Principal Component Analysis

Biswajit Biswas
Ph.D.(C.S.E)

Department of Computer Science
VIVEKANANDA CENTENARY COLLEGE
RAHARA, KOLKATA-700 118

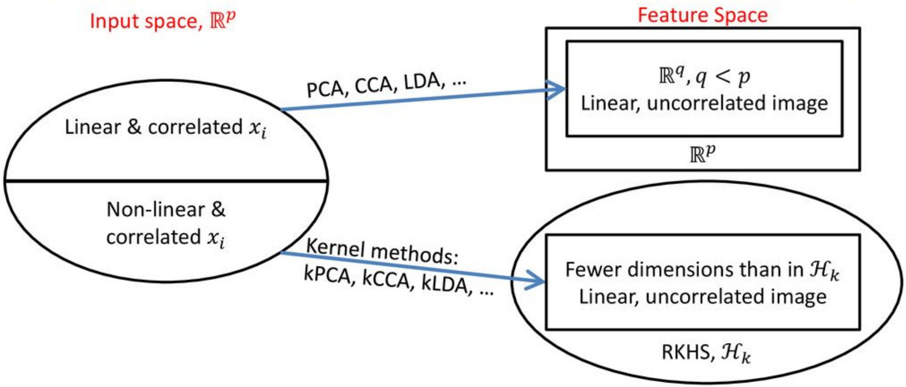
March 29, 2025

Dimensionality reduction

- Dimensionality reduction is the process of reducing the number of random variables or attributes or features under consideration;
- Dimensionality reduction methods include principal components analysis (PCA), which is a linear method that transforms or projects the original data onto a smaller space;
- Attribute subset selection is a method of dimensionality reduction in which irrelevant, weakly relevant, or redundant attributes or dimensions are detected and removed;
- There are many linear and nonlinear methods for dimensionality reduction such as PCA, kernel PCA and ICA;

Dimensionality reduction

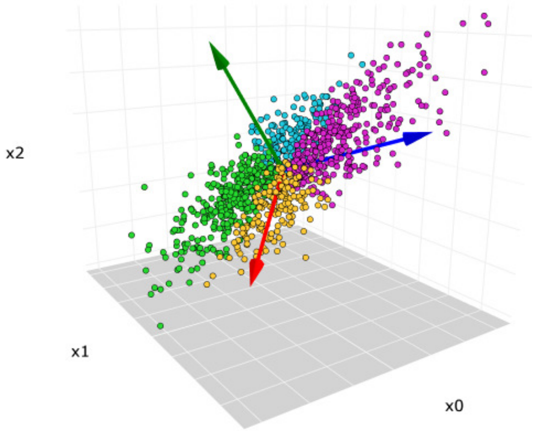
Dimension reduction


Input space, $\mathbb{R}^p$
Linear & correlated x_i
Non-linear & correlated x_i
PCA, CCA, LDA, …
Kernel methods: kPCA, kCCA, kLDA, …
Feature Space
$\mathbb{R}^q, q < p$
Linear, uncorrelated image
$\mathbb{R}^p$
Fewer dimensions than in $\mathcal{H}_k$
Linear, uncorrelated image
RKHS, $\mathcal{H}_k$

- PCA, LDA, CCA: feature extraction and dimension reduction
  - Note: dimension reduction is in feature space, not the input space
    - So, the model uses fewer features, not observed variables

Principal Component Analysis (PCA)

- Principal component analysis (PCA) is a statistical technique of representing high-dimensional data in a low-dimensional space;
- PCA is usually used to reduce the dimensionality of data so that the data can be further visualized or analyzed in a low-dimensional space;



Principal Component Analysis (PCA)

1. The given data $[X]_d^n$ to be reduced consist of tuples or data vectors described by d attributes or dimensions.
2. Principal components analysis (PCA; also called the **Karhunen-Loeve**, or **K-L** method) searches for *kd*-dimensional orthonormal vectors that can best be used to represent the data, where $k \leq d$.
3. The original data are thus projected onto a much smaller space, resulting in dimensionality reduction;
4. The initial data can then be projected onto this smaller set;

Principal Component Analysis (PCA)

The basic procedure of the PCA is as follows:

Step I: The input data are normalized, so that each attribute falls within the same range;

Step II: PCA computes k-orthonormal vectors that provide a basis for the normalized input data and these unit vectors are perpendicular with each other;

Step III: The principal components are sorted in order of decreasing "significance" or strength;

Step IV: The components are sorted in descending order of "significance" by which the data size can be reduced by eliminating the weaker components with low variance;

Principal Component Analysis (PCA)

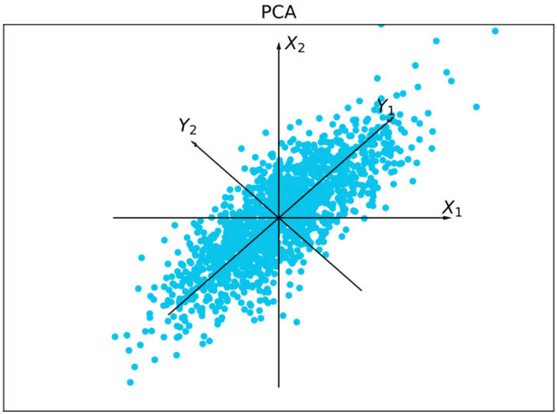
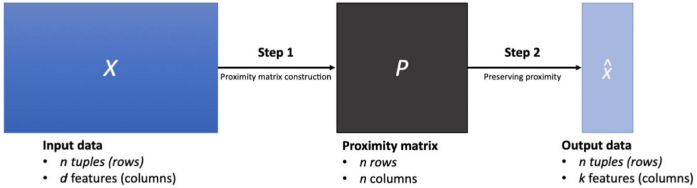


Figure: Principal components analysis. Y_1 and Y_2 are the first two principal components for the given data.

Principal Component Analysis (PCA)



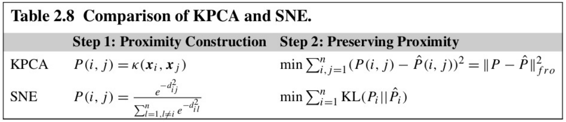


Table 2.8 Comparison of KPCA and SNE.

	Step 1: Proximity Construction	Step 2: Preserving Proximity
KPCA	$P(i,j) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$	$\min \sum_{i,j=1}^{n} (P(i,j) - \hat{P}(i,j))^2 = \lVert P - \hat{P} \rVert_{fro}^2$
SNE	$P(i,j) = \frac{e^{-d_{ij}^2}}{\sum_{l=1, l \neq i}^{n} e^{-d_{il}^2}}$	$\min \sum_{i=1}^{n} \mathrm{KL}(P_i \Vert \hat{P}_i)$

Figure: An illustration of nonlinear dimensionality reduction.

Multivariate Statistics

If x_i is a continuous random variable with continuous values and probability density function $f_i(x_i)$, the mean and variance of the random variable, u_i and σ_i^2, are defined as follows:

$$u_i = E(x_i) = \int_{-\infty}^{\infty} x_i f_i(x_i) \partial x_i$$

$$\sigma_i^2 = \int_{-\infty}^{\infty} (x_i - u_i)^2 f_i(x_i) \partial x_i$$

If x_i is a discrete random variable with discrete values and probability function $P(x_i)$,

$$u_i = E(x_i) = \sum_{\forall x_i} x_i P(x_i)$$

$$\sigma_i^2 = \sum_{\forall x_i} (x_i - u_i)^2 P(x_i)$$

Multivariate Statistics

If x_i and x_j are continuous random variables with the joint probability density function $f_{ij}(x_i, x_j)$, the covariance of two random variables, x_i and x_j, is defined as follows:

$$\sigma_{ij} = E(x_i - \mu_i)(x_j - \mu_j) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x_i - u_i)(x_j - u_j)f_{ij}(x_i, x_j)\partial x_i \partial x_j$$

If x_i and x_j are discrete random variables with the joint probability density function $P(x_i, x_j)$,

$$\sigma_{ij} = E(x_i - \mu_i)(x_j - \mu_j) = \sum_{\forall x_i}\sum_{\forall x_j}(x_i - u_i)(x_j - u_j)P(x_i, x_j)$$

The correlation coefficient is ρ

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_i} \cdot \sqrt{\sigma_j}}$$

Multivariate Statistics

For a vector of random variables, $\mathrm{x} = (x_1, x_2, \cdots, x_p)$, the mean vector is:

$$E(\mathrm{x}) = \begin{bmatrix} E(x_1) \\ E(x_2) \\ \vdots \\ E(x_p) \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix} = \mu$$

and the variance–covariance matrix Σ is

$$\boldsymbol{\Sigma} = E(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})' = E\left(\begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \\ \vdots \\ x_p - \mu_p \end{bmatrix} \begin{bmatrix} x_1 - \mu_1 & x_2 - \mu_2 & \cdots & x_p - \mu_p \end{bmatrix} \right)$$

Multivariate Statistics

$$= E\begin{pmatrix} (x_1-\mu_1)^2 & (x_1-\mu_1)(x_2-\mu_2) & \cdots & (x_1-\mu_1)(x_p-\mu_p) \\ (x_2-\mu_2)(x_1-\mu_1) & (x_2-\mu_2)^2 & \cdots & (x_1-\mu_1)(x_2-\mu_2) \\ \vdots & \vdots & \ddots & \vdots \\ (x_p-\mu_p)(x_1-\mu_1) & (x_p-\mu_p)(x_2-\mu_2) & \cdots & (x_p-\mu_p)^2 \end{pmatrix}$$

$$= \begin{pmatrix} E(x_1-\mu_1)^2 & E(x_1-\mu_1)(x_2-\mu_2) & \cdots & E(x_1-\mu_1)(x_p-\mu_p) \\ E(x_2-\mu_2)(x_1-\mu_1) & E(x_2-\mu_2)^2 & \cdots & E(x_2-\mu_2)(x_p-\mu_p) \\ \vdots & \vdots & \ddots & \vdots \\ E(x_p-\mu_p)(x_1-\mu_1) & E(x_p-\mu_p)(x_2-\mu_2) & \cdots & E(x_p-\mu_p)^2 \end{pmatrix}$$

Multivariate Statistics

$$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ldots \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{bmatrix}$$

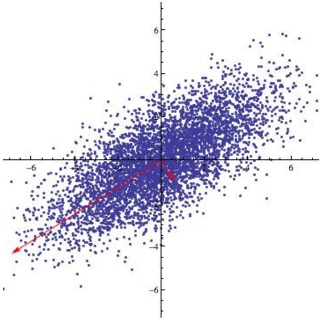


Figure: Covariance Matrix

Principal Component Analysis

Principal component analysis explains the variance–covariance matrix of variables. Given a vector of variables $\boldsymbol{x}' = [x_1, \ldots, x_p]$ with the variance–covariance matrix $\boldsymbol{\Sigma}$, the following is a linear combination of these variables:

$$y_i = \boldsymbol{a}_i'\boldsymbol{x} = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ip}x_p.$$

The variance and covariance of y_i can be computed as follows:

$$\operatorname{var}(y_i) = \boldsymbol{a}_i'\boldsymbol{\Sigma}\boldsymbol{a}_i$$

$$\operatorname{cov}(y_i, y_j) = \boldsymbol{a}_i'\boldsymbol{\Sigma}\boldsymbol{a}_j.$$

The principal components $\boldsymbol{y}' = [y_1, y_2, \ldots, y_p]$ are chosen to be linear combinations of $\boldsymbol{x}'$ that satisfy the following:

Principal Component Analysis

$$y_1 = \boldsymbol{a}_1'\boldsymbol{x} = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1p}x_p,$$

$$\boldsymbol{a}_1'\boldsymbol{a}_1 = 1,\ \boldsymbol{a}_1 \text{ is chosen to maximize } \operatorname{var}(y_1)$$

$$y_2 = \boldsymbol{a}_2'\boldsymbol{x} = a_{21}x_1 + a_{22}x_2 + \cdots + a_{2p}x_p,$$

$$\boldsymbol{a}_2'\boldsymbol{a}_2 = 1, \operatorname{cov}(y_2, y_1) = 0,\ a_2 \text{ is chosen to maximize } \operatorname{var}(y_2)$$

$$\vdots$$

$$y_i = \boldsymbol{a}_i'\boldsymbol{x} = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ip}x_p,$$

$$\boldsymbol{a}_i'\boldsymbol{a}_i = 1, \operatorname{cov}(y_i, y_j) = 0 \quad \text{for } j < i,\ a_i \text{ is chosen to maximize } \operatorname{var}(y_i).$$

Let $(\lambda_i, \boldsymbol{e}_i)$, $i = 1, \ldots, p$, be eigenvalues and orthogonal eigenvectors of $\boldsymbol{\Sigma}$, $\boldsymbol{e}_i'\boldsymbol{e}_i = 1$, and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$. Setting $\boldsymbol{a}_1 = \boldsymbol{e}_1, \ldots, \boldsymbol{a}_p = \boldsymbol{e}_p$, we have

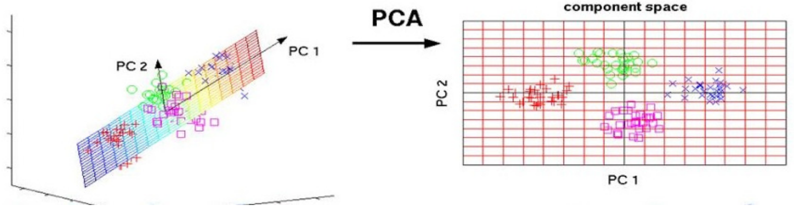
$$y_i = \boldsymbol{e}_i'\boldsymbol{x} \quad i = 1, \ldots, p$$

Principal Component Analysis

$$e_i' e_i = 1$$

$$\mathrm{var}(y_i) = e_i' \boldsymbol{\Sigma} e_i = \lambda_i$$

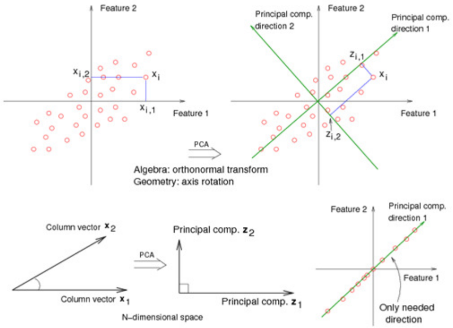
$$\mathrm{cov}(y_i, y_j) = e_i' \boldsymbol{\Sigma} e_j = 0 \quad \text{for } j < i.$$



Principal Component Analysis

Let $x_1, \ldots, x_p$ have variances of $\sigma_1, \ldots, \sigma_p$, respectively. The sum of variances of $x_1, \ldots, x_p$ is equal to the sum of variances of $y_1, \ldots, y_p$

$$\sum_{i=1}^{p} \mathrm{var}(x_i) = \sigma_1 + \cdots + \sigma_p = \sum_{i=1}^{p} \mathrm{var}(y_i) = \lambda_1 + \cdots + \lambda_p.$$



Feature 2
$x_{i,2}$
x_i
$x_{i,1}$
Feature 1
PCA
Algebra: orthonormal transform
Geometry: axis rotation
Principal comp. direction 2
Feature 2
Principal comp. direction 1
$z_{i,1}$
x_i
Feature 1
$z_{i,2}$
Column vector $\mathbf{x}_2$
Column vector $\mathbf{x}_1$
PCA
Principal comp. $\mathbf{z}_2$
Principal comp. $\mathbf{z}_1$
N–dimensional space
Feature 2
Principal comp. direction 1
Feature 1
Only needed direction

PCA: Example

For the two variables $\acute{x} = [x_7, x_8]$ and the variance-covariance matrix Σ is

$$\Sigma = \begin{bmatrix} 0.2469 & -0.1358 \\ -0.1358 & 0.2469 \end{bmatrix}$$

with determined eigenvalues and eigenvectors are
$\lambda_1 = 0.3824, \ \lambda_2 = 0.1115$ and

$$e_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$$

,

$$e_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

The principal components are

PCA: Example

$$y_1 = \boldsymbol{e}_1'\boldsymbol{x} = \frac{1}{\sqrt{2}}x_7 - \frac{1}{\sqrt{2}}x_8$$

$$y_2 = \boldsymbol{e}_2'\boldsymbol{x} = \frac{1}{\sqrt{2}}x_7 + \frac{1}{\sqrt{2}}x_8.$$

Uses of Principal Component Analysis (PCA):

- It is used to find interrelations between variables in the data;
- It is used to interpret and visualize data;
- The number of variables is decreasing which makes further analysis simpler;
- It's often used to visualize genetic distance between populations;

PCA: Example

The variances of y_1 and y_2 are

$$\begin{aligned}
\text{var}(y_1) &= \text{var}\left(\frac{1}{\sqrt{2}}x_7 - \frac{1}{\sqrt{2}}x_8\right) \\
&= \left(\frac{1}{\sqrt{2}}\right)^2 \text{var}(x_7) + \left(\frac{-1}{\sqrt{2}}\right)^2 \text{var}(x_8) + 2\left(\frac{1}{\sqrt{2}}\right)\left(\frac{-1}{\sqrt{2}}\right)\text{cov}(x_7, x_8) \\
&= \frac{1}{2}(0.2469) + \frac{1}{2}(0.2469) - (-0.1358) = 0.3827 = \lambda_1
\end{aligned}$$

$$\begin{aligned}
\text{var}(y_2) &= \text{var}\left(\frac{1}{\sqrt{2}}x_7 + \frac{1}{\sqrt{2}}x_8\right) \\
&= \left(\frac{1}{\sqrt{2}}\right)^2 \text{var}(x_7) + \left(\frac{1}{\sqrt{2}}\right)^2 \text{var}(x_8) + 2\left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right)\text{cov}(x_7, x_8) \\
&= \frac{1}{2}(0.2469) + \frac{1}{2}(0.2469) + (-0.1358) = 0.1111 = \lambda_2.
\end{aligned}$$

We also have

$$\text{var}(x_7) + \text{var}(x_8) = 0.2469 + 0.2469 = \text{var}(y_1) + \text{var}(y_2) = 0.3827 + 0.1111.$$

The proportion of the total variances accounted for by the first principal component y_1 is $0.3824/0.4939 = 0.7742$ or 77%.

Principal Component Analysis (PCA)

Algorithm 1 The PCA algorithm

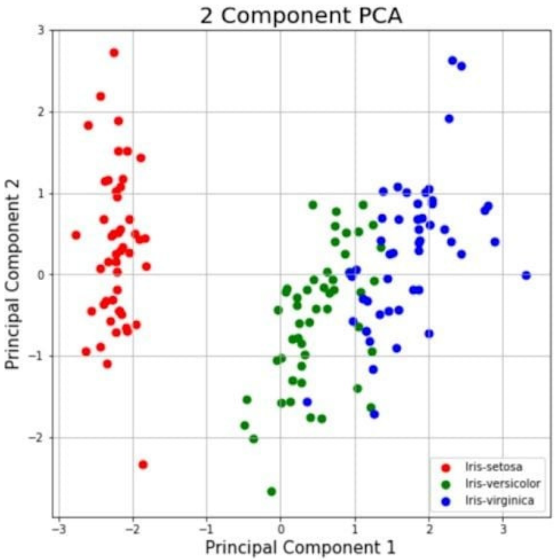
```
1: procedure PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$)    ▷ Return principal components from the given dataset.
2:     Center the data: $\mathbf{x}_t = \mathbf{x}_t - \mu$ with $\mu$ the mean vector.
3:     Construct the covariance matrix $\mathbf{C} = \frac{1}{n} \sum_t \mathbf{x}_t \mathbf{x}_t^T$.
4:     Eigen-decompose $\mathbf{C}$ and let $\lambda_1 \geq ... \geq \lambda_d$ and $\mathbf{v}_1, ..., \mathbf{v}_d$ be its eigenvalues and eigenvectors.
5:     Select $m$ as discussed above.
6:     return first $m$ eigenvectors and eigenvalues.
7: end procedure
```

Principal Component Analysis (PCA)

The following Python code:

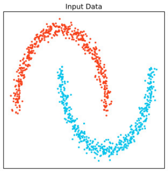
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

df = pd.read_csv(url, names=['sepal_length','sepal_width','petal_length',
'petal_width','target'])
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
# Separating out the features
x = df.loc[:, features].values
# Separating out the target
y = df.loc[:,['target']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
pca = PCA(n_components=2),pCs = pca.fit_transform(x)
principalDf = pd.DataFrame(data=pCs, columns = ['pc1', 'pc2'])
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)

fig = plt.figure(figsize = (8,8)), ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize=15)
ax.set_ylabel('Principal Component 2', fontsize=15)
ax.set_title('2 component PCA', fontsize = 20)

targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
    ,finalDf.loc[indicesToKeep, 'principal component 2'] ,c=color, s=50)
ax.legend(targets),ax.grid()
```

Principal Component Analysis (PCA)

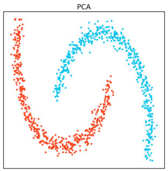


Kernel Principal Component Analysis (kPCA)

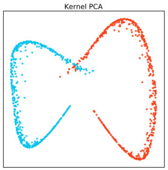
- In **kernel PCA (KPCA)**, a kernel function $K(\cdot)$ uses to construct the proximity matrix called kernel matrix: $P(i,j) = K(x_i, x_j)$, $(i, j = 1, \cdots, n)$;
- A **kernel** function computes the similarity of a pair of input data tuples in nonlinear high-dimensional space;



(a) Input data



(b) PCA



(c) KPCA

Kernel Principal Component Analysis (kPCA)

Typical choices for the kernel functions as follows:

Polynomial kernel function (PKF):

$$k(x_i, x_j) = (1 + xi \cdot x_j)^p$$

where p is the parameter and if $p = 1$ then it is a liner kernel i.e.

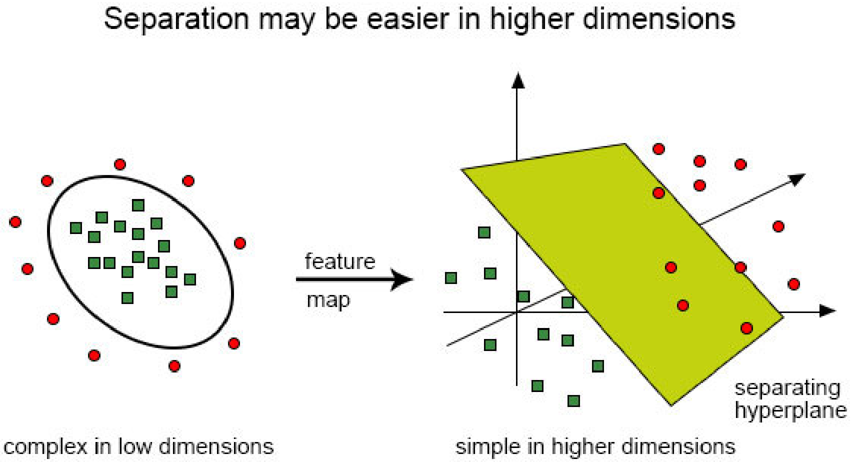
$$k(x_i, x_j) = (1 + xi \cdot x_j)$$

Radial basis function (RBF):

$$k(x_i, x_j) = e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}}$$

where σ is the parameter.

Kernel Principal Component Analysis (kPCA)

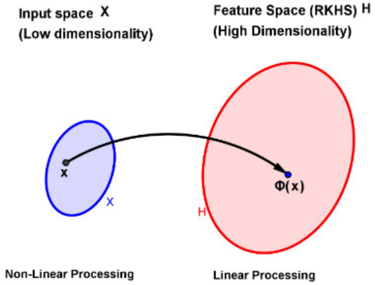

Separation may be easier in higher dimensions
feature
map
separating
hyperplane
complex in low dimensions
simple in higher dimensions

Kernel Principal Component Analysis (kPCA)

In **Hilbert space**, if K is a real-valued kernel function, then there exists a mapping function Φ such that

$$K(\mathrm{x}, \mathrm{y}) = \langle \Phi(\mathrm{x}), \Phi(\mathrm{y}) \rangle$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product and the mapping Φ is called the **kernel Hilbert space** or simple **kernel space**.

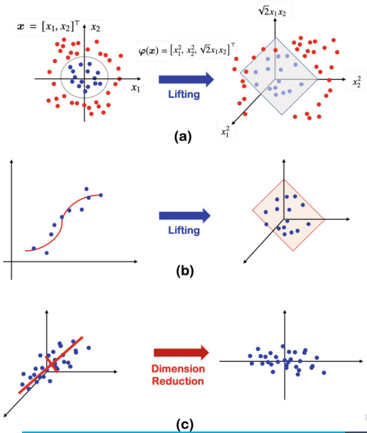


Kernel Principal Component Analysis (kPCA)

In **kernel space** the **Euclidean** distance between $\mathbf{x}, \mathbf{y}$ is given by:

$$\|\Phi(\mathrm{x}), \Phi(\mathrm{y})\|^2 = K(\mathrm{x}, \mathrm{x}) + K(\mathrm{y}, \mathrm{y}) - 2K(\mathrm{x}, \mathrm{y})$$

This calculation allows transforming the linear model into a non-linear model at the kernel space.


$\boldsymbol{x} = [x_1, x_2]^\top$
x_2
x_1
$\varphi(\boldsymbol{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2]^\top$
Lifting
$\sqrt{2}x_1x_2$
x_2^2
x_1^2
(a)
Lifting
(b)
Dimension Reduction
(c)

Kernel Principal Component Analysis (kPCA)

In the kernel space, the mean vector μ can be defined implicitly via Φ as follows:

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\Phi(x_i)$$

In kernel space, compute the dot product between two data points $\mathbf{x}, \mathbf{y}$ after mean μ subtraction as follows:

$$\langle\Phi(x)-\mu, \Phi(y)-\mu\rangle = \langle\Phi(x), \Phi(y)\rangle - \langle\Phi(x), \Phi(\mu)\rangle - \langle\Phi(y), \Phi(\mu)\rangle + \langle\Phi(\mu), \Phi(mu)\rangle$$

$$\begin{aligned}
&=\langle\Phi(\mathbf{x}),\Phi(\mathbf{y})\rangle - \frac{1}{n}\sum_{t=1}^{n}\langle\Phi(\mathbf{x}),\Phi(\mathbf{x}_t)\rangle - \frac{1}{n}\sum_{t=1}^{n}\langle\Phi(\mathbf{y}),\Phi(\mathbf{x}_t)\rangle \\
&\quad + \frac{1}{n^2}\sum_{t=1}^{n}\sum_{t'=1}^{n}\langle\Phi(\mathbf{x}_t),\Phi(\mathbf{x}_{t'})\rangle \\
&=\kappa(\mathbf{x},\mathbf{y}) - \frac{1}{n}\sum_{t=1}^{n}\kappa(\mathbf{x},\mathbf{x}_t) - \frac{1}{n}\sum_{t=1}^{n}\kappa(\mathbf{y},\mathbf{x}_t)\rangle + \frac{1}{n^2}\sum_{t=1}^{n}\sum_{t'=1}^{n}\kappa(\mathbf{x}_t,\mathbf{x}_{t'})\rangle \\
&\equiv\hat{\kappa}(\mathbf{x},\mathbf{y}).
\end{aligned}$$

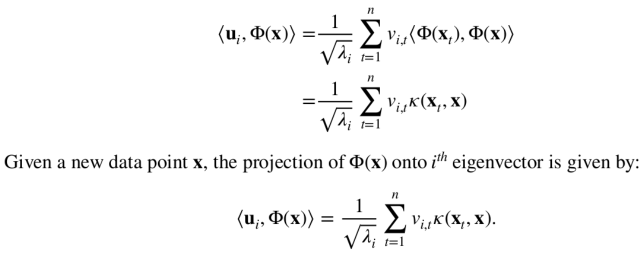
$$K_{i,j} = k(x_i, y_j), 0 \leq i, j \leq n$$

Kernel Principal Component Analysis (kPCA)

An eigenvector $\mathbf{u}_i$ of the covariance matrix in kernel space must be of the form:

$$\mathbf{u}_i = \frac{1}{\sqrt{\lambda_i}} \sum_{t=1}^{n} v_{i,t} \Phi(\mathbf{x}_t)$$

where $\mathbf{v}_i = [v_{i,1}, \cdots, v_{i,n}]^T$ is the i^{th} eigenvector of the kernel matrix and λ_i its corresponding eigenvalue and the projection of $\Phi(x)$ onto $\mathbf{u}_i$ as follows:



$$\begin{aligned}\langle \mathbf{u}_i, \Phi(\mathbf{x})\rangle &= \frac{1}{\sqrt{\lambda_i}} \sum_{t=1}^{n} v_{i,t} \langle \Phi(\mathbf{x}_t), \Phi(\mathbf{x})\rangle \\ &= \frac{1}{\sqrt{\lambda_i}} \sum_{t=1}^{n} v_{i,t} \kappa(\mathbf{x}_t, \mathbf{x})\end{aligned}$$

Given a new data point $\mathbf{x}$, the projection of $\Phi(\mathbf{x})$ onto i^{th} eigenvector is given by:

$$\langle \mathbf{u}_i, \Phi(\mathbf{x})\rangle = \frac{1}{\sqrt{\lambda_i}} \sum_{t=1}^{n} v_{i,t} \kappa(\mathbf{x}_t, \mathbf{x}).$$

Reproducing kernel map (RK map), φ

- Feature map maps points to features (vectors in feature space), i.e. $\varphi: \mathcal{X} \to (V, \langle\cdot,\cdot\rangle)$
- **Def**: reproducing kernel map is a feature map, φ, that maps points to kernels (as vectors in RKHS, a Hilbert space of functions defined by fixed kernel k):

$$\varphi: \mathcal{X} \to \mathcal{H}_k \triangleq \{f: \mathcal{X} \to \mathbb{R}\} \quad by \quad x \mapsto k(\cdot, x)$$

  - i.e. $\varphi(x)(\cdot) = k(\cdot, x)$
- Since $\varphi(x)$ is a function, we can evaluate φ once more:

$$\varphi(x)(y) = \big(k(\cdot, x)\big)(y) = k(x, y)$$
$$= \langle \varphi(x), \varphi(y) \rangle = \langle k(\cdot, x), k(\cdot, y) \rangle$$

- Reproducing property: $k(x, y) = \langle k(\cdot, x), k(\cdot, y) \rangle$
- Correspondence: $\varphi \leftrightarrow k \leftrightarrow \mathcal{H}_k$

Reproducing kernel map (RK map), φ

- Feature map maps points to features (vectors in feature space), i.e. $\varphi: \mathcal{X} \to (V, \langle\cdot,\cdot\rangle)$
- **Def**: reproducing kernel map is a feature map, φ, that maps points to kernels (as vectors in RKHS, a Hilbert space of functions defined by fixed kernel k):

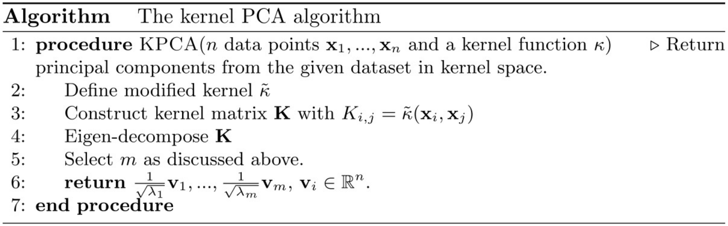
$$\varphi: \mathcal{X} \to \mathcal{H}_k \triangleq \{f: \mathcal{X} \to \mathbb{R}\} \quad by \quad x \mapsto k(\cdot, x)$$

  - i.e. $\varphi(x)(\cdot) = k(\cdot, x)$
- Since $\varphi(x)$ is a function, we can evaluate φ once more:

$$\varphi(x)(y) = \big(k(\cdot, x)\big)(y) = k(x, y)$$
$$= \langle \varphi(x), \varphi(y) \rangle = \langle k(\cdot, x), k(\cdot, y) \rangle$$

- Reproducing property: $k(x, y) = \langle k(\cdot, x), k(\cdot, y) \rangle$
- Correspondence: $\varphi \leftrightarrow k \leftrightarrow \mathcal{H}_k$

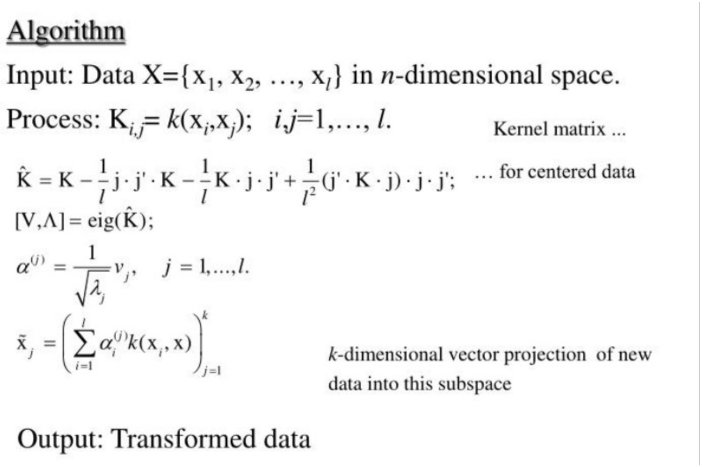
Kernel Principal Component Analysis (kPCA)



```
Algorithm  The kernel PCA algorithm
1: procedure KPCA($n$ data points $\mathbf{x}_1, ..., \mathbf{x}_n$ and a kernel function $\kappa$)    ▷ Return
   principal components from the given dataset in kernel space.
2:     Define modified kernel $\tilde{\kappa}$
3:     Construct kernel matrix $\mathbf{K}$ with $K_{i,j} = \tilde{\kappa}(\mathbf{x}_i, \mathbf{x}_j)$
4:     Eigen-decompose $\mathbf{K}$
5:     Select $m$ as discussed above.
6:     return $\frac{1}{\sqrt{\lambda_1}}\mathbf{v}_1, ..., \frac{1}{\sqrt{\lambda_m}}\mathbf{v}_m,\ \mathbf{v}_i \in \mathbb{R}^n$.
7: end procedure
```

Figure: kPCA Algorithm

Kernel Principal Component Analysis (kPCA)



Algorithm

Input: Data X={x_1, x_2, …, x_l} in *n*-dimensional space.

Process: $K_{i,j} = k(x_i, x_j)$; $i,j = 1, \ldots, l$. Kernel matrix ...

$$\hat{K} = K - \frac{1}{l} j \cdot j' \cdot K - \frac{1}{l} K \cdot j \cdot j' + \frac{1}{l^2}(j' \cdot K \cdot j) \cdot j \cdot j';$$ … for centered data

$$[V, \Lambda] = \mathrm{eig}(\hat{K});$$

$$\alpha^{(j)} = \frac{1}{\sqrt{\lambda_j}} v_j, \quad j = 1, \ldots, l.$$

$$\tilde{x}_j = \left(\sum_{i=1}^{l} \alpha_i^{(j)} k(x_i, x) \right)_{j=1}^{k}$$

k-dimensional vector projection of new data into this subspace

Output: Transformed data

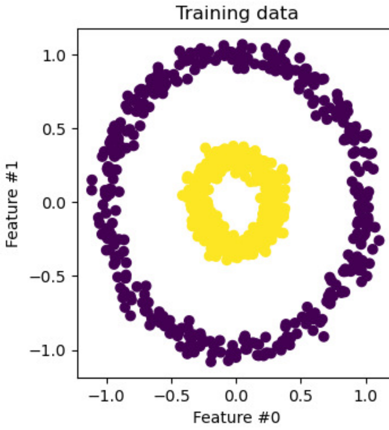
Figure: kPCA Algorithm

Kernel Principal Component Analysis (kPCA)

The following Python code:

```
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA, KernelPCA

X, y = make_circles(n_samples=1_000, factor=0.3, noise=0.05, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=0)
import matplotlib.pyplot as plt

_, (train_ax, test_ax) = plt.subplots(ncols=2, sharex=True, sharey=True, figsize=(8, 4))
train_ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
train_ax.set_ylabel("Feature␣#1")
train_ax.set_xlabel("Feature␣#0")
train_ax.set_title("Training␣data")
test_ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
test_ax.set_xlabel("Feature␣#0")
_ = test_ax.set_title("Testing␣data")
```

Kernel Principal Component Analysis (kPCA)



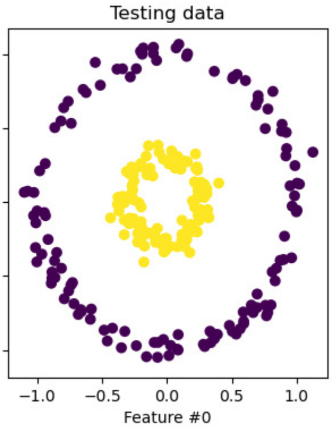


Figure: kPCA Algorithm

Kernel Principal Component Analysis (kPCA)

The following Python code:

```
pca = PCA(n_components=2)
kernel_pca = KernelPCA(n_components=None, kernel="rbf", gamma=10,
fit_inverse_transform=True, alpha=0.1)

X_test_pca = pca.fit(X_train).transform(X_test)
X_test_kernel_pca = kernel_pca.fit(X_train).transform(X_test)
fig, (orig_data_ax, pca_proj_ax, kernel_pca_proj_ax) = plt.subplots(ncols=3,figsize=(14, 4)
orig_data_ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
orig_data_ax.set_ylabel("Feature #1")
orig_data_ax.set_xlabel("Feature #0")
orig_data_ax.set_title("Testing data")

pca_proj_ax.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test)
pca_proj_ax.set_ylabel("Principal component #1")
pca_proj_ax.set_xlabel("Principal component #0")
pca_proj_ax.set_title("Projection of testing data using PCA")

kernel_pca_proj_ax.scatter(X_test_kernel_pca[:, 0], X_test_kernel_pca[:, 1], c=y_test)
kernel_pca_proj_ax.set_ylabel("Principal component #1")
kernel_pca_proj_ax.set_xlabel("Principal component #0")
_ = kernel_pca_proj_ax.set_title("Projection of testing data using KernelPCA")
```

Kernel Principal Component Analysis (kPCA)

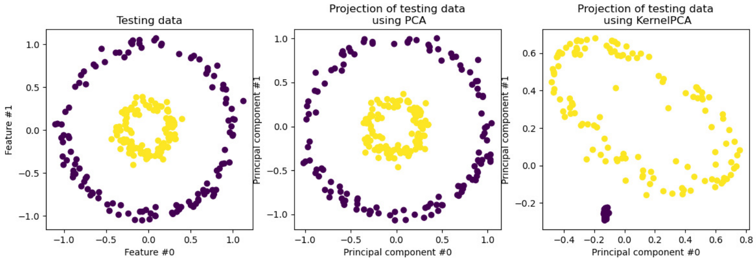


Figure: kPCA Algorithm

Kernel Principal Component Analysis (kPCA)

The following Python code:

```
X_reconstructed_pca = pca.inverse_transform(pca.transform(X_test))
X_reconstructed_kernel_pca = kernel_pca.inverse_transform(kernel_pca.transform(X_test))

fig, (orig_data_ax, pca_back_proj_ax, kernel_pca_back_proj_ax) =
plt.subplots(ncols=3, sharex=True, sharey=True, figsize=(13, 4))

orig_data_ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
orig_data_ax.set_ylabel("Feature #1")
orig_data_ax.set_xlabel("Feature #0")
orig_data_ax.set_title("Original test data")

pca_back_proj_ax.scatter(X_reconstructed_pca[:, 0], X_reconstructed_pca[:, 1], c=y_test)
pca_back_proj_ax.set_xlabel("Feature #0")
pca_back_proj_ax.set_title("Reconstruction via PCA")

kernel_pca_back_proj_ax.scatter(
    X_reconstructed_kernel_pca[:, 0], X_reconstructed_kernel_pca[:, 1], c=y_test
)
kernel_pca_back_proj_ax.set_xlabel("Feature #0")
_ = kernel_pca_back_proj_ax.set_title("Reconstruction via KernelPCA")
```

Kernel Principal Component Analysis (kPCA)

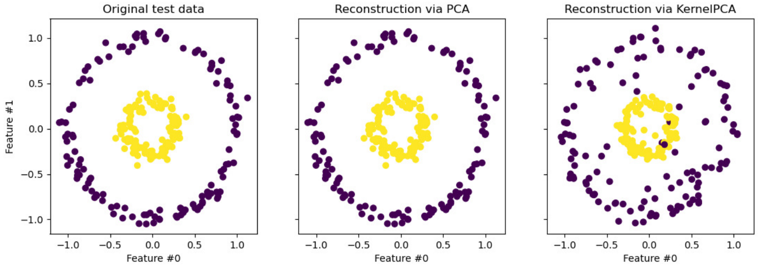


Figure: kPCA Algorithm

Applications of kPCA:

Some notable applications of **kPCA** include:

1. **Image Recognition:** kPCA can effectively capture the nonlinear patterns in image data;
2. **Natural language processing (NLP):** can be applied to analyze and reduce the dimensionality of textual data for tasks such as text classification, sentiment analysis, and document clustering
3. **Genomics and bio-informatics:** in genomics, kPCA can help analysis gene expression data, DNA sequencing data, and protein structure data;
4. **Finance-kernel PCA:** is used in financial modeling to capture complex, nonlinear relationships in stock markets and financial data.

PCA vs. kPCA

Key Differences Summarized:

Feature	PCA	kPCA
Linearity:	Linear method	Non-linear method
Data Mapping:	No mapping	Maps to a HD space using a kernel
Applicability:	Suitable for linear data	Suitable for non-linear data
Complexity:	Simpler	More complex

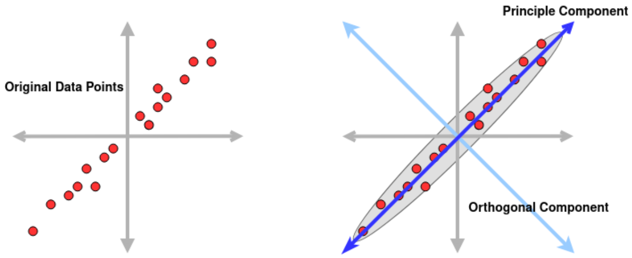


Figure: PCA vs. kPCA

Kernel Principal Component Analysis (kPCA)

Advantages of kPCA:

1. **Non-linearity:** kPCA easily can capture non-linear patterns in the data;
2. **Robustness:** kPCA can be more robust to outliers and noise in the data;
3. **Versatility:** Different types of kernel functions can be used in kPCA to suit different types of data;
4. kPCA can handle nonlinear relationships between the input features and allowing for more accurate dimensionality reduction;
5. kPCA can preserve the most important information in high-dimensional datasets and making it easier to visualize and analyze.
6. kPCA can be used for a variety of tasks, including data visualization, clustering, and classification.

Kernel Principal Component Analysis (kPCA)

Disadvantages of kPCA:

1. **Complexity:** kPCA can be computationally expensive;
2. **Model selection:** Choosing the right kernel function and the right number of components can be challenging;
3. Choosing an appropriate kernel function and its parameters can be challenging;
4. kPCA can be computationally expensive, especially for large datasets;
5. kPCA requires the massive computation of the kernel matrix for all pairs of data points;
6. kPCA is not suitable for datasets with many missing values or outliers;

Thank You