Unsupervised Learning

Fabio Massimo Zennaro
fabiomz@ifi.uio.no

University of Oslo

INF3050/INF4050
2020

1. Contents

Outline

2. Introduction

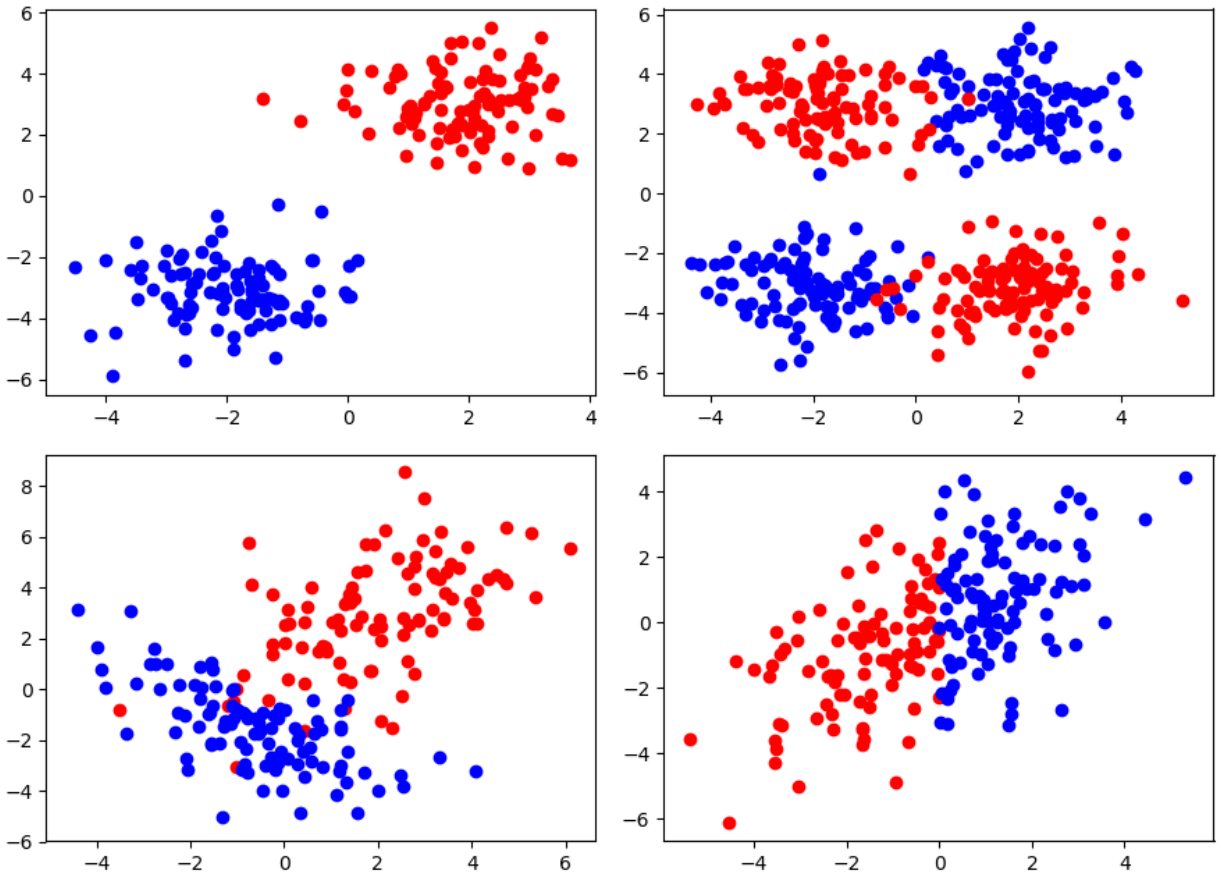
2.1. Review of Supervised Learning

The three domains of machine learning

Machine learning is traditionally divided in three main areas/problems:

- *Supervised learning*: learn with a direction
- *Unsupervised learning*: learn without direction
- *Reinforcement learning*: learn interacting within an environment

Review of supervised learning



Review of supervised learning

In SL we are given a *data matrix* ($\boldsymbol{X}$) and a *label vector* ($\boldsymbol{y}$):

$$\boldsymbol{X} = \begin{bmatrix} 8 & 13 & 4 & 7 & 10 & 12 \\ 15 & 5 & 3 & 12 & 4 & 5 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 4 & 13 & 2 & 3 & 4 & 5 \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ \dots \\ \dots \\ \dots \\ 1 \end{bmatrix}$$

Rows of $\boldsymbol{X}$ are *samples* or *observations*:

$$\boldsymbol{x}_1 = \begin{bmatrix} 8 & 13 & 4 & 7 & 10 & 12 \end{bmatrix}$$

Columns of $\boldsymbol{X}$ are *features* or *descriptors*.

Review of supervised learning

We want to connect samples to their respective label:

$$\boldsymbol{X} = \begin{bmatrix} 8 & 13 & 4 & 7 & 10 & 12 \\ 15 & 5 & 3 & 12 & 4 & 5 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 4 & 13 & 2 & 3 & 4 & 5 \end{bmatrix} \begin{matrix} \longrightarrow \\ \longrightarrow \\ \dots \\ \dots \\ \dots \\ \longrightarrow \end{matrix} \begin{bmatrix} 1 \\ 0 \\ \dots \\ \dots \\ \dots \\ 1 \end{bmatrix} = \boldsymbol{y}$$

Values of $\boldsymbol{y}$ may be categorical ($\Rightarrow$ *classification*) or continuous ($\Rightarrow$ *regression*).

Review of supervised learning

We want to learn a *function* from samples to labels

$$f : \mathbb{X} \to \mathbb{Y}$$

that is, a function that for any sample gives us a label:

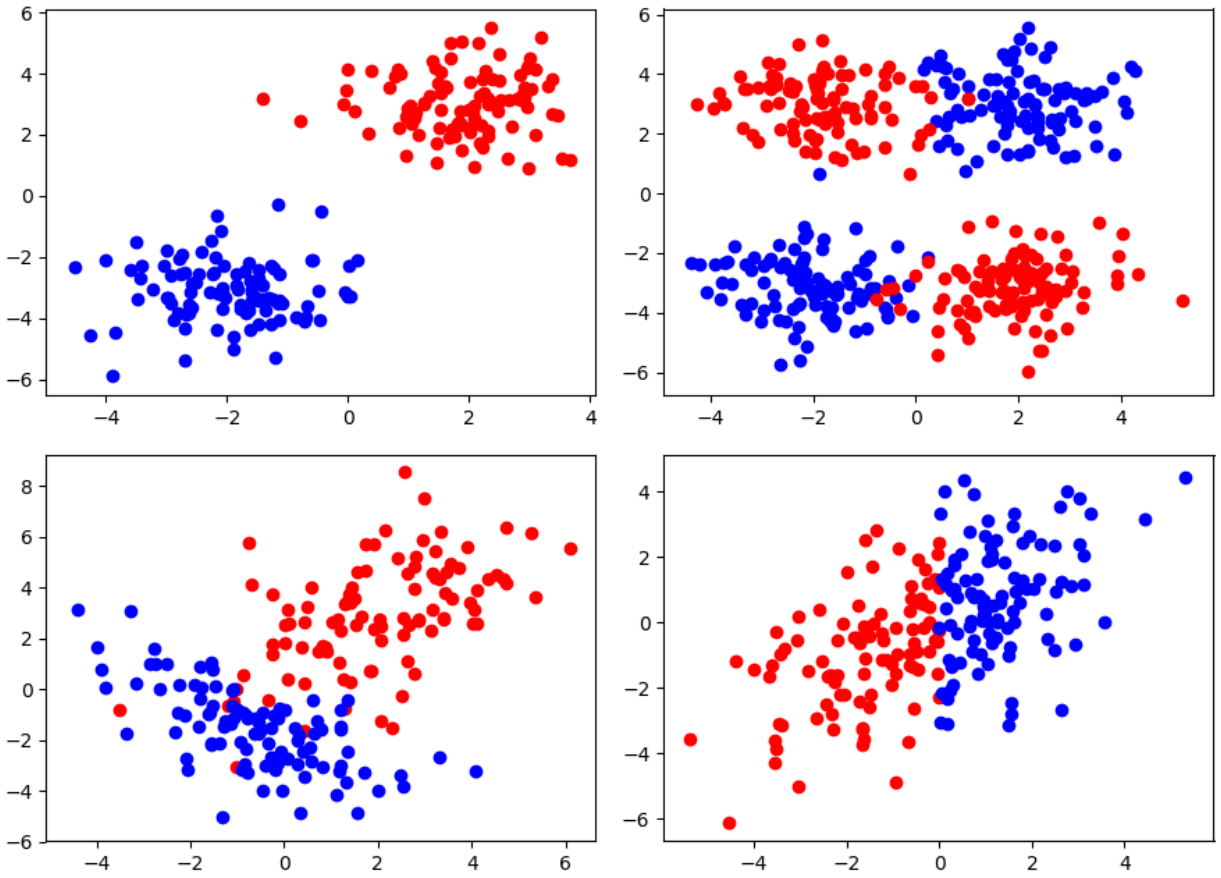
$$y_i = f(\boldsymbol{x}_i)$$

We use *machine learning algorithms* (linear regression, neural networks, SVMs) to learn a *generalizing* function f.

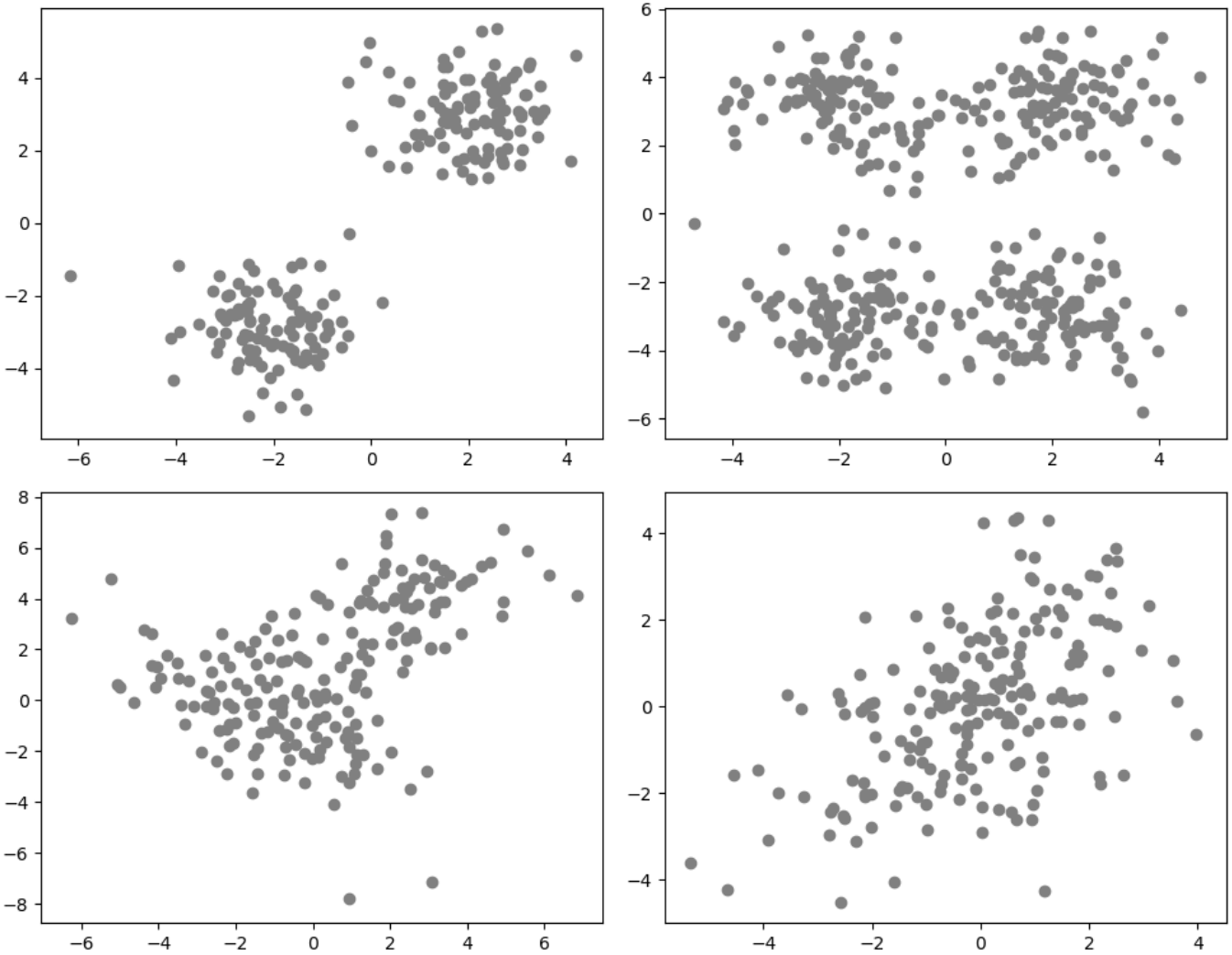
The main problem in SL is **HOW** to learn?

2.2. Unsupervised Learning

Review of supervised learning



Going unsupervised



Unsupervised learning

In UL we are given only a *data matrix* ($\boldsymbol{X}$):

$$\boldsymbol{X} = \begin{bmatrix} 8 & 13 & 4 & 7 & 10 & 12 \\ 15 & 5 & 3 & 12 & 4 & 5 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 4 & 13 & 2 & 3 & 4 & 5 \end{bmatrix}$$

No explicit label is provided (no $\boldsymbol{y}$).

Where are the labels?

Why are the labels missing?

- Labelling data is *costly*
- Labelling data may be *unreliable*
- Labelling data may be *impossible*

And **real** learning (i.e.: humans) happens in an unsupervised way!
(very little supervision was provided when you were learning as an infant!)

Unsupervised learning

Where are we mapping this *data matrix*?

$$\boldsymbol{X} = \begin{bmatrix} 8 & 13 & 4 & 7 & 10 & 12 \\ 15 & 5 & 3 & 12 & 4 & 5 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 4 & 13 & 2 & 3 & 4 & 5 \end{bmatrix} \begin{matrix} \longrightarrow \\ \longrightarrow \\ \dots \\ \dots \\ \dots \\ \longrightarrow \end{matrix} \begin{bmatrix} ? \\ ? \\ \dots \\ \dots \\ \dots \\ ? \end{bmatrix}$$

Review of supervised learning

What function do we learn?

$$f : \mathbb{X} \rightarrow ?$$

How do we compute outputs for the samples?

$$? = f(\boldsymbol{x}_i)$$

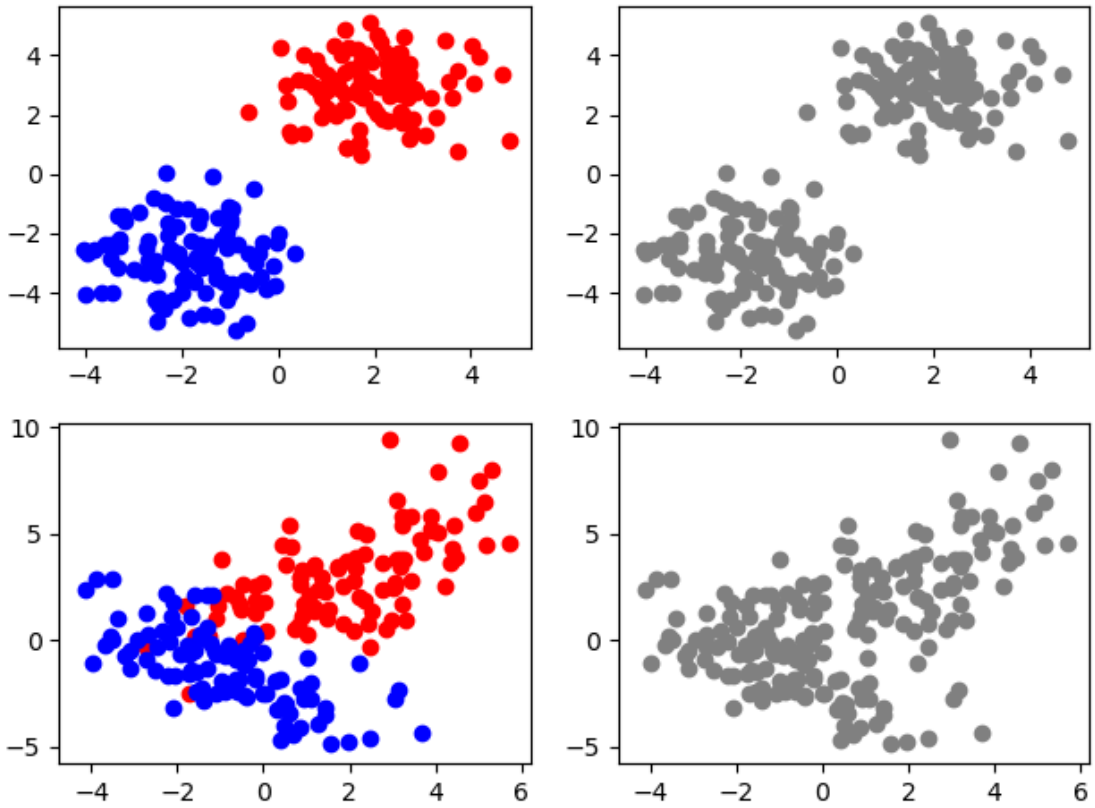
The problems in UL are (i) **WHAT** to learn? and (ii) **HOW** to learn?

3. Theory

Supervised vs unsupervised learning

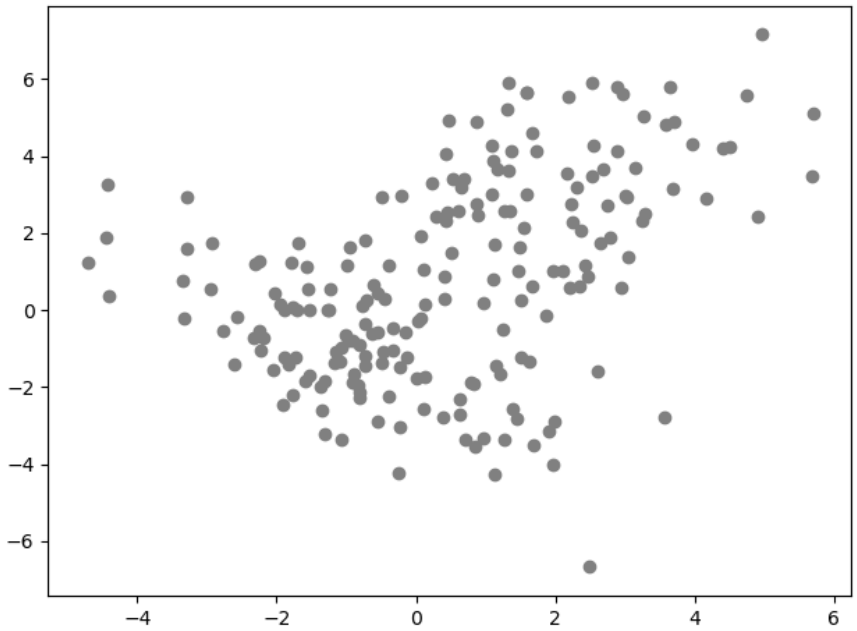
Supervised: $\mathbb{R}^2 \to \{0, 1\}$

Unsupervised: $\mathbb{R}^2 \to ?$



How to perform unsupervised learning?

In general, we can not solve the unsupervised learning problem without making some **assumptions**.



1. *What do we want to learn here?*
2. *What does it matter here?*

3.1. Representations

The concept of representation

1- What do we want to learn?

We try to learn new **representation** of the data:

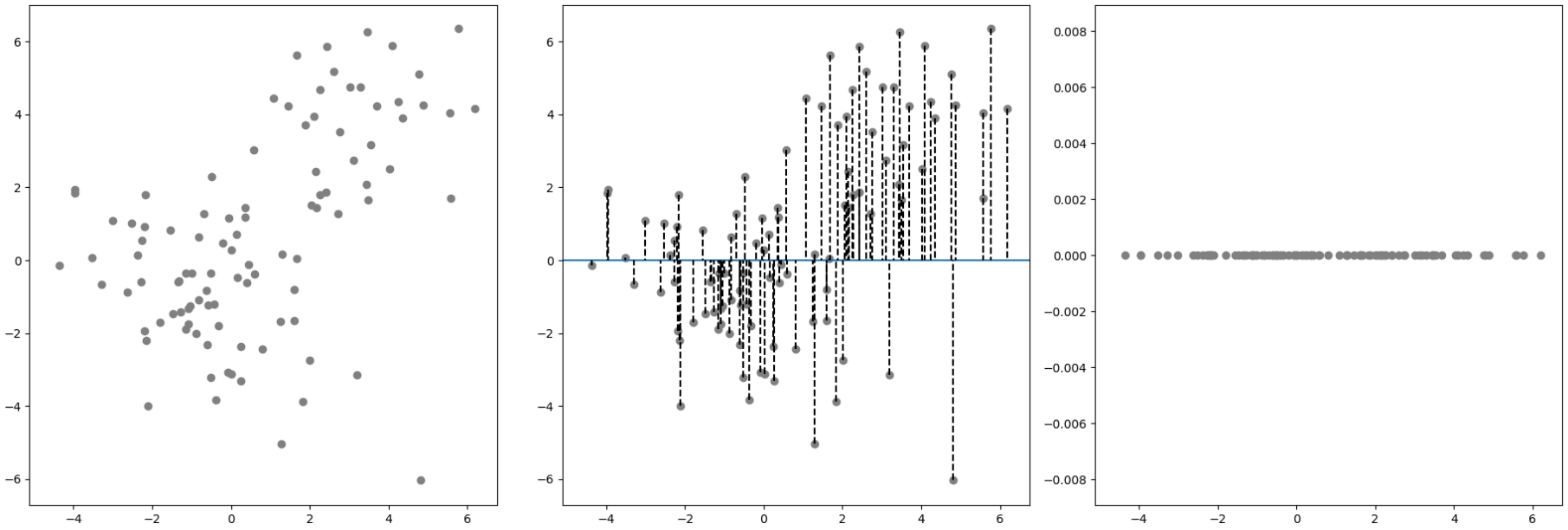
$$\mathbb{X} \rightarrow \mathbb{Z}$$

$$\mathbb{R}^n \rightarrow \mathbb{R}^m$$

Representations $\mathbb{Z}$ are often called *learned representations*, *intermediate representation* (in deep pipelines), *latent representations* (in generative models).

Rigorous formalization of this concept: *mapping between continuous/discrete spaces.*

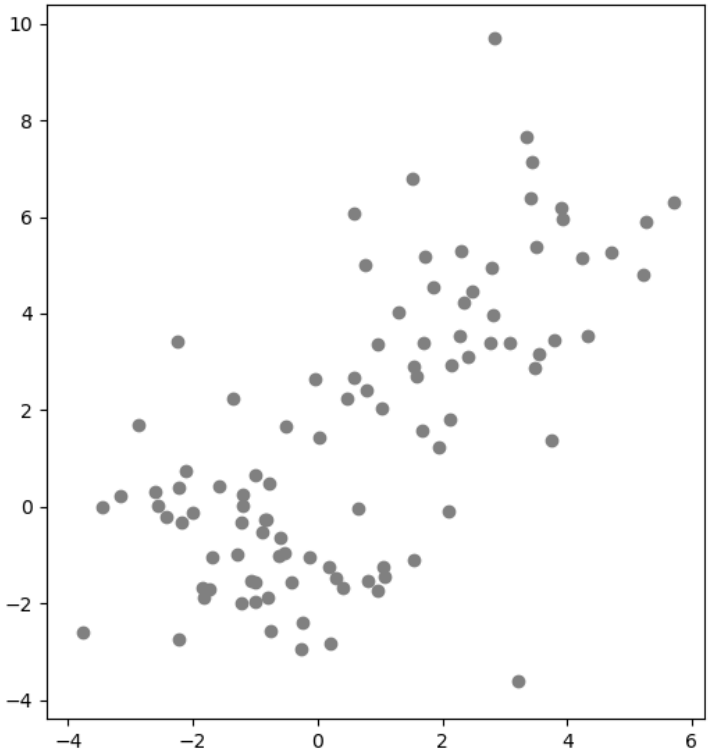
Examples of learning representations

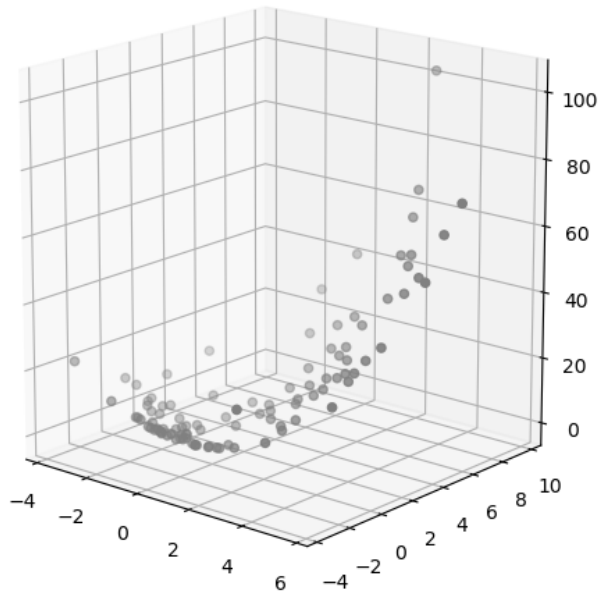


$$(x, y) \mapsto (x, 0)$$

$$\mathbb{R}^2 \to \mathbb{R}^1$$

Examples of learning representations

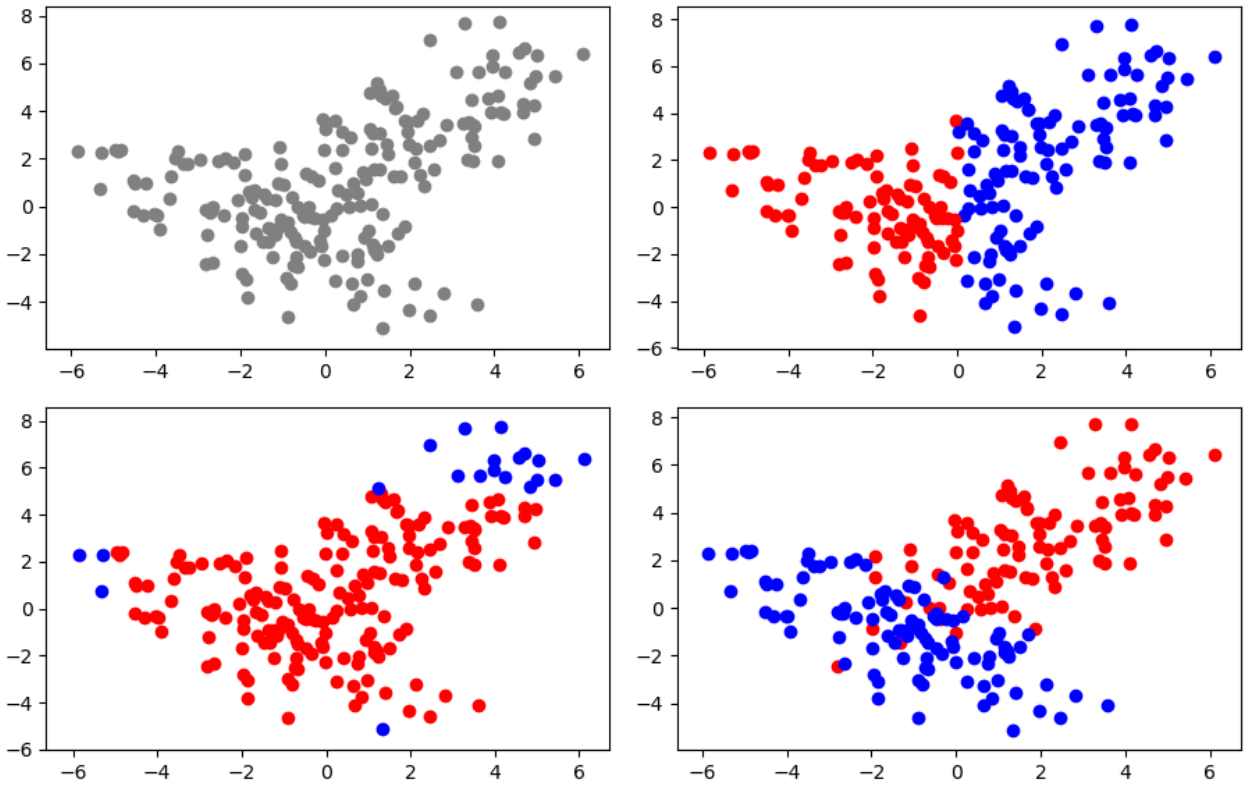




$$(x, y) \mapsto (x, y, x^2 + y^2)$$

$$\mathbb{R}^2 \to \mathbb{R}^3$$

Examples of learning representations



$$(x, y) \mapsto \{red, blue\}$$

$$\mathbb{R}^2 \to \{0, 1\}$$

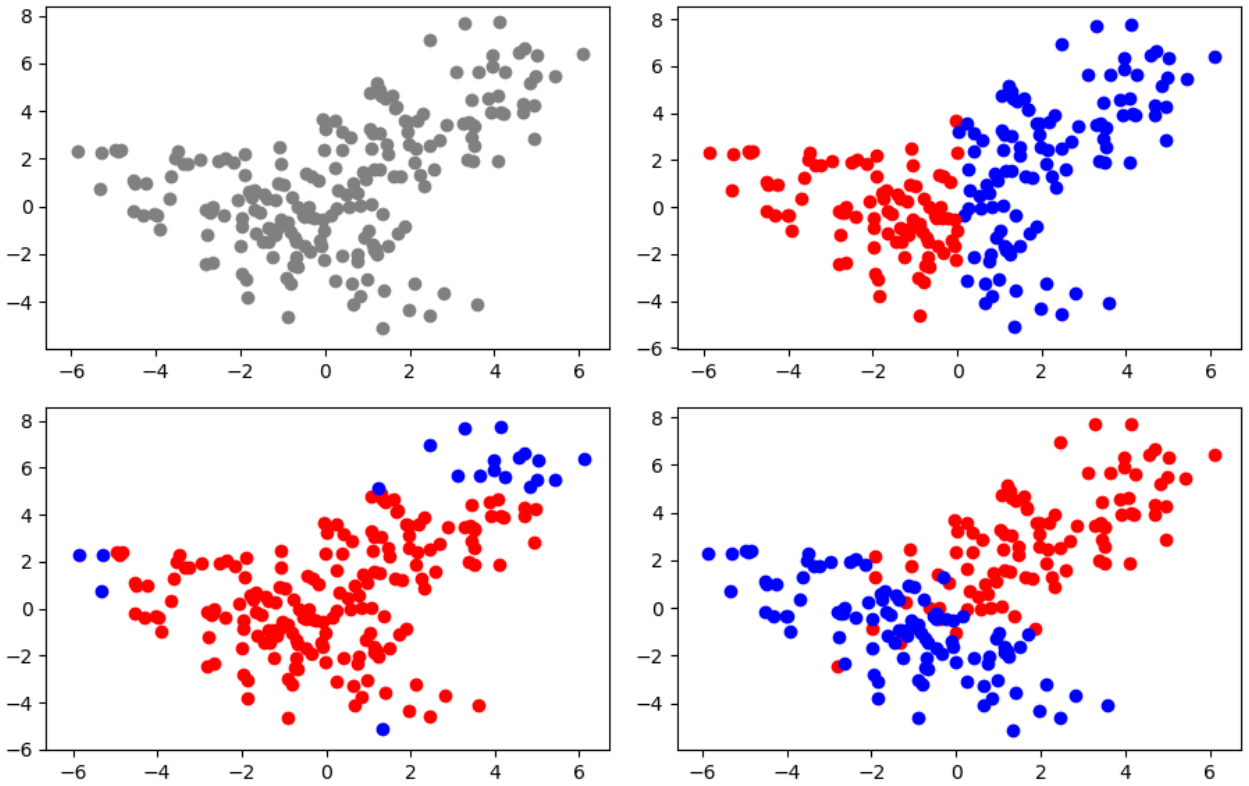
Examples of learning representations

- *Feature selection*.
  A rigid transformation that discards some features.
- *Normalization of the data*.
  A statistical transformation of the data.
- *Any pre-processing of the data* (subsampling/rounding, Fourier transform).
  Pre-processing is often a hard human-defined (not learned) transformation.
- *Intermediate representations in a deep network*.
  Each layer of a deep network is a transformation $\mathbb{R}^n \to \mathbb{R}^m$.
- *Kernels for SVMs*.
  Often treated as implicit representations.

The concept of representation

1- What do we want to learn? We learn *representations*.

This is not enough.



Which mapping is correct?

3.2. Structure

The concept of structure

2- What does matter?

We want to preserve **relevant structure** in the data:

$$\mathbb{X} \rightarrow \mathbb{Z}$$

where:

- $\mathbb{Z}$ preserves **relevant** information useful for *your* objective; relevant information is kept, *noise* is discarded;
- Natural **structure**/organization of the data is preserved; relevant relationships between data points are maintained.

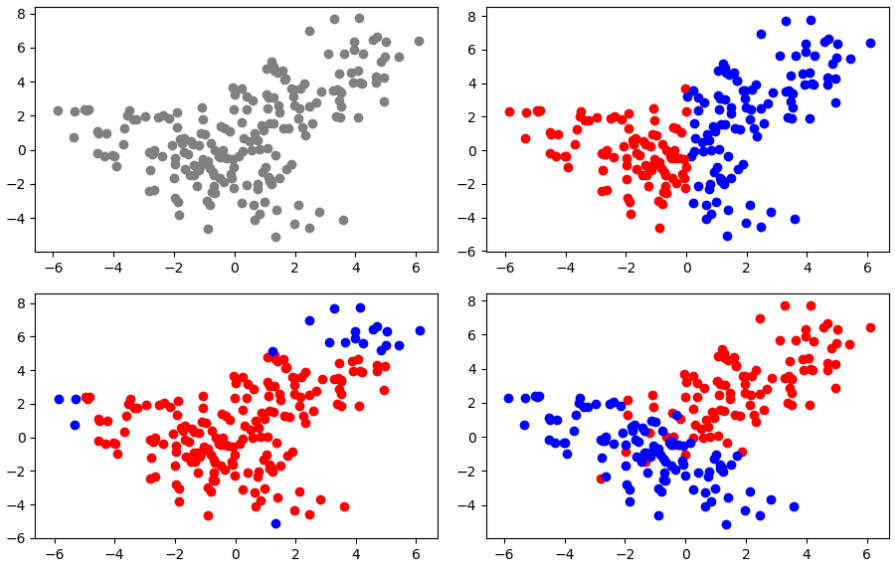
Define *relevant structures* through **assumptions.**

Rigorous formalization of this concept: *metric spaces*; *probability distribution functions*; *information-theoretic measures*.

Examples on assumptions about structure

Some simple and intuitive assumptions about structure:

- *Locality*: points close to each other in the original space are similar; points close to each other in the original space should be mapped to similar representations.
- *Smoothness*: transitions in representations should be smooth.

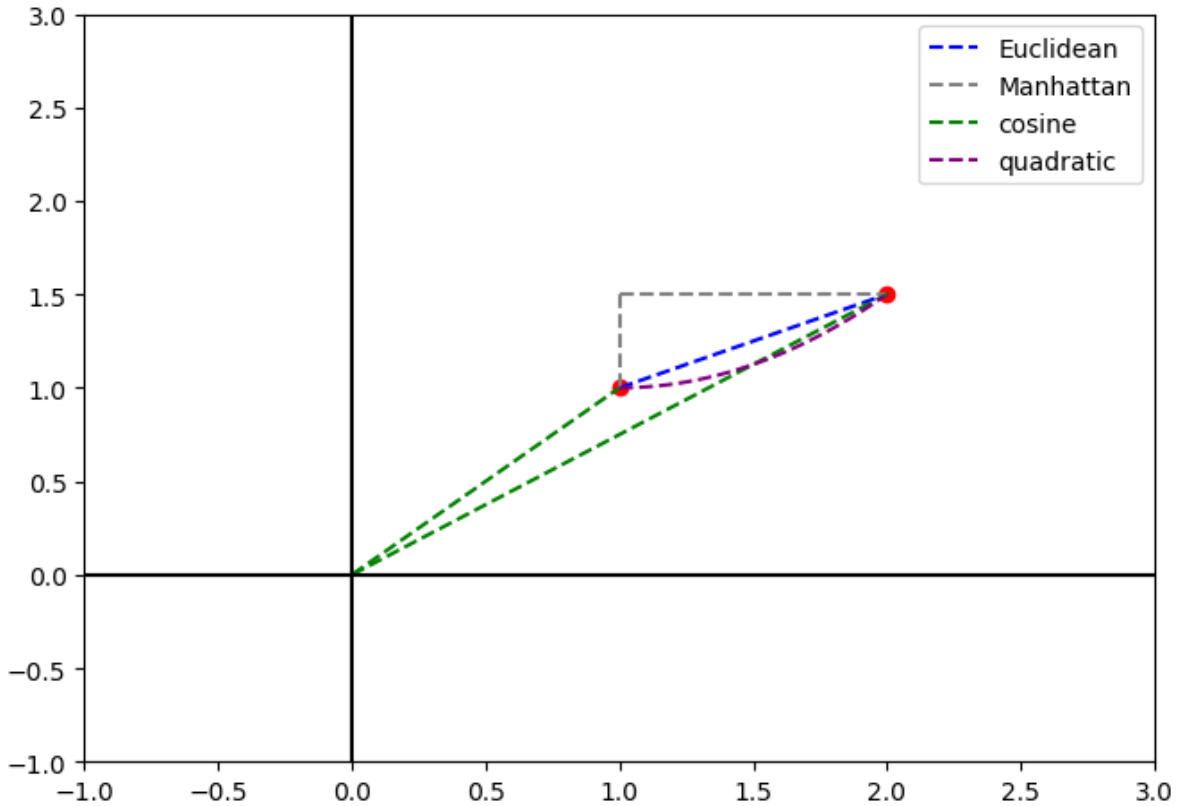


Examples on assumptions about structure

Even simple assumptions may require careful evaluation.

- *Locality*: points close to each other in the original space are similar.

How do we measure closeness?



Structure and representation

In unsupervised learning we try to learn **representations** preserving relevant **structure**.
This requires making **assumptions**.

- If we design a UL algorithm, we need to decide what structure matters;
- If we use existing algorithms, we need to understand what structure they preserve.

Assumptions are strongly related to the **aim** of unsupervised learning.

4. Types of Unsupervised Learning

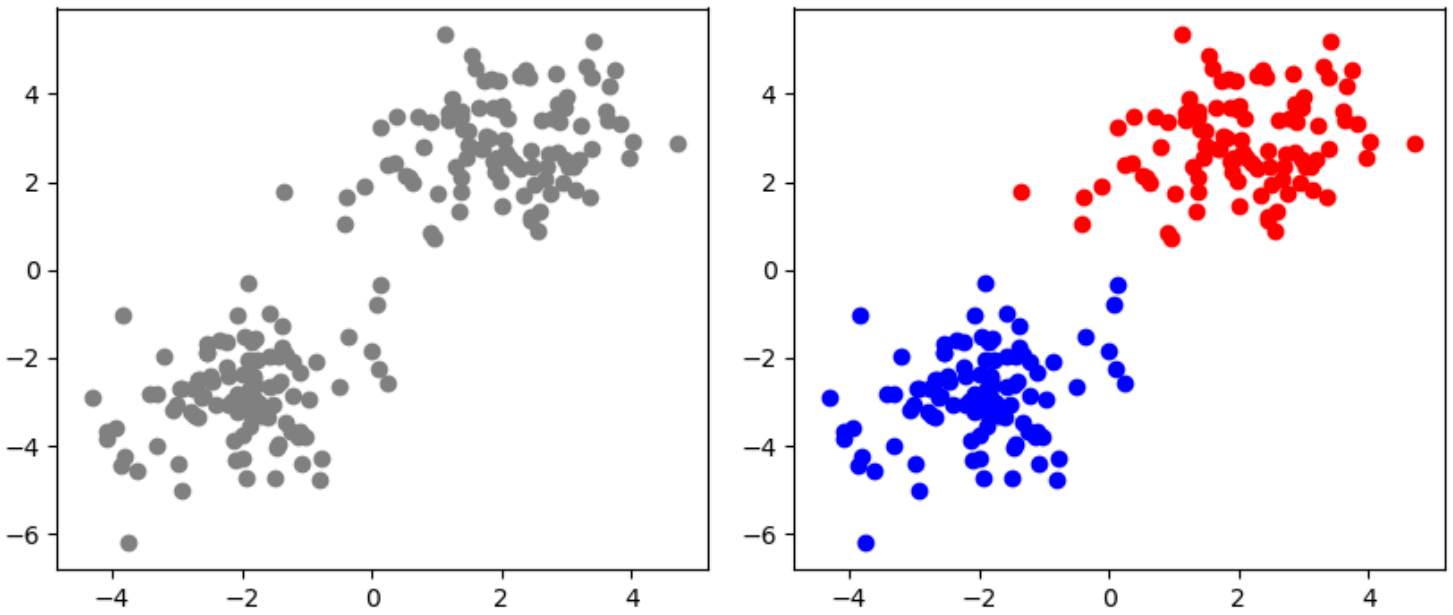
4.1. Clustering

Clustering

Aim: we want to find meaningful groupings of the data.

Representation: typically, a discrete representation.

Structure: a metric that preserves similarities between data points.



Clustering

Clusters resemble hidden labels. Cluster centers are often take to constitute (noiseless) *exemplars* or *prototype* of a class.

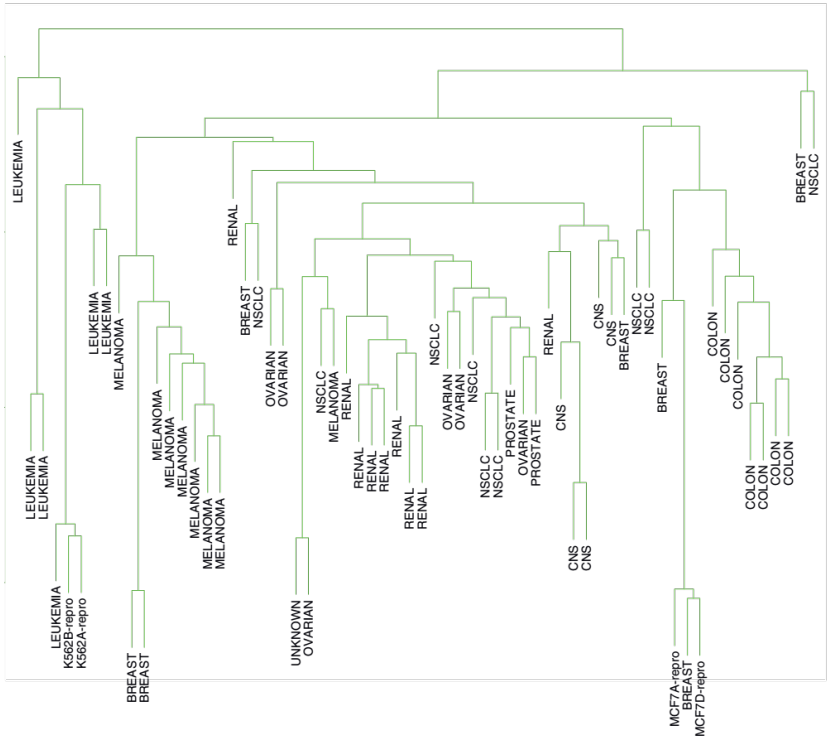


Image from: [2]

Examples: k-means, k-centroids, self-organizing maps

4.2. Dimensionality reduction / visualization

Dimensionality reduction / visualization

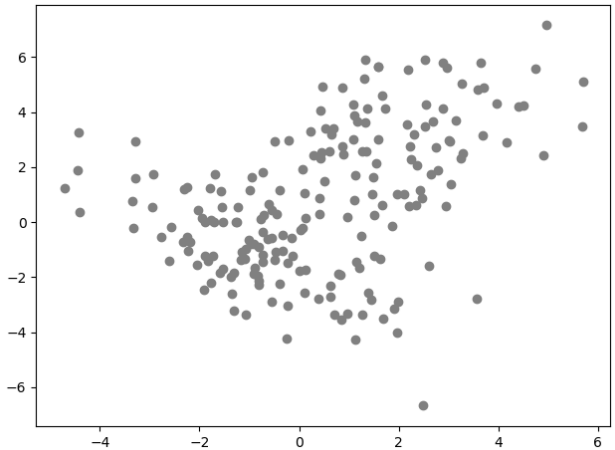
Aim: we want to plot the data for visual inspection.

Representation: typically, a low-dimensional continuous representation in 2D or 3D.

Structure: a metric that preserves similarity between data points.

	F_1	**F_2**	**F_3**	**F_4**	**F_5**
Obs 1	0.3	0.4	0.7	0.4	0.3
Obs 2	0.5	0.5	0.6	0.5	0.4
Obs 3	0.4	0.3	0.5	0.3	0.6
...	...	...	...	...	...
Obs N	0.6	0.8	0.7	0.2	0.7

$\rightarrow$



$$\mathbb{R}^5 \rightarrow \mathbb{R}^2$$

Dimensionality reduction / visualization

Dimensionality reduction is often used as an *exploratory* approach to the data. Different metrics and similarity may be used in order to probe the data.

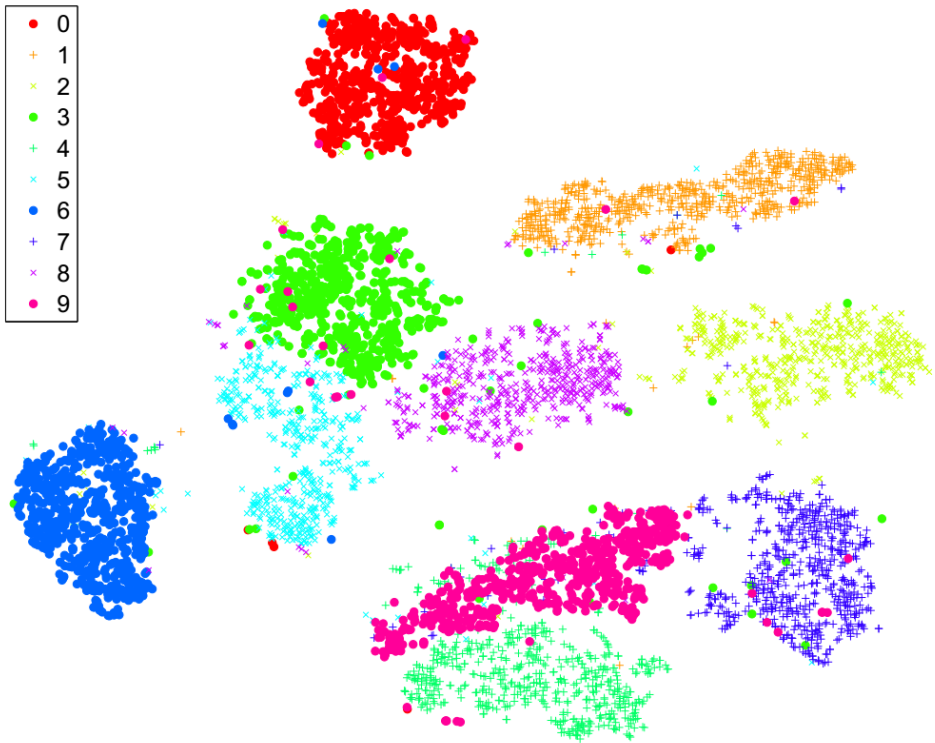


Image from: [4]

Examples: PCA, t-SNE, UMAP

Dimensionality reduction / visualization

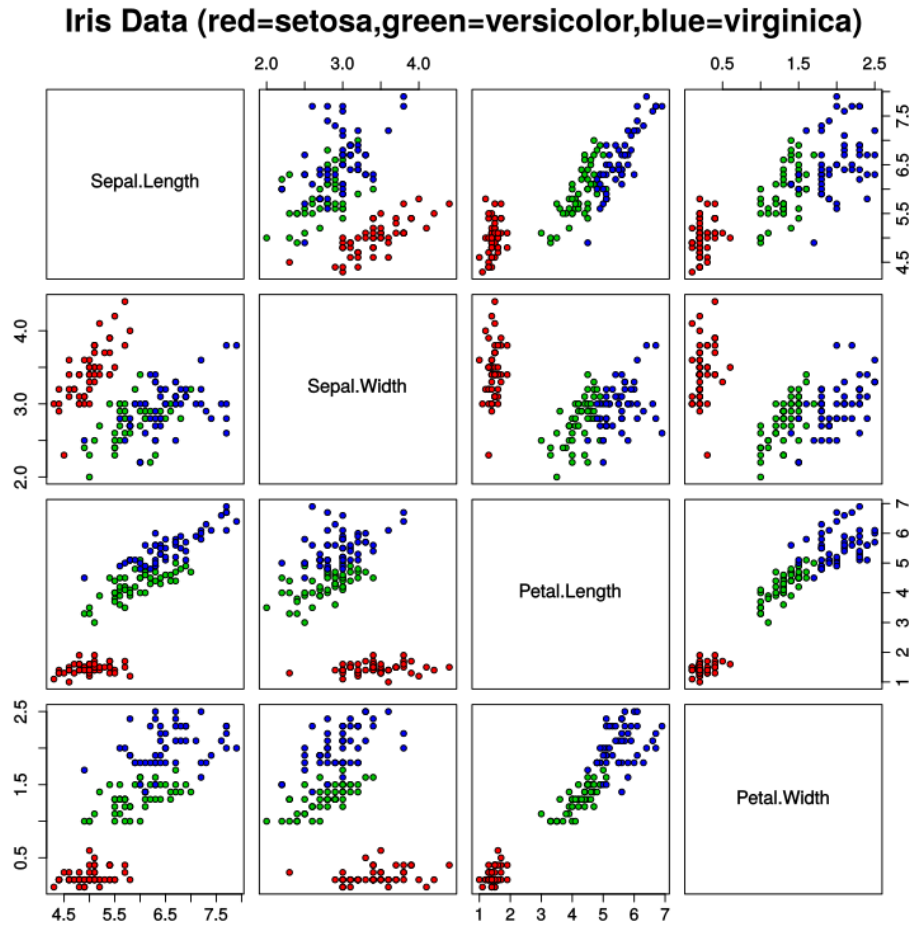


Image from: Wikipedia

4.3. Dimensionality reduction / manifold learning

Dimensionality reduction / manifold learning

Aim: we want to discover lower dimensional planes on which the relevant structure lies.

Representation: typically, a lower-dimensional continuous representation.

Structure: the manifold on which the data lie.

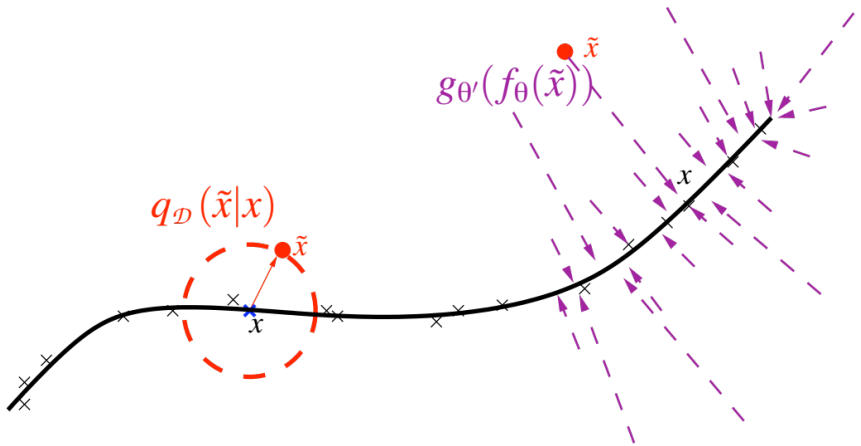


Image from: [7]

Dimensionality reduction / manifold learning

Manifold learning often used as a way to discover the *intrinsic* dimensionality of the data. Discarded dimensions are often associated with noise.

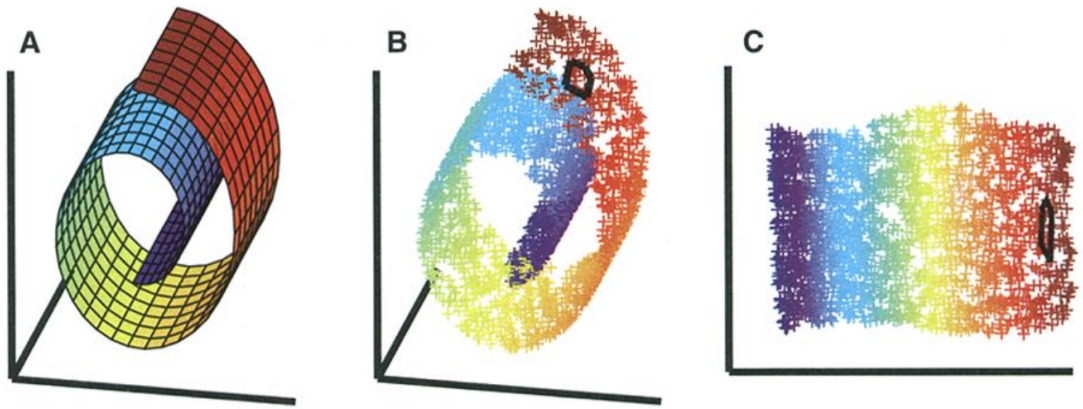


Image from: [5]

Examples: denoising autoencoders, local linear embedding, multi-dimensional scaling.

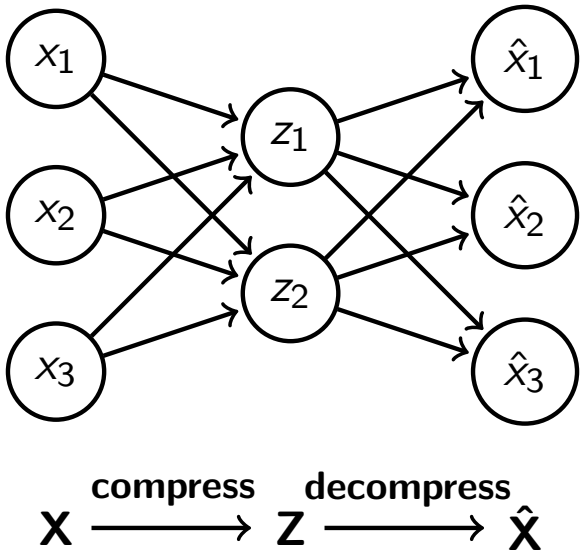
4.4. Dimensionality Reduction / compression

Dimensionality Reduction / compression

Aim: we want to find reduce the dimensionality of the data.

Representation: typically, a lower-dimensional continuous representation that allows the reconstruction of the original data.

Structure: relevant information contained in the original data.



Dimensionality Reduction / compression

Compression is a more *signal-theoretic* or *information-theoretic* methods that sees representations as an *encoding* of the original data. Representations are often expected to be *decodable* back in the original data.

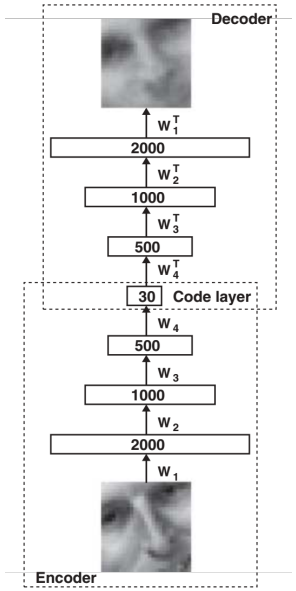


Image from: [3]

Examples: autoencoders, denoising autoencoders, restricted Boltzmann machines, information bottleneck.

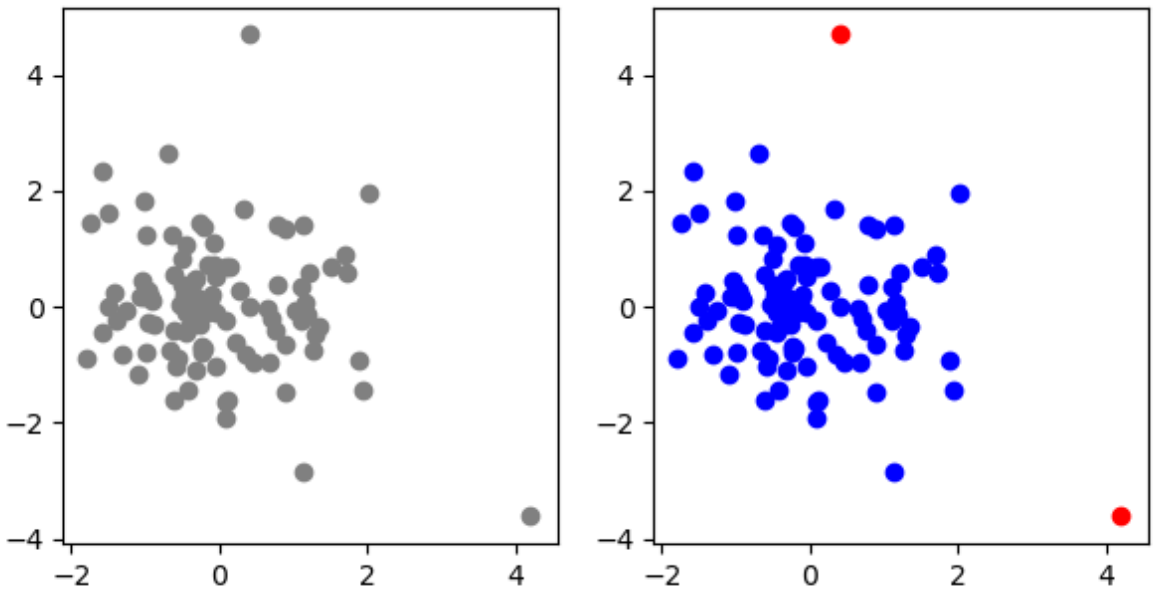
4.5. Anomaly detection

Anomaly detection

Aim: we want to detect outliers in the data.

Representation: typically, a binary representation.

Structure: a suitable metric that allows to filter out outliers.



Anomaly detection

Anomaly detection is a sort of binary classification aimed at raising an alert when non-conforming data are detected.

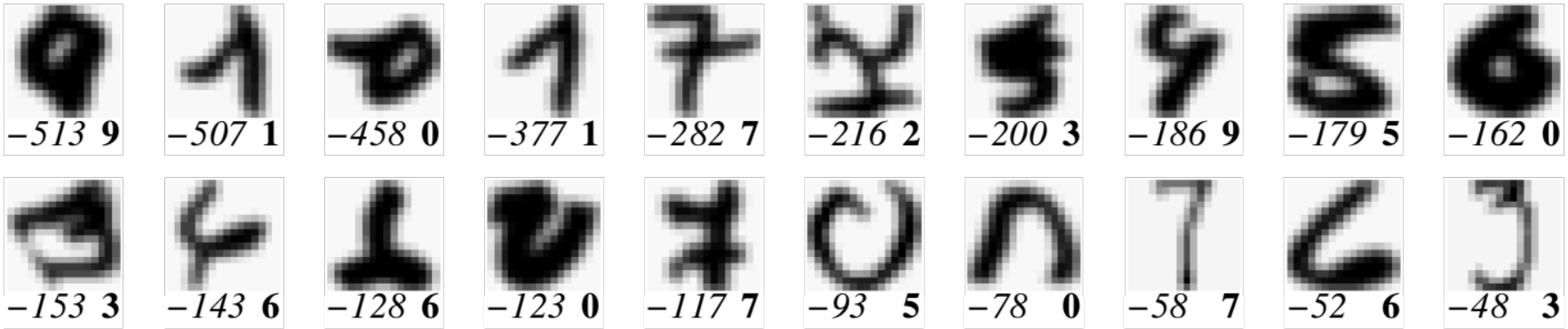


Image from: [6]

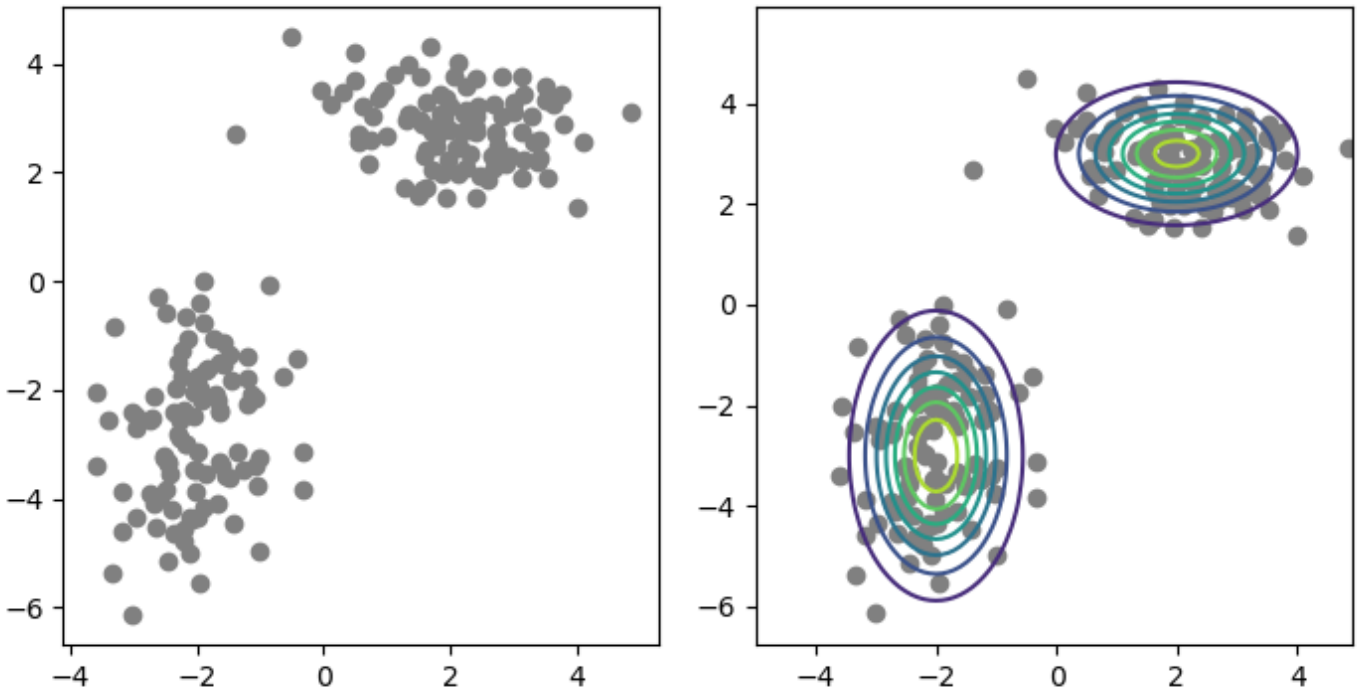
4.6. Generative models

Generative models

Aim: we want to reconstruct the model that generated the data we observed.

Representation: typically, a statistical parametric model that may have generated the data.

Structure: the data themselves we observed.



Generative models

Generative modeling is a more refined approach that tries to explain the data we observed by modelling the mechanism that generated the data.

Image from: [1]

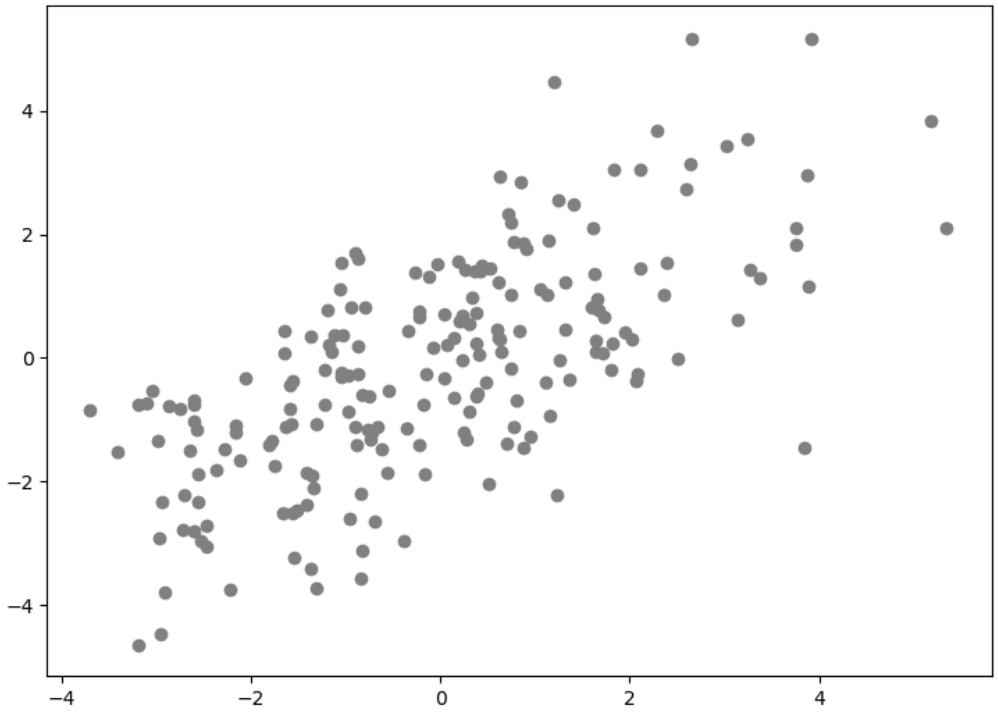
Examples: Gaussian mixture models, Boltzmann machines, generative adversarial networks.

5. Algorithms

5.1. PCA

PCA: Intuition

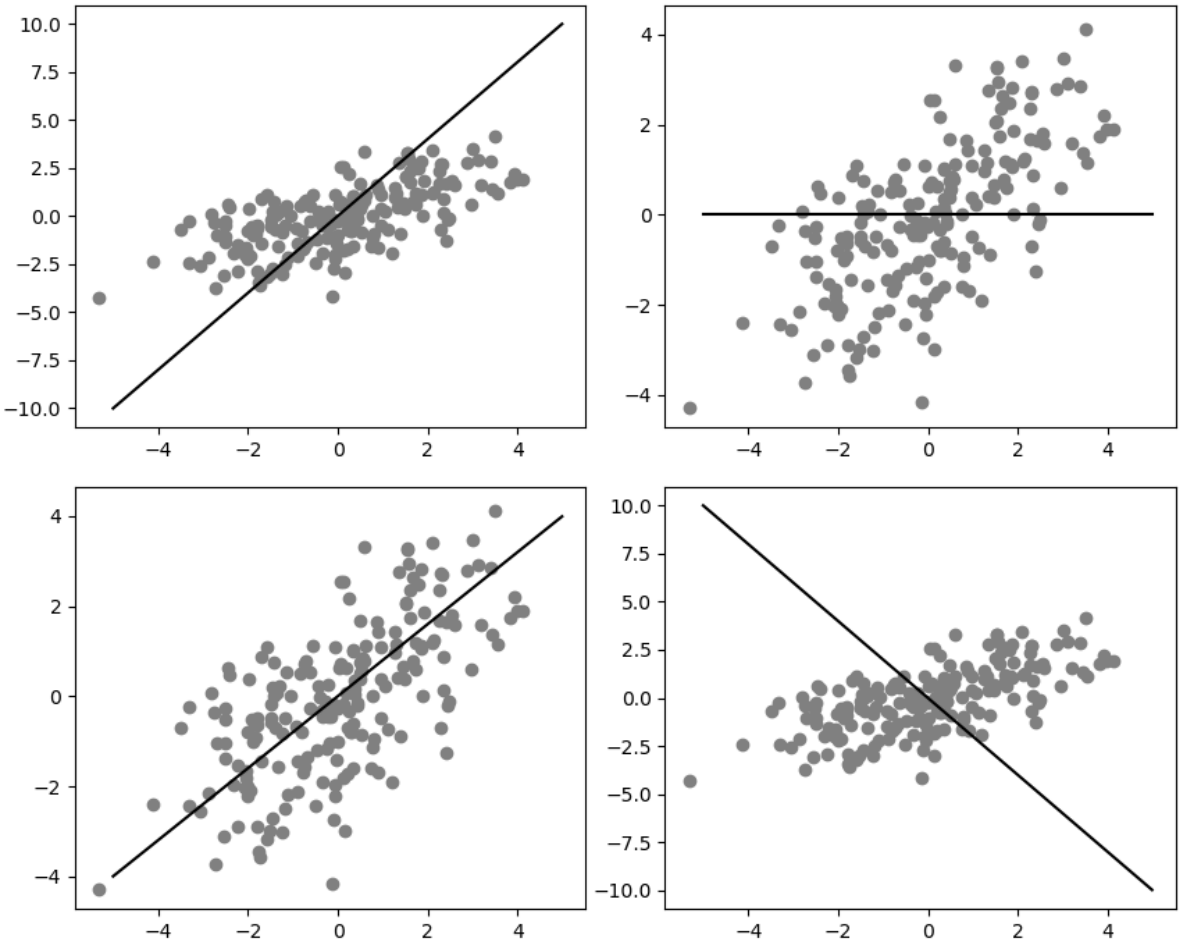
Principal Component Analysis (PCA) is an unsupervised learning technique for *dimensionality reduction* and *compression*.



(Also known as: discrete Karhunen-Loeve transform, Hotelling transform)

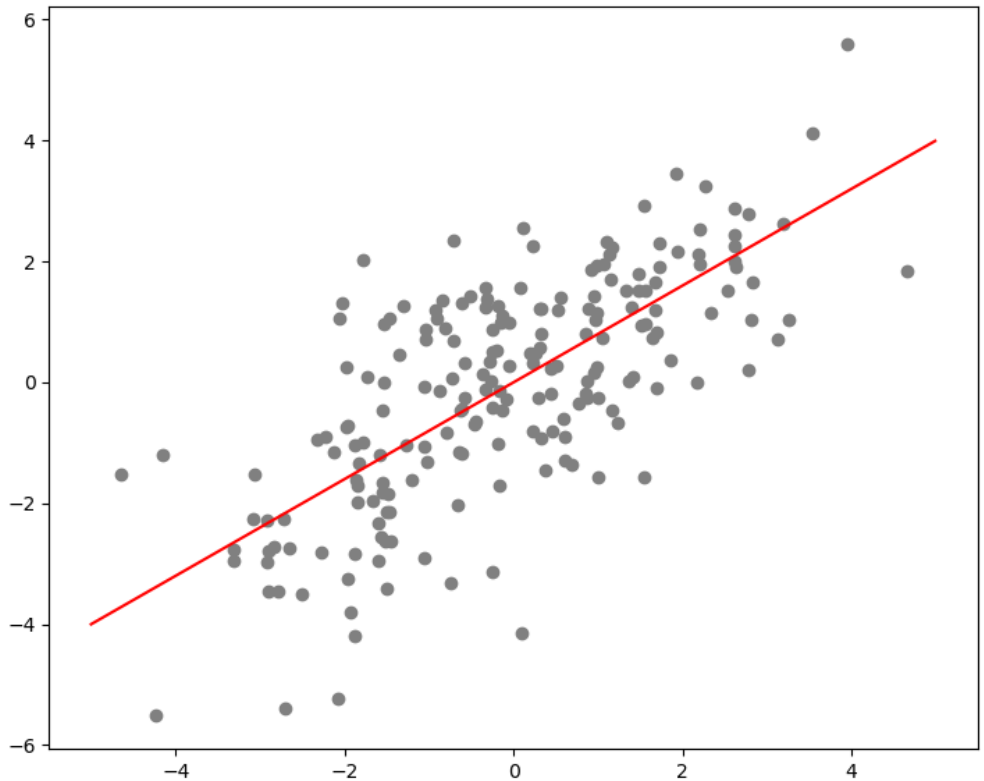
PCA: Intuition

If we were to preserve only one dimensions which one would we choose?



PCA: Intuition

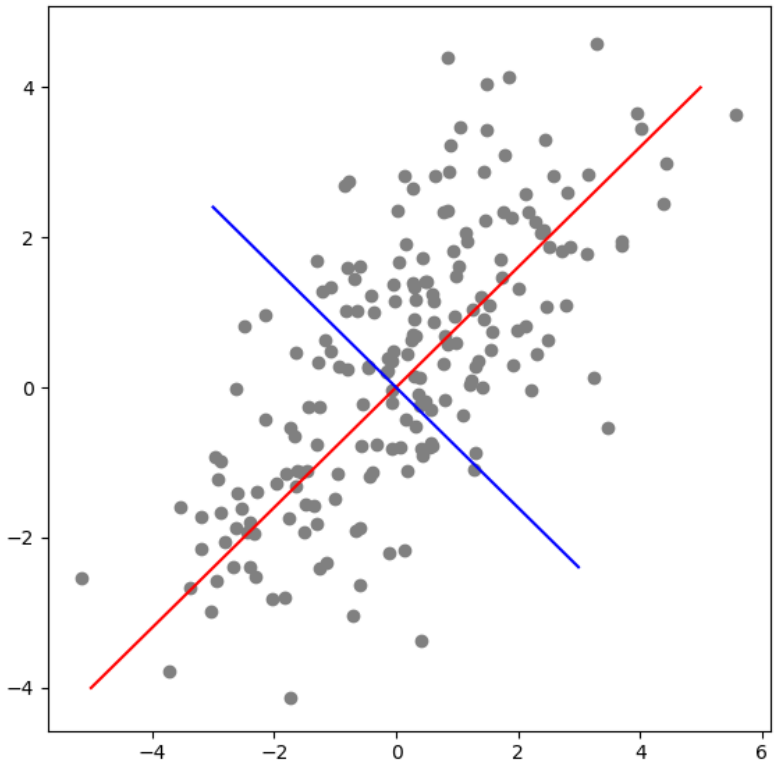
PCA selects that dimension along which the data spread the most.



(Formally, PCA solves a square minimization optimization error.)

PCA: Intuition

Further dimensions are chosen to be perpendicular to the one already selected.



(Formally, PCA chooses a new set of basis for our space.)

PCA: Algorithm

PCA tries to learn a *lower-dimensional representation* of the data on the assumptions that the *relevant structure* is captured by the dimensions with *higher variance*.

To do this we exploit a couple of ideas from statistics and linear algebra:

- We use the *covariance matrix* to account how datapoints vary with respect to each other.
- We use *eigenvalues* and *eigenvectors* to discover the orthogonal dimensions of the covariance matrix we want to preserve.

(The PCA algorithm is grounded in **linear algebra** *(sub-space computation))*

PCA: Algorithm

Given data matrix $\mathbf{X}$ with dimension $N \times D$ (N samples, D dimensions), we want to compute the lower-dimensional representation $\mathbf{Z}$ with dimension $N \times M$:

1. (Center the data $\mathbf{X}$)
2. Compute the *coviariance matrix* of the data:

$$\mathbf{C} = \frac{1}{N}\mathbf{X}^T\mathbf{X}$$

3. Compute the *eigenvalues* $\lambda_1, \lambda_2, ..., \lambda_D$ and the associated *eigenvectors* $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_D$;
4. Sort eigenvalues from big to small and select top-M eigenvalues and their associated eigenvectors;
5. Assemble the chosen eigenvectors into a matrix:

$$\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_M]$$

PCA: Algorithm

6. Project the data into the lower M-dimensional space:

$$\mathbf{Z} = \mathbf{XE}$$

.

In summary, we have a PCA function that allows us to project all the data:

$$\mathrm{PCA}(\mathbf{X}) = \mathbf{XE} = \mathbf{Z}$$

A single datapoint $\mathbf{x}_i$ is projected onto $\mathbf{z}_i$:

$$\mathbf{x}_i \overset{\mathrm{PCA}}{\longmapsto} \mathbf{z}_i$$

and its dimensionality is reduced:

$$\mathbb{R}^D \rightarrow \mathbb{R}^M$$

PCA: Algorithm

PCA allows us to *decompress* or *reconstruct* the original data.

7. Reconstruct the original data:

$$\hat{\mathbf{X}} = \mathbf{Z}\mathbf{E}^T$$

.

This gives us a sort of *inverse* of the PCA function:

$$\mathrm{PCA}^{-1}(\mathbf{Z}) = \mathbf{Z}\mathbf{E}^T = \hat{\mathbf{X}}$$

A single representation $\mathbf{z}_i$ is projected back onto $\hat{\mathbf{x}}_i$:

$$\mathbf{z}_i \overset{\mathrm{PCA}^{-1}}{\longmapsto} \hat{\mathbf{x}}_i$$

and the original dimensionality is restored:

$$\mathbb{R}^M \rightarrow \mathbb{R}^D$$

Notice that PCA performs a *lossy compression*, therefore the reconstruction is not perfect (hence the "hat" over $\hat{\mathbf{x}}$).

PCA: Algorithm

How do we select the number M of eigenvalues/dimensions to preserve?

- Too small M may lead to losing too much information.
- Too large M makes compression/reduction ineffective.

Simple formula for choosing M is based on computing the *proportion of variance*, that is the sum of the selected *eigenvalues* against all the available *eigenvalues*:

$$\text{POV} = \frac{\sum_{i=1}^{M} \lambda_i}{\sum_{j=1}^{D} \lambda_j}$$

and select M so that the proportion of variance is higher than a given threshold (e.g.: 0.9).

PCA: Limitations

The **PCA** algorithm has intrinsic limitations:

- Reliance on the assumption of relevance of variance
- Sensitivity to data scale
- Sensitivity to outlier
- Intrinsic linearity
- Poor scalability

However, when possible, PCA is often chosen to reduce the dimensionality of the data due to its simplicity and understandability.

A nice visualization of PCA in action:
`http://setosa.io/ev/principal-component-analysis/`

PCA: Extensions

Alternatives and extensions try to address some of the above problems:

- *SVD-based PCA*
- *Kernel PCA*
- *Non-linear PCA*
- *Probabilistic PCA*
- *Sparse PCA*
- ...

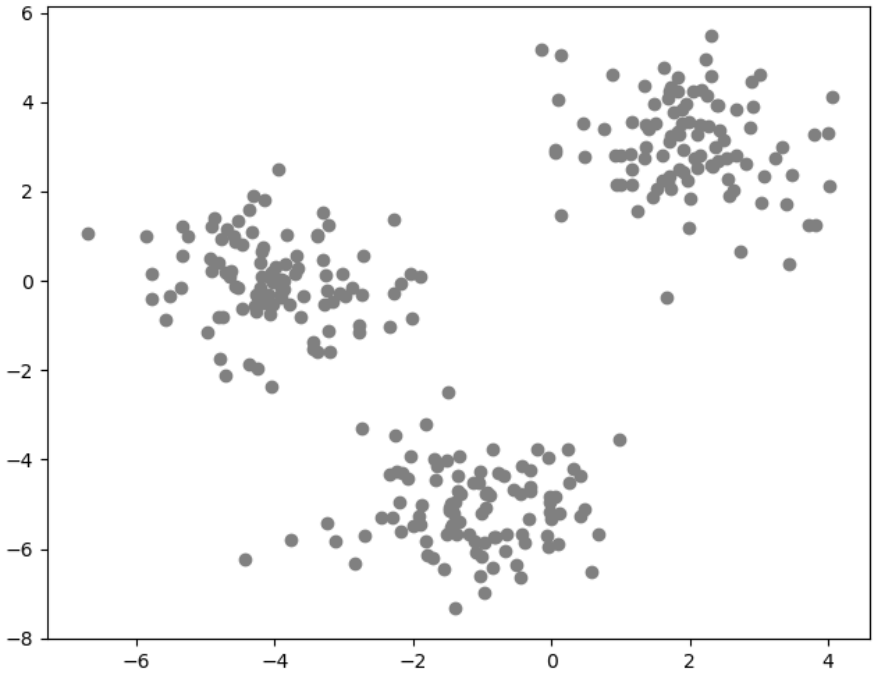
More on PCA

More on PCA in the mandatory assignment.

5.2. K-Means

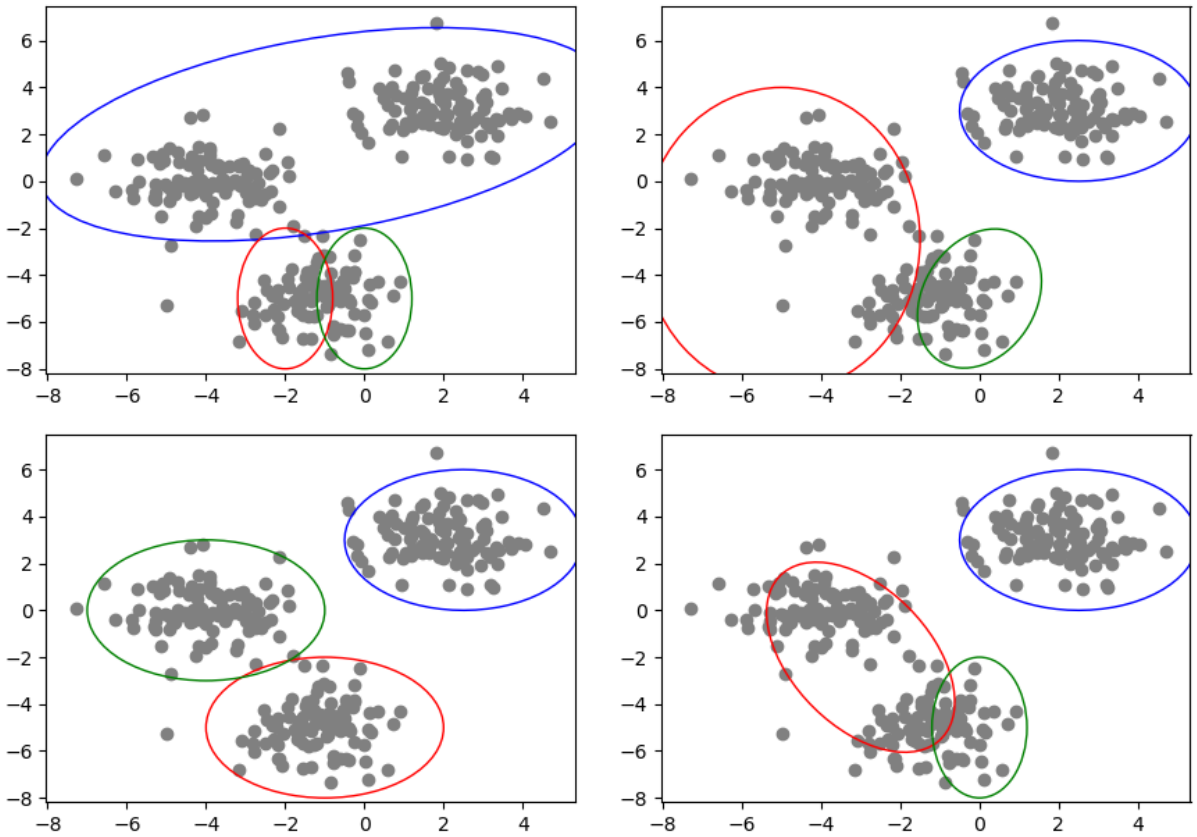
K-Means: Intuition

K-Means is an unsupervised learning technique for *clustering*



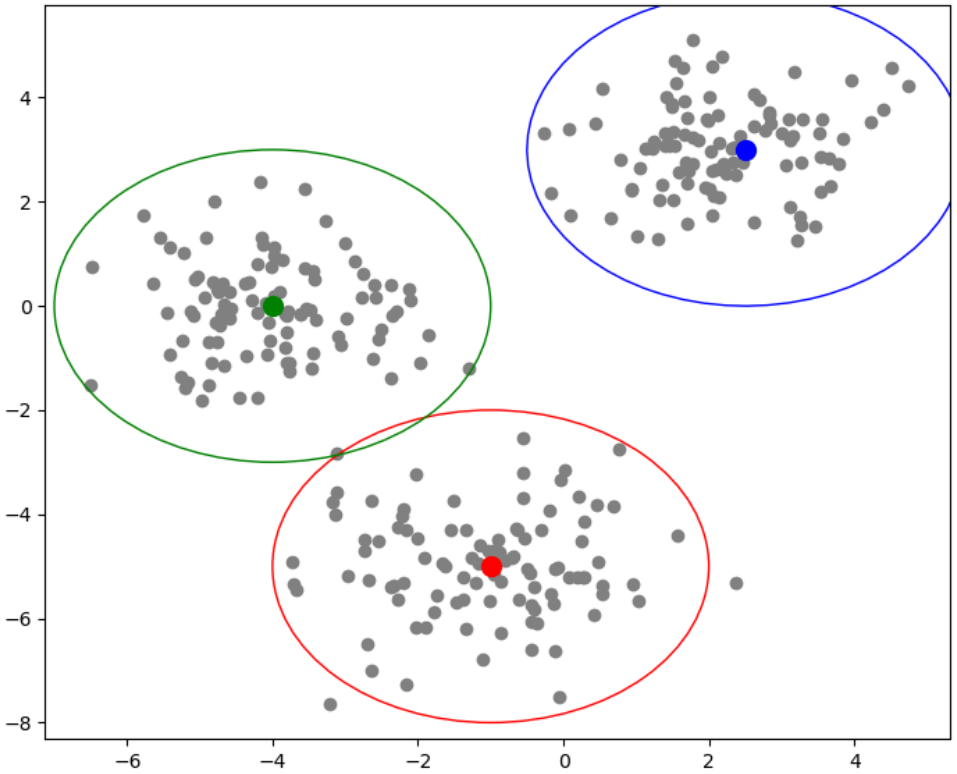
K-Means: Intuition

If we have to group points in a fixed number of groups, say 3, which one would we choose?



K-Means: Intuition

K-Means finds iteratively the centers of clusters.



K-Means: Algorithm

K-Means tries to learn a *lower-dimensional representation (clusters)* of the data on the assumptions that the *relevant structure* is captured by *distances* among points.

To do this we rely on a couple of alternating steps:

- Given cluster centers, we *assign each data point* to the closest cluster center.
- Given the assignment of the data points, we *recompute the cluster centers* by taking the mean of all the points in the cluster.

Notice the dependence of one step from the other. In order to start, we need to *bootstrap* (we take an initial guess)

(The PCA algorithm is grounded in **statistics** *(EM algorithm))*

K-Means: Algorithm

Given data matrix $\mathbf{X}$ with dimension $N \times D$ (N samples, D dimensions), we want to partition the data in K cluster:

1. Randomly initialize K cluster centers c_k with dimension D.
2. Repeat until *convergence*:
   1. For each data point x_i, compute the *distance* $D(x_i, c_k)$ between the data point and all the cluster centers c_k
   2. Assign each point x_i to the cluster c_k at minimal distance

$$\text{cluster}\,(x_i) = \underset{k}{\text{argmin}}\, D(x_i, c_k)$$

   3. Recompute the cluster centers k_j by taking the mean of all the data points x_i assigned to k_j.

$$c_k = \frac{1}{N_k} \sum_{\text{cluster}(x_i)=k} x_i$$

K-Means: Algorithm

How do we define convergence?

- Usually take to be the change in cluster centers:

$$\left| c_k^{old} - c_k^{new} \right| \leq \epsilon$$

- If this hold for all the clusters for a small ϵ, we conclude that the algorithm has converged.

How do we define distance?

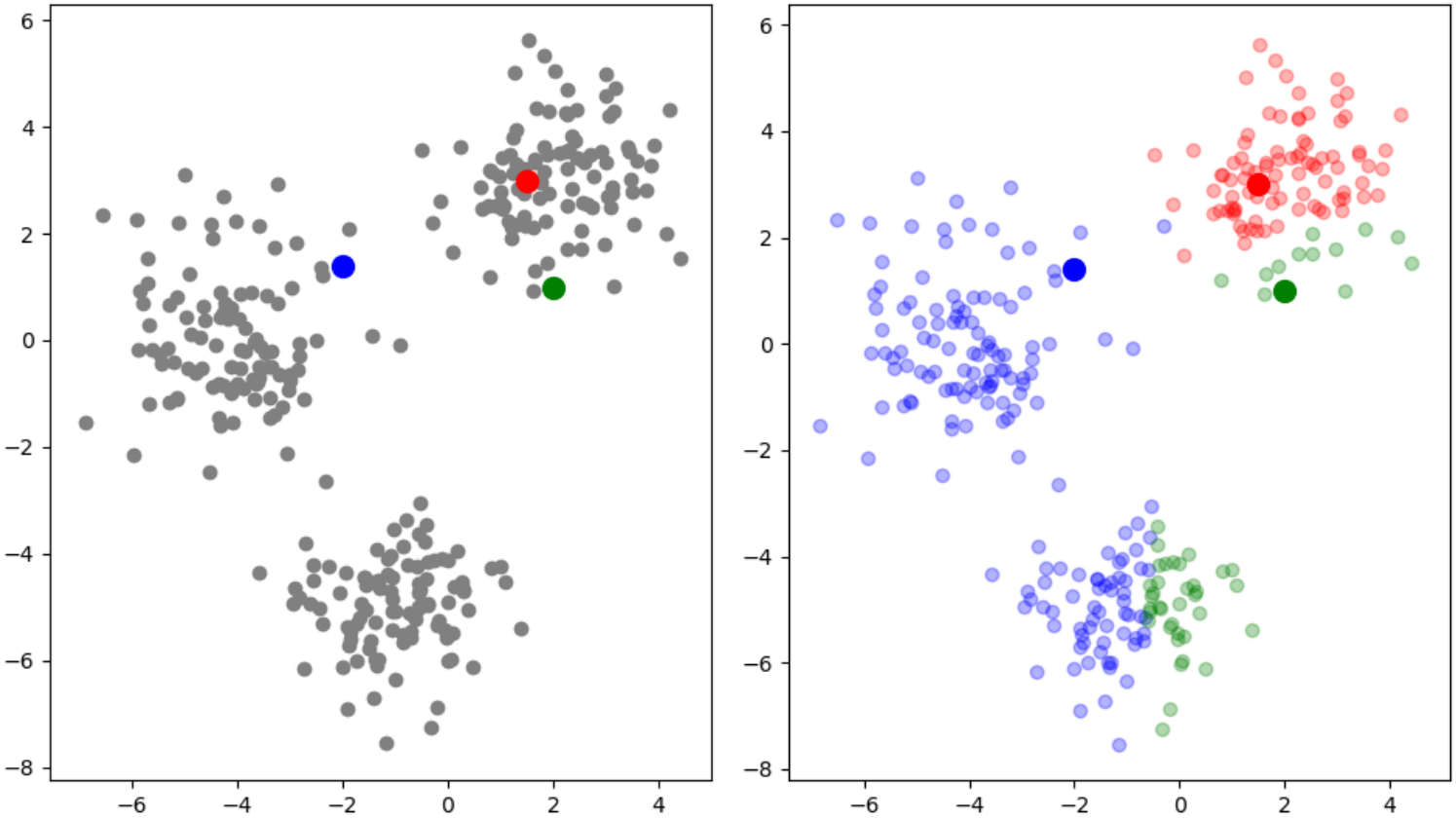
- Usually taken to be the standard *Euclidean* distance:

$$D(x_i, c_k) = \sqrt{(x_i - c_k)^2}$$

- This encode an *assumption* on the structure of the space.
- Other distances may be used.

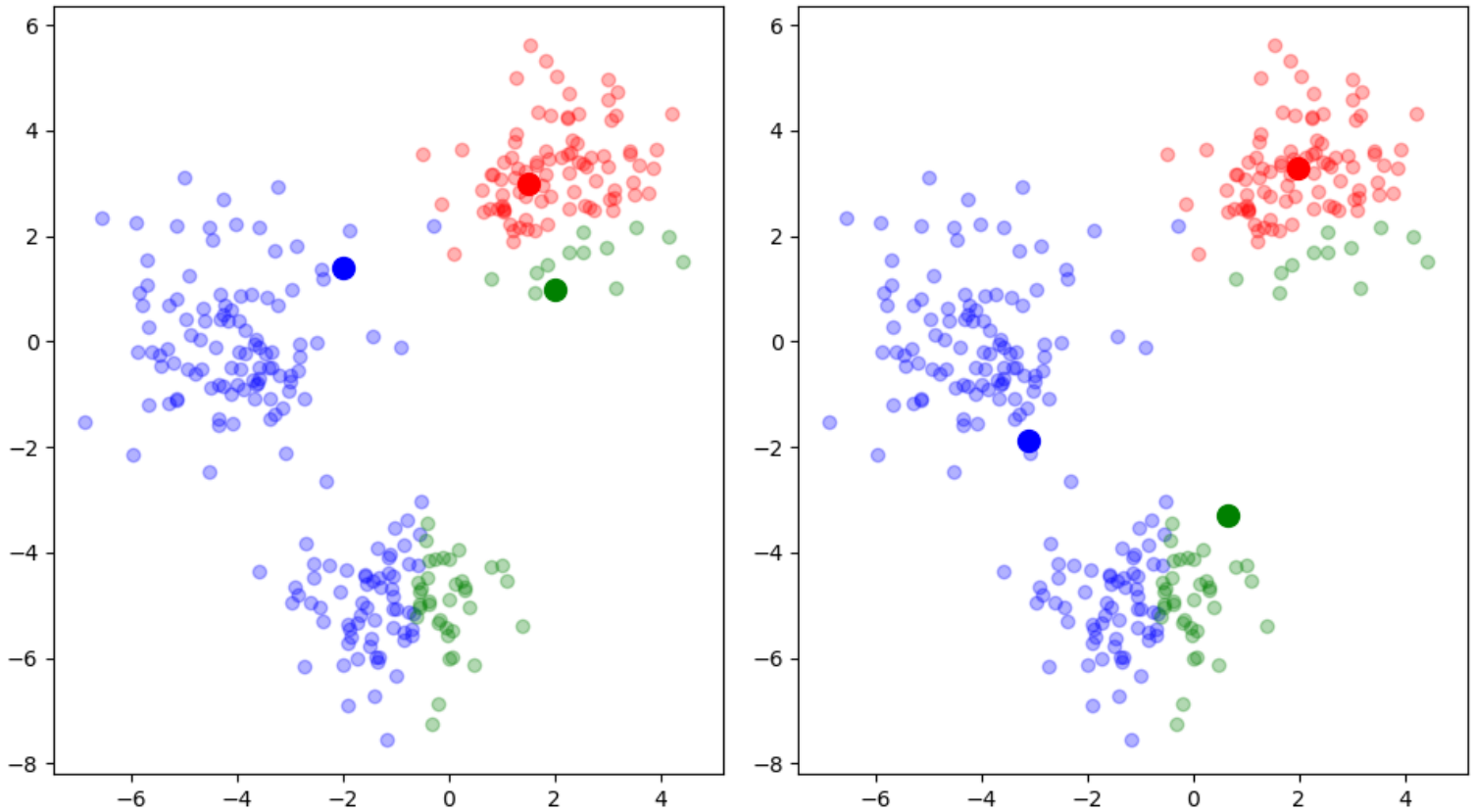
K-Means: Algorithm

Assignment of data points to randomly initialized cluster centers



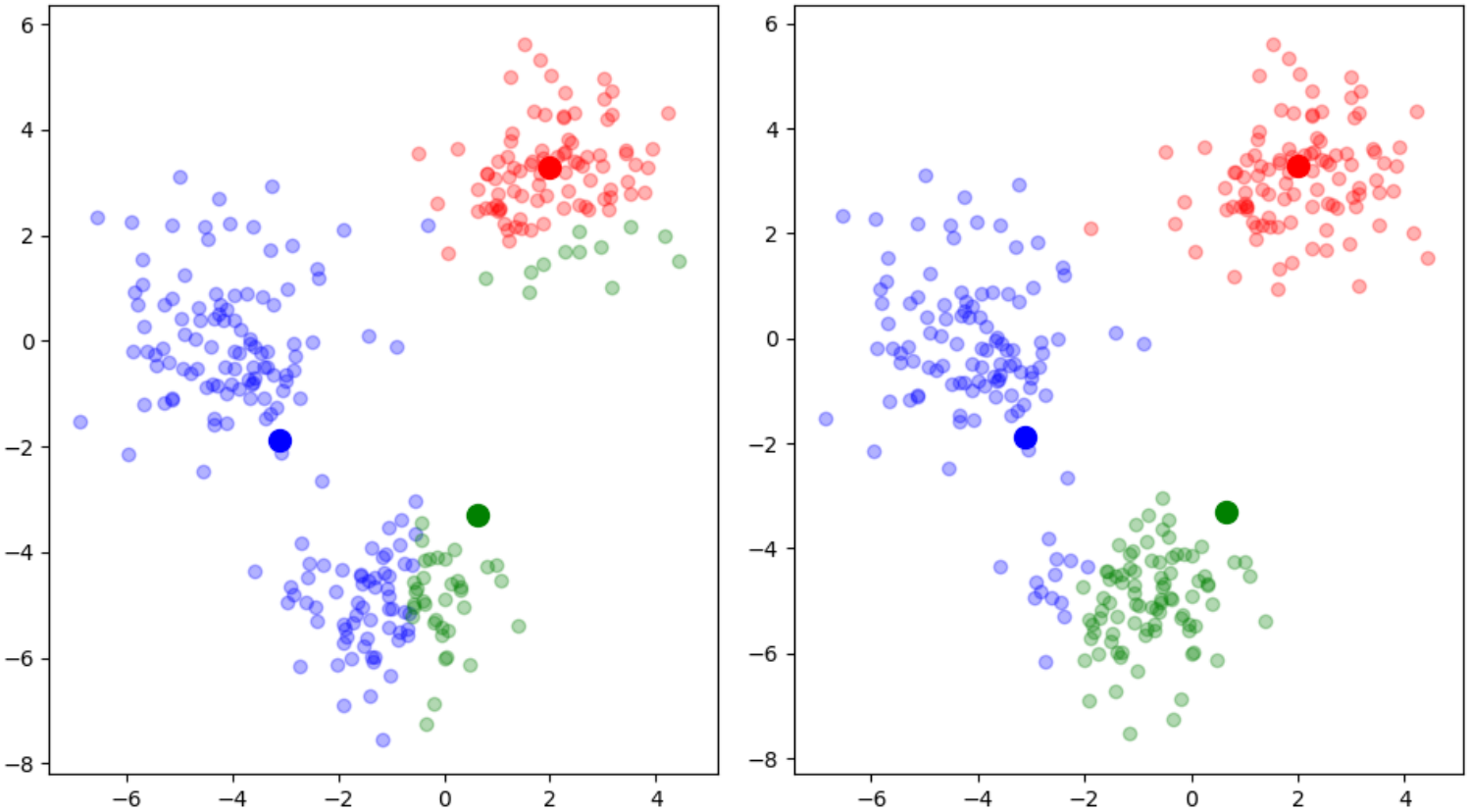
K-Means: Algorithm

Computation of new cluster centers from the previous assignment



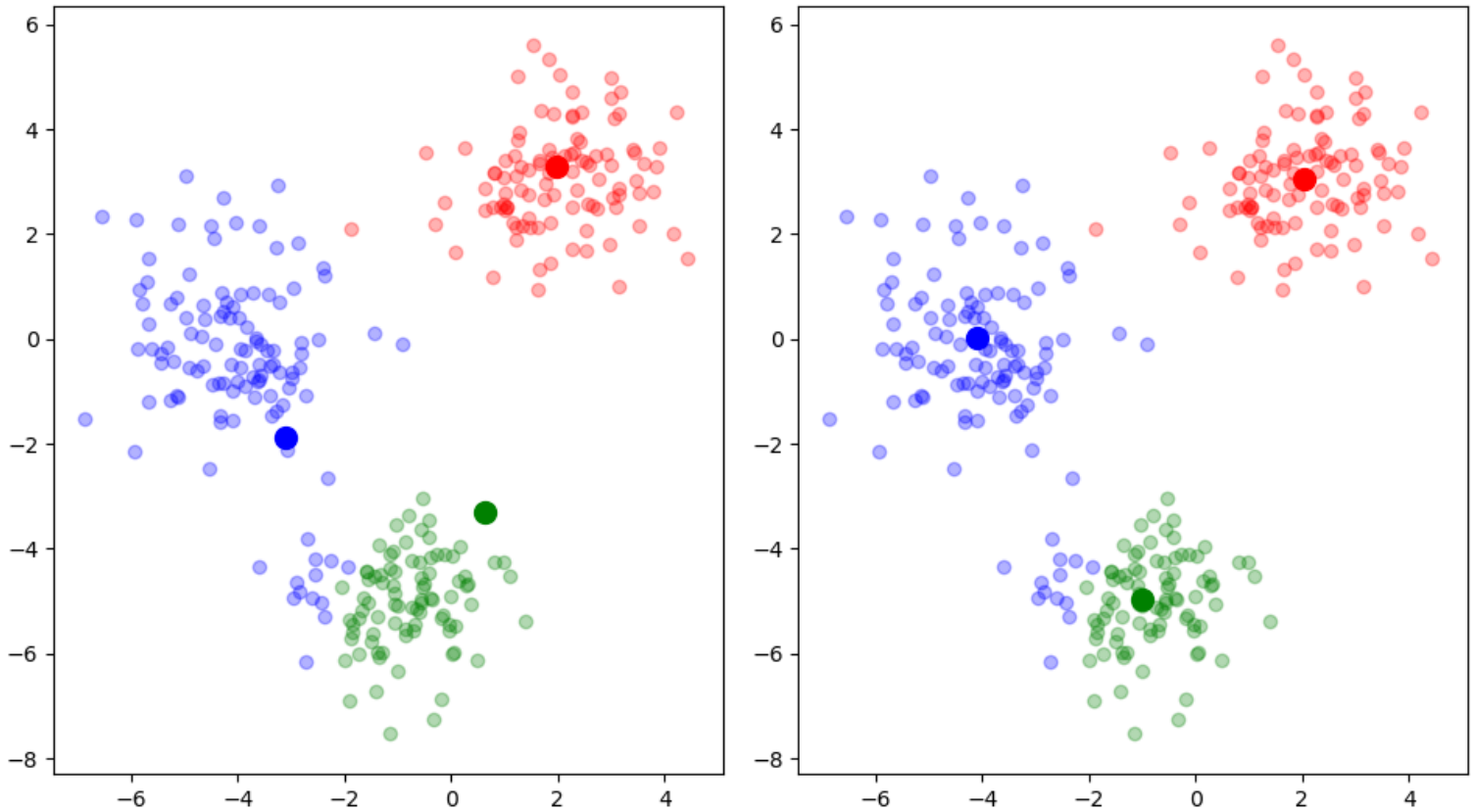
K-Means: Algorithm

Assignment of data points to new cluster centers



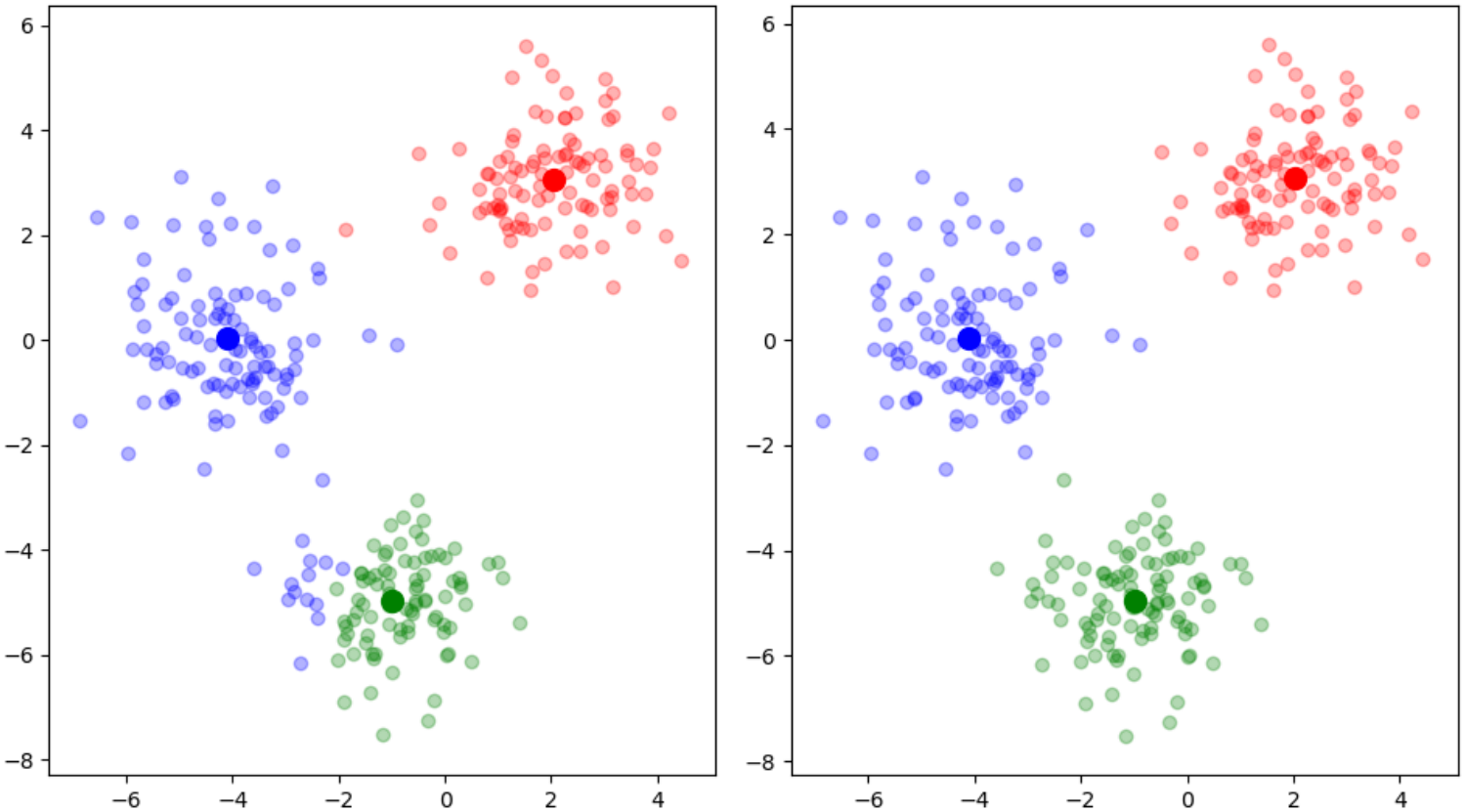
K-Means: Algorithm

Computation of new cluster centers from the previous assignment



K-Means: Algorithm

Assignment of data points to new cluster centers



K-Means: Limitations

The **K-Means** algorithm has intrinsic limitations:

- Reliance on the assumption of type of distance
- Sensitivity to data scale
- Local minima from random initialization
- Hardness of assignments

K-Means: Extensions

Alternatives and extensions try to address some of the above problems:

- *K-median clustering*
- *K-means++*
- ℓ_1-distance *k-means clustering*
- *Cosine k-means clustering*
- *Gaussian mixture models*
- ...

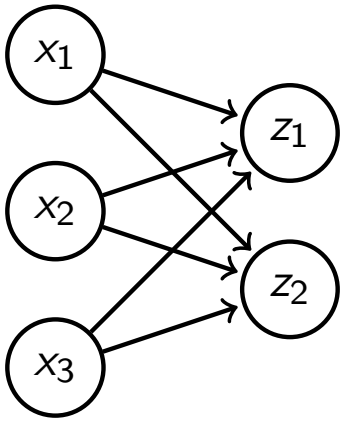
More on K-Means

More on k-means in the mandatory assignment.

5.3. Autoencoders

Autoencoders: Intuition

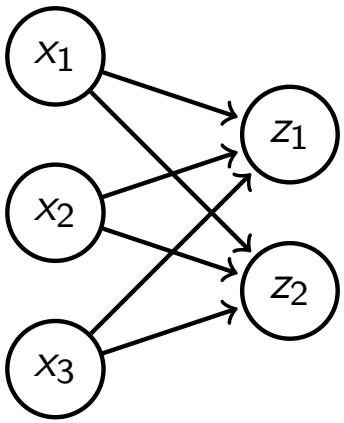
Autoencoders are unsupervised learning models for *representation learning* and *dimensionality reduction*.



(Also known as: Diabolo network)

Autoencoders: Intuition

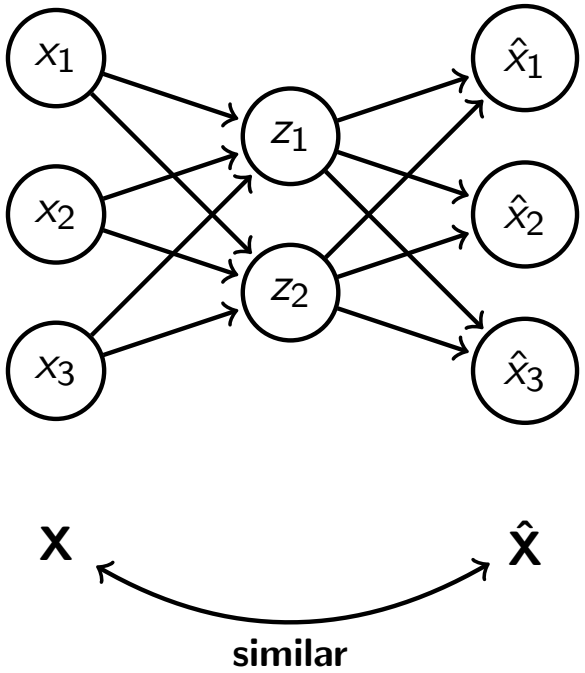
Given a set of data X and a neural network, how could we *train* the neural network without labels y?



$$\mathbf{X} \xrightarrow{\text{compress}} \mathbf{Z}$$

Autoencoders: Intuition

An **autoencoder** uses the same original data $\mathbf{X}$ as a target for training.



The original data $\mathbf{X}$ and the reconstruction $\hat{\mathbf{X}}$ are forced to be as similar as possible.

Autoencoders: Algorithm

Autoencoders try to learn a *lower-dimensional representation (compression)* of the data on the assumptions that the *relevant structure* is captured by the information necessary to reconstruct as well as possible the original input.

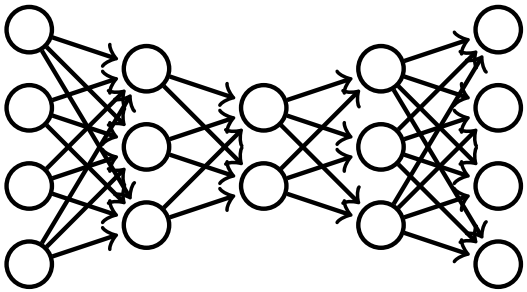
To do this we rely on *neural networks* to learn to compress and decompress the data.

(The PCA algorithm is grounded in **optimization** */* **neural networks***)*

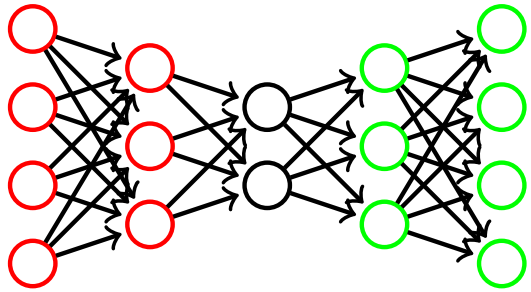
Autoencoders: Algorithm

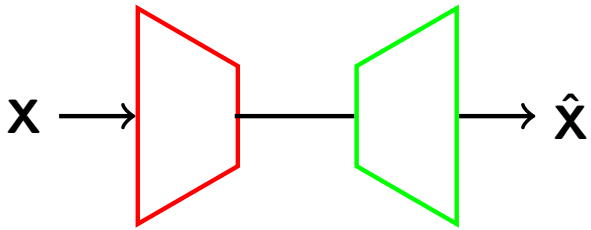
An **autoencoder** can be viewed as:

Bottleneck single neural network



An encoder and decoder network





(Strictly speaking, an autoencoder must not necessarily have a bottleneck shape)

Autoencoders: Algorithm

Given data matrix $\mathbf{X}$ with N samples:

1. Setup your autoencoder architecture (assume here a one-layer encoder and one-layer decoder).
2. Compute the output of the *encoder network* given the input $\mathbf{X}$.

$$\mathbf{Z} = f\left(W_{enc}\mathbf{X} + b_{enc}\right)$$

3. Compute the output of the *decoder network* given the encoding $\mathbf{Z}$.

$$\hat{\mathbf{X}} = g\left(W_{dec}\mathbf{Z} + b_{dec}\right),$$

4. Compute a *reconstruction loss*, such as mean square loss:

$$\mathcal{L}\left(\mathbf{x}, \hat{\mathbf{x}}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{x}_i - \hat{\mathbf{x}}_i\right)^2$$

5. Optimize by *gradient descent*.

Autoencoders: Limitations

Autoencoders have limitations similar to neural networks:

- Hyperparameter tuning
- Sample complexity
- Local minima
- Assumption that reconstruction under the given loss preserves relevant information

Autoencoders: Extensions

Alternatives and extensions to improve autoencoders:

- *Denoising autoencoders*
- *Contrastive autoencoders*
- *Variational autoencoders*
- ...

Questions?

Feel free to ask questions at fabiomz@ifi.uio.no

References I

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[3] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[4] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

References II

[5] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[6] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[7] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.