15. (Non-Linear) Dimensionality Reduction

Dongwoo Kim

dongwoo.kim@postech.ac.kr

CSED515 - 2023 Spring

From Subspace to Nonlinear Manifold

Linear subspace may be inefficient for some datasets. If the data is embedded on a manifold, we should capture that structure in order for an efficient PCA.

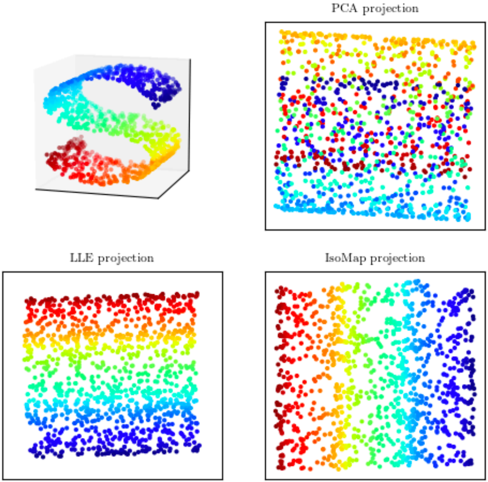
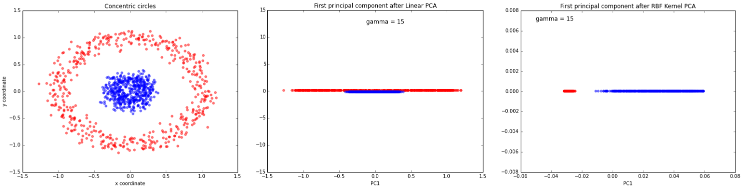


Table of Contents

A Famous Example of Nonlinear Dimensionality Reduction

A better dimensionality reduction can be done by PCA after mapping ϕ of data points to feature space, e.g., $\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$:



https://sebastianraschka.com/Articles/2014_kernel_pca.html

A good dimensionality reduction may provide coincidence between the distance in the reduced dimensionality, and the distance which we believe.

Objective in Kernel PCA

For a given non-linear mapping ϕ, the objective for dimensionality reduction can be formulated as:

$$\underset{V \text{with normalized columns}}{\text{minimize}} \sum_{n=1}^{N} \left\|\phi(x_n) - VV^\top \phi(x_n)\right\|^2 ,$$

i.e., finding eigenvectors of $\bar{S} \stackrel{\text{def}}{=} \frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\phi(x_n)^\top$ (when $\sum_{n=1}^{N} \phi(x_n) = 0$), which may be intractable as the feature space can have larger dimensionality than the original one.

Review: PCA

For a set of zero-mean data, i.e., $\sum_{n=1}^{N} x_n = 0$, the linear/regular PCA solves the eigenvalue equation of the data covariance matrix $S = \frac{1}{N}\sum_{n=1}^{N} x_n x_n^\top$:

$$Su_i = \lambda_i u_i \ ,$$

where u such that $\|u_i\|_2 = u_i^\top u_i = 1$.

Denoting dot-product by $\langle x, y\rangle \stackrel{\text{def}}{=} x^\top y$, note that

$$\begin{aligned} Su_i &= \left(\frac{1}{N}\sum_{n=1}^{N} x_n x_n^\top\right) u_i \\ &= \frac{1}{N}\sum_{n=1}^{N}\langle x_n, u_i\rangle x_n \ , \end{aligned}$$

which implies that all solution u_i's with $\lambda_i \neq 0$ must lies in the span of $x_1, ..., x_N$. Hence, u_i can be represented as

$$u_i = \sum_{n=1}^{N} \alpha_{in} x_n$$

PCA in Feature Space

Consider a nonlinear mapping $\phi : \mathbb{R}^D \mapsto \mathcal{F}$ (feature space).
Assume $\sum_{n=1}^{N} \phi(x_n) = 0$, and define the covariance matrix $\bar{S}$ in feature space $\mathcal{F}$:

$$\bar{S} \stackrel{\text{def}}{=} \frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\phi(x_n)^\top \ .$$

Then, PCA in feature space $\mathcal{F}$ is solving

$$\bar{S} v_i = \lambda_i v_i \quad \text{with } v_i^\top v_i = 1 \ .$$

Again, every solution v_i with $\lambda_i \neq 0$ lies in the span of $\phi(x_1), ..., \phi(x_N)$, which leads to the existence of coefficients $\{\alpha_{in}\}_{n=1,\ldots,N}$ for each solution v_i such that

$$v_i = \sum_{n=1}^{N} \alpha_{in} \phi(x_n) \ .$$

Hence, we will find $\{\alpha_{in}\}$'s.

Kernel Matrix

Define kernel matrix $K \in \mathbb{R}^{N \times N}$ with for $n, m \in [N]$,

$$K_{nm} = K_{mn} = \langle \phi(x_n), \phi(x_m) \rangle = k(x, y) \ .$$

Using $v_i = \sum_{n=1}^{N} \alpha_{in} \phi(x_n)$, the eigenvalue equation is written as:

$$\frac{1}{N} \left(\sum_{n=1}^{N} \phi(x_n) \phi(x_n)^\top \right) \left(\sum_{m=1}^{N} \alpha_{im} \phi(x_m) \right) = \lambda_i \sum_{n=1}^{N} \alpha_{in} \phi(x_n) \ .$$

By multiplying both sides by $\phi(x_\ell)^\top$ to left, and defining $k(x, y) = \phi(x)^\top \phi(y)$, we have

$$\frac{1}{N} \sum_{n=1}^{N} \phi(x_\ell)^\top \phi(x_n) \phi(x_n)^\top \sum_{m=1}^{N} \alpha_{im} \phi(x_m)$$
$$= \frac{1}{N} \sum_{n=1}^{N} K_{\ell n} \phi(x_n)^\top \sum_{m=1}^{N} \alpha_{im} \phi(x_m) = \frac{1}{N} \sum_{n=1}^{N} K_{\ell n} \sum_{m=1}^{N} \alpha_{im} K_{nm} = \lambda_i \sum_{n=1}^{N} \alpha_{in} \phi(x_\ell)^\top \phi(x_n) \ .$$

Denoting a column vector $\alpha_i = [\alpha_{i1}, ..., \alpha_{iN}]^\top$, this gives:

$$\lambda_i N K \alpha_i = K^2 \alpha_i \ .$$

Equivalent Eigenvalue Equation with Kernel Matrix

The equation equivalent the eigenvalue equation in the original space, $\lambda_i N K \alpha_i = K^2 \alpha_i$, can be further simplified as follows:

$$\lambda_i N \alpha_i = K \alpha_i \ ,$$

since two equations differ only by eigenvectors of K having zero eigenvalues that do not affect the principal components projection, i.e., the solution to $\lambda_i N \alpha_i = K \alpha_i$ with non-zero λ_i has to be a solution to $\lambda_i N K \alpha_i = K^2 \alpha_i$.

Note that the computational cost to solve this scales with N rather than the dimensionality of feature space.

Normalization in Kernel PCA

Note that v_i is constrained to be a unit vector. This can be translated into the constraint on α_i as follows:

$$1 = v_i^\top v_i = \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_{in} \alpha_{im} \phi(x_n)^\top \phi(x_m) = \alpha_i^\top K \alpha_i = \lambda_i N \alpha_i^\top \alpha_i \ ,$$

i.e., KPCA finds α_i's such that

$$\lambda_i N \alpha_i = K \alpha_i \ , \quad \text{and} \quad \|\alpha_i\|^2 = \frac{1}{\lambda_i N} \ .$$

Compute Nonlinear Components

In linear PCA, principal components are extracted by projecting the data x onto the eigenvectors u_i of the covariance matrix S, i.e.,

$$\langle u_i, x \rangle \; .$$

In kernel PCA, we also project x onto the eigenvectors v_i of $\bar{S}$, i.e.,

$$\langle v_i, \phi(x) \rangle = \sum_{n=1}^{N} \alpha_{in} \phi(x)^\top \phi(x_n) = \sum_{n=1}^{N} \alpha_{in} k(x, x_n) \; .$$

Centering in Feature Space

So far, we've assumed $\sum_{n=1}^{N} \phi(x_n) = 0$, which is often untrue in practice.

We obtain a closed form of the centered kernel matrix $\tilde{K}$ with $\tilde{K}_{nm} = \langle \tilde{\phi}(x_n), \tilde{\phi}(x_m) \rangle$, where

$$\tilde{\phi}(x_n) \stackrel{\text{def}}{=} \phi(x_n) - \frac{1}{N} \sum_{\ell=1}^{N} \phi(x_\ell) \; .$$

From the definitions,

$$\begin{aligned}
\tilde{K}_{nm} &= \langle \tilde{\phi}(x_n), \tilde{\phi}(x_m) \rangle \\
&= \left\langle \phi(x_n) - \frac{1}{N} \sum_{\ell=1}^{N} \phi(x_\ell), \phi(x_m) - \frac{1}{N} \sum_{\ell=1}^{N} \phi(x_\ell) \right\rangle \\
&= K_{nm} - \frac{1}{N} \sum_{\ell=1}^{N} K_{n\ell} - \frac{1}{N} \sum_{\ell=1}^{N} K_{m\ell} + \frac{1}{N^2} \sum_{\ell=1}^{N} \sum_{\ell'=1}^{N} K_{\ell\ell'}
\end{aligned}$$

which implies the centered kernel matrix $\tilde{K}$ is given by:

$$\tilde{K} = K - 1_N K - K 1_N + 1_N K 1_N \quad \text{with} \quad 1_N \stackrel{\text{def}}{=} \frac{1}{N} \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \vdots & \vdots \\ 1 & \dots & 1 \end{bmatrix} .$$

Summary of KPCA

- Compute the kernel matrix $K \in \mathbb{R}^{N \times N}$, and the centered kernel matrix $\tilde{K}$:

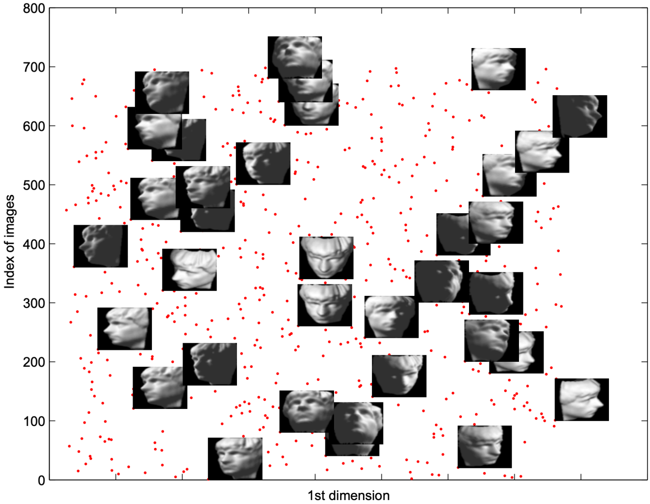
$$\tilde{K} = K - 1_N K - K 1_N + 1_N K 1_N \ .$$

- Solve the eigenvalue problem $N\lambda_i \alpha_i = \tilde{K}\alpha_i$, and normalize α_i such that $\|\alpha_i\|^2 = \frac{1}{N\lambda_i}$.
- For a test pattern x, we extract a nonlinear component via:

$$\langle v_i, x \rangle = \sum_{n=1}^{N} \alpha_{in} k(x_n, x) \ .$$

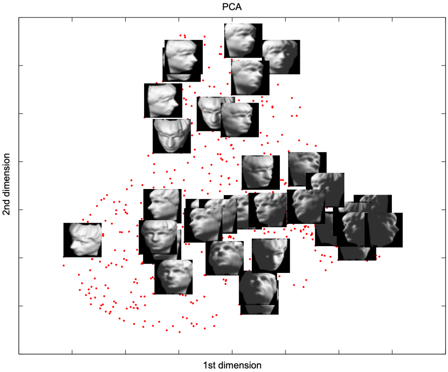
Example of KPCA

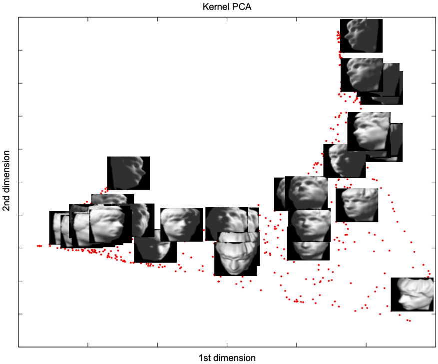
Images of a rotating sculpture: Ghodsi (2006)



Example of KPCA

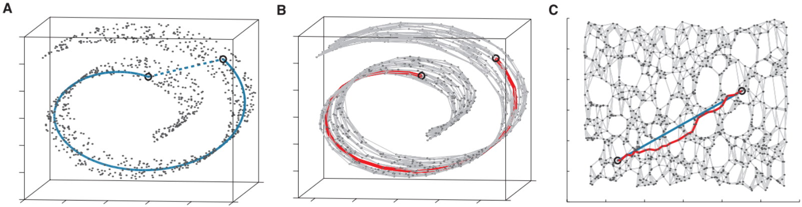
Linear PCA vs. kernel PCA by Ghodsi (2006)





Other Nonlinear Dimensionality Reduction

Isomap (i) generates a graph of data points, where an edge is drawn if the distance between two data points is smaller than certain threshold, and is assigned weight of the distance; (ii) and defines the distance in the feature space as the weights on the shortest path in the data graph.



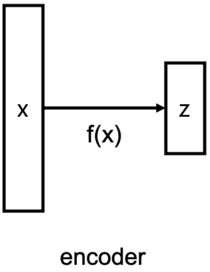
- Note that we need to explicitly describe the distance metric
- More automated approaches?

Table of Contents

Encoder?

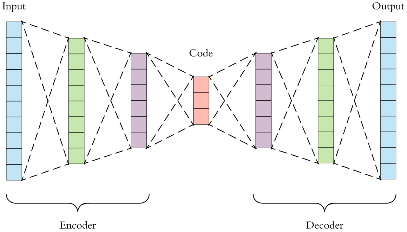
What we need is nothing but just encoder: a function f mapping $x \in \mathbb{R}^D$ to latent feature $z \in \mathbb{R}^M$ $(D \gg M)$

- How do we train f?
- What do we want from f?



Autoencoder

- We want that latent feature $z = f(x)$ captures most information about data x
- If true, we may be able to reconstruct data x from latent feature z
- Consider the following autoencoder structure, which encodes data to feature and decodes from feature to data:



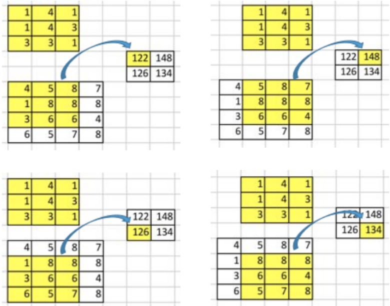
$$x \to f(x) = z \to g(z) = \hat{x}$$

  - Loss function $d(x, g(f(x)))$ for some distance metric d, e.g., $\|x - g(f(x))\|^2$
  - Encoder f and decoder g can be parametric functions, e.g., neural network (MLP, CNN, ...)

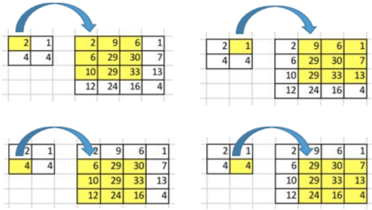
A Typical Choice of f and g for Image

To process image data, convolutional operator considering spatial correlation is widely used

For encoder f, convolution



For decoder g, transposed-convolution

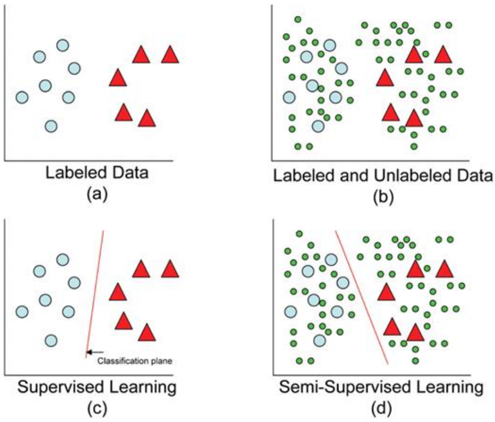


from https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0

An Application of Autoencoder: Semisupervised learning

Suppose we want to perform binary image classification task

- Since labeling is expensive (money and time), un-labeled $\gg$ labeled
- Is there any way to exploit unlabeled data? Semi-supervised learning

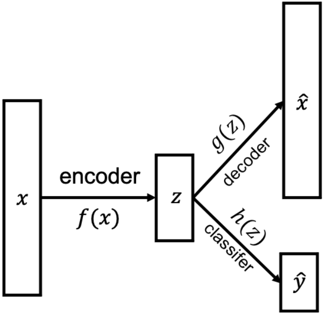


from https://www.digitalvidya.com/blog/semi-supervised-learning/

An Application of Autoencoder: Semisupervised learning

A learning framework of semisupervised learning

- Step1. Train autoencoder for the entire dataset, where encoder f compresses image $x \in \mathbb{R}^D$ into feature $z \in \mathbb{R}^M$
- Step2. Construct a classifier $h : \mathbb{R}^M \to [0, 1]$ using feature, i.e., classifier $h(f(x))$ for image x
- Step3. Fine-tune h using the labeled dataset



Remark: Dimensionality Reduction for Visualization

- So far, we've learned about dimensionality reductions of which goal is information preserving, i.e., data compression
- For visualization, such approaches might not be suitable
- What we want for visualization is similarity preserving rather than information preserving
- A popular tool for visualization: t-SNE (t-Stochastic Neighbor Embedding)

Example of t-SNE

To explain what a model has been learning, one can use t-SNE plot for feature vectors of train/test dataset.

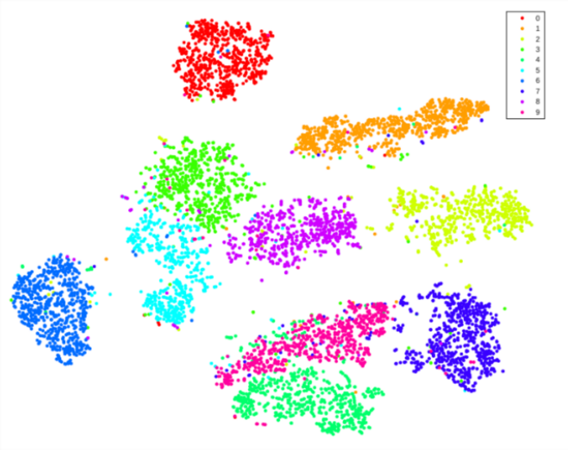


Figure 1 : Illustration of t-SNE on MNIST dataset