UiO : University of Oslo

institutt for informatikk

IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning

Lecture 11

Unsupervised Learning

Pooya Zakeri

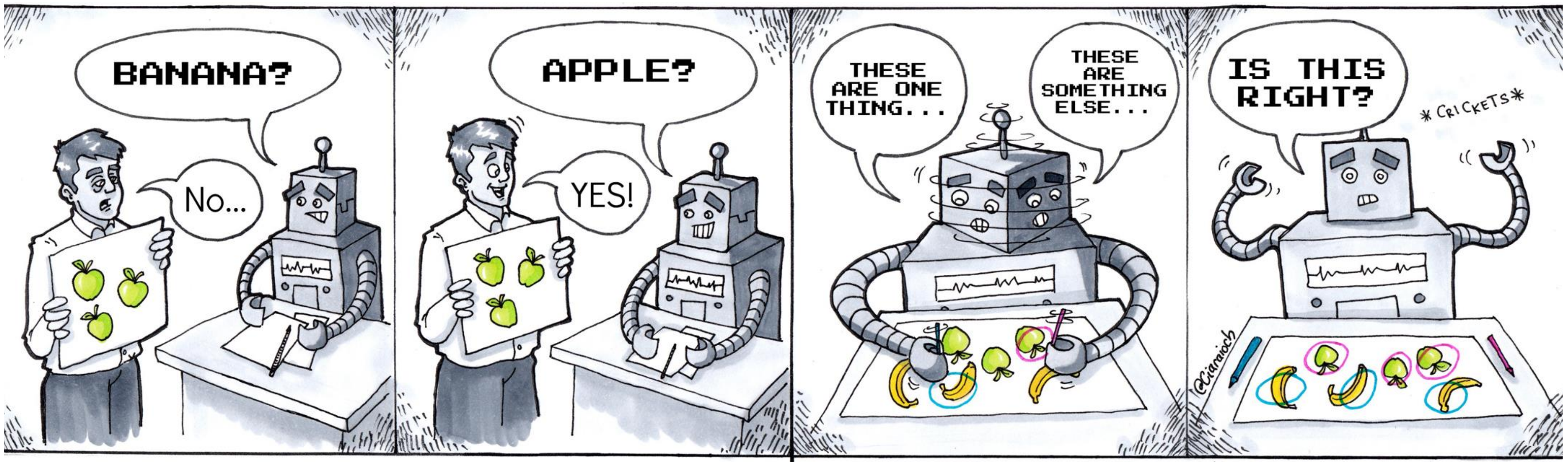
Slides are compiled from many sources; thanks to those who made their slides available online; most of the slides are by Ole Christian Lingjærde and Andrew Ng.

IN3050/IN4050, Lecture 10
Unsupervised Learning

1: Introduction
2: Learning how to generate similar data
3: Learning how data are grouped
4: Learning how to compress data
5: Learning how to represent data in low dimension

IN3050/IN4050, Lecture 10
Unsupervised Learning

1: Introduction
Ole Christian Lingjærde



Supervised Learning

Unsupervised Learning

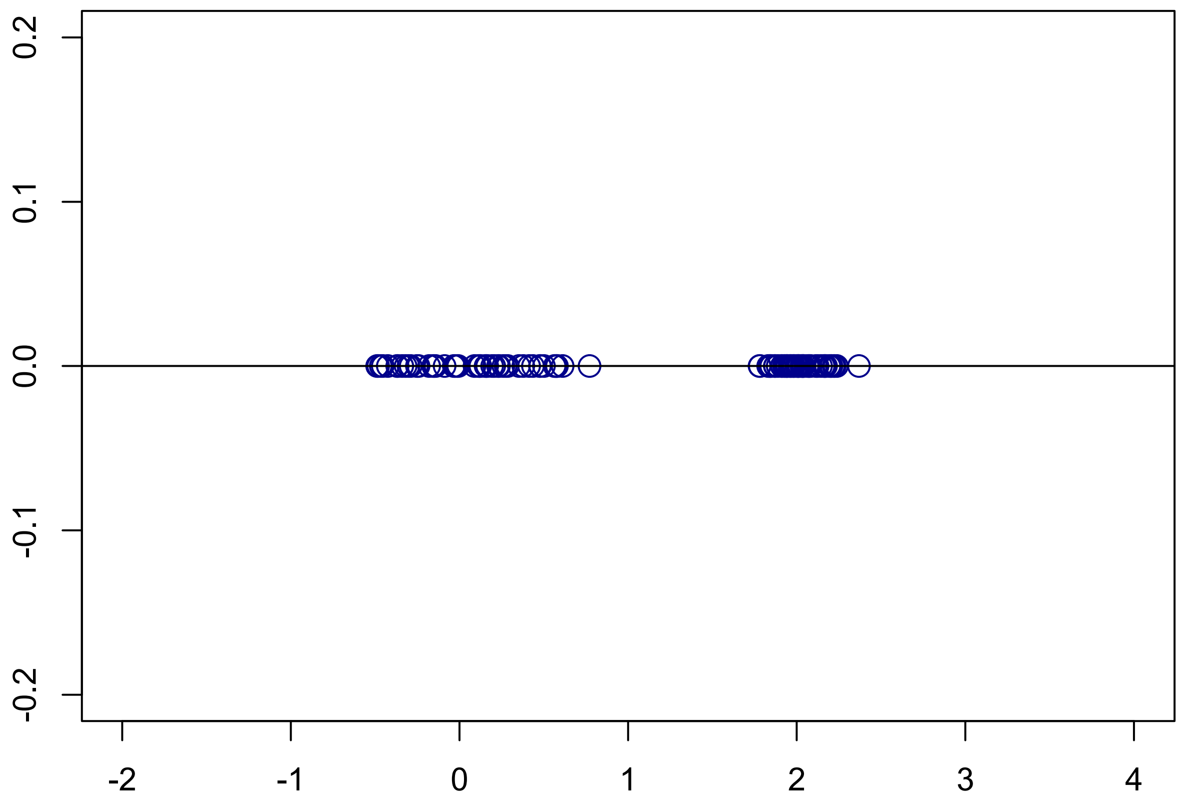
https://twitter.com/athena_schools/status/1063013435779223553/photo/1

Supervised vs. unsupervised learning

Supervised learning: we have a training set consisting of a feature vector $\boldsymbol{x} = (x_1, \dots, x_m)$ and a response y for each sample. Goal is to learn the mapping $\mathrm{f}\colon \boldsymbol{x} \mapsto y$.

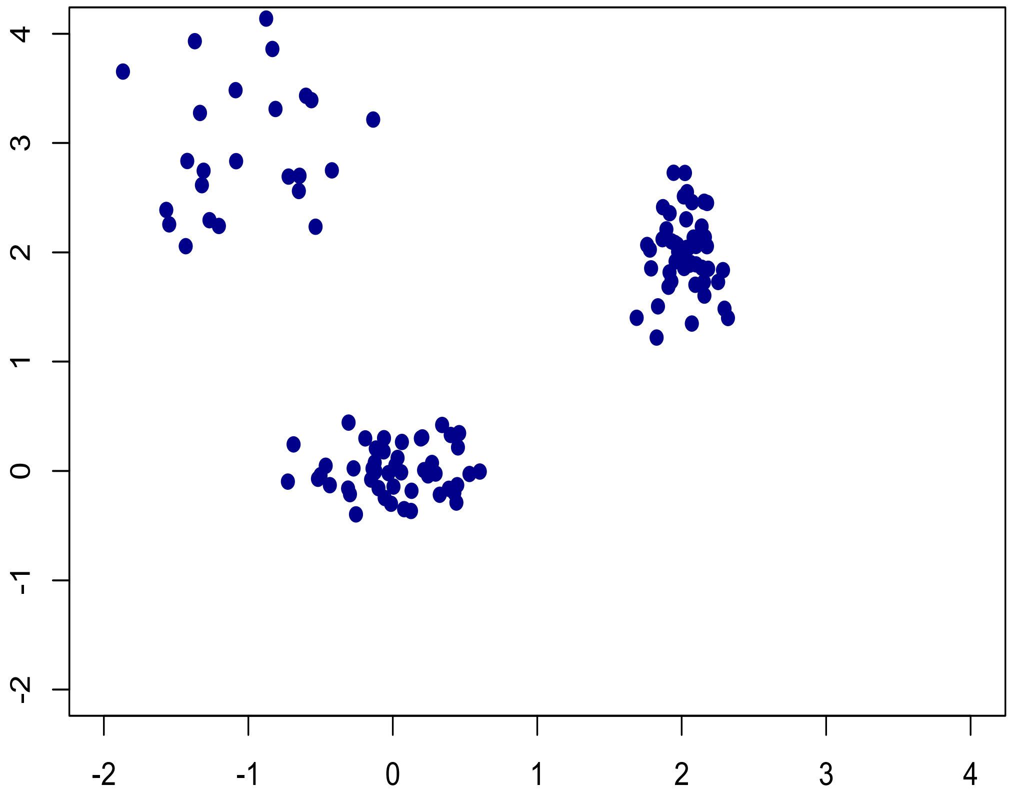
Unsupervised learning: we have a training set consisting of only a feature vector $\boldsymbol{x} = (x_1, \dots, x_m)$ for each sample. Goal is to learn something interesting about the distribution of the $\boldsymbol{x}$'s.

Example (1D)



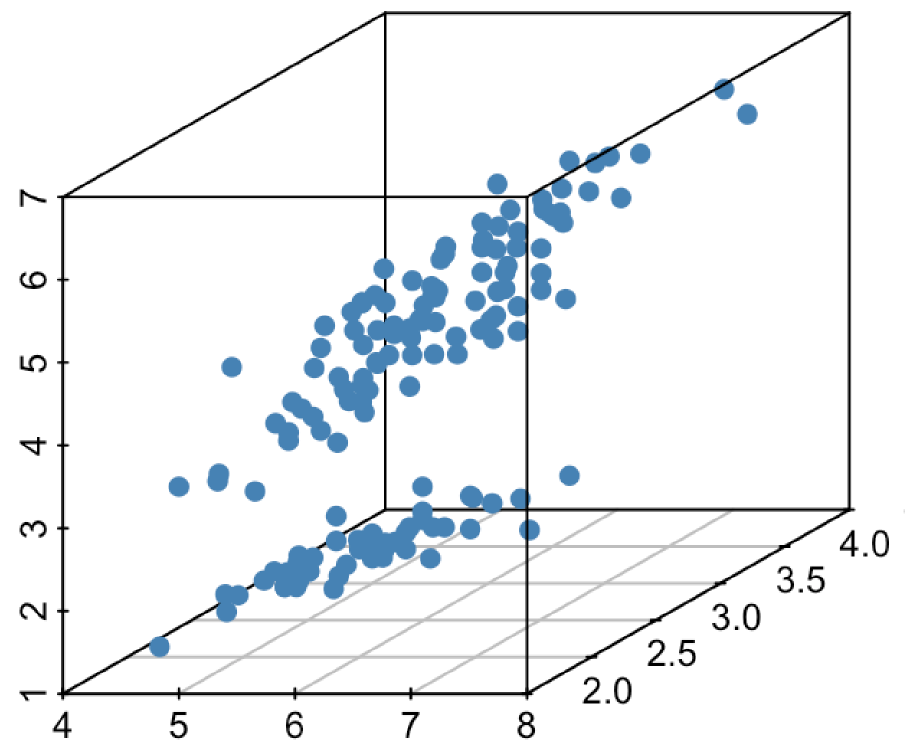
Each sample has a single feature x (a real number)

Example (2D)



Each sample has two features $\boldsymbol{x} = (x_1, x_2)$

Example (3D)

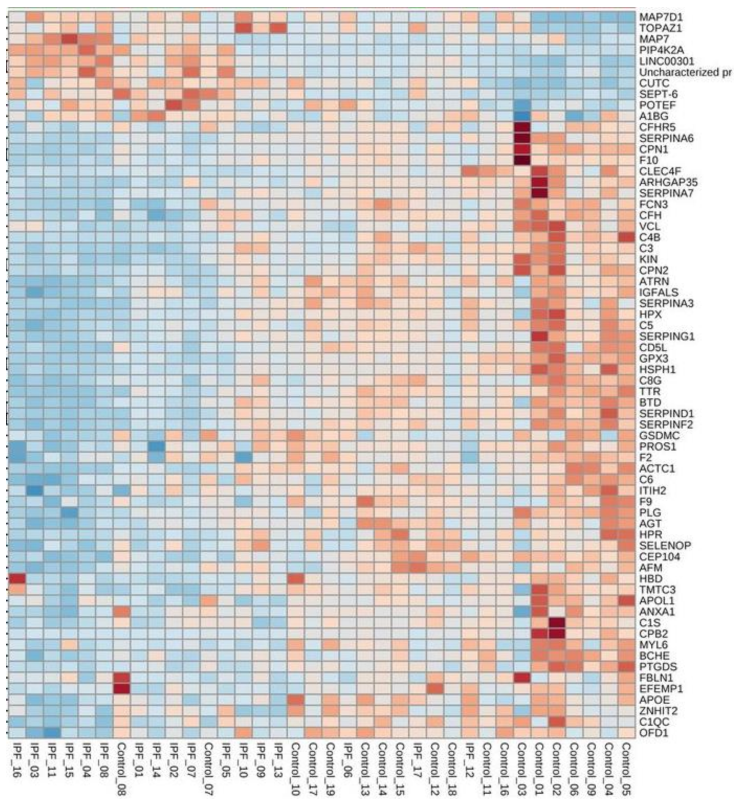


Each sample has three features $\boldsymbol{x} = (x_1, x_2, x_3)$

Example (p dimensions)

samples

features



Each sample has 66 features $\boldsymbol{x} = (x_1, x_2, x_{3,\ldots,}x_{66})$

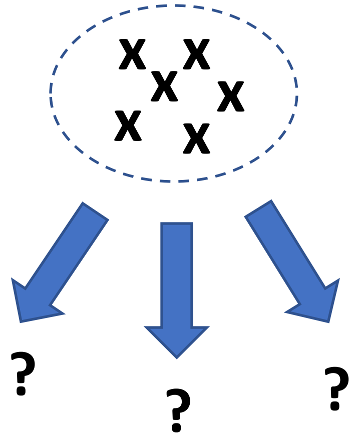
What can we learn in unsupervised learning?

- How to generate similar data
  → Density estimation, generative models

- How many (and which) groups are present
  → hierarchical clustering, k-means clustering

- How to compress the data
  → Autoencoders

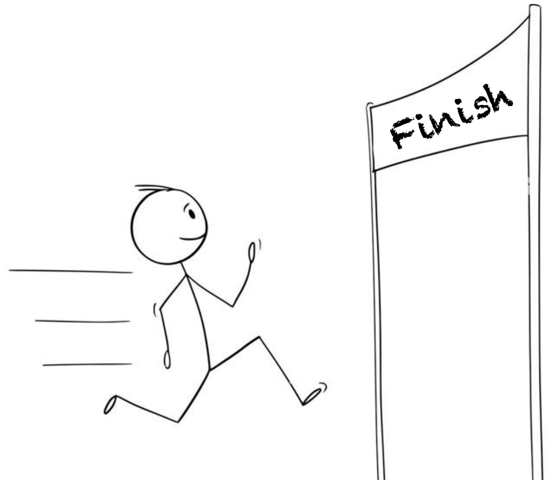
- How to visualize the data in low dimension
  → PCA, learning vector quantization, self organizing maps



Evaluating the performance

Supervised learning:

- Goodness of fit to training data
- Goodness of fit to new data
- Prediction performance



Unsupervised learning:

- No "truth" to compare with
- Performance more open to interpretation
- Still: often possible to define a useful loss function and to determine which of two solutions is best

IN3050/IN4050, Lecture 10
Unsupervised Learning

3: Learning how data are grouped

Clustering

Learning how the data are grouped is called clustering.

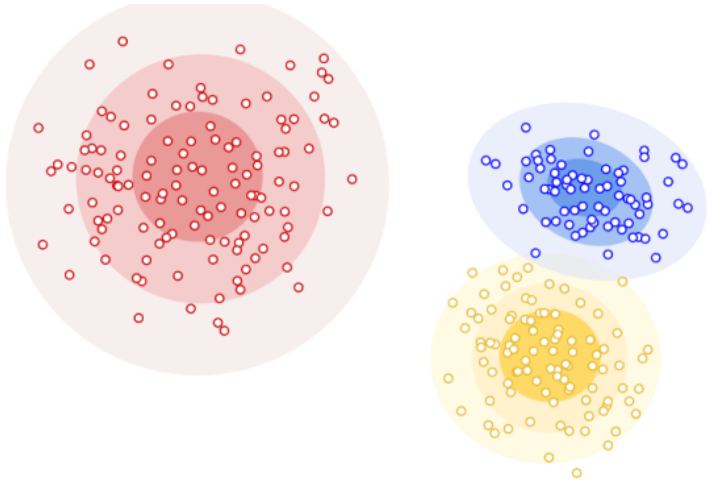
Several main types of clustering methods:

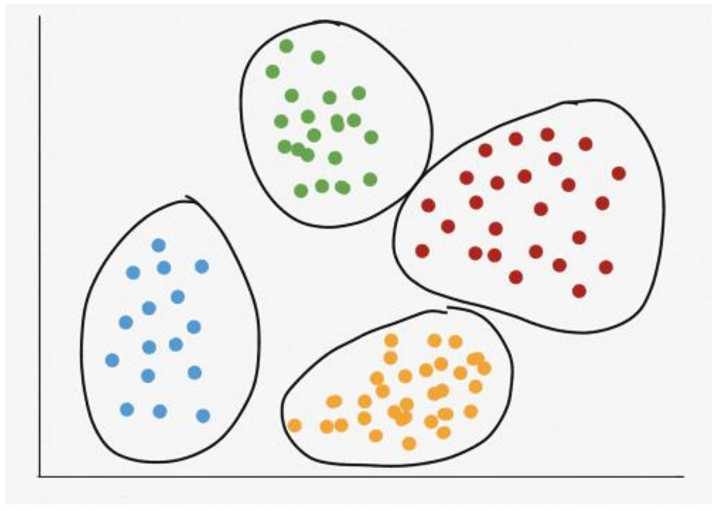
Model-based Clustering

- Assume data are generated by a mixture of underlying probabilistic models.
- E.g., **Gaussian Mixture Models (GMMs)**

Density-based Clustering

- Groups are defined by regions of high data density separated by low-density regions.
- E.g., **DBSCAN**



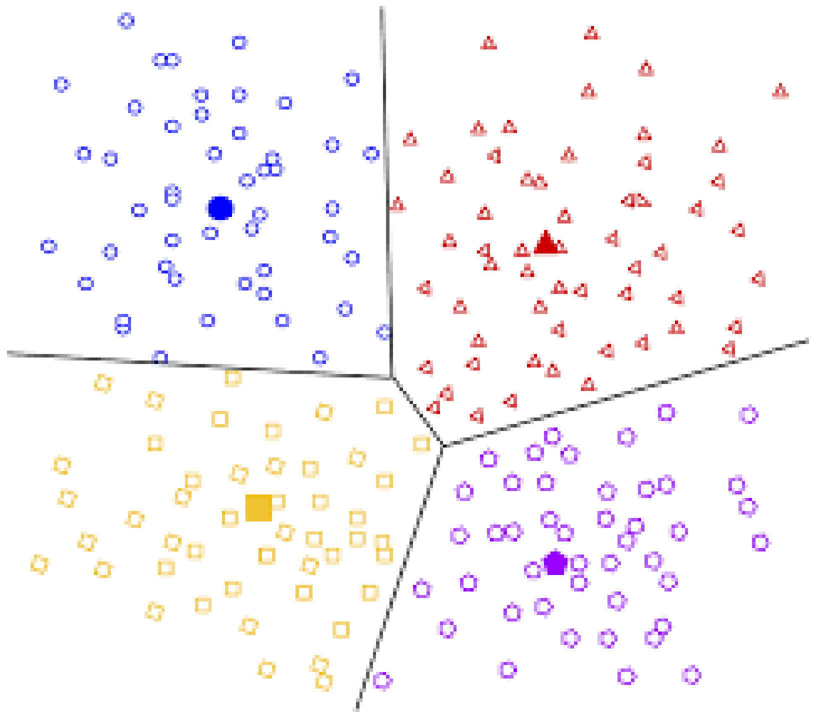


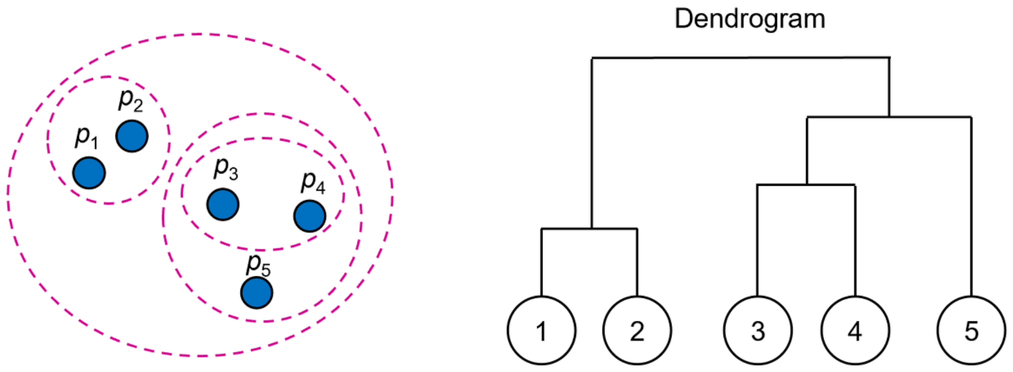
Partition-based Clustering

- Divide the data into a fixed number of clusters (*k*) such that each point belongs to exactly one cluster.
- Centroid-based clustering
  - E.g., **k-means clustering**

Hierarchical clustering

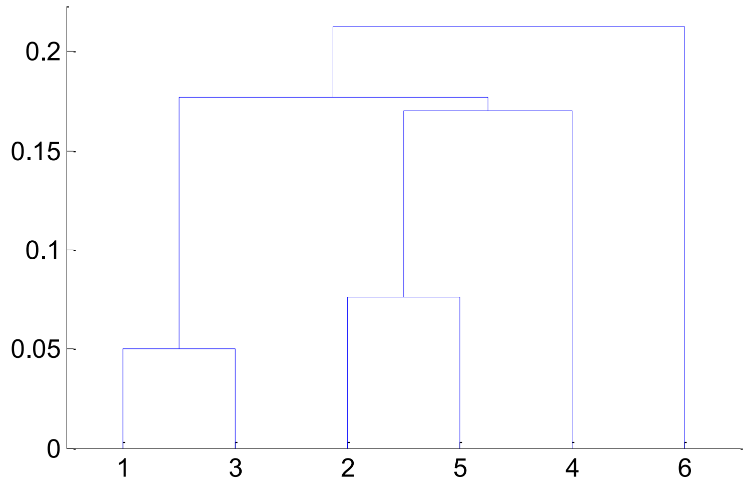
- Build a hierarchy (tree) of clusters.
- E.g., **Agglomerative**
- E.g., **Divisive**

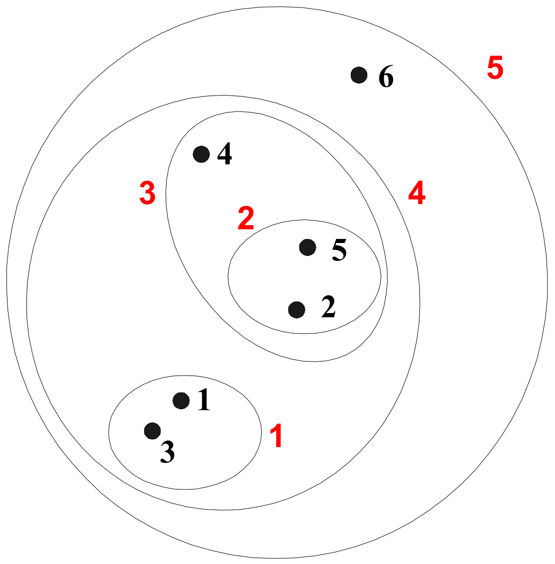




Hierarchical Clustering

- Produces a set of ***nested clusters*** organized as a hierarchical tree
- Can be visualized as a **dendrogram**
  - A tree-like diagram that records the sequences of merges or splits





Hierarchical Clustering

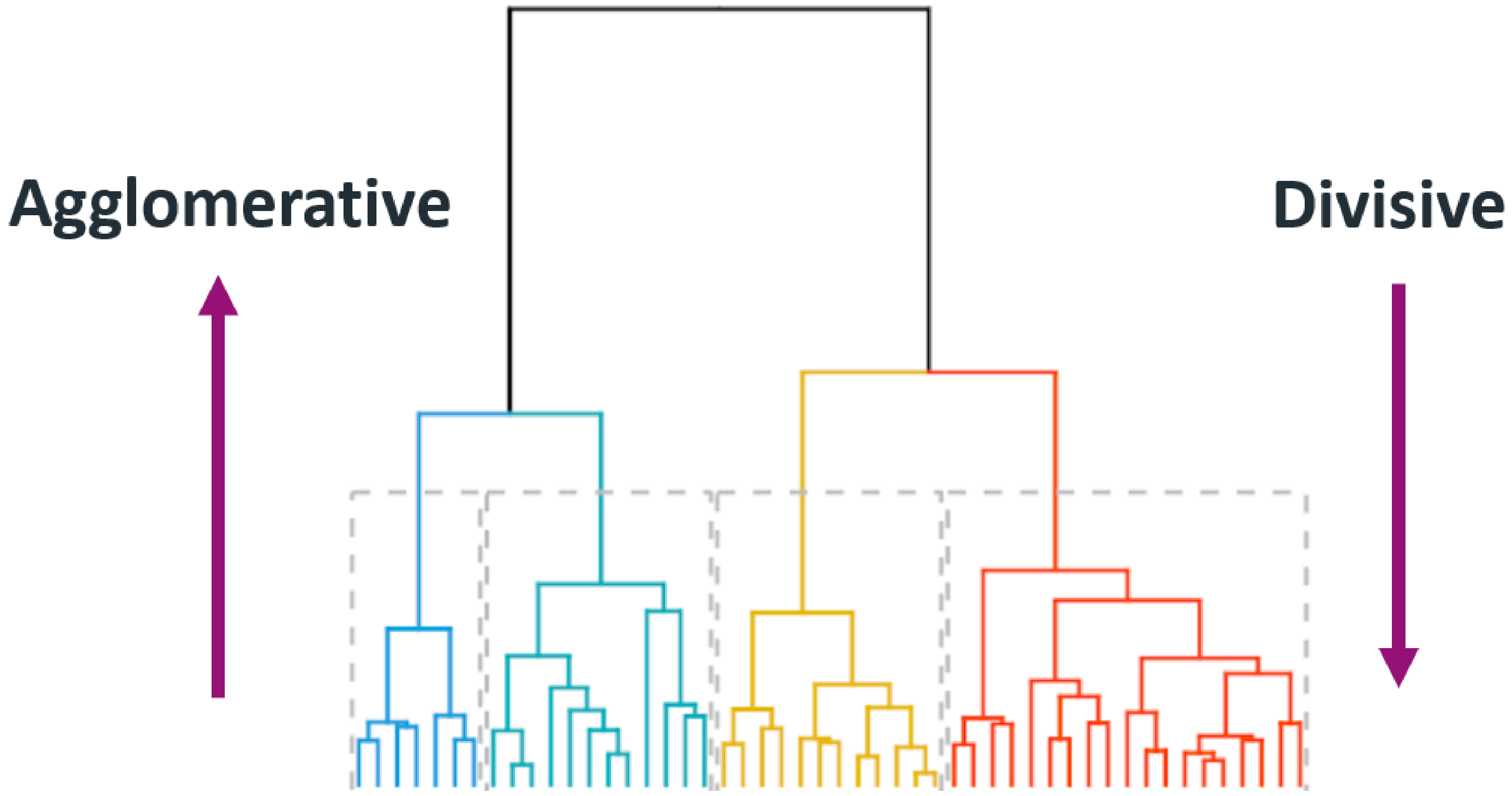
- Two main types of hierarchical clustering:

  - **Agglomerative:**
    1. Initially, each sample is a separate cluster
    2. Find the two clusters that are closest together and merge them into one cluster
    3. Repeat point 2 until there is only one cluster left

  - **Divisive:**
    1. Initially, all samples are in a single cluster.
    2. Select a cluster to split (for example, the one with the largest dissimilarity among its members) and divide it into two smaller clusters.
    3. Repeat point 2 until each sample is its own cluster.

- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

Hierarchical clustering



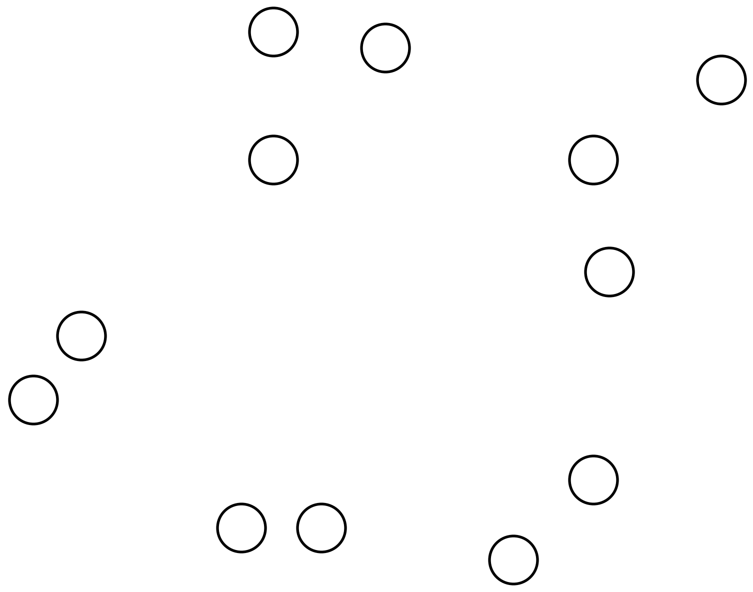
https://harshsharma1091996.medium.com/hierarchical-clustering-996745fe656b

Hierarchical clustering: important aspects

- Must keep track of the whole history of cluster mergers in order to plot the tree (= dendrogram)
- In each iteration we have to compute the distance between every pair of the remaining clusters
- We have to define what we mean by the distance between two clusters

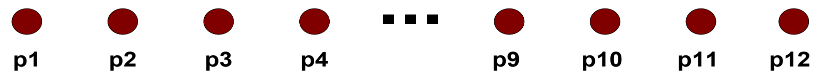
Input/ Initial setting

- Start with clusters of individual points and a distance/proximity matrix



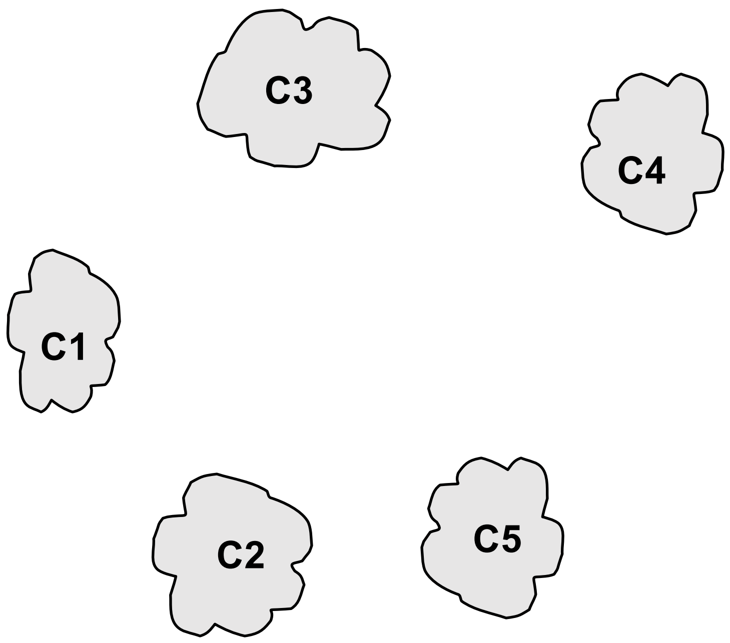
	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
. . .						

Distance/Proximity Matrix



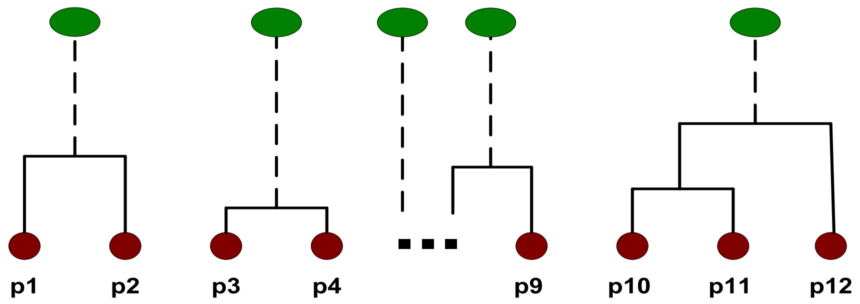
Intermediate State

- After some merging steps, we have some clusters



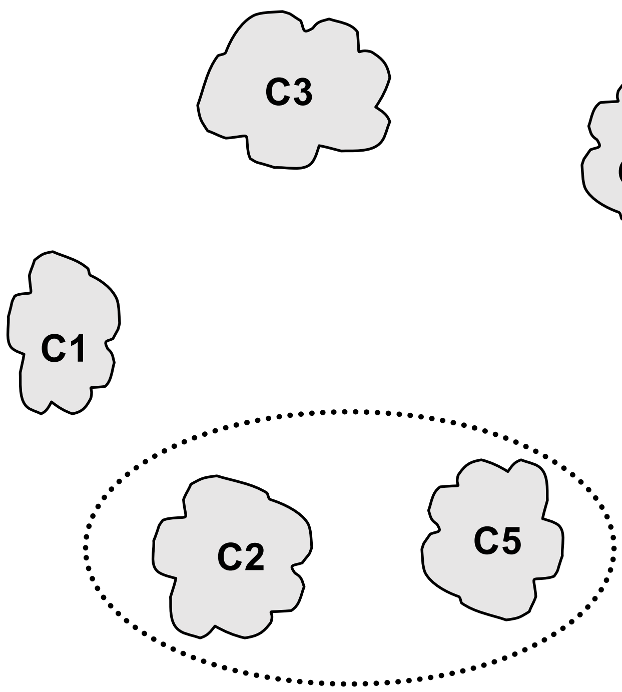
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

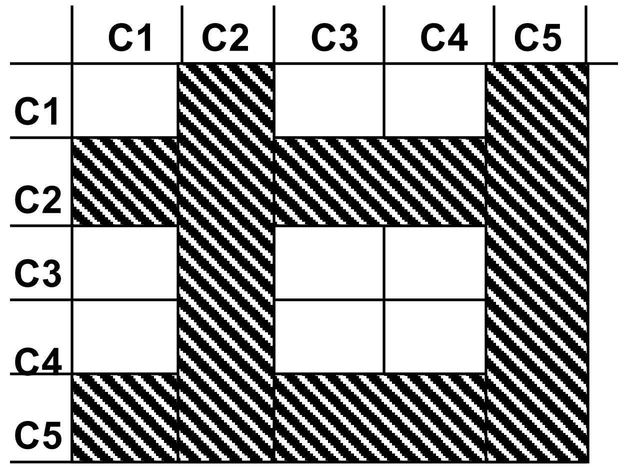
Distance/Proximity Matrix



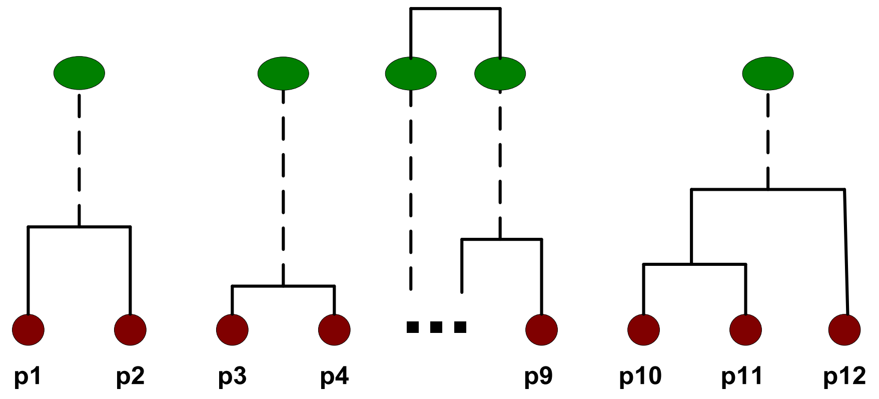
Intermediate State

- Merge the two closest clusters (C2 and C5) and update the distance matrix.



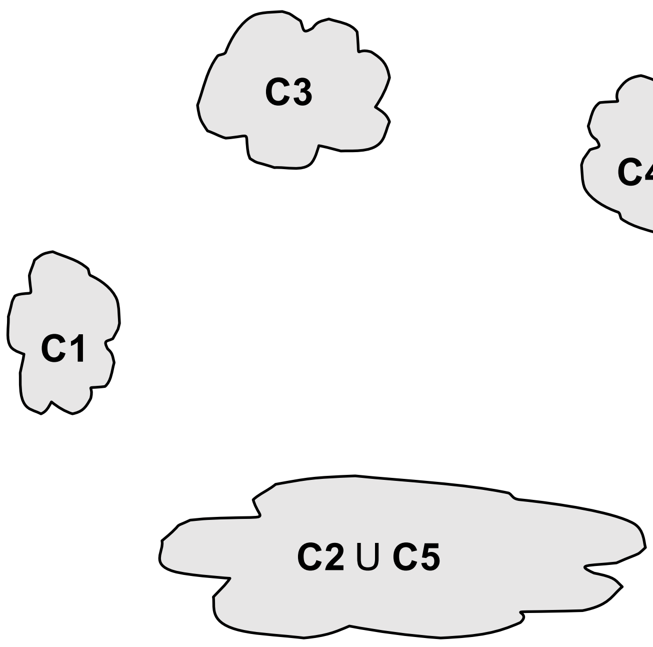


Distance/Proximity Matrix

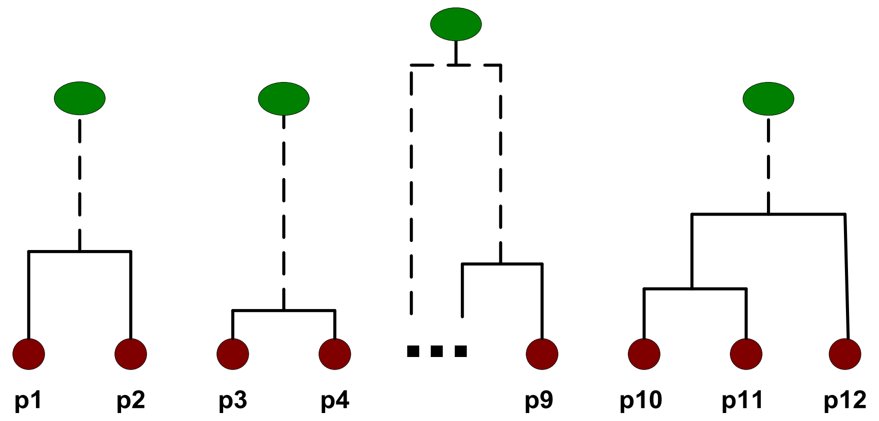


After Merging

- "How do we update the distance matrix?"

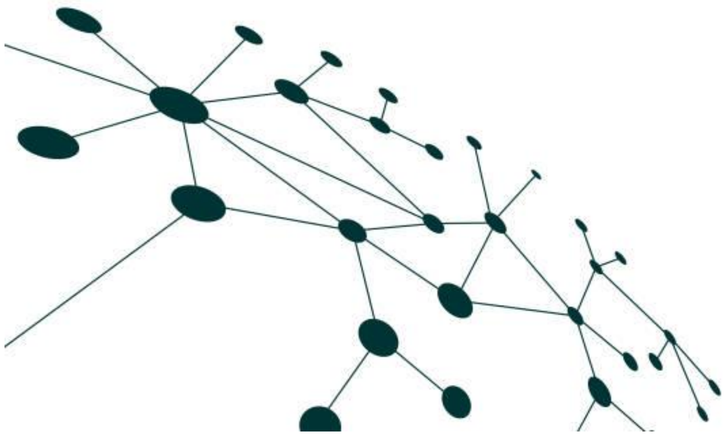


	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?	?	?	?
C3		?		
C4		?		



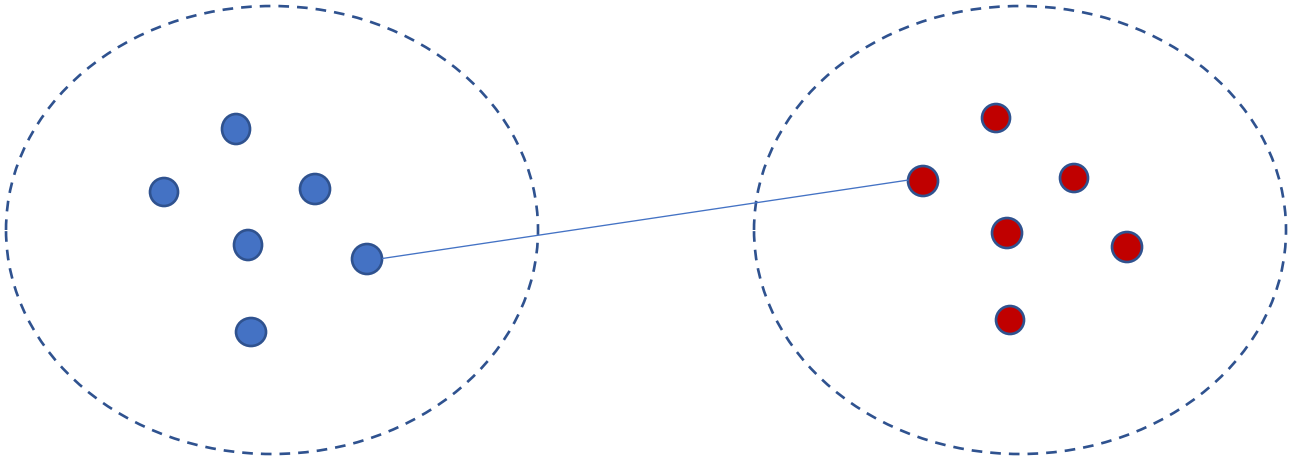
Hierarchical clustering: how to define the distance between two clusters C_1 and C_2

- Referred to as linkage method
- Has great influence on result
- Most common linkage methods:
    - Single linkage
    - Average linkage
    - Complete linkage
    - Ward linkage



Single linkage

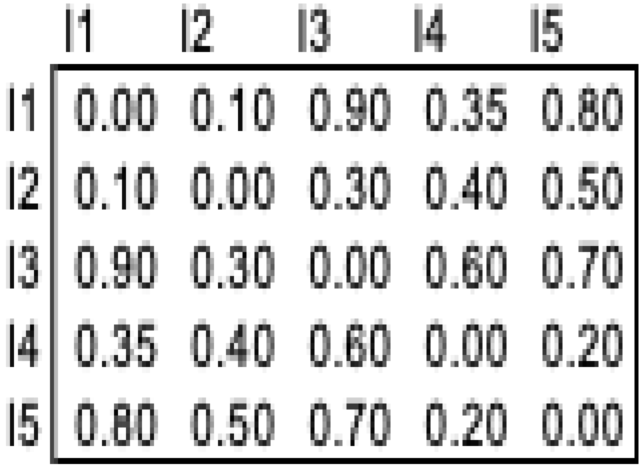
$d(C_1,C_2)$ = smallest distance between a sample in C_1 and a sample in C_2



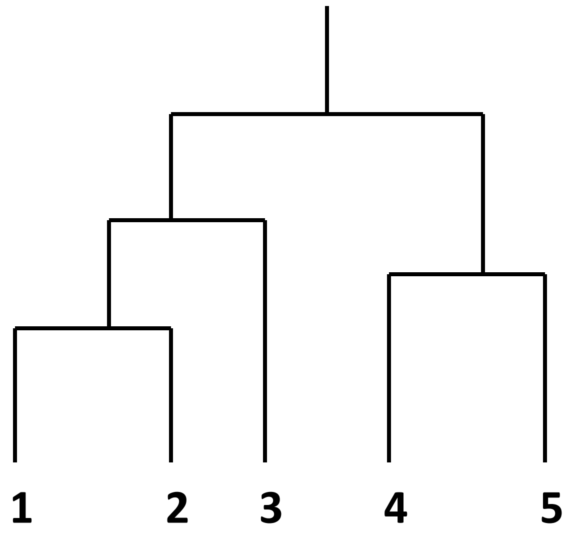
$$D_{sl}\left(C_i, C_j\right) = \min_{x,y}\left\{d(x,y) \middle| x \in C_i, y \in C_j\right\}$$

Single-link clustering: example

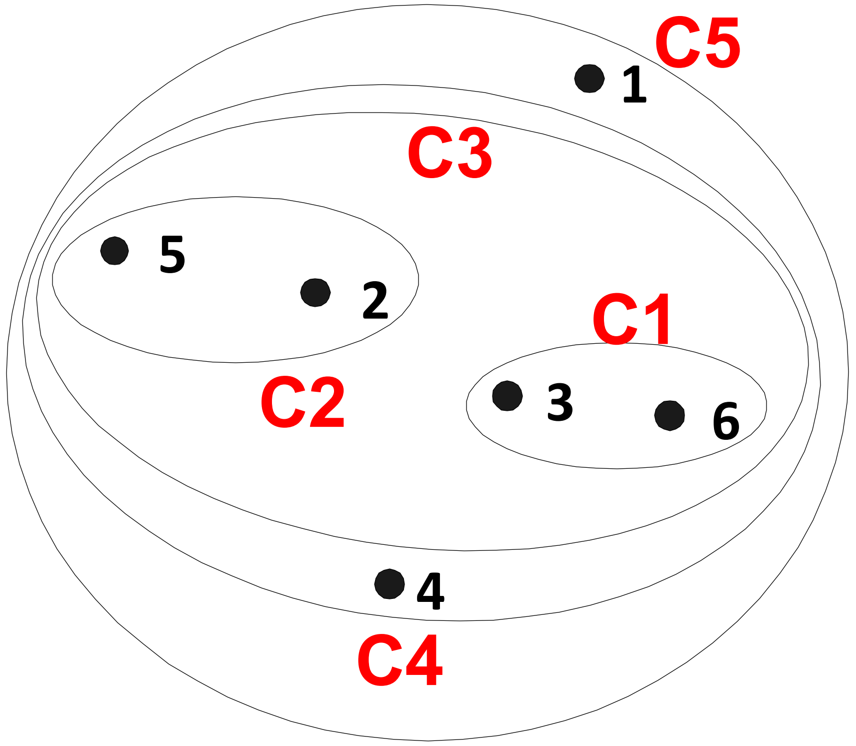
- Determined by one pair of points, i.e., by one link in the proximity graph.

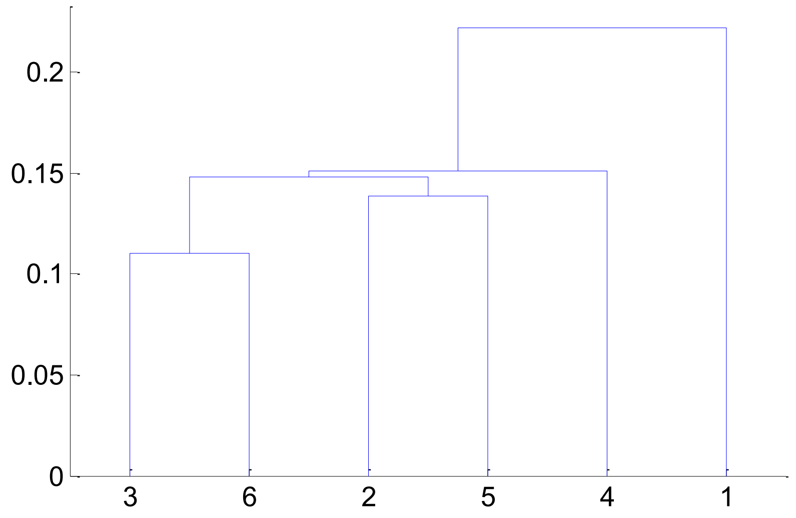


	I1	I2	I3	I4	I5
I1	0.00	0.10	0.90	0.35	0.80
I2	0.10	0.00	0.30	0.40	0.50
I3	0.90	0.30	0.00	0.60	0.70
I4	0.35	0.40	0.60	0.00	0.20
I5	0.80	0.50	0.70	0.20	0.00



Single-link clustering: example

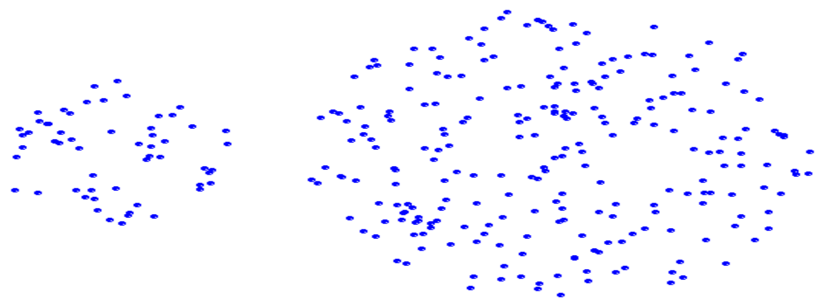


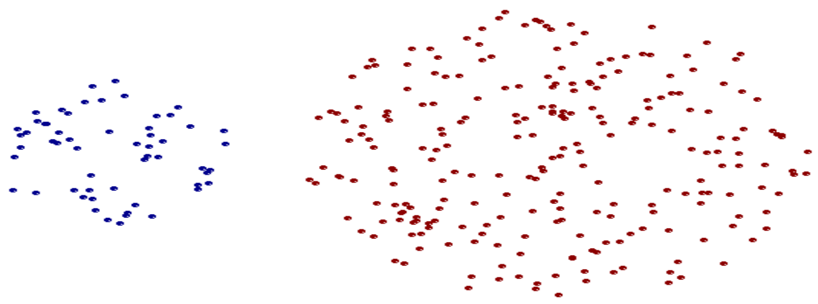


Nested Clusters

Dendrogram

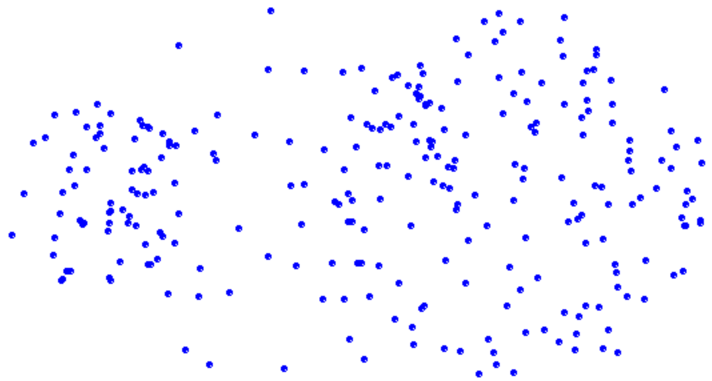
Strengths of single-link clustering

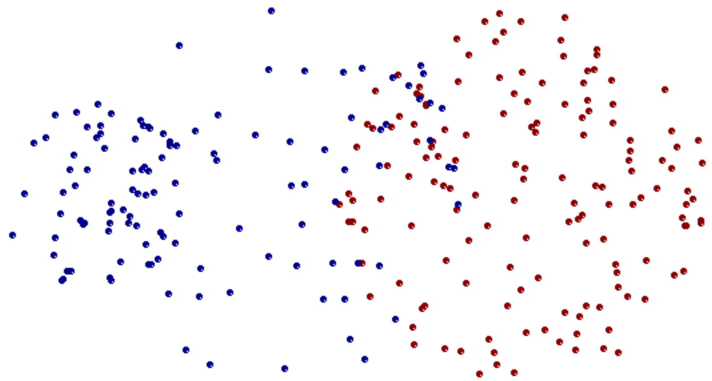

Original Points


Two Clusters

- Can handle non-elliptical shapes

Limitations of single-link clustering

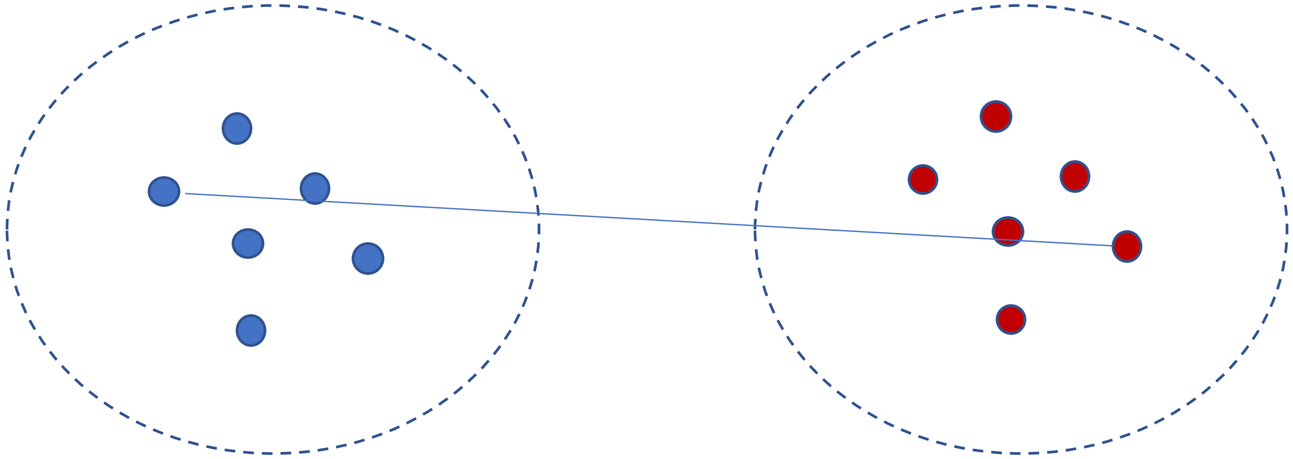

Original Points


Two Clusters

- Sensitive to noise and outliers
- It produces long, elongated clusters

Complete linkage

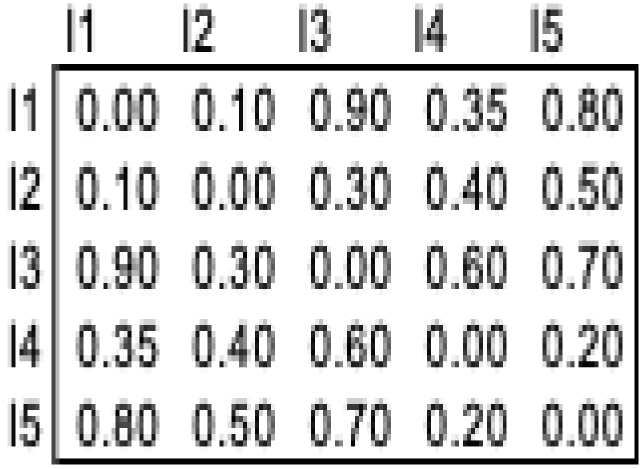
d(C_1,C_2) = maximal distance between a sample in C_1 and a sample in C_2



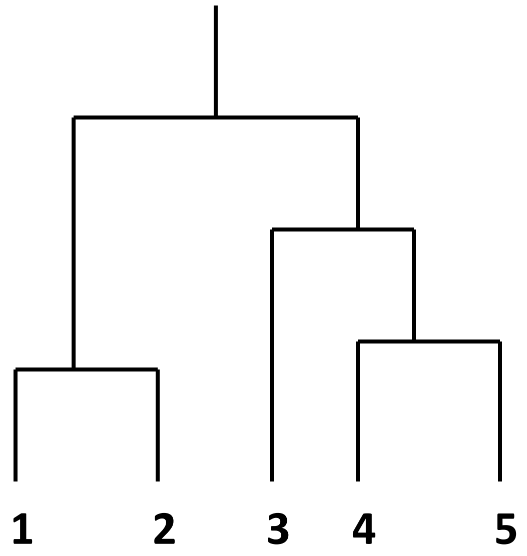
$$D_{cl}\left(C_i, C_j\right) = \max_{x,y}\left\{d(x,y) \,\middle|\, x \in C_i, y \in C_j\right\}$$

Complete-link clustering: example

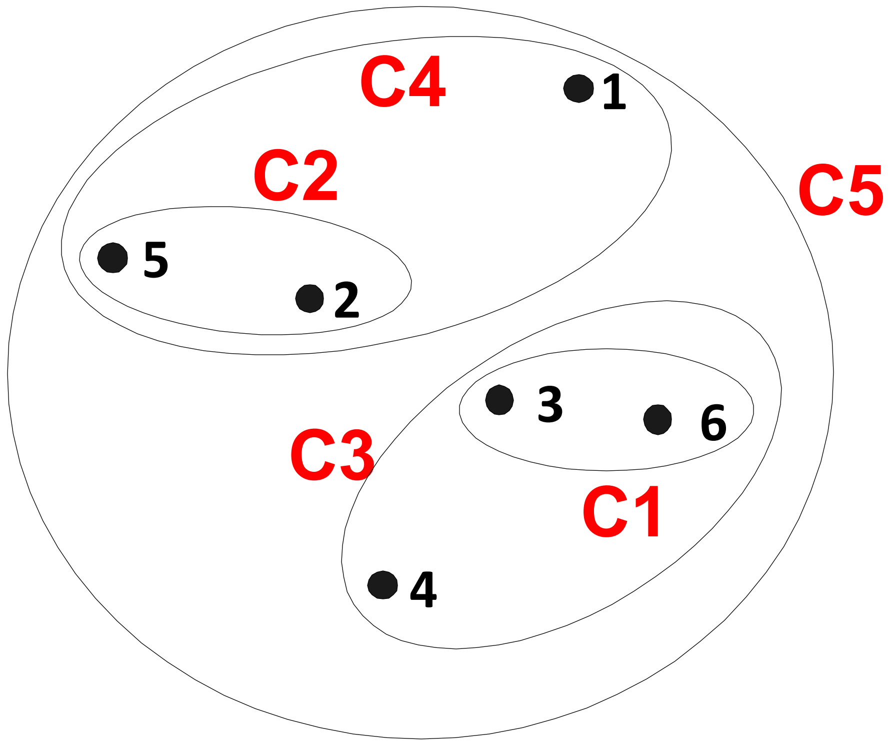
- Distance between clusters is determined by the two most distant points in the different clusters



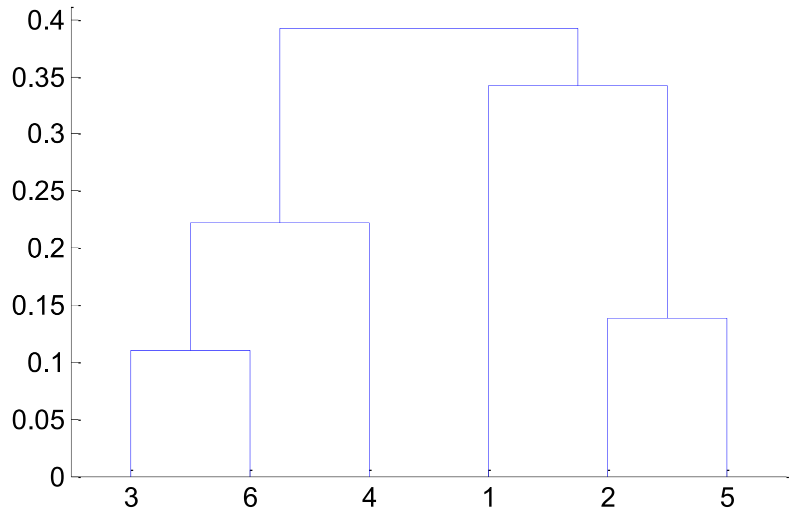
	I1	I2	I3	I4	I5
I1	0.00	0.10	0.90	0.35	0.80
I2	0.10	0.00	0.30	0.40	0.50
I3	0.90	0.30	0.00	0.60	0.70
I4	0.35	0.40	0.60	0.00	0.20
I5	0.80	0.50	0.70	0.20	0.00



Complete-link clustering: example

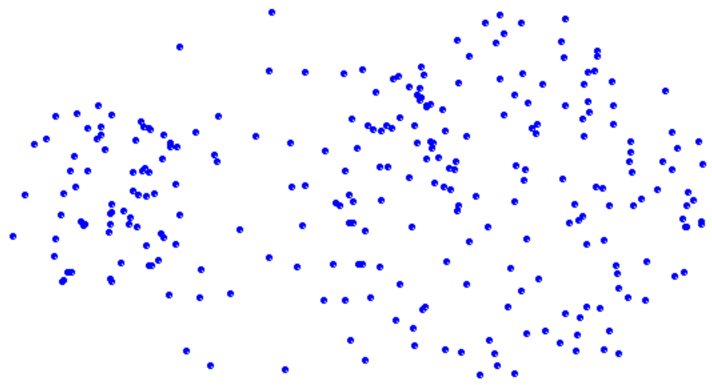


Nested Clusters

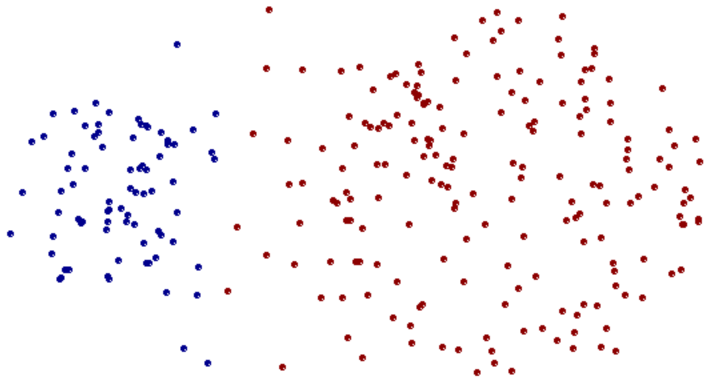


Dendrogram

Strengths of complete-link clustering



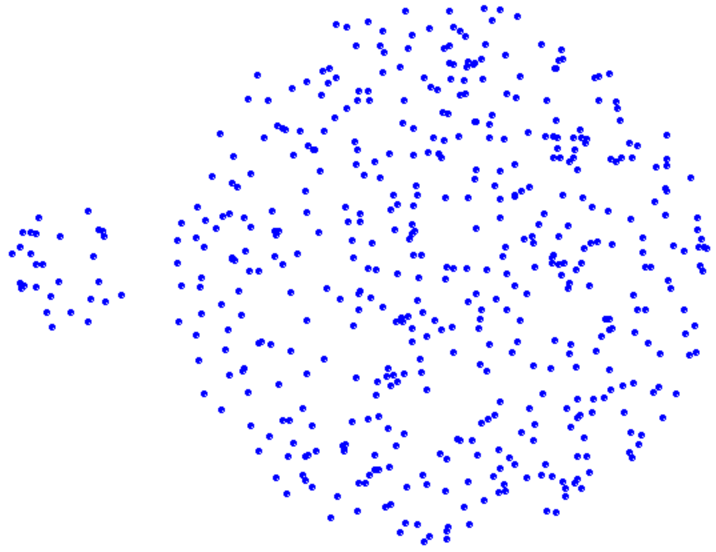
Original Points

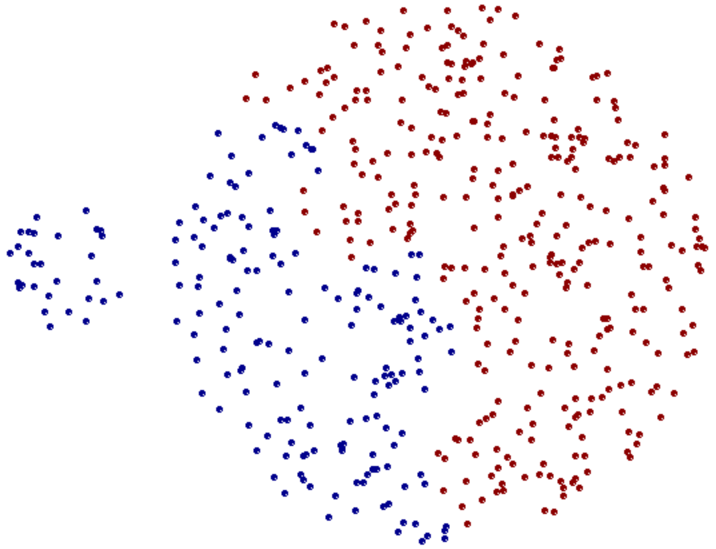


Two Clusters

- More balanced clusters (with equal diameter)
- Less susceptible to noise

Limitations of complete-link clustering

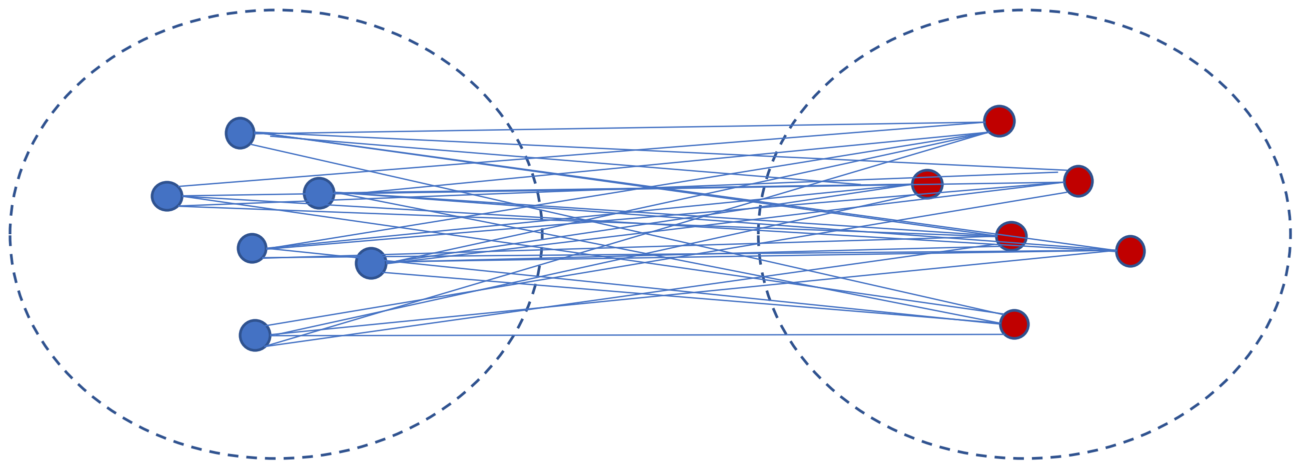

Original Points


Two Clusters

- Tends to break large clusters
- All clusters tend to have the same diameter – small clusters are merged with larger ones

Average linkage

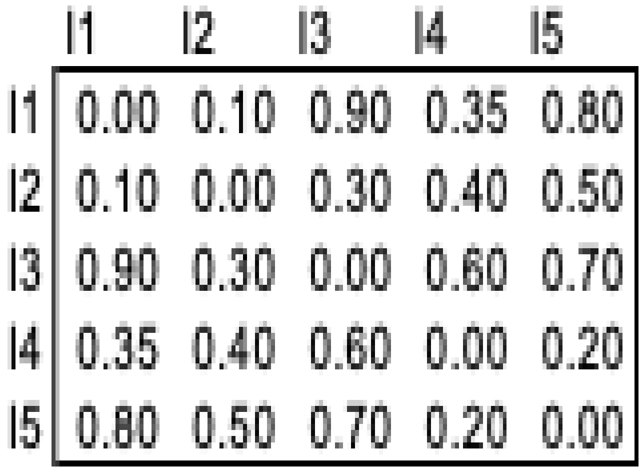
$d(C_1, C_2)$ = average distance between samples in C_1 and samples in C_2



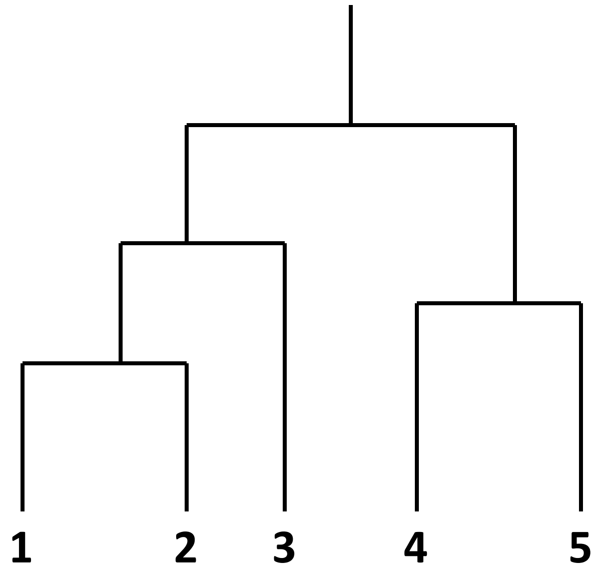
$$D_{avg}(C_i, C_j) = \frac{1}{|C_i| \times |C_j|} \sum_{x \in C_i, y \in C_j} d(x, y)$$

Average-link clustering: example

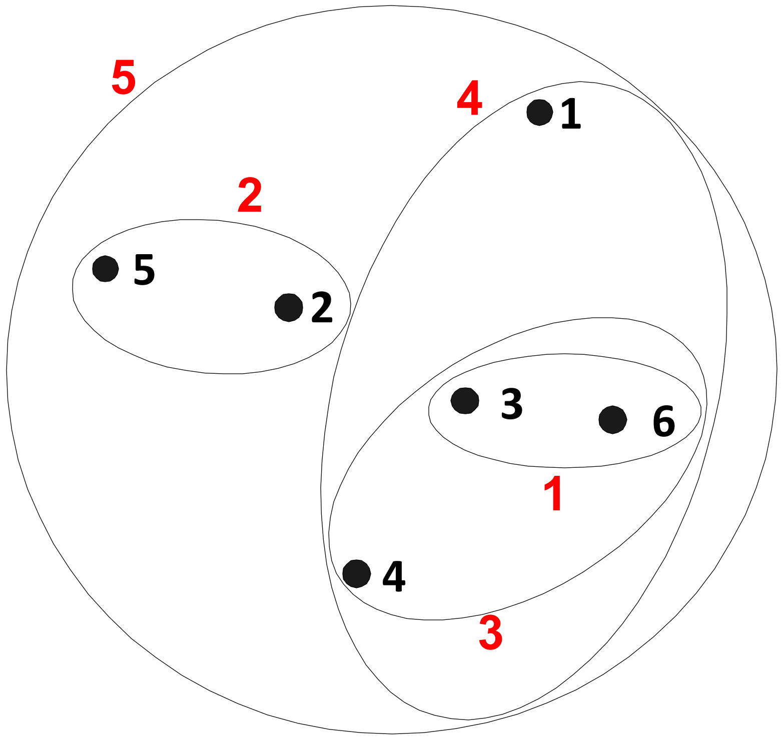
- Proximity of two clusters is the average of pairwise proximity between points in the two clusters.



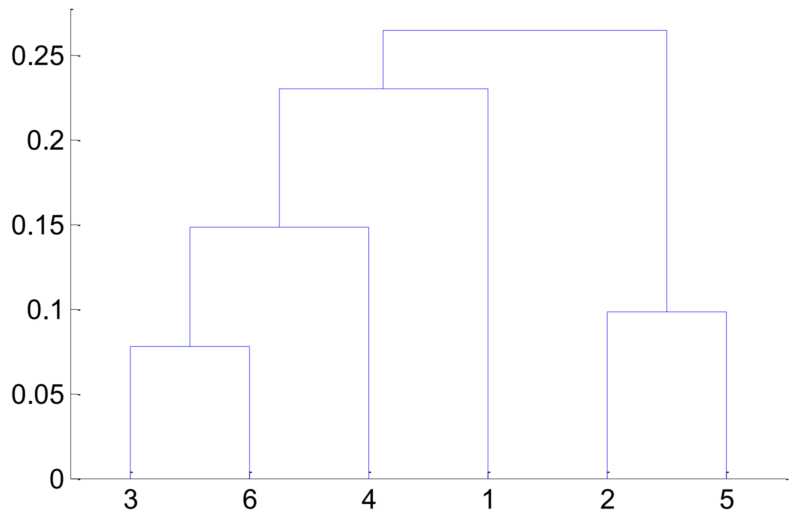
	I1	I2	I3	I4	I5
I1	0.00	0.10	0.90	0.35	0.80
I2	0.10	0.00	0.30	0.40	0.50
I3	0.90	0.30	0.00	0.60	0.70
I4	0.35	0.40	0.60	0.00	0.20
I5	0.80	0.50	0.70	0.20	0.00



Average-link clustering: example



Nested Clusters



Dendrogram

Average-link clustering: discussion

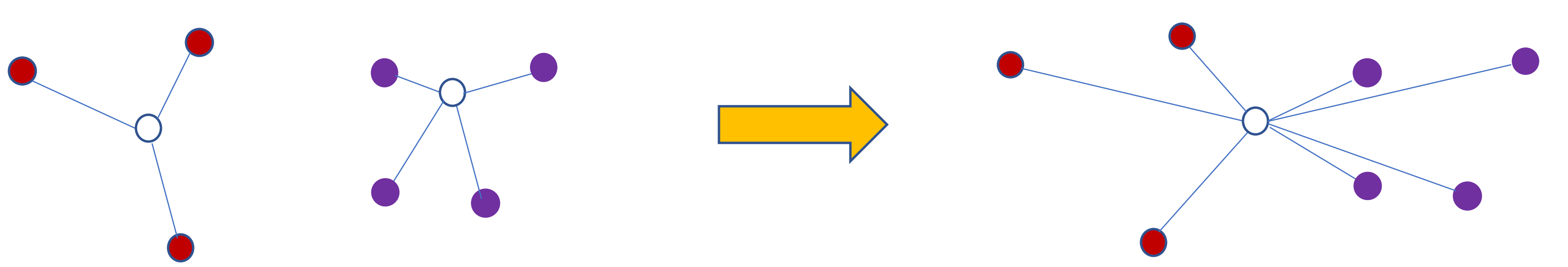
- Compromise between Single and Complete Link

- Strengths
  - Less susceptible to noise and outliers

- Limitations
  - Biased towards globular clusters

Ward linkage

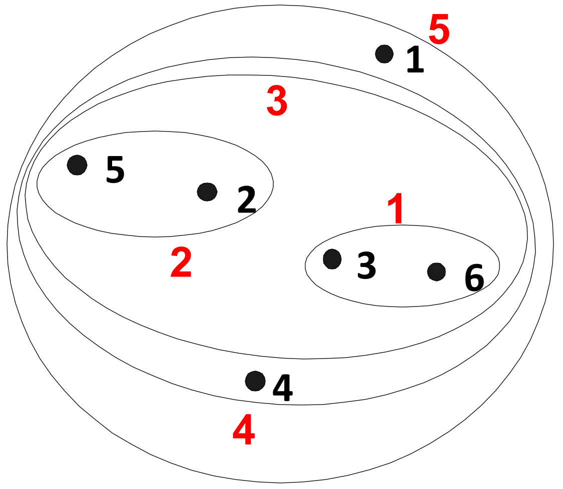
$d(C_1,C_2)$ = increase in within-cluster variance if clusters are merged



Ward’s distance for clusters

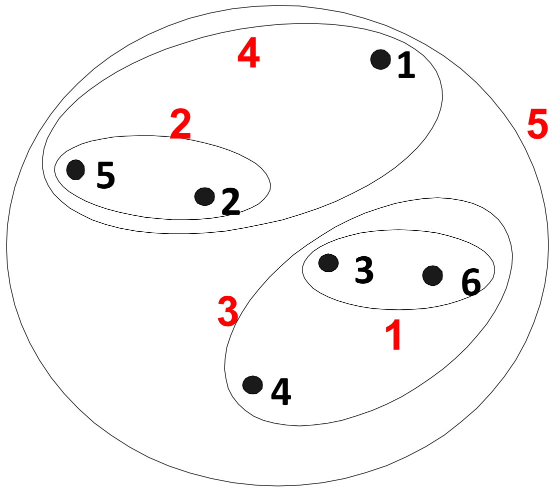
- Similar to group average and centroid distance
- Less susceptible to noise and outliers
- Biased towards globular clusters
- Hierarchical analogue of k-means
    - Can be used to initialize k-means

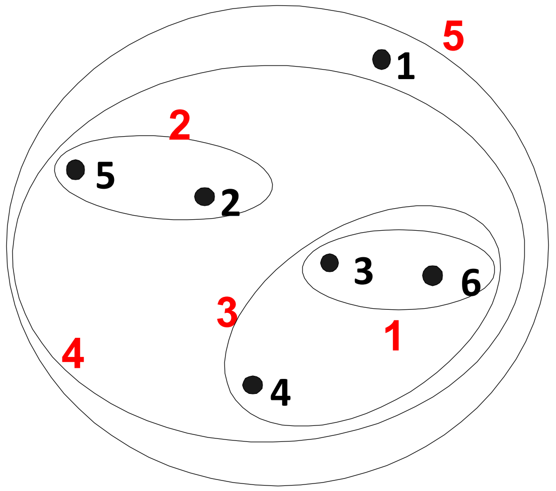
Hierarchical Clustering: Comparison



MIN

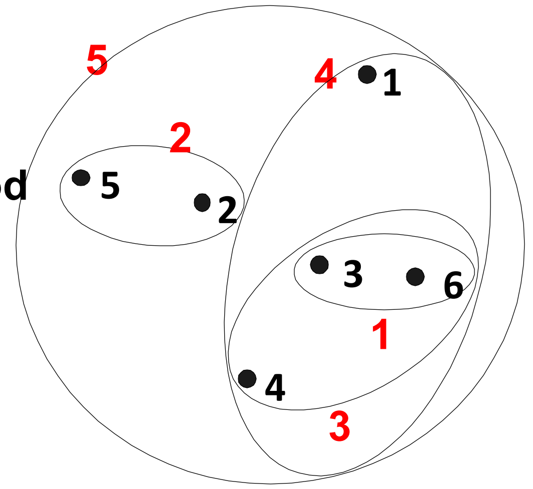
MAX





Group Average

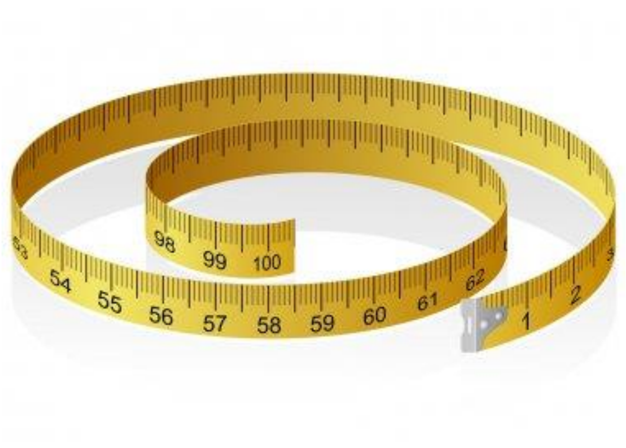
Ward's Method



Hierarchical clustering: how to define the distance between two samples $\mathbf{x}_1$ and $\mathbf{x}_2$

Manhattan distance:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=1}^{p} | \, x_{1i} - x_{2i} \, |$$



Euclidean distance:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=1}^{p} (x_{1i} - x_{2i})^2$$

Cosine distance:

$$d(\mathbf{x}_1, \mathbf{x}_2) = 1 - \cos\theta = 1 - \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{||\mathbf{x}_1|| \cdot ||\mathbf{x}_1||}$$

Strengths of Hierarchical Clustering

- No assumptions on the number of clusters
    - Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level

- Hierarchical clustering may correspond to meaningful taxonomies
    - Example in biological sciences (e.g., phylogeny reconstruction), web (e.g., product catalogs) etc.

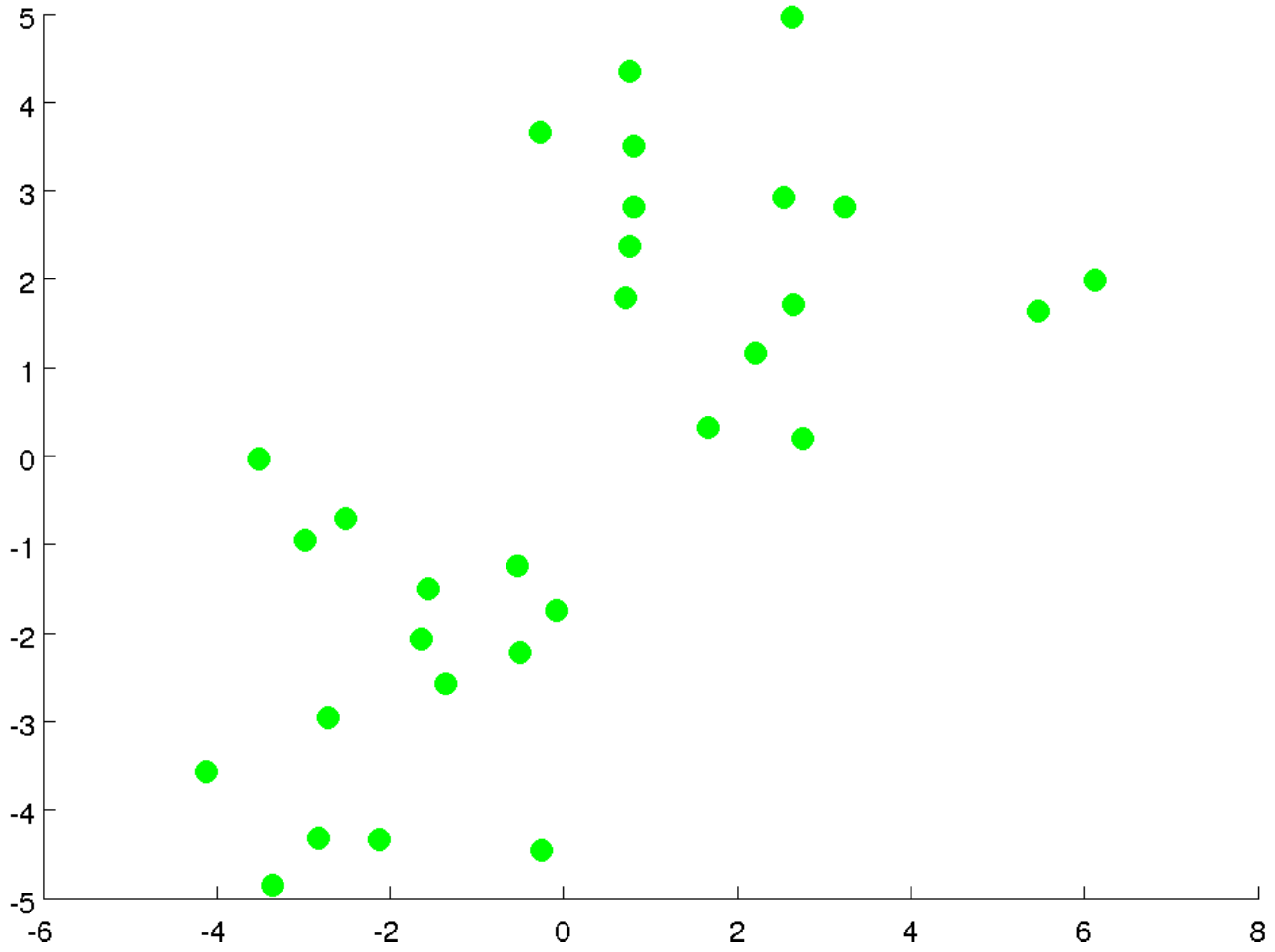
Complexity of hierarchical clustering

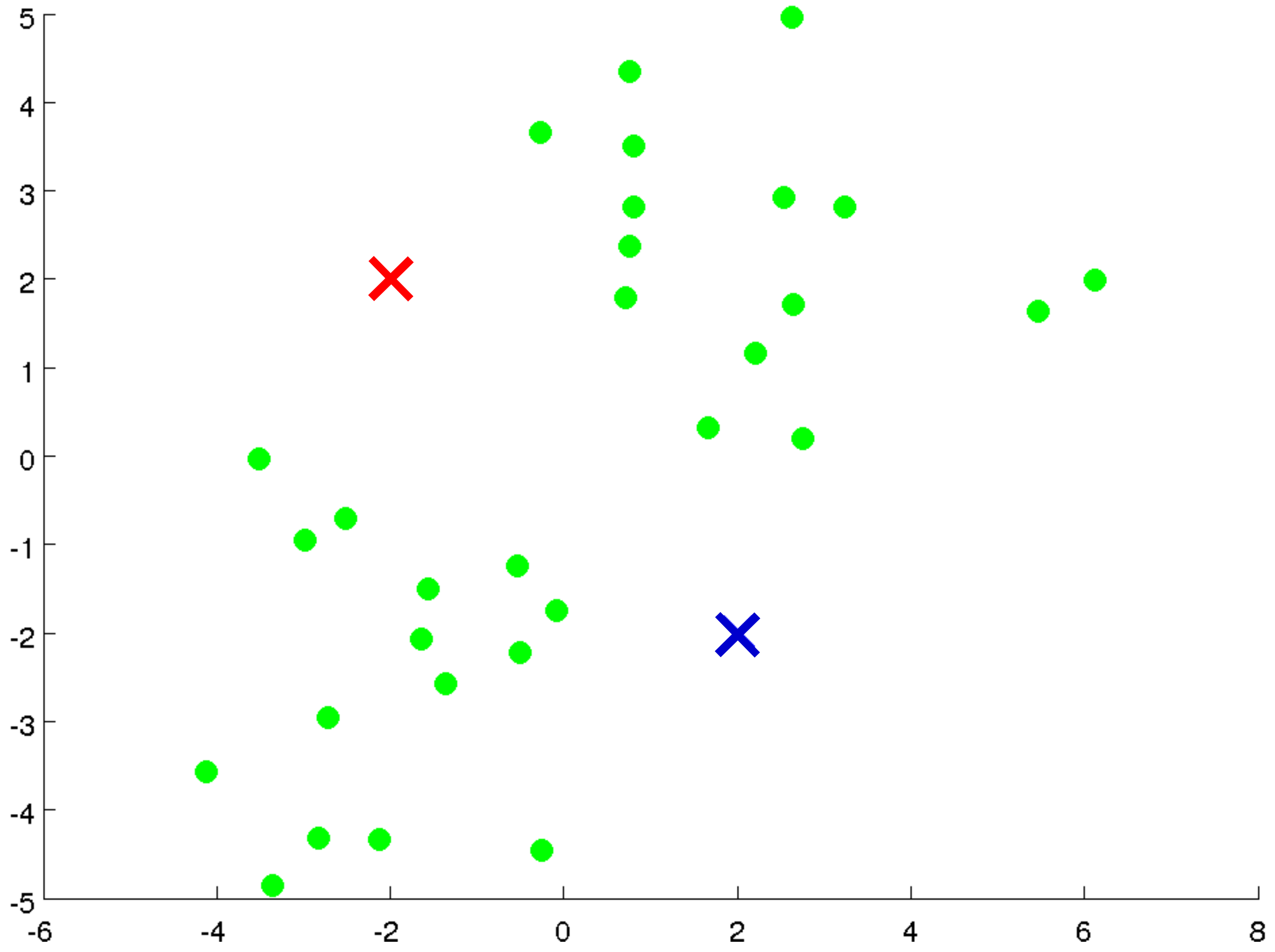
- The distance matrix is used for deciding which clusters to merge/split
- At least quadratic in the number of data points
  - For a dataset X consisting of n points
  - $O(n^3)$ time in most cases
  - There are n steps, and at each step, the size n^2 distance matrix must be updated and searched
  - Complexity can be reduced to $O(n^2 \log(n))$ time for some approaches by using appropriate data structures
- Not usable for large datasets

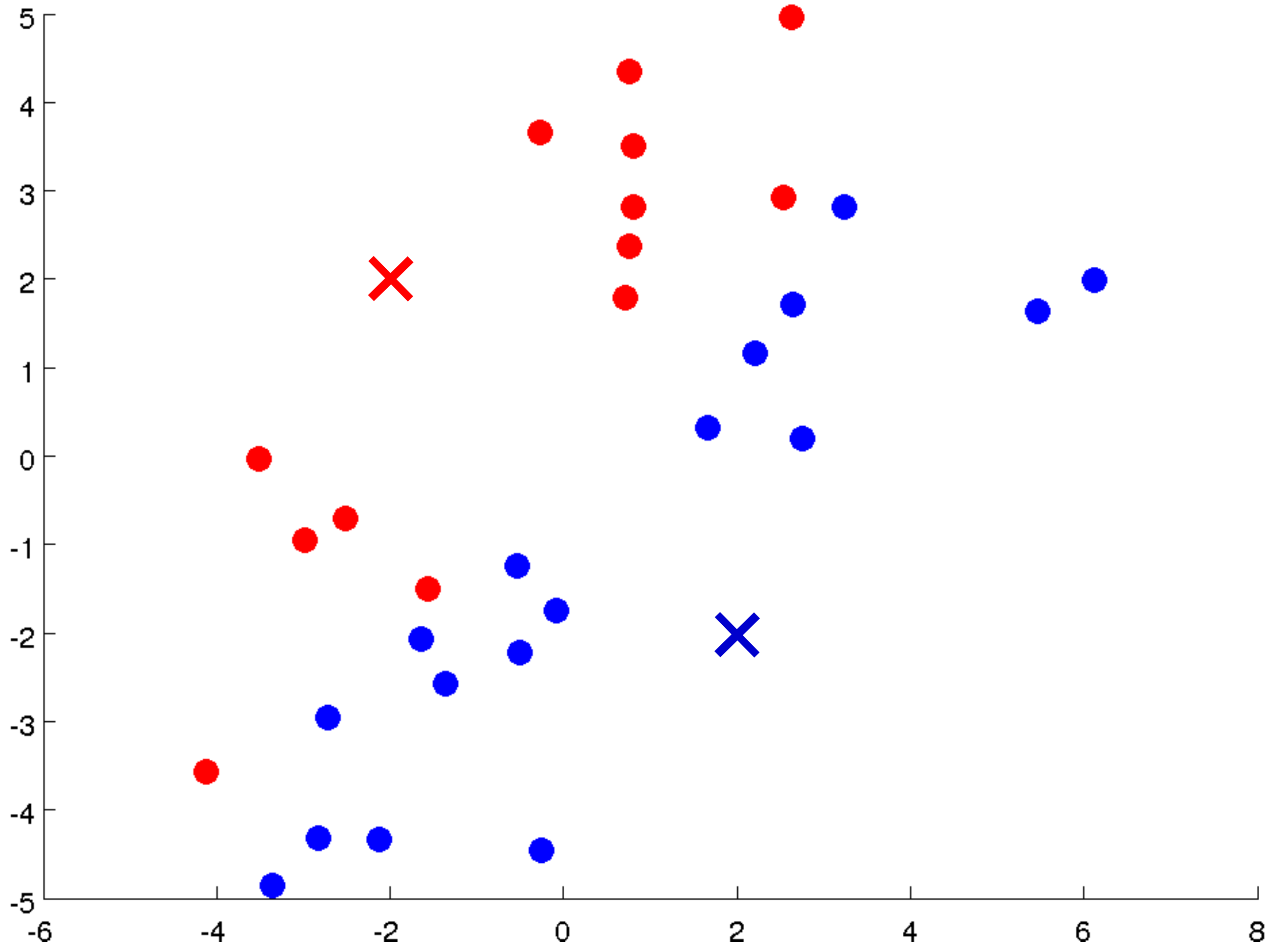
k-means clustering

As before we wish to cluster samples $x_1, \ldots, x_n \in \mathbf{R}^p$

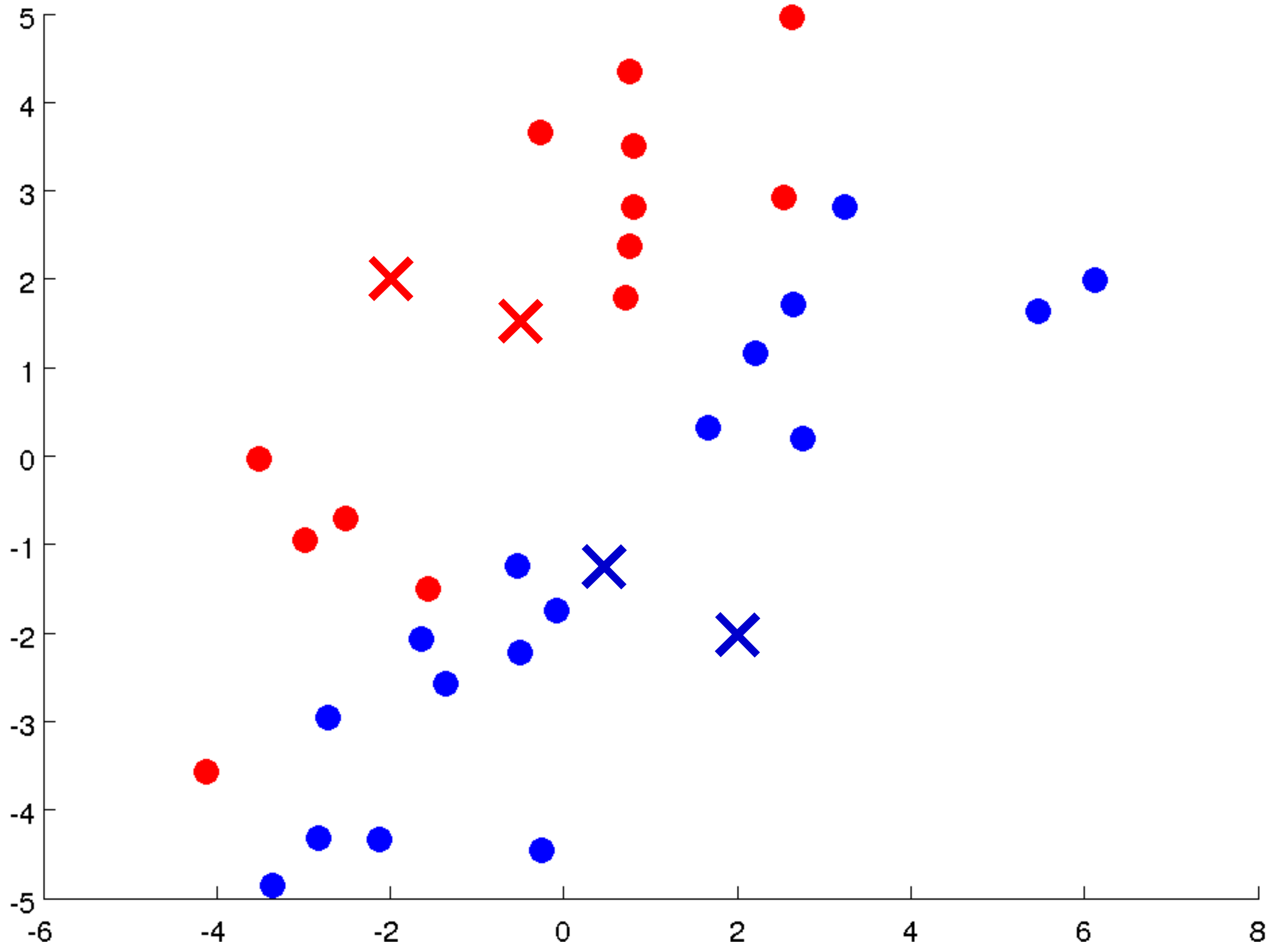
1) Randomly select K cluster centers $\mu_1, \ldots, \mu_K \in \mathbf{R}^p$
2) Assign each sample to the nearest cluster center
3) Update cluster centers to mean of samples in cluster
4) Repeat steps 2-3 until convergence

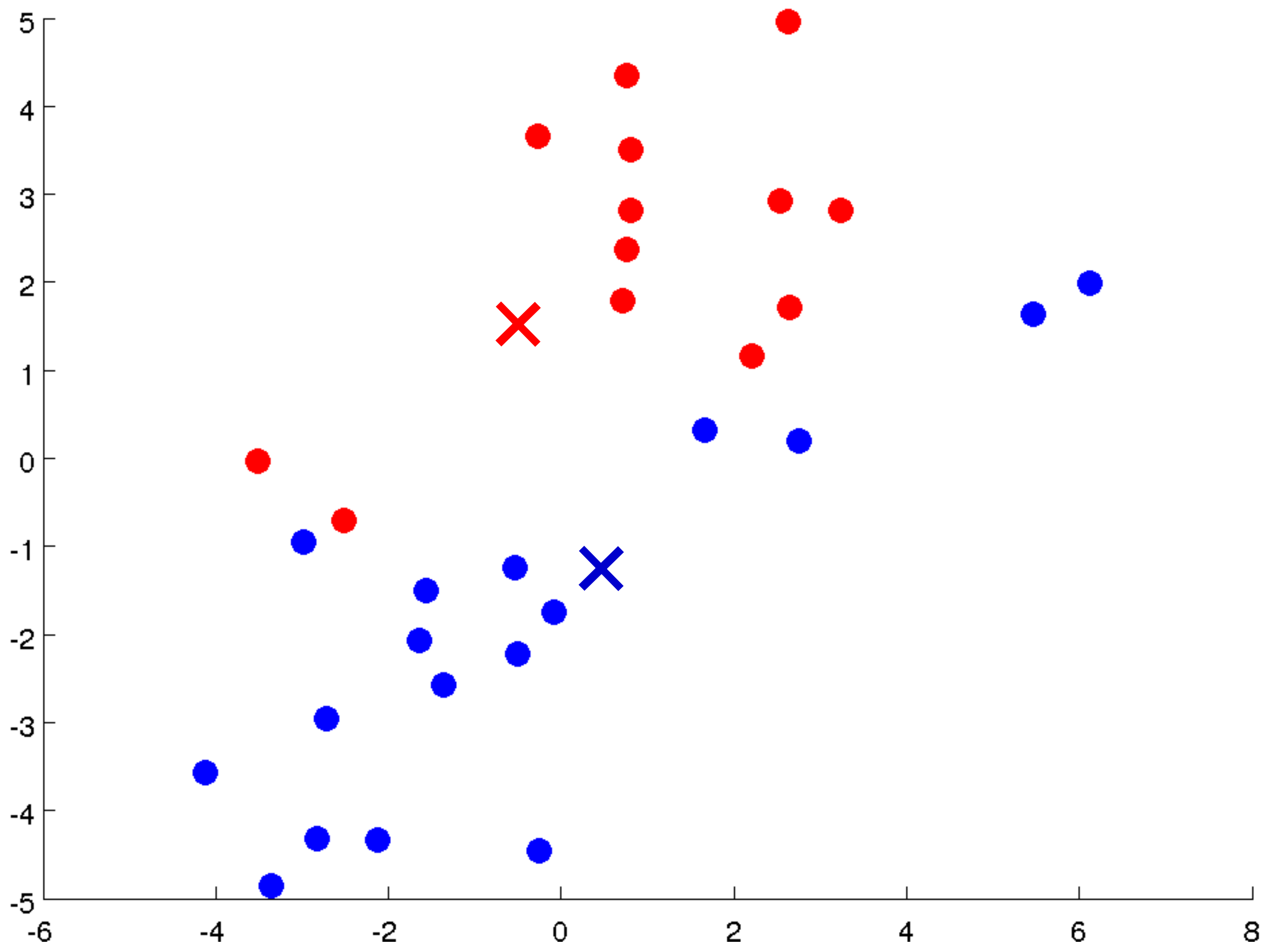

5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8

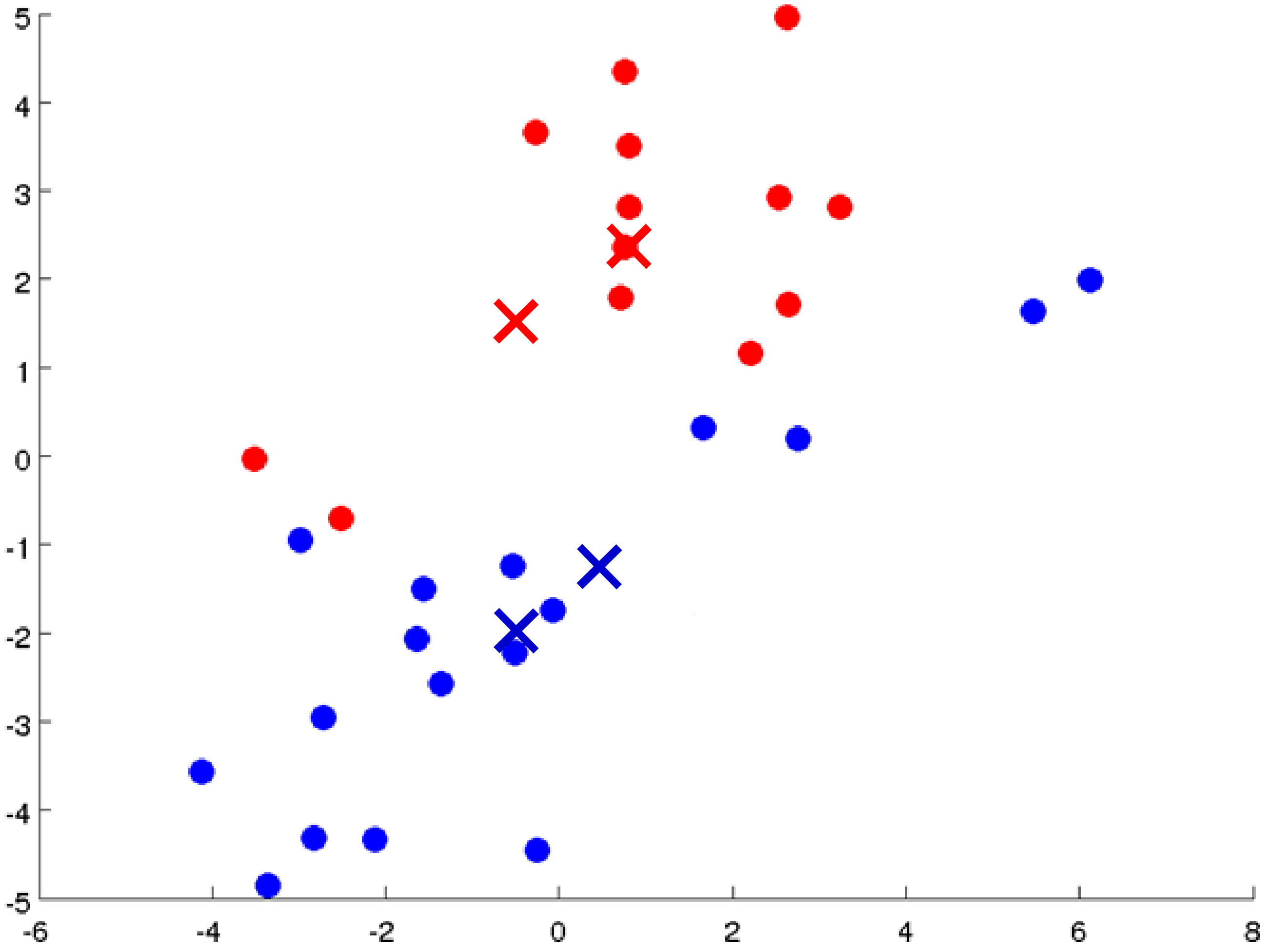

5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8

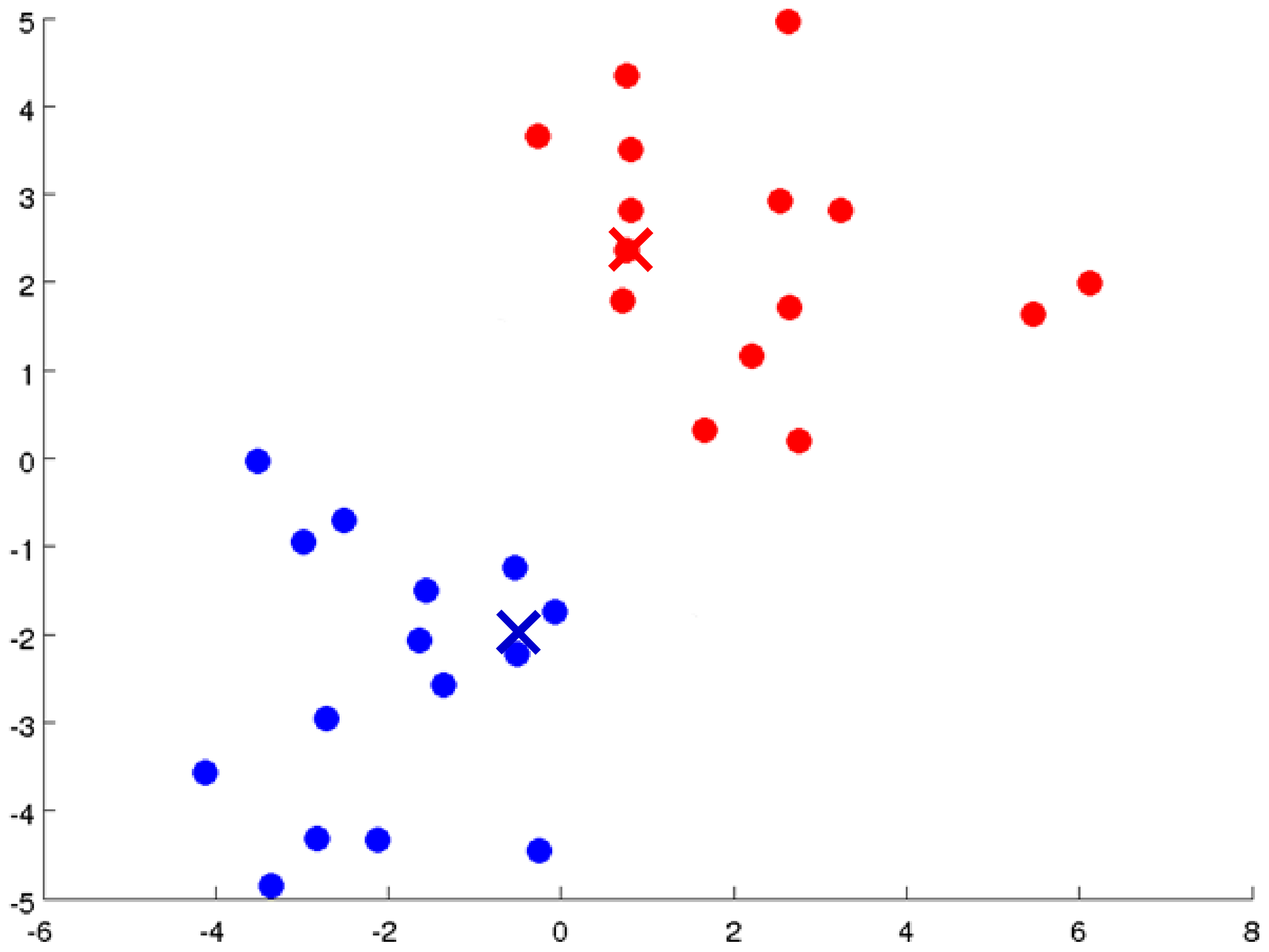


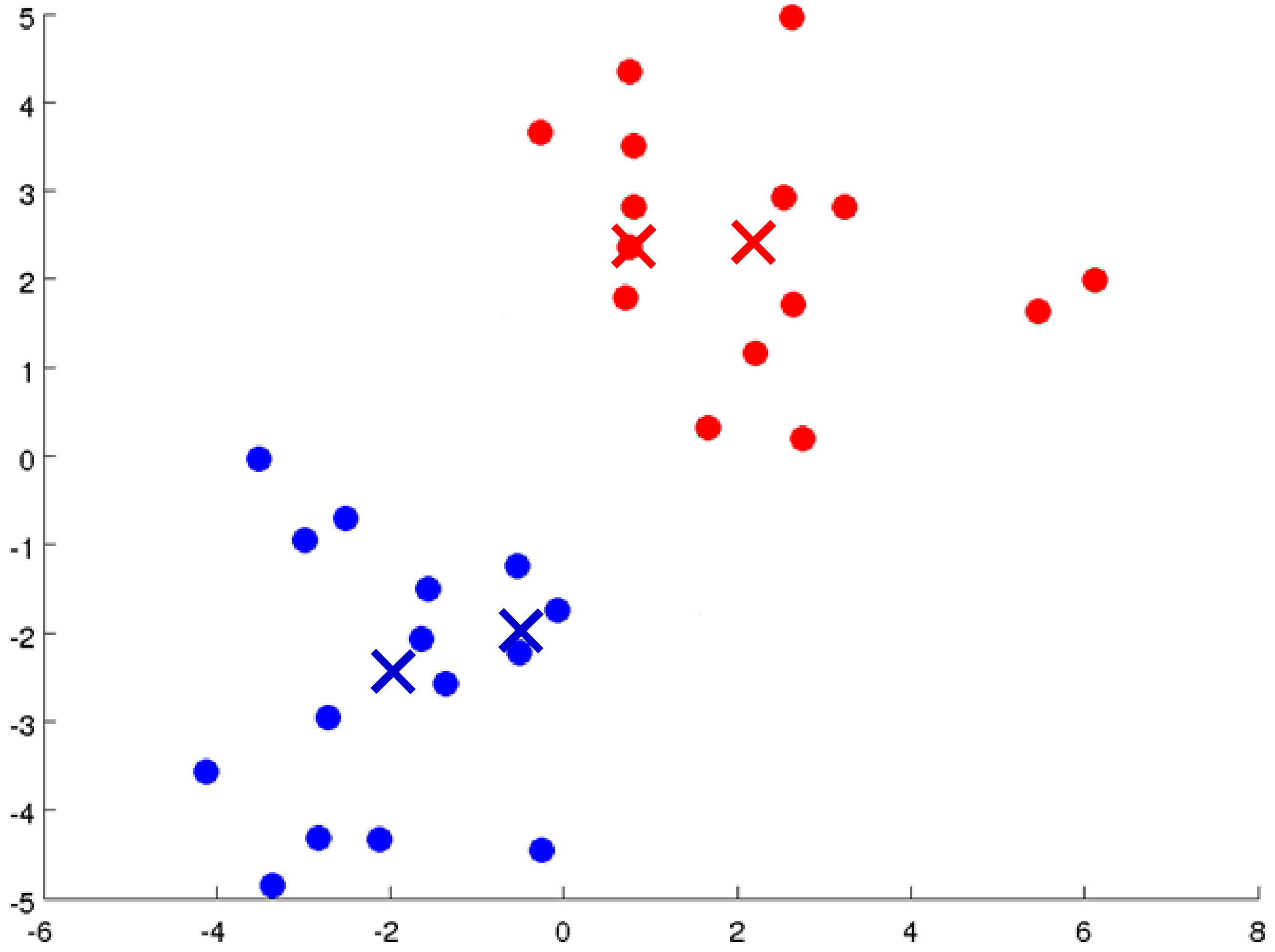
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8

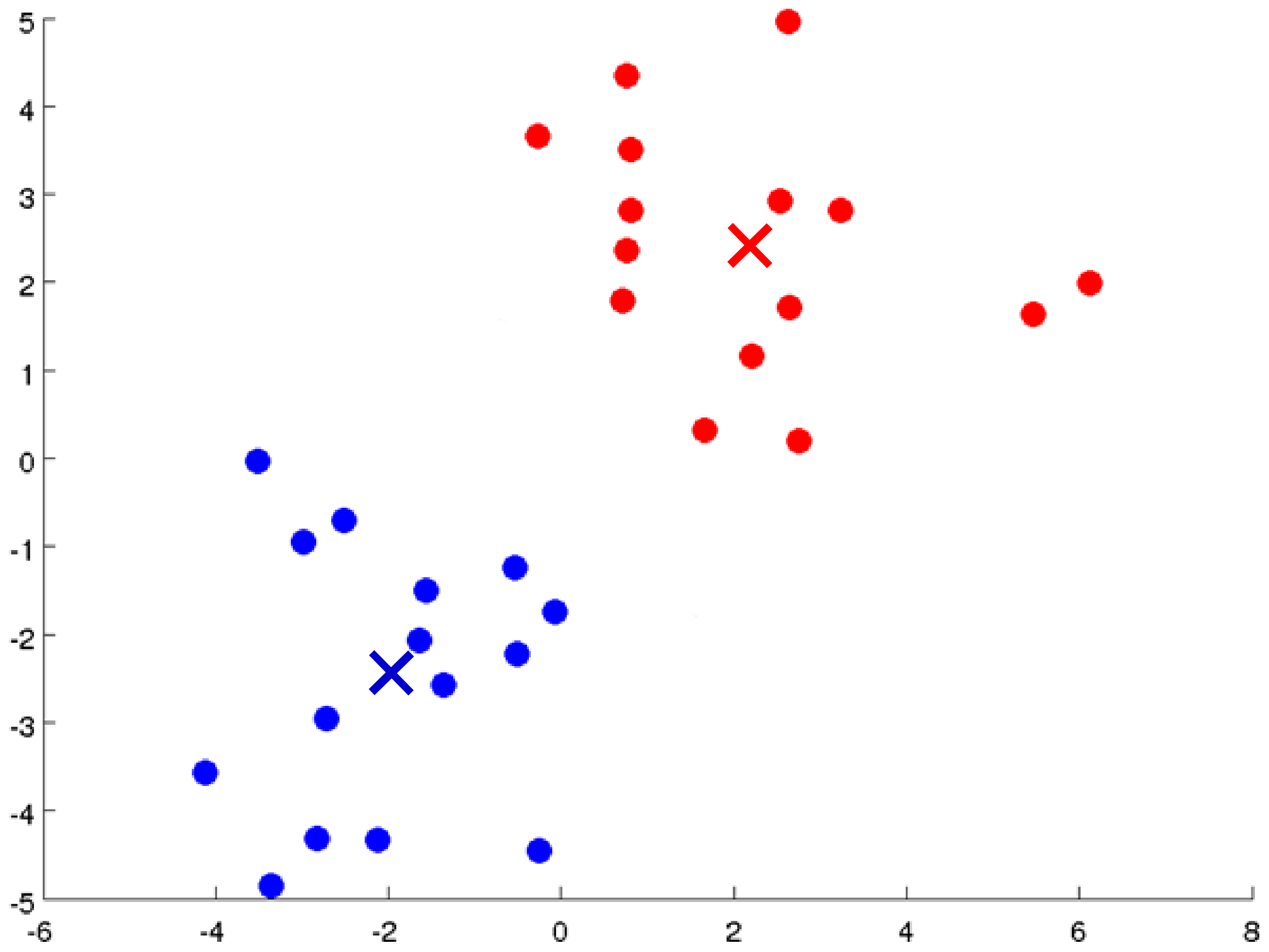

5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8

K-means algorithm

Input:

- K (number of clusters)
- Training set $\{x_1, x_2, \ldots, x_n\}$

$x_i \in R^p$ (drop $x_0 = 1$ convention)

K-means algorithm

```
Randomly initialize $K$ cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbf{R}^p$
Repeat {
        for $i$ = 1 to $n$
            $c^{(i)}$ := index (from 1 to $K$) of cluster centroid
                        closest to $x_i$
        for $k$ = 1 to $K$
            $\mu_k$ := average (mean) of points assigned to cluster $k$

        }
```

Andrew Ng

k-means clustering: algorithmic details

- Initial cluster center selection:
  - Random (uniform, normal) coordinates within data range
  - Random selection of the data points
  - Manual selection

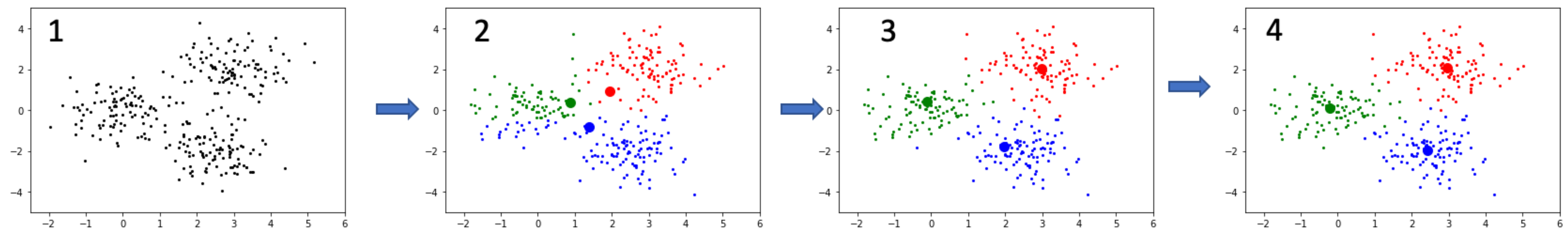
- Distance between a sample x_i and a cluster center μ_k:

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_k) = \Sigma\left(x_{ij} - \mu_{kj}\right)^2 \qquad \text{(or some other metric)}$$

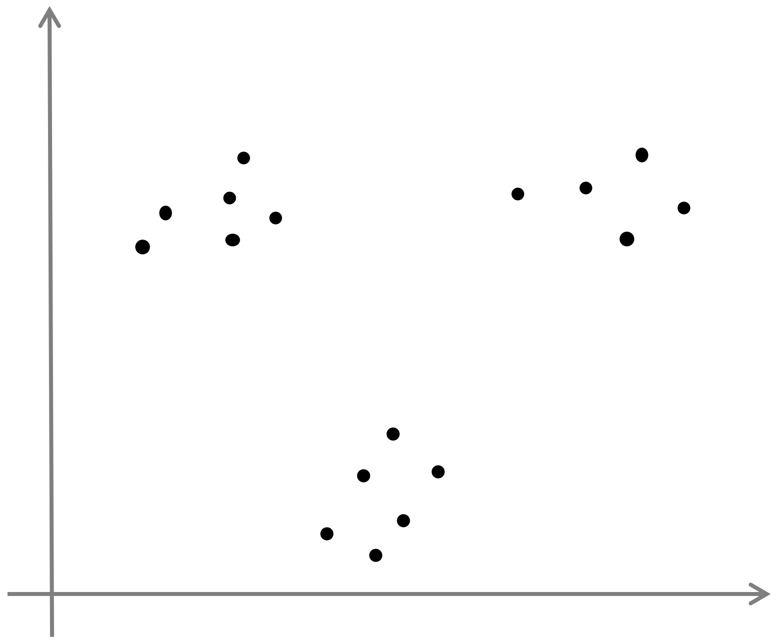
- Cluster centers are updated by calculating the average of all the sample vectors assigned to it:

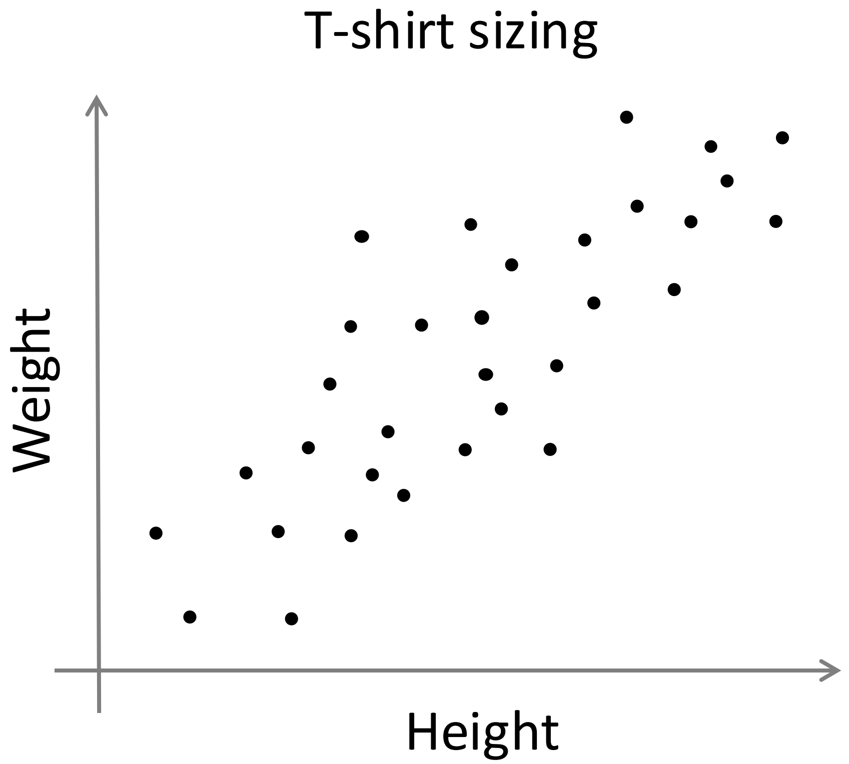
$$\boldsymbol{\mu}_k^{new} = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{(or median)}$$

Example: k-means with k=3



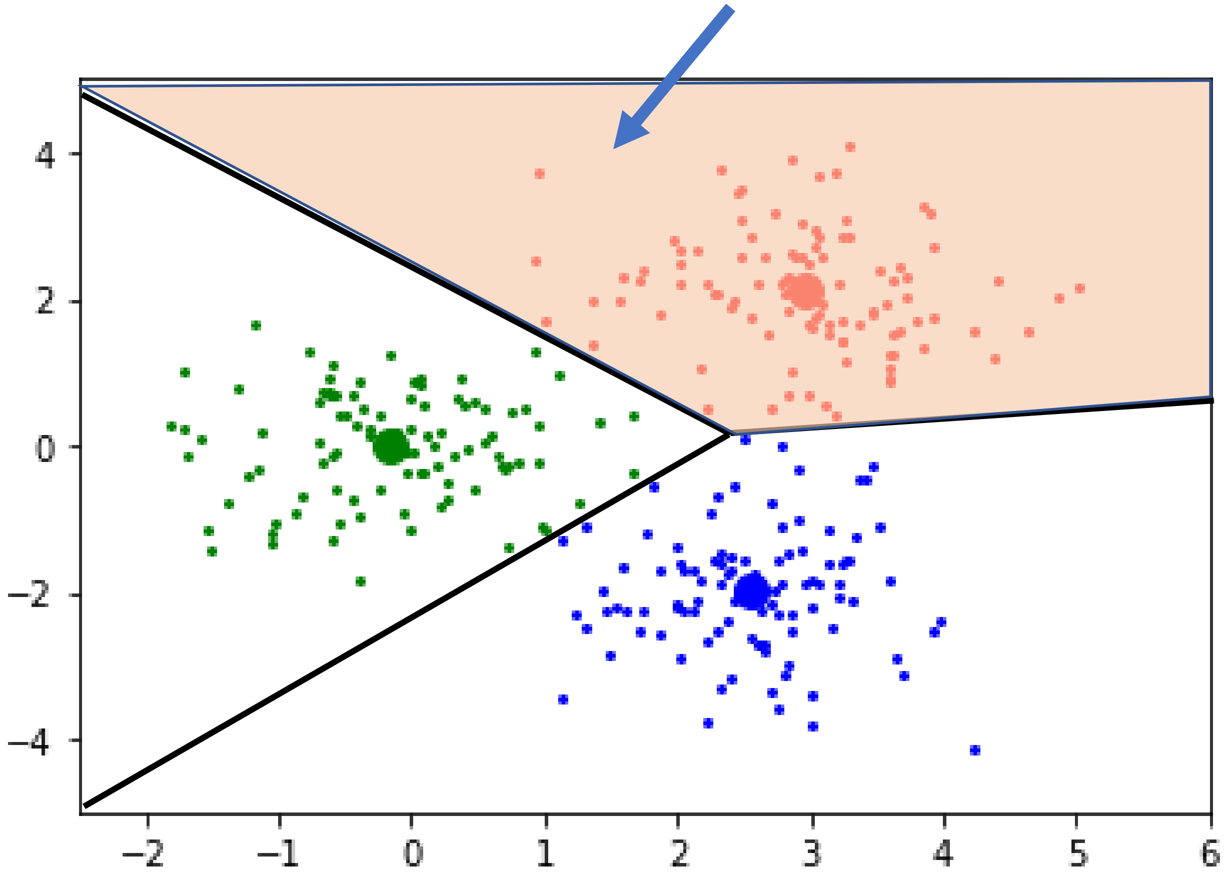
K-means for non-separated clusters





Voronoi diagram

Voronoi cell: part of the plane that is closer to one particular cluster center than to any other cluster center

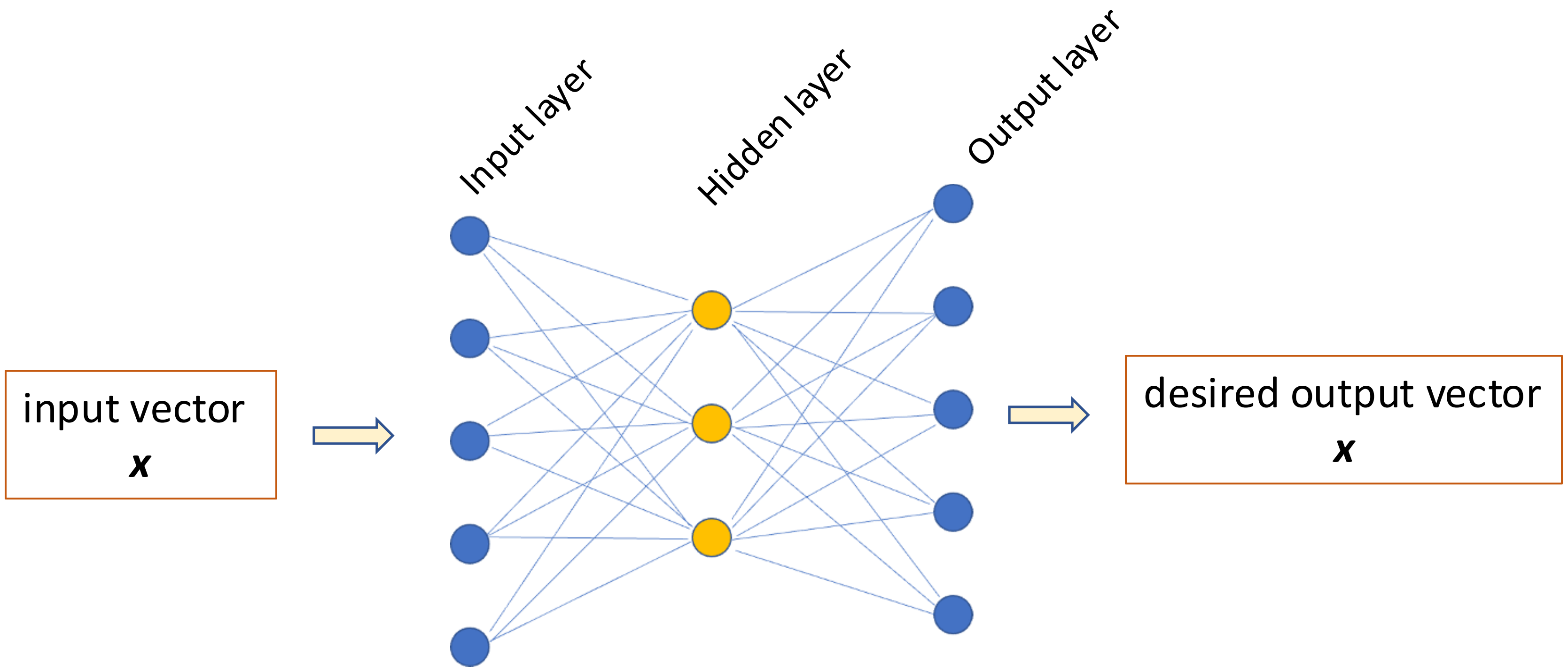


IN3050/IN4050, Lecture 10
Unsupervised Learning

4: Learning how to compress data
Ole Christian Lingjærde

Autoencoders

An autoencoder is a neural network that learns to produce an output similar to the input:



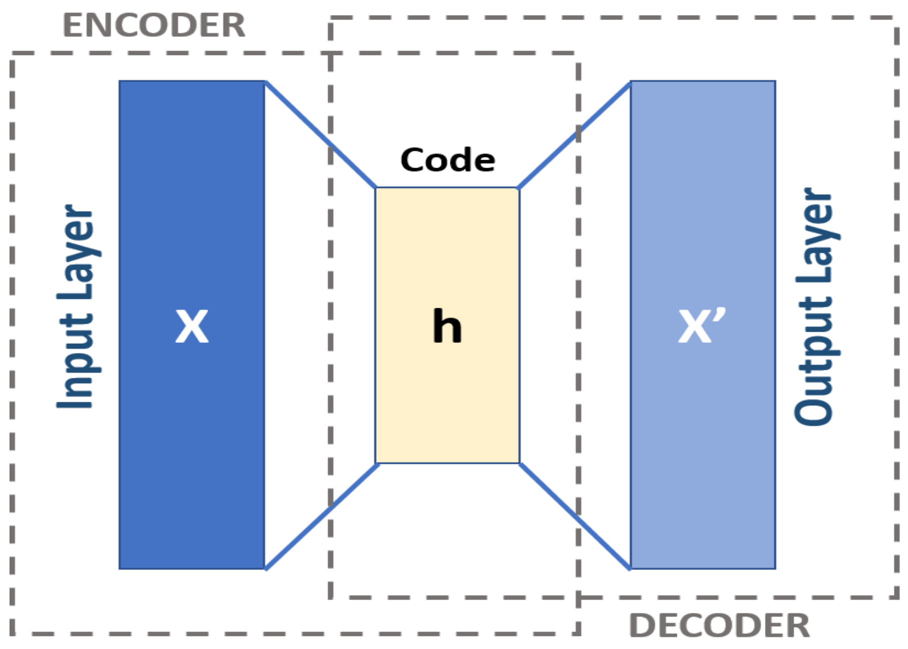
Autoencoders: loss function

To use backpropagation (or any other supervised learning algorithm) we need to define a loss function. For autoencoders, we often use <u>reconstruction loss</u>:

$$Loss(W) = \sum \left(\boldsymbol{x}_i - output(\boldsymbol{x}_i)\right)^2$$

Autoencoders learn to compress

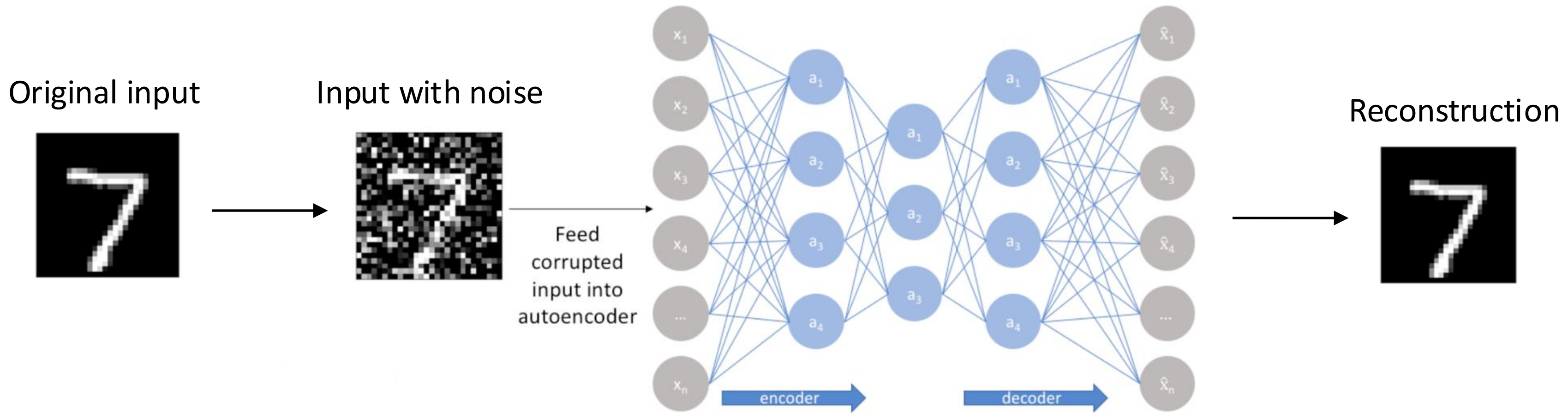
If the hidden layer is smaller than the input/output layers, the network learns a compact representation of the data:



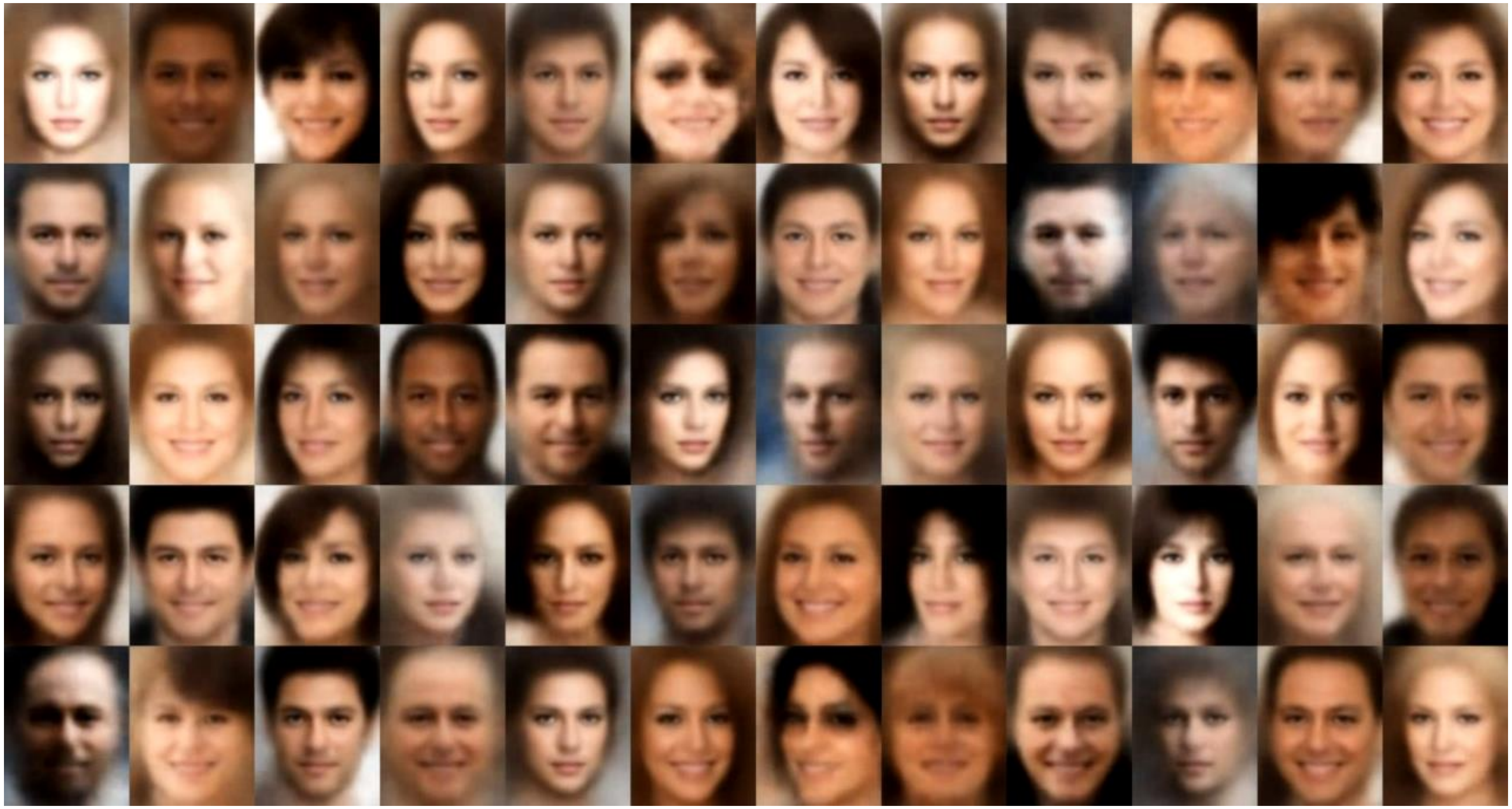
Special types of autoencoders

- Sparse autoencoders: trained to obtain a sparse hidden layer representation (= only a few hidden nodes active at a time)
- Denoising autoencoders: trained to recapture a denoised version of the input vector
- Contractive autoencoders: trained to obtain a hidden layer representation that is robust to small changes in the input
- Variational autoencoders: trained to obtain a representation useful also for generating new data from same distribution

Denoising autoencoder – example of use



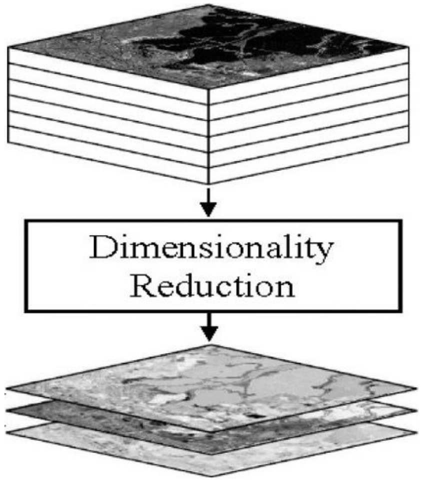
Variational autoencoder – example of use



Pictures generated by a variational autoencoder (https://github.com/wojciechmo/vae)

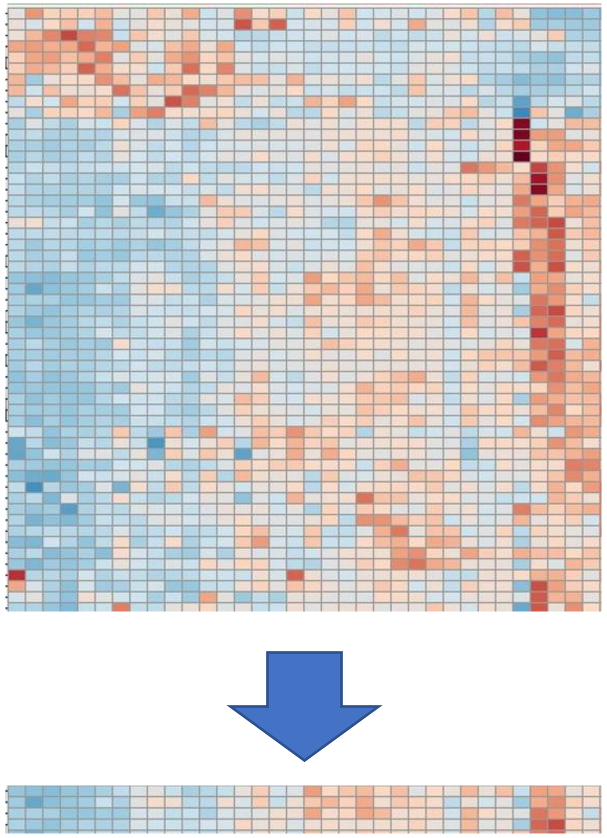
IN3050/IN4050, Lecture 10
Unsupervised Learning

5: Learning how to represent data in low dimension
Ole Christian Lingjærde



Dimensionality reduction: purpose

- To plot high-dimensional data
- To visually explore structural features of a data set
- To reduce the number of features before further analyses
- To reduce noise



How is dimension reduction different from compression?

Autoencoders can also be used to compress data – but the aim is different!

Compression: encoded version of data should be compact and possible to decode to original format. Not for human interpretation.

Dimension reduction: encoded version of data should preserve only important structure and be useful for human interpretation or input to other analyses. Decoding is not an aim.

Dimensionality reduction methods

Goal: find Low-Dimensional Representation (LDR) of the data

Principal components analysis (PCA):

- Find LDR maximizing the variance

Multidimensional scaling (MDS)

- Find LDR preserving pairwise distances

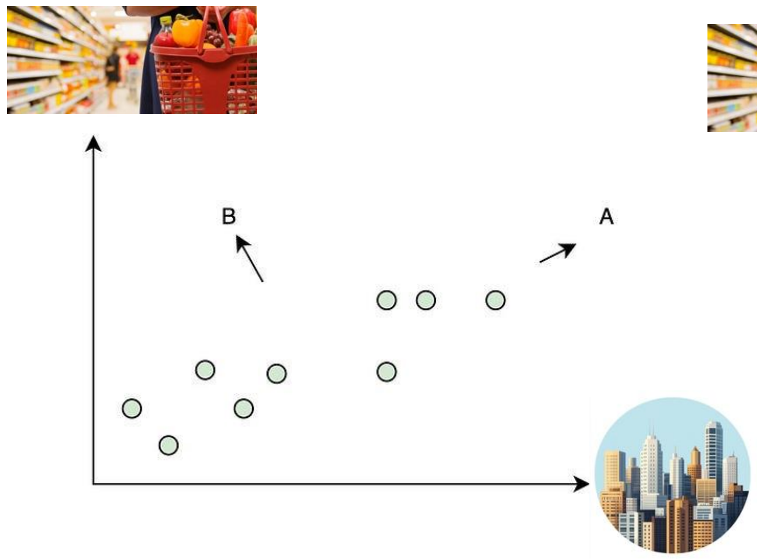
T-distributed stochastic neighbour embedding (t-SNE)

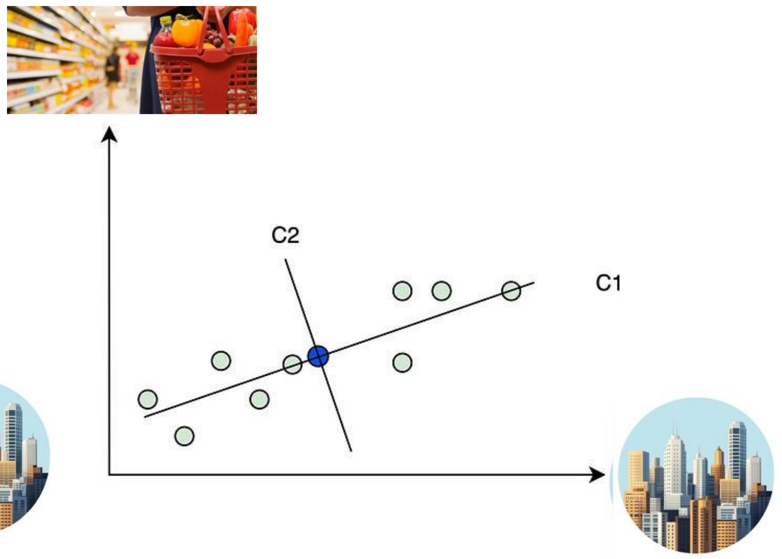
- Find LDR preserving neighbor relations in probabilistic sense

Uniform Manifold Approximation and Projection (UMAP)

- Find LDR preserving topological structure of data

Principal components analysis (PCA)

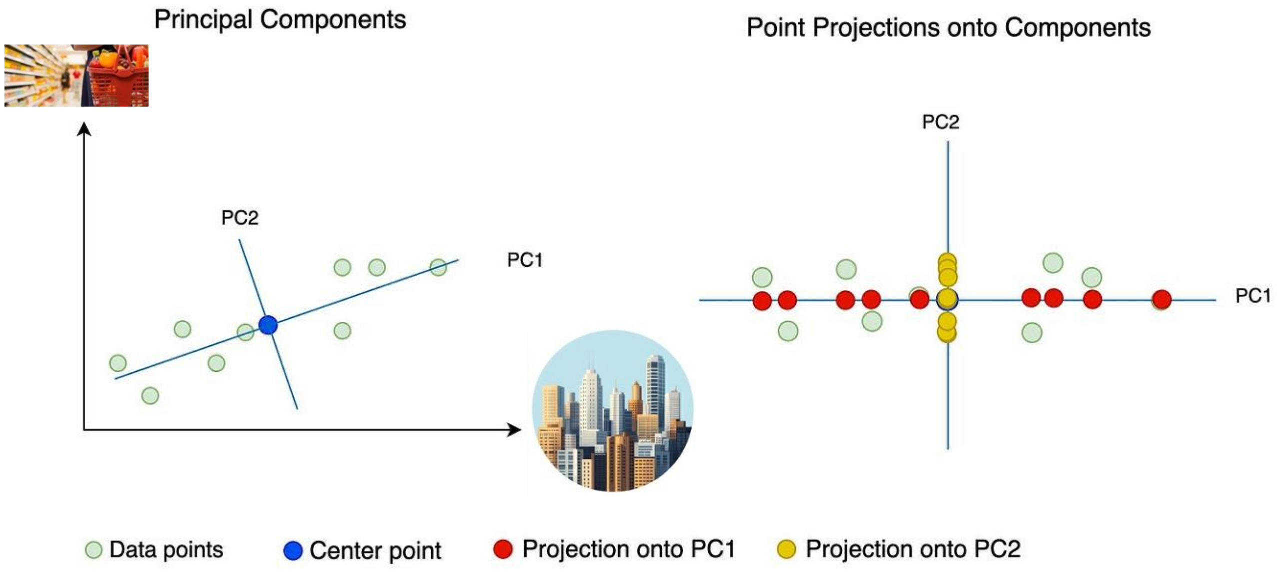


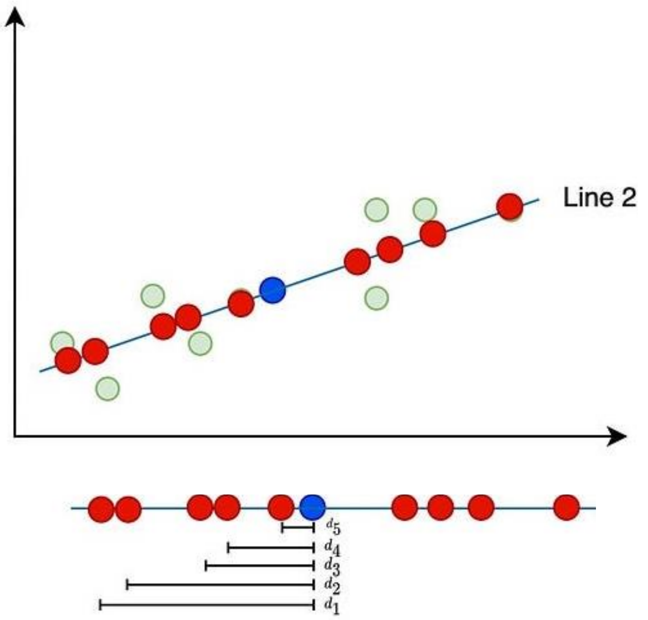


Principal Component Analysis (PCA) is about finding the best way to represent data in fewer dimensions while preserving the most important information. Instead of working with raw features, PCA identifies new axes — called **principal components** — that capture the most variance in the data.

Image source

Principal components analysis (PCA)





The first principal component, PC1, always represents the linear component that lies on the highest variation. The first component explains the higher variation in the dataset, so it has more information than the other.

Sum of Squared Distances

$$\sum_{i=1}^{n} d_i^2$$

The line that gives the highest sum of squared distances will be **the PC1**.

Image source

Principal components analysis (PCA)

1) Start with a data set:

$$X = \begin{array}{cl} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{bmatrix} & \begin{array}{l} \text{sample 1} \\ \text{sample 2} \\ \text{......} \\ \text{sample n} \end{array} \end{array}$$

2) Center the data:

$$Y = \begin{bmatrix} x_{11} - m_1 & x_{12} - m_2 \\ x_{21} - m_1 & x_{22} - m_2 \\ \vdots & \vdots \\ x_{n1} - m_1 & x_{n2} - m_2 \end{bmatrix}$$

3) Find weights $\boldsymbol{w} = (w_1, w_2)$ maximizing length of $Y\boldsymbol{w}$:

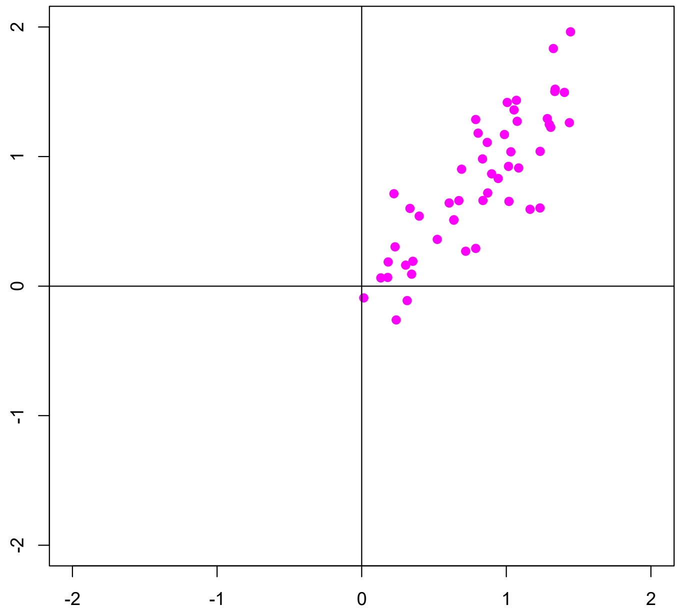
$$\widehat{\boldsymbol{w}} = \underset{\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \|Y\boldsymbol{w}\|$$

That's it! The vector $\widehat{\boldsymbol{w}}$ is called the 1st principal component of X.

PCA: visual interpretation

1) Start with a data set:

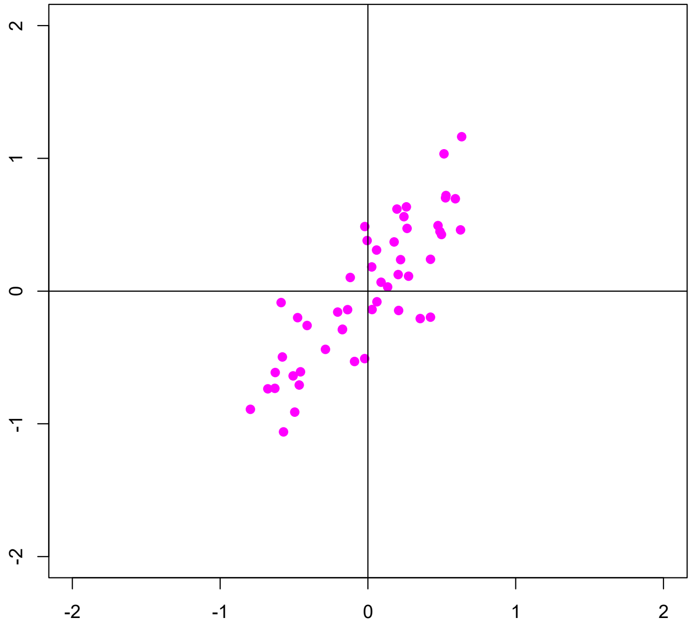
$$X = \begin{array}{c|l} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{bmatrix} & \begin{array}{l} \text{sample 1} \\ \text{sample 2} \\ \ldots\ldots \\ \text{sample n} \end{array} \end{array}$$



PCA: visual interpretation

2) Center the data:

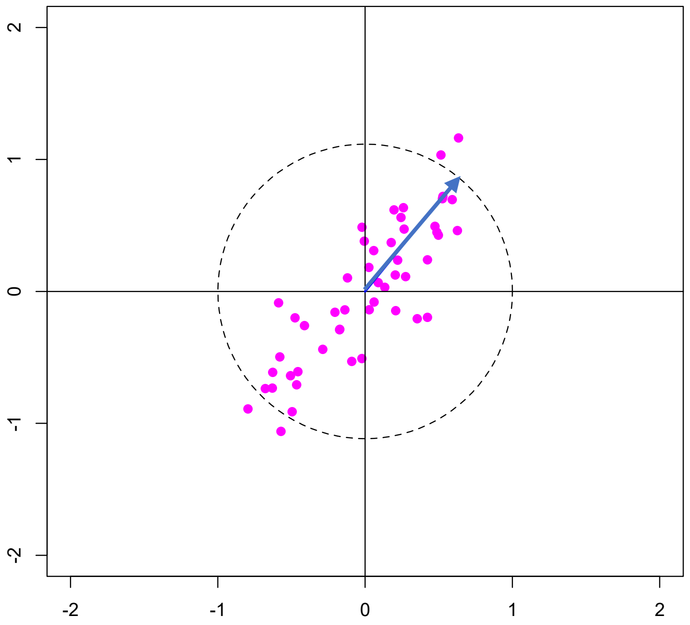
$$Y = \begin{bmatrix} x_{11} - m_1 & x_{12} - m_2 \\ x_{21} - m_1 & x_{22} - m_2 \\ \vdots & \vdots \\ x_{n1} - m_1 & x_{n2} - m_2 \end{bmatrix}$$



PCA: visual interpretation

3) Find weights $\boldsymbol{w} = (w_1, w_2)$ maximizing length of $Y\boldsymbol{w}$:

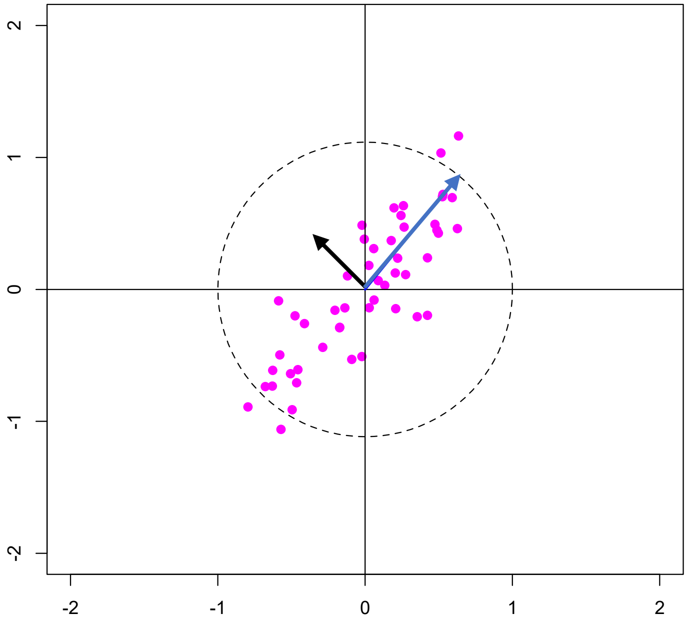
$$\hat{\boldsymbol{w}} = \underset{\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \|Y\boldsymbol{w}\|$$



PCA: visual interpretation

3) Find weights $\boldsymbol{w} = (w_1, w_2)$ maximizing length of $Y\boldsymbol{w}$:

$$\hat{\boldsymbol{w}} = \underset{\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \|Y\boldsymbol{w}\|$$



IMPORTANT: The weight vector points in the direction where the data are most spread out.

If we project the data points onto this weight vector we get a 1d-version of the data points.

No other weight vector will give higher variance of the 1d projections.

PCA: how do we compute it?

```
import numpy as np

# Create a small toy data set with 100 samples and 2 features
from random import normalvariate as rnorm
x = [rnorm(2,0.8) for e in range(100)]
y = [x[e] + rnorm(0,0.8) for e in range(100)]
X = np.transpose(np.array([x,y]))

# Center columns in X
Y = X - X.mean(axis=0)

# Calculate weight vector w
Z = np.transpose(Y) @ Y
eigenvalues, eigenvectors = np.linalg.eig(Z)
w = eigenvectors[:, np.argmax(eigenvalues)]
```

Note: there are also dedicated tools for PCA in python, for example in scikit-learn.