Normalization of Eigenvectors

$$\boldsymbol{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$$

- Eigenvectors $\{\boldsymbol{\alpha}^{(i)}\}_{i=1}^{m}$ are orthogonal.

$$\langle \boldsymbol{\alpha}^{(i)}, \boldsymbol{\alpha}^{(j)} \rangle = 0 \quad \text{for} \quad i \neq j$$

- When the eigenproblem is solved by some software, eigenvectors would be normalized.

$$\|\boldsymbol{\alpha}^{(i)}\| = 1$$

- What about primal eigenvectors $\{\boldsymbol{\psi}^{(i)}\}_{i=1}^{m}$?

Normalization of Eigenvectors

- $\langle \boldsymbol{\psi}^{(i)}, \boldsymbol{\psi}^{(j)} \rangle = \langle \boldsymbol{K}\boldsymbol{\alpha}^{(i)}, \boldsymbol{\alpha}^{(j)} \rangle$
  $= 0 \quad \text{for} \quad i \neq j$

$$\boldsymbol{\psi}^{(i)} = \sum_{j=1}^{n} \alpha_j^{(i)} \boldsymbol{f}_j$$

- $\|\boldsymbol{\psi}^{(i)}\| = \sqrt{\lambda_i}$
- $\{\boldsymbol{\psi}^{(i)}\}_{i=1}^{m}$ are orthogonal but not normalized!
- Normalization:

$$\boldsymbol{\psi}^{(i)} \longleftarrow \frac{\boldsymbol{\psi}^{(i)}}{\|\boldsymbol{\psi}^{(i)}\|} = \frac{1}{\sqrt{\lambda_i}} \sum_{j=1}^{n} \alpha_j^{(i)} \boldsymbol{f}_j$$

Kernel PCA Projection

- Kernel PCA projection of a feature vector $\boldsymbol{f}' = \phi(\boldsymbol{x}')$:

$$\boldsymbol{g}' = \boldsymbol{B}_{KPCA}\boldsymbol{f}' \qquad \boldsymbol{B}_{KPCA} = (\boldsymbol{\psi}_1|\boldsymbol{\psi}_2|\cdots|\boldsymbol{\psi}_m)^\top$$

- Since $\langle \boldsymbol{f}', \boldsymbol{\psi}_i \rangle = \frac{1}{\sqrt{\lambda_i}} \sum_{j=1}^{n} \alpha_j^{(i)} K(\boldsymbol{x}', \boldsymbol{x}_j)$

  we have

$$\boldsymbol{g}' = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{A}\boldsymbol{k}'$$

$$\boldsymbol{A} = (\boldsymbol{\alpha}^{(1)}|\boldsymbol{\alpha}^{(2)}|\cdots|\boldsymbol{\alpha}^{(m)})^\top$$

$$\boldsymbol{\Lambda} = \mathrm{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_m\right)$$

$$\boldsymbol{k}' = (K(\boldsymbol{x}', \boldsymbol{x}_1), K(\boldsymbol{x}', \boldsymbol{x}_2), \ldots K(\boldsymbol{x}', \boldsymbol{x}_n))^\top$$

Kernel PCA Projection

$$\langle \boldsymbol{f}_i, \boldsymbol{\psi}_k \rangle = \frac{1}{\sqrt{\lambda_k}} \sum_{j=1}^{n} \alpha_j^{(k)} K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

- If we embed given feature vectors $\{\boldsymbol{f}_i\}_{i=1}^{n}$

$$\boldsymbol{G} = (\boldsymbol{g}_1 | \boldsymbol{g}_2 | \cdots | \boldsymbol{g}_n)$$
$$= \boldsymbol{\Lambda}^{-1/2} \boldsymbol{A} \boldsymbol{K}$$
$$= \boldsymbol{\Lambda}^{1/2} \boldsymbol{A}$$

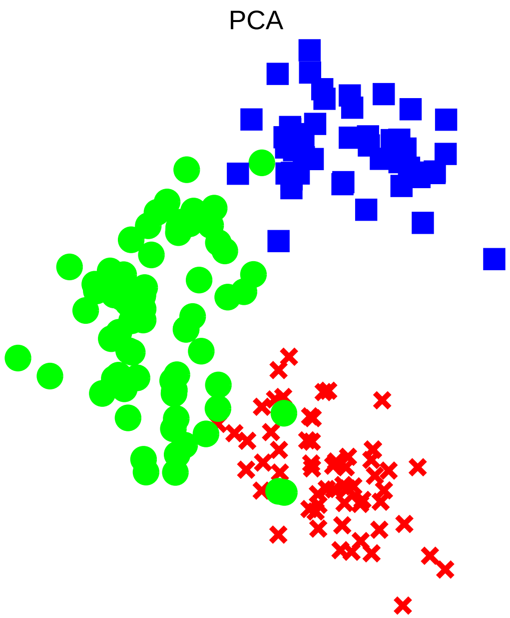
$$\boldsymbol{A}\boldsymbol{K} = \boldsymbol{\Lambda}\boldsymbol{A}$$

$$\boldsymbol{A} = (\boldsymbol{\alpha}^{(1)} | \boldsymbol{\alpha}^{(2)} | \cdots | \boldsymbol{\alpha}^{(m)})^\top$$
$$\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$$
$$\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

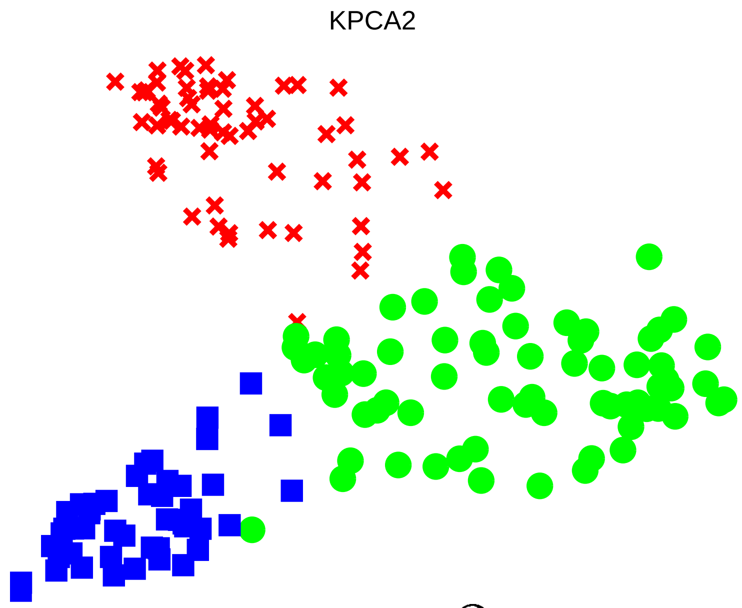
Example

- Wine data (UCI): 13-dim, 178 samples

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2 / c^2\right)$$



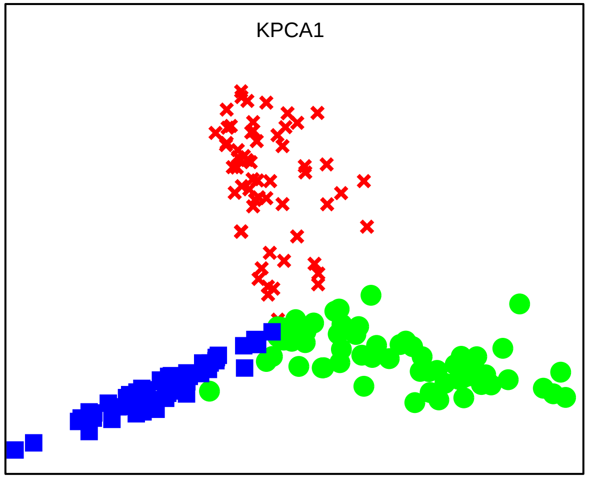
Linear PCA



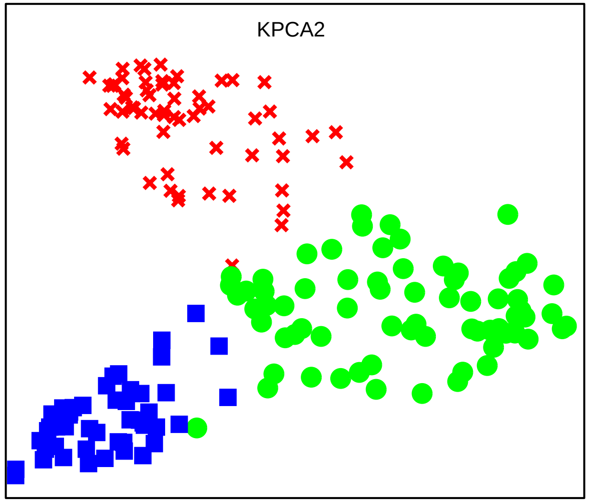
$c = 3$

Gaussian KPCA

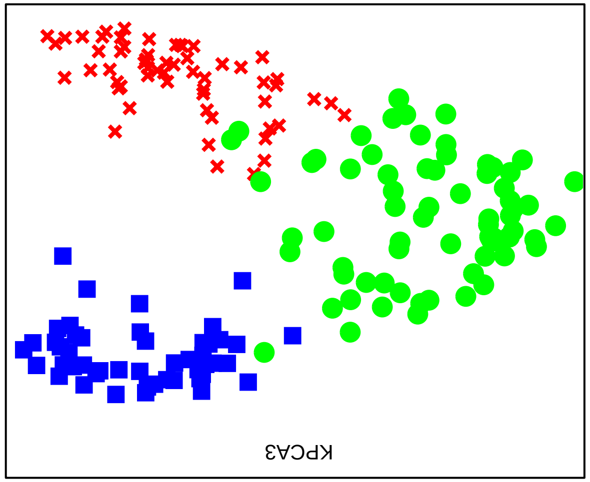
Example (cont.)



$c = 2$



$c = 3$

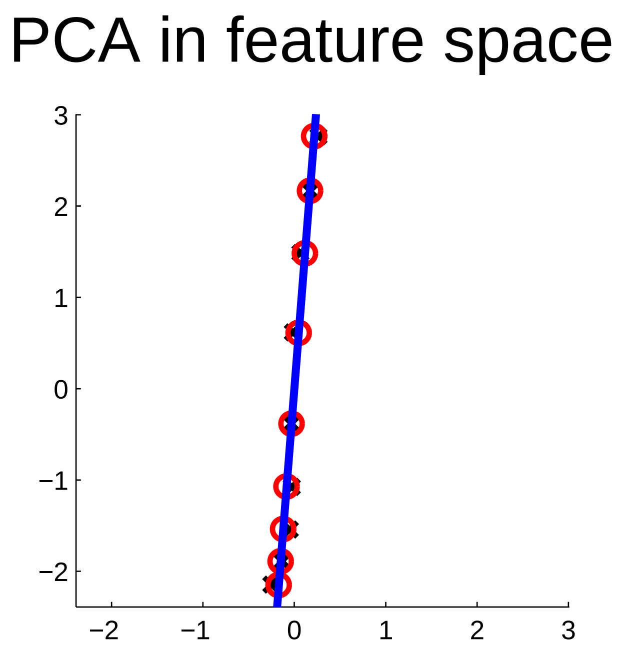


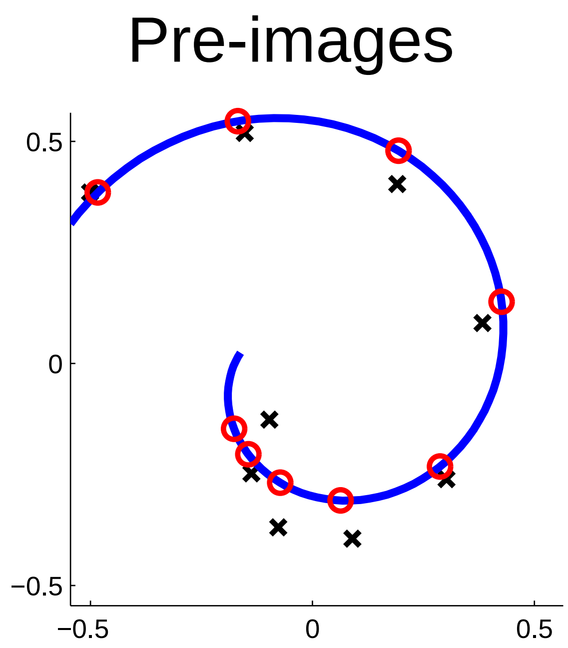
$c = 6$

- Choice of kernels (type and parameter) depends on the result.
- Appropriately choosing kernels is not easy in practice.

Pre-Images

- Pre-images: the embedded data pulled back in the original input space.
- Obtaining pre-images is sometimes useful to interpret the result.





Pre-Images (cont.)

- When an inverse mapping $\phi^{-1}(\boldsymbol{z})$ exists, pre-images can be obtained.
- Otherwise it is in principle impossible.
- Idea: Find approximate pre-images:
  - Naïve idea:

$$\min_{\boldsymbol{x}} \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_0)\|^2$$

  - What else?

Suggestion

- If you are interested in the pre-image problem, the following article would be interesting.
- J.T. Kwok and I.W. Tsang.
  The pre-image problem in kernel methods.
  IEEE Transactions on Neural Networks,
  15(6):1517-1525, Nov 2004

Kernel Trick Revisited

$$\langle \boldsymbol{f}_i, \boldsymbol{f}_j \rangle = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

- An inner product in the feature space can be efficiently calculated by the kernel function.
- If a linear algorithm is expressed only in terms of the inner product, it can be non-linearlized by the kernel trick:
  - Principal component analysis
  - Locality preserving projection
  - K-means clustering
  - Perceptron (support vector machine)
  - Fisher discriminant analysis

LPP in Feature Space

- $\{\boldsymbol{f}_i \mid \boldsymbol{f}_i = \phi(\boldsymbol{x}_i)\}_{i=1}^{n}$: Feature vectors
- Suppose $\mathrm{rank}\,(\boldsymbol{F}) = n$ $\qquad \boldsymbol{F} = (\boldsymbol{f}_1|\boldsymbol{f}_2|\cdots|\boldsymbol{f}_n)$
- Eigenproblem:

$$\boldsymbol{F}\boldsymbol{L}\boldsymbol{F}^\top\psi = \lambda'\boldsymbol{F}\boldsymbol{D}\boldsymbol{F}^\top\psi \quad \text{(A)}$$

- (A) has n positive generalized eigenvalues:

$$\lambda'_1 \geq \lambda'_2 \geq \cdots \geq \lambda'_n > 0$$

- Associated generalized eigenvectors: $\{\psi_i\}_{i=1}^{n}$
- Embedding of $\boldsymbol{x}'$: $\qquad \boldsymbol{g}' = \boldsymbol{B}_{LPP}\boldsymbol{f}'$

$$\boldsymbol{B}_{LPP} = (\psi_{n-m+1}|\psi_{n-m+2}|\cdots|\psi_n)^\top \qquad \boldsymbol{f}' = \phi(\boldsymbol{x}')$$

Dual Generalized Eigenproblem

$$\boldsymbol{KLK\alpha} = \lambda \boldsymbol{KDK\alpha} \qquad \text{(B)}$$

$$\boldsymbol{K}_{i,j} = \langle \boldsymbol{f}_i, \boldsymbol{f}_j \rangle$$

- (B) has n positive generalized eigenvalues:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0$$

- Associated generalized eigenvectors: $\{\boldsymbol{\alpha}^{(i)}\}_{i=1}^{n}$
- Then eigenvectors $\{\psi_i\}_{i=1}^{n}$ are given by

$$\psi_i = \sum_{j=1}^{n} \alpha_j^{(i)} \boldsymbol{f}_j + \boldsymbol{f}^{\perp}$$

Proof

- ψ is expressed by using some $\boldsymbol{\alpha}$ and $\langle \boldsymbol{f}^{\perp}, \boldsymbol{f}_i \rangle = 0 \text{ for all } i$ as

$$\psi = \sum_{j=1}^{n} \alpha_j \boldsymbol{f}_j + \boldsymbol{f}^{\perp}$$

- Then (A) is expressed as

$$\boldsymbol{F}\boldsymbol{L}\boldsymbol{K}\boldsymbol{\alpha} = \lambda \boldsymbol{F}\boldsymbol{D}\boldsymbol{K}\boldsymbol{\alpha} \quad \text{(C)}$$

- Multiplying $\boldsymbol{F}^{\top}$ to (C) from left-hand side, we have (B)

Kernel LPP

$$\psi_i = \sum_{j=1}^{n} \alpha_j^{(i)} \boldsymbol{f}_j + \boldsymbol{f}^{\perp}$$

- $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$
- Let $\boldsymbol{f}^{\perp} = \boldsymbol{0}$.
- Embedding of $\boldsymbol{x}'$:

$$\boldsymbol{g}' = \boldsymbol{A}^{\top} \boldsymbol{k}'$$

$$\boldsymbol{A} = (\boldsymbol{\alpha}^{(n-m+1)} | \boldsymbol{\alpha}^{(n-m+2)} | \cdots | \boldsymbol{\alpha}^{(n)})^{\top}$$

$$\boldsymbol{k}' = (K(\boldsymbol{x}', \boldsymbol{x}_1), K(\boldsymbol{x}', \boldsymbol{x}_2), \ldots K(\boldsymbol{x}', \boldsymbol{x}_n))^{\top}$$

Kernel LPP Embedding for Given Features

- Kernel LPP embedding for given features:

$$\begin{aligned} \boldsymbol{G} &= (\boldsymbol{g}_1 | \boldsymbol{g}_2 | \cdots | \boldsymbol{g}_n) \\ &= \boldsymbol{A}\boldsymbol{K} \end{aligned}$$

- Let $\boldsymbol{G} = (\boldsymbol{y}^{(n-m+1)} | \boldsymbol{y}^{(n-m+2)} | \cdots | \boldsymbol{y}^{(n)})^\top$

$$\boldsymbol{y}^{(j)} = \boldsymbol{K}\boldsymbol{\alpha}^{(j)}$$

Kernel LPP Embedding for Given Features (cont.)

- $\boldsymbol{y}^{(j)}$ can be directly obtained as follows.
- Since $\boldsymbol{y}^{(j)} = \boldsymbol{K}\boldsymbol{\alpha}^{(j)}$, $\boldsymbol{K}\boldsymbol{L}\boldsymbol{K}\boldsymbol{\alpha} = \lambda\boldsymbol{K}\boldsymbol{D}\boldsymbol{K}\boldsymbol{\alpha}$ yields

$$\boldsymbol{K}\boldsymbol{L}\boldsymbol{y} = \lambda\boldsymbol{K}\boldsymbol{D}\boldsymbol{y}$$

- Solution of $\boldsymbol{K}\boldsymbol{L}\boldsymbol{y} = \lambda\boldsymbol{K}\boldsymbol{D}\boldsymbol{y}$ is given by the following simpler eigenproblem.

$$\boldsymbol{L}\boldsymbol{y} = \lambda\boldsymbol{D}\boldsymbol{y}$$

- Note: When similarity matrix $\boldsymbol{W}$ is sparse, $\boldsymbol{L}$ and $\boldsymbol{D}$ are also sparse!

Laplacian Eigenmap Embedding

$$\boldsymbol{L}\boldsymbol{y} = \lambda \boldsymbol{D}\boldsymbol{y}$$

$$\boldsymbol{D} = \mathrm{diag}(\textstyle\sum_{j=1}^{n} \boldsymbol{W}_{i,j})$$

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$$

- Definition of $\boldsymbol{L}$ implies $\boldsymbol{L}\mathbf{1} = \mathbf{0}$

$$\Rightarrow \boldsymbol{y}^{(n)} = \mathbf{1}$$

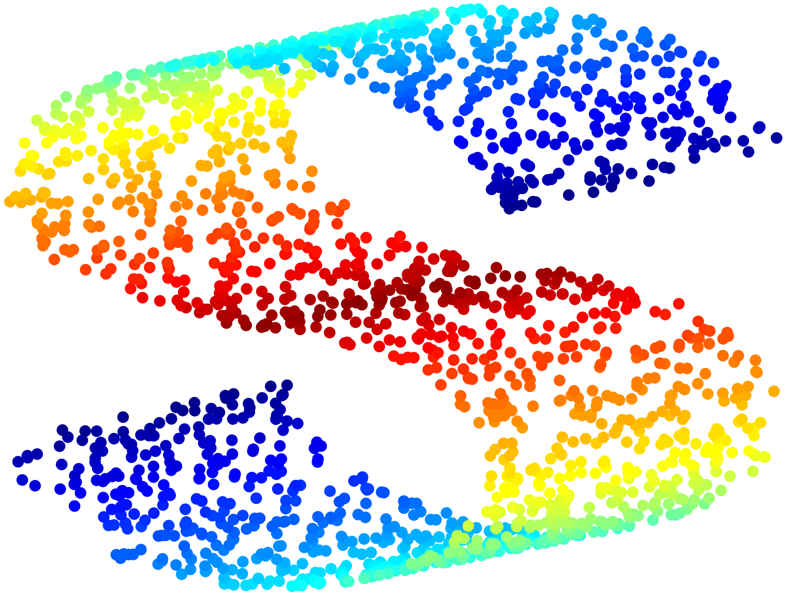
- In practice, we remove $\boldsymbol{y}^{(n)}$ and use

$$\boldsymbol{G} = (\boldsymbol{y}^{(n-m)} | \boldsymbol{y}^{(n-m+1)} | \cdots | \boldsymbol{y}^{(n-1)})^{\top}$$

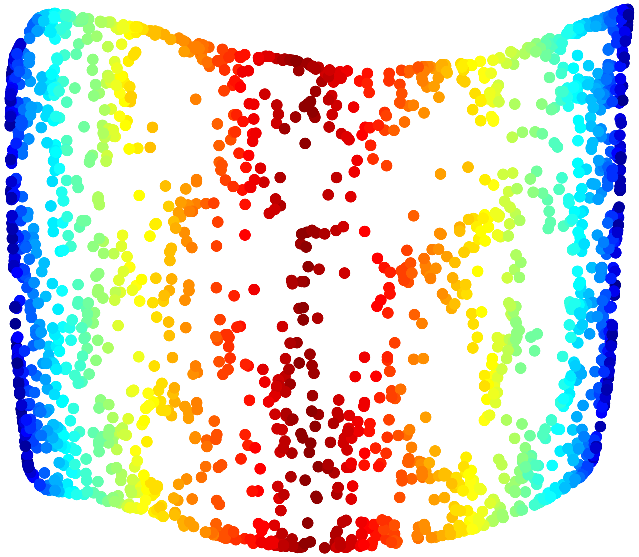
- This non-linear embedding method is called Laplacian eigenmap embedding.

Example

Original data (3D)



Embedded Data (2D)



- Laplacian eigenmap can successfully unfold the non-linear manifold.

Non-Linear Dimensionality Reduction Methods: Summary

Method	Advantages	Disadvantages
Kernel PCA	Highly Flexible	How to choose kernels is not clear
Kernel LPP	Local structure preservation in a non-linear fashion	How to choose kernels is not clear

Dimensionality Reduction Methods: Summary

Linear	Non-Linear
Principal Component Analysis (PCA)	Kernel PCA
Locality Preserving Projection (LPP)	Kernel LPP (Laplacian Eigenmap)
Projection Pursuit (PP)	
Non-Gaussian Component Analysis (NGCA)	

Homework

1. Data visualization: Embed your data sets into 2- or 3-dimensional subspace by kernel PCA or Laplacian eigenmap.
2. Data mining: Find something interesting from the visualized data.