

# Machine/Statistical Learning

## PCA and Kernel PCA

Nicolas Keriven  
CNRS, IRISA, Rennes

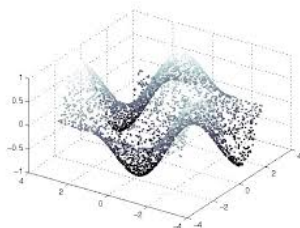
ENSTA 2024

## Dimension Reduction : motivation

- ▶ **Compression** : reduction of the size of the data
- ▶ **Interpretability** : which dimensions are **important/linked** for prediction, **data** visualization

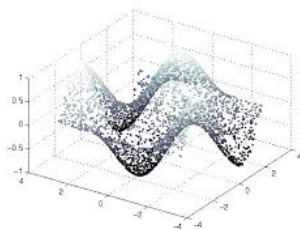
## Dimension Reduction : motivation

- ▶ **Compression** : reduction of the size of the data
- ▶ **Interpretability** : which dimensions are **important/linked** for prediction, **data visualization**
- ▶ **Curse of dimensionality** : the amount of data to “fill” the space is **exponential in the dimension**. High dimensional spaces are **mostly empty** !
- ▶ **Intrinsic dim.**  $\ll$  representation dim. : high-dim data often live in an (unknown) **lower dimensional space**



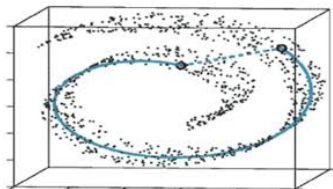
## Dimension Reduction : motivation

- ▶ **Compression** : reduction of the size of the data
- ▶ **Interpretability** : which dimensions are **important/linked** for prediction, **data visualization**
- ▶ **Curse of dimensionality** : the amount of data to “fill” the space is **exponential in the dimension**. High dimensional spaces are **mostly empty** !
- ▶ **Intrinsic dim.**  $\ll$  representation dim. : high-dim data often live in an (unknown) **lower dimensional space**
- ▶ dimension reduction  $\rightarrow$  **noise reduction** : **noise** "visits" all possible dimensions



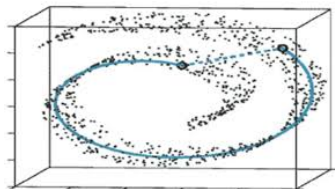
## Extraction techniques (manifold learning)

- ▶ Physically based methods, k-NN graphs, geodesic distances...
- ▶ Statistical methods, dictionary learning
- ▶ Filtering (linear/non linear)
- ▶ Deep autoencoders...



## Extraction techniques (manifold learning)

- ▶ Physically based methods, k-NN graphs, geodesic distances...
- ▶ Statistical methods, dictionary learning
- ▶ Filtering (linear/non linear)
- ▶ Deep autoencoders...



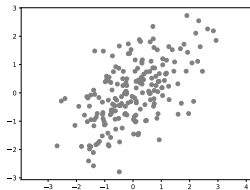
🔍 Finding/learning **data representation** ("features extraction") is somehow the core of modern ML. **PCA** is a foundational idea.

## Classical (non probabilistic) PCA

- ▶ PCA is a **linear** dimension reduction method

## Classical (non probabilistic) PCA

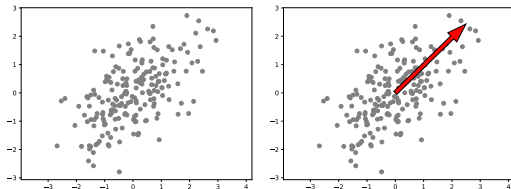
- ▶ PCA is a **linear** dimension reduction method
- ▶ Idea : find “**directions**” in the data such that :





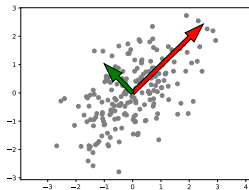
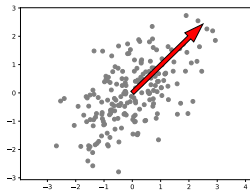
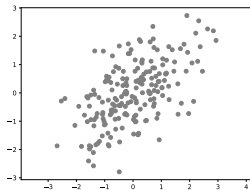
## Classical (non probabilistic) PCA

- ▶ PCA is a **linear** dimension reduction method
- ▶ Idea : find “**directions**” in the data such that :
  - ▶ the first explains **most the variation of the data**



## Classical (non probabilistic) PCA

- ▶ PCA is a **linear** dimension reduction method
- ▶ Idea : find “**directions**” in the data such that :
  - ▶ the first explains **most the variation of the data**
  - ▶ the second most of the **remaining** variation (while being orthogonal to the first)
  - ▶ etc. (here  $d = 2$  but usually  $d$  very high !)

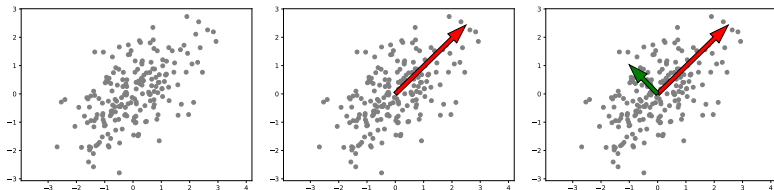


## Classical (non probabilistic) PCA

- ▶ PCA is a **linear** dimension reduction method
- ▶ Idea : find “**directions**” in the data such that :
  - ▶ the first explains **most the variation of the data**
  - ▶ the second most of the **remaining** variation (while being orthogonal to the first)
  - ▶ etc. (here  $d = 2$  but usually  $d$  very high !)

Mathematically :

- ▶ Find  $p$  orthonormal vectors  $v_1, \dots, v_p$  where  $p \ll d$  such that :

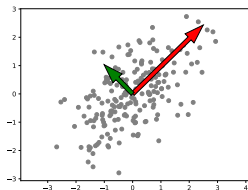
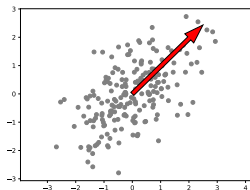
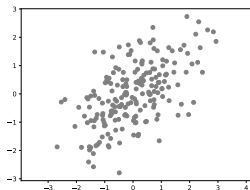


## Classical (non probabilistic) PCA

- ▶ PCA is a **linear** dimension reduction method
- ▶ Idea : find “**directions**” in the data such that :
  - ▶ the first explains **most the variation of the data**
  - ▶ the second most of the **remaining** variation (while being orthogonal to the first)
  - ▶ etc. (here  $d = 2$  but usually  $d$  very high !)

Mathematically :

- ▶ Find  $p$  orthonormal vectors  $v_1, \dots, v_p$  where  $p \ll d$  such that :
  - ▶  $v_1 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1$

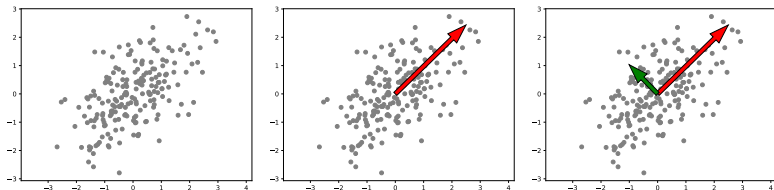


## Classical (non probabilistic) PCA

- ▶ PCA is a **linear** dimension reduction method
- ▶ Idea : find “**directions**” in the data such that :
  - ▶ the first explains **most the variation of the data**
  - ▶ the second most of the **remaining** variation (while being orthogonal to the first)
  - ▶ etc. (here  $d = 2$  but usually  $d$  very high !)

Mathematically :

- ▶ Find  $p$  orthonormal vectors  $v_1, \dots, v_p$  where  $p \ll d$  such that :
  - ▶  $v_1 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1$
  - ▶  $v_2 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0$

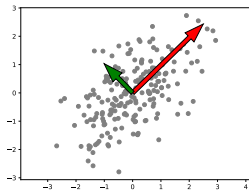
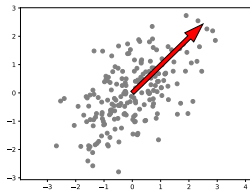
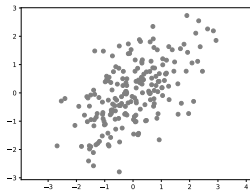


## Classical (non probabilistic) PCA

- ▶ PCA is a **linear** dimension reduction method
- ▶ Idea : find “**directions**” in the data such that :
  - ▶ the first explains **most the variation of the data**
  - ▶ the second most of the **remaining** variation (while being orthogonal to the first)
  - ▶ etc. (here  $d = 2$  but usually  $d$  very high !)

Mathematically :

- ▶ Find  $p$  orthonormal vectors  $v_1, \dots, v_p$  where  $p \ll d$  such that :
  - ▶  $v_1 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1$
  - ▶  $v_2 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0$
  - ▶  $v_3 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0, \langle v, v_2 \rangle = 0$ , etc



## Computing PCA

- Find  $v_1 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1$  : we have

$$\text{var}(\langle v, x \rangle) = \mathbb{E}(\langle v, x - \mathbb{E}(x) \rangle)^2 = v^\top \mathbb{E}[(x - \mathbb{E}(x))(x - \mathbb{E}(x))^\top] v = v^\top \Sigma v$$

where  $\Sigma = \text{Cov}(x)$ .

## Computing PCA

- Find  $v_1 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1$  : we have

$$\text{var}(\langle v, x \rangle) = \mathbb{E}(\langle v, x - \mathbb{E}(x) \rangle)^2 = v^\top \mathbb{E}[(x - \mathbb{E}(x))(x - \mathbb{E}(x))^\top] v = v^\top \Sigma v$$

where  $\Sigma = \text{Cov}(x)$ .

- Form the Lagrangian (Recall : constrained optimization)

$$\mathcal{L}(v, \lambda) = v^\top \Sigma v + \lambda(1 - v^\top v)$$



## Computing PCA

- Find  $v_1 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1$  : we have

$$\text{var}(\langle v, x \rangle) = \mathbb{E}(\langle v, x - \mathbb{E}(x) \rangle)^2 = v^\top \mathbb{E}[(x - \mathbb{E}(x))(x - \mathbb{E}(x))^\top] v = v^\top \Sigma v$$

where  $\Sigma = \text{Cov}(x)$ .

- Form the Lagrangian (Recall : constrained optimization)

$$\mathcal{L}(v, \lambda) = v^\top \Sigma v + \lambda(1 - v^\top v)$$

- Compute the gradient wrt  $v$

$$\nabla_v \mathcal{L} = 2\Sigma v - 2\lambda v$$

- $v$  is an eigenvector of  $\Sigma$ ,  $\lambda$  is an eigenvalue :

$$[\nabla \mathcal{L} = 0] \Leftrightarrow [\Sigma v = \lambda v]$$

- The variance is  $v^\top \Sigma v = \lambda$ .

## Computing PCA

- Find  $v_1 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1$  : we have

$$\text{var}(\langle v, x \rangle) = \mathbb{E}(\langle v, x - \mathbb{E}(x) \rangle)^2 = v^\top \mathbb{E}[(x - \mathbb{E}(x))(x - \mathbb{E}(x))^\top] v = v^\top \Sigma v$$

where  $\Sigma = \text{Cov}(x)$ .

- Form the Lagrangian (Recall : constrained optimization)

$$\mathcal{L}(v, \lambda) = v^\top \Sigma v + \lambda(1 - v^\top v)$$

- Compute the gradient wrt  $v$

$$\nabla_v \mathcal{L} = 2\Sigma v - 2\lambda v$$

- $v$  is an eigenvector of  $\Sigma$ ,  $\lambda$  is an eigenvalue :

$$[\nabla \mathcal{L} = 0] \Leftrightarrow [\Sigma v = \lambda v]$$

- The variance is  $v^\top \Sigma v = \lambda$ .

🔑  $v_1$  is the eigenvector that corresponds to the **maximum** eigenvalue of the covariance of the data

Let's do  $v_2$  !

- Find  $v_2 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0$

Let's do  $v_2$  !

- ▶ Find  $v_2 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0$
- ▶ Form the Lagrangian

$$\mathcal{L}(v, \lambda, \beta) = v^\top \Sigma v + \lambda(1 - v^\top v) + \beta(0 - v_1^\top v)$$

- ▶ Compute the gradient

$$\nabla \mathcal{L} = 2\Sigma v - 2\lambda v - \beta v_1 = 0$$

Let's do  $v_2$  !

- ▶ Find  $v_2 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0$
- ▶ Form the Lagrangian

$$\mathcal{L}(v, \lambda, \beta) = v^T \Sigma v + \lambda(1 - v^T v) + \beta(0 - v_1^T v)$$

- ▶ Compute the gradient

$$\nabla \mathcal{L} = 2\Sigma v - 2\lambda v - \beta v_1 = 0$$

- ▶ Multiplying by  $v_1^T$ , and using  $v_1^T v = 0, \|v_1\| = 1, \Sigma v_1 = \lambda_1 v_1 : \beta = 0$
- ▶ Finally,  $\Sigma v = \lambda v : (\lambda, v)$  is again a pair eigval./eigvec., and thus  $\lambda = \lambda_2$

Let's do  $v_2$  !

- ▶ Find  $v_2 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0$
- ▶ Form the Lagrangian

$$\mathcal{L}(v, \lambda, \beta) = v^T \Sigma v + \lambda(1 - v^T v) + \beta(0 - v_1^T v)$$

- ▶ Compute the gradient

$$\nabla \mathcal{L} = 2\Sigma v - 2\lambda v - \beta v_1 = 0$$

- ▶ Multiplying by  $v_1^T$ , and using  $v_1^T v = 0, \|v_1\| = 1, \Sigma v_1 = \lambda_1 v_1 : \beta = 0$
- ▶ Finally,  $\Sigma v = \lambda v : (\lambda, v)$  is again a pair eigval./eigvec., and thus  $\lambda = \lambda_2$
- 👉  $v_2$  is the eigenvector that corresponds to the **second maximum** eigenvalue of the covariance of the data

Let's do  $v_2$  !

- ▶ Find  $v_2 = \arg \max_v [\text{var}(\langle v, x \rangle)]$  s.t.  $\|v\| = 1, \langle v, v_1 \rangle = 0$
- ▶ Form the Lagrangian

$$\mathcal{L}(v, \lambda, \beta) = v^T \Sigma v + \lambda(1 - v^T v) + \beta(0 - v_1^T v)$$

- ▶ Compute the gradient

$$\nabla \mathcal{L} = 2\Sigma v - 2\lambda v - \beta v_1 = 0$$

- ▶ Multiplying by  $v_1^T$ , and using  $v_1^T v = 0, \|v_1\| = 1, \Sigma v_1 = \lambda_1 v_1 : \beta = 0$
- ▶ Finally,  $\Sigma v = \lambda v : (\lambda, v)$  is again a pair eigval./eigvec., and thus  $\lambda = \lambda_2$

☞  $v_2$  is the eigenvector that corresponds to the **second maximum** eigenvalue of the covariance of the data

☞ Same for  $v_3, \dots, v_p$

## Computing PCA in practice

- ▶ Of course, for real data, we use the **empirical** covariance  $\hat{\Sigma}$



## Computing PCA in practice

- Of course, for real data, we use the **empirical** covariance  $\hat{\Sigma}$

### Algorithm : PCA

1. Compute empirical mean :  $\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i$  (cost  $O(nd)$ )
2. Compute empirical cov. matrix :  $\hat{\Sigma} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \hat{\mu})(x_i - \hat{\mu})^\top$  (cost  $O(nd^2)$ )
3. Compute the eigenvectors of the  $p$  largest eigenvalues of  $\hat{\Sigma}$  (cost  $O(d^3)$ )

## Computing PCA in practice

- Of course, for real data, we use the **empirical** covariance  $\hat{\Sigma}$

### Algorithm : PCA

1. Compute empirical mean :  $\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i$  (cost  $O(nd)$ )
2. Compute empirical cov. matrix :  $\hat{\Sigma} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \hat{\mu})(x_i - \hat{\mu})^\top$  (cost  $O(nd^2)$ )
3. Compute the eigenvectors of the  $p$  largest eigenvalues of  $\hat{\Sigma}$  (cost  $O(d^3)$ )

### Remark : SVD

Taking  $X_c$  the matrix of centered data  $x_i - \hat{\mu}$ , such that  $\hat{\Sigma} = X_c^\top X_c / (n-1)$ , and the **Singular Value Decomposition**

$$X_c = U \Gamma V^\top \quad (\text{cost } O(nd^2 + d^3))$$

we have directly  $\hat{\Sigma} = V \Gamma^2 V^\top / (n-1)$ . SVD is often “more convenient” (more algorithms available, more stable, etc.)

## PCA : remarks

## Standardization matter

- ▶ Often center and “whiten” data to unit norm column  $\tilde{x}_i = \text{diag}(1/\sigma_k)(x_i - \hat{\mu})$
- ▶ The low-dimensional representation of  $x$  is  $z = [\tilde{x}^\top v_k]_k \in \mathbb{R}^p$ . **Don't forget the same pre-processing !! Scikit-learn does it automatically.**

## PCA : remarks

## Standardization matter

- ▶ Often center and “whiten” data to unit norm column  $\tilde{x}_i = \text{diag}(1/\sigma_k)(x_i - \hat{\mu})$
- ▶ The low-dimensional representation of  $x$  is  $z = [\tilde{x}^\top v_k]_k \in \mathbb{R}^p$ . **Don't forget the same pre-processing !! Scikit-learn does it automatically.**

## Computational cost

SVD is **costly** for high  $n$  or  $d$ , but **we need only the  $p$  first eigenvectors** ! There are strategies : power iterations/Lanczos, randomized SVD...

## PCA : remarks

### Standardization matter

- ▶ Often center and “whiten” data to unit norm column  $\tilde{x}_i = \text{diag}(1/\sigma_k)(x_i - \hat{\mu})$
- ▶ The low-dimensional representation of  $x$  is  $z = [\tilde{x}^\top v_k]_k \in \mathbb{R}^p$ . **Don't forget the same pre-processing !! Scikit-learn does it automatically.**

### Computational cost

SVD is **costly** for high  $n$  or  $d$ , but **we need only the  $p$  first eigenvectors** ! There are strategies : power iterations/Lanczos, randomized SVD...

### Model selection

Popular criterion : increase  $p$  until the "explained variance"

$$\frac{\sum_{k=1}^p \lambda_k}{\sum_{k=1}^d \lambda_k}$$

is close enough to 1. But computing full SVD may be too costly ! Myriad of other strategies.

## PCA for high dimensional data ( $n < d$ )

- ▶ When  $d$  is high, we want to avoid the cost  $O(d^3)$ .

PCA for high dimensional data ( $n < d$ )

- ▶ When  $d$  is high, we want to avoid the cost  $O(d^3)$ .
- ▶ Remember : the eigenvalue equation

$$\Sigma v = \frac{1}{n-1} X_c^\top X_c v = \lambda v$$

PCA for high dimensional data ( $n < d$ )

- ▶ When  $d$  is high, we want to avoid the cost  $O(d^3)$ .
- ▶ Remember : the eigenvalue equation

$$\Sigma v = \frac{1}{n-1} X_c^\top X_c v = \lambda v$$

This can be written as

$$\frac{1}{n-1} X_c X_c^\top X_c v = \lambda X_c v \Leftrightarrow S u = \lambda u$$

with  $u = X_c v$ , i.e. an eigenvalue problem on  $S = \frac{1}{n-1} X_c X_c^\top$  instead.



## PCA for high dimensional data ( $n < d$ )

- ▶ When  $d$  is high, we want to avoid the cost  $O(d^3)$ .
- ▶ Remember : the eigenvalue equation

$$\Sigma v = \frac{1}{n-1} X_c^\top X_c v = \lambda v$$

This can be written as

$$\frac{1}{n-1} X_c X_c^\top X_c v = \lambda X_c v \Leftrightarrow S u = \lambda u$$

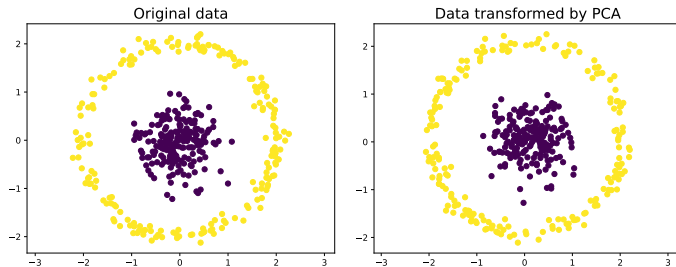
with  $u = X_c v$ , i.e. an eigenvalue problem on  $S = \frac{1}{n-1} X_c X_c^\top$  instead.

- ▶ Properties :
  - ▶ the matrix  $\Sigma$  is rank  $n < d$ , it has only  $n$  non-zero eigenvalues.
  - ▶ Same non-zero eigenvalues as full-rank  $S$ .
  - ▶ Given  $v_i$ ,  $X_c v_i$  is an eigenvector of  $S$ , and given  $u_i$ ,  $X_c^\top v_i$  is an eigenvector of  $\Sigma$ . **Not necessarily normalized !!**
- ▶ **The decomposition can be done on  $S$  instead! Cost  $O(n^3)$ .** Given  $u_i$  with  $\|u_i\| = 1$ ,  $v_i$  can be recovered (after normalization  $\|v_i\| = 1$ ) as

$$v_i = \frac{1}{\sqrt{(n-1)\lambda_i}} X_c^\top u_i$$

## Kernel PCA

What to do when data are far from being linearly “oriented” ?



## Kernel PCA

What to do when data are far from being linearly “oriented” ?

- ▶ Assume the data is centered  $\hat{\mu} = 0$ . Then  $S_{ij} = \frac{1}{n-1} x_i^\top x_j$ .

## Kernel PCA

What to do when data are far from being linearly “oriented” ?

- ▶ Assume the data is centered  $\hat{\mu} = 0$ . Then  $S_{ij} = \frac{1}{n-1} x_i^\top x_j$ .
- ▶ (High-dim) PCA **depends only on the inner products between samples.**

## Kernel PCA

What to do when data are far from being linearly “oriented” ?

- ▶ Assume the data is centered  $\hat{\mu} = 0$ . Then  $S_{ij} = \frac{1}{n-1} x_i^\top x_j$ .
- ▶ (High-dim) PCA **depends only on the inner products between samples.**
- ▶ We can improve PCA by transforming the data into a higher-dimensional space  $\Phi(x_i) \in \mathcal{H}$ , **as long as we know how to compute  $\langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}$**

## Some properties of Kernel function

### Definition (Positive semi-definite kernel)

A **symmetric** kernel  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  is said to be positive semi-definite (psd) iff

$$\forall n \in \mathbb{N}, \quad \forall \xi_1 \dots \xi_n \in \mathbb{R}, \quad \forall \mathbf{x}_1 \dots \mathbf{x}_n \in \mathbb{R}^d, \quad \sum_{i,j}^n \xi_i \xi_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

## Some properties of Kernel function

### Definition (Positive semi-definite kernel)

A **symmetric** kernel  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  is said to be positive semi-definite (psd) iff

$$\forall n \in \mathbb{N}, \quad \forall \xi_1 \dots \xi_n \in \mathbb{R}, \quad \forall \mathbf{x}_1 \dots \mathbf{x}_n \in \mathbb{R}^d, \quad \sum_{i,j}^n \xi_i \xi_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

This is true for any inner product :  $\sum_{i,j} \xi_i \xi_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} = \| \sum_i \xi_i \phi(\mathbf{x}_i) \|_{\mathcal{H}}^2$ .

## Some properties of Kernel function

### Definition (Positive semi-definite kernel)

A **symmetric** kernel  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  is said to be positive semi-definite (psd) iff

$$\forall n \in \mathbb{N}, \quad \forall \xi_1 \dots \xi_n \in \mathbb{R}, \quad \forall \mathbf{x}_1 \dots \mathbf{x}_n \in \mathbb{R}^d, \quad \sum_{i,j} \xi_i \xi_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

This is true for any inner product :  $\sum_{i,j} \xi_i \xi_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} = \| \sum_i \xi_i \phi(\mathbf{x}_i) \|_{\mathcal{H}}^2$ .

**But the converse is also true !!**

### Theorem (Mercer Theorem)

*For every positive semi-definite kernel  $k$ , there exists a Hilbert space  $\mathcal{H}$  and a feature map  $\phi : \mathbb{R}^d \rightarrow \mathcal{H}$  such that  $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$ .*



## Some properties of Kernel function

### Definition (Positive semi-definite kernel)

A **symmetric** kernel  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  is said to be positive semi-definite (psd) iff

$$\forall n \in \mathbb{N}, \quad \forall \xi_1 \dots \xi_n \in \mathbb{R}, \quad \forall \mathbf{x}_1 \dots \mathbf{x}_n \in \mathbb{R}^d, \quad \sum_{i,j} \xi_i \xi_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

This is true for any inner product :  $\sum_{i,j} \xi_i \xi_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} = \| \sum_i \xi_i \phi(\mathbf{x}_i) \|_{\mathcal{H}}^2$ .

**But the converse is also true !!**

### Theorem (Mercer Theorem)

*For every positive semi-definite kernel  $k$ , there exists a Hilbert space  $\mathcal{H}$  and a feature map  $\phi : \mathbb{R}^d \rightarrow \mathcal{H}$  such that  $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$ .*

It is possible to choose  $\mathcal{H}$  as a **space of functions**, such that  $k(\cdot, \mathbf{x}) \in \mathcal{H}$ , and  $\langle f, k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = f(\mathbf{x})$ . It is called the **Reproducing Kernel Hilbert Space** (RKHS) associated to  $k$ .

## Operations on kernels

Let  $k_1$  and  $k_2$  be psd, and  $\lambda_{1,2} > 0$  then :

1.  $\lambda_1 k_1$ , (multiplication by a positive scalar)
2.  $\lambda_1 k_1 + \lambda_2 k_2$ , (sum of kernels),
3.  $k_1 k_2$ , (product of kernels),
4.  $\exp(k_1)$ , (exponential of kernel),
5.  $(x_i, x_j) \mapsto g(x_i)g(x_j)k_1(x_i, x_j)$ , with  $g : \mathbb{R}^d \rightarrow \mathbb{R}$ , (multiplication by a function)

are all positive semi-definite, hence **valid kernels**.

- 🔗 These operations allow us to create more complicated kernels by combining simple ones.

## Choosing the Kernel function

### Usual kernel functions

- ▶ Linear kernel ( $\mathcal{F} \equiv \mathbb{R}^d$ ) :  $k(x, x') = x^T x'$
- ▶ Polynomial kernel (dimension of  $\mathcal{F}$  increases with the order  $d$ )


$$k(x, x') = (x^T x')^d \quad \text{or} \quad (x^T x' + 1)^d$$

- ▶ Gaussian radial function ( $\mathcal{F}$  with infinite dimension)

$$k(x, x') = \exp(-\gamma \|x - x'\|^2)$$

- ▶ Neural net kernel ( $\mathcal{F}$  with infinite dimension)

$$k(x, x') = \tanh(\kappa_1 x^T x' + \kappa_2)$$

- ▶  standard practice is to estimate the optimal kernel parameters by **cross-validation**

## Kernel PCA

## Idea

Replace  $x_i^\top x_j$  with  $k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle_{\mathcal{H}}$ ...

☞ Just replace  $S = \frac{1}{n-1} X_c X_c^\top$  with  $S_K = \frac{1}{n-1} K$ , where  $K = [k(x_i, x_j)]_{ij}$ .

## Kernel PCA

### Idea

Replace  $x_i^\top x_j$  with  $k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle_{\mathcal{H}}$ ...

☞ Just replace  $S = \frac{1}{n-1} X_c X_c^\top$  with  $S_K = \frac{1}{n-1} K$ , where  $K = [k(x_i, x_j)]_{ij}$ .

- Compute the eigenvalues/eigenvectors  $(\lambda_k, u_k)$  of  $S_K$ .

## Kernel PCA

## Idea

Replace  $x_i^\top x_j$  with  $k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle_{\mathcal{H}}$ ...

☞ Just replace  $S = \frac{1}{n-1} X_c X_c^\top$  with  $S_K = \frac{1}{n-1} K$ , where  $K = [k(x_i, x_j)]_{ij}$ .

- ▶ Compute the eigenvalues/eigenvectors  $(\lambda_k, u_k)$  of  $S_K$ .
- ▶ It is generally impossible to compute  $v_k = \frac{\sum_i \Phi(x_i)(u_k)_i}{\sqrt{(n-1)\lambda_k}}$  since it is in  $\mathcal{H}$ ...
- ▶ ... however for any  $x$  we can compute the **projection on the components**  $\langle \Phi(x), v_k \rangle_{\mathcal{H}} = \frac{1}{\sqrt{(n-1)\lambda_k}} \sum_i k(x, x_i)(u_k)_i$

## Kernel PCA

### Idea

Replace  $x_i^\top x_j$  with  $k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle_{\mathcal{H}}$ ...

Just replace  $S = \frac{1}{n-1} X_c X_c^\top$  with  $S_K = \frac{1}{n-1} K$ , where  $K = [k(x_i, x_j)]_{ij}$ .

- ▶ Compute the eigenvalues/eigenvectors  $(\lambda_k, u_k)$  of  $S_K$ .
- ▶ It is generally impossible to compute  $v_k = \frac{\sum_i \Phi(x_i)(u_k)_i}{\sqrt{(n-1)\lambda_k}}$  since it is in  $\mathcal{H}$ ...
- ▶ ... however for any  $x$  we can compute the **projection on the components**  

$$\langle \Phi(x), v_k \rangle_{\mathcal{H}} = \frac{1}{\sqrt{(n-1)\lambda_k}} \sum_i k(x, x_i)(u_k)_i$$

### Centering

- ▶ Recall that we have assumed  $\hat{\mu} = 0$ , or equivalently, we work with  $\tilde{x} = x - \hat{\mu}$ .
- ▶ In the kernel space, we would like to work with  $\tilde{\Phi}(x) = \Phi(x) - \frac{1}{n} \sum_i \Phi(x_i)$ .
- ▶ The inner product becomes  

$$\langle \tilde{\Phi}(x), \tilde{\Phi}(x') \rangle_{\mathcal{H}} = k(x, x') - \frac{1}{n} \sum_i [k(x, x_i) + k(x', x_i)] + \frac{1}{n^2} \sum_{ij} k(x_i, x_j)$$

## Kernel PCA summary

### Kernel PCA

1. Construct the normalized Kernel Matrix

$$\tilde{K} = K - UK - KU + UKU$$

where  $U = \mathbf{1}_{n \times n} / n$ .



## Kernel PCA summary

### Kernel PCA

1. Construct the normalized Kernel Matrix

$$\tilde{K} = K - UK - KU + UKU$$

where  $U = \mathbf{1}_{n \times n} / n$ .

2. Solve the eigenvalue problem

$$S_{\tilde{K}} u_k = \lambda_k u_k$$

## Kernel PCA summary

### Kernel PCA

1. Construct the normalized Kernel Matrix

$$\tilde{K} = K - UK - KU + UKU$$

where  $U = \mathbf{1}_{n \times n} / n$ .

2. Solve the eigenvalue problem

$$S_{\tilde{K}} u_k = \lambda_k u_k$$

3. The projection of  $x$  is  $z_k = \langle \tilde{\Phi}(x), v_k \rangle$  with  $v_k = \frac{\sum_i (u_k)_i \tilde{\Phi}(x_i)}{\sqrt{\lambda_k(n-1)}}$ , i.e.

$$z_k = \frac{\tilde{K}_x^\top u_k}{\sqrt{\lambda_k(n-1)}}$$

where  $K_x = [k(x, x_i)]_i$  and  $\tilde{K}_x = K_x - UK_x - K \frac{\mathbf{1}_n}{n} + UK \frac{\mathbf{1}_n}{n}$ .

## Example of KPCA

## Example using Gaussian kernel

