INTRODUCTION TO MACHINE LEARNING

INTRODUCTION

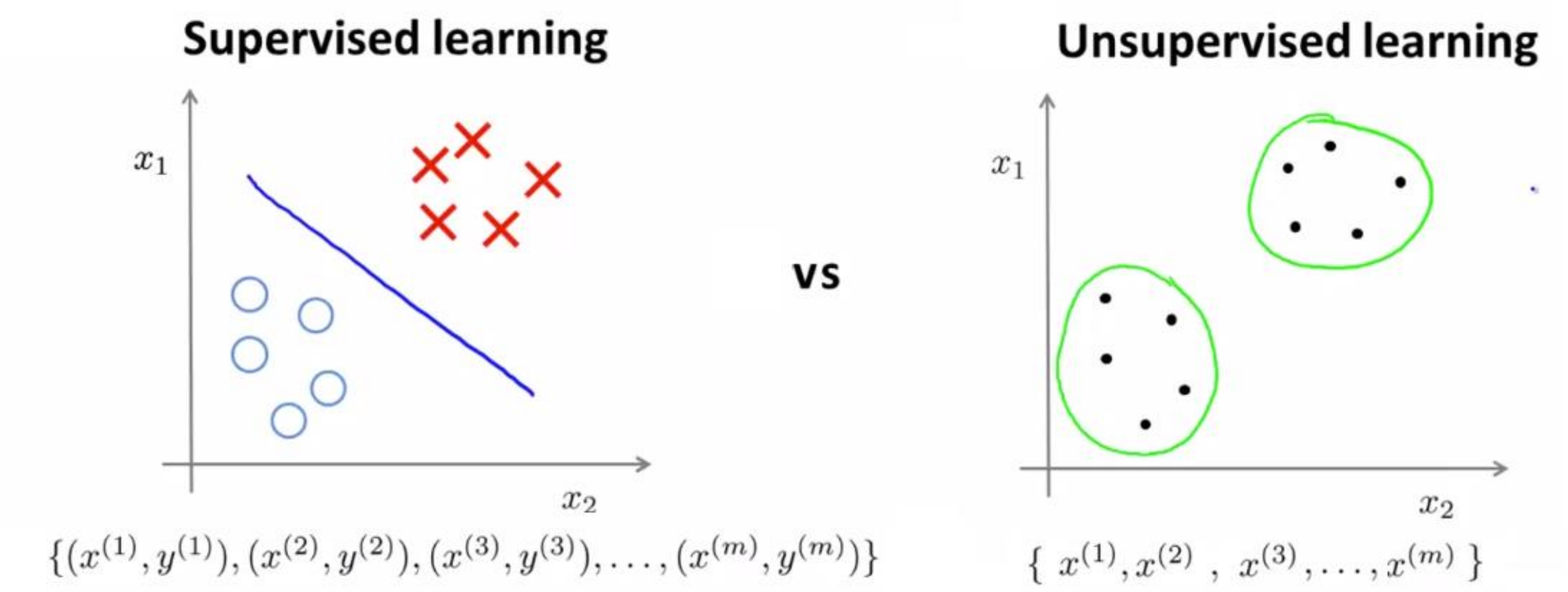
Andreu Arderiu

NORTHIN SUMMER SCHOOL

Barcelona, July 2022

Unsupervised learning: overview

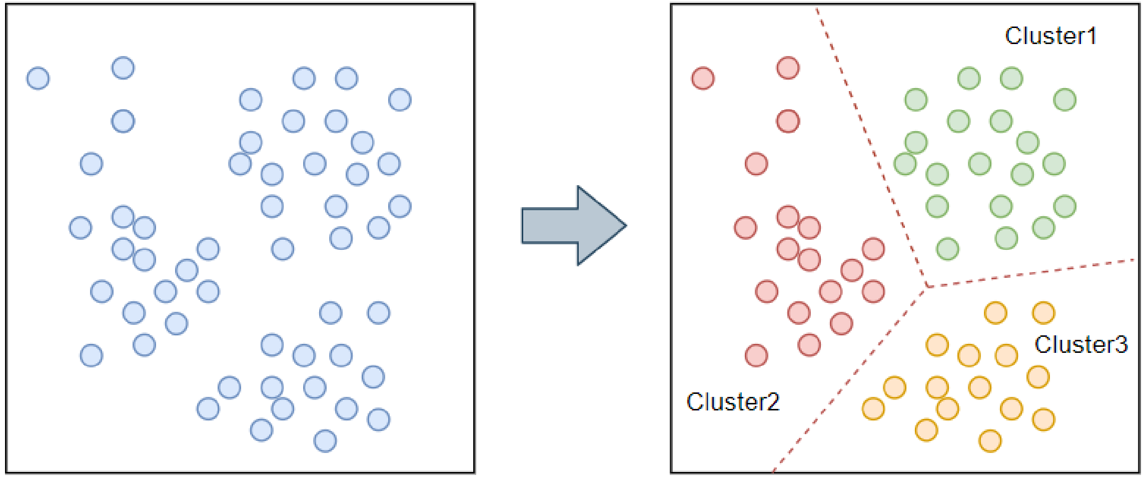
- **Unsupervised**: Only **inputs** X are given. We compute f such that $y = f(X)$ is a "simpler" representation
  - **Clustering:** discrete y (groups)
  - **Dimensionality reduction**: continuous y



Source: coursera

Clustering :overview

- Given a **set of points** and a notion of **distance** between points, **group** them into a number of **clusters** so that
  - members of the **same cluster** are **close**
  - members of **different clusters** are **far** from each other

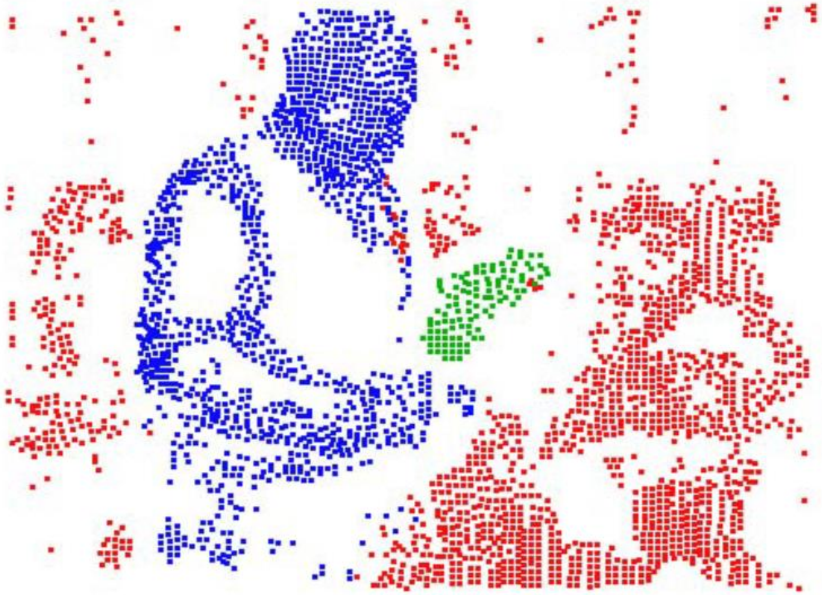


Source: ecloud valley

Clustering: use cases

- Data exploration
- Image or data segmentation
- …

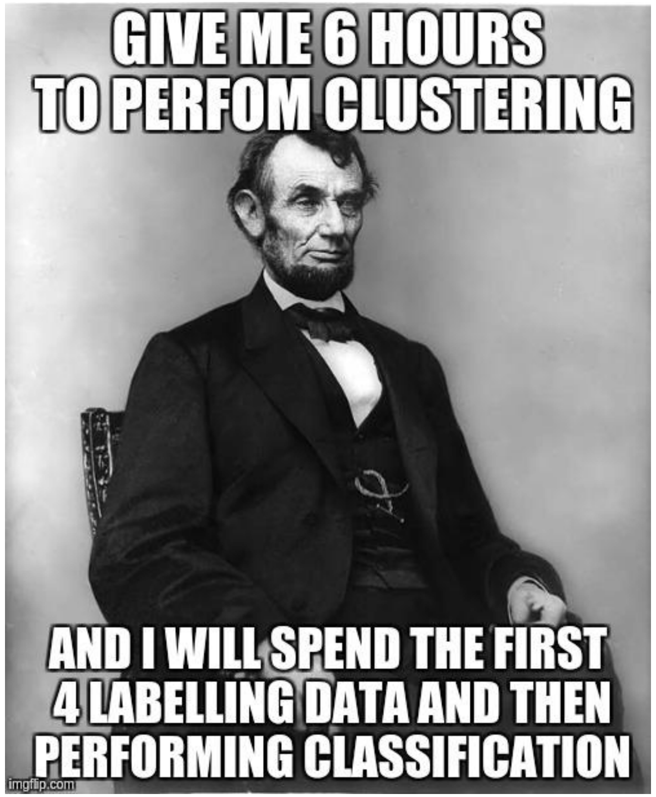
Source: Brox and Mali, ECCV 2012



Clustering: a hard problem!

- **Clustering** in **2D** and small amounts of data looks **easy**, but….
- Usually data have lots of features, **high-dimensional spaces**!

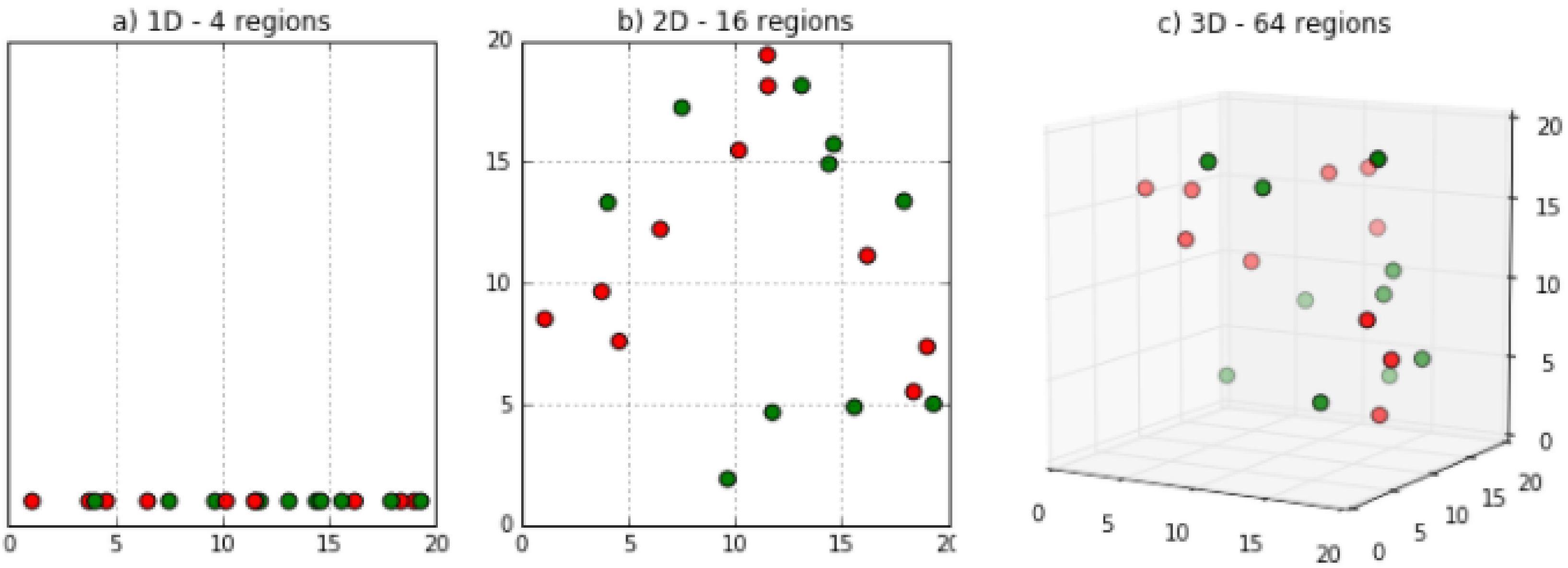
→ **Curse of dimensionality**: In very high dimensions, almost all pair of points are about at the same distance



Source: Imgflip

Curse of dimensionality

- As dimensionality grows, there are fewer observations per region.
  - 1d: 4 regions, 2d: 4^2 regions, 1000d: hopeless

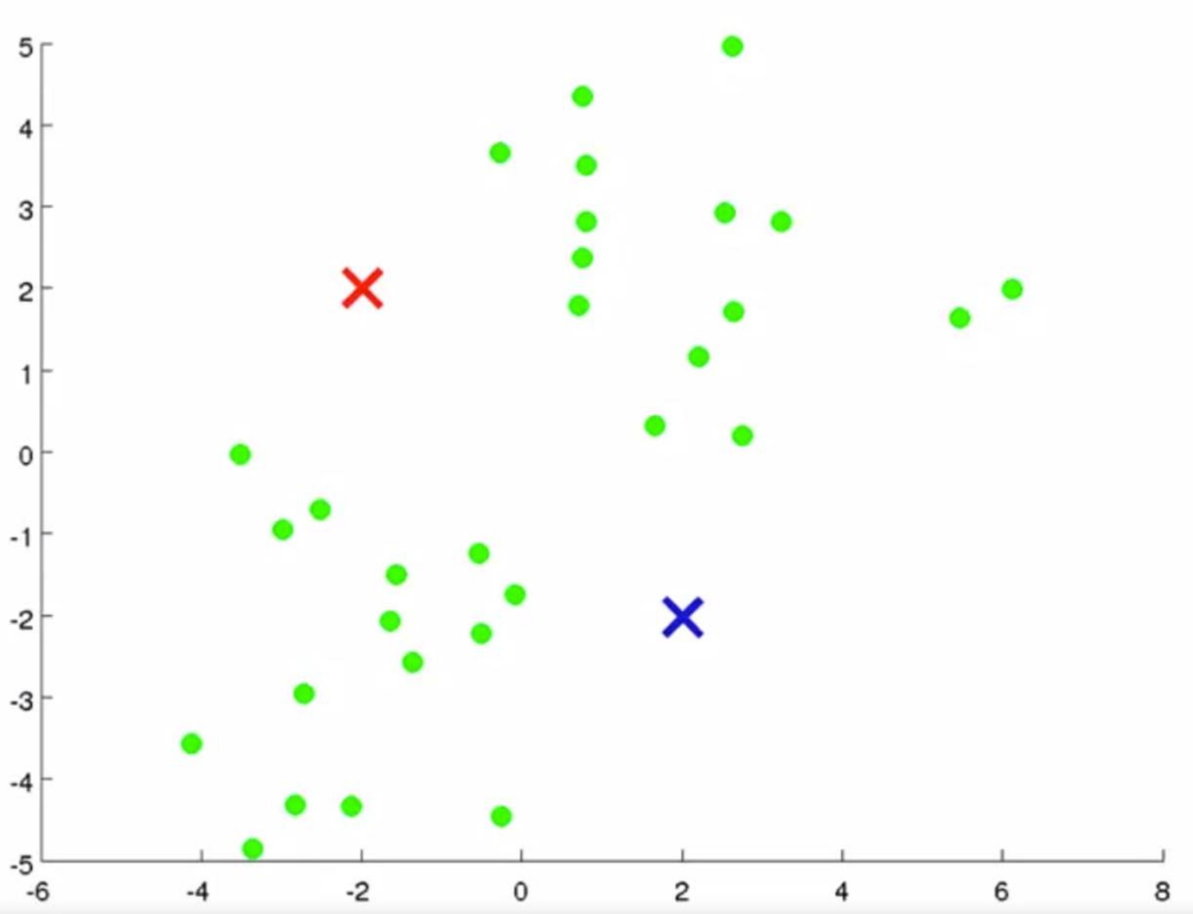


Source: Prasanth Damodharan

Clustering: K-means algorithm

- **Goal**: **assign** each **data** point to one of **k clusters** so that the total **distance** of **points** to their **centroids** is **minimized**
- K-means algorithm:
    1. Randomly initialise K cluster centroids
    2. Assign points to their closest centroid
    3. For each cluster recompute its centroid (mean coordinates of points of the cluster)
    4. Iterate 2-3 until convergence
- Usually **Euclidean** distance, but other alternatives can be also used

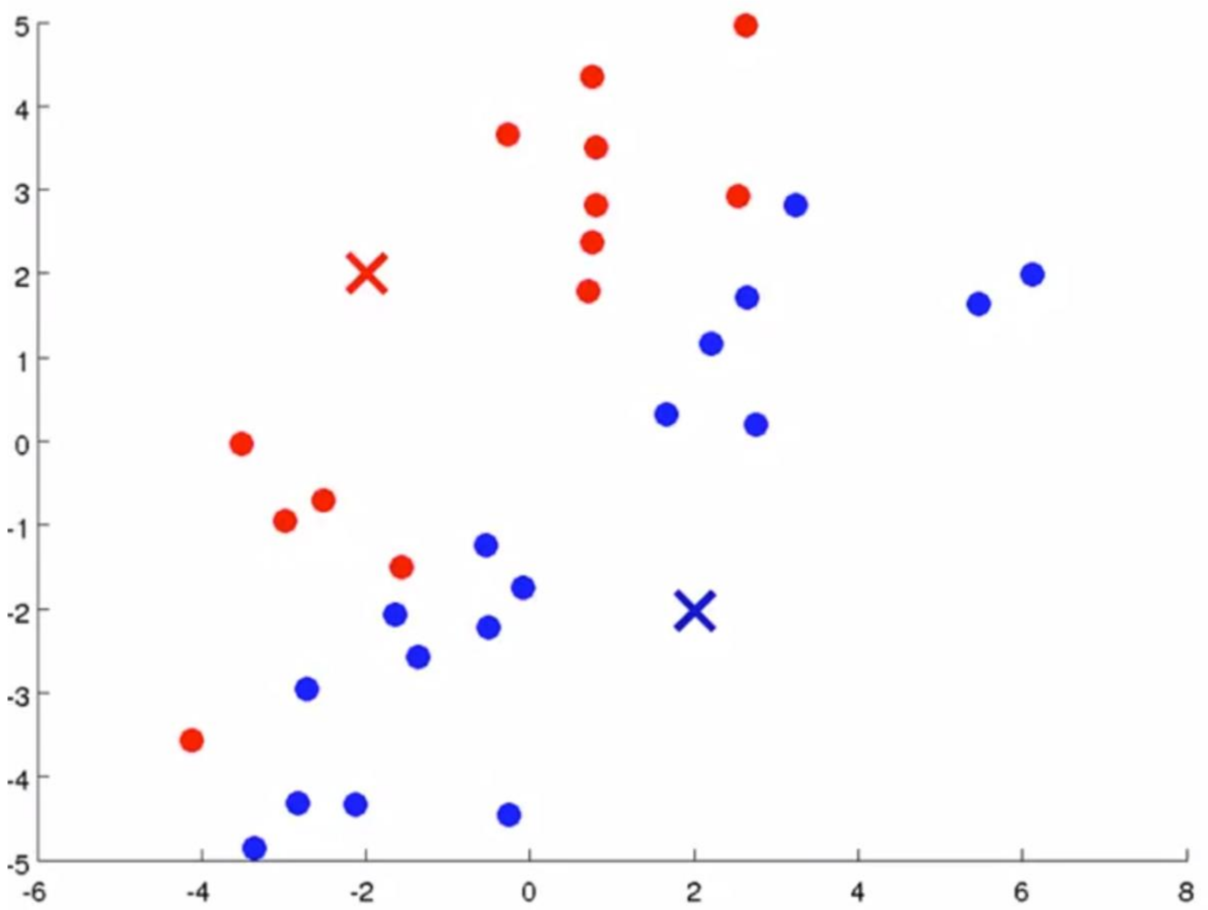
K-means algorithm example (k=2)



Initialize cluster centroids

Credit: Guillaume Dehaene, EPFL

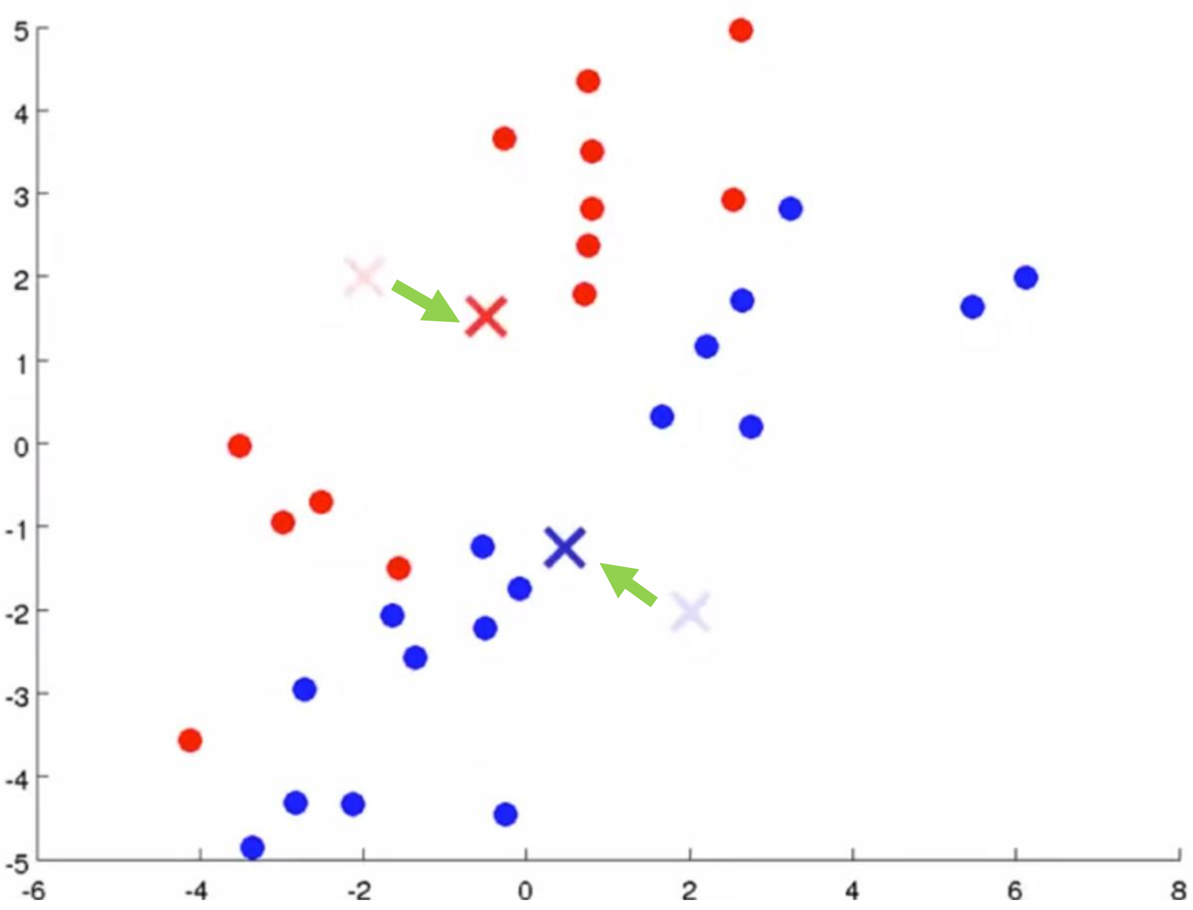
K-means algorithm example (k=2)



Assign points to cluster

Credit: Guillaume Dehaene, EPFL

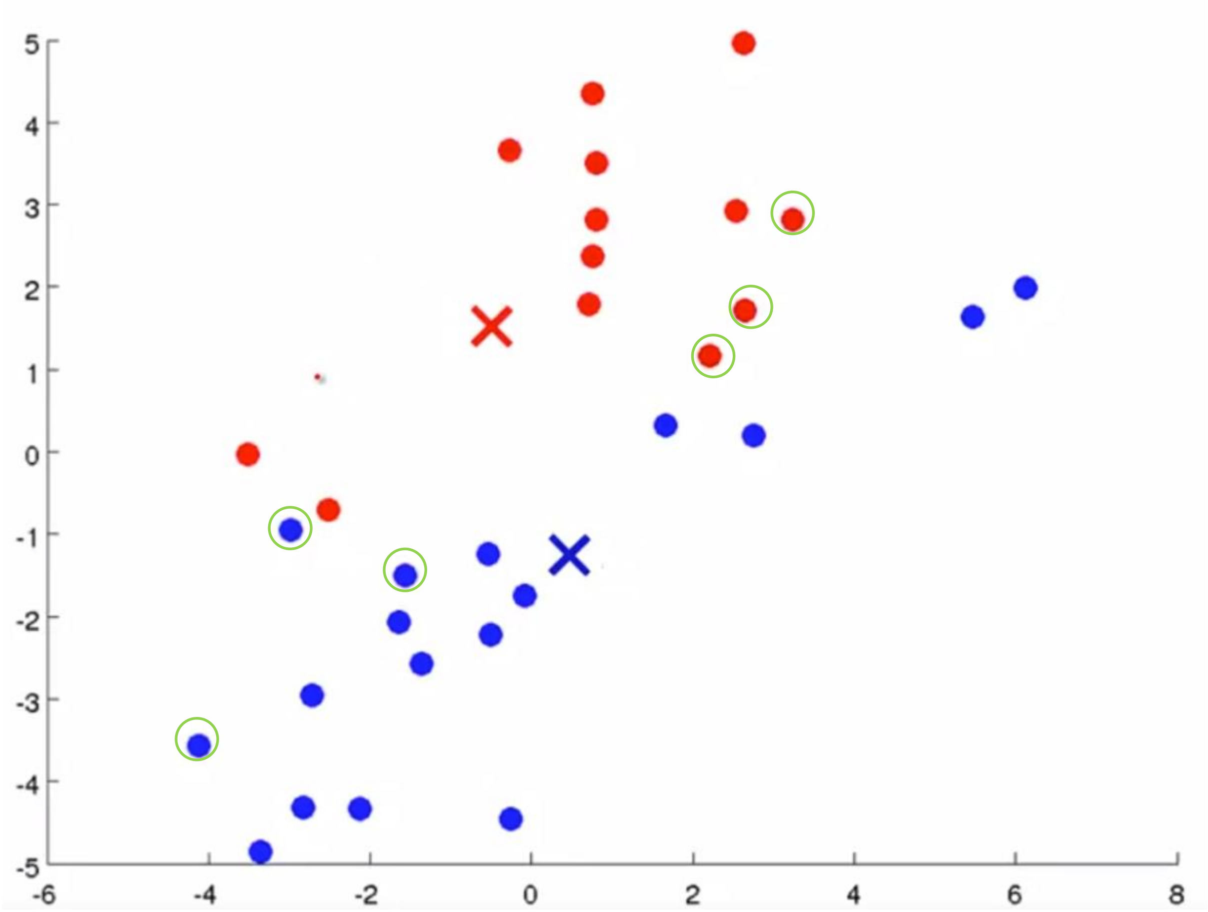
K-means algorithm example (k=2)



Update centroids

Credit: Guillaume Dehaene, EPFL

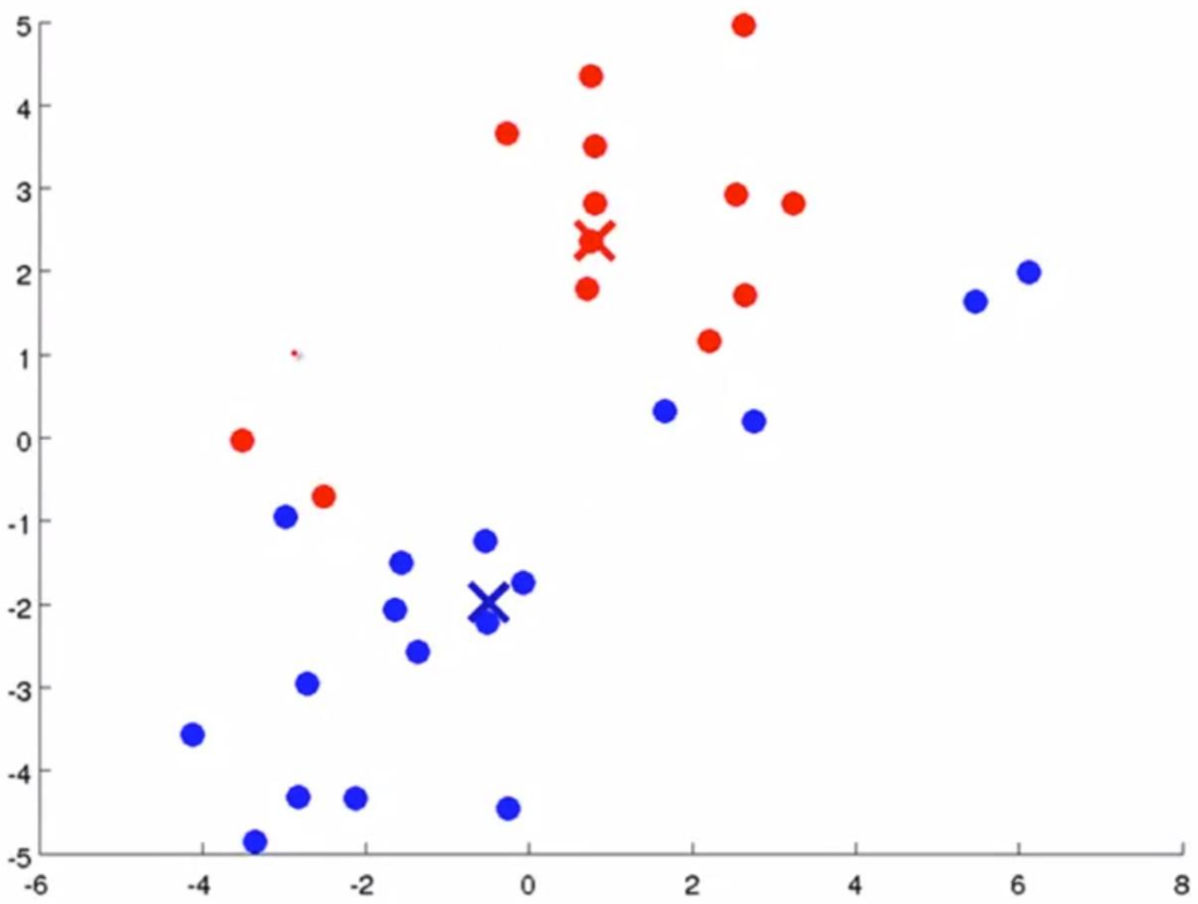
K-means algorithm example (k=2)



Assign points to cluster

Credit: Guillaume Dehaene, EPFL

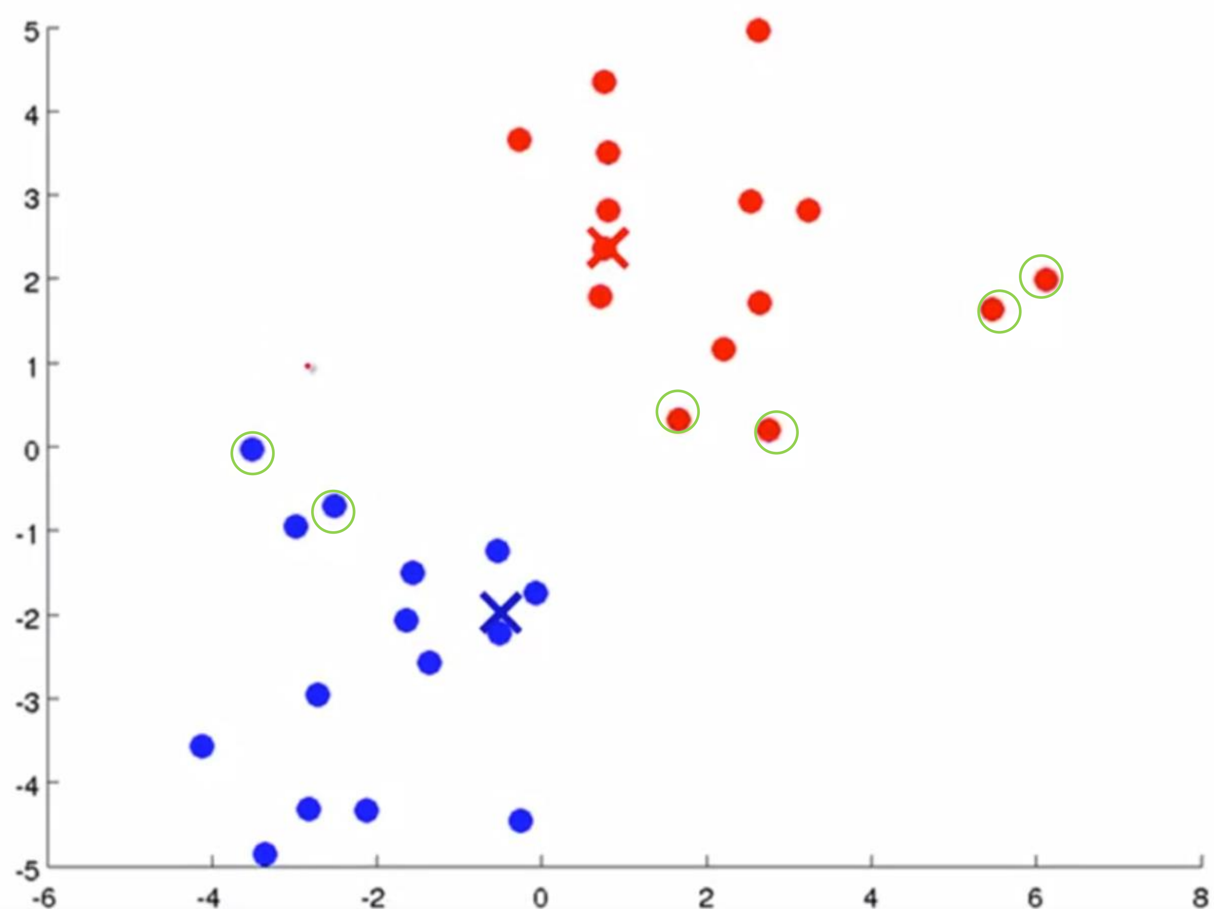
K-means algorithm example (k=2)



Update centroids

Credit: Guillaume Dehaene, EPFL

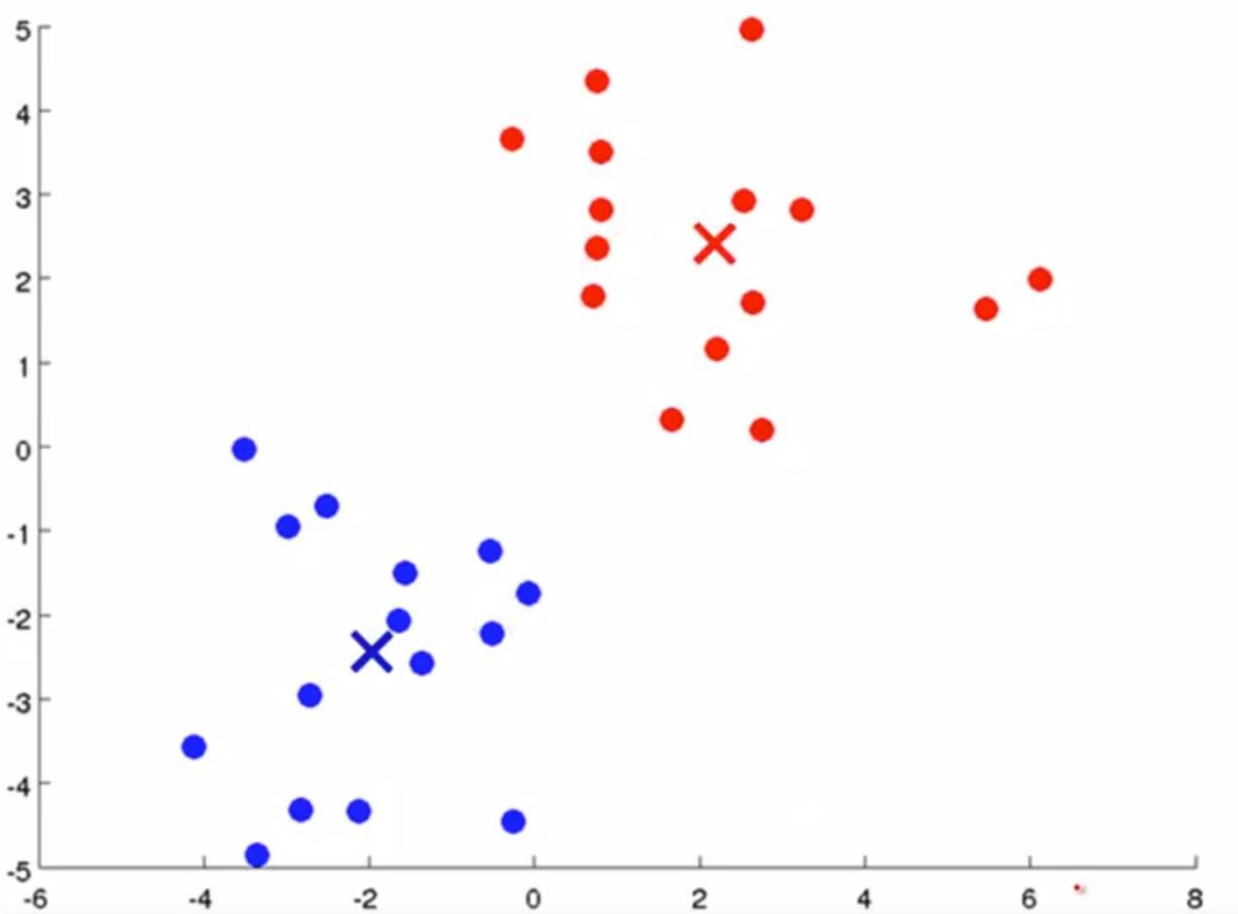
K-means algorithm example (k=2)



Assign points to cluster

Credit: Guillaume Dehaene, EPFL

K-means algorithm example (k=2)



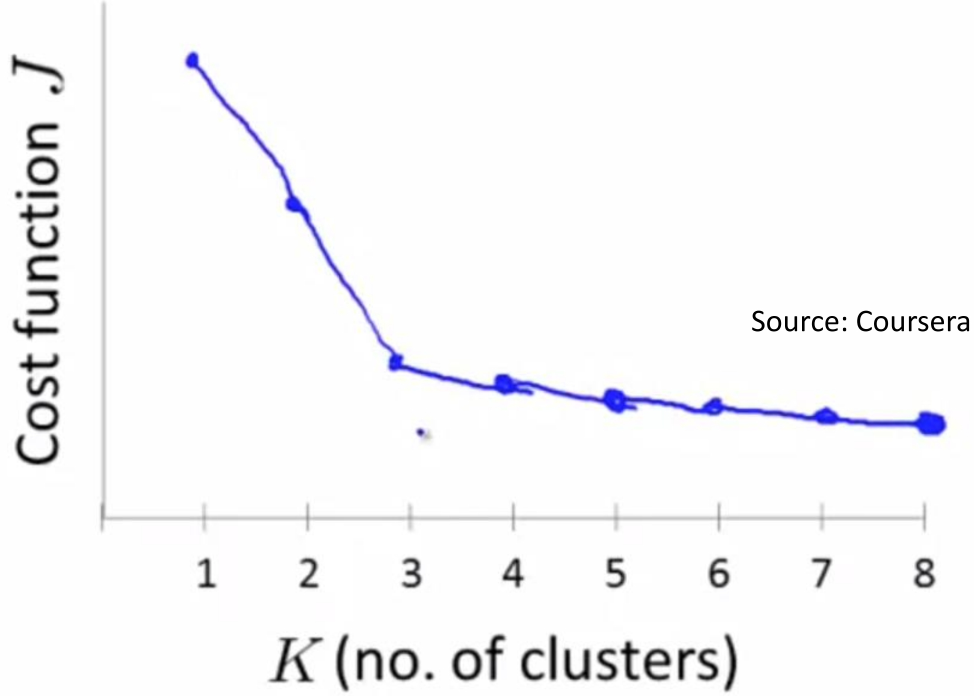
Update centroids
Done!

Credit: Guillaume Dehaene, EPFL

K-means: choosing k

- **Elbow method**: run k-means with different values of k and compute certain cost function J (usually "silhouette")

- QUIZZ: Which k should we choose ?



Source: Coursera

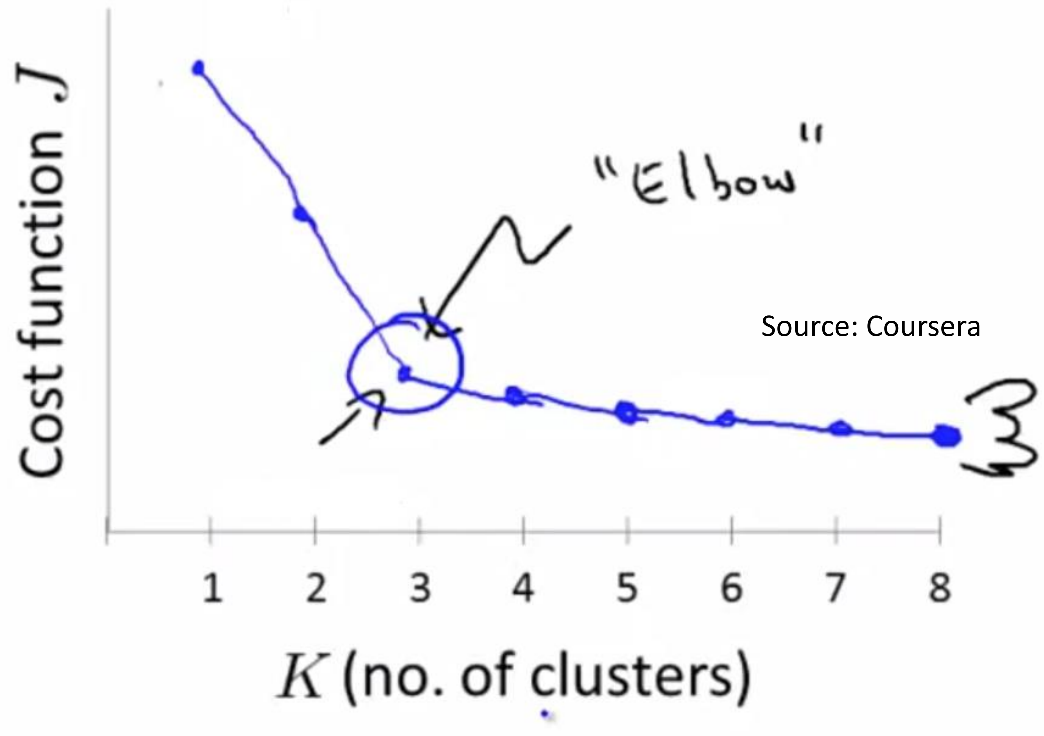
K-means: choosing k

- **Elbow method**: run k-means with different values of k and compute certain cost function J (usually “silhouette”)

- QUIZZ: Which k should we choose ?

?

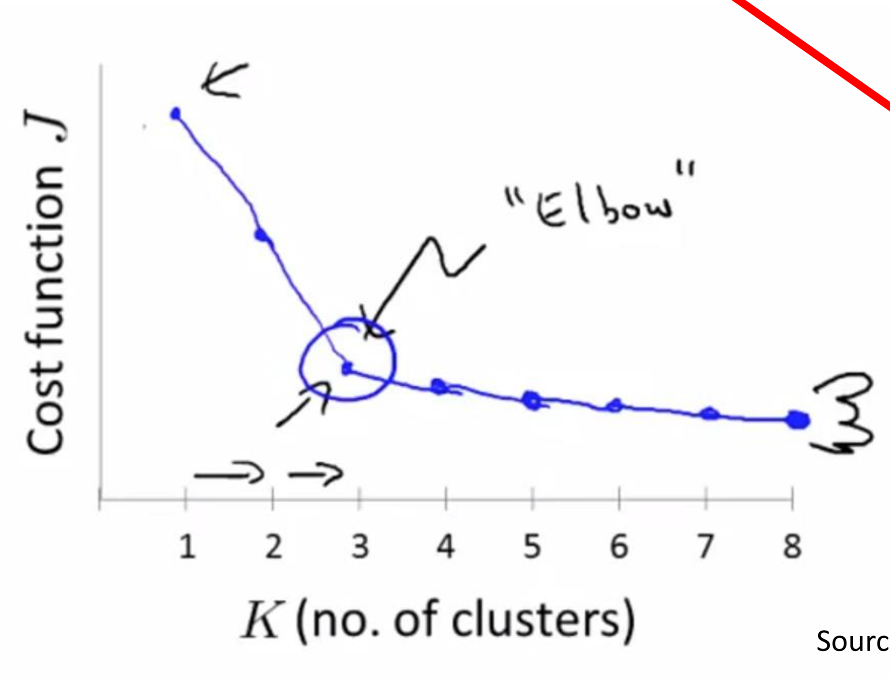
→ **K = 3**



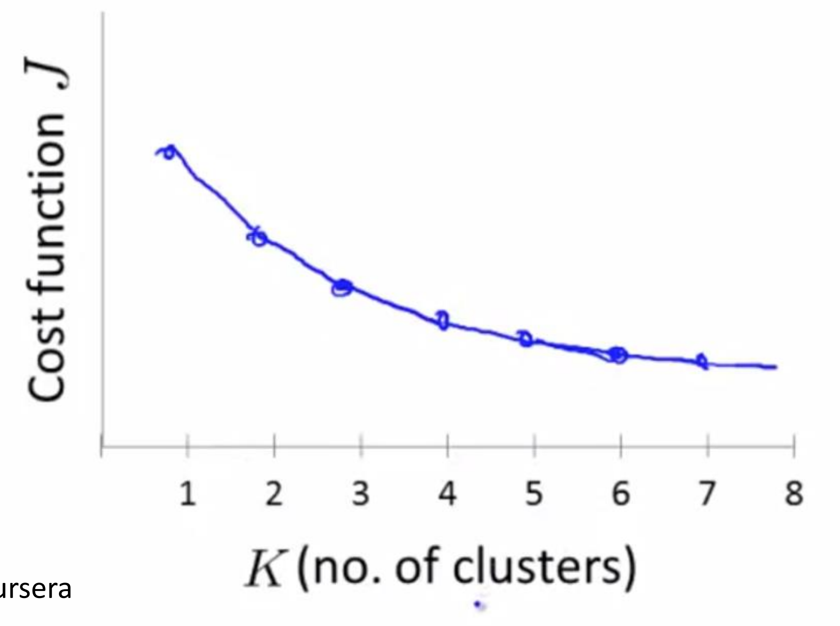
Source: Coursera

K-means: choosing k issues

- Usually "elbow" is not visible, and the choice of k is somewhat arbitrary



Source: Coursera



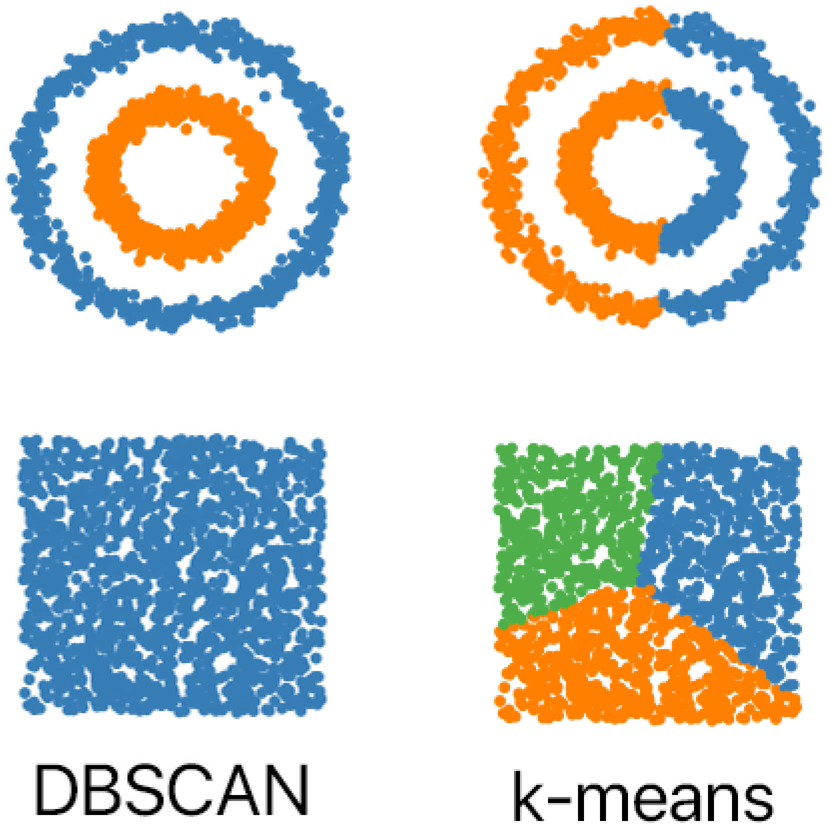
K-means: pros and cons

- Pros:
  - Simple and easy to interpret
  - Efficient and scales well with large datasets
  - Convergence is guaranteed

- Cons:
  - Choice of k is not trivial
  - It is sensitive to centroid initialisation
  - Not robust to outliers
  - Poor performance in very high dimensions
  - Poor performance with “complex” shape data

Clustering alternative: DBSCAN

- “**D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise”
- Does not require k choice
- Good performance with “complex” shapes
- Density-based clustering, follows the shape of dense neighbourhoods of points
- Slower than k-means, but no need to choose k a priori



Dimensionality reduction: overview

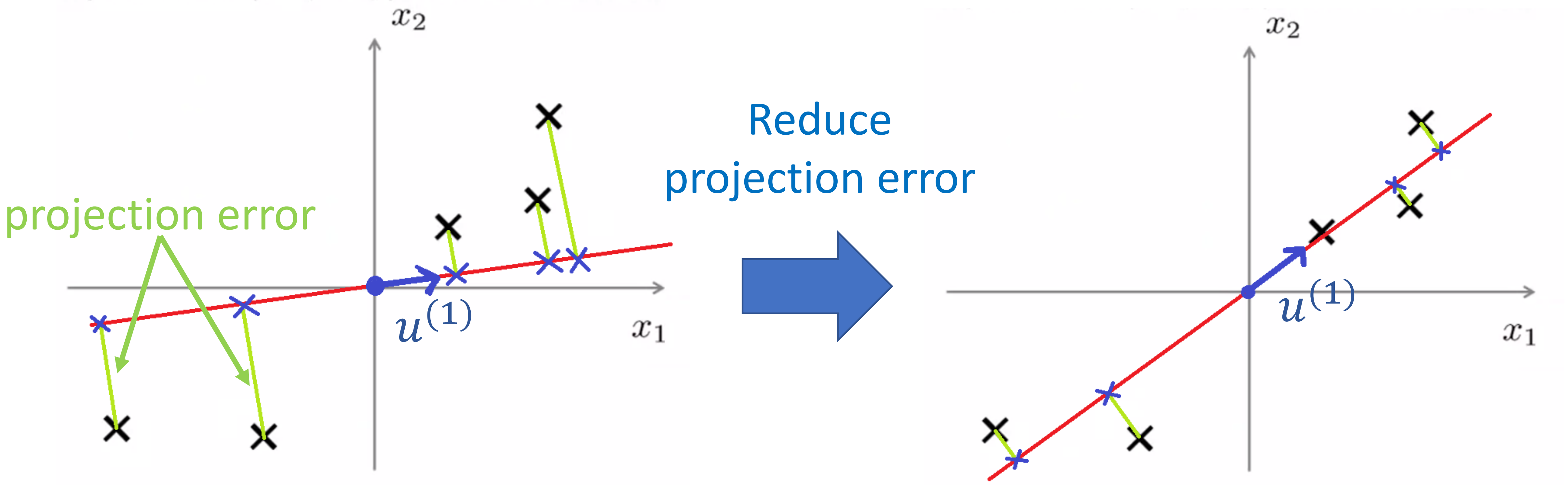
- Two main motivations:

1. **Data compression**: Save computer **memory** and **speed up learning algorithms**

2. **Visualization**: Plots in 2D, 3D are human-**interpretable**

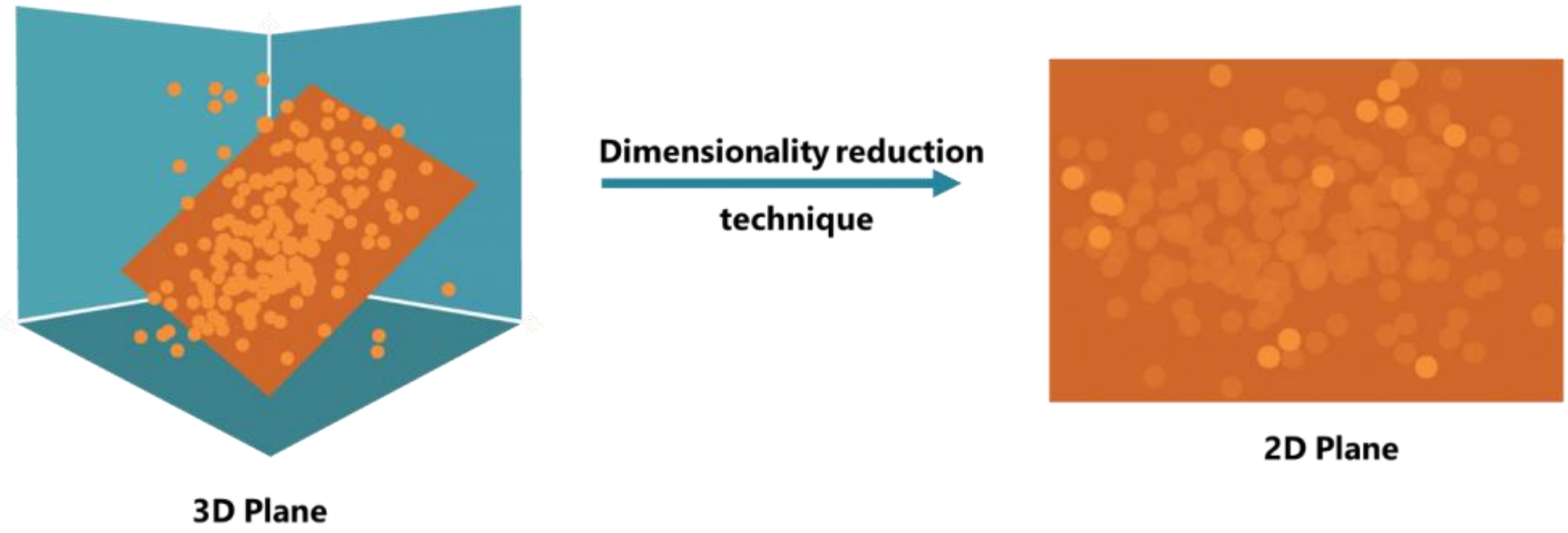
PCA: intuition

- Reduce from n-dimension to k-dimension: find k vectors $u^{(1)}, \ldots,$ $u^{(k)} \in \mathbb{R}^n$ onto which the **projection error** of the data is **minimized**



PCA: intuition

- Reduce from 3D to 2D: find 2 vectors $u^{(1)}, u^{(2)} \in \mathbb{R}^3$ onto which the **projection error** of the data is **minimized**



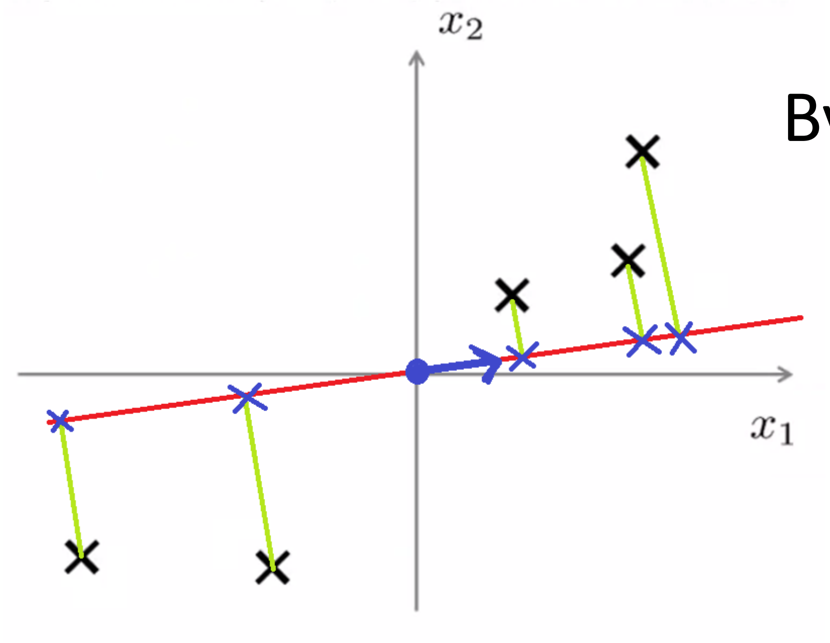
PCA: algorithm summary

1. **Normalize** all input features: $x \rightarrow (x - \bar{x})/\mathrm{sd(x)}$
2. Compute **covariance** matrix $\Sigma = \frac{1}{n}\sum_{i=1}^{n}\left(x^{(i)}\right)(x^{(i)})^T$
3. Compute Σ **eigenvectors** ($u^{(j)}$, or **principal components** PC) and **eigenvalues** (**variance** of data along eigenvectors $u^{(j)}$)
4. Choose top k PC with highest eigenvalues.
5. Project original normalised data on PCA space: dot product of data by chosen $u^{(k)}$

PCA: choice of dimension (k)

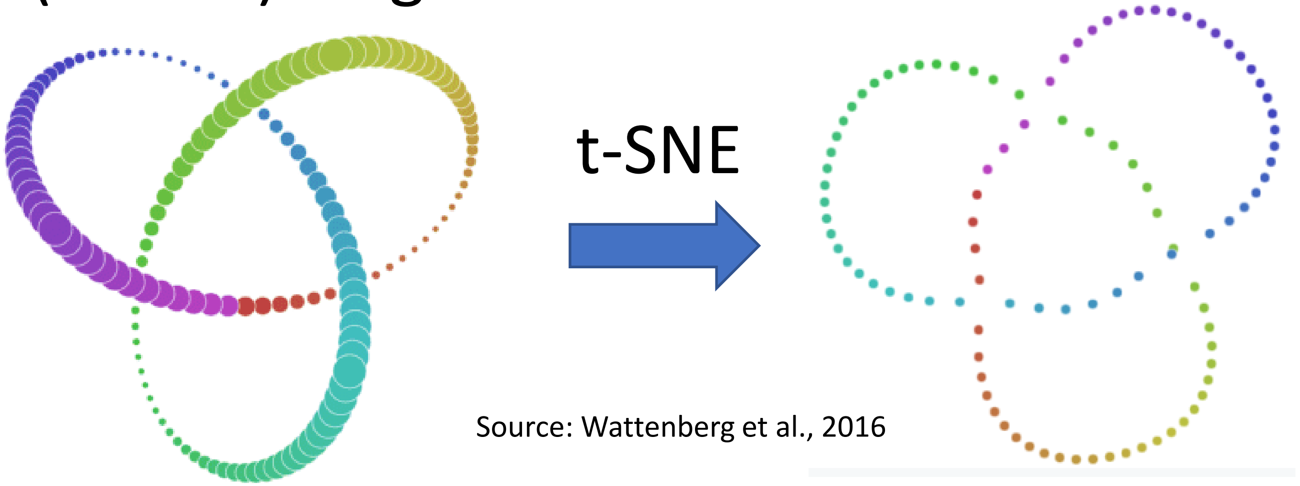
- Typically choose smallest k so that “99% of variance is retained”
- $\frac{\text{Average squared projection error}}{\text{Total variation in the data}} \leq 0.01$

By projecting we “loose” 1% of the variance



Alternative for visualization: t-SNE

- t-distributed **S**tochastic **N**eighbour **E**mbedding
- While PCA preserves global structure of the data, t-SNE preserves only local similarities
- Non-linear method, very flexible preserves local structures that with other methods (as PCA) might be lost



Source: Wattenberg et al., 2016