Unsupervised Learning

Pietro Gori

Ass. Professor
Equipe IMAGES - Télécom Paris - IPParis
pietro.gori@telecom-paris.fr

Useful links for Python:

- **Python**: http://greenteapress.com/wp/think-python/
- http://mathesaurus.sourceforge.net/matlab-numpy.html
- http://www.loria.fr/~rougier/teaching/matplotlib/matplotlib.html
- http://jrjohansson.github.io/numericalpython.html
- **Scikit-learn**: http://scikit-learn.org/stable/index.html
- **Anaconda**: https://docs.anaconda.com/anaconda/user-guide/getting-started/
- **Jupyter Notebook**: http://jupyter.org/

- **Book 1**: Hastie, Tibshirani and Friedman (2009). The Element of Statistical Learning. Springer.
  https://web.stanford.edu/~hastie/Papers/ESLII.pdf
- **Book 2**: Christopher M. Bishop (2006). Pattern Recognition and Machine Learning. Springer

Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

Statistical Machine Learning

Statistical machine learning is a field at the intersection of *statistics*, *computer science* and *optimization*. It is a branch of Artificial Intelligence (AI). The main goal is to develop algorithms that "learn" from the data. It can be divided into:

- **Supervised Learning**
  - *Regression*
  - *Classification*
- **Unsupervised Learning**
  - *Clustering*
  - *Dimensionality reduction*
  - *Outlier detection*

Statistical Machine Learning

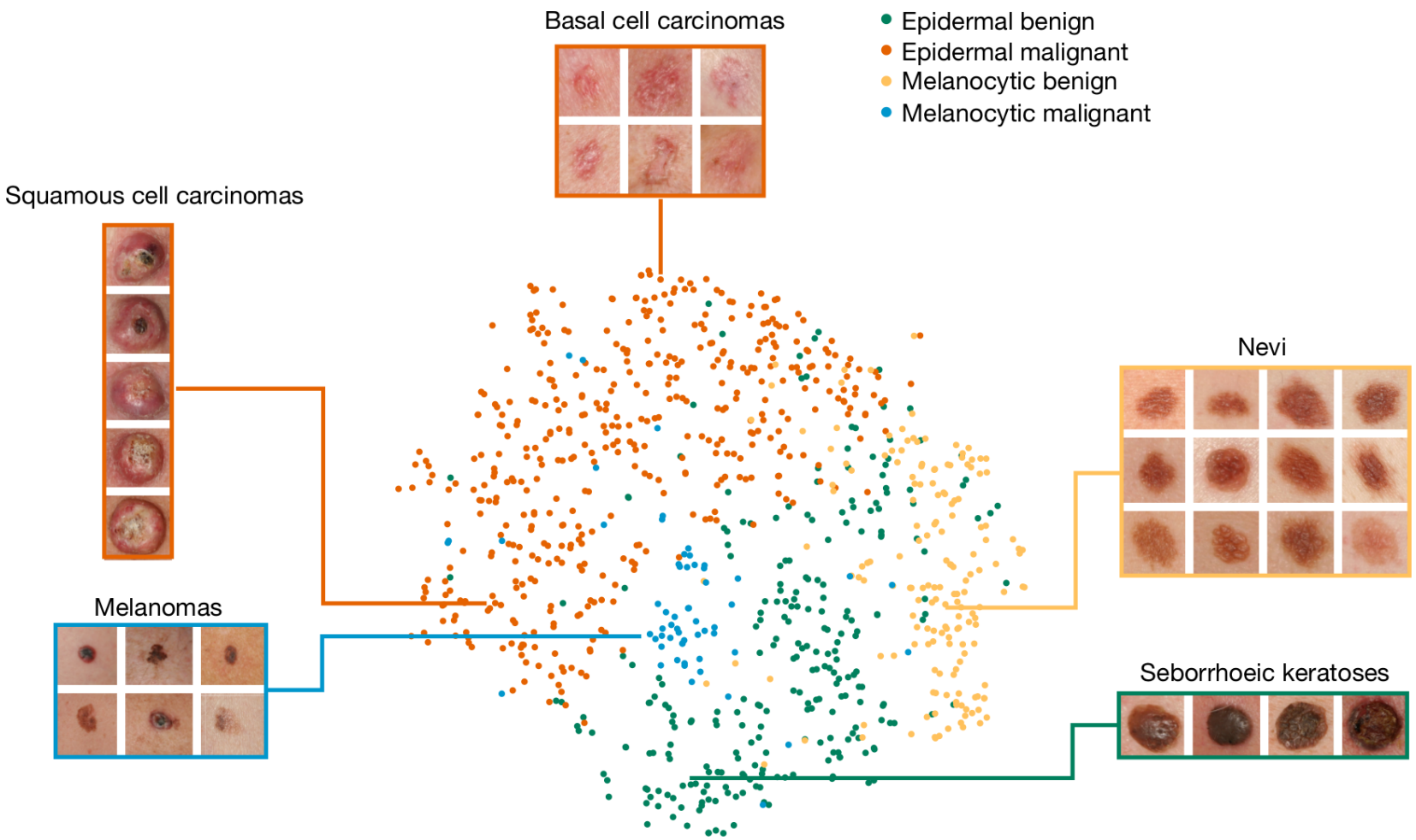
- **Supervised Learning**: inferring a rule f from a labeled training set $(X,Y)=\{(x_1,y_1),...,(x_N,y_N)\}$ such that x_i is the vector of features describing the i-th observation and y_i is its label. The rule f will map any new observation t described by x_t to its corresponding y_t.
  - **Regression**: y_i is a set of continuous variables (e.g., $y_i \in \mathbb{R}$)
  - **Classification**: y_i is a set of discrete categories/classes

Statistical Machine Learning

- **Supervised Learning**: inferring a rule f from a labeled training set $(X,Y)=\{(x_1,y_1),...,(x_N,y_N)\}$ such that x_i is the vector of features describing the i-th observation and y_i is its label. The rule f will map any new observation t described by x_t to its corresponding y_t.
  - **Regression**: y_i is a set of continuous variables (e.g., $y_i \in \mathbb{R}$)
  - **Classification**: y_i is a set of discrete categories/classes

- **Unsupervised Learning**: No labels are given, we only have X. Possible goals are:
  - **Clustering**: divide X into homogeneous groups
  - **Dimensionality reduction**: simplify X by mapping it into a lower dimensional-space (e.g., PCA)
  - **Outlier detection**

Unsupervised Learning



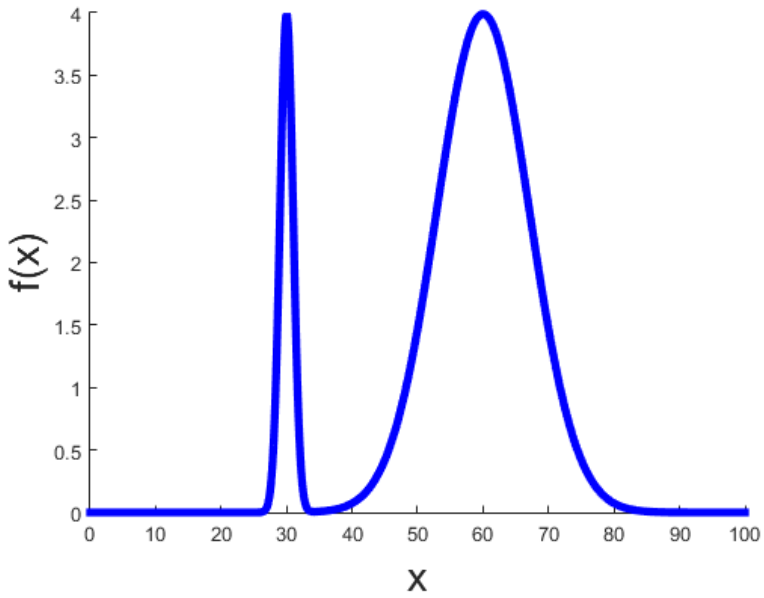
Example of dimensionality reduction and clustering (using Deep Learning and t-SNE) from Esteva et al., Nature, 2017

Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

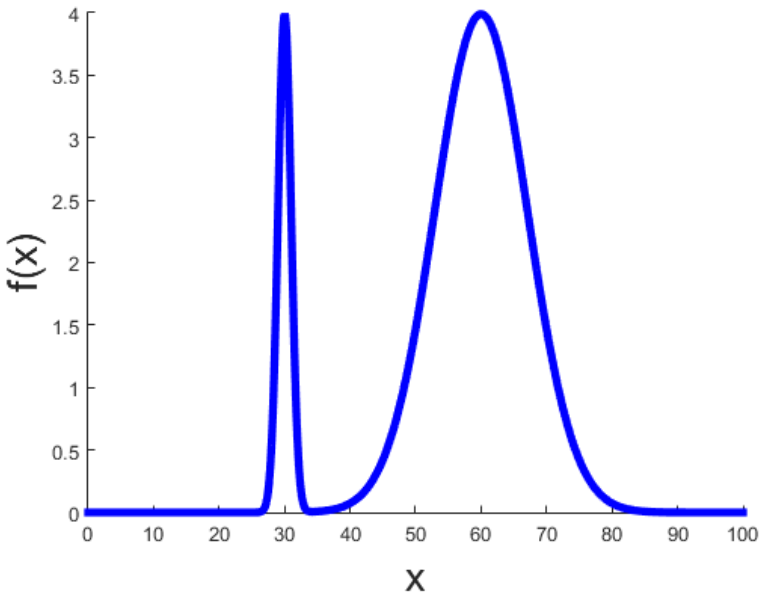
Curse of dimensionality

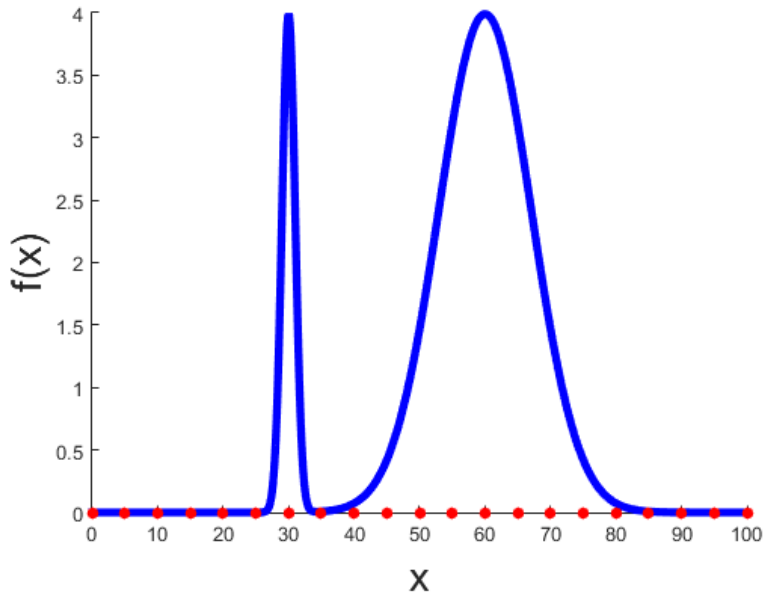
- Let $f : X \to Y$ be a 1-D unknown function of x (i.e. $x \in \mathcal{R}$)



Curse of dimensionality

- Let $f : X \to Y$ be a 1-D unknown function of x (i.e. $x \in \mathcal{R}$)

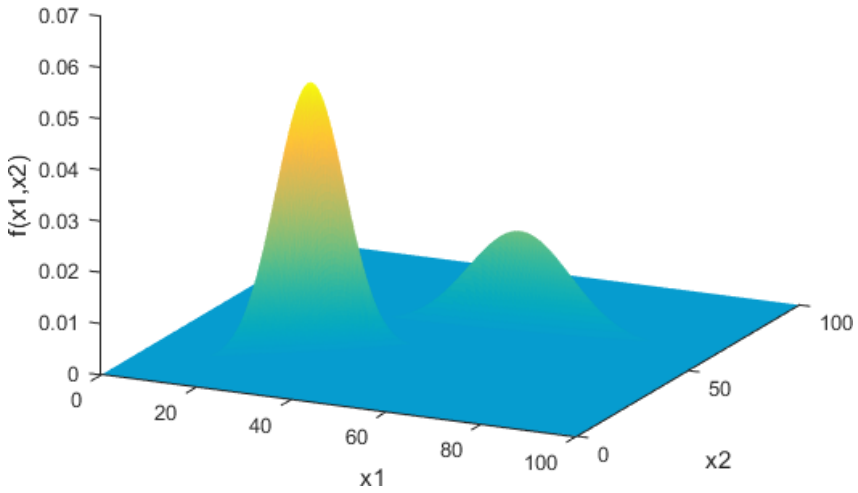




- We know that $x \in [0\ 100]$ and we notice that with 21 sample points (constant interval of 5) we correctly retrieve the shape of the function

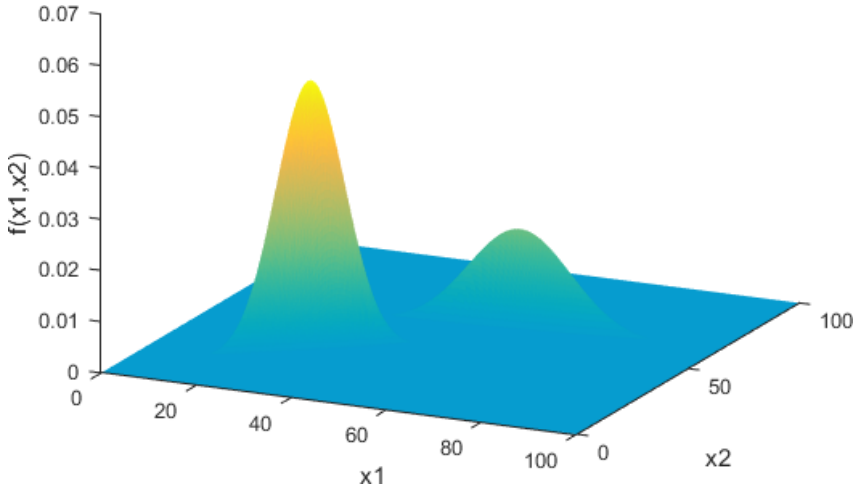
Curse of dimensionality

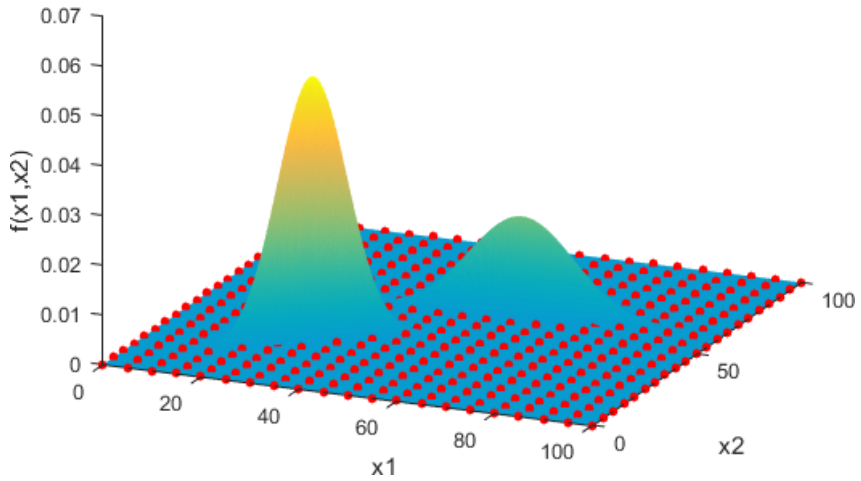
- How many points do we need to obtain an equivalent sampling in 2D ?



Curse of dimensionality

- How many points do we need to obtain an equivalent sampling in 2D ?
  $21^2 = 441$

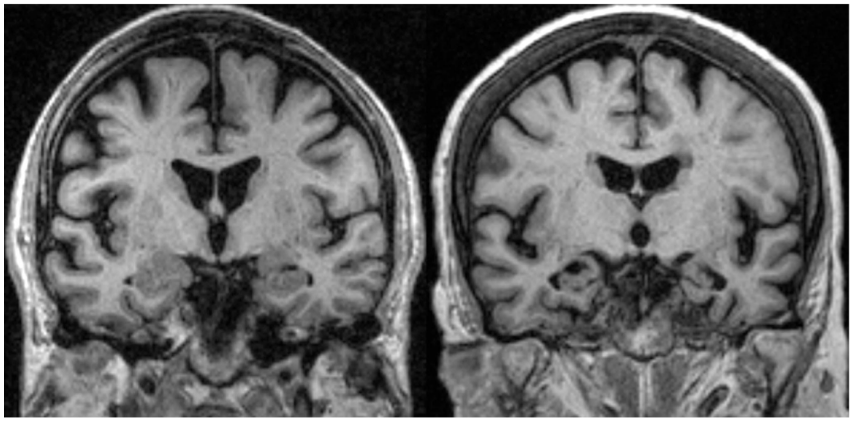


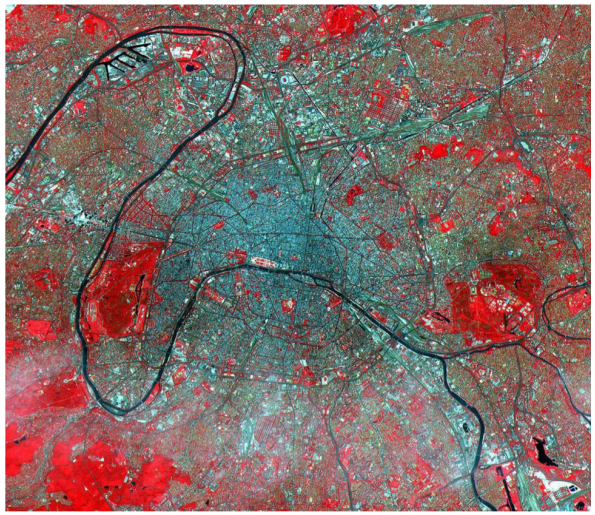


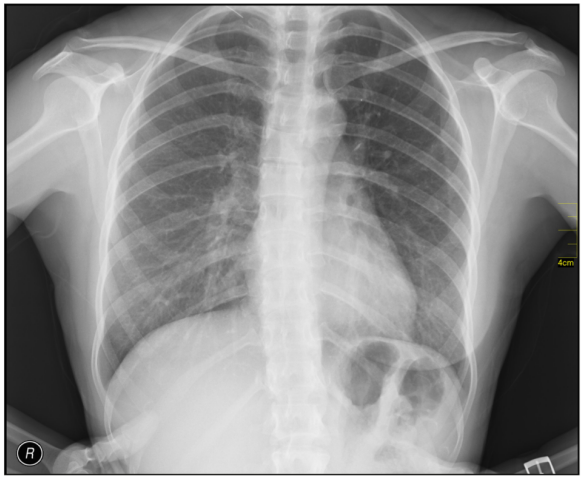
- In $\mathcal{R}^{10}$ we would need 21^{10} sample points !
- This is called **curse of dimensionality** ! There is an exponential increase in the number of sampling points when increasing the number of dimensions

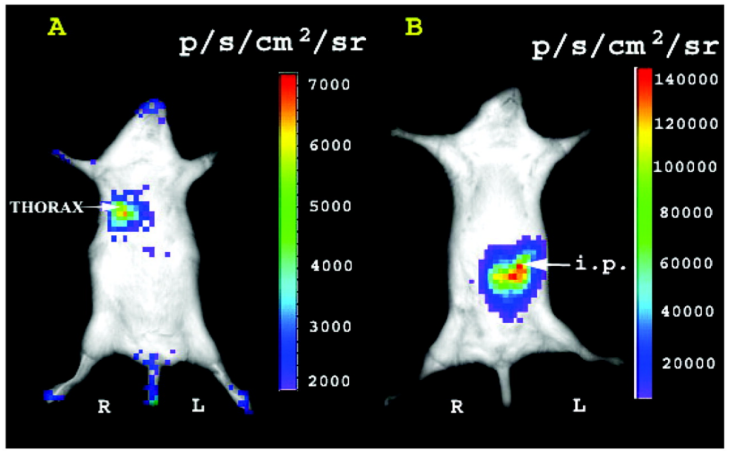
Curse of dimensionality

- In most image-based applications X is a high-dimensional space









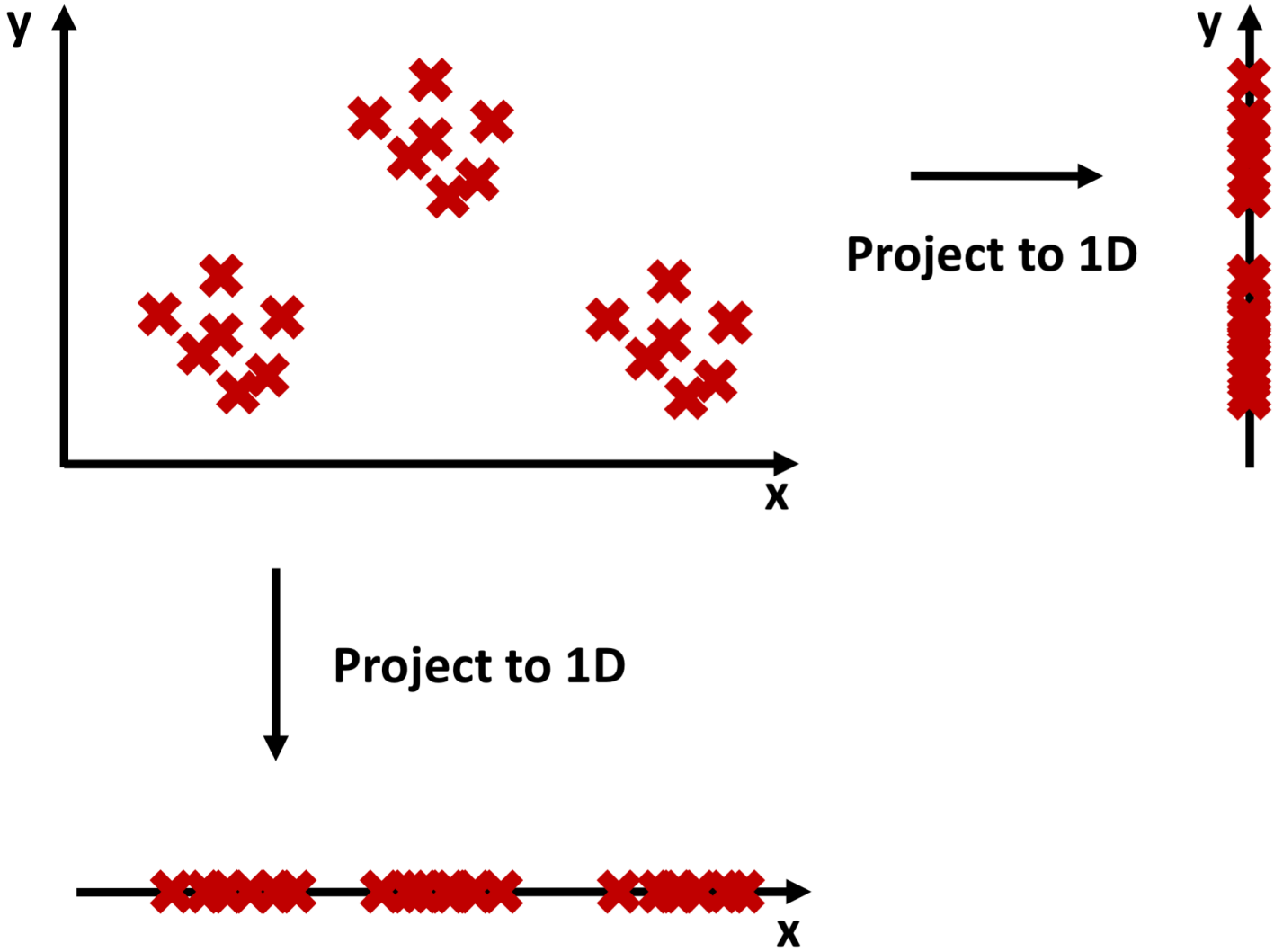
Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

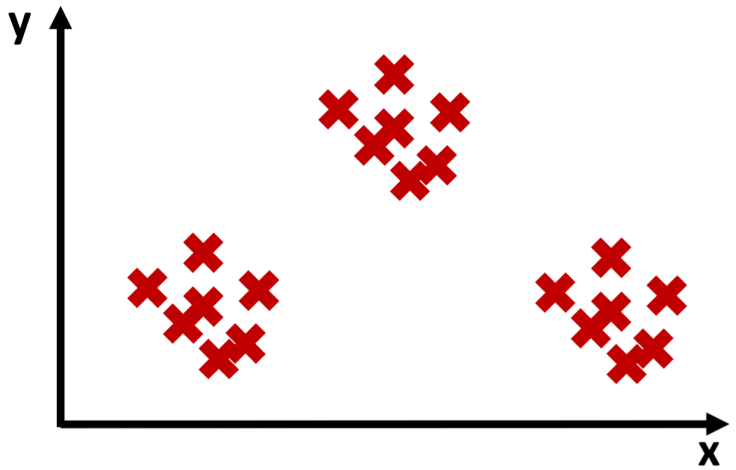
Hypotheses

- **Data are redundant**: in many cases data can be explained with less "latent" features (dimensions)
- Data can be decomposed into smaller "hidden" **internal representations** (clusters)
- Data might be **structured** (i.e. organized) and/or follow **physical/anatomical rules**
- **Law of parsimony** (W. Ockham): given all other things being equal, the simplest solution is the best, which means the one that makes as few assumptions as possible (less features/dimensions)

Redundancy



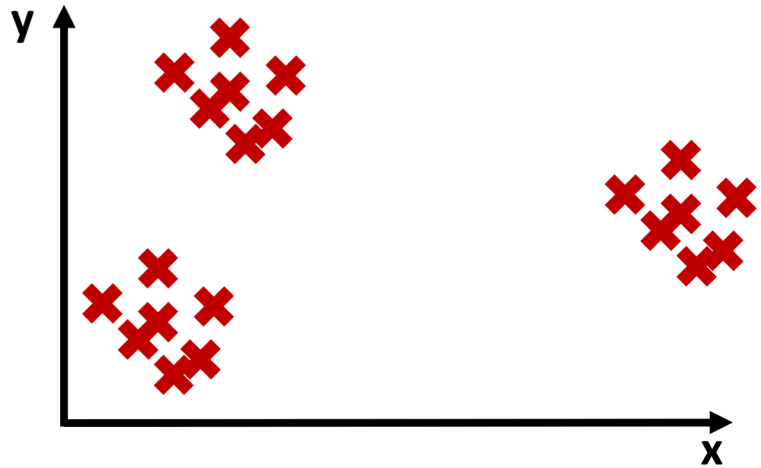
Redundancy



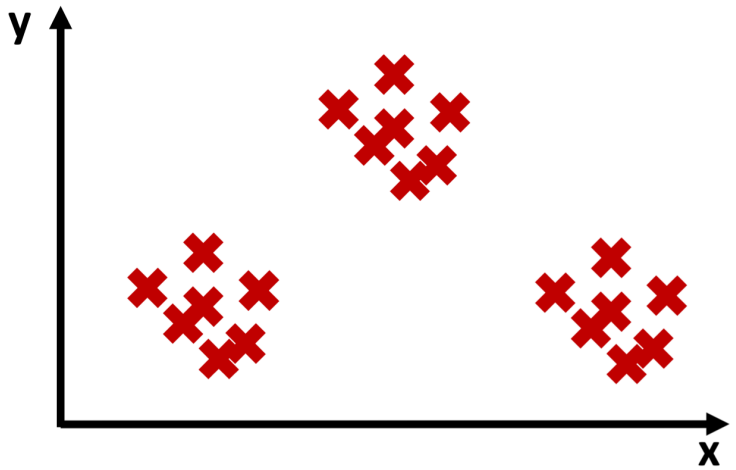
Project to 1D

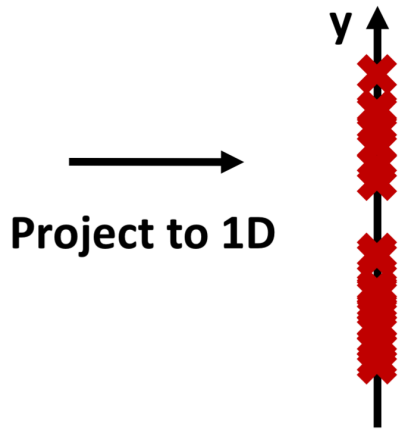
Project to 1D

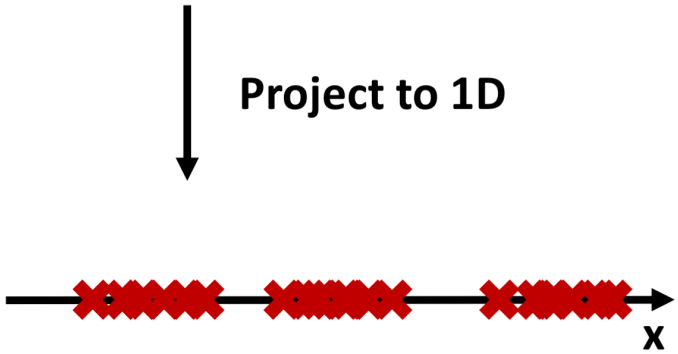
And what if ?

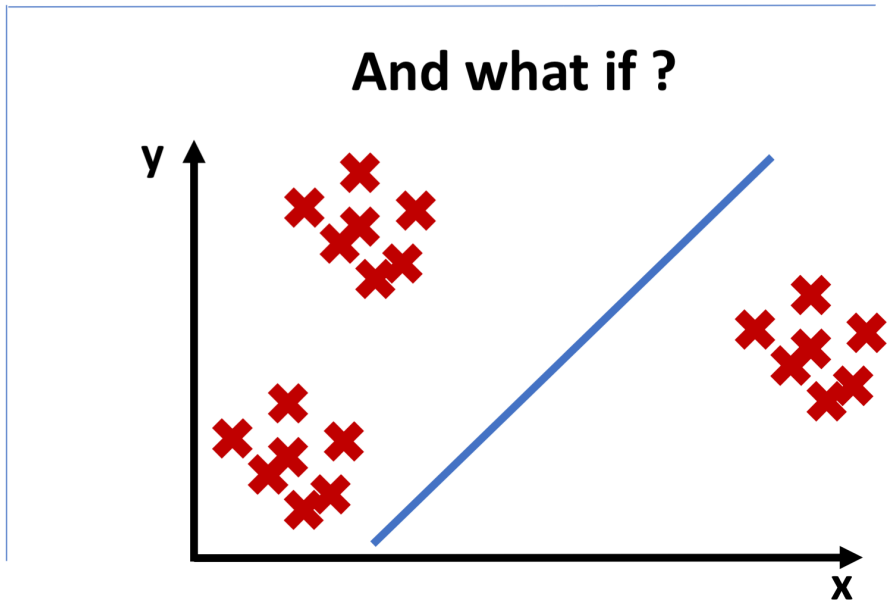


Redundancy









Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

Goals of Unsupervised Learning

- **Dimensionality reduction**: map high-dimensional data to a low-dimensional space. Describe the data with a smaller set of latent variables
  1. Principal Component Analysis (PCA)
  2. Kernel PCA
  3. Independent Component Analysis (ICA)
  4. Non-negative Matrix Factorization (NNMF)
- **Clustering**: group data into homogeneous groups based on a similarity measure
  1. K-means
- **Outlier detection**: detect few observations that deviate from the distribution of the majority (considered as the actual distribution of X)

Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

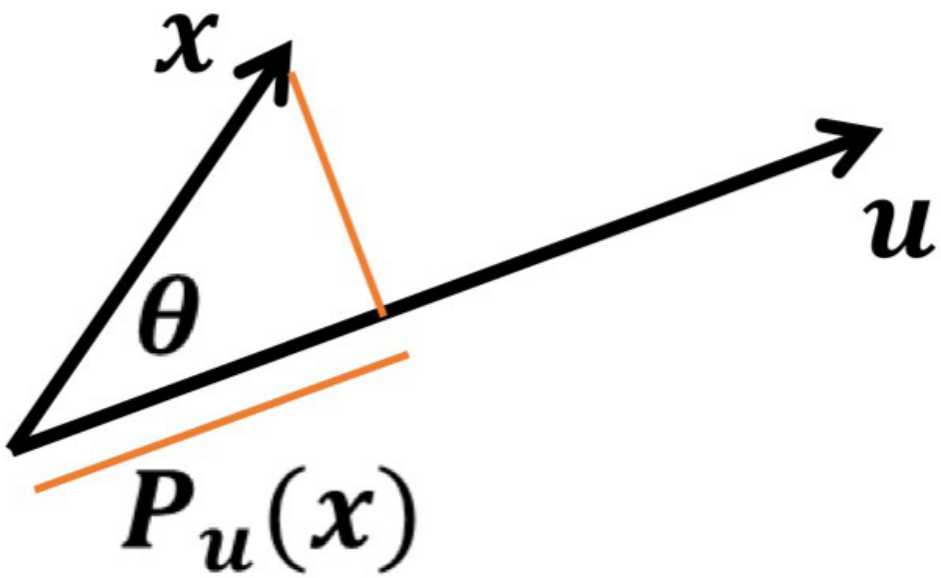
PCA - Principal Component Analysis

Definition

- **Pearson** (1901): linear projection that minimizes the average projection cost
- **Hotelling** (1933): orthogonal projection of the data onto a smaller linear subspace such that the variance of the projected data is maximized
- **Jolliffe** (2002): orthogonal linear transformation that maps the data to a new coordinate system which are the (few) orthonormal directions that retain most of the original variability

PCA - Notation

- Let N and d be the number of observations and feature respectively
- Let x_i be a column vector belonging to $\mathcal{X} = \mathcal{R}^d$. It represents one observation with d features
- Remember that the definition of orthogonal projection of a vector x onto a unit-length vector u is : $P_u(x) = (x^T u)u$ where the length of the projection is $||P_u(x)||_2 = x^T u$



Definition

- **Hotelling** (1933): orthogonal projection of the data onto a smaller linear subspace such that the variance of the projected data is maximized

The variance of the projection of all x_i onto u is:

$$\text{Var}(|P_u(\{x_i\})|) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i^T u - \bar{x}^T u)^2 = u^T C u \qquad (1)$$

where $C = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$ and $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$

PCA

- By definition, we look for the direction u such that the variance of the projected data is maximized: $\arg\max_u u^T C u$
- We are only interested in the direction of u not in its magnitude. To prevent $||u||_2 \to \infty$ we choose a unit vector so that $||u||_2^2 = 1$

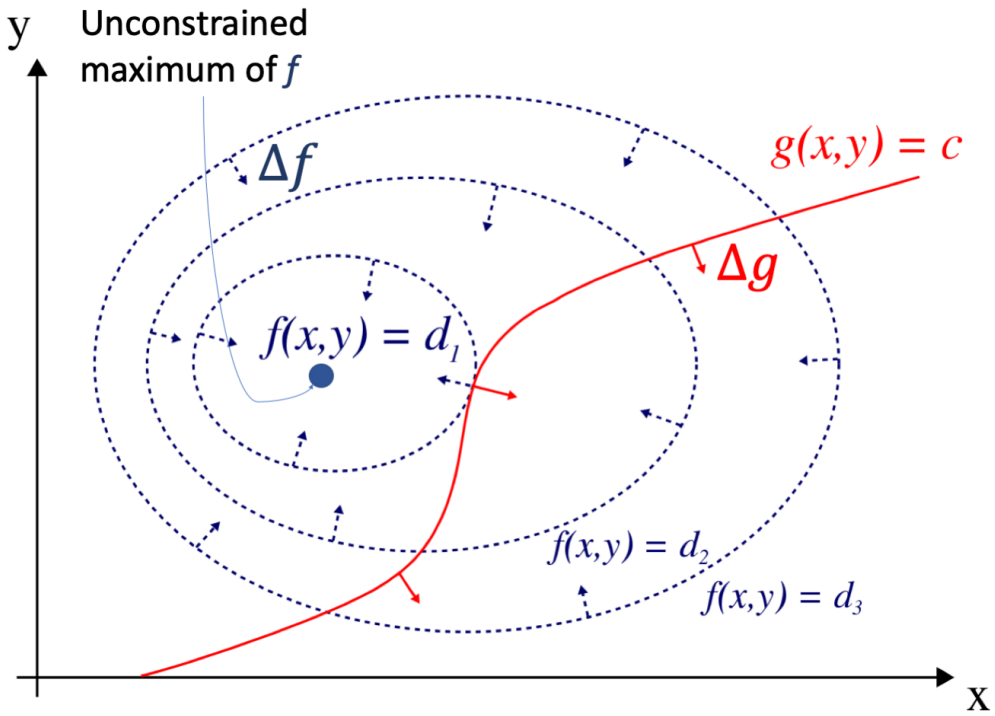
$$u_1^* = \arg\max_u u^T C u \quad \text{s.t.} \quad ||u||_2^2 = 1 \tag{2}$$

- This is a constrained optimization problem. How do we solve it ?

Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

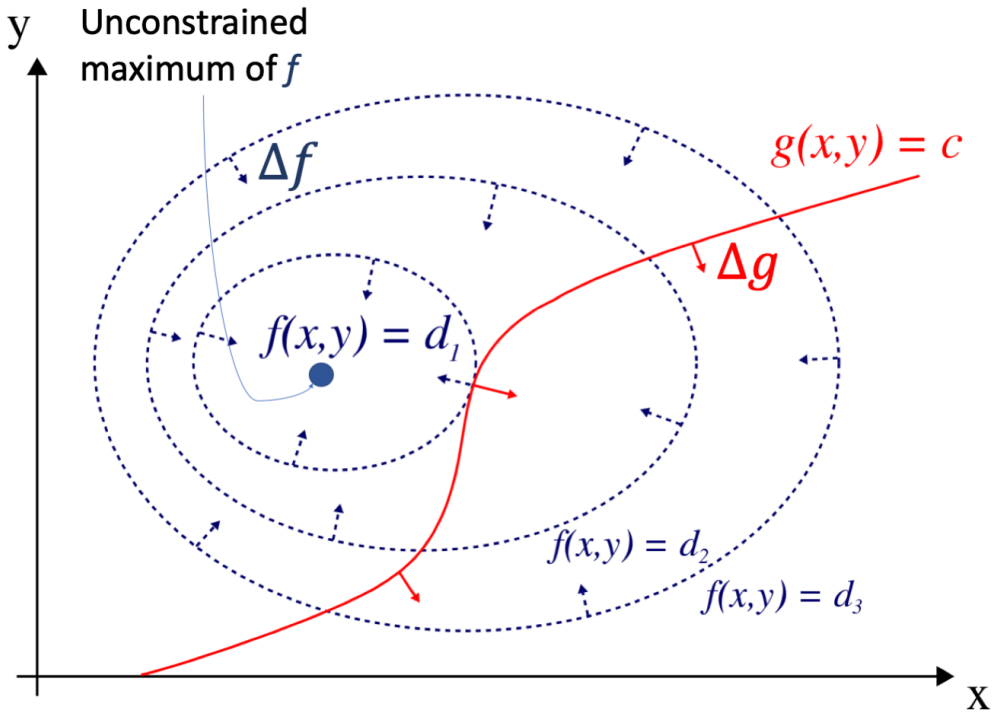
Lagrange Multipliers



$$\begin{aligned} \max \quad & f(\mathbf{x}) \\ s.t. \quad & g(\mathbf{x}) = 0 \quad (\text{or} \quad g(\mathbf{x}) - c = 0) \end{aligned} \tag{3}$$

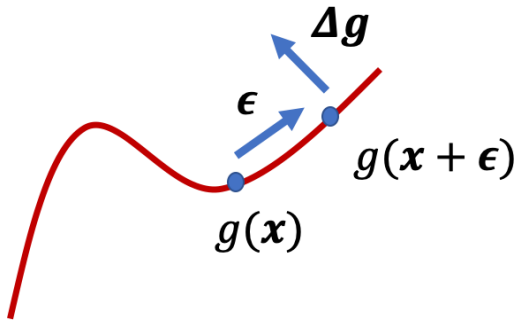
- **Idea**: we walk along the *constraint line* $g(\mathbf{x}) = 0$ and we look for *stationary points* of $f(\mathbf{x})$, namely points where $f(\mathbf{x})$ remains locally constant (always where $g(\mathbf{x}) = 0$)

Lagrange Multipliers



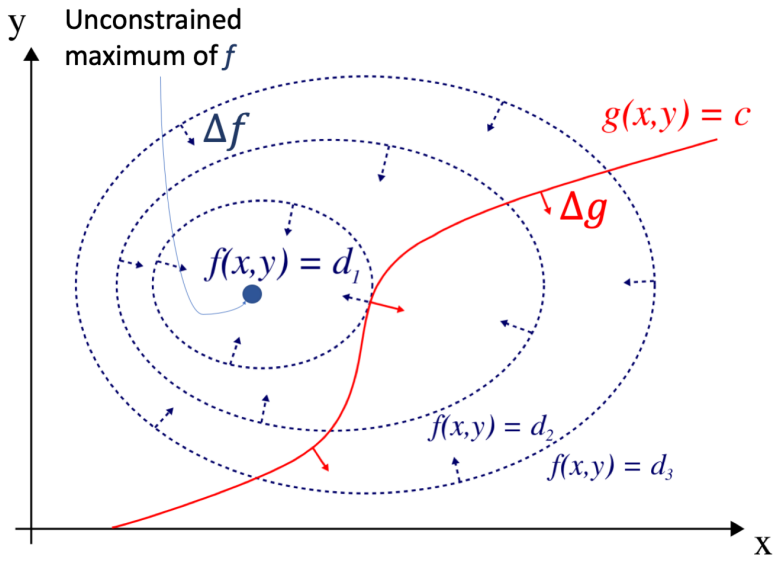
- $f(\mathbf{x})$ remains locally constant along its contour lines $f(\mathbf{x}) = d$
- Thus, we have stationary points when we are walking along $g(\mathbf{x}) = 0$ and at the same time we are also following a contour line of f. This happens when $f(\mathbf{x}) = d$ and $g(\mathbf{x}) = 0$ are parallel ! How to find this constrained stationary condition ?

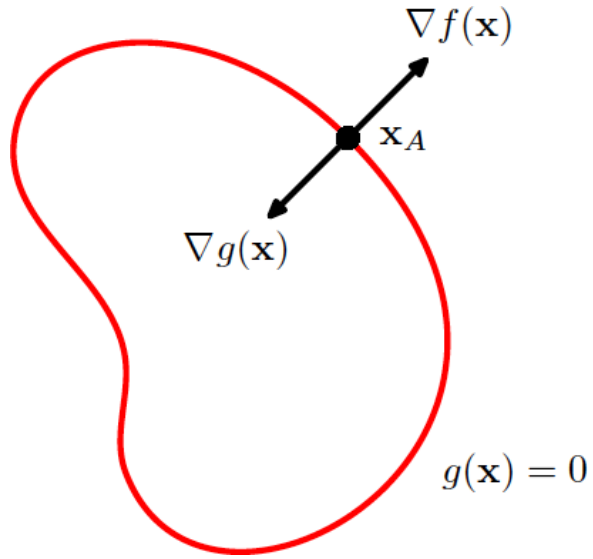
Lagrange Multipliers



- We first note that the gradient Δg is always perpendicular to g at any point $\mathbf{x}$ on the constraint line $g(\mathbf{x}) = 0$.
- Let $\mathbf{x}$ and $\mathbf{x} + \epsilon$ be two points that lie onto $g(\mathbf{x}) = 0$
- Using a Taylor expansion around $\mathbf{x}$ we have: $g(\mathbf{x} + \epsilon) \approx g(\mathbf{x}) + \epsilon^T \Delta g(\mathbf{x})$
- Since $g(\mathbf{x} + \epsilon) = g(\mathbf{x}) = 0$ (they both lie on the constraint line), it results that $\epsilon^T \Delta g(\mathbf{x}) = 0$ at $||\epsilon|| \to 0$
- Being ϵ parallel to the constraint line $g(\mathbf{x}) = 0$, it results that Δg is normal to the constraint line
- Same reasoning is also valid for the contour line of f. Δf is perpendicular to $f(\mathbf{x}) = d$

Lagrange Multipliers





- Remember that we look for points where $f(\mathbf{x}) = d$ and $g(\mathbf{x}) = 0$ are parallel, which means when Δf and Δg are also parallel !!
- This means looking for points $\mathbf{x}$ where $\Delta f = \lambda \Delta g$
- $\lambda \in \mathcal{R}$ is called *Lagrange multiplier* and we need it since the magnitude of the two gradients might be different. It can be positive or negative (Δf and Δg can have same or different orientation)

Lagrange Multipliers

- It is convenient to introduce the *Lagrangian function*:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}) \tag{4}$$

- we note that we obtain the constrained stationary condition $\Delta f = \lambda \Delta g$ and the constraint equation $g(\mathbf{x}) = 0$ by computing $\Delta_{\mathbf{x},\lambda} L$:

$$\begin{aligned} \Delta_{\mathbf{x}} L = 0 \rightarrow \Delta f + \lambda \Delta g = 0 \\ \Delta_{\lambda} L = 0 \rightarrow g(\mathbf{x}) = 0 \end{aligned} \tag{5}$$

- To find the maximum of a function $f(\mathbf{x})$ subject to $g(\mathbf{x}) = 0$, we define the Lagrangian function L and compute ΔL wrt $\mathbf{x}$ and λ. Furthermore, one should also verify that the Hessian $\Delta^2_{\mathbf{xx}} L$ is negative semi-definite

PCA

- By definition, we look for the direction u such that the variance of the projected data is maximized: $\arg\max_u u^T C u$
- We are only interested in the direction of u not in its magnitude. To prevent $||u||_2 \to \infty$ we choose a unit vector so that $||u||_2^2 = 1$

$$u_1^* = \arg\max_u u^T C u \quad \text{s.t.} \quad ||u||_2^2 = 1 \tag{6}$$

- The corresponding Lagrange equation is:

$$L(u, \lambda; C) = u^T C u + \lambda(1 - u^T u) \tag{7}$$

- By differentiating wrt u and λ and setting equal to 0 we obtain:

$$\begin{aligned} \Delta_u L = 2Cu - 2\lambda u = 0 \to Cu = \lambda u \\ \Delta_\lambda L = 1 - u^T u = 0 \to u^T u = 1 \end{aligned} \tag{8}$$

$$\underbrace{Cu = \lambda u}_{Eigenequation} \rightarrow u^T C u = \lambda \rightarrow \max(u^T C u) = \max \lambda \tag{9}$$

- In order to maximize $u^T C u$, i.e. projected variance, we need to compute the eigenvalues and eigenvectors of C and select the greatest eigenvalue λ_1 and the corresponding eigenvector u_1
- The second direction u_2 is the one that maximizes the projected variance among all possible directions **orthogonal** to u_1

$$u_2^* = \arg\max_u u^T C u \text{ s.t. } ||u||_2 = 1 \text{ and } \text{Cov}(|P_u(\{x_i\})|, |P_{u_1}(\{x_i\})|) = 0 \tag{10}$$

- NB $\text{Cov}(|P_u(\{x_i\})|, |P_{u_1}(\{x_i\})|) = u^T(Cu_1) = u^T(\lambda_1 u_1) = 0$, $\lambda_1 \neq 0$

$$L(u, \lambda, \theta; C, u_1) = u^T C u + \lambda(1 - u^T u) + \theta(u^T C u_1) \tag{11}$$

- By differentiating wrt u, λ, θ and setting equal to 0 we obtain:

$$\begin{aligned} &\nabla_u L = 2Cu - 2\lambda u + \theta C u_1 = 0 \\ &\nabla_\lambda L = 1 - u^T u = 0 \rightarrow u^T u = 1 \\ &\nabla_\theta L = u^T C u_1 = u_1^T C u = \lambda_1 u^T u_1 = \lambda_1 u_1^T u = 0, \quad \lambda_1 \neq 0 \end{aligned} \tag{12}$$

- If we left multiply the first Eq. by u_1^T we obtain $2u_1^T Cu - 2\lambda u_1^T u + \theta u_1^T C u_1 = 0$ which is equal to $0 - 0 + \theta\lambda_1 = 0$. It results that $\theta = 0$ and thus:

$$\nabla_u L = 2Cu - 2\lambda u = 0 \rightarrow Cu = \lambda u \tag{13}$$

PCA

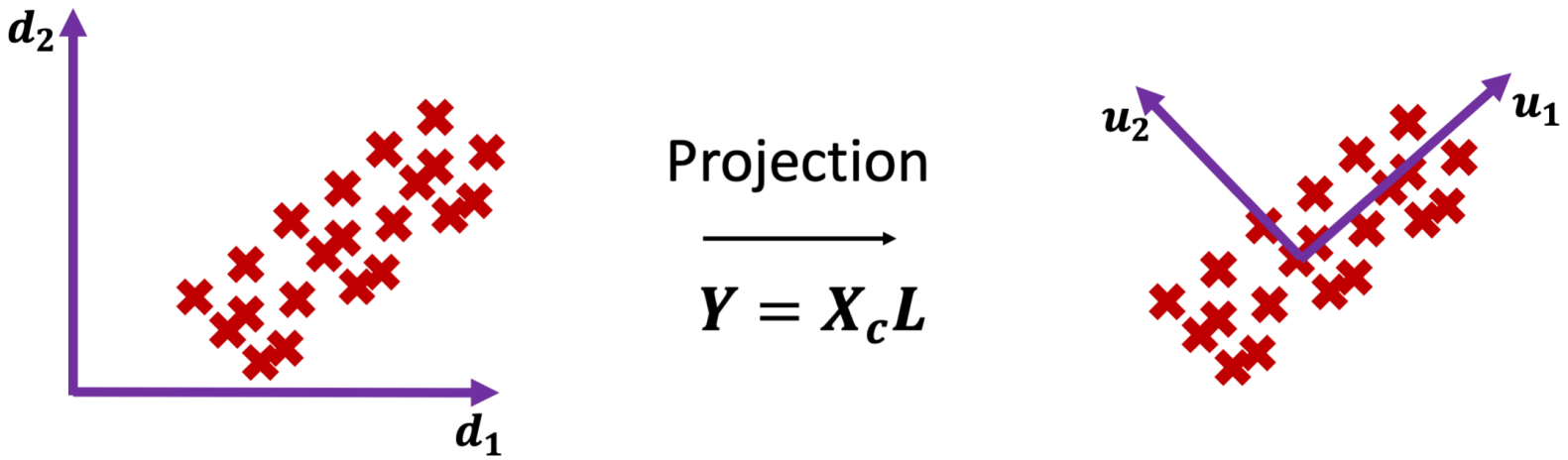
- The second direction u_2 is thus the second eigenvector (orthogonal to u_1) which corresponds to the second greatest eigenvalue λ_2
- We can repeat this process up to d times ($x_i \in \mathcal{R}^d$)
- Furthermore, note that C is a symmetric positive-semidefinite matrix with real entries. The finite-dimensional spectral theorem asserts that:
  1. C has always d linearly independent eigenvectors mutually orthogonal U ($U^T = U^{-1}$)
  2. All eigenvalues of C are real and non-negative and can we written as a diagonal matrix D

$$C = UDU^T \rightarrow CU = UD \tag{14}$$

$$C = UDU^T \rightarrow CU = UD \tag{15}$$

- Calling X_c the $[N, d]$ matrix where each row is a **centered** observation $x_i - \bar{x}$, we can compute the sample covariance matrix as $C = \frac{1}{N-1} X_c^T X_c$ where C is a $[d, d]$ matrix
- U is a $[d, d]$ matrix whose columns are the normalized ($u_j^T u_j = 1$) right eigenvectors of C ordered such that the first column represents the eigenvector relative to the greatest eigenvalue λ_1. It is an orthogonal matrix and it represents a linear transformation (either a rotation $\det(U) = +1$ or a reflection $\det(U) = -1$)
- D is a $[d, d]$ diagonal matrix whose entries are the eigenvalues λ of C (decreasing order)

PCA



- The eigenvectors U represent a new orthonormal basis such that the projected data has maximal variance
- $L = U(:, 1:k)$ is a $[d, k]$ matrix, where $k \leq d$, containing the *loadings*. If $k \neq d$, it is no more an orthogonal matrix.
- Y is a $[N, k]$ matrix containing the *scores*. Its columns are called Principal Components (PC) and they are uncorrelated since their covariance matrix is diagonal (i.e. $D(1:k, 1:k)$). Even if $k \neq d$.
- The first k PC explain $(\sum_{t=1}^{k} \lambda_t)/(\sum_{t=1}^{d} \lambda_t)$ of the total variability

PCA

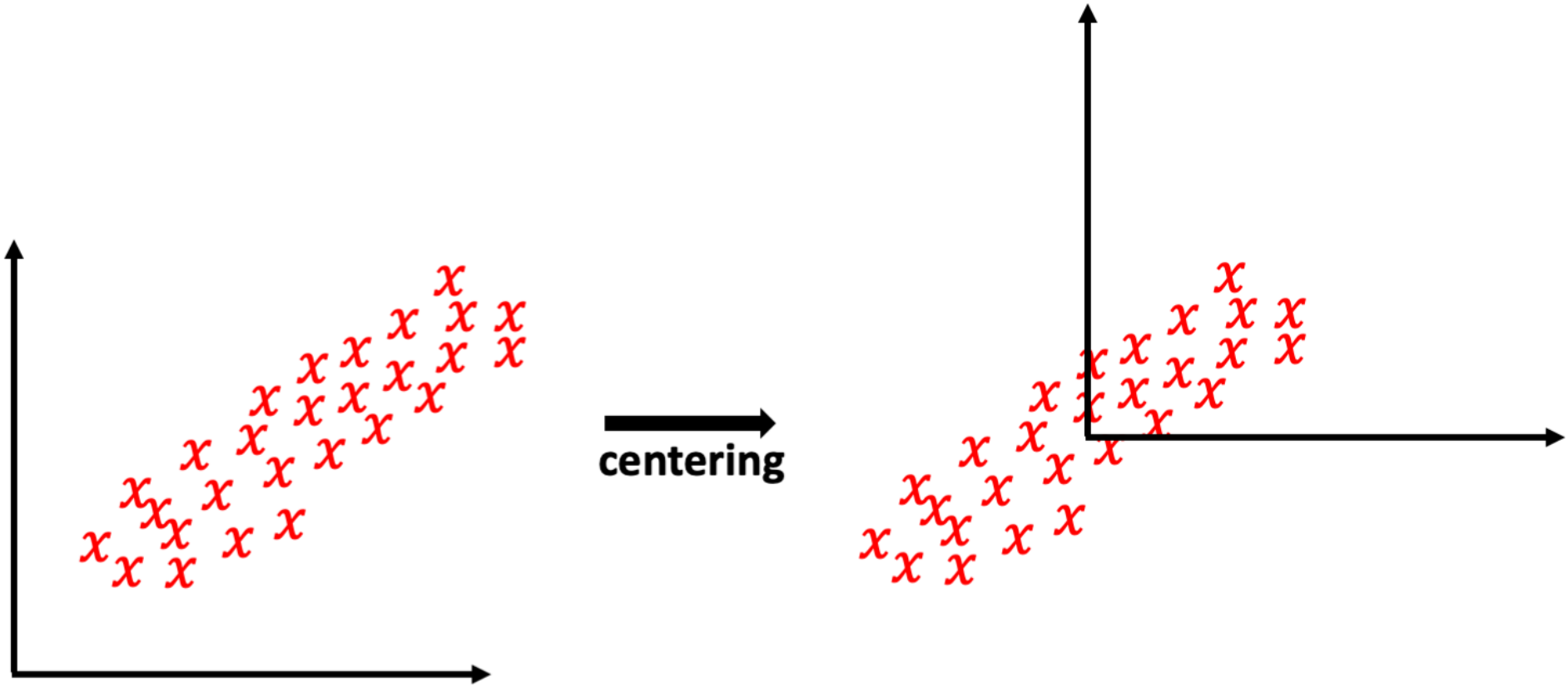
- How to compute PCA ?
  1. Center the data $X_c = X - \bar{X}$
  2. Use Singular Value Decomposition SVD (i.e. $X_c = R\Sigma W^T$ and so $X_c^T X_c = W\Sigma^2 W^T$)
- High-dimensional data ($d >> N$)
  - C has the same eigenvalues different from zero as $\tilde{C} = \frac{1}{N-1} X_c X_c^T$ which is a $[N, N]$ matrix.
  - The eigenvectors of C can be computed from the ones of $\tilde{C}$

$$\tilde{C}\tilde{U} = \tilde{U}D \rightarrow X_c^T \tilde{C}\tilde{U} = X_c^T \tilde{U} D \rightarrow$$
$$C(X_c^T \tilde{U}) = (X_c^T \tilde{U})D \rightarrow CU = UD$$

  - where $X_c^T \tilde{C} = (\frac{1}{N-1} X_c^T X_c) X_c^T = C X_c^T$. Thus, $U = X_c^T \tilde{U}$

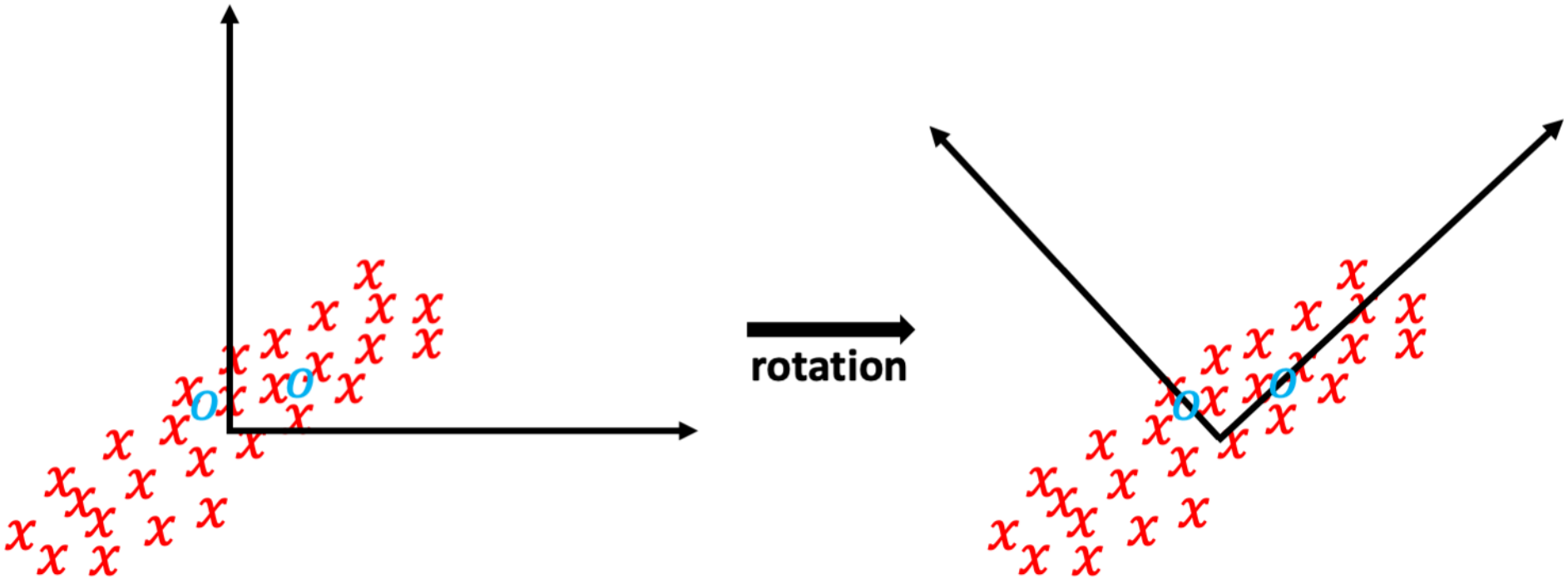
PCA - Visual explanation

- Let assume we have N images and that each image only contains two pixels. The first thing to do is to center the data
- This can be seen as translating the reference frame so that the origin of the coordinate system is at the center of the original data



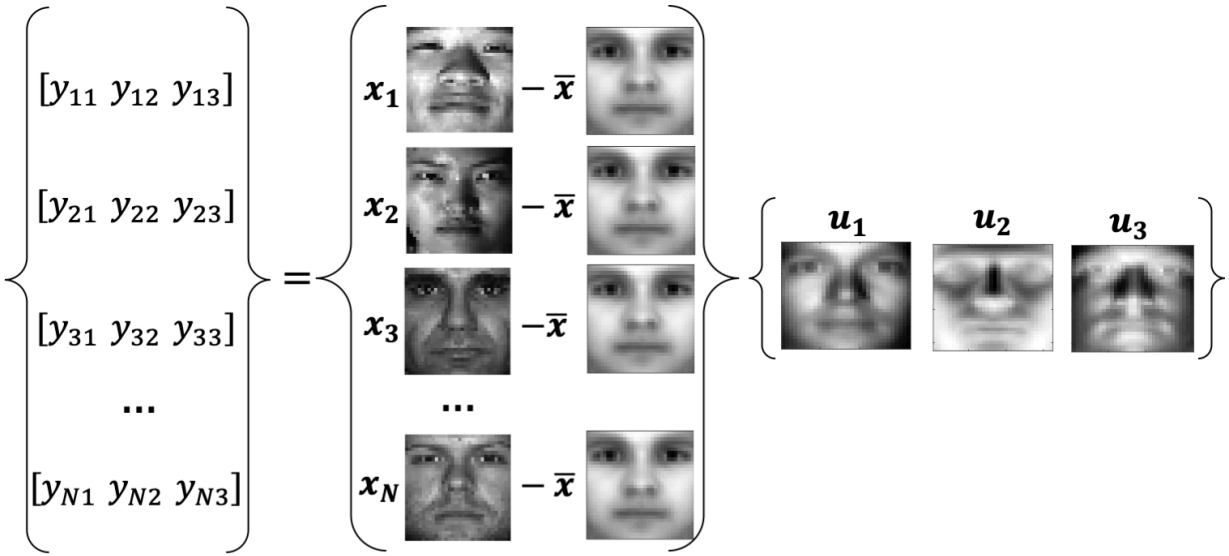
PCA - Visual explanation

- After that, we compute the eigenvectors of C which can be represented as points in the same space as the images (in this case 2D).
- Applying PCA to the data is like rotating the coordinate system so that the *new* coordinate directions connect the origin (original data average) and the eigenvectors
- Data can then be described with coordinates in this new reference frame



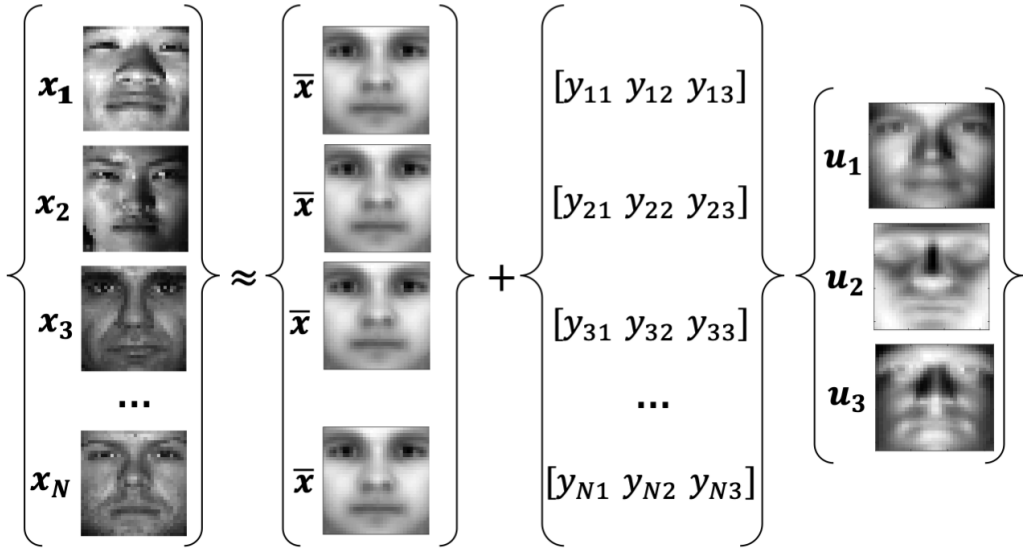
PCA - Visual explanation

- Let now assume that the N images have actually 1024 pixels and represent human faces.
- Let $Y = (X - \bar{X})L$ be the PCA projection. The rows of X contain the facial images (images have been vectorized).
- The columns of L contain, for instance, the first three eigenvectors. Remember that eigenvectors are also images !
- The rows of Y contains the projections of each image onto the new coordinate frames (the eigenvectors)

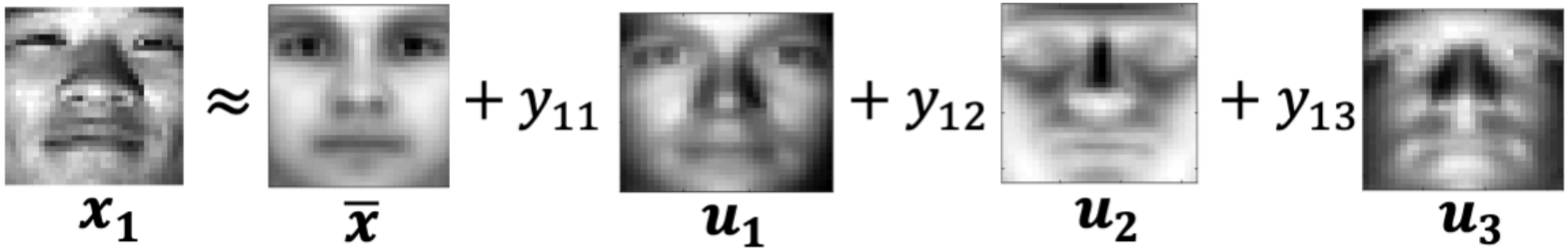


PCA - Visual explanation

- We can also use the PCA projections y and the eigenvectors u to approximate the original images $\rightarrow YU^T = (X - \bar{X})$ and so $X \approx \bar{X} + YL^T$

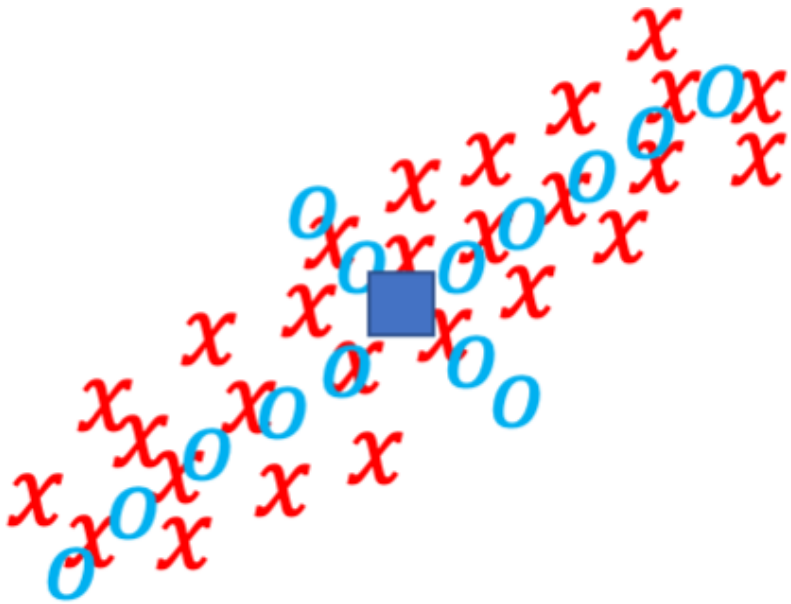


- The projection vector y_i contains the new coordinates of the image x_i in the new reference frame given by the three eigenvectors u_1, u_2 and u_3

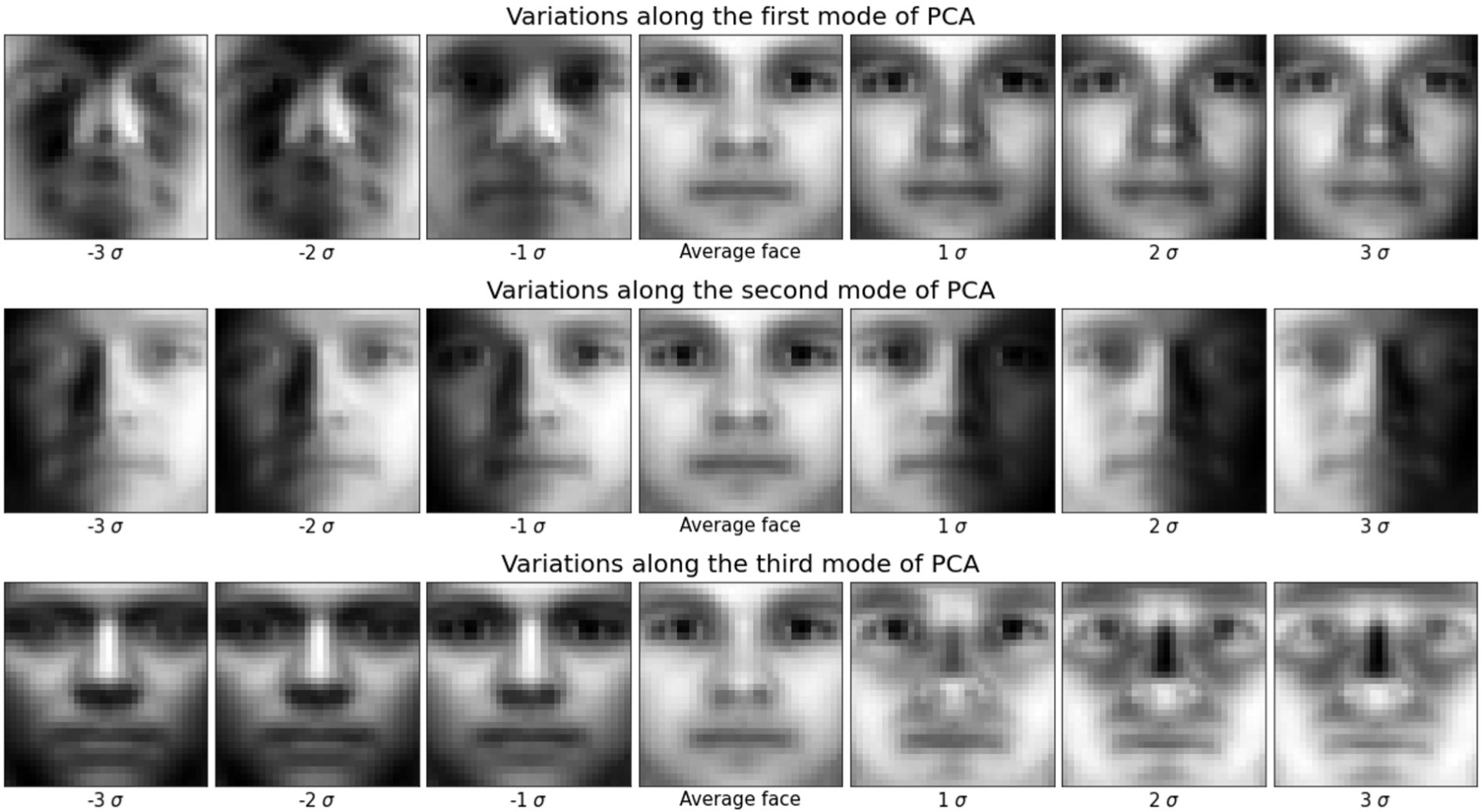


PCA - Visual explanation

- We could also use PCA to sample new images along the main directions of variations (new reference frame) assuming that data follows a Gaussian distribution
- We would use $x' = x_m \pm \alpha\sqrt{\lambda_t}u_t$ where x_m is the average image, α a scalar value usually between 1 and 3, λ_t the eigenvalue of the t eigenvector u_t



PCA - Visual explanation

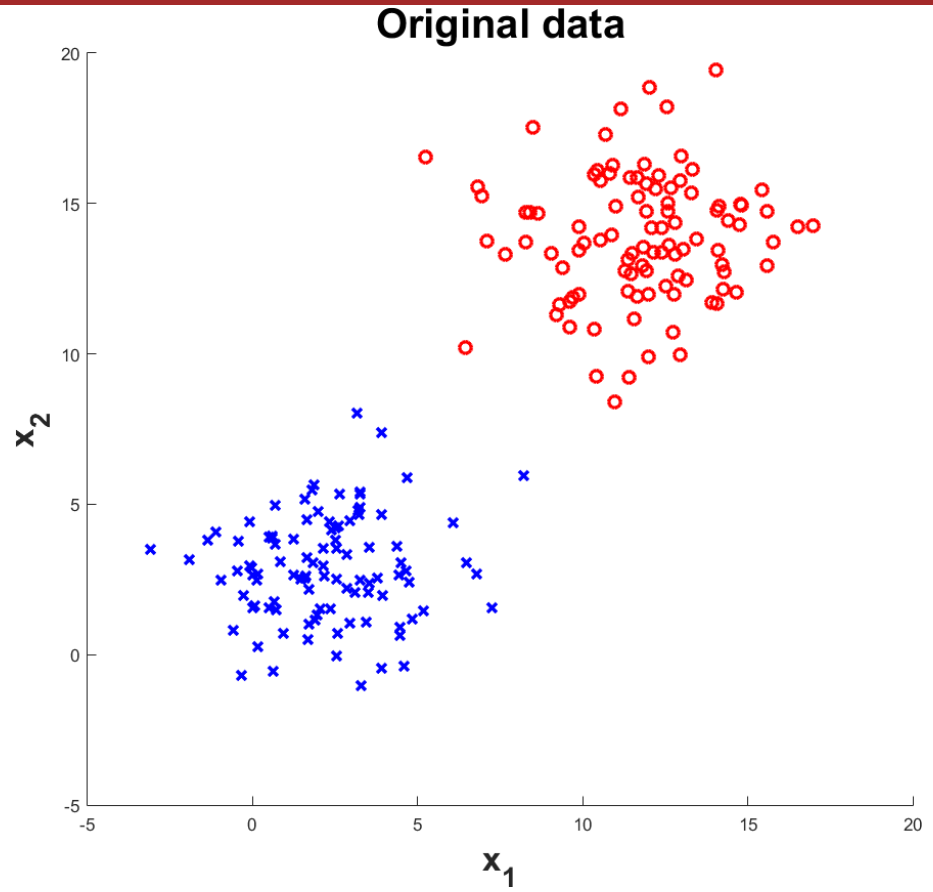


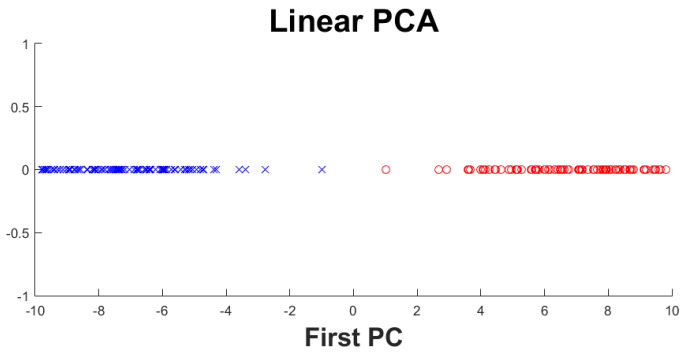
PCA

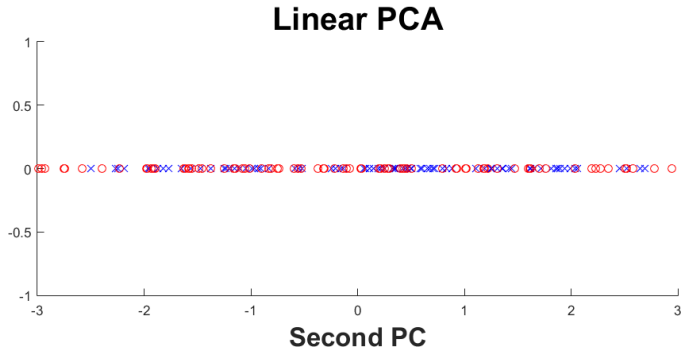
Assumptions and limitations

- **Linearity**: linear transformation of the data
- **Normality**: data are assumed to follow a Gaussian distribution (mean and variance are assumed to be sufficient statistics)
- **Large variance means interesting**: directions with largest variance are assumed to be the ones with most interest
- **Not scale invariant**: it depends on the scaling of the features

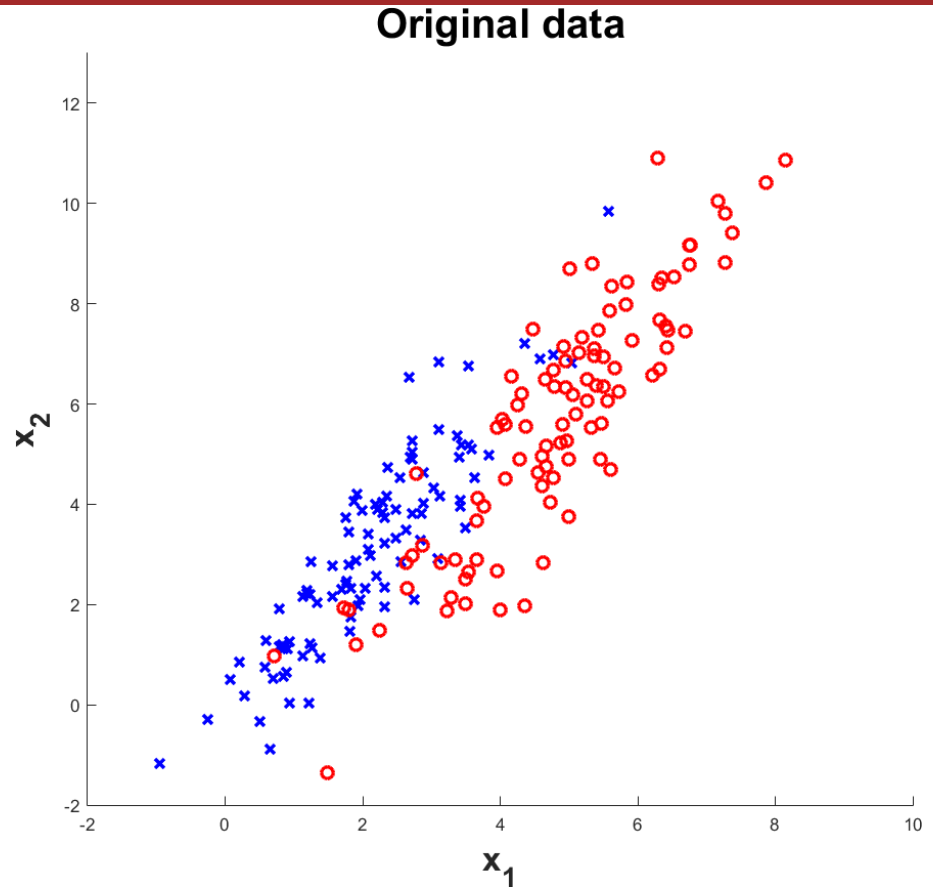
PCA - Example with two well separated sources

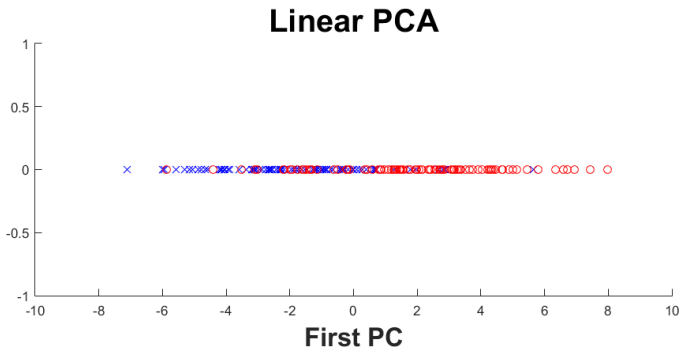


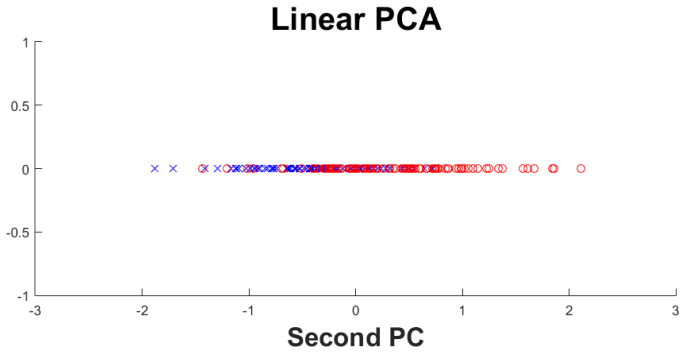




PCA - Example with two overlapping sources





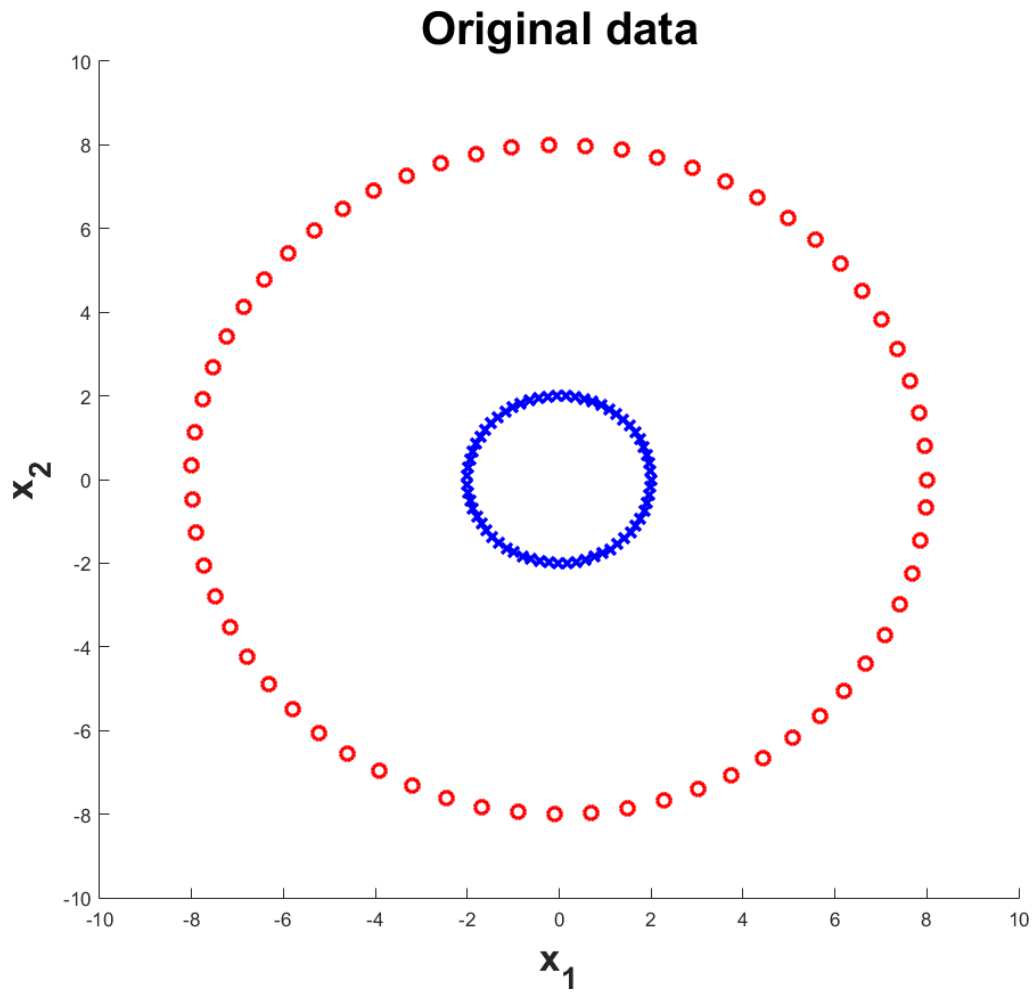


Summary

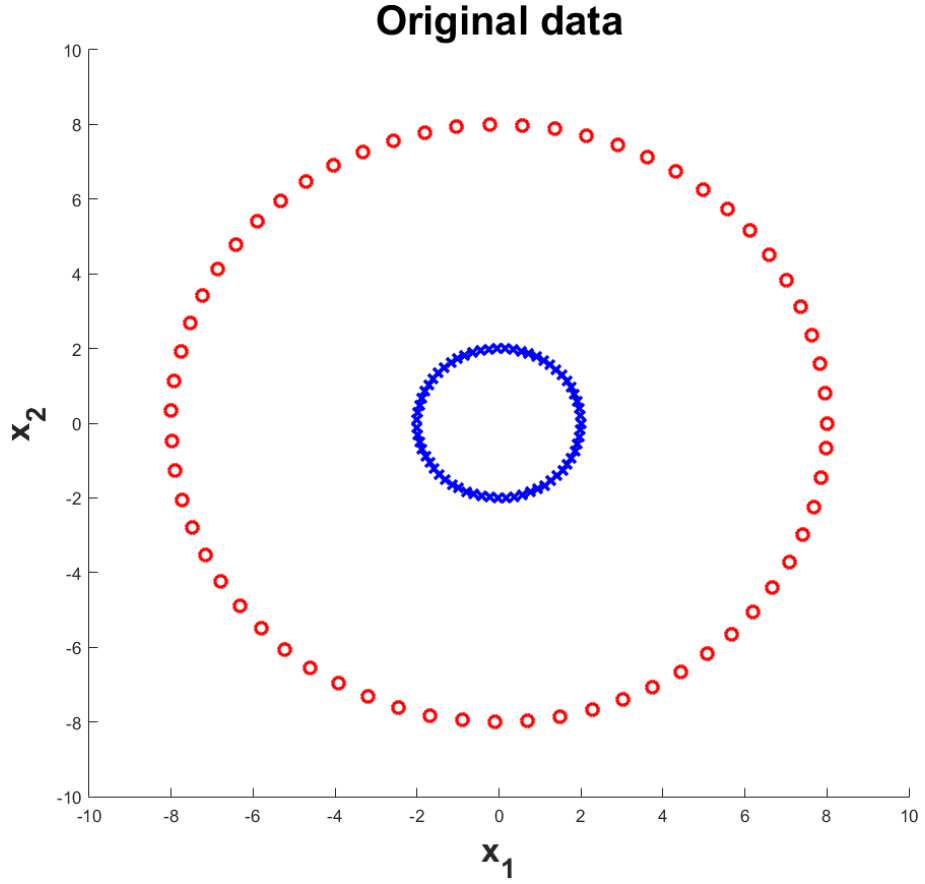
1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

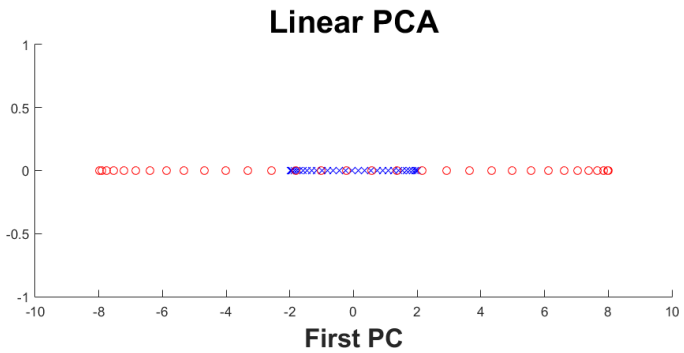
Kernel PCA

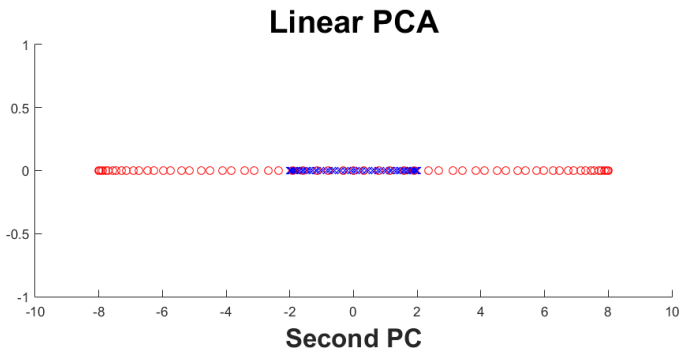
- What about this data-set ? Do you think that a PC (or a combination of PCs) of a linear PCA could well separate the two sources ? Why ?



PCA - Example with two concentric sources







Kernel PCA

- The idea of Kernel PCA is to map the x_i into a higher dimensional space $\mathcal{R}^f$ where sources (clusters) of the data could be well identified
- Let ϕ be a non-linear transformation:

$$\begin{aligned} \phi : \mathcal{R}^d &\to \mathcal{R}^f \\ x &\to \phi(x) \end{aligned}$$

- We assume that the mapped data has zero mean (i.e. $\sum_{i=1}^{N} \phi(x_i) = 0$)
- The sample covariance matrix of size $[f, f]$ of the mapped data is:

$$C = \frac{1}{N-1} \sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T \tag{16}$$

Kernel PCA

- As before, we could solve a PCA using the Eigenequation: $Cu = \lambda u$
- However, we do not know the space of mappings which could also be huge. We want to avoid working with it. We notice that:

$$\frac{1}{\lambda} Cu = u \rightarrow u = \frac{1}{\lambda(N-1)} \sum_{i=1}^{N} \phi(x_i) \left[\phi(x_i)^T u \right]$$

$$u = \sum_{i=1}^{N} \underbrace{\frac{1}{\lambda(N-1)} \left[\phi(x_i)^T u \right]}_{\alpha_i} \phi(x_i) = \sum_{i=1}^{N} \alpha_i \phi(x_i)$$

- Substituting it back into $Cu = \lambda u$, we obtain:

$$\frac{1}{N-1} \sum_{i=1}^{N} \phi(x_i) \left[\sum_{s=1}^{N} \alpha_s \phi(x_i)^T \phi(x_s) \right] = \lambda \sum_{i=1}^{N} \alpha_i \phi(x_i)$$

Kernel PCA

- Multiplying both sides by $\phi(x_j)^T$, we obtain:

$$\frac{1}{N-1}\sum_{i=1}^{N}\underline{\phi(x_j)^T\phi(x_i)}\left[\sum_{s=1}^{N}\alpha_s\underline{\phi(x_i)^T\phi(x_s)}\right]=\lambda\sum_{i=1}^{N}\alpha_i\underline{\phi(x_j)^T\phi(x_i)}$$

- We can notice that we only need to compute the inner products between mappings. We do not need to compute the actual mappings (Kernel trick) ! Calling $k(x_i,x_j)=\phi(x_i)^T\phi(x_j)$, it results:

$$\frac{1}{N-1}\sum_{i=1}^{N}\underline{k(x_j,x_i)}\left[\sum_{s=1}^{N}\alpha_s\underline{k(x_i,x_s)}\right]=\lambda\sum_{i=1}^{N}\alpha_i\underline{k(x_j,x_i)}$$

Kernel PCA

$$\frac{1}{N-1}\sum_{i=1}^{N}\underline{k(x_j,x_i)}\left[\sum_{s=1}^{N}\alpha_s\underline{k(x_i,x_s)}\right]=\lambda\sum_{i=1}^{N}\alpha_i\underline{k(x_j,x_i)}$$

$$\frac{1}{N-1}\sum_{s=1}^{N}\alpha_s\sum_{i=1}^{N}k(x_j,x_i)k(x_i,x_s)=\lambda\sum_{i=1}^{N}\alpha_i k(x_j,x_i)$$

- Calling $\boldsymbol{\alpha}=\{\alpha_1,\alpha_2,...,\alpha_N\}$ and $\mathbb{K}$ the $[N,N]$ matrix whose entry at row i and column j is $k(x_i,x_j)$, we can rewrite the previous Eq. as:

$$\mathbb{K}^2\boldsymbol{\alpha}=\lambda(N-1)\mathbb{K}\boldsymbol{\alpha}\rightarrow\mathbb{K}\boldsymbol{\alpha}=\lambda(N-1)\boldsymbol{\alpha}$$

- This is an Eigenequation. The resulting $\boldsymbol{\alpha}$ is normalized such that $1=u^Tu=\left(\sum_{i=1}^{N}\alpha_i\phi(x_i)\right)^T\left(\sum_{i=1}^{N}\alpha_i\phi(x_i)\right)=\boldsymbol{\alpha}^T\mathbb{K}\boldsymbol{\alpha}=\lambda(N-1)\boldsymbol{\alpha}^T\boldsymbol{\alpha}$

Kernel PCA

- Calling $\rho = \lambda(N-1)$ the eigenvalues of $\mathbb{K}$, we can solve the Eigenequation $\mathbb{K}\boldsymbol{\alpha} = \rho\boldsymbol{\alpha}$ finding the N eigenvectors $\{\boldsymbol{\alpha}_i\}_1^N$ and the corresponding eigenvalues $\{\rho_i\}_1^N$ in decreasing order.
- We normalize every eigenvector such that $\boldsymbol{\alpha}_i = \frac{\boldsymbol{\alpha}_i}{\sqrt{\rho_i}}$
- Then, calling y^l the l-th PC of the sample x, it is equal to:

$$y^l = (u^l)^T \phi(x) = \sum_{i=1}^{N} \alpha_i^l k(x, x_i) \tag{17}$$

- where α_i^l is the i-th element of the l-th eigenvector.

Kernel PCA

- The Kernel trick $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is possible only when the function k is symmetric and it is a kernel, namely it satisfies the Mercer's condition.
- **Mercer's condition**: Let k be a symmetric function $\mathcal{X} \times \mathcal{X} \to \mathcal{R}$ and $L_2(\mathcal{X})$ the Hilbert space of square integrable functions in $\mathcal{X}$, k is a kernel iif:

$$\int_{\mathcal{X} \times \mathcal{X}} k(x, x') f(x) f(x') dx dx' \geq 0 \quad \forall f \in L_2(\mathcal{X}) \tag{18}$$

- It can be shown that, if $\mathcal{X}$ is a finite space composed of N elements x_i, this condition is equivalent of saying that the associated square matrix $\mathbb{K}$ must be symmetric semi-definite positive

$$c^T \mathbb{K} c \geq 0 \quad \forall c \in \mathcal{R}^N \tag{19}$$

- Examples of kernels: $k(x, y) = \exp \frac{||x-y||^2}{\sigma^2}$ and $k(x, y) = \langle x, y \rangle^p$

Kernel PCA

Few considerations

- We do not compute the mapping ϕ. We only define its inner product using a kernel.
- We only need to compute $\mathbb{K}$ for the N points $\{x_i\}$ present in the data-set. Computational complexity depends on the number of samples N and it does not grow if we increase d !
- At the beginning we made the hypothesis that the mapped data had zero mean. This is not always true. It can be shown that the matrix kernel $\tilde{\mathbb{K}}$ of the centered projected data $\tilde{\phi(x_i)} = \phi(x) - \sum_{i=1}^{N} \phi(x_i)$ only depends on the kernel matrix $\mathbb{K}$:

$$\tilde{\mathbb{K}} = \mathbb{K} - \mathbb{1}_n \mathbb{K} - \mathbb{K}\mathbb{1}_n + \mathbb{1}_n \mathbb{K} \mathbb{1}_n \tag{20}$$

- where $\mathbb{1}$ a $[N \times N]$ matrix whose values are equal to $1/N$:

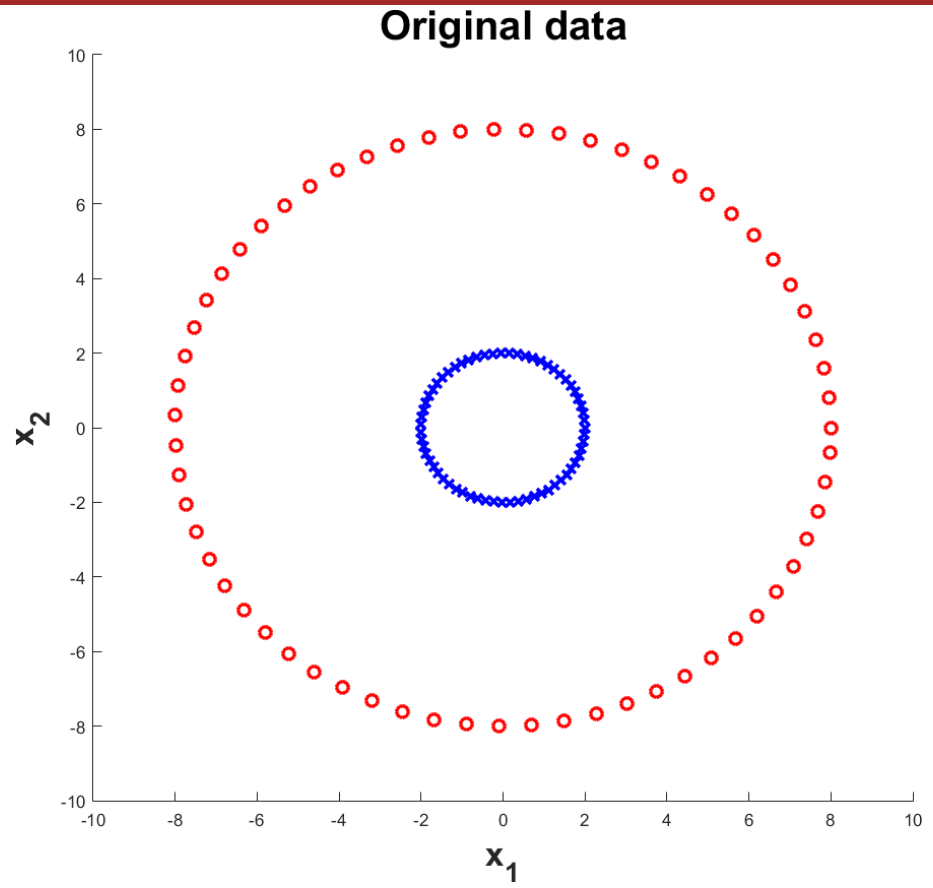
Kernel PCA

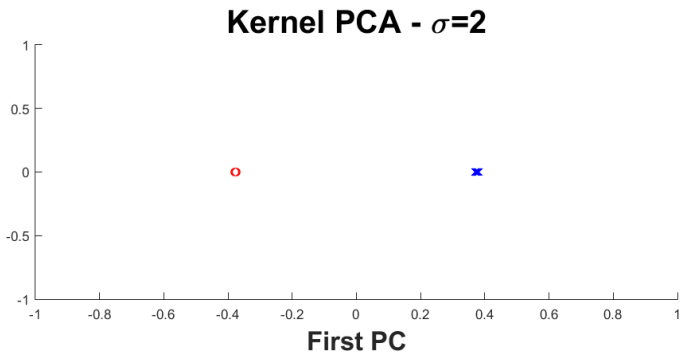
Recipe

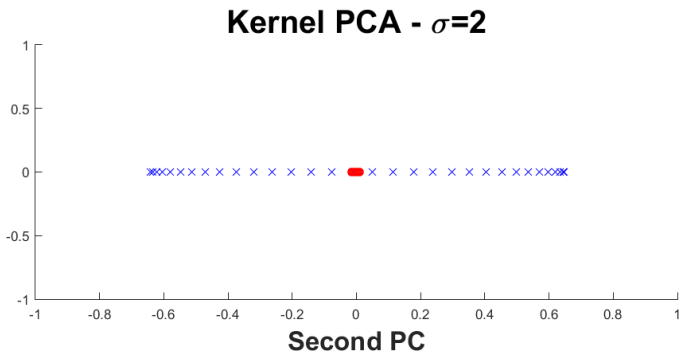
1. Compute the kernel matrix $\mathbb{K}$ between all samples x_i
2. Center the kernel matrix $\tilde{\mathbb{K}} = \mathbb{K} - \mathbb{1}_n\mathbb{K} - \mathbb{K}\mathbb{1}_n + \mathbb{1}_n\mathbb{K}\mathbb{1}_n$
3. Solve the Eigenequation $\tilde{\mathbb{K}}\boldsymbol{\alpha} = \rho\boldsymbol{\alpha}$ normalizing every eigenvector such that $\boldsymbol{\alpha}_i = \frac{\boldsymbol{\alpha}_i}{\sqrt{\rho_i}}$
4. For a test sample t compute $y^l = \sum_{i=1}^{N} \alpha_i^l \tilde{k}(t, x_i)$ where $\tilde{k}(t, x_i) = \tilde{\phi(t)}^T \tilde{\phi(x_i)}$

Warning: In linear PCA the sample t can be reconstructed as $\hat{t} = \sum_{j=1}^{k}(t^T u_j)u_j$. In KPCA we only know the projections of $\phi(t)$ onto the first principal components in $\mathcal{R}^f$. Techniques to find an approximate pre-image of t in $\mathcal{R}^d$ have been proposed [Bakir et al., 2004]

KPCA - Example with two concentric sources







Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

ICA - Independent Component Analysis

- We start from the classical *cocktail-party problem* (or *source separation problem*).
- There are 3 people speaking simultaneously and two microphones (in different positions) which record 2 time signals: $x_1(t)$ and $x_2(t)$. These are modeled as weighted sums of the speech signals of the 3 people $s_1(t)$, $s_2(t)$ and $s_3(t)$.

$$x_i(t) = \sum_{j=1}^{3} a_{ij} s_j(t) \tag{21}$$

- Goal is to estimate all $s_j(t)$ and a_{ij}
- We do not know $a_{ij} \rightarrow$ we need prior information
- ICA assumption: $s_j(t)$ are statistically independent at each time t

ICA - Independent Component Analysis

- With 4 time points, we can rewrite the previous problem in matrix notation where x_{it} is the sample of the *i-th* microphone at time point t:

$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \end{bmatrix}}_{X:[d,N]} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}}_{A:[d,p]} \underbrace{\begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \end{bmatrix}}_{S:[p,N]}$$

- Each row s_j^T of S is a source signal which contains N independent realizations of a random variable s_j
- Furthermore, we also assume that all p random variables s_j are mutually independent
- Thus, each row x_i^T of X is a linear mixture of the unknown independent source signals s_j^T and it also contains independent realizations of a random variable x_i

ICA - Independent Component Analysis

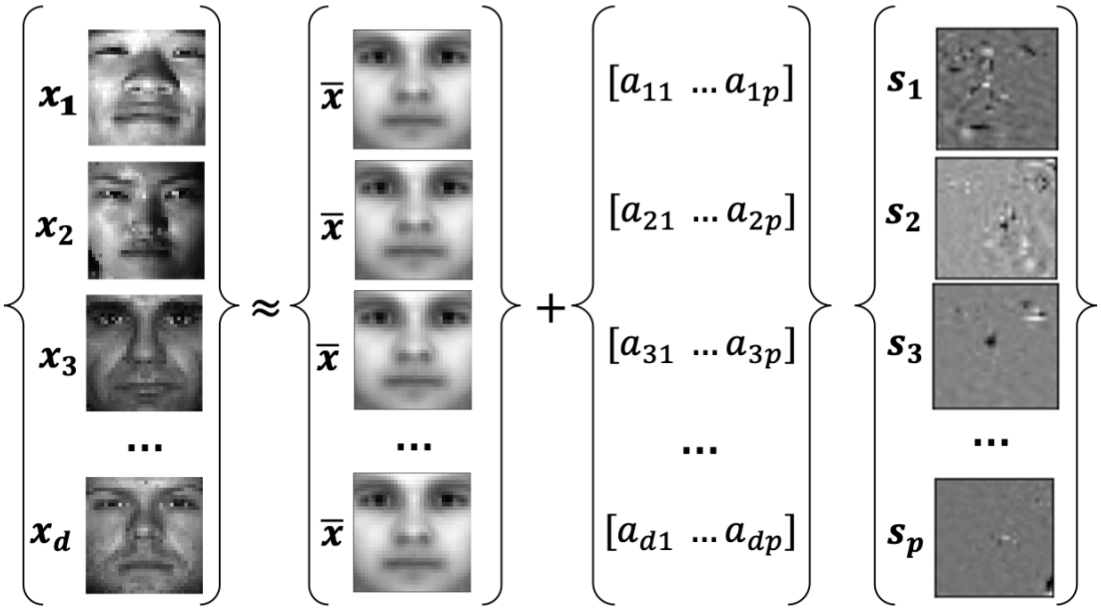
$$X = AS$$

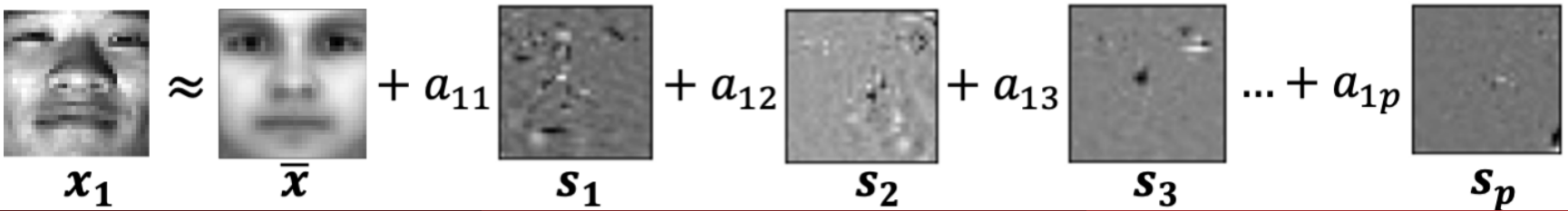
- We usually make the assumption that $d = p$
- Two possible ways for using ICA with images:
  1. Rows of X are images and columns are pixels. Images are considered as random variables and we look for a set of statistically independent basis images contained in the rows of S.
  2. Column of X are images and rows are pixels. Pixels are considered as random variables and we look for a set of statistically independent coefficients contained in the rows of S and a set of basis images in the columns of A.

ICA - Visual explanation

First architecture

Rows of X are images and columns are pixels. Images are considered as random variables and we look for a set of statistically independent basis images contained in the rows of S.

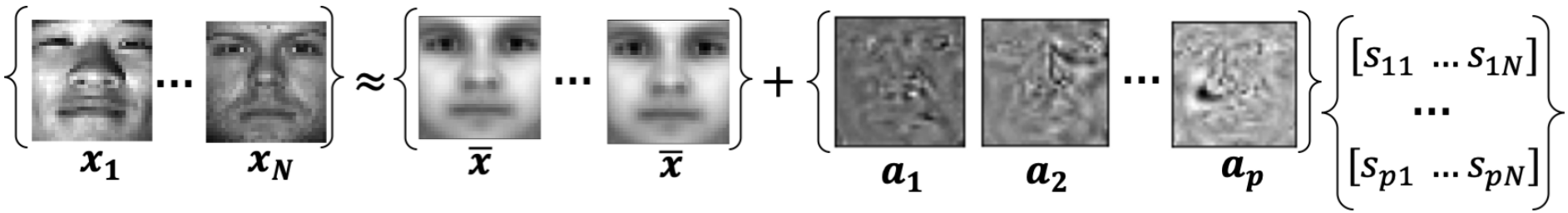


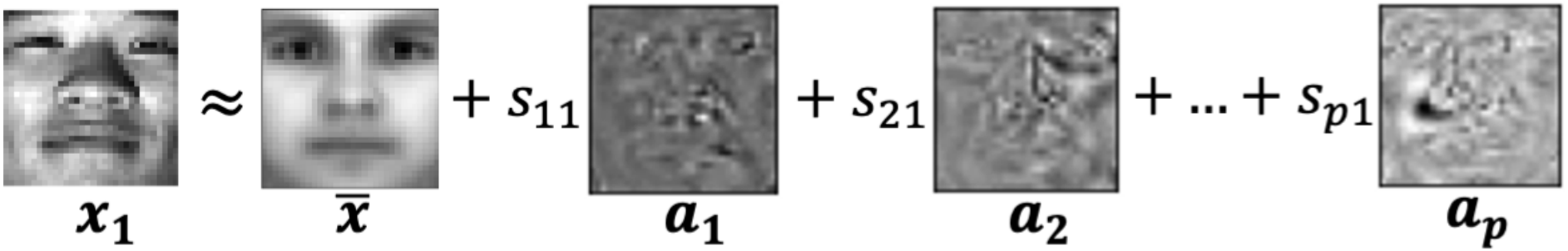


ICA - Visual explanation

Second architecture

Column of X are images and rows are pixels. Pixels are considered as random variables and we look for a set of statistically independent coefficients contained in the rows of S and a set of basis images in the columns of A.





ICA

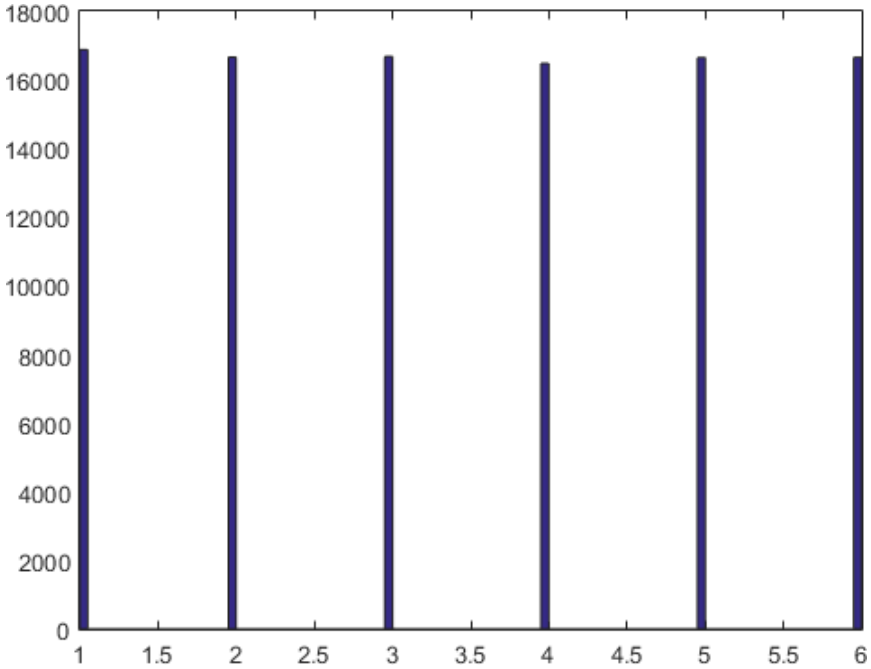
- **Goal**: Estimate A (the mixing matrix) and S knowing X
- **Assumptions to make the model identifiable**:
  1. All s_j^T are mutually statistically independent
  2. All s_j^T are *not* Gaussian (actually at most one...)
  3. A is invertible
- **Ambiguities**:
  - Estimation of A and S is up to a scaling $X = AS = (AC^{-1})(CS)$ with $C = cI$ where c is a scalar and I is the identity matrix.
  - For this reason, we fix the variance of every s_j^T by assuming $E[s_j^2] = 1 \rightarrow E[SS^T] = I$. However, we still have a sign ambiguity: one could multiply s_j^T and a_{ij} by -1
  - Being a sum, the s_j^T are not in a particular order $\rightarrow$ swapping the s_j^T would not change the result

Why the s_j can not be Gaussian ?

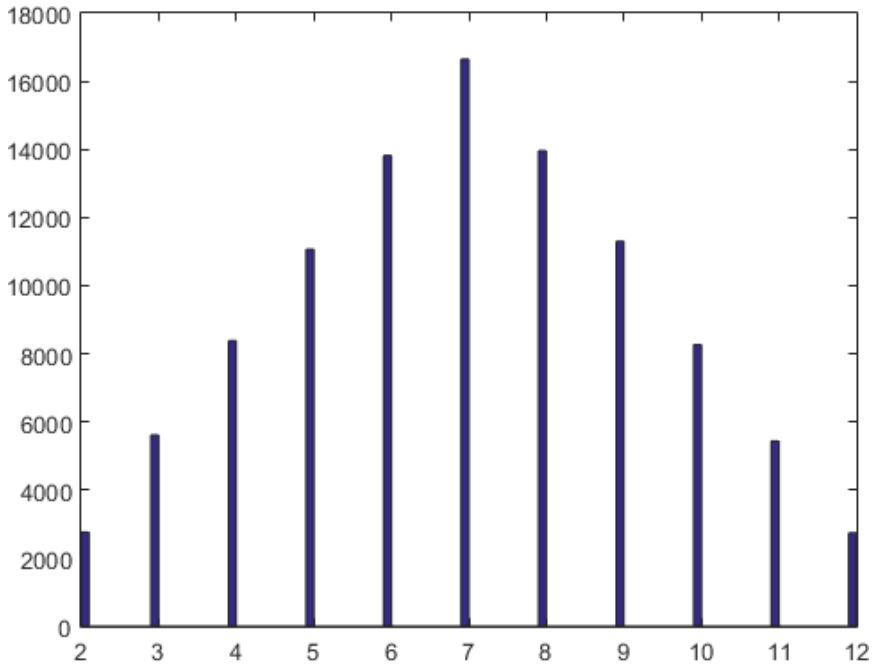
- We observe $X = AS$. If $S \sim \mathcal{N}(0, I)$, it follows that $X \sim \mathcal{N}(0, AA^T)$
- Let R be a rotation matrix (i.e. $R^T = R^{-1}$), we define $B = AR$
- If the data were mixed with B instead than A, we would observe $Y = BS$ where $Y \sim \mathcal{N}(0, BB^T)$
- Note that $BB^T = AA^T$! This means that there is an arbitrary rotation in the mixing matrix that cannot be determined from the data. The matrices S and A could not be recovered.

ICA

- **Central limit theorem**: the sum of independent and identically distributed random variables (with finite variance) tend to a normal distribution as the number of variables grows.
- Let $x_i = a_{i1}s_1 + a_{i2}s_2$ and assume that both s_1 and s_2 follow a uniform distribution and can take integer values between 1 and 6 (i.e. a dice)
- Their sum follows a Gaussian distribution centered in 7



Distribution of s_1 and s_2



Distribution of $s_1 + s_2$

ICA

- Let $W = A^{-1}$ so that $S = WX$
- Let w_j^T be the j-th row of W so that $s_j^T = w_j^T X$
- Our goal is to estimate W and S knowing X and using the Central Limit Theorem
- We define $z_j^T = w_j^T A$ and we call y^T an estimate of one of the s_j^T:

$$y^T = w_j^T X = w_j^T AS = z_j^T S = z_{j1} s_1^T + z_{j2} s_2^T + ... + z_{jM} s_M^T \qquad (22)$$

- y^T is a linear combination of all s_j^T and therefore must be "more" Gaussian than the s_j^T since we assume that all s_j^T are *not* Gaussian
- y^T becomes less Gaussian only when it is exactly equal to one of the s_j^T, namely only one z_j^T is nonzero

ICA

- **Goal**: The goal becomes thus to estimate w_j^T such that y^T is the least Gaussian
- Measures of Non-Gaussianity:
  1. **Kurtosis**: $kurt(y) = E[y^4] - 3(E[y^2])^2$
     - Equal to 0 when y is Gaussian
     - It can be positive or negative
     - Sensitive to outliers, not robust
  2. **Negentropy**: $J(y) \approx (E[G(y)] - E[G(\beta)])^2$
     - $\beta \sim \mathcal{N}(0,1)$
     - $G(t) = -\exp\left(-\frac{1}{2}t^2\right)$ or $G(t) = \log(\cosh(t))$
     - More robust than *Kurtosis*
- Negentropy is usually preferred. Thus one needs to estimate the w_j^T that maximizes $J(y)$

$$\arg\max_{w_j} J(w_j^T X) \quad \text{s.t. } E[(w_j^T X - w_j^T \bar{X})(w_j^T X - w_j^T \bar{X})^T] = 1 \qquad (23)$$

- **Usual preprocessing**:
  - Centering: $X_C = X - \bar{X}$
  - Whitening: $X \to X_W$ s.t. $E[X_W X_W^T] = I$
    - Whitening is a linear transformation that makes the covariance matrix of the observations an identity matrix $\to$ transformed observations become white noise, so uncorrelated and with variance equal to one
    - Given X_C and Σ, the covariance matrix of X, we define X_W as $X_W = \mathcal{W} X_C$ where the whitening matrix $\mathcal{W}$ satisfies $\mathcal{W}^T \mathcal{W} = \Sigma^{-1}$
    - Common choices of $\mathcal{W}$ are: $\mathcal{W} = \Sigma^{-\frac{1}{2}}$, the Cholesky decomposition or the eigen-decomposition of Σ

$$\arg\max_{w_j} J(w_j^T X_W) = (E[G(w_j^T X_W)] - E[G(\beta)])^2 \quad \text{s.t. } ||w_j||_2^2 = 1 \quad (24)$$

- **First derivative**: we call $g = \frac{dG}{dw_j}$, we note that $E[G(\beta)] = 0$ and we use a Lagrangian multiplier λ

$$J' = \frac{dJ}{dw_j} = E[X_W g(w_j^T X_W)] - \lambda w_j \quad (25)$$

- **Second derivative**

$$\begin{aligned} J'' &= E[X_W X_W^T g'(w_j^T X_W)] - \lambda \\ &\approx E[X_W X_W^T] E[g'(w_j^T X_W)] - \lambda \\ &\approx E[g'(w_j^T X_W)] - \lambda \end{aligned} \quad (26)$$

- We can also note that $\lambda = E[w_j^T X_W g(w_j^T X_W)]$ when w is at the optimum

- Using Newton's method, the update step is equal to:

$$\begin{aligned} w_j^{n+1} &= w_j^n - (J''(w_j^n))^{-1} J'(w_j^n) \\ &= w_j^n - (E[g'(w_j^T X_W)] - \lambda)^{-1} (E[X_W g(w_j^T X_W)] - \lambda w_j) \end{aligned} \tag{27}$$

- If we multiply both sides by $(\lambda - E[g'(w_j^T X_W)])$ and use $\lambda = E[w_j^T X_W g(w_j^T X_W)]$ with the current estimate of w_j (approximation) we obtain the following algorithm called **Fast-ICA**:

$$w_j^{n+1} = E[X_W g(w_j^T X_W)] - E[g'(w_j^T X_W)] w_j \tag{28}$$

- where we normalize at every step: $w_j^{n+1} = \frac{w_j^{n+1}}{||w_j^{n+1}||_2}$ to improve stability

ICA

- Using the previous algorithm, we iteratively and independently estimate each w_j
- It depends on the initialization and the estimate of some components w_j might not converge
- With a different perspective, the **symmetric FastICA** estimates all w_j in parallel, and each step is completed by a symmetric orthonormalization
  - Center and whiten the matrix X
  - Choose a random orthogonal matrix W
  - Iterate until $1 - \min(|diag(W^T W_{\text{old}})|) < \epsilon$
    1. $W^+ = g(WX_W)X_W^T - diag[g'(WX_W)1_N]W$
    2. $W = (W^+(W^+)^T)^{-1/2}W^+$
  - where 1_N is a column vector of ones of size N

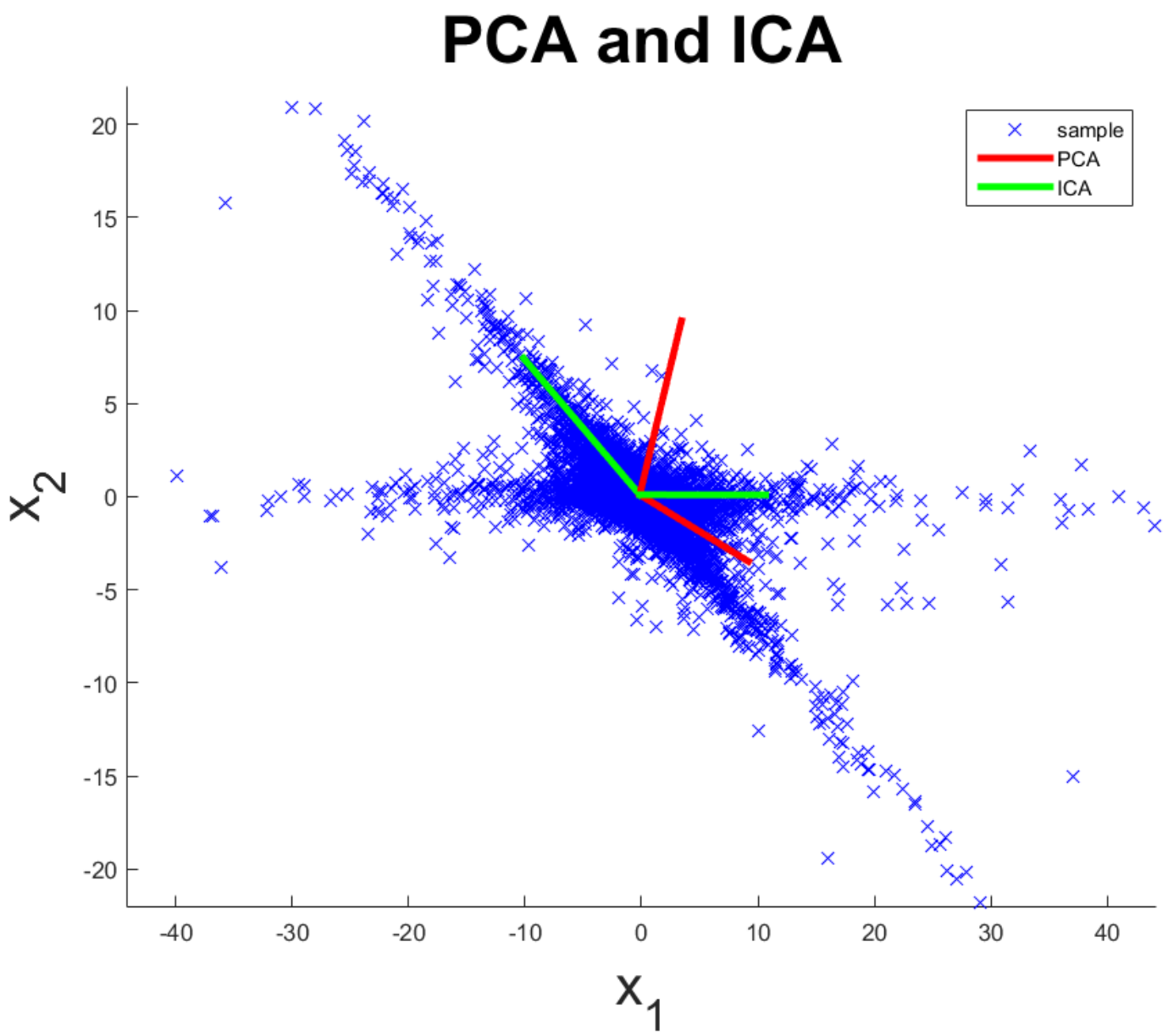
PCA and ICA

PCA

- Second order method, it only uses covariance matrix
- Data are assumed to follow a Gaussian distribution
- Linear transformation that yields a new orthogonal basis
- Directions with the maximal projected variance are assumed to be more interesting

ICA

- Higher-order method, not only covariance matrix
- Data are **not** assumed to follow a Gaussian distribution
- Linear transformation that yields a new basis which is **not** necessarily orthogonal
- Directions are mutually statistically independent and equally important


PCA and ICA
sample
PCA
ICA
20
15
10
5
0
-5
-10
-15
-20
-40
-30
-20
-10
0
10
20
30
40
x2
x1

Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

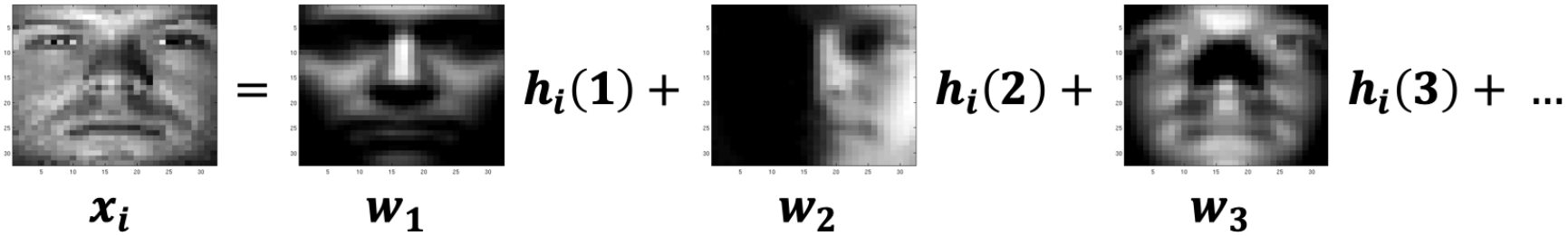
NNMF

$$X = WH \rightarrow x_i = Wh_i \rightarrow x_i = \sum_{j=1}^{r} w_j h_i(j) \quad \text{with } x_i(\cdot), w_j(\cdot), h_i(\cdot) \geq 0$$

Property: All matrices X, W and H have no negative elements

Where:

- $X \in \mathcal{R}^{d \times N}$ is a matrix whose columns x_i are observations (i.e. images)
- $W \in \mathcal{R}^{d \times r}$ is a matrix whose columns w_j are basis vectors
- $H \in \mathcal{R}^{r \times N}$ is a matrix whose columns h_i are the coefficients (or loadings)



NNMF

- **Goal**: Estimate W and H knowing X
- **Remarks**:
  1. Each observation x_i is approximated with a purely additive combination of non-negative bases w_j
  2. Basis w_j represents parts of the data
  3. Basis w_j is either present ($h_i(j) > 0$) or not ($h_i(j) = 0$)
- **Ambiguities**:
  - Factorization is not unique. Given a non-negative generalized permutation matrix P, we can see that: $WH = (WP)(P^{-1}H)$. This means that if we scale and permute columns of W and rows of H the approximation does not change ! $\rightarrow$ this is usually tackled adding priors and regularization terms to the objective function

NNMF

$$\underset{W\geq 0, H\geq 0}{\arg\min}\ F(W,H) = \frac{1}{2}||X - WH||_F^2 = \frac{1}{2}\sum_{ij}(X_{ij} - (WH)_{ij})^2 \qquad (29)$$

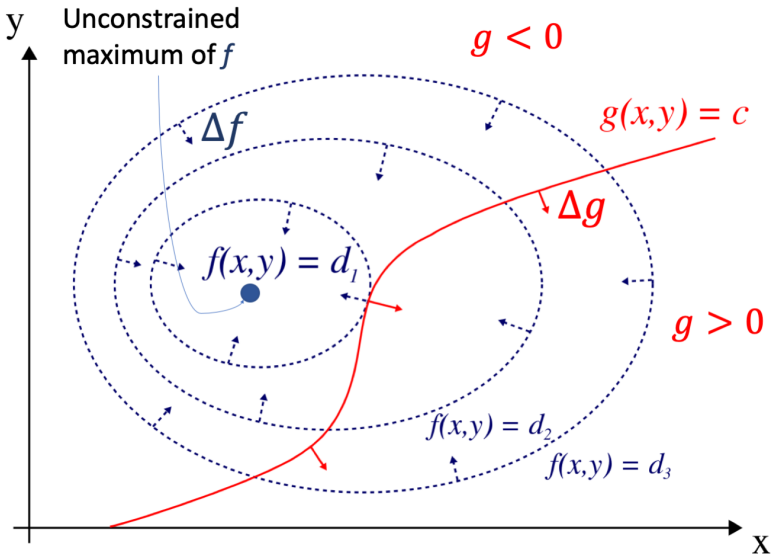
- Using L2-norm we assume additive Gaussian noise (i.e. $X \approx WH + \mathcal{N}$)
- This is a non-convex problem (it is convex only for W and H separately) with inequality constraints ! How do we solve it ?

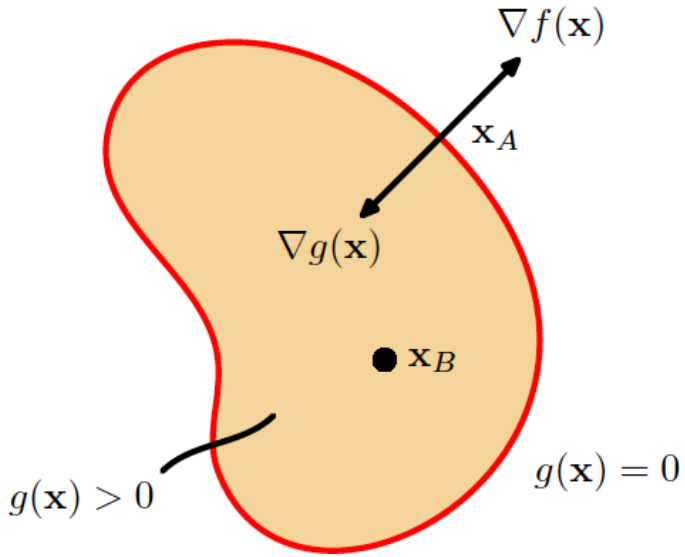
Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

Lagrange Multipliers

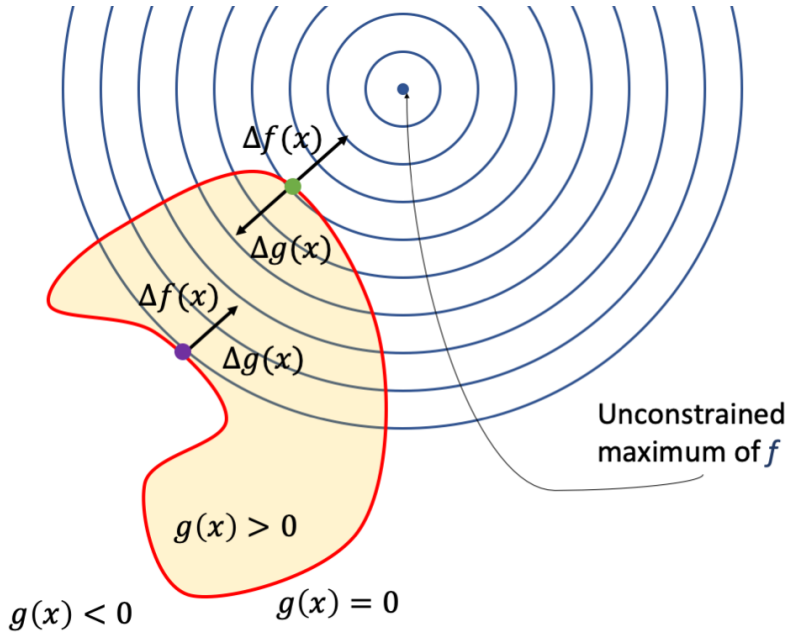
- Previously, we have only looked at *equality constraint* (i.e. $g(\mathbf{x}) = 0$), what about *inequality constraint* (i.e. $g(\mathbf{x}) \geq 0$ or $g(\mathbf{x}) \leq 0$) ? Two cases:
  1. unconstrained maximum (or minimum) of $f(\mathbf{x})$ lies **within** the feasible region $g(\mathbf{x}) \geq 0$ (or $g(\mathbf{x}) \leq 0$) $\rightarrow$ $g(\mathbf{x})$ plays no role, stationary point occurs at $\nabla f(\mathbf{x}) = 0$ with $\lambda = 0$
  2. unconstrained maximum (or minimum) of $f(\mathbf{x})$ lies **outside** the feasible region $g(\mathbf{x}) \geq 0$ (or $g(\mathbf{x}) \leq 0$) $\rightarrow$ constrained stationary point are at the boundary $g(\mathbf{x}) = 0$. So $\nabla f // \nabla g$ but...





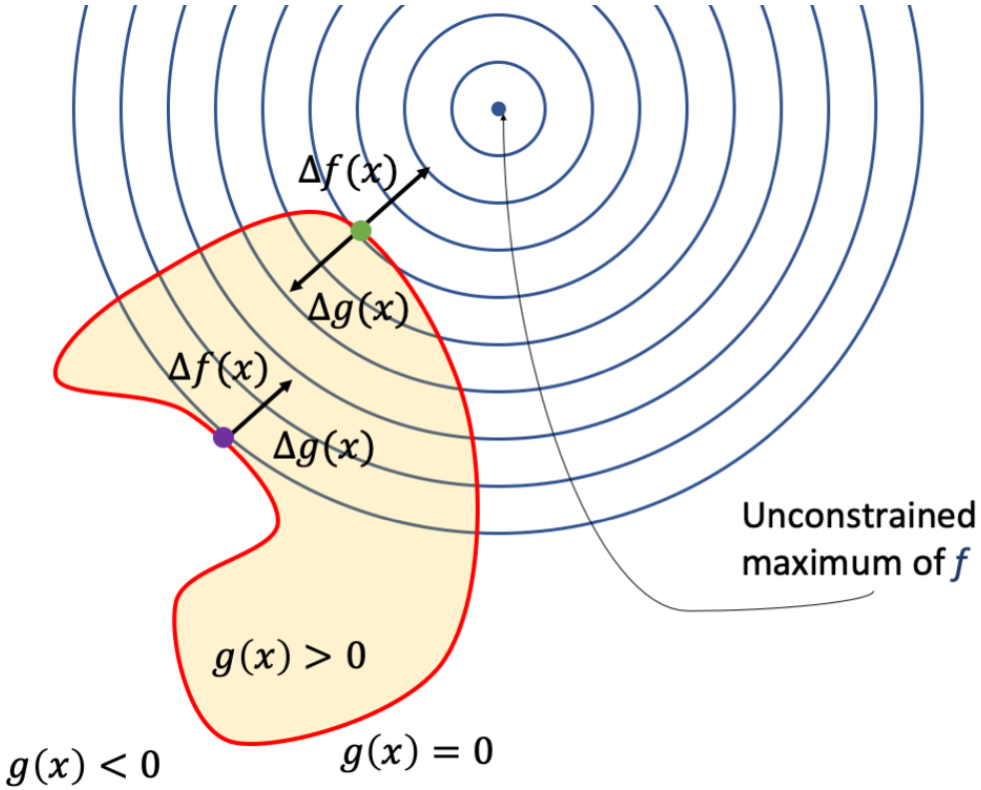
Lagrange Multipliers

- ... but we can directly find constrained local maxima (or minima).
- Two stationary points (green and purple) but the purple is **not** a constrained local maximum since $\nabla f(\mathbf{x})$ points towards the inside of the feasible region $\rightarrow$ there are certainly points with a greater $f(\mathbf{x})$
- The green point is a constrained local maximum since $\nabla f(\mathbf{x})$ points towards the outside of the feasible region $\rightarrow$ no points, within the feasible region, with a greater $f(\mathbf{x})$



Lagrange Multipliers

- The sign of the Lagrange multiplier is crucial !
- Constrained local maxima occur only when $\nabla f(\mathbf{x})$ and $\nabla g(\mathbf{x})$ have an opposite orientation, namely $\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x})$ for $\lambda > 0$



Lagrange Multipliers

To sum up:

- unconstrained maximum lies **within** the feasible region $\rightarrow$ stationary point of L with $\lambda = 0$ (i.e. $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$)
- unconstrained maximum lies **outside** the feasible region $\rightarrow$ constrained maximum lies on the boundary of the feasible region (i.e. $g(\mathbf{x}) = 0$) $\rightarrow$ stationary point of L with $\lambda > 0$ (this is why we put a plus in Eq.4))
- We can notice that in both cases $\lambda g(\mathbf{x}) = 0$ (either $\lambda = 0$ or $g(\mathbf{x}) = 0$). This gives us three conditions called Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} g(\mathbf{x}) &\geq 0 \\ \lambda &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned} \tag{30}$$

Lagrange Multipliers

- Given the optimization problem: $\max \quad f(\mathbf{x}) \quad s.t. \quad g(\mathbf{x}) \geq 0$
- Define the Lagrangian as: $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$
- We need to solve the following system of Eq.:

$$\begin{aligned} \nabla_{\mathbf{x}} L &= 0 \\ g(\mathbf{x}) &\geq 0 \\ \lambda &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned} \tag{31}$$

- Plus negative definite constraint on $\nabla^2_{\mathbf{xx}} L$

Lagrange Multipliers

- Given the optimization problem: $\min \quad f(\mathbf{x}) \quad s.t. \quad g(\mathbf{x}) \geq 0$
- Define the Lagrangian as: $L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$
- We need to solve the following system of Eq.:

$$\begin{aligned} \nabla_{\mathbf{x}} L &= 0 \\ g(\mathbf{x}) &\geq 0 \\ \lambda &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned} \tag{32}$$

- Plus positive definite constraint on $\nabla^2_{\mathbf{xx}} L$

NNMF

$$\underset{W\geq 0, H\geq 0}{\arg\min}\ F(W,H) = \frac{1}{2}||X - WH||_F^2 = \frac{1}{2}\sum_{ij}(X_{ij} - (WH)_{ij})^2 \qquad (33)$$

- To derive the KKT conditions, we can first notice that the condition $W \geq 0$ can be expressed using a matrix of multipliers $\Lambda .* W$. It follows that $\nabla_W L = 2(WHH^T - XH^T) - \Lambda = 0 \rightarrow 2(WHH^T - XH^T) = \Lambda$ and thus :

$$\begin{aligned} W \geq 0, \quad & \nabla_W F = WHH^T - XH^T \geq 0, \quad & W .* \nabla_W F = 0 \\ H \geq 0, \quad & \nabla_H F = W^T WH - W^T X \geq 0, \quad & H .* \nabla_H F = 0 \end{aligned}$$

- where $.*$ indicates the component-wise product between two matrices
- To note that NNMF gives sparse solution since either $W_{ik} = 0$ or $\nabla_{W_{ik}} F = 0$, same for H

NNMF - optimization

- Most of the optimization algorithms use a two-block coordinate descent scheme: either W or H is optimized while keeping the other fixed. Each sub-problem is therefore convex. It is called nonnegative least squares problem.

Algorithm 1 Two-Block Coordinate Descent

Input: X and r, the factorization rank
Output: W and H
Generate random positive initial matrices H and W
while certain stopping criteria **do**
  Update W, using the previous estimate of W and H
  Update H, using the current estimate of W and previous estimate of H
end while

NNMF

- One of the easiest algorithm is the Multiplicative Updates (MU) method
- It can be seen as a rescaled gradient descent method with specific step lengths μ_W and μ_H to preserve non-negativity

$$W - \mu_W .* \nabla_W F, \quad \mu_W = W./(WHH^T), \quad W \leftarrow W. * (XH^T)./(WHH^T)$$
$$H - \mu_H .* \nabla_H F, \quad \mu_H = H./(W^TWH), \quad H \leftarrow H. * (W^TX)./(W^TWH)$$

- where $./$ indicates component-wise division between two matrices
- It converges to a stationary point (not necessarily a local minima...)
- Other techniques exist such as alternating nonnegative least squares
- Many extensions exist like imposing orthogonality on W (i.e. $W^TW = I$). This removes the scaling ambiguity (not the permutation though) and it is useful for clustering: two nonnegative vectors are orthogonal iff their non-zero dimensions do not overlap

NNMF

Two implementation tricks for the MU method

- **To guarantee convergence**: during update re-initialize zero entries of W to a small positive constant when their partial derivatives become negative
  (From *Chi, E., Kolda, T.: On tensors, sparsity, and nonnegative factorizations. SIAM J. on Matrix Analysis and Applications 33(4), (2012)*)
- **To accelerate**: update W several times before updating H. In this way you do not need to recompute HH^T and XHT
  (From *Gillis, N., Glineur, F.: Accelerated multiplicative updates and hierarchical ALS algorithms for nonnegative matrix factorization. Neural Computation 24(4), (2012)*)

PCA and ICA and NNMF

PCA

- Data are assumed to follow a Gaussian distribution
- New basis that maximizes the variance of projected data
- Directions are orthogonal

ICA

- Data (at most one) must **not** follow a Gaussian distribution
- New basis which removes statistical dependency between directions
- Directions are not necessarily orthogonal

NNMF

- No assumption about distribution of data (but Gaussian noise)
- New non-negative basis whose components are purely added
- Directions are not necessarily orthogonal

Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

K-means

- Let N and d be the number of observations and feature respectively
- Let x_i be a column vector belonging to $\mathcal{X} = \mathcal{R}^d$. It represents one observation with d features

K-means Goal (Lloyd - 1957)

The goal of K-means (like any other clustering algorithm) is to partition the data-set into K homogeneous clusters. In addition to that, in K-means the clusters are non-overlapping (i.e. each observation belongs to only one cluster).

K-means cluster definition

A homogeneous cluster is a group of observations x_i whose variance (i.e. sum of squares of the distances from the average) is small compared to the distances with the points outside the cluster

K-means

- The total variation of a data-set X ($V(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$) divided into K clusters can be defined as the sum of two terms $V(X) = WV(X) + BV(X)$:
  - The within-cluster variation $WV(X) = \frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{N} z_{ik}(x_i - \mu_k)^2$
  - The between-cluster variation $BV(X) = \frac{1}{N}\sum_{k=1}^{K} N_k(\mu_k - \mu)^2$
- where μ is the global average of X, μ_k is the average of each cluster C_k, N_k is the number of observations in class k and z_{ik} is a binary indicator function (1 if $x_i \in C_k, 0$ otherwise).

K-means formulation

K-means aims to find a partition of X so as to minimize the within-cluster variation:

$$\underset{\{z_{ik}\},\{\mu_k\}}{\arg\min}\ WV(X) = \sum_{k=1}^{K}\sum_{i=1}^{N} z_{ik}(x_i - \mu_k)^2 \tag{34}$$

K-means - Observations

- Please note that since $V(X)$ is fixed, minimizing $WV(X)$ is equivalent to maximize $BV(X)$
- The decomposition of the total variation follows from the Law of total variance: $Var(X) = E[Var(X|C)] + Var(E[X|C])$ where C is the cluster decomposition
- One can also show that $\sum_{i=1}^{N} z_{ik}(x_i - \mu_k)^2 = \frac{1}{N_k} \sum_{i=1}^{N} \sum_{\substack{i=1 \\ j>i}}^{N} z_{ik} z_{jk} (x_i - x_j)^2 = \frac{1}{2N_k} \sum_{i=1}^{N} \sum_{\substack{i=1 \\ j \neq i}}^{N} z_{ik} z_{jk} (x_i - x_j)^2$ which means that the within-cluster variance of cluster C_k is equivalent to the pairwise squared distance between all elements belonging to cluster C_k.

K-means (Lloyd's algorithm)

- The number of ways in which a set of N observations can be partitioned into K non-empty clusters can be approximated as $P \approx \frac{K^N}{K!}$ (if $N = 100$ and $K = 3$ then $P \approx 8.6\text{x}10^{46}$). This is a non-convex, NP-hard, computationally prohibitive problem $\rightarrow$ Need for approximate solutions
- One possible solution is to optimize the assignments $\{z_{ik}\}$ and the averages $\{\mu_k\}$ separately and iteratively.

Lloyd's algorithm

- **Input**: Data X, number of classes K, initial means $\{\mu_k\}$
- Iterate until assignments step stops changing
  - **Assignment step**: Assign each observation to the closest average μ_k
  - **Update step**: Compute the new averages μ_k^*

- There is no guarantee to find the global optimum. This is why one usually starts several times this algorithm (with different initializations) and take the best result.

K-means (Lloyd's algorithm)

- **Assignment step**: we fix all μ_k and solve for every point x_i independently. We obtain:

$$z_{ik}^* = \begin{cases} 1 & \text{if } k = \arg\min_t (x_i - \mu_j)^2 \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

- **Update step**: we fix all z_{ik}^* and solve for μ_k:

$$\frac{\partial \left[\sum_{i=1}^N z_{ik}(x_i - \mu_k)^2\right]}{\partial \mu_k} = -2\sum_{i=1}^N z_{ik}(x_i - \mu_k) = 0 \tag{36}$$

$$\mu_k^* = \frac{\sum_{i=1}^N z_{ik} x_i}{\sum_{i=1}^N z_{ik}} = \frac{\sum_{i=1}^N z_{ik} x_i}{N_k} \tag{37}$$

K-means - Initializations

Possible initializations:

- Randomly choose K observations as initial means
- Randomly assigns a cluster to each observations and then compute the initial means
- Use as μ_1 the center of X and then traverse the other points and choose as center the points which are at least T units apart from the previously selected centers
- Centers should be spread out. *K-means++*:
  - Choose μ_1 randomly among the observations
  - Repeat next two steps until K centers have been choosen
    - $\forall x$ compute $D(x)$ as the distance between x and the nearest center
    - Choose randomly the next center using a weighted distribution where the weights are defined as $D^2(x)$

K-means - Distances

- The most common distance is the Euclidean one but any distance could be used (cosine, Mahalanobis, etc.)
- Please note that even for K-means we could use the Kernel trick:

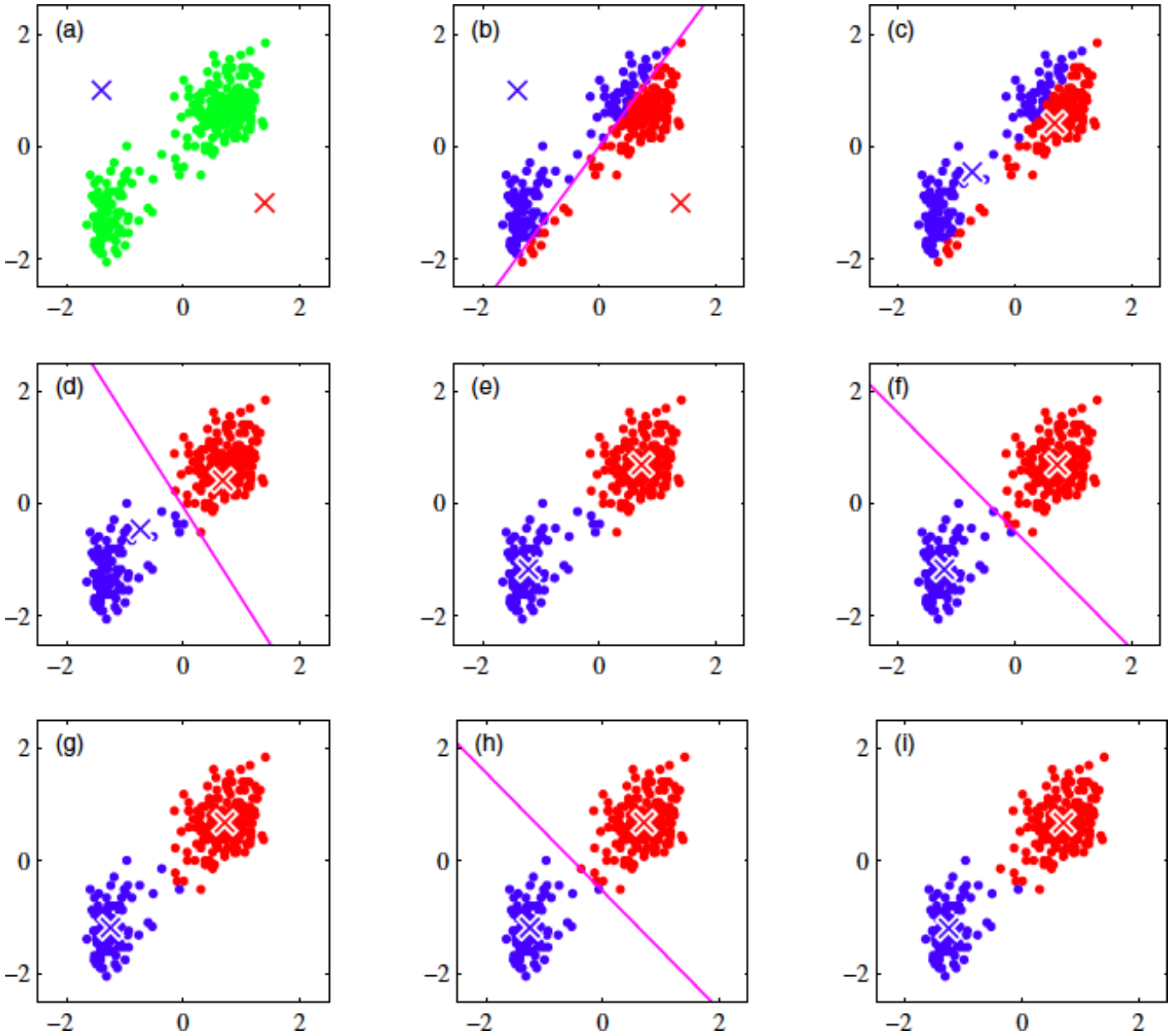
$$\arg\min_{C_1,\ldots,C_K} \sum_{k=1}^{K}\sum_{i=1}^{N_k} z_{ik}||x_i-\mu_k||^2 = \sum_{k=1}^{K}\frac{1}{2N_k}\sum_{i=1}^{N_k}\sum_{\substack{i=1\\ j\neq i}}^{N_k} z_{ik}z_{jk}||x_i-x_j||^2 =$$

$$\sum_{k=1}^{K}\frac{1}{2N_k}\sum_{i=1}^{N_k}\sum_{\substack{i=1\\ j\neq i}}^{N_k} z_{ik}z_{jk}(\langle x_i,x_i\rangle + \langle x_j,x_j\rangle - 2\langle x_i,x_j\rangle) =$$

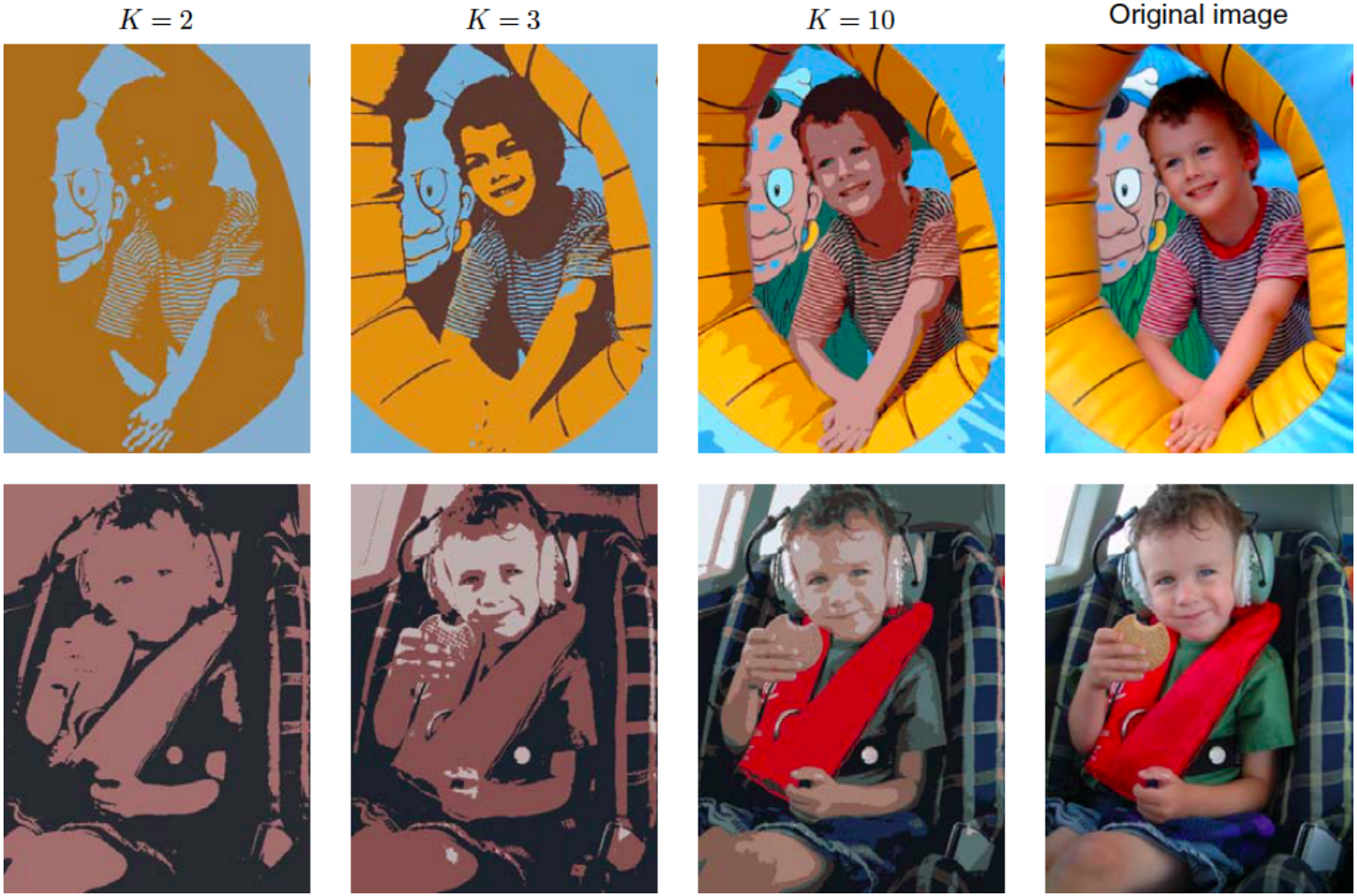
$$\arg\max_{C_1,\ldots,C_K} \sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k}\sum_{\substack{i=1\\ j\neq i}}^{N_k} z_{ik}z_{jk}(\langle x_i,x_j\rangle) \tag{38}$$

- we could define inner products $\langle\cdot,\cdot\rangle$ using kernels

K-means - example from Bishop



K-means - segmentation example from Bishop



K-means - color quantization example from scikit-learn

Original image (96,615 colors)

Quantized image (64 colors, K-Means)

K-means - cluster validation

- To validate a clustering results one could look at the values of $WV(X)$ and $BV(X)$ (pick the model that gives the minimum $WV(X)$ and thus maximum $BV(X)$)
- *Silhouette coefficient* s: let $a(x)$ be the average distance between x and all the other points within the same cluster of x. Let $b_k(x)$ the average distance between x and all points in cluster C_k (x does not belong to cluster C_k). The neighboring cluster of x, or the second best fit, is defined as the cluster with the smallest average distance: $b_{min}(x) = \min_k b_k(x)$. We define:

$$s(x) = \frac{b_{min}(x) - a(x)}{\max\{a(x), b_{min}(x)\}} \tag{39}$$

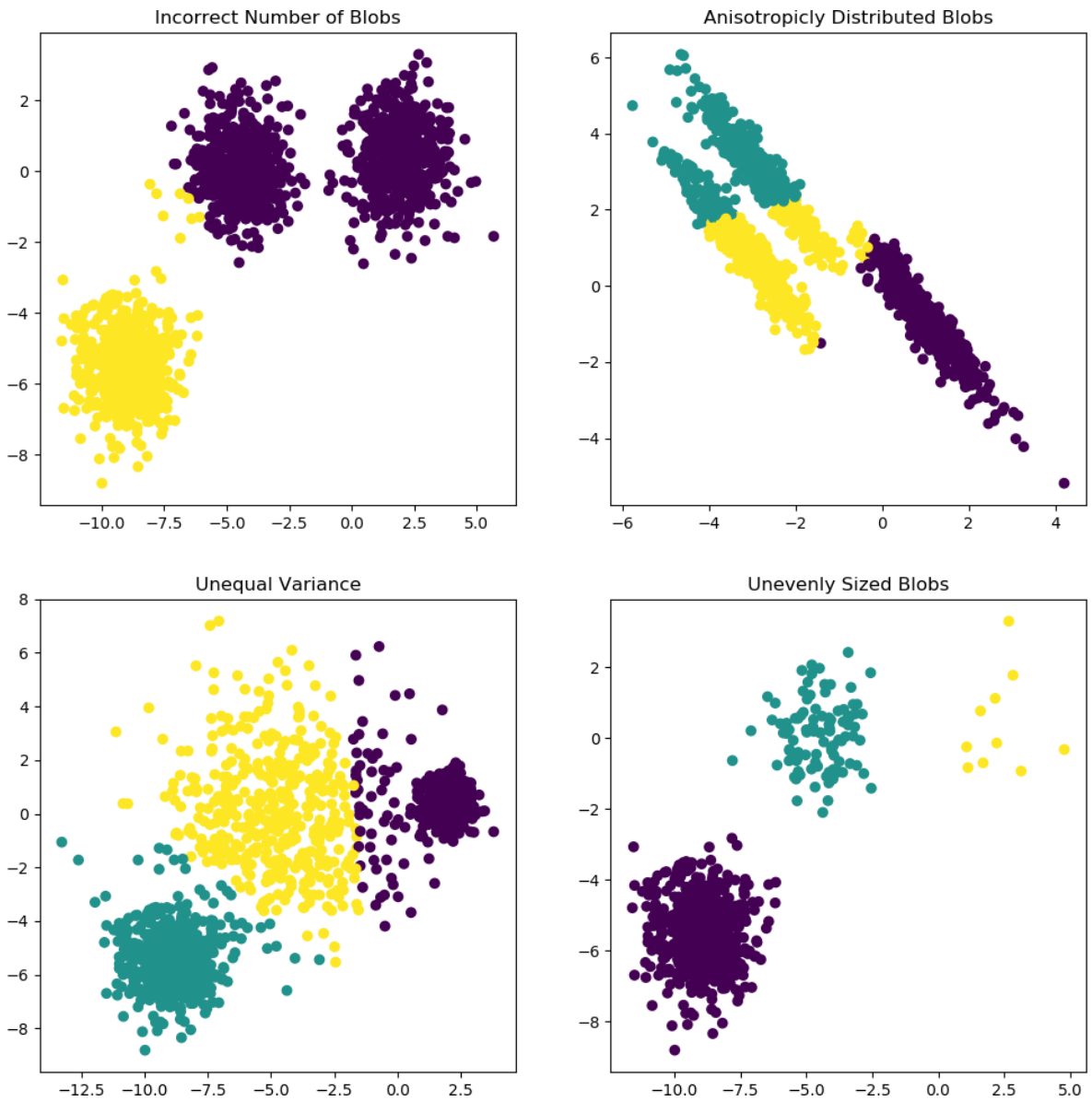
- $s(x) = 1$ means that x is appropriately clustered, $s(x) = -1$ means that the neighboring cluster would be more appropriate, $s(x) = 0$ means that x is close to the border of the two clusters

K-means - considerations

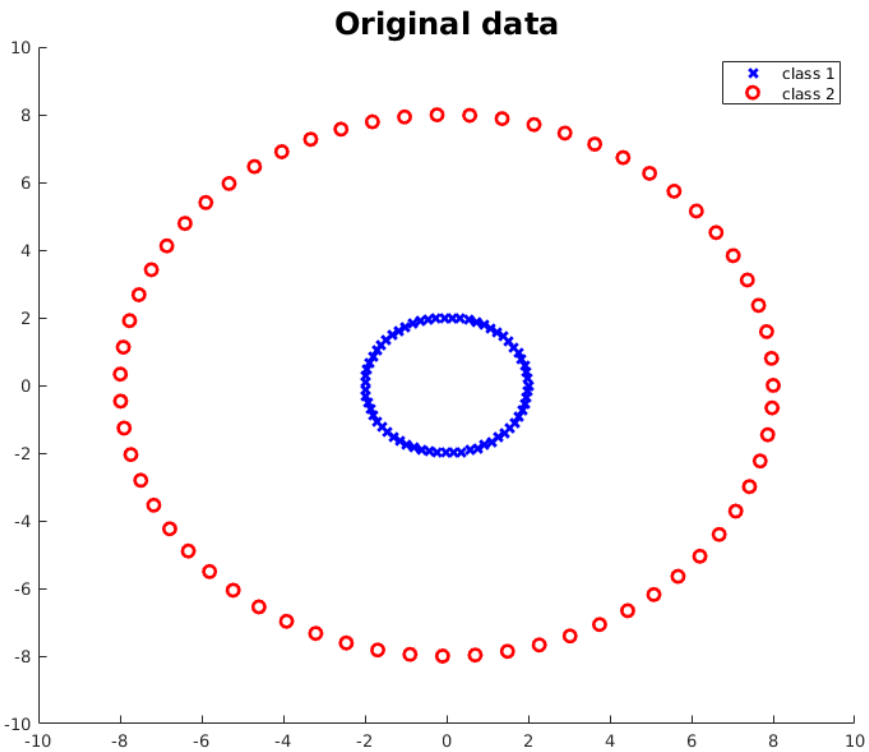
K-means assumptions

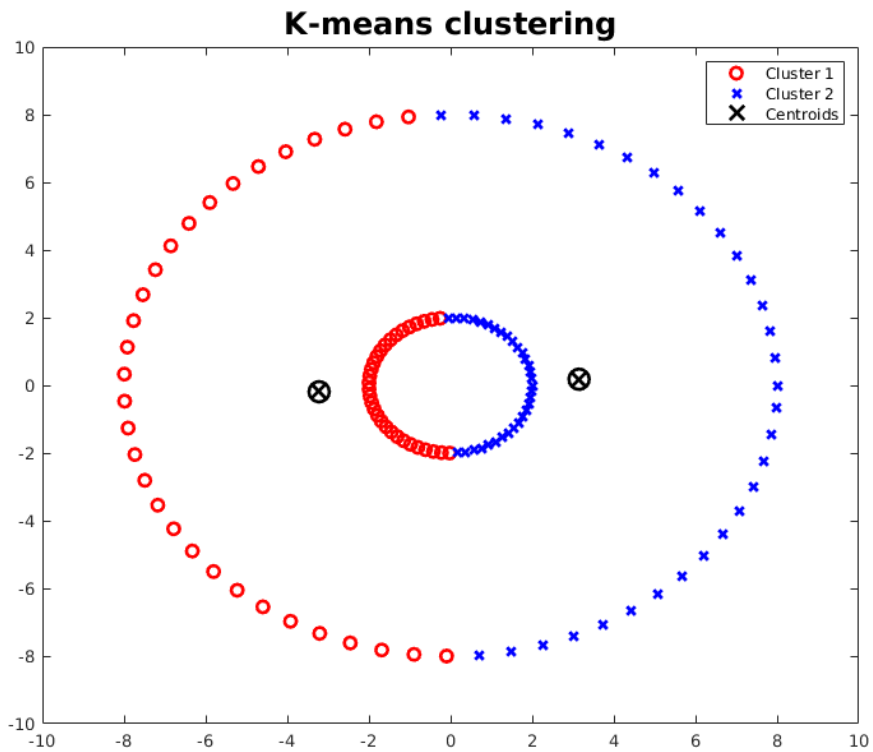
- it always find K clusters even when data comes from the same distribution (i.e. $K = 1$) or when the number of clusters K is not correct
- *hard assignment* of data points to clusters $\rightarrow$ a point that is roughly midway between two clusters should probably have a *soft assignment*
- clusters should have similar variance (spatial extent)
- clusters should be isotropic (spherical/Gaussian)

K-means - assumptions - Image from scikit-learn



K-means - importance of the distance





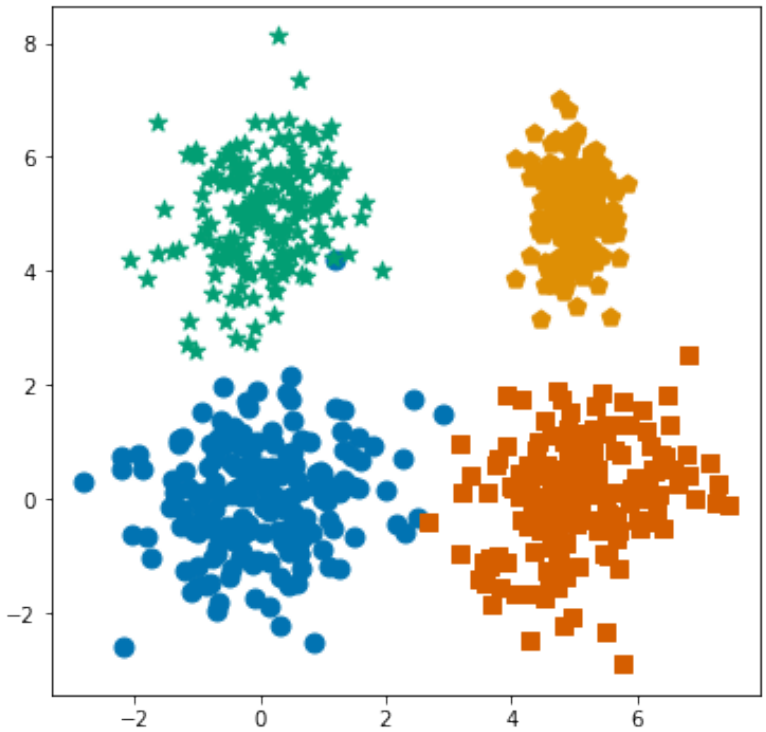
This result is based on the Euclidean distance. Remember the Kernel trick...

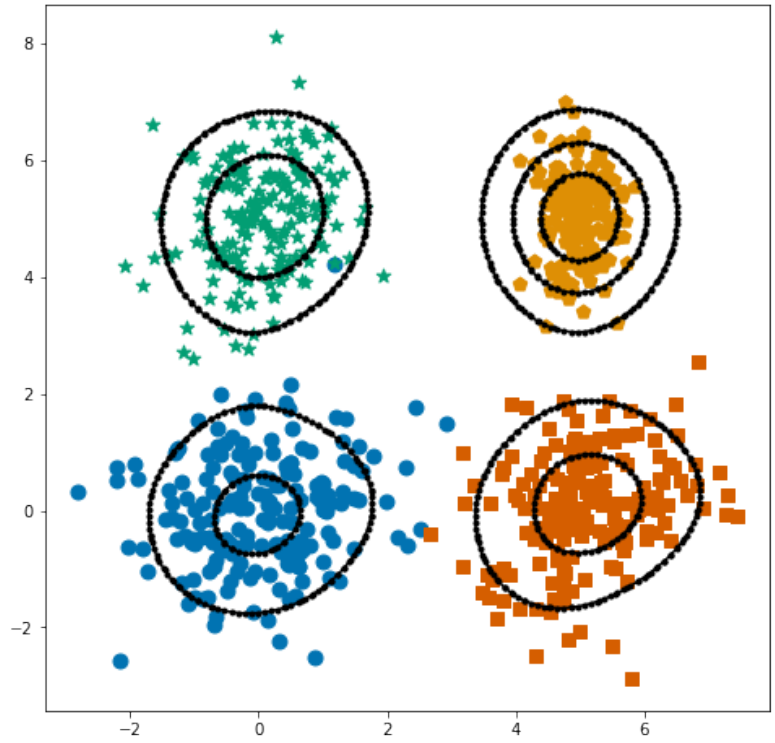
Summary

1. Introduction
   - Curse of dimensionality
   - Hypothesis behind unsupervised learning
   - Goals of Unsupervised Learning
2. Dimensionality reduction
   - Principal Component Analysis (PCA)
   - Constrained Optimization - Equality constraint
   - Kernel Principal Component Analysis (K-PCA)
   - Independent Component Analysis (ICA)
   - Non-negative matrix factorization (NNMF)
   - Constrained Optimization - Inequality constraint
3. Clustering
   - K-means
   - Mixture models

Mixture models for clustering

- We can also consider a probabilistic (generative) model for clustering where we assume that the data has been generated from a mixture of different distributions (linear superposition)





Mixture models for clustering

Definition

A **mixture model** is a probability density function $p(x)$ that can be written as a *convex combination* of a finite number of probability density functions $p_k(x)$

$$p(x) = \sum_{k=1}^{K} \pi_k p_k(x) \tag{40}$$

where the mixing coefficients $\pi_k \geq 0$, $\sum_{k=1}^{K} \pi_k = 1$

- We usually make the hypothesis that all components $p_k(x)$ follow the same distribution (e.g. Gaussian or Bernoulli) but with different parameters

Mixture models for clustering

- The mixing coefficients π_k can be seen as prior probabilities for each component k
- We can thus define a random variable z that models the membership to the different components and where the probability of each component is separately specified by π_k
- More formally, z follows a K-dimensional categorical distribution, namely it is a binary random variable of size K where only one element z_k is equal to 1 and all other elements are equal to 0
- The variable z has therefore K possible states, namely the number of components, and it satisfies $z_k \in \{0, 1\}$ and $\sum_k z_k = 1$
- The marginal distribution over z is $p(z_k = 1) = \pi_k$ and $p(z) = \prod_{k=1}^{K} \pi_k^{z_k}$

Mixture models for clustering

- z is not observed, it is a hidden (or latent) variable $\rightarrow$ this makes it a **latent variable model**
- We have one variable z for each observation x_i, which specifies the mixture component to which x_i belongs
- We should therefore work with the joint distribution between x and z defined as $p(x, z)$ which can be rewritten as:

$$p(x, z) = p(z)p(x|z) \tag{41}$$

- Here, we make the hypothesis that $p(x|z)$ is Gaussian ! $\rightarrow$ $p(x|z_k = 1) = \mathcal{N}(x|\mu_k, \Sigma_k)$ or $p(x|z) = \prod_{k=1}^{K} \mathcal{N}(x|\mu_k, \Sigma_k)^{z_k}$

Mixture models for clustering

- Can we work with the joint distribution $p(x, z)$? Do we usually know it ?

Mixture models for clustering

- Can we work with the joint distribution $p(x, z)$? Do we usually know it ? No ! We only observe x and therefore we can work only with $p(x)$, the marginal distribution

$$p(x) = \sum_z p(x, z) = \sum_z p(z)p(x|z) = \sum_z \prod_{k=1}^{K} \pi_k^{z_k} \mathcal{N}(x|\mu_k, \Sigma_k)^{z_k} =$$
$$= \sum_{j=1}^{K} \prod_{k=1}^{K} (\pi_k \mathcal{N}(x|\mu_k, \Sigma_k))^{\mathcal{I}_{kj}} = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

- where we have exploited the binary 1-of-K representation of z defining $\mathcal{I}_{kj} = 1$ if $k = j$ and 0 otherwise

Mixture models for clustering

- Given a set of N observations in $\mathcal{R}^D$ $\{x_1, x_2, ..., x_N\}$ represented as a matrix X of size $[N \times D]$ and the corresponding latent variables in matrix Z of size $[N \times K]$, we assume that observations are i.i.d. and consider $\theta = \{\pi_k\}, \{\mu_k\}, \{\Sigma_k\}$ the parameters of our model. The log likelihood is:

$$\ln p(X;\theta) = \ln \left(\prod_{i=1}^{N} p(x_i) \right) = \sum_{i=1}^{N} \ln p(x_i) =$$
$$= \sum_{i=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k) \right\}$$

- Recall that:
$\mathcal{N}(x_i|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma_k|^{1/2}} \exp(-\frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k))$

Mixture models for clustering

- We can first notice that, if we had only a single component , computations would be quite easy:

$$\ln p(X;\theta) = \sum_{i=1}^{N} \ln \mathcal{N}(x_i|\mu,\Sigma) =$$

$$= -\frac{N}{2}\ln|\Sigma| - \frac{1}{2}\sum_{i=1}^{N}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu))$$

- If we compute the derivatives wrt μ and Σ, we obtain the standard ML estimates. Of course the prior π_k has disappeared...
- But what happens when we have more than one component ?

Mixture models for clustering

- We can first notice that, if we had only a single component , computations would be quite easy:

$$\ln p(X;\theta) = \sum_{i=1}^{N} \ln \mathcal{N}(x_i|\mu,\Sigma) =$$

$$= -\frac{N}{2}\ln|\Sigma| - \frac{1}{2}\sum_{i=1}^{N}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu))$$

- If we compute the derivatives wrt μ and Σ, we obtain the standard ML estimates. Of course the prior π_k has disappeared...
- But what happens when we have more than one component ? Expression more complicated ! No closed form solution

Mixture models for clustering

$$\frac{\partial}{\partial \mu_k} \ln p(X;\theta) = \sum_{i=1}^{N} \frac{\partial}{\partial \mu_k} \ln \left\{ \sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j) \right\} =$$

$$= \sum_{i=1}^{N} \frac{\frac{\partial}{\partial \mu_k} \left(\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j) \right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} =$$

$$= \sum_{i=1}^{N} \frac{\pi_k \frac{\partial}{\partial \mu_k} \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} =$$

$$= \sum_{i=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}}_{\gamma(z_{ik})} \Sigma^{-1}(x_i - \mu_k) = 0$$

- Multiplying by Σ, it results: $\mu_k^* = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$ where $N_k = \sum_{i=1}^{N} \gamma(z_{ik})$ is the effective number of x_i assigned to cluster k

Mixture models for clustering

- Let's have a look at $\gamma(z_{ik})$, this is the conditional probability of z given x. Using the Bayes' theorem, we obtain:

$$\gamma(z_{ik}) = p(z_k = 1|x) = \frac{p(z_k = 1)p(x|z_k = 1)}{p(x)} = \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)} \tag{42}$$

- $\gamma(z_{ik})$ can be seen as the *responsibility* that component k takes for 'explaining' observations x
- This means that $\mu_k^* = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$ is a weighted average of all observations, and weights explain how much component k was responsible for generating x_i

Mixture models for clustering

- We can also compute the derivatives wrt Σ_k:

$$\frac{\partial}{\partial \Sigma_k} \ln p(X;\theta) = \sum_{i=1}^{N} \frac{\pi_k \frac{\partial}{\partial \Sigma_k} \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} =$$

$$= \frac{1}{2} \sum_{i=1}^{N} \gamma(z_{ik})(-\Sigma_k^{-1} + \Sigma_k^{-1}(x_i - \mu_k)(x_i - \mu_k)^T \Sigma_k^{-1}) = 0$$

- where we have used $\frac{\partial (a^T X^{-1} a)}{\partial X} = -X^{-1} a a^T X^{-1}$ and $\frac{\partial |X|}{\partial X} = |X| X^{-1}$ considering X symmetric
- It results $\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik})(x_i - \mu_k)(x_i - \mu_k)^T$

Mixture models for clustering

- ... and the derivatives wrt π_k where we need to take into account the fact that $\pi_k \geq 0$ and $\sum_{k=1}^{K} \pi_k = 1$. Calling π the vector containing all $\{\pi_k\}$, the optimization problem thus becomes:

$$\begin{aligned} \max_{\pi_k} \quad & \ln p(X;\theta) \\ s.t. \quad & \pi_k \geq 0 \quad \forall k \quad \rightarrow \quad \pi \geq 0 \\ s.t. \quad & \sum_{k=1}^{K} \pi_k = 1 \quad \rightarrow \quad \pi^T \mathbf{1} = 1 \end{aligned} \tag{43}$$

- The Lagrangian is

$$L(\pi, \alpha, \lambda) = \ln p(X;\theta) + \alpha^T \pi + \lambda(\pi^T \mathbf{1} - 1) \tag{44}$$

- where α is a vector of K Lagrange multipliers and λ is a (scalar) Lagrange multiplier

Mixture models for clustering

- The KKT conditions are:

$$\nabla_{\pi_k} L = \frac{\partial}{\partial \pi_k} \ln p(X;\theta) + \alpha_k + \lambda = 0$$

$$\pi_k \geq 0$$

$$\sum_{k=1}^{K} \pi_k = 1$$

$$\alpha_k \geq 0$$

$$\alpha_k \pi_k = 0$$

- which results

$$\sum_{i=1}^{N} \frac{\mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} + \alpha_k + \lambda = 0 \tag{45}$$

Mixture models for clustering

- We then estimate the values (or eliminate) of the Lagrange multipliers by using the other KKT conditions. Let's see how ! First multiply both sides by π_k:

$$\sum_{i=1}^{N} \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} + \underbrace{\alpha_k \pi_k}_{=0} + \lambda \pi_k = 0 \tag{46}$$

- and then sum over k:

$$\sum_{i=1}^{N} \underbrace{\frac{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}}_{=1} + \lambda \underbrace{\sum_{k=1}^{K} \pi_k}_{=1} = 0 \tag{47}$$

- it results:

$$\lambda = -N \tag{48}$$

Mixture models for clustering

- Let's use this result back into Eq.46:

$$\sum_{i=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}}_{\gamma(z_{ik})} - N\pi_k = 0 \tag{49}$$

- remember that $N_k = \sum_{i=1}^{N} \gamma(z_{ik})$ is the effective number of observations assigned to cluster k, it results:

$$\pi_k = \frac{N_k}{N} \tag{50}$$

- Thus π_k is the average responsibility that component k takes for explaining all observations

Mixture models for clustering

- We have thus obtained a formula for estimating each parameter:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik})(x_i - \mu_k)(x_i - \mu_k)^T$$

$$\pi_k = \frac{\sum_{i=1}^{N} \gamma(z_{ik})}{N}$$

- Great we have a closed-form solution ! Actually no... Why ?

Mixture models for clustering

- We have thus obtained a formula for estimating each parameter:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik})(x_i - \mu_k)(x_i - \mu_k)^T$$

$$\pi_k = \frac{\sum_{i=1}^{N} \gamma(z_{ik})}{N}$$

- Great we have a closed-form solution ! Actually no... Why ? They all depend on $\gamma(z_{ik})$ which depends on them !
- We should use an iterative scheme

Mixture models for clustering

- First initialize μ_k, Σ_k and π_k
- **E step**: Evaluate $\gamma(z_{ik})$ using the current estimates of μ_k, Σ_k and π_k

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{51}$$

- **M step**: Estimate μ_k, Σ_k and π_k using current estimate of $\gamma(z_{ik})$

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik})(x_i - \mu_k)(x_i - \mu_k)^T$$

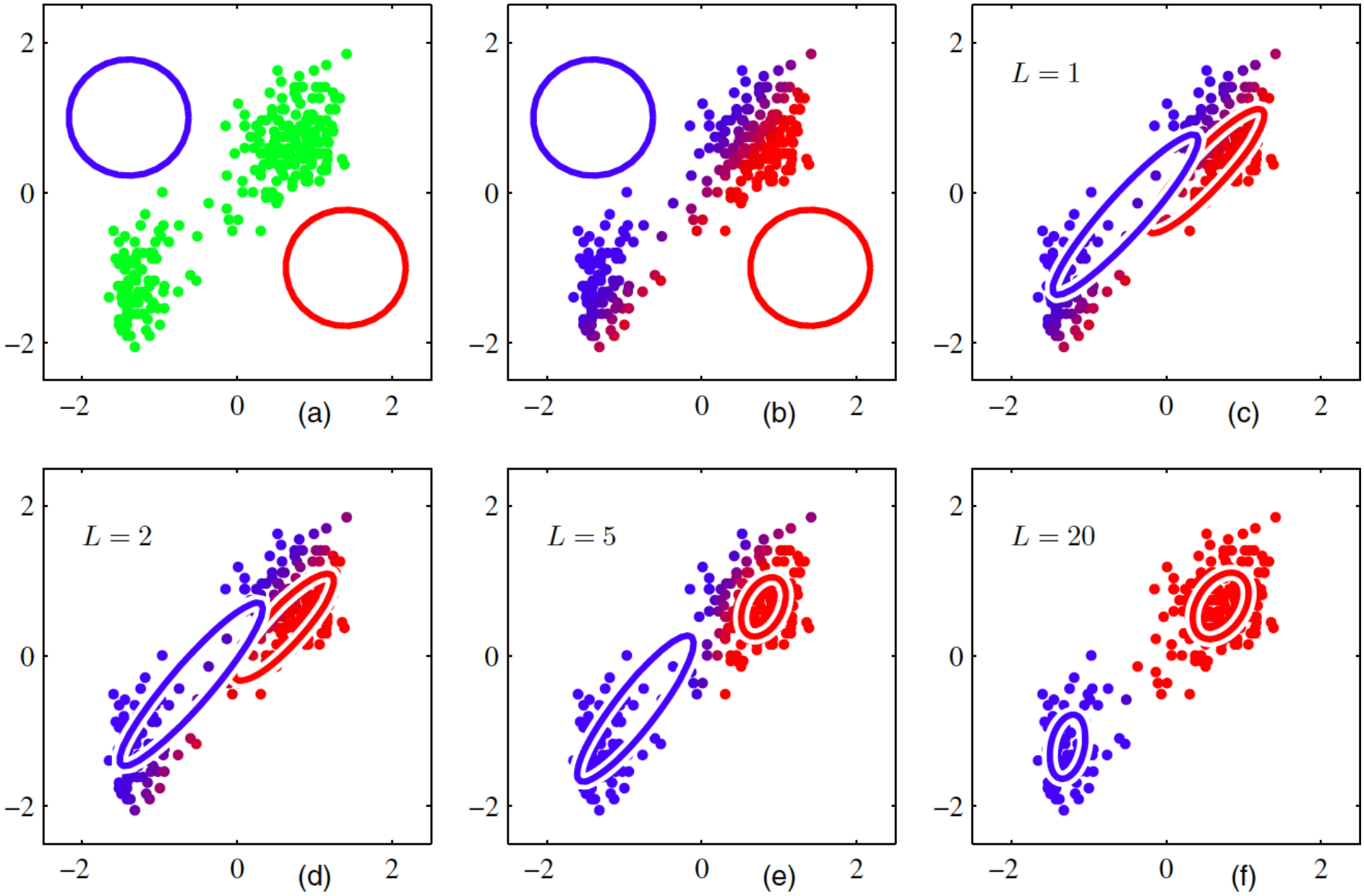
$$\pi_k = \frac{N_k}{N}$$

- Iterate E and M steps until convergence

Mixture models for clustering

- This is the EM (Expectation and Maximization) algorithm applied to Gaussian mixture models
- It usually takes more iterations (and more computational time) to convergence compared to K-means $\rightarrow$ K-means is usually used for initializing μ_k, Σ_k and π_k
- Possible multiple local maxima $\rightarrow$ re-run the optimization with different initializations
- Problem of singularities when one Gaussian component collapses onto one single data point ($\mu_k = x_i$ and $\Sigma_k \rightarrow 0$). This makes the log likelihood go to infinity ! $\rightarrow$ if a component k is collapsing, randomly reset μ_k and Σ_k
- For more information on EM, please read Chapter 9.3 of Bishop book

Mixture models for clustering - from Bishop



References

PCA and KPCA

1. I.T. Jolliffe - Principal Component Analysis - 2002 - Springer

ICA

1. Hyvärinen et al. - Independent Component Analysis - 2001 - Wiley
2. Hyvärinen - Fast and Robust Fixed-Point ... - IEEE TNN - 1999
3. Tichavsky et al. - Performance Analysis of the FastICA ... - IEEE TSP - 2006
4. Bartlett et al. - Face recognition by ICA - IEEE TNN - 2002

NNMF

1. Lee and Seung - Learning the parts of objects ... - Nature - 1999
2. Lee and Seung - Algorithms for NNMF - Nips - 2000
3. N. Gillis - The why and how of nonnegative matrix factorization - 2014

References

K-means and mixture models

1. Steinley, D. - K-means clustering: A half-century synthesis. Brit. J. Math. and Stat. Psych. - 2006
2. McLachlan, G.J. etal. - Mixture models: Inference and applications to clustering - 1988
3. Corduneanu and Bishop - Variational Bayesian Model Selection for Mixture Distributions - Artificial Intelligence and Statistics 2001
4. Bishop book (Chapter 9)