster centres.

Python: `sklearn.cluster.KMeans` (scikit-learn.org)

Try it yourself: generate 3 random datasets (use `np.random.normal`), cluster them with K-means and $K = 3$, then try moving the datasets closer together and farther apart and re-clustering, visualising every result, to help you understand how the algorithm works.

K-Means Clustering: recipe

1. Pick K, the number of clusters.
2. Select K cluster centres (multiple ways to initiate these).
3. Iterate until members no longer switch clusters:
    a. Assign each observation (data point) to its closest cluster centre.
    b. Re-calculate cluster centres.

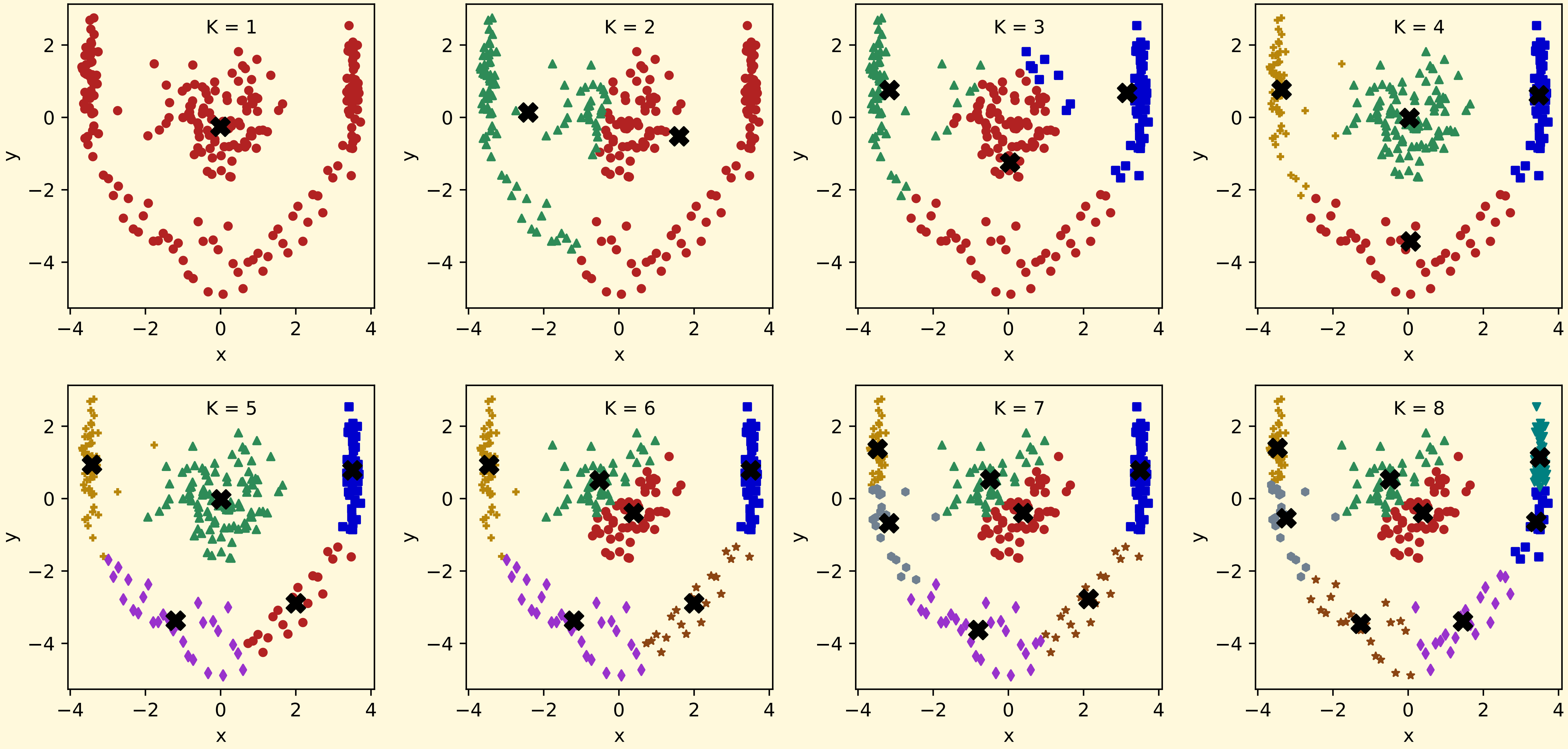
Python: `sklearn.cluster.KMeans` (scikit-learn.org)

Try it yourself: generate 3 random datasets (use `np.random.normal`), cluster them with K-means and $K = 3$, then try moving the datasets closer together and farther apart and re-clustering, visualising every result, to help you understand how the algorithm works.

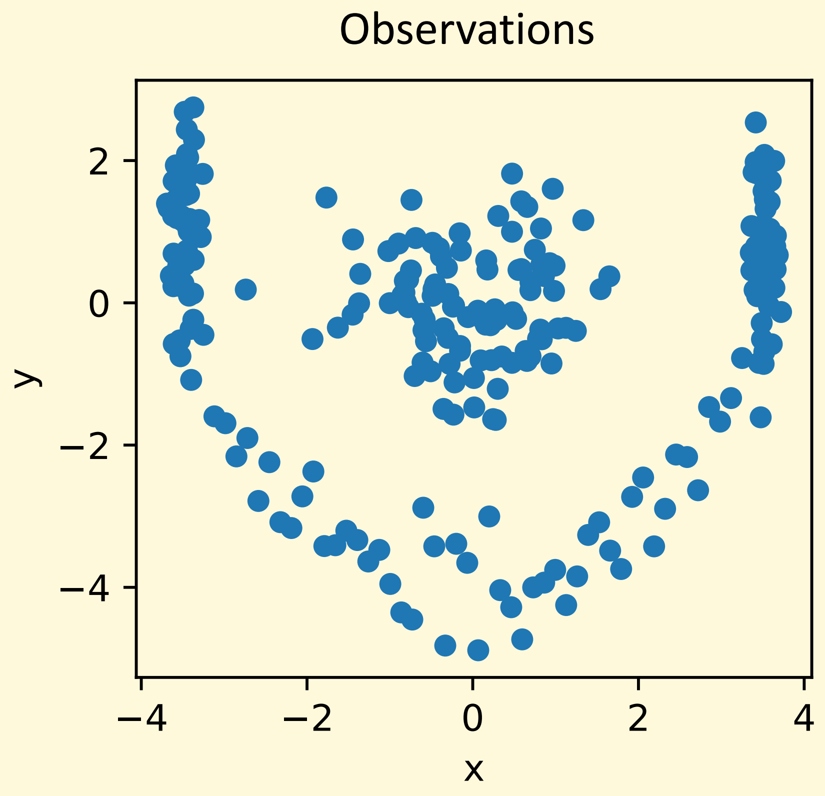
Example:

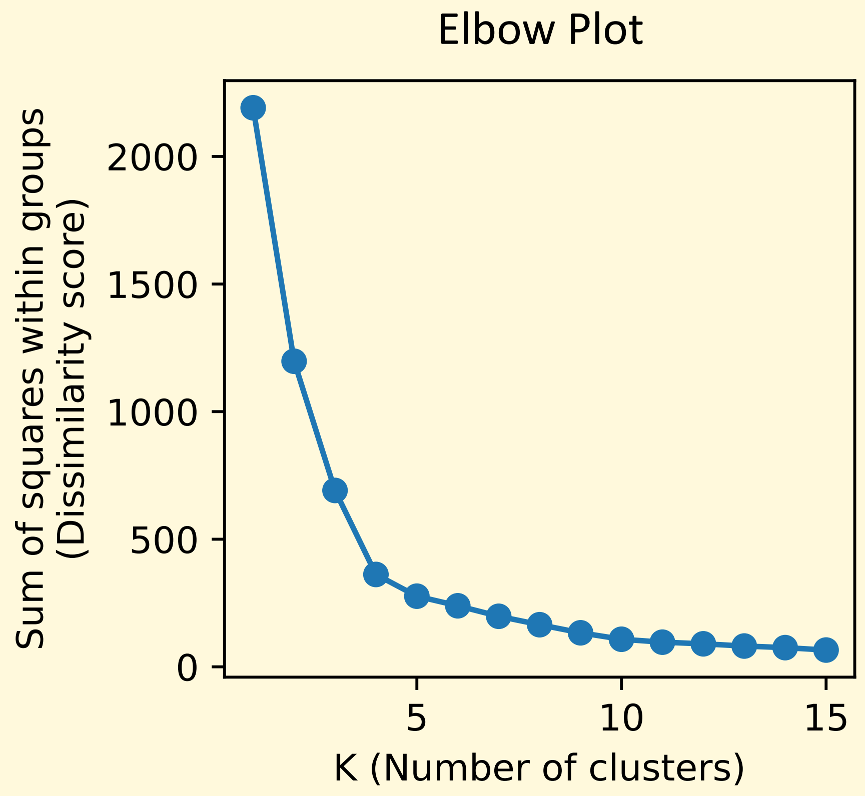
```
import np.random.normal as normal
from sklearn.cluster import KMeans
n = 50
y = [-1., 1., 3]
data1 = np.array([normal(0.0, 1.0, size=n), normal(y[0], 0.1, size=n)])
data2 = np.array([normal(0.0, 1.0, size=n), normal(y[1], 0.1, size=n)])
data3 = np.array([normal(0.0, 1.0, size=n), normal(y[2], 0.1, size=n)])
data_play = np.vstack([data1.T, data2.T, data3.T])
kmeans = KMeans(n_clusters=3)
kmeans.fit(data_play)
```

K-Means Clustering: determining *K*



K-Means Clustering: determining K



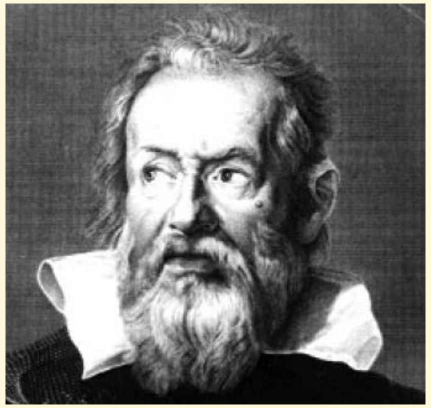


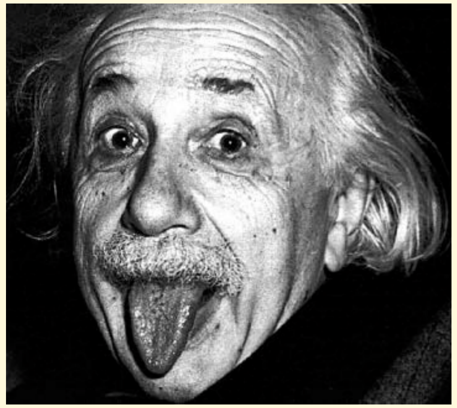
Choose the k that has the last “significant” reduction in the within-groups sum-of-squares statistic (*i.e.,* find the “elbow”)

The statistic used can change depending on clustering method, but the concept is sound.

K-Means Clustering: Tips & Caveats

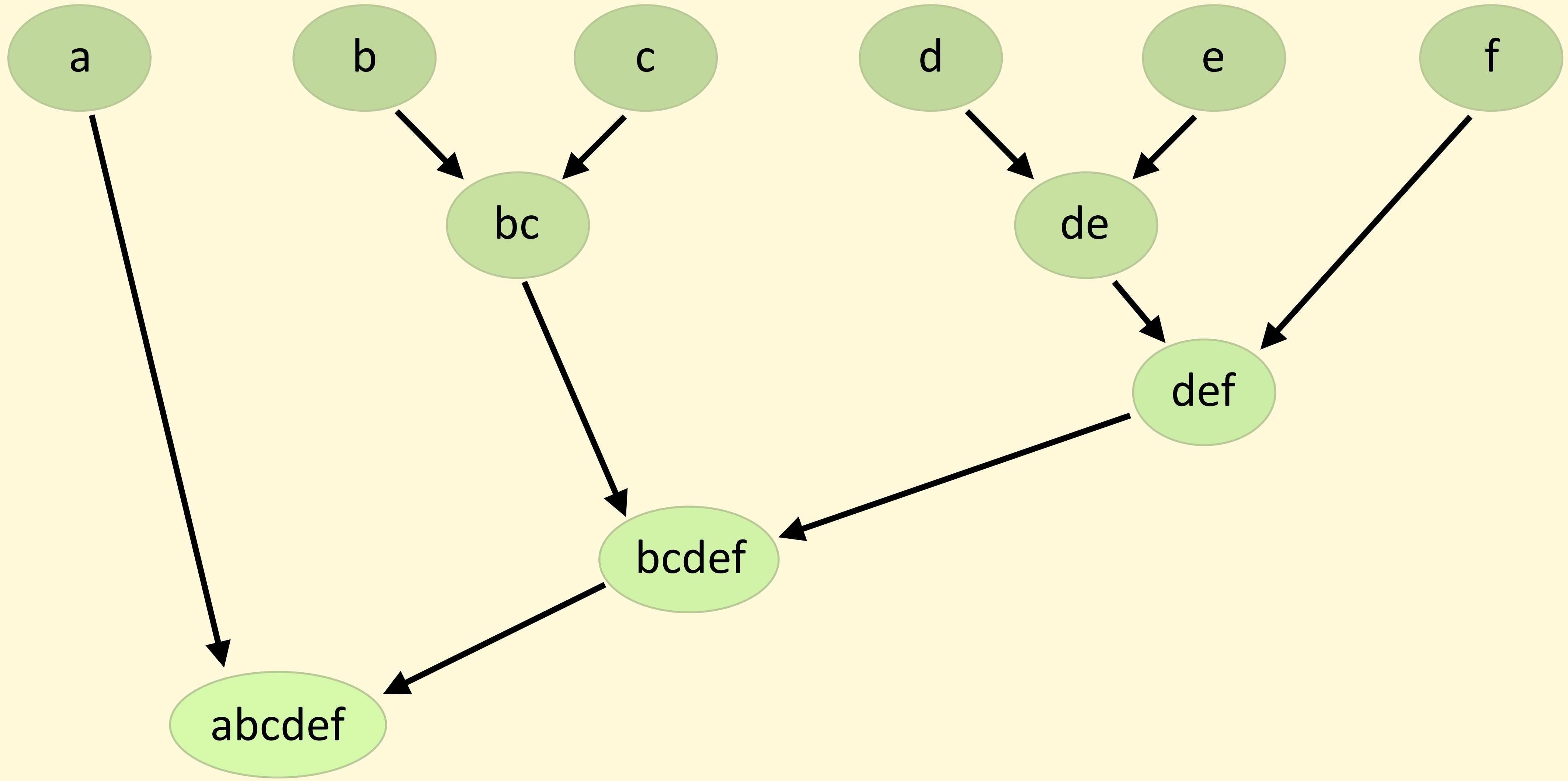
- Can be unstable; solution depends somewhat on the starting set of centres
    - finds local optima, but we may want the global optima
    - try starting with different centres, run at least 10 iterations to make sure you're not stuck in a local optimum
- Cluster assignments are strict: no notion of degree or strength of membership
- Possible lack of interpretability of centroids.
    - Centres are averages. Fine for clustering many things, but not all. Example: what if the observations are faces?





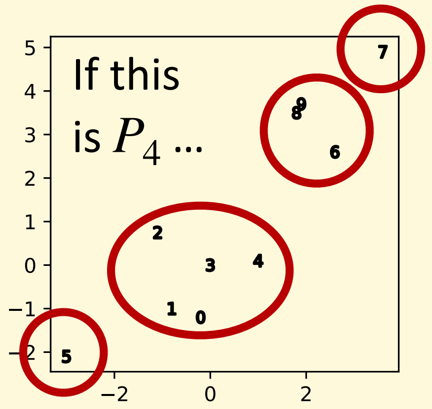
Hierarchical Clustering

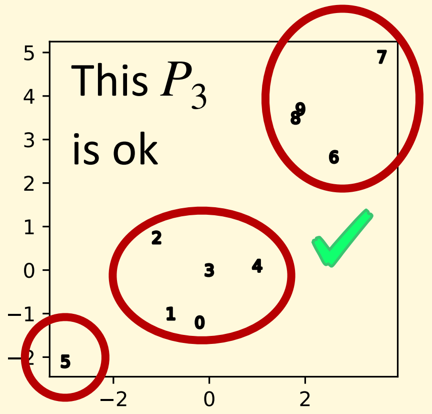
Because you don't always want to determine K in advance:
Start with each point as its own "cluster" and then group by similarities.

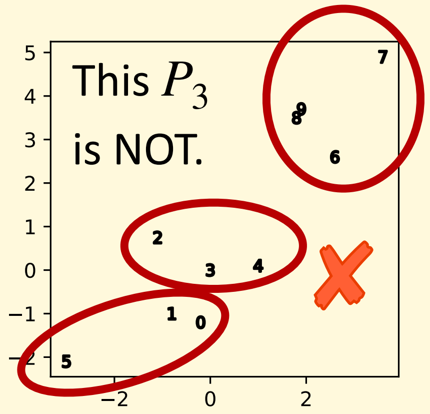


Hierarchical Partitioning vs Flat Partitioning

- Flat Partitioning (*e.g.,* K-means clustering):
  - Partitions data into K partitions, where K is set in advance by the user
  - No sense of the relationships between clusters
- Hierarchical partitioning:
  - Generates a hierarchy of partitions; user selects which partition to use after the full hierarchy is created
  - $P_1 = 1$ cluster, …, $P_n = n$ clusters (“agglomerative” clustering)
  - Partition P_i is the **union** of one or more clusters from Partition P_{i+1}



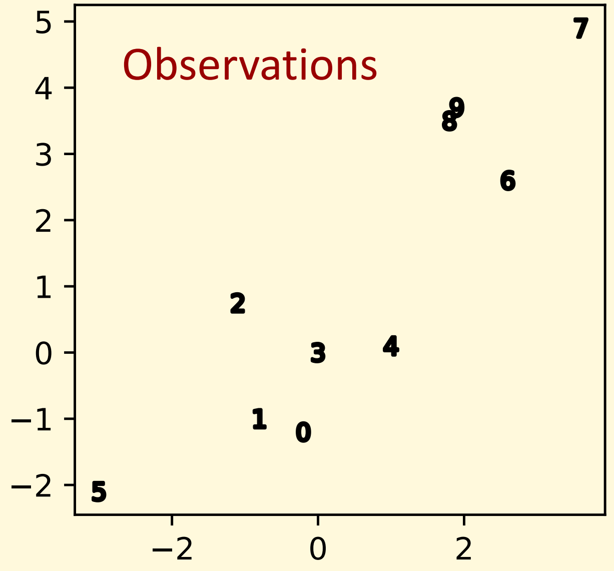


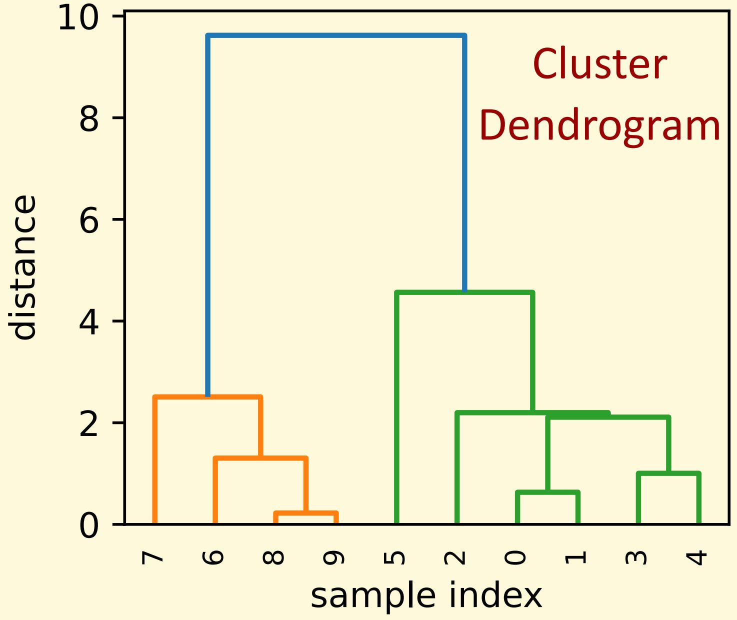


Hierarchical Clustering: recipe

Define a dissimilarity measure $d_{kk'} = d\left(C_k, C_{k'}\right)$ between clusters C_k and $C_{k'}$ as a function of distance between points in the clusters

1. Start with every observation in its own cluster
2. Find $\min\left(d\left(C_k, C_{k'}\right)\right)$ across all cluster pairs $\longrightarrow$ merge C_k and $C_{k'}$
3. Repeat until only 1 cluster remains





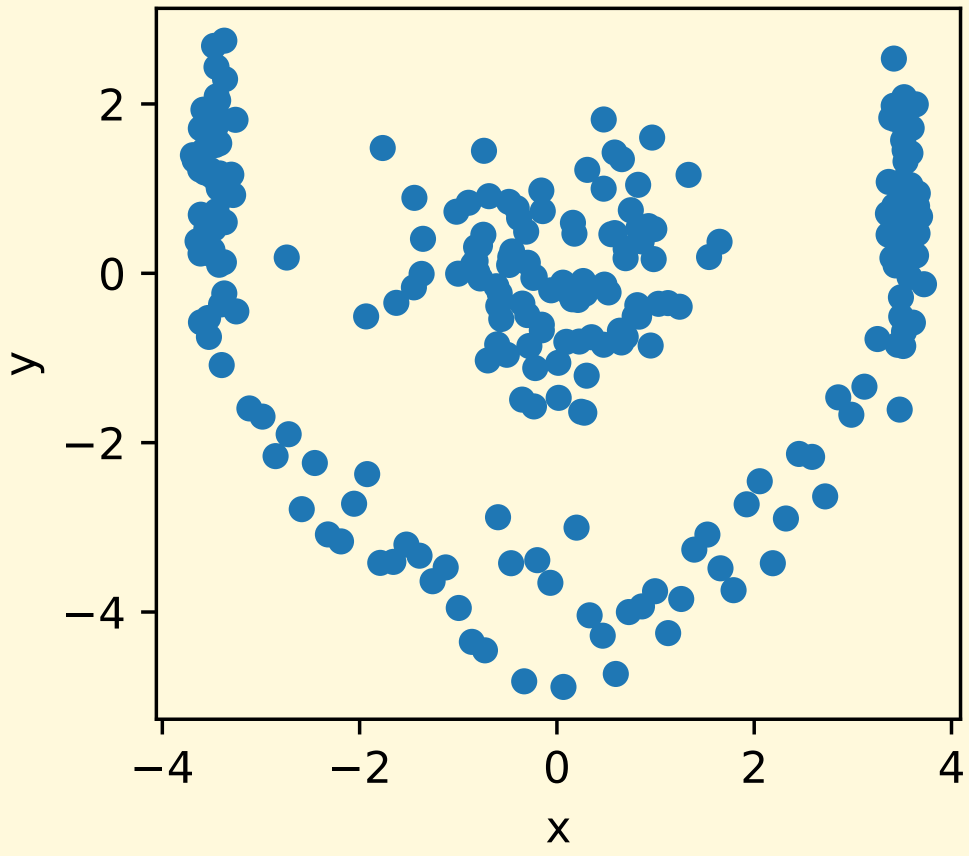
a. Read the dendrogram from the bottom up.
b. If you change the dissimilarity measure, the distances (y-axis) in the dendrogram will change.

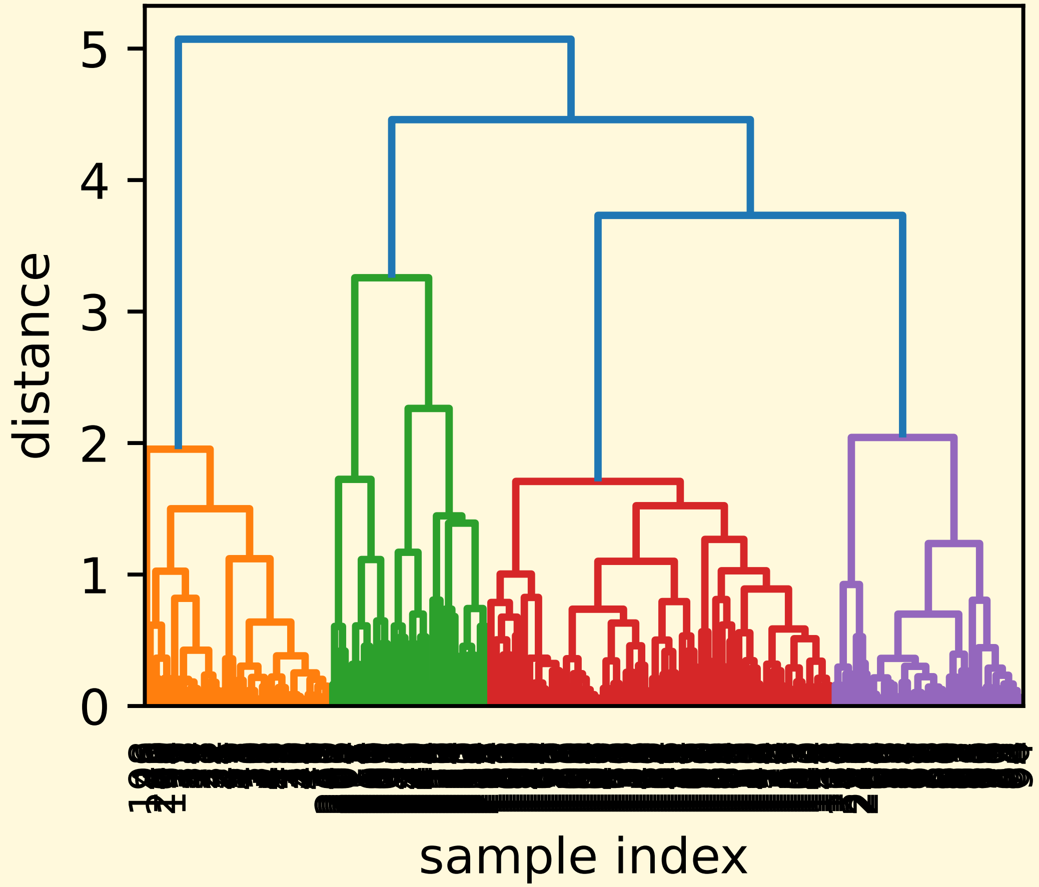
Python: use `scipy.cluster.hierarchy`

Hierarchical Clustering: recipe

Define a dissimilarity measure $d_{kk'} = d\left(C_k, C_{k'}\right)$ between clusters C_k and $C_{k'}$ as a function of distance between points in the clusters

1. Start with every observation in its own cluster
2. Find $\min\left(d\left(C_k, C_{k'}\right)\right)$ across all cluster pairs $\longrightarrow$ merge C_k and $C_{k'}$
3. Repeat until only 1 cluster remains





Hierarchical Clustering: common distances

- **Single-linkage** clustering (friends-of-friends): the intergroup distance (y-axis on the dendrogram) is the smallest possible distance between clusters

$$d\left(C_k, C_{k'}\right) = \min_{x \in C_k,\, y \in C_{k'}} \left(d(x, y)\right)$$

- **Complete-linkage** clustering: intergroup distance is largest possible distance

$$d\left(C_k, C_{k'}\right) = \max_{x \in C_k,\, y \in C_{k'}} \left(d(x, y)\right)$$

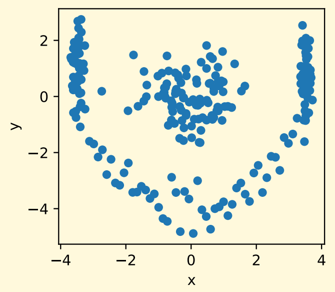
- **Average-linkage** clustering: uses the average distance between clusters

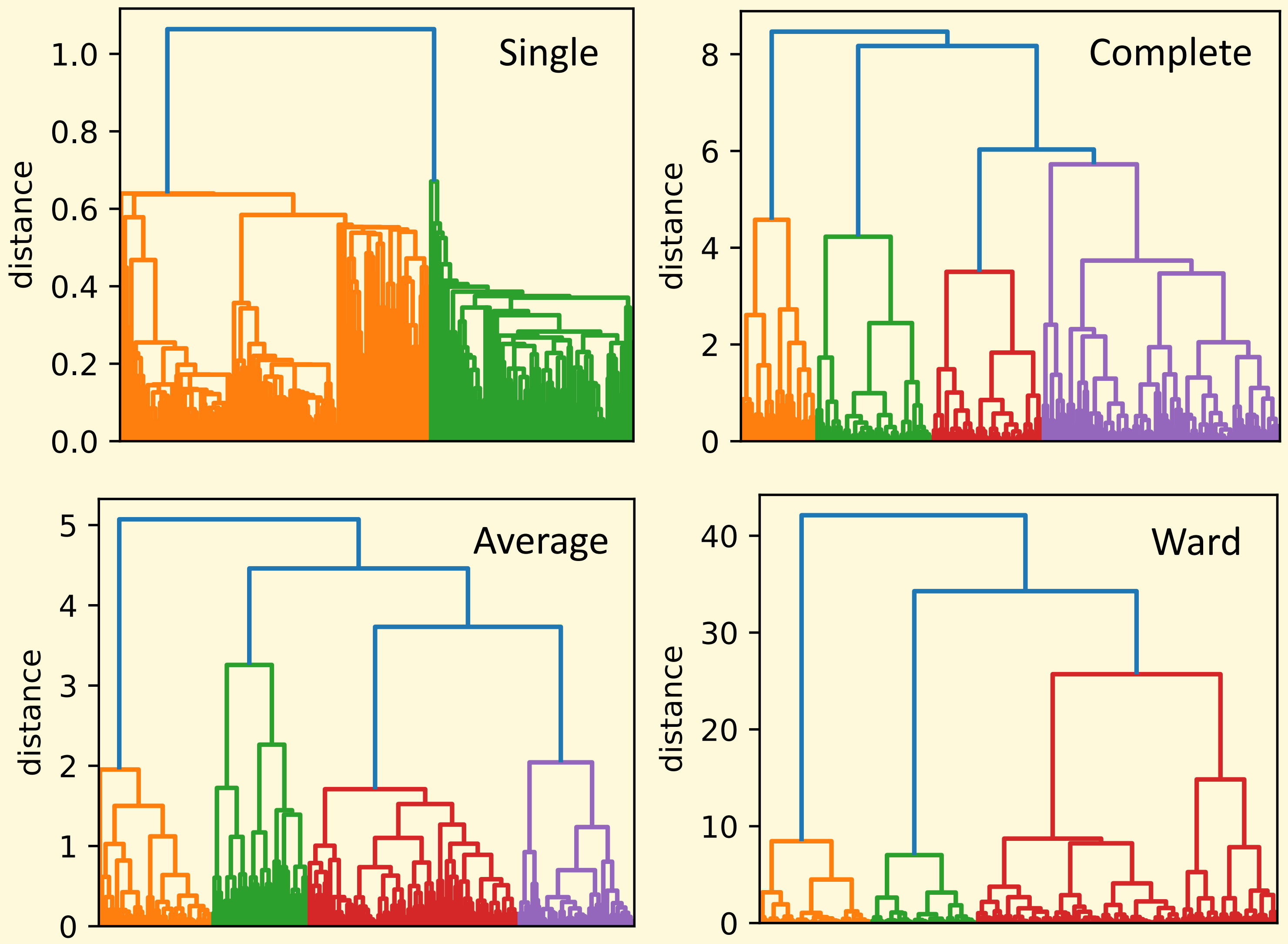
$$d\left(C_k, C_{k'}\right) = \mathrm{Ave}_{x \in C_k,\, y \in C_{k'}} \left(d(x, y)\right)$$

- **Ward’s** clustering:

$$d\left(C_k, C_{k'}\right) = \frac{2\left(\left|C_k\right| \cdot \left|C_{k'}\right|\right)}{\left|C_k\right| + \left|C_{k'}\right|} \|\bar{X}_{C_k} - \bar{X}_{C_{k'}}\|^2$$

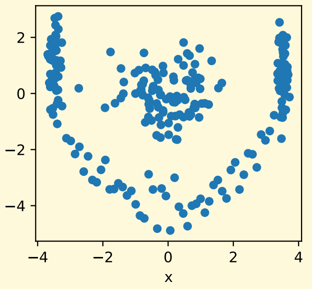
Hierarchical Clustering: example dataset

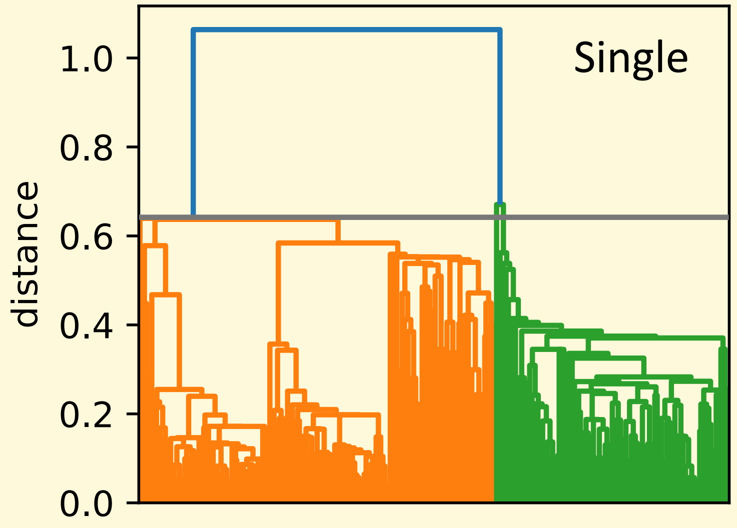


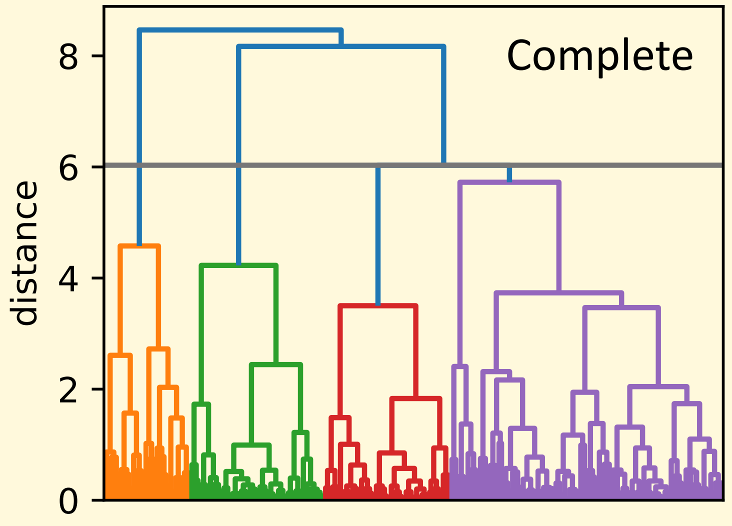


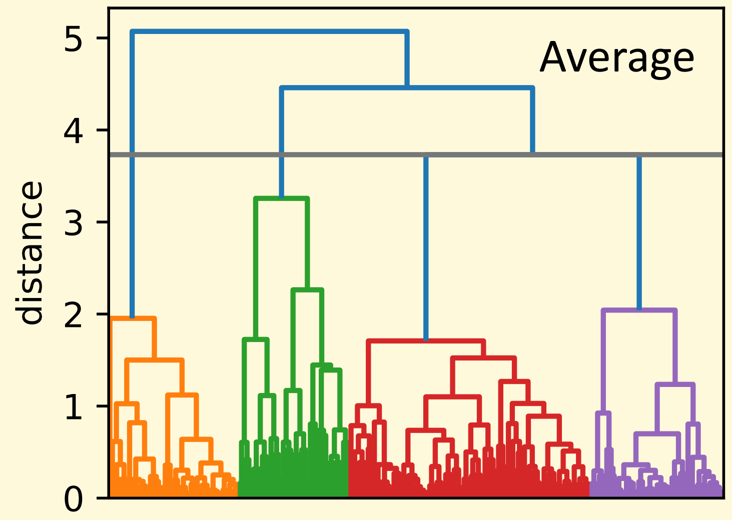
- Given the different definitions of distance $d\left(C_k, C_{k'}\right)$ (see previous slide), we expect that the y-axis scales will be different.
- The user must choose a distance value at which to use the dendrogram to define cluster membership. Example: "choose d such that there are 4 clusters"

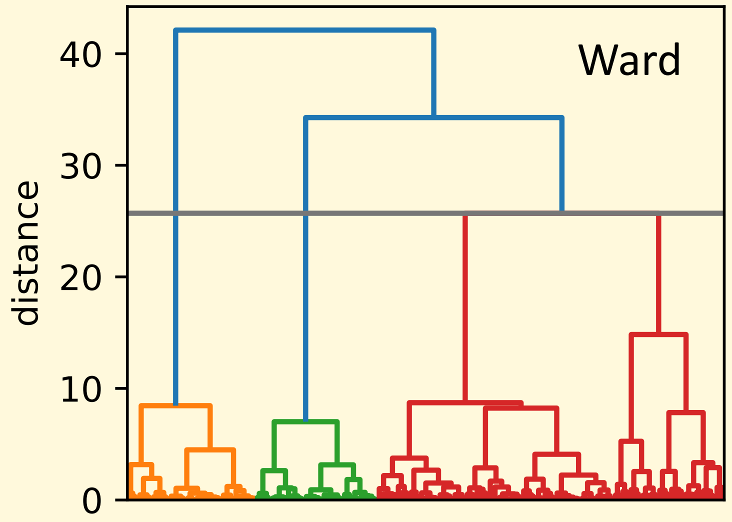
Hierarchical Clustering: choose d : $K = 4$





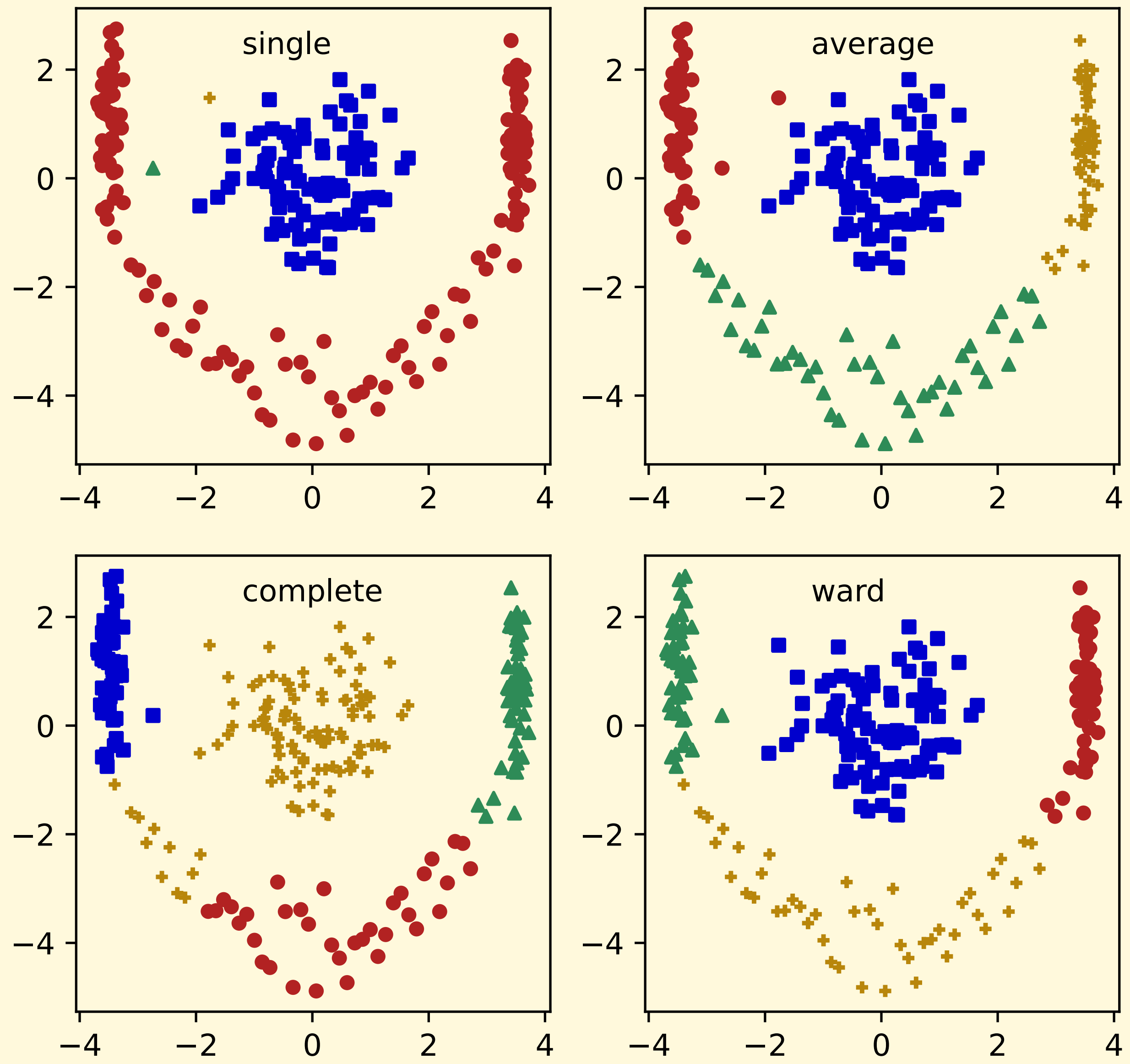




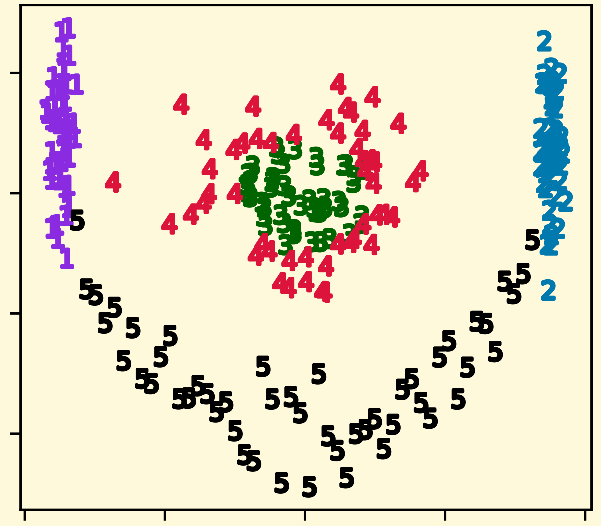


- In the dendrograms, the amount by which you have to change the cutoff d to change K can vary greatly.
- In general, you want to choose a distance measure such that you can achieve clustering that is useful in your context (which sometimes means a target K) and robust to small changes in d_{cut} .

Hierarchical Clustering: choose $d : K = 4$

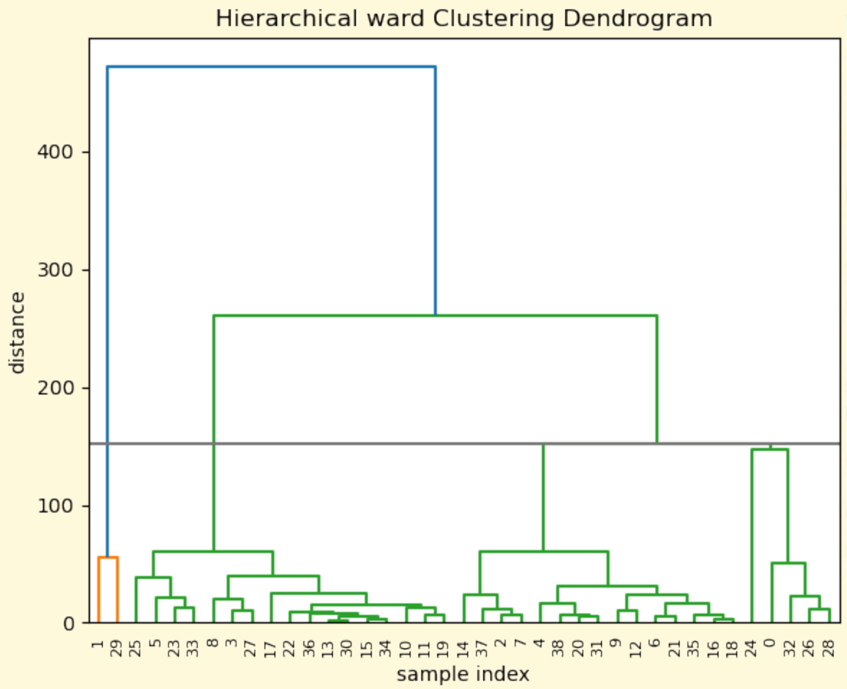


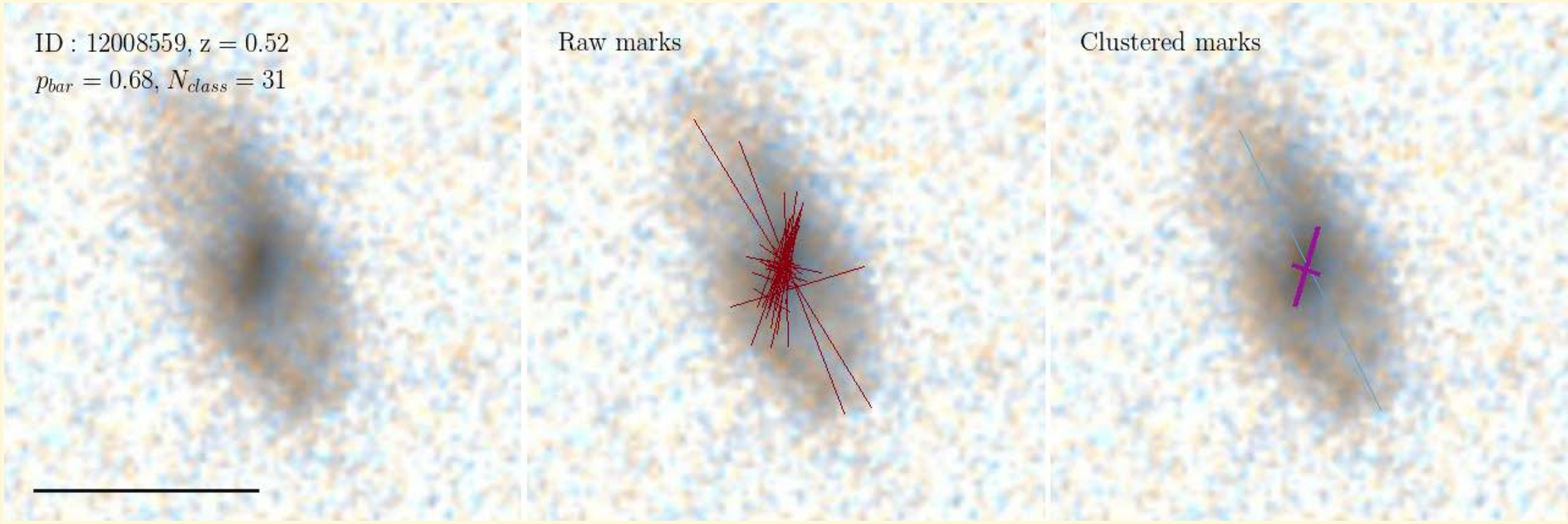
- For this dataset, you would probably choose the Average or Ward distance, but for the next dataset it might be different.
- Always consider your science goals and your specific data's properties.
- Try it yourself: is there a value of d for which you can recover the intrinsic $K = 5$ of the data?



Hierarchical Clustering: choosing d

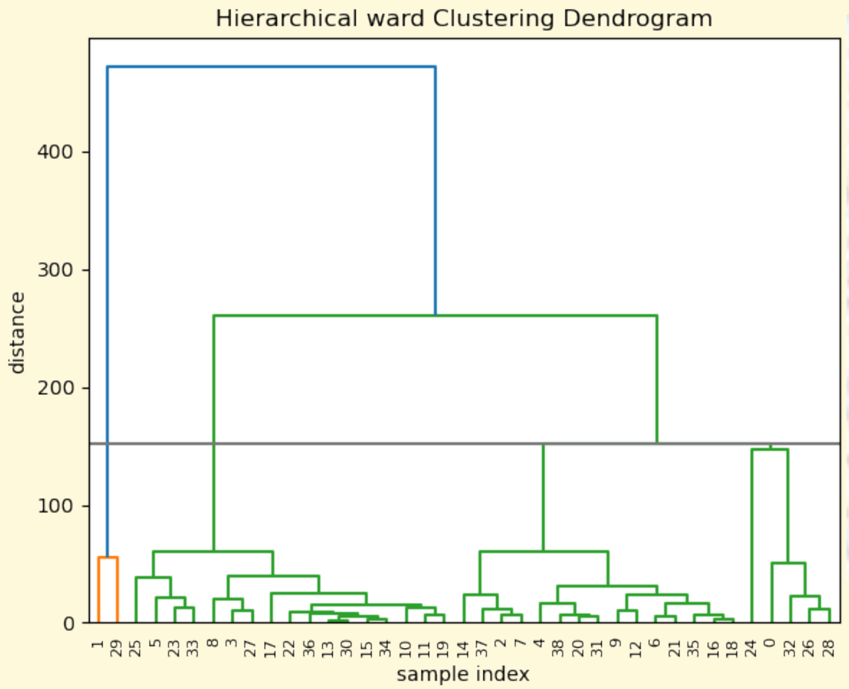
- Galaxy Zoo Bar Lengths: volunteers (citizen scientists) were asked to mark 2 lines: bar length & width
- Agglomerative clustering to combine volunteer marks (31 people looked at this image)
  - Assume volunteers haven't double-marked: d_{cut} chosen to be the highest value that **doesn't** combine 2 marks from the same volunteer into the same cluster. $K = 2$ **not** directly enforced.
  - Given these requirements, Ward distance chosen as best performing $d\left(C_k, C_{k'}\right)$

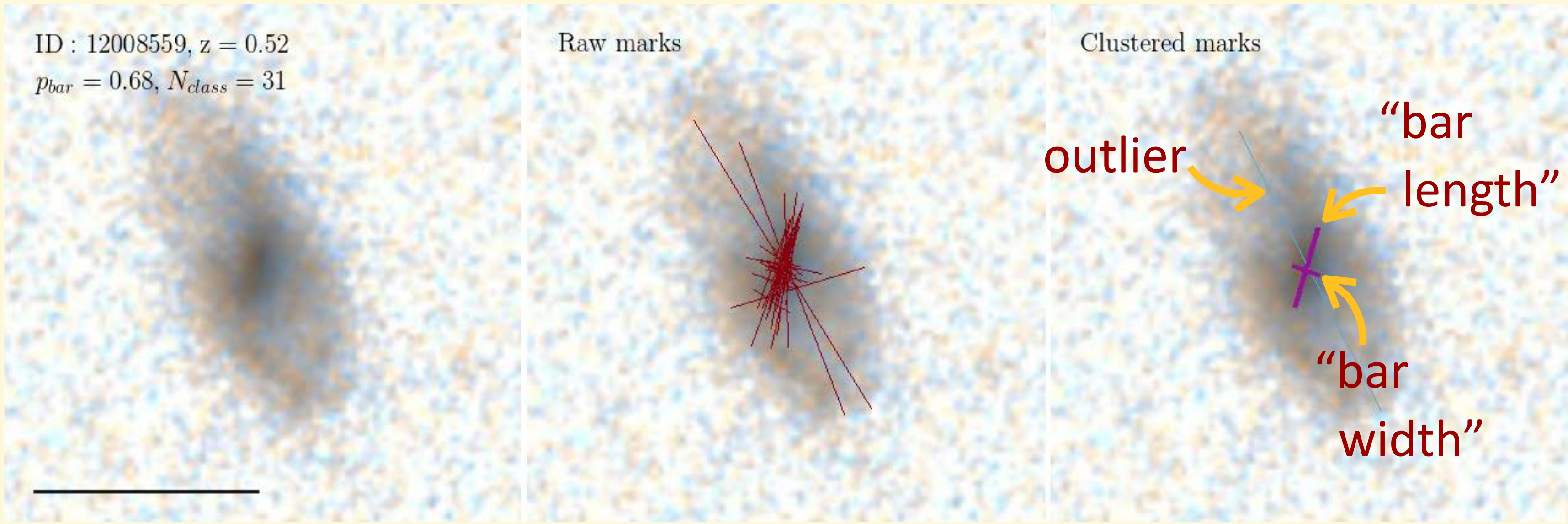




Hierarchical Clustering: choosing d

- Galaxy Zoo Bar Lengths: volunteers (citizen scientists) were asked to mark 2 lines: bar length & width
- Agglomerative clustering to combine volunteer marks (31 people looked at this image)
  - Assume volunteers haven't double-marked: d_{cut} chosen to be the highest value that **doesn't** combine 2 marks from the same volunteer into the same cluster. $K = 2$ **not** directly enforced.
  - Given these requirements, Ward distance chosen as best performing $d\left(C_k, C_{k'}\right)$





One more clustering algorithm: DBSCAN

Density-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise

first published by Ester et al. (1996)

- Defines points as belonging to a cluster or outside a cluster (outliers) based on a distance threshold between points, ε
- If a point q is within ε of a "core point" p, it is "directly reachable" from p.
- A point q is "reachable" from p if there is a path $p_1, \ldots, p_n$ where $p_1 = p$ and $p_n = q$ and where each point p_{i+1} is directly reachable from p_i .
- Points not reachable from any other point are "outliers" or "noise points".
- **Core points** are defined as points where at least $N_{\min}$ other points are directly reachable from the core point.
- $N_{\min}$ (the minimum number of points required to form a cluster) and ε (the maximum separation between adjacent clustered points) are the only 2 parameters specified by the user.

Python: `sklearn.cluster.DBSCAN`

DBSCAN: Advantages

- Don’t need to know the number of clusters in advance
- Can find arbitrarily-shaped clusters
- Has a formalised definition of noise/outliers
- Requires only 2 parameters
- *Mostly* insensitive to the ordering of points in your array/database/table
- Can *sometimes* work out an optimal ε and $N_{\min}$ in advance, if you know your data well enough.

DBSCAN: Disadvantages

- Points on the edge of a cluster can swap membership occasionally if they are in a different order in your array/database/table. Sometimes this might be an advantage, but it usually is an additional source of uncertainty.
- DBSCAN depends highly on the distance measure you use for ε.

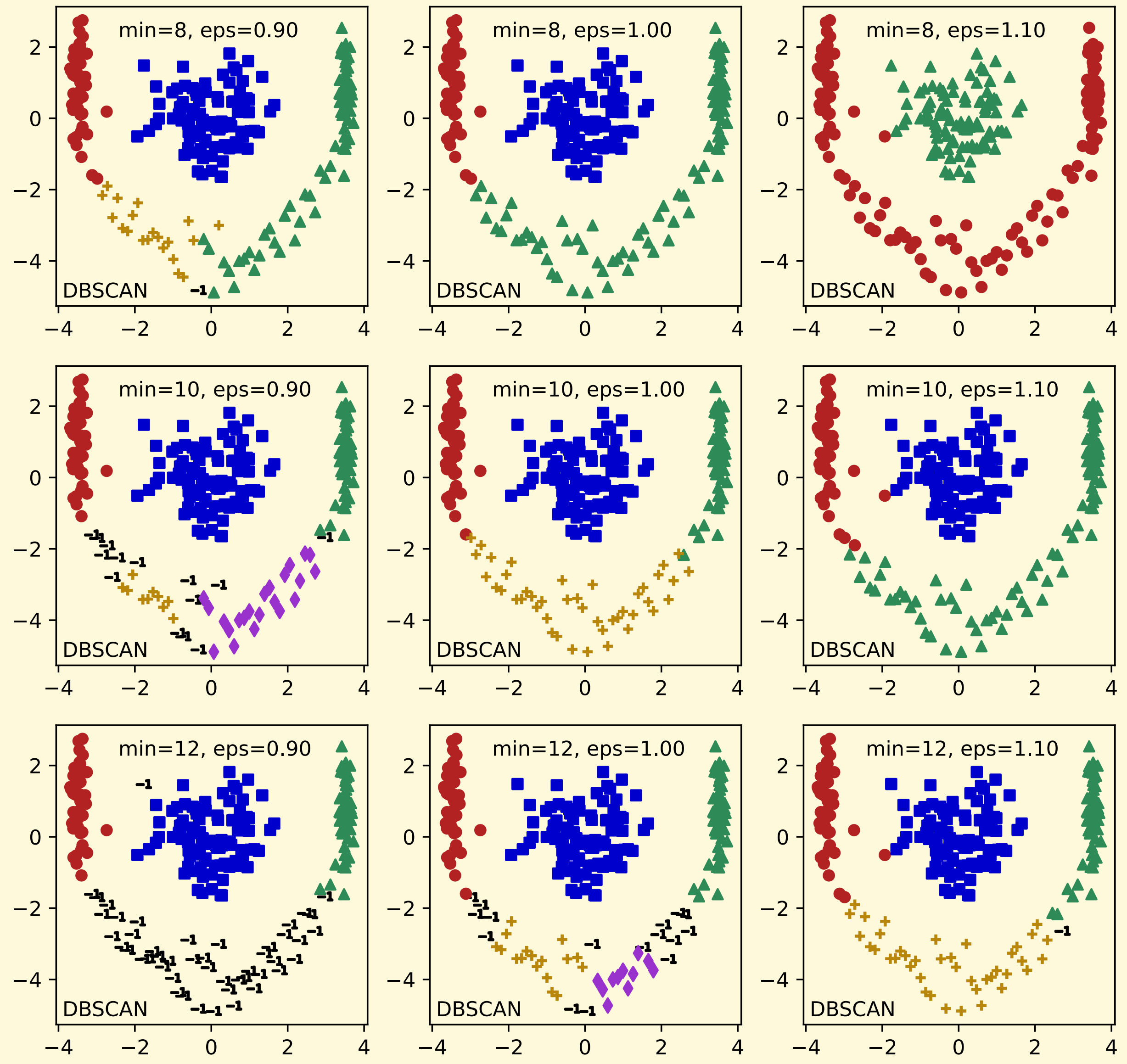
  Most common: Euclidean, e.g. $\sqrt{x^2 + y^2}$ in 2 dimensions.

  For higher dimensional data, it can become almost impossible to find an acceptable value for ε .

  Curse of dimensionality: as dimensionality increases, the volume of parameter space increases so rapidly that the data become very sparse and all objects appear dissimilar.

- Relies on the density being similar between real clusters: can't vary the combination of $(\varepsilon, N_{\min})$ across the dataset
- If you don't know the dataset well, choosing $(\varepsilon, N_{\min})$ can be very challenging.

DBSCAN, with our example dataset



- While DBSCAN can have advantages over K-means and agglomerative hierarchical clustering, you still need to try different parameters and examine your data to choose the values that will provide accurate, robust cluster membership.
- To explore on your own: HDBSCAN (basically: like DBSCAN, but hierarchical)

Clustering Recap

- **We have discussed** “algorithmic clustering”:
  - K-means
  - Hierarchical linkage (agglomerative)
  - DBSCAN

- **We have not discussed** “statistical clustering”:
  - Parametric: associates a specific model with the density in each cluster (*e.g.,* Gaussian, Poisson)
  - Non-parametric: examines contours of density to find cluster information (*e.g.,* kernel density estimate)

There are many ways to cluster data, and we have covered the start of them.

Clustering and Classification

Clustering

Find subtypes or groups that are not defined *a priori* based on measurements

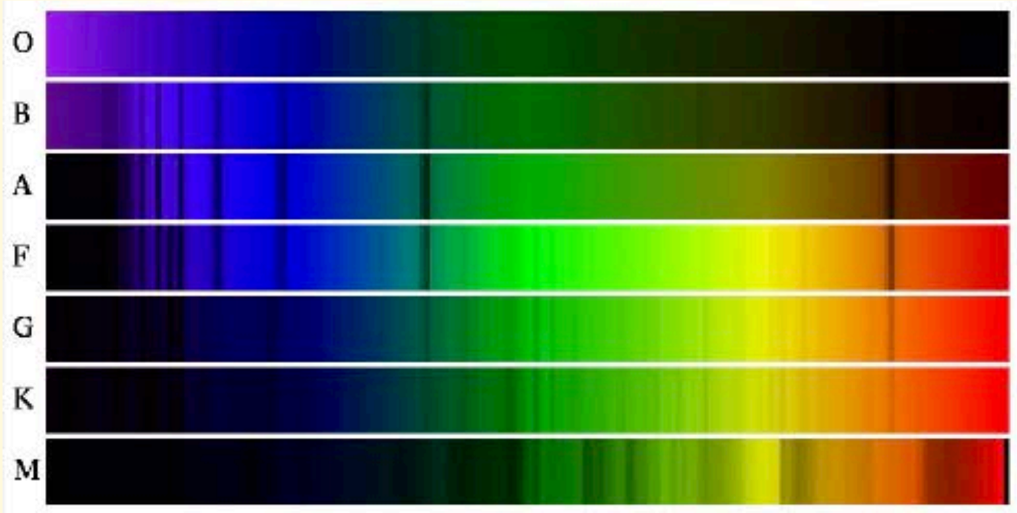
➞ "Unsupervised learning" or "Learning without labels"

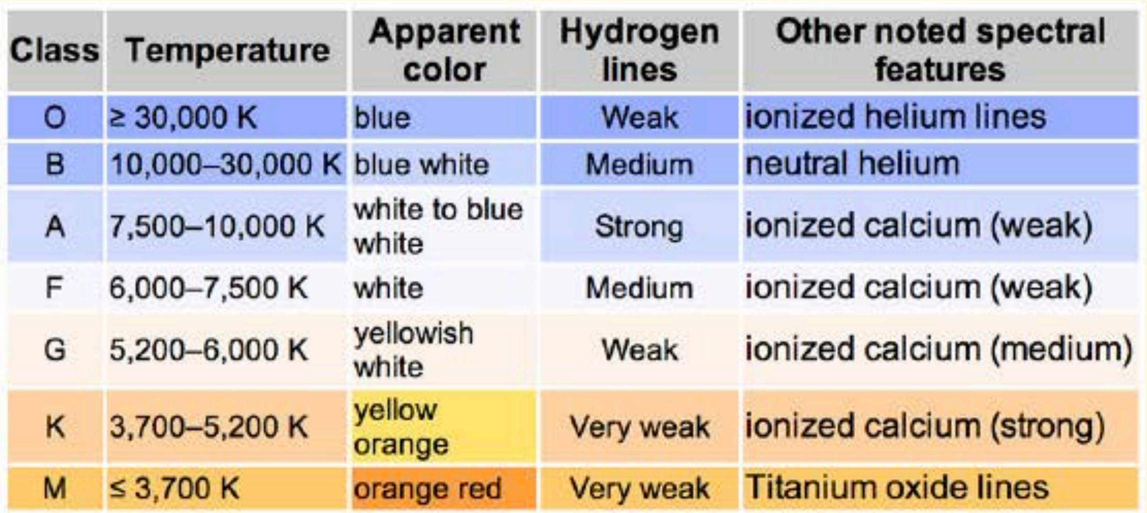
Classification

Use *a priori* group labels in analysis to assign new observations to a particular group or class

➞ Sometimes applied once some type of clustering has been applied; can be separate

➞ "Supervised learning" or "Learning with labels"





Class	Temperature	Apparent color	Hydrogen lines	Other noted spectral features
O	≥ 30,000 K	blue	Weak	ionized helium lines
B	10,000–30,000 K	blue white	Medium	neutral helium
A	7,500–10,000 K	white to blue white	Strong	ionized calcium (weak)
F	6,000–7,500 K	white	Medium	ionized calcium (weak)
G	5,200–6,000 K	yellowish white	Weak	ionized calcium (medium)
K	3,700–5,200 K	yellow orange	Very weak	ionized calcium (strong)
M	≤ 3,700 K	orange red	Very weak	Titanium oxide lines

Classification: the setup

Given vectors $\boldsymbol{X} = \{\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n\} \in \mathbb{R}^p$ and qualitative class labels $\boldsymbol{Y} = \{y_1, y_2, \ldots, y_n\}$

$\longrightarrow$ let $\hat{y}_i$ be the predicted label for observation i

$\longrightarrow$ our main interest is the probability space, $P\left(\boldsymbol{Y} | \boldsymbol{X}\right)$

- The classification **training error** rate is often estimated using a training dataset as

$$\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{I}\left(y_i \neq \hat{y}_i\right)$$

where $\boldsymbol{I}(\cdot)$ is called the indicator function, and returns 1 or 0 for a label being True/False.

- The classification **test error** rate is often estimated using a test dataset, $\left(x_{\text{test}}, y_{\text{test}}\right)$, as

$$E\left(\boldsymbol{I}\left(y_{\text{test}} \neq \hat{y}_{\text{test}}\right)\right)$$

$\longrightarrow$ Supervised classifiers try to minimise the training error.
Good classifiers have small test errors.

Classification: some example classifiers

Bayes classifiers

Assign label to the class that has the largest probability $P\left(Y=j \mid X=x\right)$ for classes $j=1,\ \ldots\ ,J$, assuming you know the distribution of $Y \mid X$ (which you sometimes don't)

K Nearest Neighbours (KNN)

Assign label based on the K observations in the training set that are "nearest" to it.
Note: **<u>not</u>** the same K as K-means clustering.

Linear Classifiers

Decision boundary between classes is linear. ~Simple, but not good for all datasets.
Ex: Logistic regression (binary), Linear Discriminant Analysis (multi-class)

Other Classifiers

Support Vector Machines: find the hyperplane that maximises the distance between classes

Classification Trees: determine which variables are "best" at separating data into labelled groups, by partitioning the predictor space into hyper-rectangles. Often physically easier to interpret than other methods.

Convolutional Neural Networks: layers of matrices which iteratively operate on each other to minimise training loss. Can be very difficult to interpret why they make decisions.

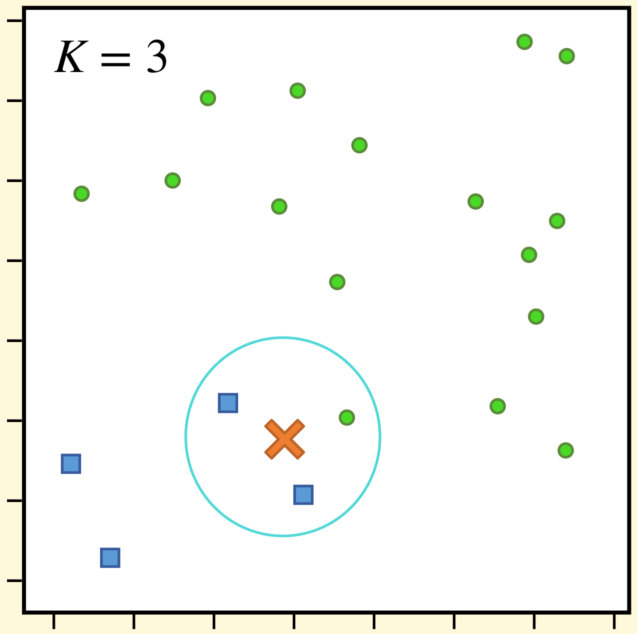
Example: K Nearest Neighbours (KNN)

Main idea: an observation is classified based on the K observations in the training set nearest to that observation.

A probability of each class can be estimated by

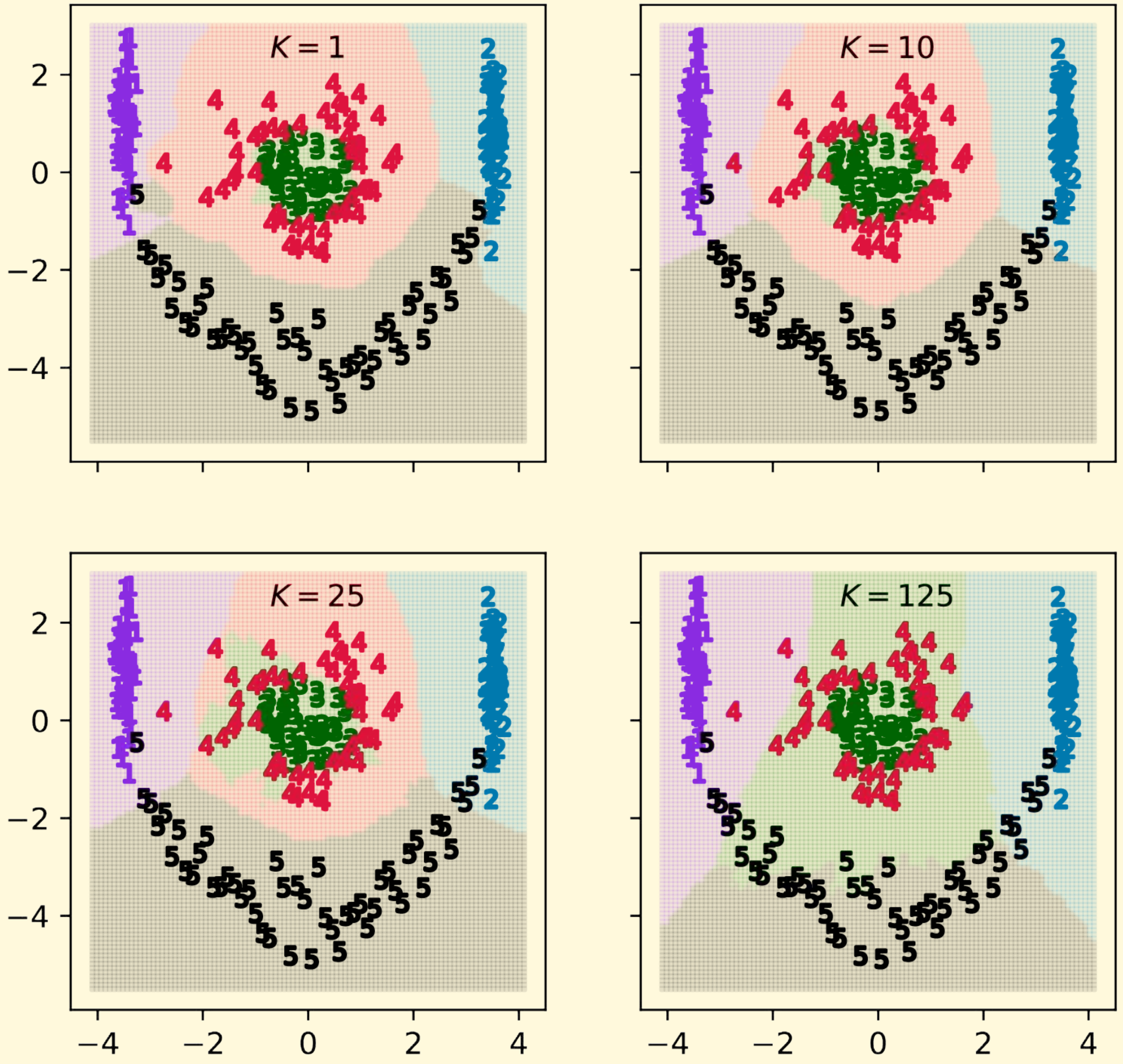
$$P\left(Y = j \mid X = x\right) = \frac{1}{K} \sum_{i \in N(x)} \boldsymbol{I}\left(y_i = j\right)$$

where, if there are N classes in the training set, $j = 1, \ldots, N$, and $\boldsymbol{I}$ is the indicator function.



- There are $K = 3$ nearest neighbours to the **X** within the circle.
- The predicted class of **X** would be "blue square" because there are more blue square observations than green circles among the 3 NN.

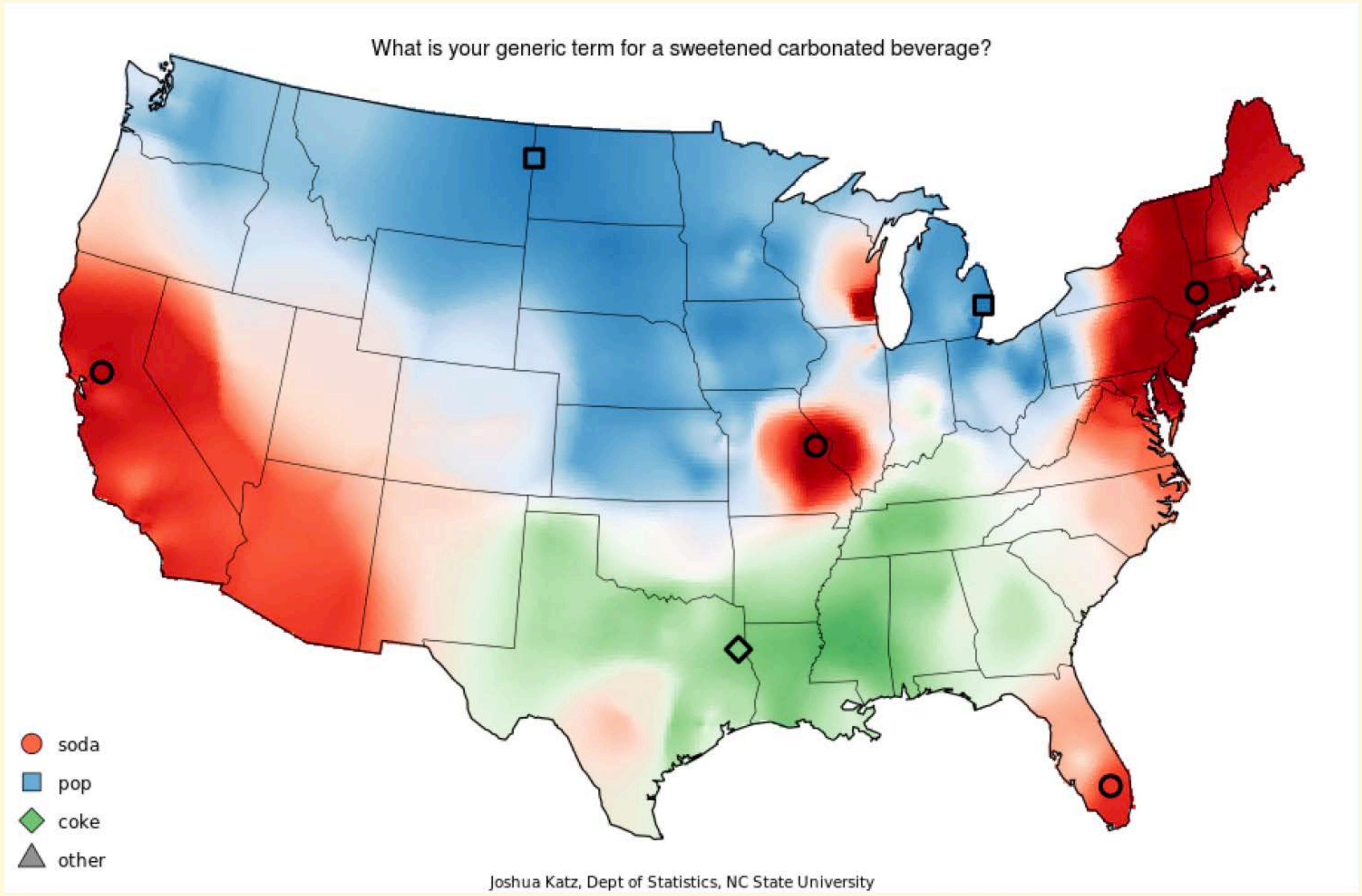
K Nearest Neighbours (KNN): example data



- Numbered points show training data and true labels (`data_u`).
- Coloured meshpoints show what the predicted label would be at that point, for each value of K (top centre).
- Smaller K is more likely to capture small-scale features, but risks overfitting.
- Larger K loses information about real class boundaries.

python:
`sklearn.neighbors.KNeighborsClassifier`

K Nearest Neighbours (KNN): real data


What is your generic term for a sweetened carbonated beverage?
soda
pop
coke
other
Joshua Katz, Dept of Statistics, NC State University

- Predicts a probability space for dialect usage
- Could be used to predict labels (was not, in this case)

Credit: Josh Katz, was North Carolina State University,
now New York Times