Advanced Data Analysis: More on Kernels

Masashi Sugiyama (Computer Science)

W8E-505, sugi@cs.titech.ac.jp

http://sugiyama-www.cs.titech.ac.jp/~sugi

Kernel Trick with Reproducing Kernel

- For some transformation $\phi(\boldsymbol{x})$ $(= \boldsymbol{f})$, there exists a bivariate function $K(\boldsymbol{x}, \boldsymbol{x}')$ such that

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

- Such implicit mapping $\phi(\boldsymbol{x})$ exists if
  - $\boldsymbol{K}$ is symmetric: $\boldsymbol{K}^\top = \boldsymbol{K}$
  - $\boldsymbol{K}$ is positive semi-definite: $\forall \boldsymbol{y}, \ \ \langle \boldsymbol{K}\boldsymbol{y}, \boldsymbol{y} \rangle \geq 0$

Combination of Reproducing Kernels

1. Positive scaling of RK is still RK

$$K(\boldsymbol{x},\boldsymbol{x}') = \alpha K^{(1)}(\boldsymbol{x},\boldsymbol{x}') \qquad \alpha > 0$$

2. Sum of RKs is still RK:

$$K(\boldsymbol{x},\boldsymbol{x}') = K^{(1)}(\boldsymbol{x},\boldsymbol{x}') + K^{(2)}(\boldsymbol{x},\boldsymbol{x}')$$

3. Product of RKs is still RK:

$$K(\boldsymbol{x},\boldsymbol{x}') = K^{(1)}(\boldsymbol{x},\boldsymbol{x}')K^{(2)}(\boldsymbol{x},\boldsymbol{x}')$$

Proof

$$K(\boldsymbol{x},\boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}')\rangle$$

$$K^{(i)}(\boldsymbol{x},\boldsymbol{x}') = \langle \boldsymbol{\phi}^{(i)}(\boldsymbol{x}), \boldsymbol{\phi}^{(i)}(\boldsymbol{x}')\rangle$$

1. For $\boldsymbol{\phi}(\boldsymbol{x}) = \sqrt{\alpha}\boldsymbol{\phi}^{(1)}(\boldsymbol{x})$,

$$K(\boldsymbol{x},\boldsymbol{x}') = \alpha\langle \boldsymbol{\phi}^{(1)}(\boldsymbol{x}), \boldsymbol{\phi}^{(1)}(\boldsymbol{x}')\rangle = \alpha K^{(1)}(\boldsymbol{x},\boldsymbol{x}')$$

2. For $\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} \boldsymbol{\phi}^{(1)}(\boldsymbol{x}) \\ \boldsymbol{\phi}^{(2)}(\boldsymbol{x}) \end{pmatrix}$,

$$\begin{aligned} K(\boldsymbol{x},\boldsymbol{x}') &= \langle \boldsymbol{\phi}^{(1)}(\boldsymbol{x}), \boldsymbol{\phi}^{(1)}(\boldsymbol{x}')\rangle + \langle \boldsymbol{\phi}^{(2)}(\boldsymbol{x}), \boldsymbol{\phi}^{(2)}(\boldsymbol{x}')\rangle \\ &= K^{(1)}(\boldsymbol{x},\boldsymbol{x}') + K^{(2)}(\boldsymbol{x},\boldsymbol{x}') \end{aligned}$$

3. For $[\boldsymbol{\phi}(\boldsymbol{x})]_{i,j} = [\boldsymbol{\phi}^{(1)}(\boldsymbol{x})]_i[\boldsymbol{\phi}^{(2)}(\boldsymbol{x})]_j$,

$$\begin{aligned} K(\boldsymbol{x},\boldsymbol{x}') &= \sum_{i,j}[\boldsymbol{\phi}^{(1)}(\boldsymbol{x})]_i[\boldsymbol{\phi}^{(2)}(\boldsymbol{x})]_j[\boldsymbol{\phi}^{(1)}(\boldsymbol{x}')]_i[\boldsymbol{\phi}^{(2)}(\boldsymbol{x}')]_j \\ &= \langle \boldsymbol{\phi}^{(1)}(\boldsymbol{x}), \boldsymbol{\phi}^{(1)}(\boldsymbol{x}')\rangle\langle \boldsymbol{\phi}^{(2)}(\boldsymbol{x}), \boldsymbol{\phi}^{(2)}(\boldsymbol{x}')\rangle \\ &= K^{(1)}(\boldsymbol{x},\boldsymbol{x}')\ K^{(2)}(\boldsymbol{x},\boldsymbol{x}') \end{aligned}$$

Exercise: Playing with Kernel Trick

- Norm:

$$\|\boldsymbol{f}_i\|^2 = \sqrt{K(\boldsymbol{x}_i, \boldsymbol{x}_i)}$$

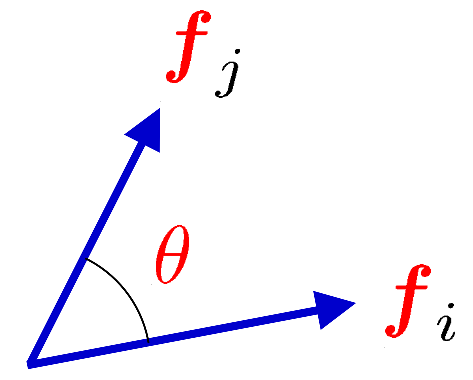
- Distance:

$$\|\boldsymbol{f}_i - \boldsymbol{f}_j\|^2 = K(\boldsymbol{x}_i, \boldsymbol{x}_i) - 2K(\boldsymbol{x}_i, \boldsymbol{x}_j) + K(\boldsymbol{x}_j, \boldsymbol{x}_j)$$

- Angle:

$$\cos\theta = \frac{K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sqrt{K(\boldsymbol{x}_i, \boldsymbol{x}_i)K(\boldsymbol{x}_j, \boldsymbol{x}_j)}}$$

$$\langle \boldsymbol{f}_i, \boldsymbol{f}_j \rangle = \|\boldsymbol{f}_i\|\|\boldsymbol{f}_j\| \cos\theta$$

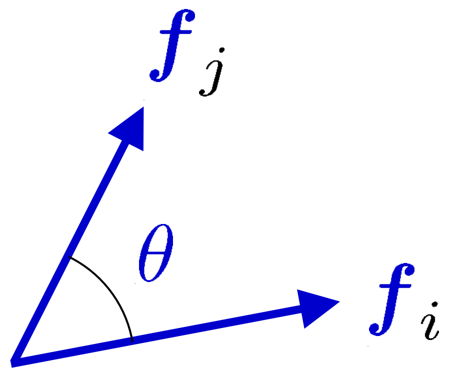


Playing with Kernel Trick (cont.)

- In particular, for Gaussian kernels,
  - $\|\boldsymbol{f}_i\|^2 = 1$
  - $\|\boldsymbol{f}_i - \boldsymbol{f}_j\|^2 = 2 - 2K(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  - $\cos\theta = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

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

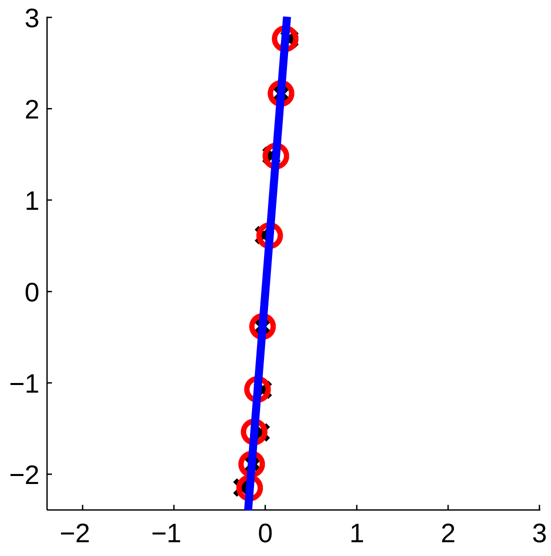
$$c > 0$$



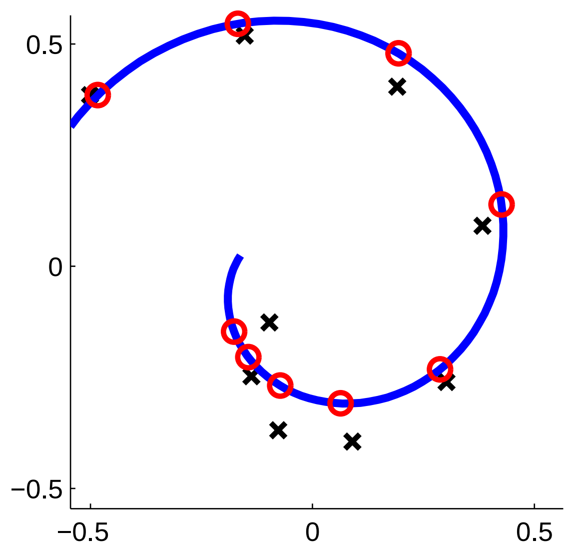
Pre-Images

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

PCA in feature space



Pre-images



Pre-Images (cont.)

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

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

  - 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, 2004.

    http://ieeexplore.ieee.org/iel5/72/29733/01353287.pdf

  - G. H. Bakir, J. Weston, and B. Schoelkopf, Learning to Find Pre-Images, In *Advances in Neural Information Processing Systems 16*, 2004.

    http://books.nips.cc/papers/files/nips16/NIPS2003_AA57.pdf

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 computed 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:
  - PCA, LPP, FDA, LFDA
  - K-means clustering
  - Perceptron (support vector machine)

Kernel LPP

- Kernel LPP embedding of a sample $\boldsymbol{f}$:

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

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

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

  - $\{\lambda_i, \boldsymbol{\alpha}_i\}_{i=1}^m$:Sorted generalized eigenvalues and normalized eigenvectors of $\boldsymbol{KLK\alpha} = \lambda \boldsymbol{KDK\alpha}$

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \qquad \langle \boldsymbol{KDK\alpha}_i, \boldsymbol{\alpha}_j \rangle = \delta_{i,j}$$

$$\boldsymbol{K} = \boldsymbol{F}^\top \boldsymbol{F} \qquad \boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$$

$$\boldsymbol{F} = (\boldsymbol{f}_1 | \boldsymbol{f}_2 | \cdots | \boldsymbol{f}_n) \qquad \boldsymbol{D} = \mathrm{diag}(\textstyle\sum_{j=1}^n \boldsymbol{W}_{i,j})$$

- Note: When $\boldsymbol{KDK}$ is not full-rank, it should be replaced by $\boldsymbol{KDK} + \varepsilon \boldsymbol{I}_n$.

ε :small positive scalar

Kernel LPP Embedding of Given Features

- Kernel LPP embedding of $\{\boldsymbol{f}_i\}_{i=1}^n$:

$$\boldsymbol{G} = \boldsymbol{A}^\top \boldsymbol{K} \qquad \boldsymbol{G} = (\boldsymbol{g}_1 | \boldsymbol{g}_2 | \cdots | \boldsymbol{g}_n)$$

- $\boldsymbol{G}$ can be directly obtained as

$$\boldsymbol{G} = \boldsymbol{\Phi}^\top \qquad \boldsymbol{\Phi} = (\boldsymbol{\varphi}_{n-m+1} | \boldsymbol{\varphi}_{n-m+2} | \cdots | \boldsymbol{\varphi}_n)$$

  - $\{\gamma_i, \boldsymbol{\varphi}_i\}_{i=1}^m$:Sorted eigenvalues and normalized eigenvectors of $\boldsymbol{L}\boldsymbol{\varphi} = \gamma \boldsymbol{D}\boldsymbol{\varphi}$

$$\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_n \qquad \langle \boldsymbol{D}\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j \rangle = \delta_{i,j}$$

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

Laplacian Eigenmap Embedding

$$\boldsymbol{L}\boldsymbol{\varphi} = \gamma \boldsymbol{D}\boldsymbol{\varphi}$$

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

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

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

  $\Rightarrow \boldsymbol{\varphi}_n \propto \mathbf{1}$

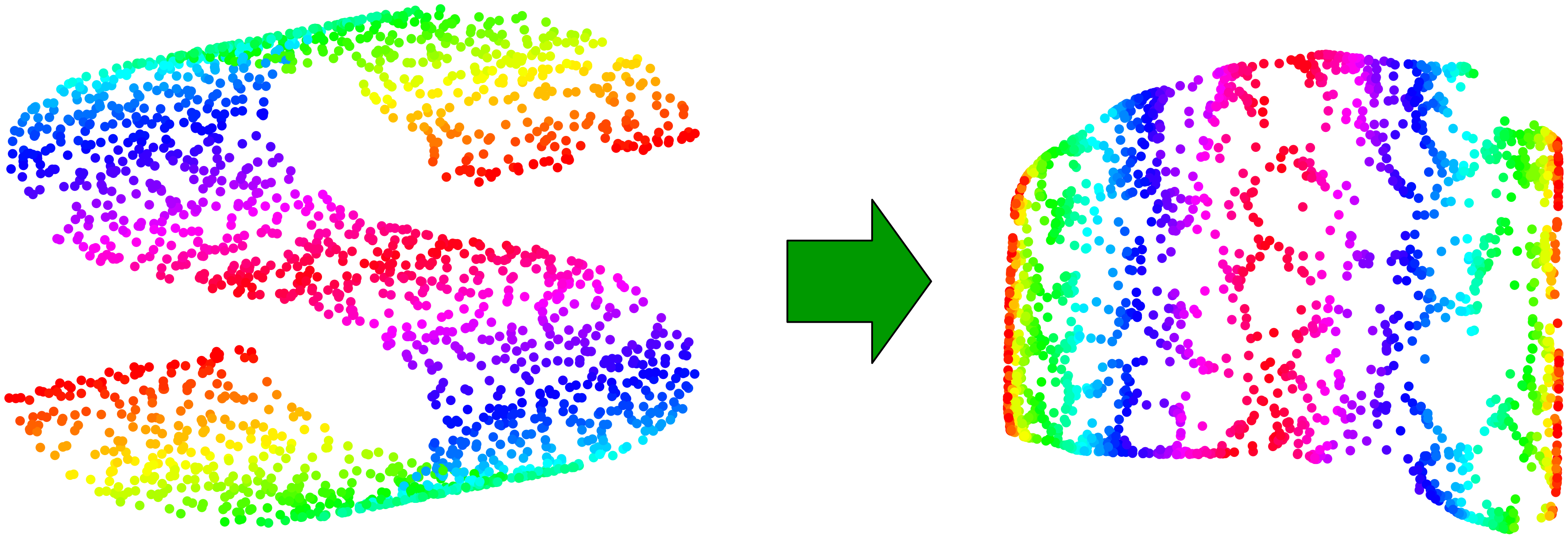
- In practice, we remove $\boldsymbol{\varphi}_n$ and use

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

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

Example

Original data (3D) Embedded Data (2D)



Note: Similarity matrix is defined by the nearest-neighbor-based method with 10 nearest neighbors.

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

Reproducing Kernel Hilbert Space

- Reproducing kernel Hilbert space (RKHS) $\mathcal{H}$: a functional Hilbert space induced by a reproducing kernel $K(\boldsymbol{x}, \boldsymbol{x}')$.

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle_{\mathcal{H}} = K(\boldsymbol{x}, \boldsymbol{x}')$$

- Reproducing property in RKHS:

$$\forall f \in \mathcal{H}, \quad \langle f, \phi(\boldsymbol{x}') \rangle_{\mathcal{H}} = f(\boldsymbol{x}')$$

Kernel Tricks for Measuring Independence

- x, y : one-dimensional random variables with mean zero.
- For a Gaussian RKHS $\mathcal{H}$, x, y are independent if and only if $\rho = 0$.

$$\rho = \max_{f,g \in \mathcal{H}} \mathrm{covariance}(f(x), g(y))$$

$$= \max_{f,g \in \mathcal{H}} \mathbb{E}[\langle f, \phi(x) \rangle \langle g, \phi(y) \rangle]$$

Kernel Tricks for Measuring Independence (cont.)

- If we have samples $\{x_i\}_{i=1}^n, \{y_i\}_{i=1}^n$

$$\rho \approx \max_{f,g\in\mathcal{H}} \left[\frac{1}{n}\sum_{i=1}^{n} \langle f, \boldsymbol{\phi}(x_i)\rangle \langle g, \boldsymbol{\phi}(y_i)\rangle \right] \equiv \widehat{\rho}$$

- Let

$$f = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(x_i) + f^{\perp} \qquad g = \sum_{i=1}^{n} \beta_i \boldsymbol{\phi}(y_i) + g^{\perp}$$

Then

$$\widehat{\rho} = \max_{\{\alpha_i\}_{i=1}^n, \{\beta_i\}_{i=1}^n} \left[\frac{1}{n} \sum_{i,j,k=1}^{n} \alpha_i \beta_j K(x_i, x_k) K(y_j, y_k) \right]$$

Homework

1. Implement Laplacian eigenmap and unfold the 3-dimensional S-curve data.

http://sugiyama-www.cs.titech.ac.jp/~sugi/data/DataAnalysis

Test Laplacian eigenmap with your own (artificial or real) data and analyze its characteristics.

Homework (cont.)

2. Prove that the dual eigenvalue problem of (local) Fisher discriminant analysis is given by

$$\boldsymbol{K}\boldsymbol{L}^{(b)}\boldsymbol{K}\boldsymbol{\alpha} = \lambda \boldsymbol{K}\boldsymbol{L}^{(w)}\boldsymbol{K}\boldsymbol{\alpha}$$

$$\boldsymbol{L}^{(b)} = \boldsymbol{D}^{(b)} - \boldsymbol{W}^{(b)}$$

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

$$\boldsymbol{W}_{i,j}^{(b)} = \begin{cases} 1/n - 1/n_\ell & (y_i = y_j = \ell) \\ 1/n & (y_i \neq y_j) \end{cases}$$

$$\boldsymbol{L}^{(w)} = \boldsymbol{D}^{(w)} - \boldsymbol{W}^{(w)}$$

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

$$\boldsymbol{W}_{i,j}^{(w)} = \begin{cases} 1/n_\ell & (y_i = y_j = \ell) \\ 0 & (y_i \neq y_j) \end{cases}$$

Note that when solving the above eigenproblem, we may need to regularize it as

$$\boldsymbol{K}\boldsymbol{L}^{(b)}\boldsymbol{K}\boldsymbol{\alpha} = \lambda(\boldsymbol{K}\boldsymbol{L}^{(w)}\boldsymbol{K} + \epsilon \boldsymbol{I}_n)\boldsymbol{\alpha}$$

- LFDA can also be kernelized similarly!