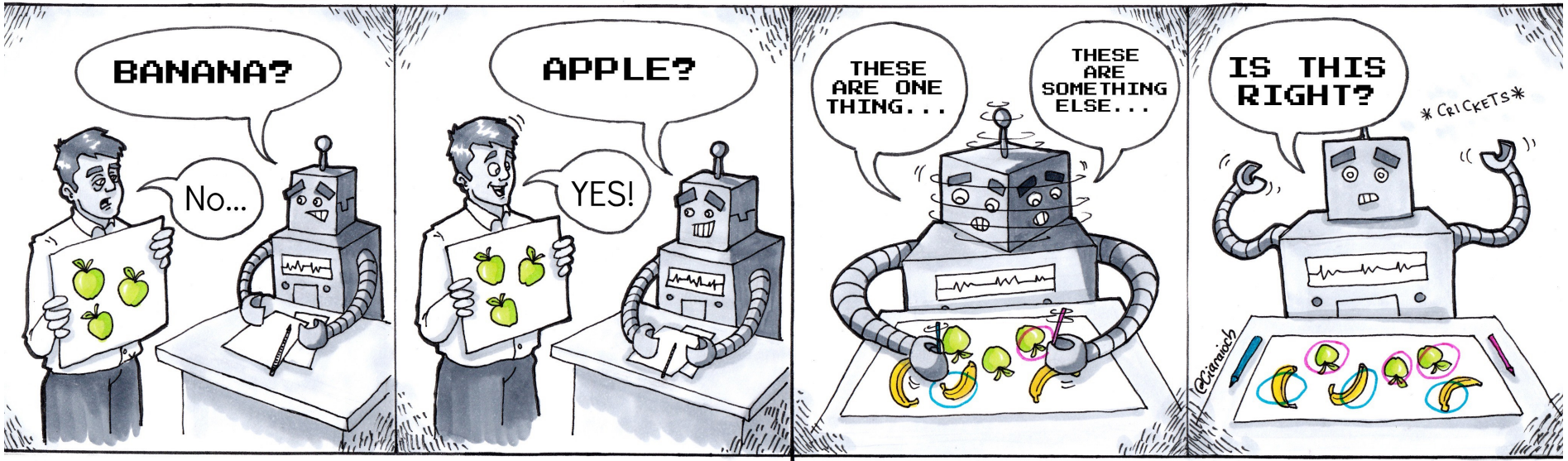
IN3050/IN4050, Lecture 11
Unsupervised Learning

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 11
Unsupervised Learning

1: Introduction

Ole Christian Lingjærde



Supervised Learning

Unsupervised Learning

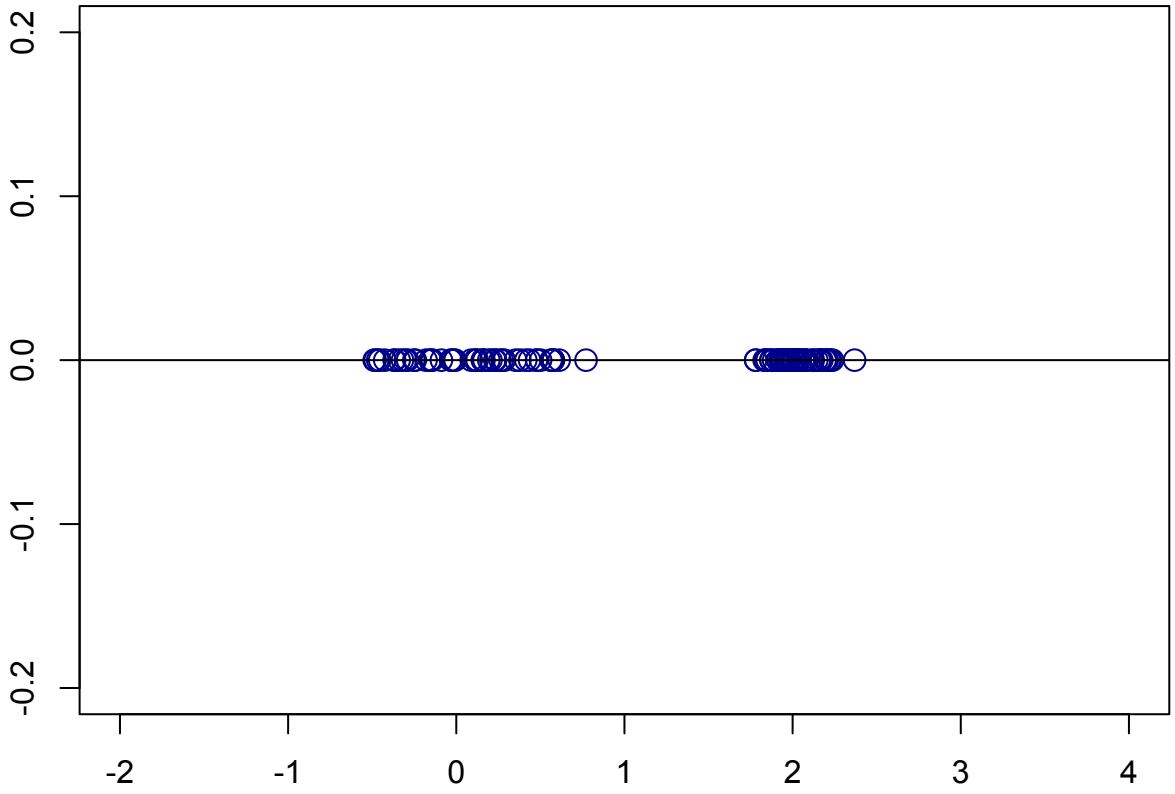
https://twitter.com/athena_schools/status/1063013435779223553/photo/1

Supervised vs. unsupervised learning

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

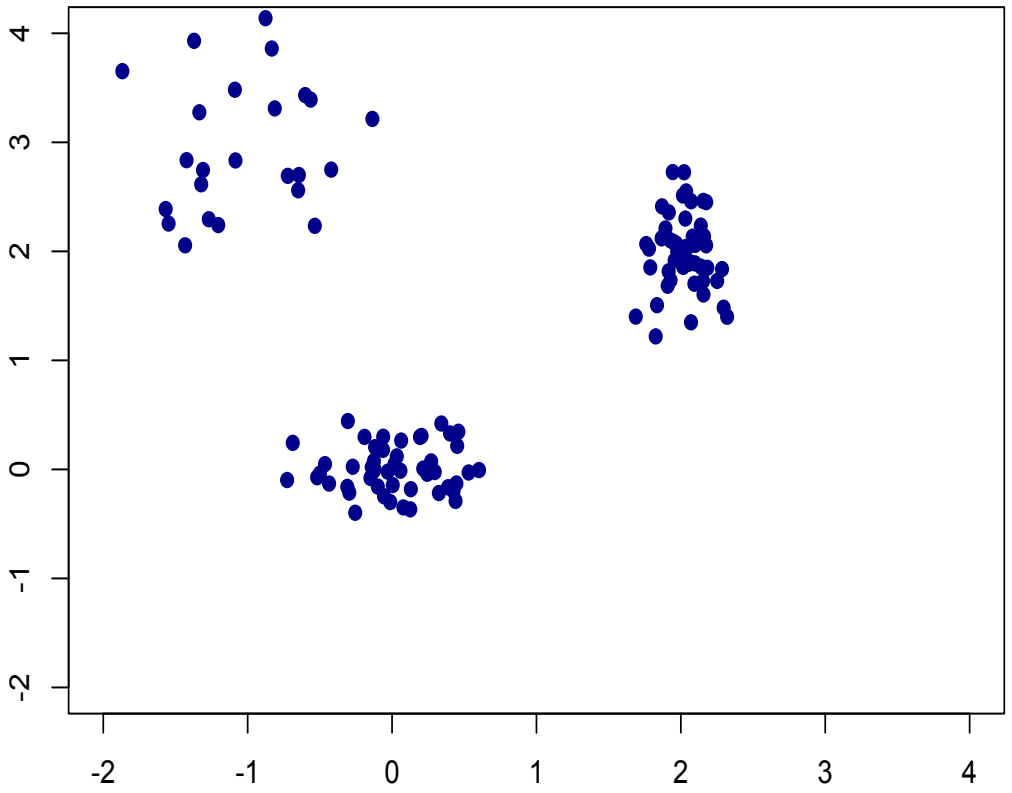
Unsupervised learning: we have a training set consisting of only a feature vector $\boldsymbol{x} = (x_1, \ldots, x_m)$ for each sample. Goal is to learn something interesting about the distribution of the $\boldsymbol{x}$'s.

Example (1D)



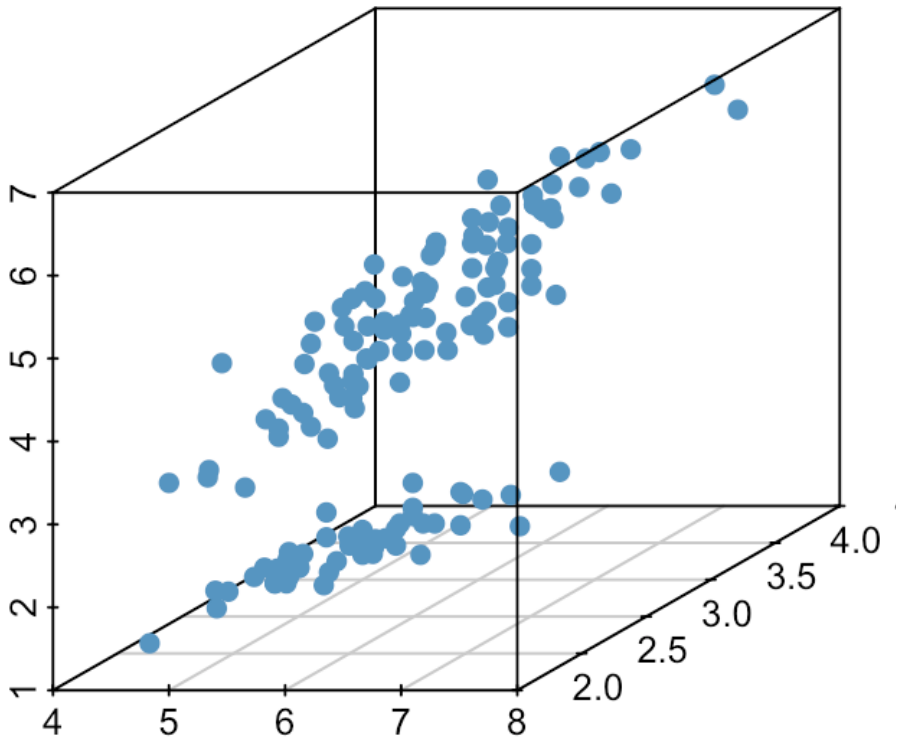
Each sample has a single feature x (a real number)

Example (2D)



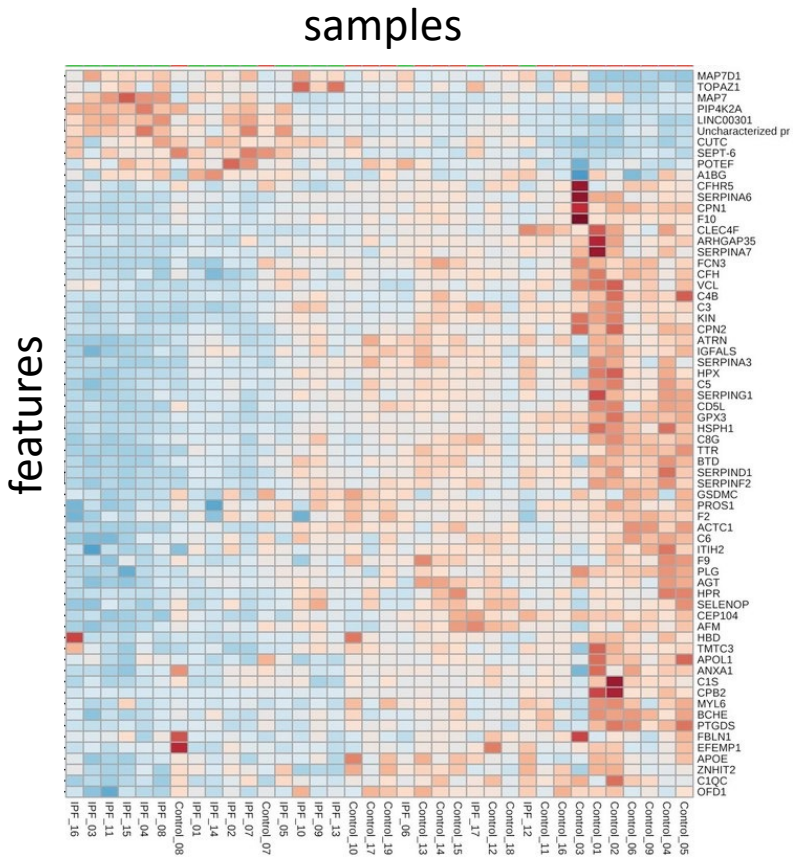
Each sample has two features $\boldsymbol{x} = (x_1, x_2)$

Example (3D)



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

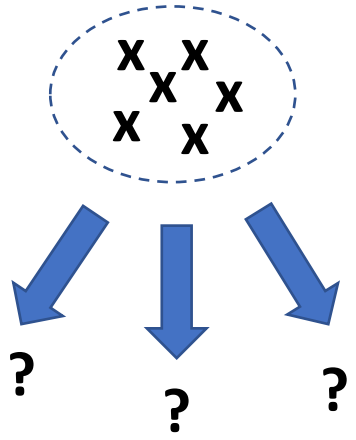
Example (p dimensions)



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



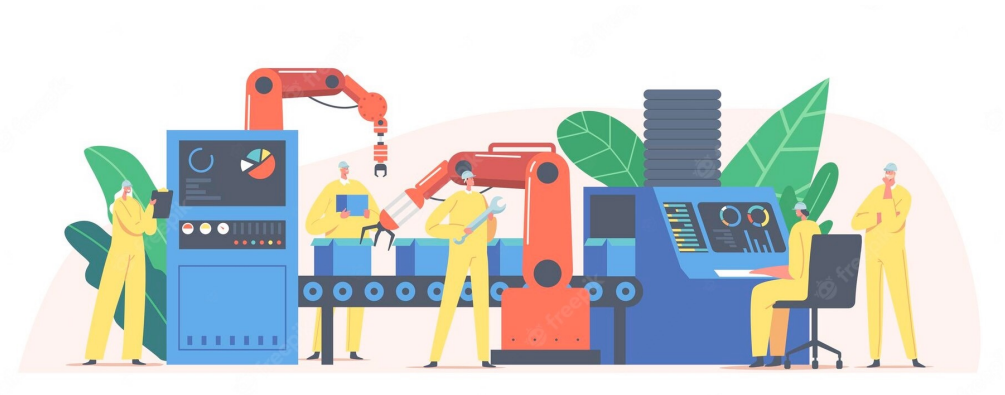
Unsupervised learning:

- 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

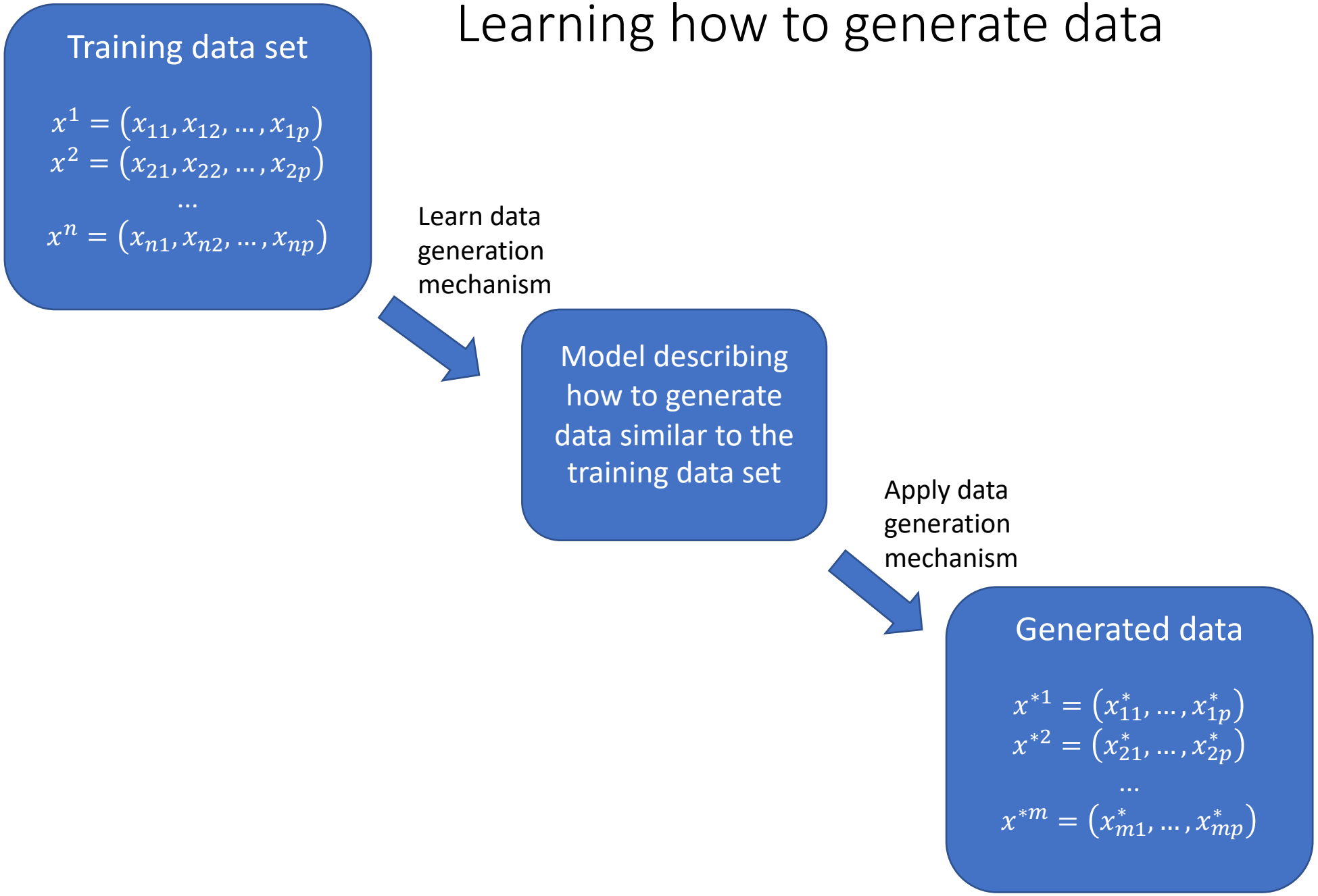
IN3050/IN4050, Lecture 11
Unsupervised Learning

2: Learning how to generate similar data

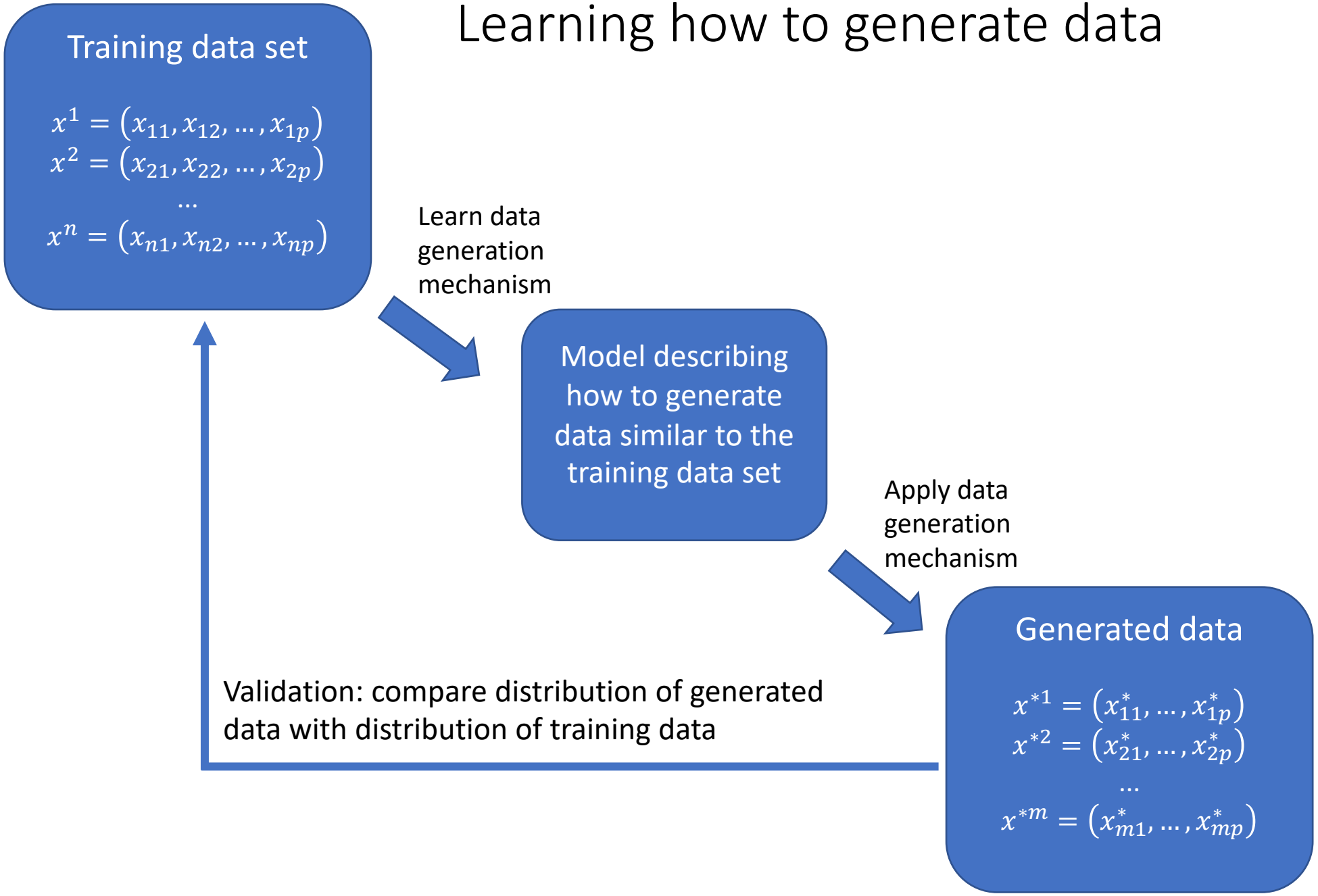
Ole Christian Lingjærde



Learning how to generate data



Learning how to generate data



Random variables

Discrete random variable:

Can take a discrete set of different values

Example: $X = 0, 1, 2, 3, \ldots$

Continuous random variable:

Can take a continuous set of different values

Example: $X \in [a, b]$

Discrete random variables

A probability distribution for a discrete random variable describes the probability of different values occurring.

Example:

- Roll a fair dice with six sides
- X = 1, 2, 3, …., 6
- P(X=x) = 1/6 for x=1,2,…,6



Continuous random variables

A probability distribution for a continuous random variable must be formulated slightly differently.

Example:

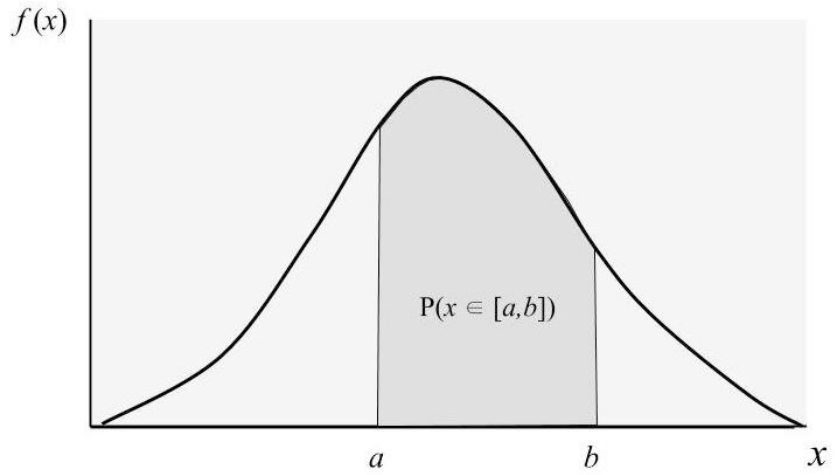
- Measure the time it takes for a soap bubble to burst (measured in hours and with decimals)
- $X \in [0, 1]$
- $P(X=x) = 0$ for any given time point x



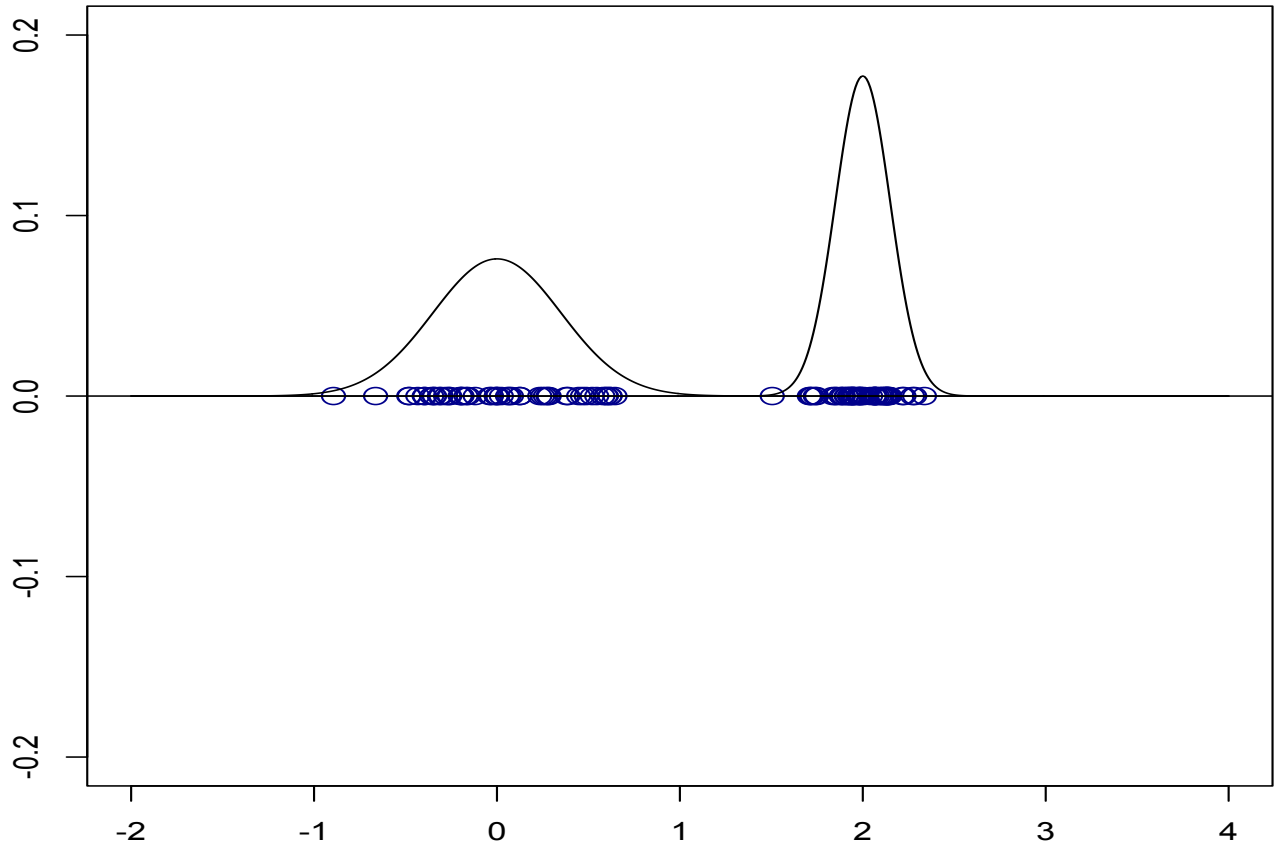
Density functions

The problem in the last example was that the probability of X taking a particular value x is infinitely small.

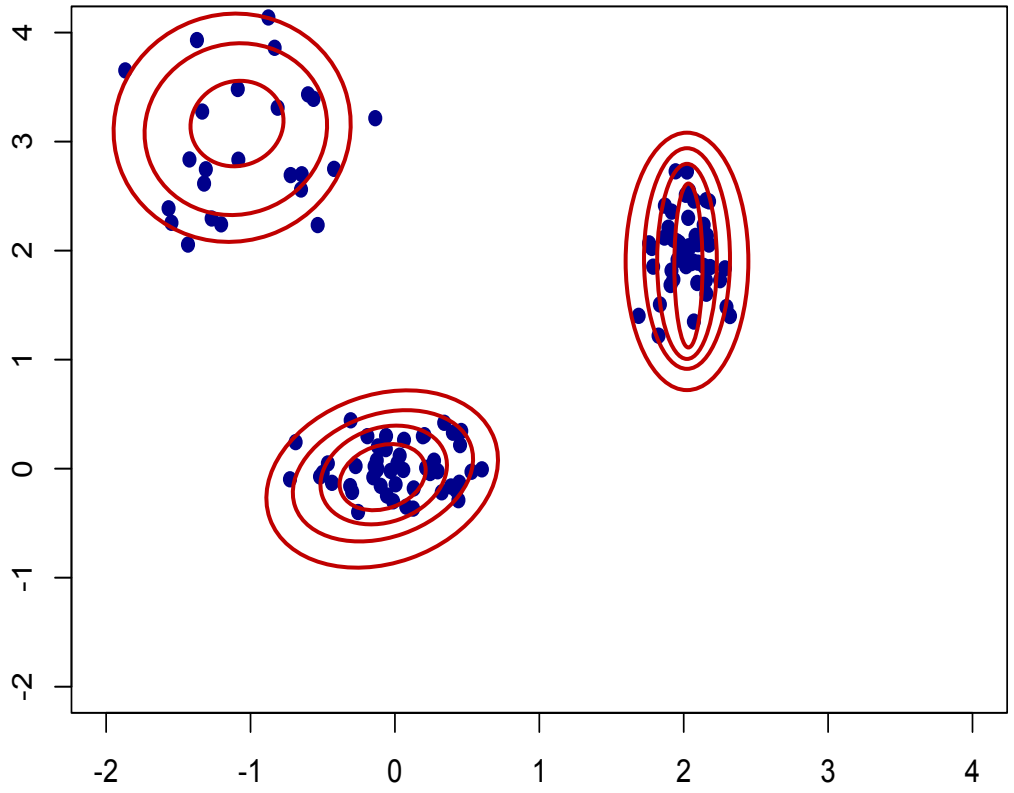
Solution: define instead a density function f(x) and define P(X∈ [a, b]) = area under curve on interval [a,b]



Density function: 1D example



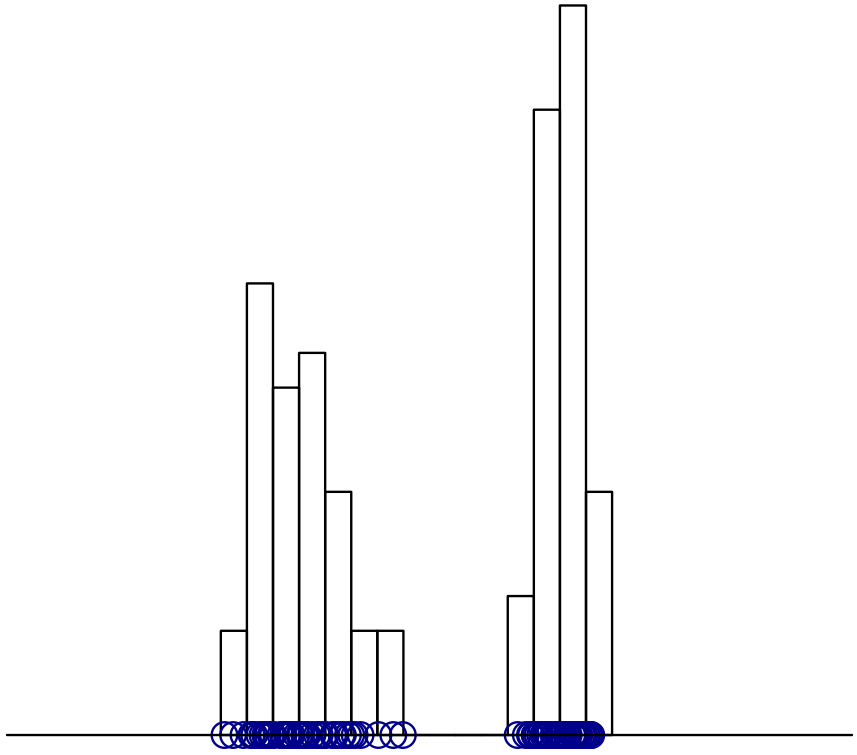
Density function: 2D example



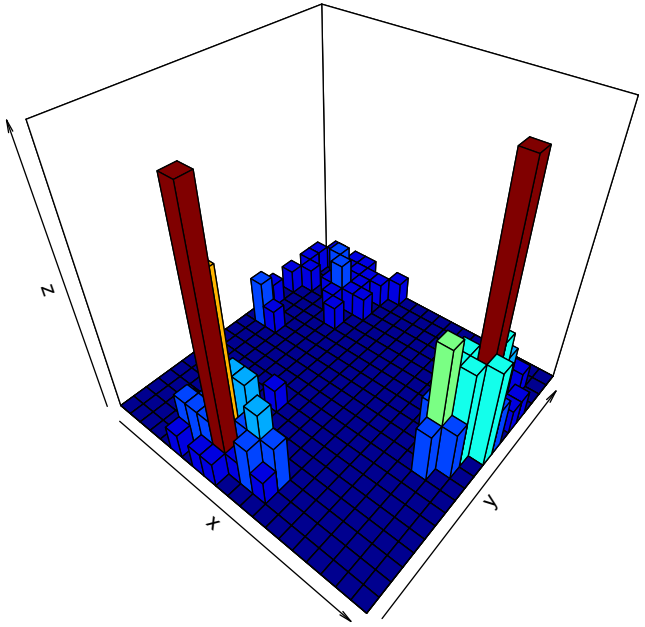
Learning a density - examples

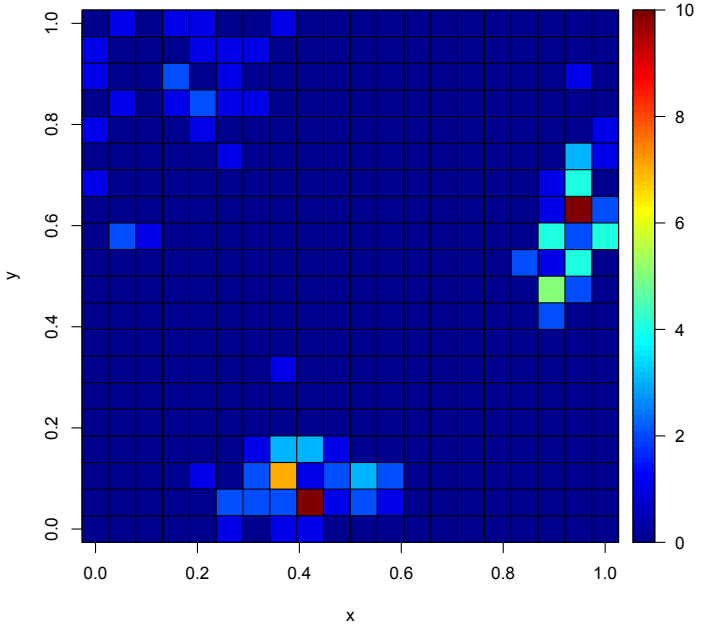
- Histogram
- Smooth density estimator
- Generative adversarial network (GAN)

Histogram: 1D example



Histogram: 2D example



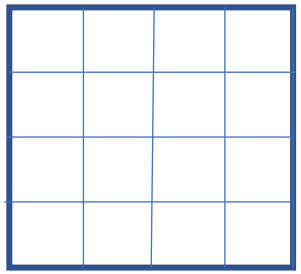


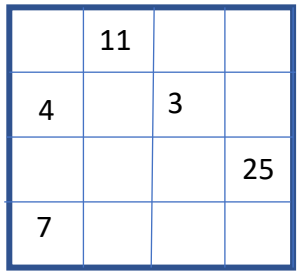
Histogram: how to learn it

Input: $x_1, \ldots, x_n \in \mathbf{R}^p$

- Divide the space into a number of cells by splitting each axis into several intervals.
- Count how many observations x_i there are in each cell
- Output a table (plot) with cell locations and cell counts

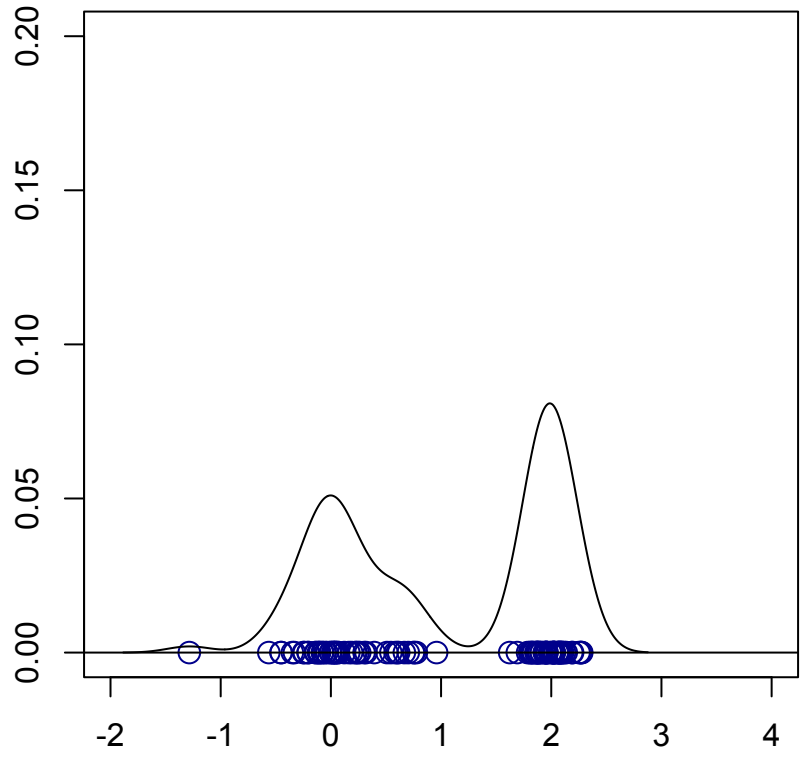






Cell	Count
(0,0)	0
(1,0)	11
(2,0)	0

Smooth density estimator: example



Smooth density estimator: how to train a kernel density estimator

Training data: $x_1, \ldots, x_n \in \mathbf{R}$ Test data: x

- Select a kernel function $K(x)$
  - A kernel is a "0-detector" with high positive value for x close to 0 and gradually lower values for x far from 0

- Compute $K(\frac{x-x_i}{h})$ for $i = 1,2, \ldots, n.$
  - This detects if x is close to any of the points in the training data set, and h>0 controls the detection sensitivity

- Let $f(x) = \frac{1}{nh}\sum_i K(\frac{x-x_i}{h})$
  - This adds together all detector values, so $f(x)$ is higher the more training data points there are in neighborhood of x

Learning a density: generative adversarial networks (GANs)



From Generative Adversarial Nets, 2014

Training set: MNIST handwritten digit database

Learning a density: generative adversarial networks (GANs)

Computer generated pictures of non-existing people:



Training set: CelebFaces Attributes Dataset (CelebA), a large-scale face attributes dataset with more than 200K celebrity images

IN3050/IN4050, Lecture 11
Unsupervised Learning

3: Learning how data are grouped

Ole Christian Lingjærde



Clustering

Learning how the data are grouped is called clustering.

Some clustering methods:

- Hierarchical clustering
- k-means clustering
- Competitive learning

Hierarchical clustering

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



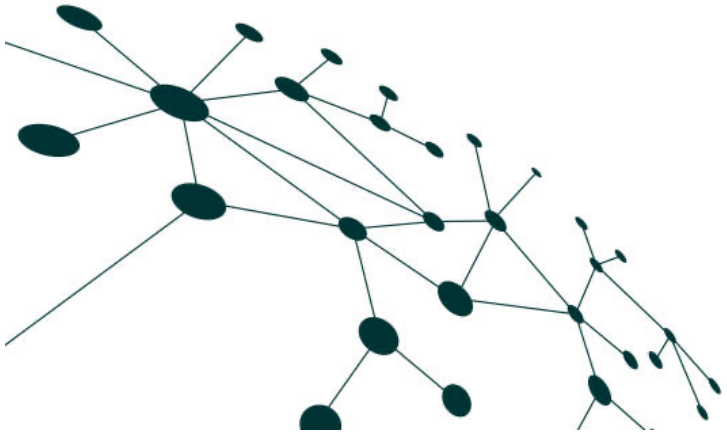
https://harshsharma1091996.medium.com/hierarchical-clustering-996745fe656b

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

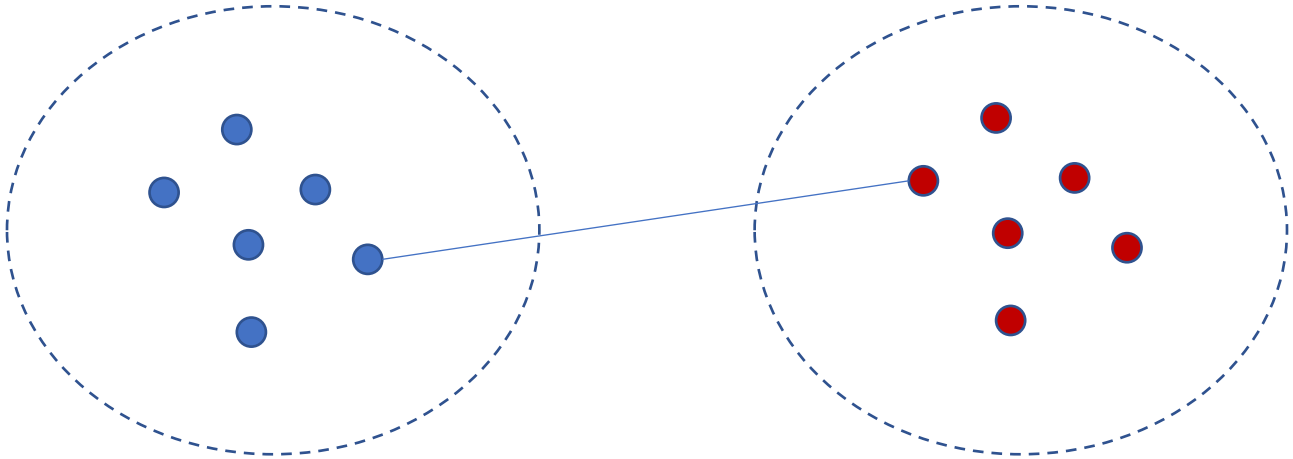
Hierarchical clustering: how to define the distance between two clusters C_1 and C_2

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



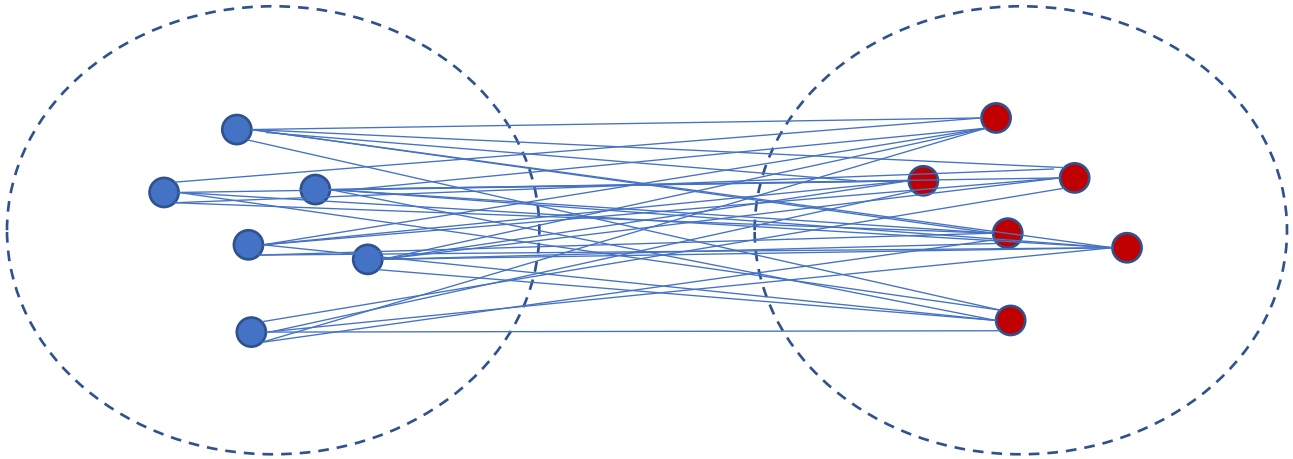
Single linkage

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



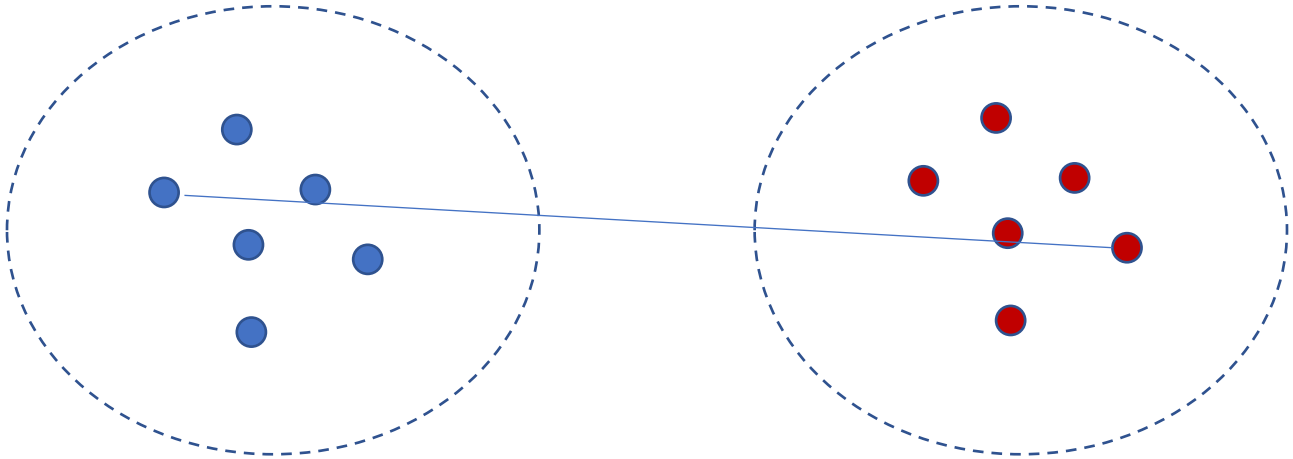
Average linkage

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



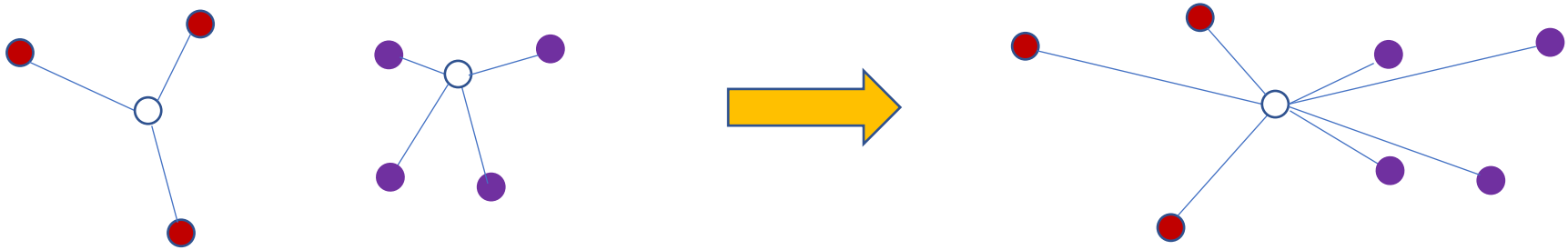
Complete linkage

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



Ward linkage

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



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:

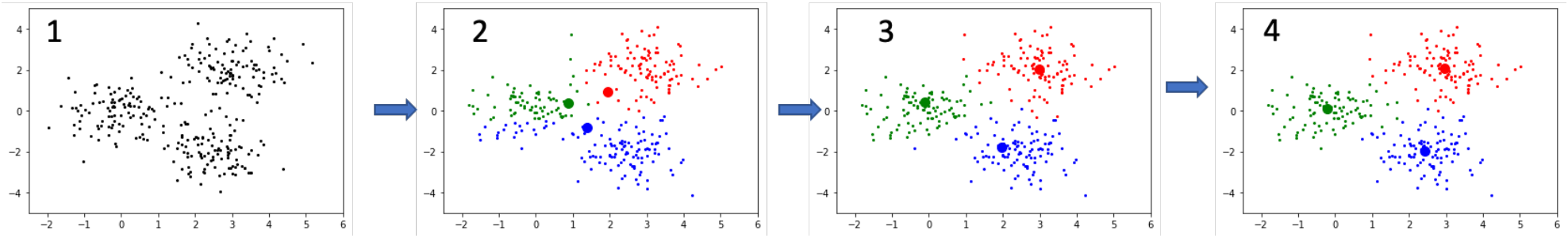
$$d(\mathbf{x}_1, \mathbf{x}_2) = 1 - \cos\theta = 1 - \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{||\mathbf{x}_1|| \cdot ||\mathbf{x}_1||}$$

k-means clustering

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

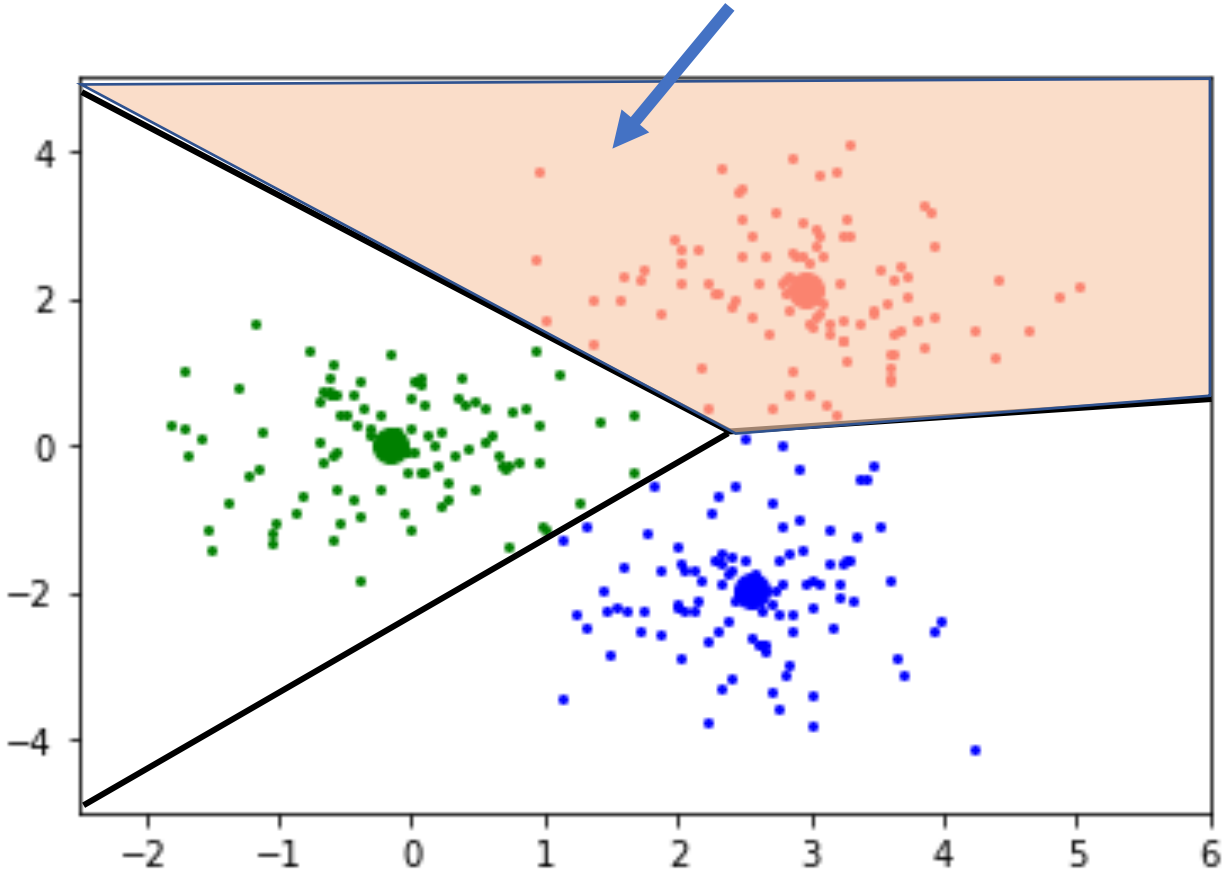
1) Randomly select k cluster centers $c_1, \ldots, c_k \in \mathbf{R}^p$
2) Assign each sample to the nearest cluster center
3) Update cluster centers to mean of samples in cluster
4) Repeat steps 2-3 until convergence

Example: k-means with k=3



Voronoi diagram

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



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 c_i:

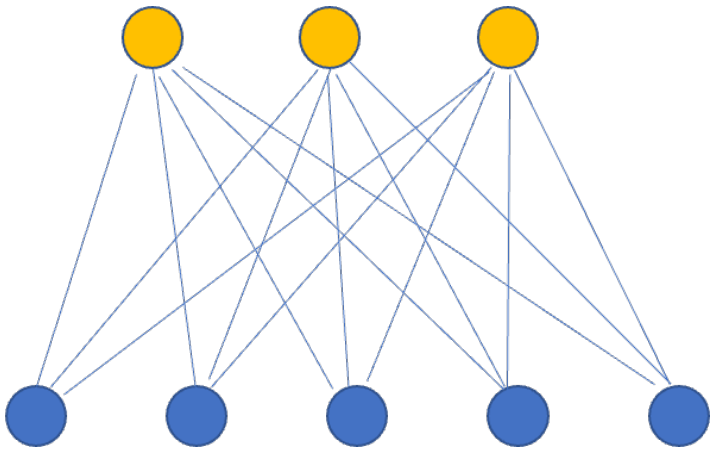
$$d(\boldsymbol{x}_i, \boldsymbol{c}_i) = \Sigma\left(x_{ij} - c_{ij}\right)^2 \qquad \text{(or some other metric)}$$

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

$$\boldsymbol{c}_i^{new} = \frac{1}{m} \sum \boldsymbol{x}_i \qquad \text{(or median)}$$

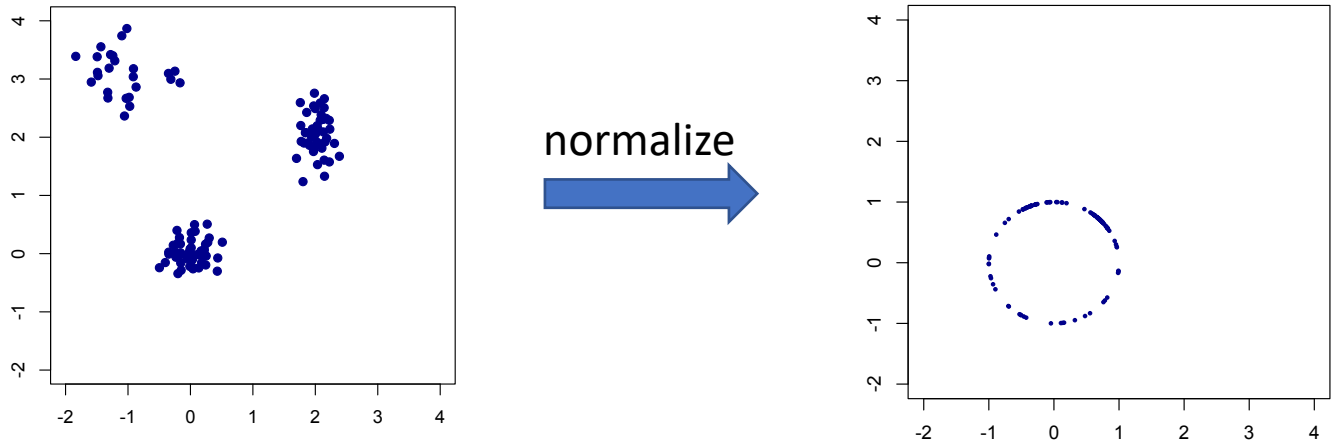
The k-means neural network

- Normalize all samples ($\|x\|$=1 for all input vectors)
- Computing output values for input x:
  - Calculate $h_k = x \cdot w_k$ for all output nodes
  - Set activation value to 1 for output node with maximal h_k and set activation to 0 for all other output nodes
- Learning rule: $\Delta w_{ij} = \eta(x_j - w_{ij})$



k-means ≠ k-means neural network

A: The normalization in k-means neural network means we lose information (only angular distance is preserved):



B: The learning rule in k-means neural network is on-line (i.e. weights are updated after each sample)

IN3050/IN4050, Lecture 11
Unsupervised Learning

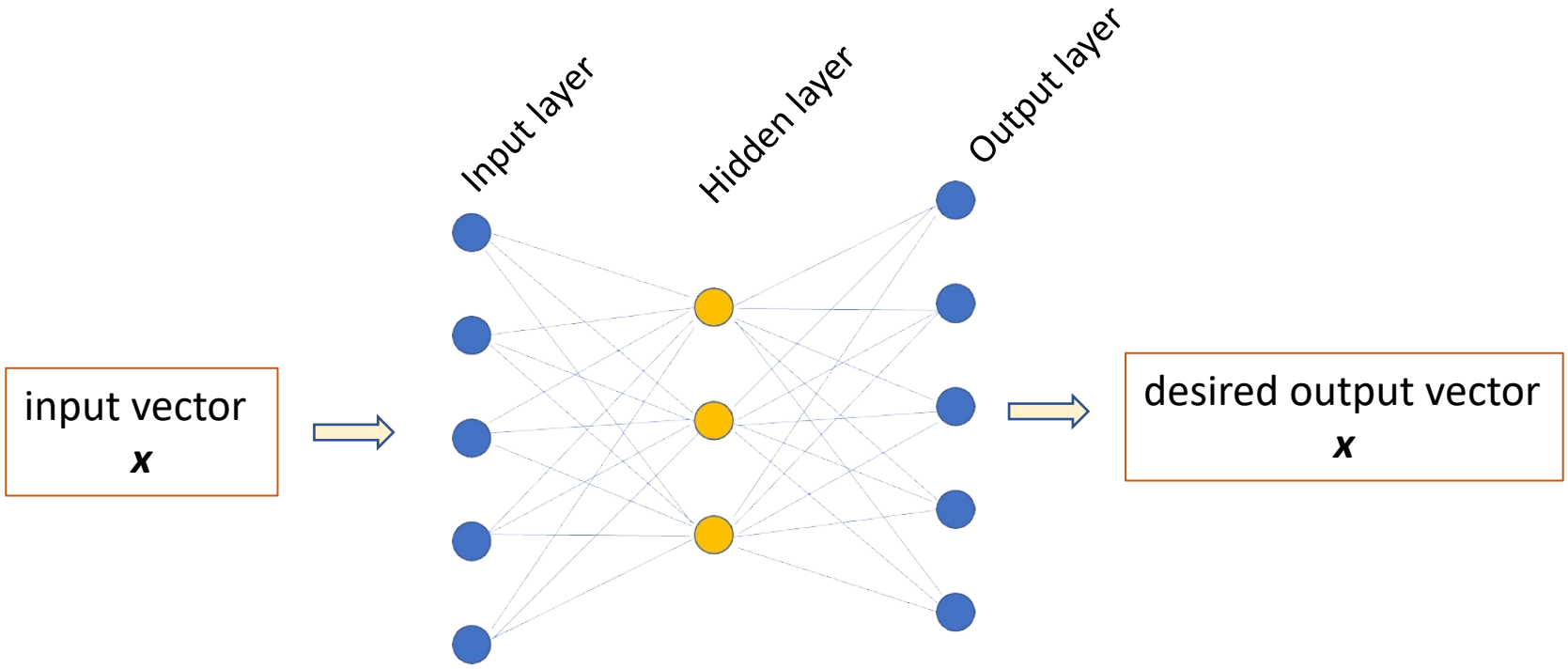
4: Learning how to compress data

Ole Christian Lingjærde



Autoencoders

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



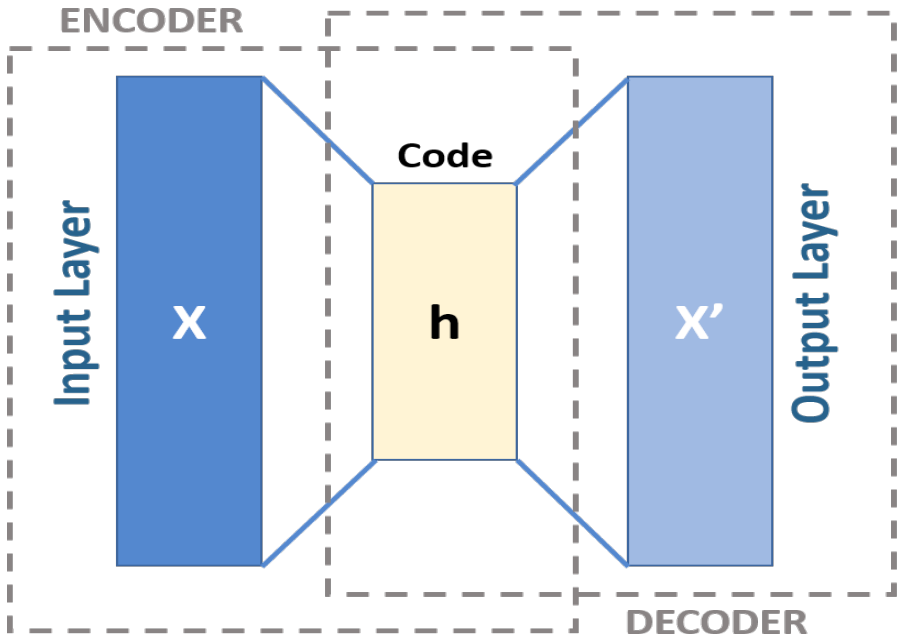
Autoencoders: loss function

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

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

Autoencoders learn to compress

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

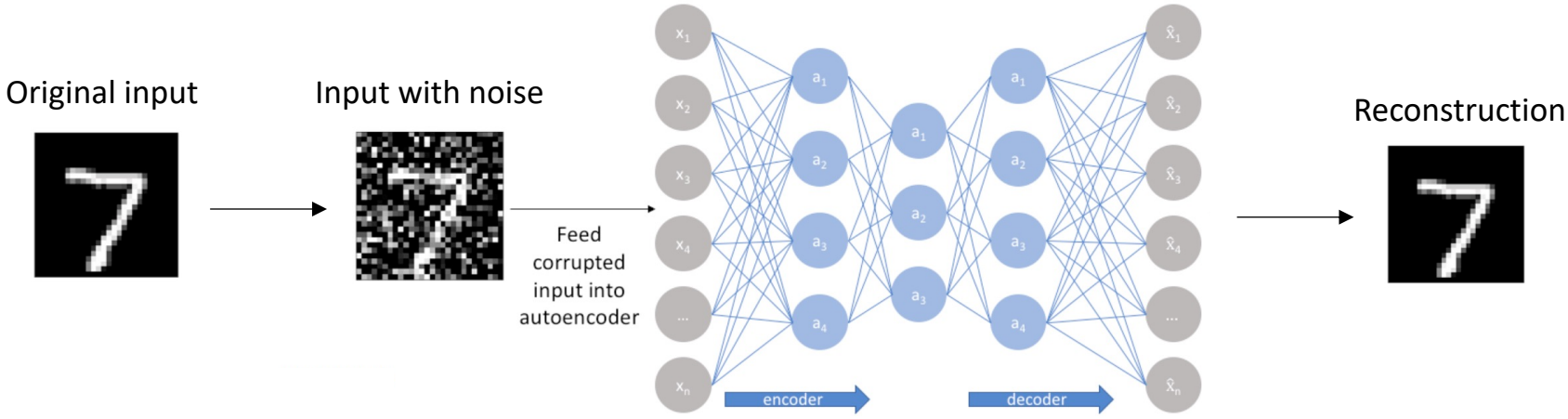


By Michela Massi - Own work, CC BY-SA 4.0

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

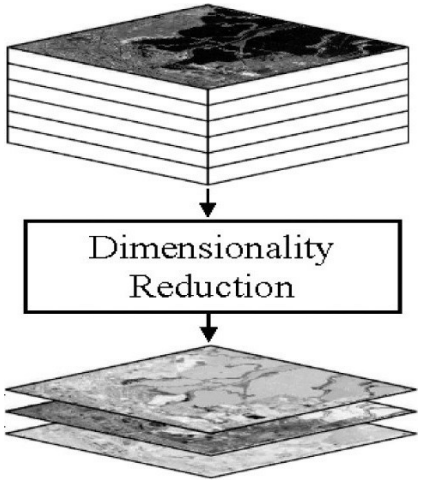


Pictures generated by a variational autoencoder (https://github.com/wojciechmo/vae)

IN3050/IN4050, Lecture 11
Unsupervised Learning

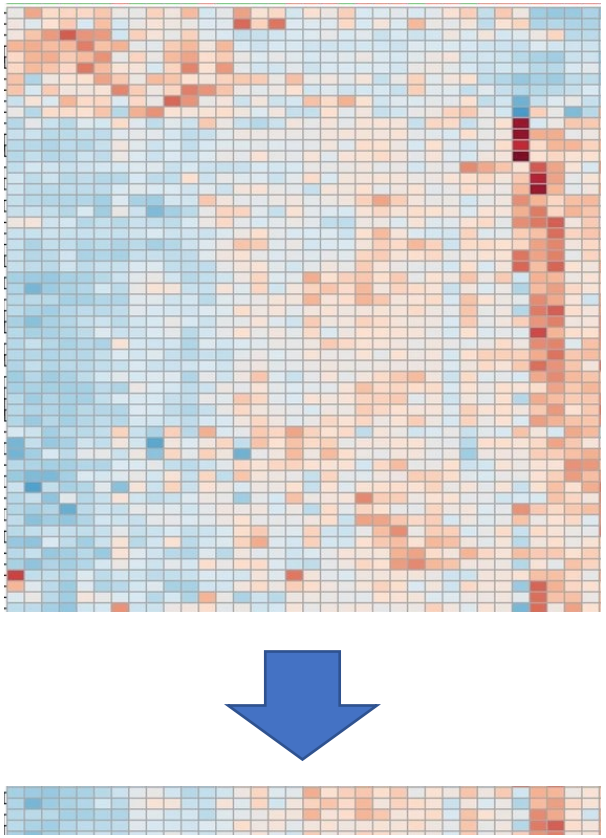
5: Learning how to represent data in low dimension

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)

1) Start with a data set:

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

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 $Y\boldsymbol{w}$:

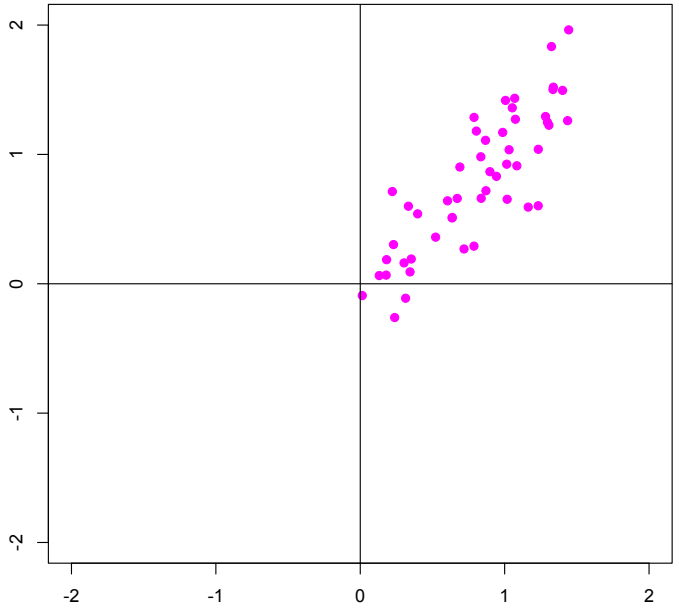
$$\hat{\boldsymbol{w}} = \underset{\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \ \|Y\boldsymbol{w}\|$$

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

PCA: visual interpretation

1) Start with a data set:

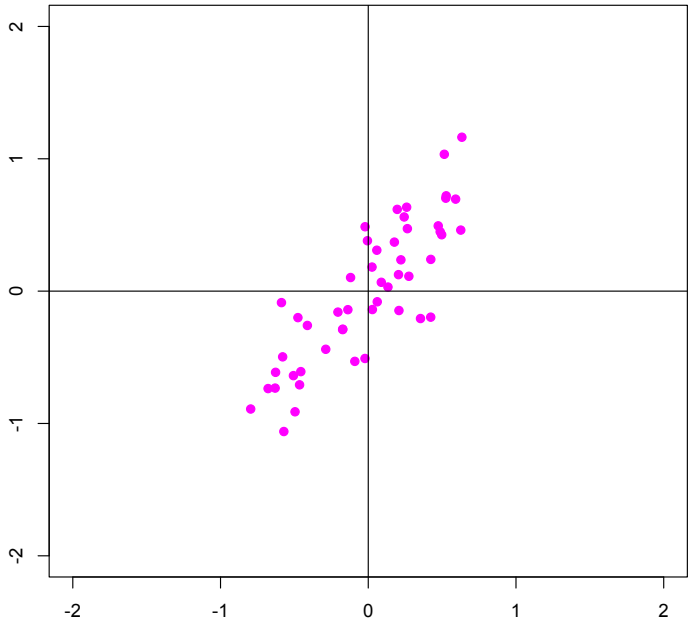
$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{bmatrix} \begin{matrix} \text{sample 1} \\ \text{sample 2} \\ \text{.....} \\ \text{sample n} \end{matrix}$$



PCA: visual interpretation

2) Center the data:

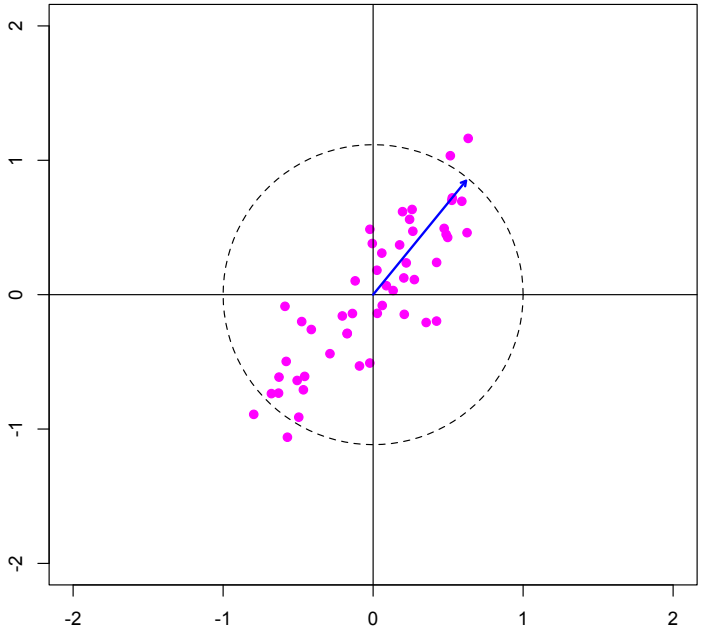
$$Y = \begin{bmatrix} x_{11} - m_1 & x_{12} - m_2 \\ x_{21} - m_1 & x_{22} - m_2 \\ \vdots & \vdots \\ x_{n1} - m_1 & x_{n2} - m_2 \end{bmatrix}$$



PCA: visual interpretation

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

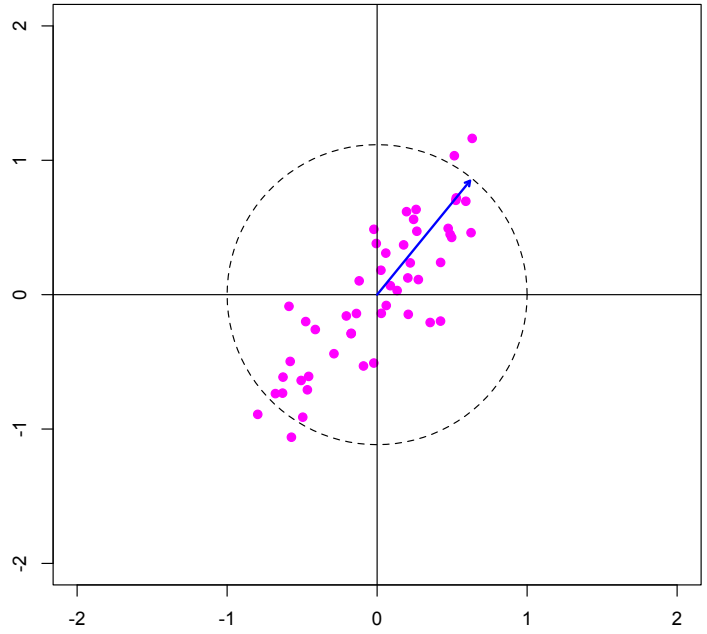
$$\widehat{\boldsymbol{w}} = \underset{\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \|\boldsymbol{Yw}\|$$



PCA: visual interpretation

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

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



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.