IFT 6390
Fondements de l'apprentissage machine

Unsupervised learning

Dimensionality reduction
Continuous latent variables

Professor: Ioannis Mitliagkas
Slides: Pascal Vincent

Context

Supervised Learning = classification, regression

Unsupervised Learning = Algorithms that don't need any explicit "target"/label in the training set.

- Density estimation
  (ex: Gaussian mixture)
- *Clustering*
  (ex: k-means)
- Dimension reduction

Dimensionality reduction
What is it?

$\mathbf{z} \in \mathbb{R}^M$

$M < D$

$$\begin{tikzcd} (0.32,\ -1.3,\ 1.2) \\ (3.5,\ -1.7,\ 2.8,\ -3,\ 5,\ -1.4,\ 2.4,\ 2.7,\ 7.5,\ -3,\ -2) \arrow[u] \end{tikzcd}$$

$\mathbf{x} \in \mathbb{R}^D$

Dimensionality reduction
What for?

- Compressing data (with loss)
- Visualize data in 2D or 3D
- Feature extraction
  potentially +fundamental, +explicative, +compact
  Pre-processing => better data representation for another algorithm (classification or regression).

Algorithms

Gaussian linear models

- Principal Component Analysis (PCA)
- Probabilistic PCA
- Factor Analysis

Non-linear/Non-Gaussian models

- Kernel PCA
- Independent Components Analysis
- Auto-encoder neural networks
- Non-linear manifold modeling

Principal Component Analysis (PCA)

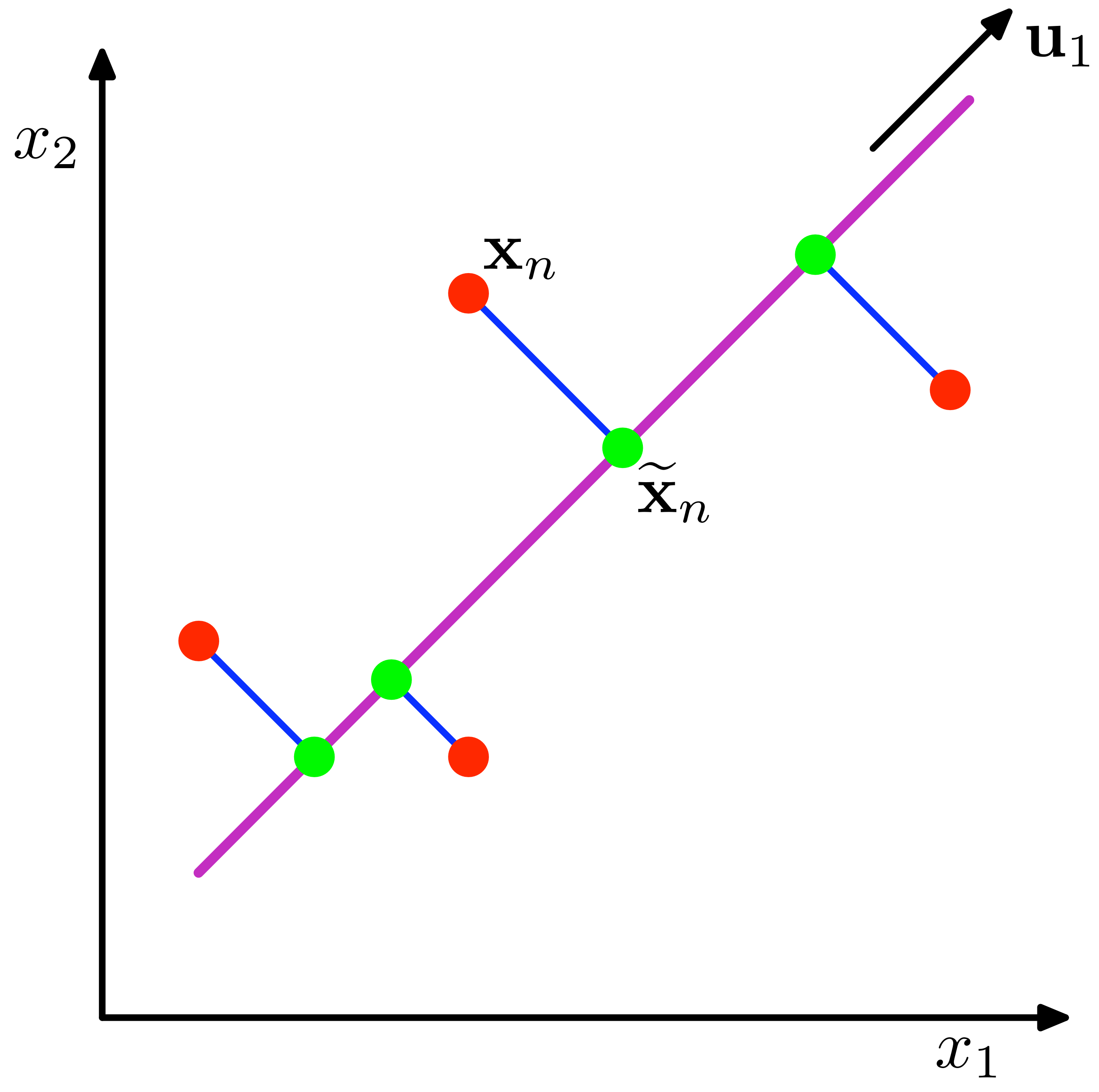
- PCA finds a linear subspace that is close to the data: orthogonal projection of $\mathbf{x} \in \mathbb{R}^D$ onto a linear subspace of lower dimension M.
- Entries of $z \in \mathbb{R}^M$ are the coordinates of the projection of x in this M dimensional subspace.
- Often used as a pre-processing step (feature extraction) or for visualization.
- An old algorithm with two equivalent formulations:
  - minimizing the reconstruction error *(Pearson 1901)*
  - maximizing the variance *(Hotelling 1933).*
- Probabilistic interpretation is more recent

PCA: two equivalent formulations

We seek the principal directions **u**: an orthonormal basis of the space onto which to project x

- Minimal reconstruction error:
  minimize the mean squared distances between the x's and their projections (blue lines).

- Maximal variance:
  maximize the variance along the projection space (variance of the green points)



PCA with variance maximization

Let $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$ be the empirical mean.

The variance of the points projected onto the direction $\mathbf{u}_1$ is given by

$$\frac{1}{N} \sum_{n=1}^{N} (\mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \bar{\mathbf{x}})^2 = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$$

where $\mathbf{S}$ is the empirical covariance matrix

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$$

Adding constraint $\|\mathbf{u}_1\| = 1$, we want to solve the minimization problem

$$\min_{\|\mathbf{u}_1\|=1} \mathbf{u}_1^\top \mathbf{S} \mathbf{u}_1$$

whose solution is the dominant eigenvector of the covariance matrix $\mathbf{S}$.

PCA with variance maximization

- Once we have the first principal direction/component, we can incrementally find the second one, third one, etc., by imposing each direction to be orthogonal to the previous ones.
- Using the *minimal reconstruction error* formulation leads to the exact same principal components.

PCA: simple overall procedure

The orthonormal basis U we obtain contains the M first eigenvectors of the empirical covariance matrix S (the ones corresponding to the M largest eigenvalues).

Estimate the covariance matrix: $\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$

Eigenvalue decomposition: $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$

Keep only the first M eigenvectors: $\mathbf{S} \approx \mathbf{U}_M\mathbf{\Lambda}_M\mathbf{U}_M^T$

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_d \end{pmatrix}$$

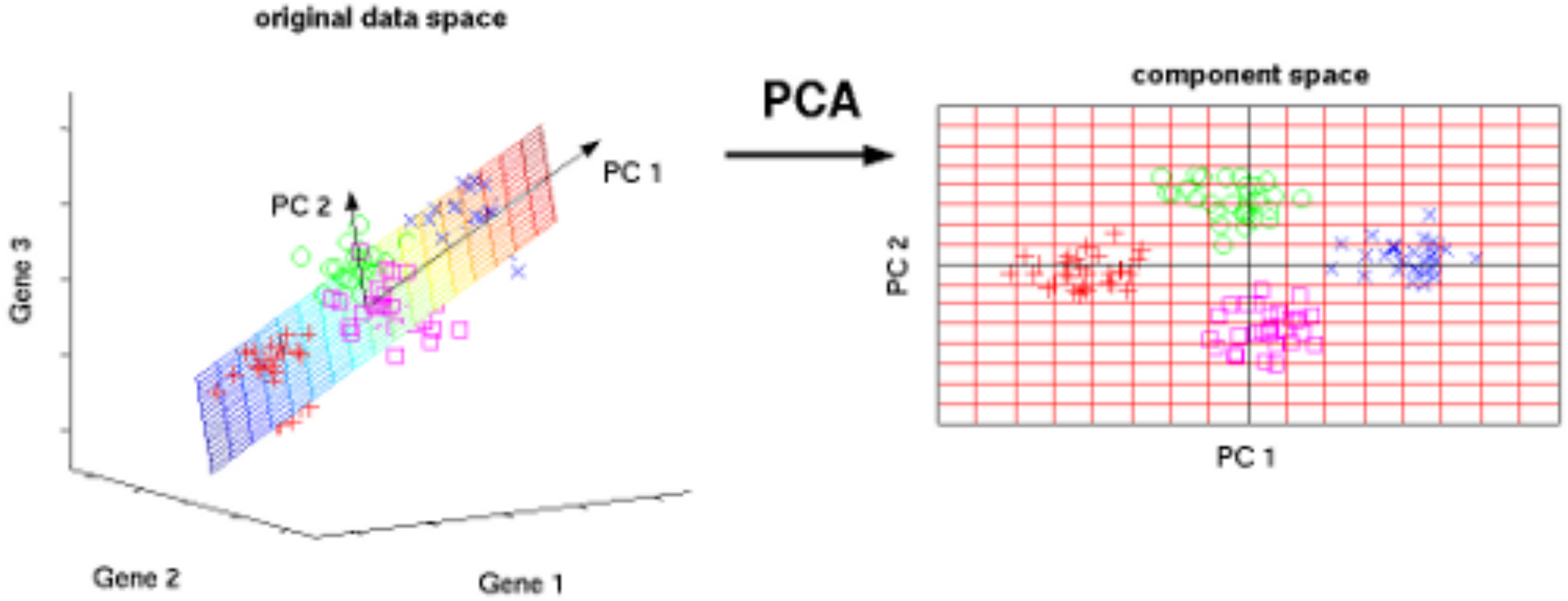
Extracting the M principal components of a vector $\mathbf{x}$ (projection onto the M principal components)

$$\mathbf{z}(\mathbf{x}) = \mathbf{\Lambda}_M^{-\frac{1}{2}}\mathbf{U}_M^T(\mathbf{x} - \bar{\mathbf{x}})$$

- Using PCA for pre-processing gives us "normalized" vectors **z**, i.e. uncorrelated (independent entries) and with unit variance
- We can project onto the principal directions for visualization in 2D or 3D.

Reconstruction (from the M principal components): $\mathbf{x} \approx \mathbf{U}_M(\Lambda_M^{\frac{1}{2}}\mathbf{z}) + \bar{\mathbf{x}}$

Ex PCA: D=3→M=2



Source: http://www.nlpca.org/pca_principal_component_analysis.html

PCA and normalization

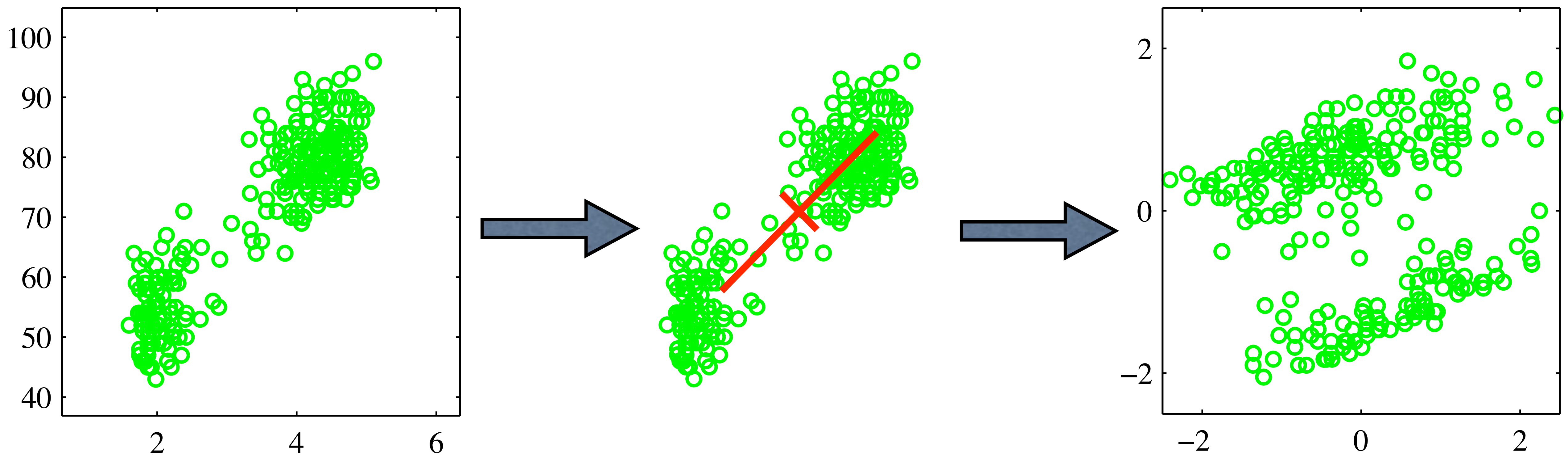
Original data

Principal components

(intervalle $\pm\lambda_i^{1/2}$*)*

Pre-processed data

(cov = I)

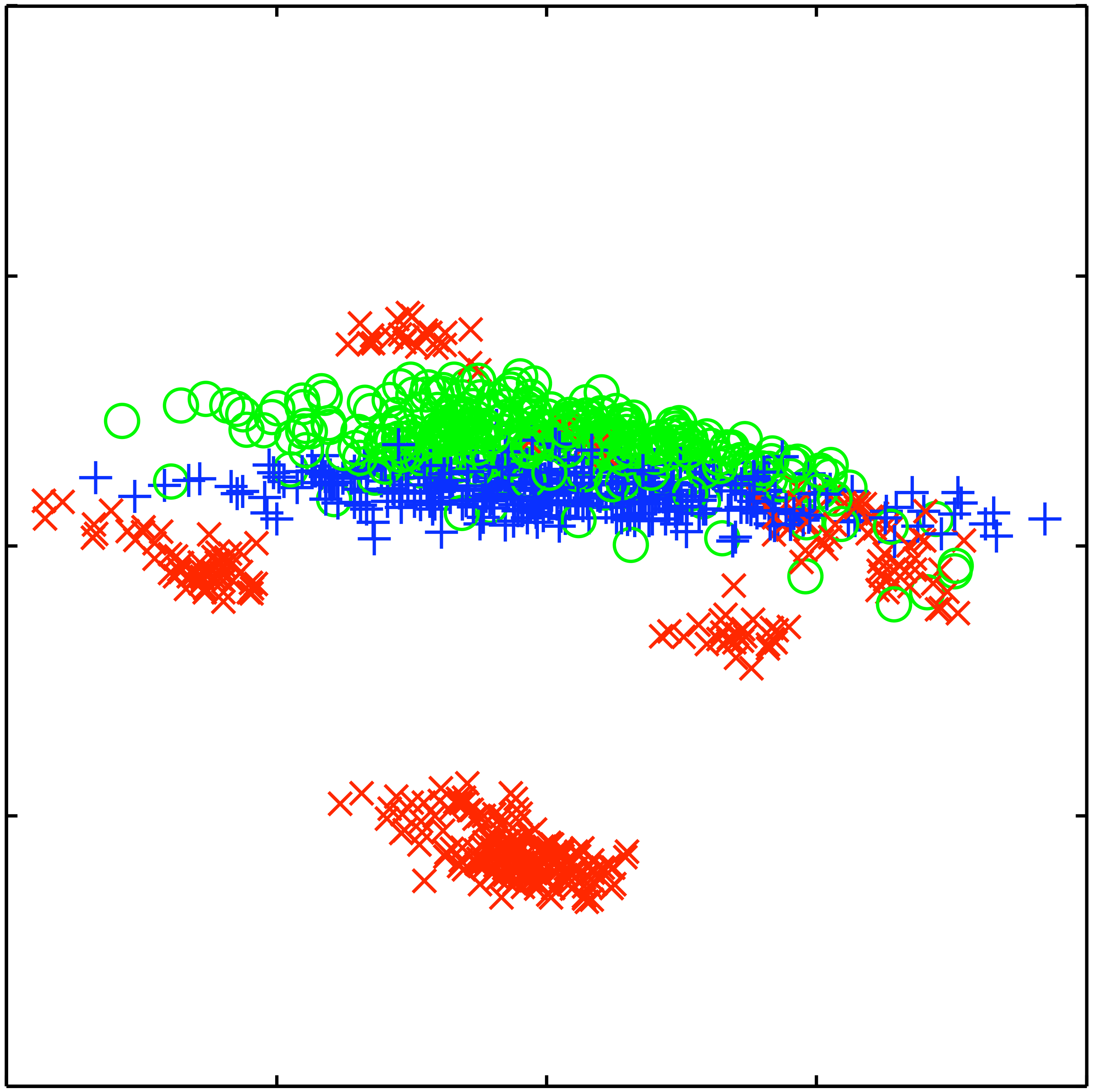


We could have kept only the first component

PCA for visualization

High-dimensional input data

2 principal components



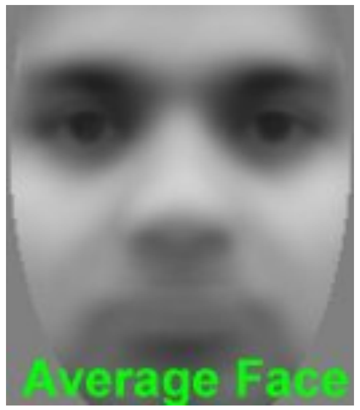
Ex: eigenfaces

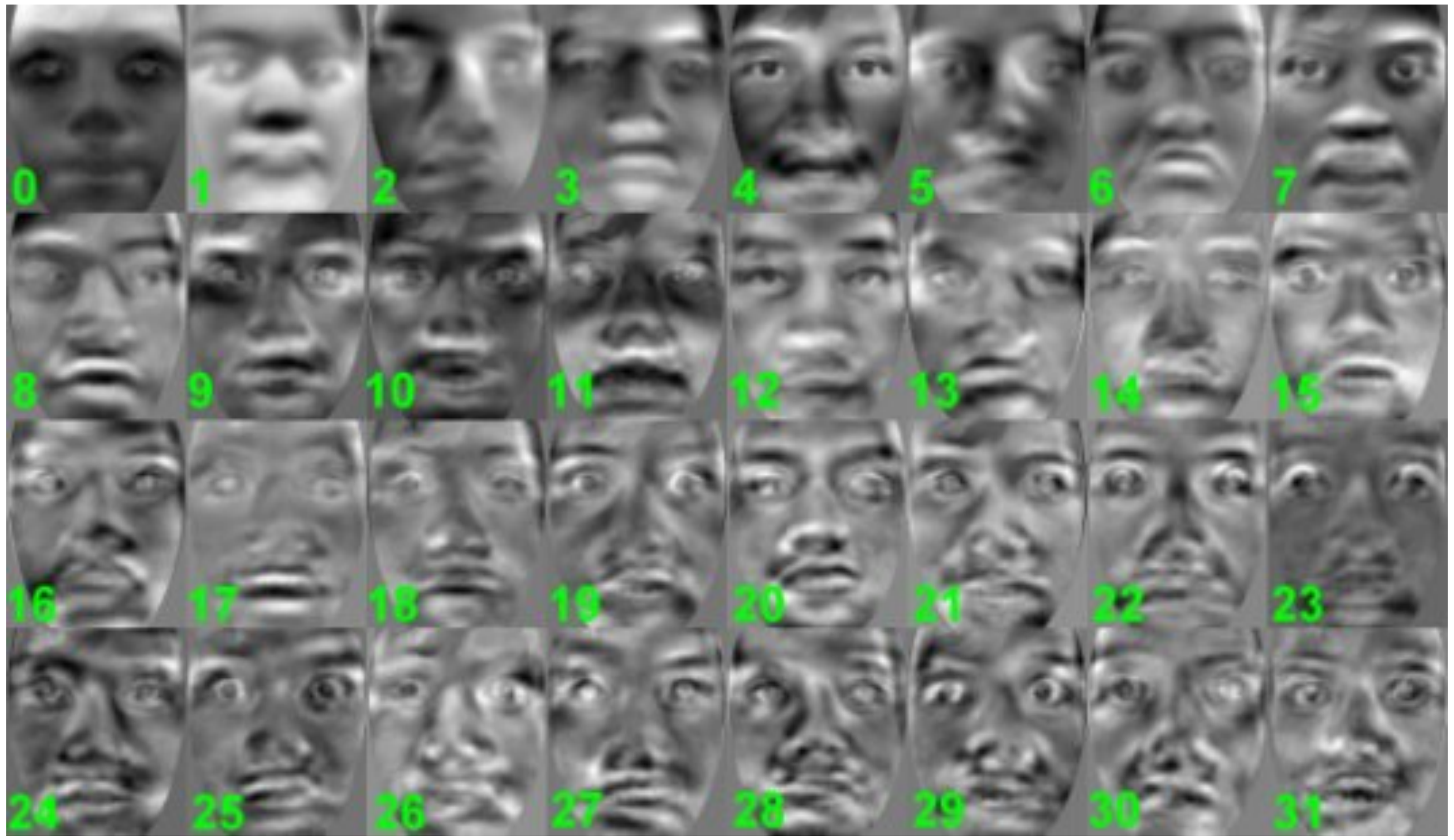
Input data: face images

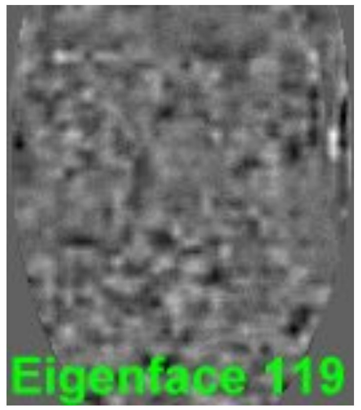
etc ...

Each example is a d-dimensional vector

The basis with the 31 first eigenvectors:







http://www.shervinemami.info/faceRecognition.html

Kernel PCA

- The transformation learned by PCA is linear. We can get a non-linear extension of PCA by embedding/projecting the data into a feature space using a non-linear mapping ϕ before applying PCA
- The transformed data has the covariance matrix

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T$$

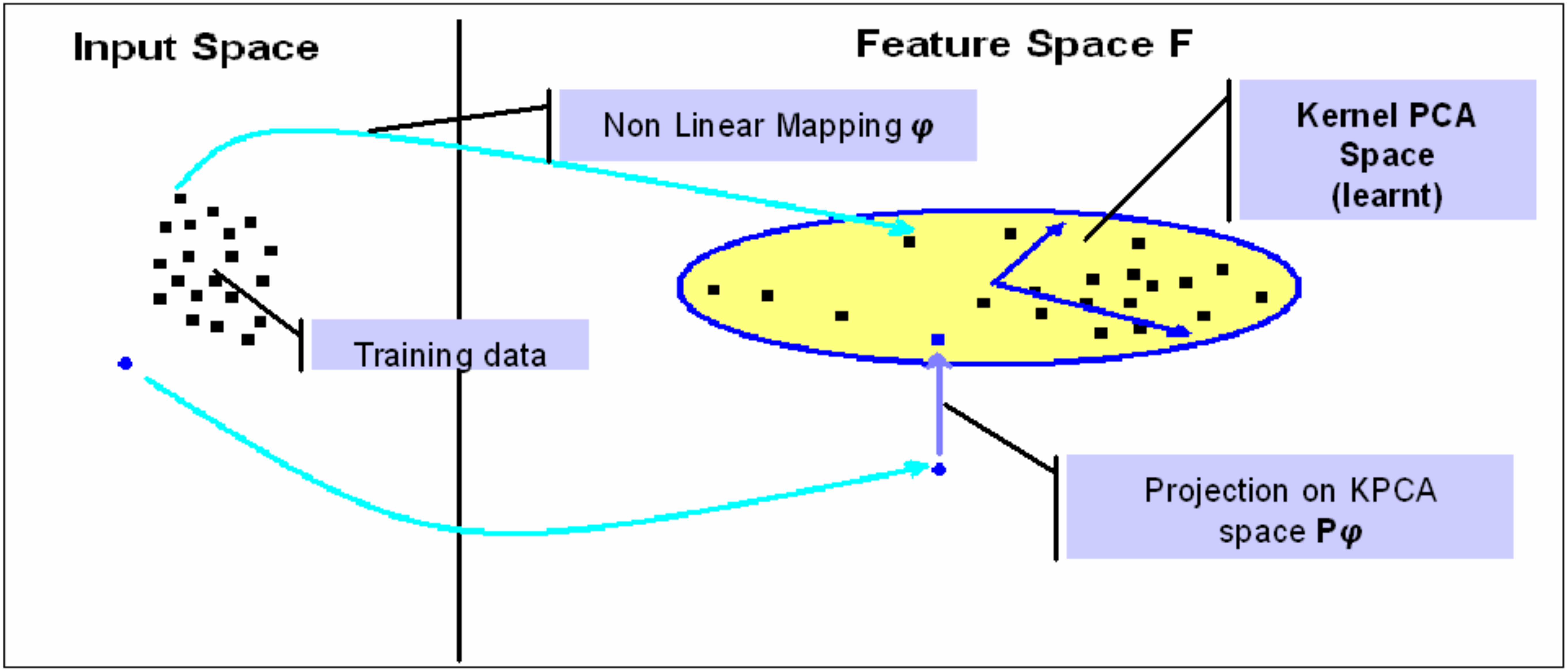
- And the eigen-vectors in the feature space, $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, can be expressed as

$$\mathbf{v}_i = \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n)$$

- With a kernel k, we can use the kernel trick to compute dot products in the feature space without ever having to explicitly compute the mapping:

$$\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$$

Kernel PCA: Illustration



Kernel PCA

- The kernel trick allow us to find the vector of coefficients $\mathbf{a}_i = (a_{1i}, \ldots, a_{ni})^T$representing the eigen-vectors by solving the following problem:

$$\mathbf{K}\mathbf{a}_i = \lambda_i N \mathbf{a}_i$$

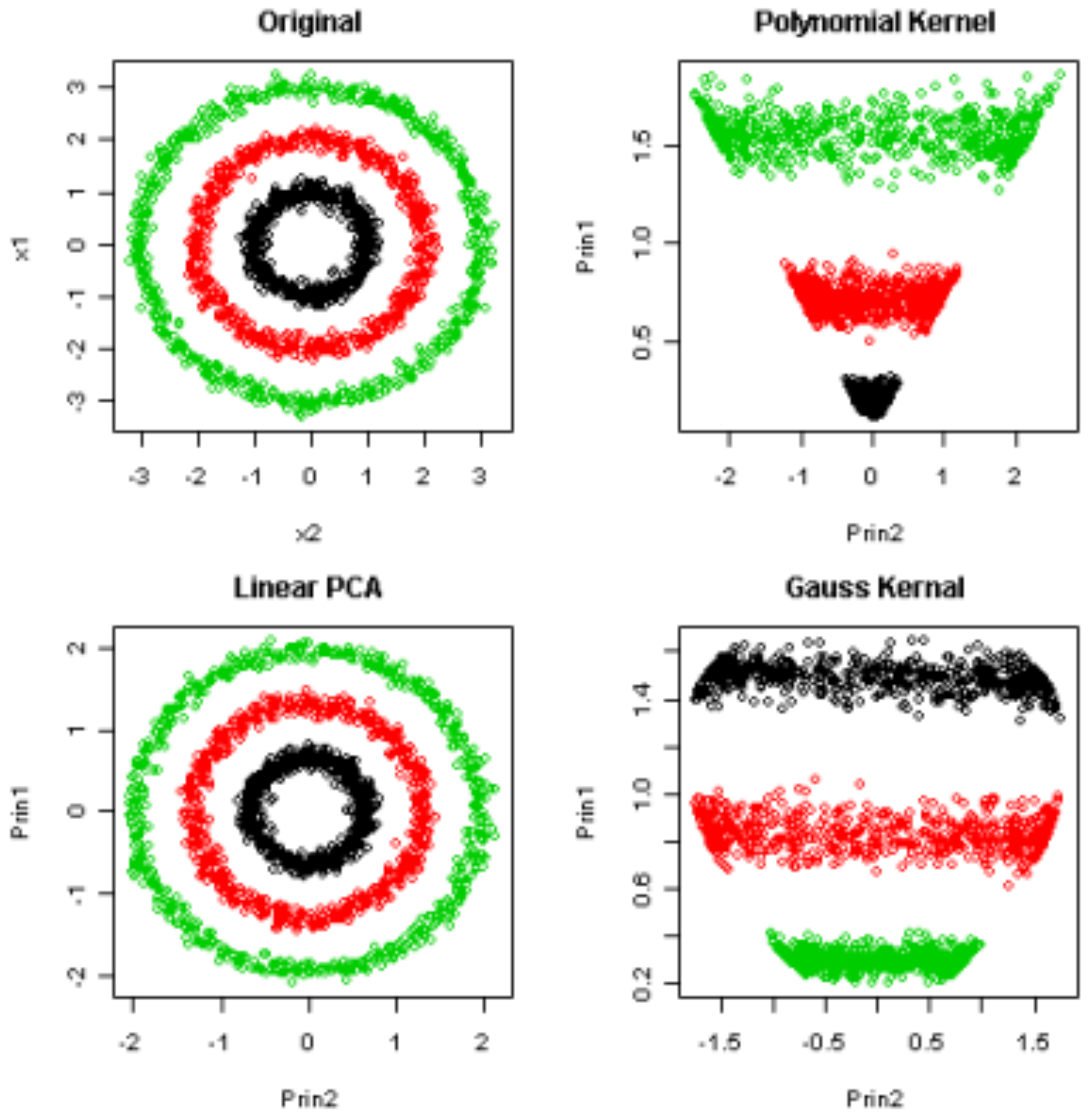
where K is the Gram matrix: $\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$

- It is then easy to compute the projections for the principal components:

$$z_i(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{v}_i = \sum_{n=1}^{N} a_{in} \phi(\mathbf{x})^T \phi(\mathbf{x}_n) = \sum_{n=1}^{N} a_{in} k(\mathbf{x}, \mathbf{x}_n)$$

- (remark: if we want to center the data in the feature space we need to use a corrected Gram matrix, c.f. Bishop 12.3)

Kernel PCA - Example



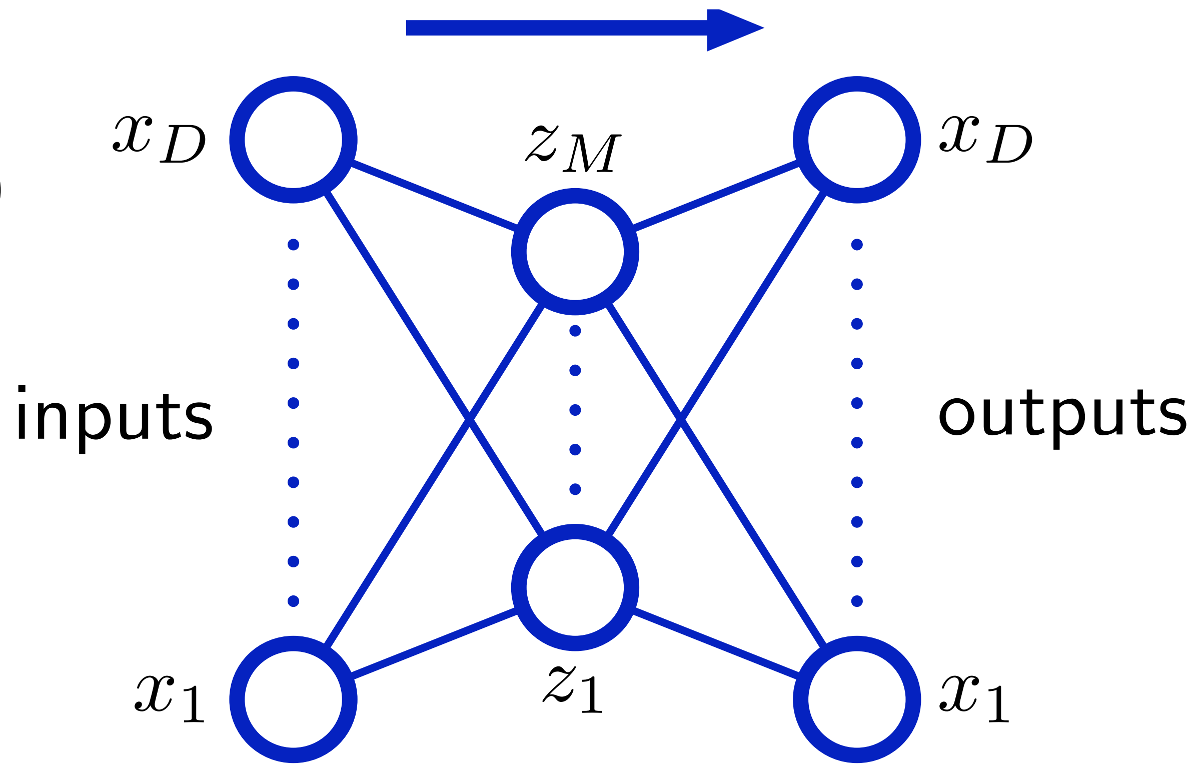
source: http://programmingsas.wordpress.com/page/3/

Kernel PCA: drawbacks

- Need to perform en eigen-decomposition of an NxN Gram matrix (rather than DxD.)
  => This can be nice for high-dimensional data with few examples, but often *N*>>*D*... Not often used with big data sets because too expensive.

AutoEncoders

- An neural network used to reproduce its input (target = input)
- The hidden layer is chosen of dimension M <D. This leads to **reconstruction errors**, which the training seeks to minimize.



- A **reduced-size representation** is obtained at the hidden layer.
- If the network is linear, or if it has only a hidden layer (even non-linear) it is equivalent to the PCA (the weights of the M hidden neurons define the same subspace as the PCA with M components).
- If there are several hidden layers with nonlinearities it is a **nonlinear dimensionality reduction method**.