Kernel Principal Component Analysis (KPCA)

1. Mathematics (Linear Algebra & Functional Analysis)

- The **kernel** of a linear transformation is the set of all vectors that map to the zero vector.
- Mathematically, for a function $f : V \to W$, the kernel is:

$$\ker(f) = \{v \in V \mid f(v) = 0\}$$

2. Computer Science (Operating Systems)

- The **kernel** is the core component of an operating system (OS) that manages system resources and communication between hardware and software.
- It is responsible for process scheduling, memory management, and device control.
- Examples: **Monolithic Kernel (Linux), Microkernel (Minix), Hybrid Kernel (Windows).**

3. Machine Learning & Support Vector Machines (SVMs)

- A **kernel function** transforms input data into a higher-dimensional space to make it easier to classify or analyze.
- Example: **Radial Basis Function (RBF), Polynomial Kernel.**

1

4. Image Processing & Computer Vision

- A **kernel** (or filter) is a small matrix used for convolution operations in image processing, such as edge detection, blurring, or sharpening.
- Example: **Sobel kernel for edge detection.**

5. Graph Theory

- A **kernel** of a directed graph is an independent set of vertices such that every vertex outside the kernel has a neighbor in the kernel.

6. Statistics & Probability

- A **kernel** in kernel density estimation (KDE) is a function used to estimate the probability density function (PDF) of a dataset.
- Example: **Gaussian Kernel in KDE.**

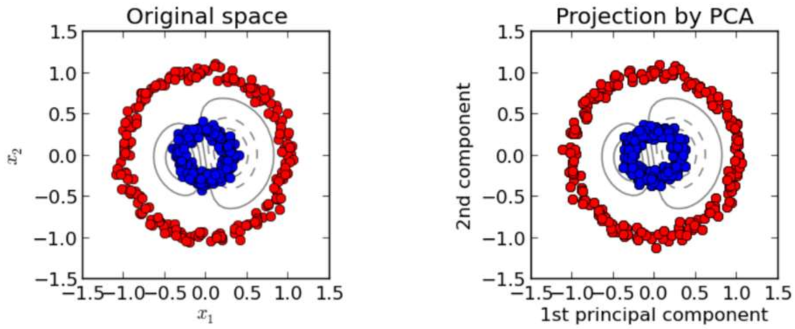
7. Neural Networks & Deep Learning

- In **Convolutional Neural Networks (CNNs)**, a **kernel** is a filter that slides over an image to extract features like edges or textures.

2

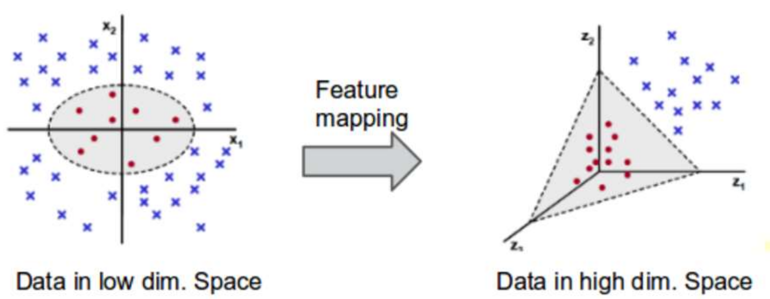
The motivation of kernel method

- Why should we need kernel method in pattern analysis?
  - We need a method to capture "non-linear data" pattern
  - PCA is linear, and cannot classify non-linear data effectively.



3

- What is kernel method?
  - Mapping data to higher dimensions (often) where contain linear data patterns
  - Data becomes linearly separable in the new feature space
  - Feature mapping function $\phi = \mathcal{R}^2 \mapsto \mathcal{R}^3$, $(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$



4

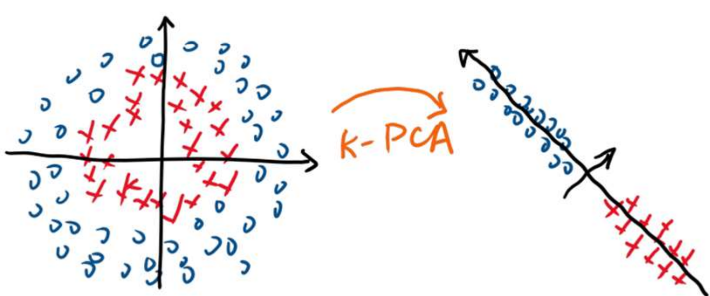
The basic idea of kernel method

- What is the goal of kernel method?
    - To get one kernel function to figure out a certain way of mapping
    - To use kernel function $\mathcal{K}$ to know geometry feature in the new space to classify data patterns

$$\begin{aligned}
\phi(x)^T\phi(z) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T(z_1^2, \sqrt{2}z_1z_2, z_2^2) \\
&= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\
&= (x_1z_1 + x_2z_2)^2 \\
&= (x^Tz)^2 \\
&= \mathcal{K}(x, z)
\end{aligned} \tag{1}$$

5

- Data is originally difficult for PCA
- Find a nonlinear transform
- Idea: Leverage the kernel trick: $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \langle\phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)})\rangle$
- Example: Left is hard for PCA. After K-PCA, right has a clear principal component.



6

Kernel for Covariance Matrix

- Assume $\phi(\boldsymbol{x}^{(n)})$ has zero mean. Then consdier the covariance matrix

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}^{(n)}(\boldsymbol{x}^{(n)})^T.$$

- Replacing the outer products by feature transforms

$$\boldsymbol{x}^{(n)} \quad \rightarrow \phi(\boldsymbol{x}^{(n)}),$$

  for some nonlinear transformation ϕ.
- If this can be done, then the covariance will become

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{n=1}^{N} \phi(\boldsymbol{x}^{(n)})\phi(\boldsymbol{x}^{(n)})^T.$$

- But this is not enough because a kernel needs an inner product

$$k(\boldsymbol{x}^{(n)}, \boldsymbol{x}^{(m)}) = \phi(\boldsymbol{x}^{(n)})^T\phi(\boldsymbol{x}^{(m)}).$$

7

Kernel Trick

- Recall: PCA solves the eigen-decomposition problem:

$$\boldsymbol{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$$

  So we also need to consider $\boldsymbol{u}$.
- How about this candidate? (Recall: In Kernel Method we express the model parameter as a linear combination of the samples):

$$\boldsymbol{u} = \sum_{n=1}^{N} \alpha_n\phi(\boldsymbol{x}^{(n)}).$$

- Substitute this into the equation $\boldsymbol{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$:

$$\underbrace{\left(\frac{1}{N}\sum_{n=1}^{N} \phi(\boldsymbol{x}^{(n)})\phi(\boldsymbol{x}^{(n)})^T\right)}_{\boldsymbol{\Sigma}}\underbrace{\left(\sum_{m=1}^{N} \alpha_m\phi(\boldsymbol{x}^{(m)})\right)}_{\boldsymbol{u}} = \underbrace{\lambda\left(\sum_{n=1}^{N} \alpha_n\phi(\boldsymbol{x}^{(n)})\right)}_{\lambda\boldsymbol{u}}$$

8

Kernel Trick

- This means

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\boldsymbol{x}^{(n)})\left(\sum_{m=1}^{N}\alpha_m\phi(\boldsymbol{x}^{(n)})^T\phi(\boldsymbol{x}^{(m)})\right)=\lambda\sum_{n=1}^{N}\alpha_n\phi(\boldsymbol{x}^{(n)})$$

- Recognizing $\phi(\boldsymbol{x}^{(n)})^T\phi(\boldsymbol{x}^{(m)})=k(\boldsymbol{x}^{(n)},\boldsymbol{x}^{(m)})$:

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\boldsymbol{x}^{(n)})\left(\sum_{m=1}^{N}\alpha_n k(\boldsymbol{x}^{(n)},\boldsymbol{x}^{(m)})\right)=\lambda\sum_{n=1}^{N}\alpha_n\phi(\boldsymbol{x}^{(n)})$$

- Multiply $\phi(\boldsymbol{x}^{(\ell)})^T$ on both sides.

$$\frac{1}{N}\sum_{n=1}^{N}k(\boldsymbol{x}^{(\ell)},\boldsymbol{x}^{(n)})\left(\sum_{m=1}^{N}\alpha_n k(\boldsymbol{x}^{(n)},\boldsymbol{x}^{(m)})\right)=\lambda\sum_{n=1}^{N}\alpha_n k(\boldsymbol{x}^{(\ell)},\boldsymbol{x}^{(n)})$$

- This is $\frac{1}{N}\boldsymbol{K}(\boldsymbol{K}\boldsymbol{\alpha})=\lambda\boldsymbol{K}\boldsymbol{\alpha}$.

9

Eigenvectors of K-PCA

- Rearrange the terms we have that $\boldsymbol{K}^2\boldsymbol{\alpha}=N\lambda\boldsymbol{K}\boldsymbol{\alpha}$.
- We can remove one of the $\boldsymbol{K}$'s since it only causes issues with zero-eigenvalues which are not important to us anyway. So we have

$$\boldsymbol{K}\boldsymbol{\alpha}=N\lambda\boldsymbol{\alpha}. \tag{2}$$

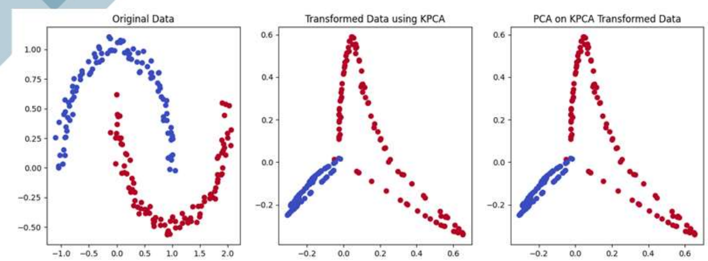
- This is just another eigen-decomposition problem. We moved from $\boldsymbol{\Sigma}\boldsymbol{u}=\lambda\boldsymbol{u}$ to $\boldsymbol{K}\boldsymbol{\alpha}=N\lambda\boldsymbol{\alpha}$. Note that $\boldsymbol{\alpha}$ is the coefficients for $\boldsymbol{u}$:

$$\boldsymbol{u}=\sum_{n=1}^{N}\alpha_n\phi(\boldsymbol{x}^{(n)})=\boldsymbol{\Phi}\boldsymbol{\alpha},$$

where $\boldsymbol{\Phi}=[\phi(\boldsymbol{x}^{(1)}),\ldots,\phi(\boldsymbol{x}^{(N)})]$ is the transformed data matrix. Recall $\boldsymbol{\Phi}\boldsymbol{\Phi}^T=\boldsymbol{K}$ is the kernel matrix where

$$[\boldsymbol{K}]_{ij}=\phi(\boldsymbol{x}^{(i)})^T\phi(\boldsymbol{x}^{(j)}).$$

10



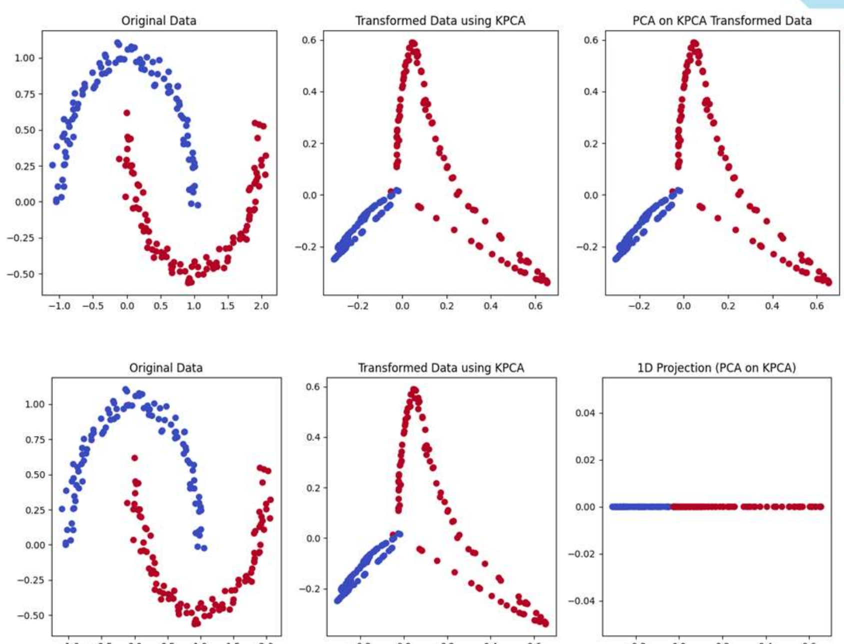

```
# Generate a non-linear dataset (moons dataset)
X, y = make_moons(n_samples=200, noise=0.05, random_state=42)

# Apply Kernel PCA with RBF kernel Radial Basis Function
kpca = KernelPCA(n_components=2, kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X)

# Apply PCA on the transformed data from KPCA
pca = PCA(n_components=2)
X_pca_after_kpca = pca.fit_transform(X_kpca)
```

Visualization of PCA vs. KPCA

•**PCA**:
- Rotates the data but **cannot separate the two classes** because they are **not linearly separable**.
- The principal components capture variance but do not separate clusters.

•**KPCA (with RBF kernel)**:
- Maps the data into a **higher-dimensional space**, where the moons can be **linearly separated**.
- Now, even a simple classifier like **linear SVM** can separate the two classes.

11



12

Comparison of PCA and KPCA

Feature	PCA (Principal Component Analysis)	KPCA (Kernel Principal Component Analysis)
Nature	Linear	Non-linear
Working Principle	Finds principal components by maximizing variance	Uses kernel trick to map data to higher-dimensional space before applying PCA
Transformation	Directly applies eigenvalue decomposition on the covariance matrix	Transforms data using a kernel function before performing PCA
Computational Complexity	Efficient, scales well with large datasets	Computationally expensive due to the kernel matrix calculation
Handles Non-Linearity?	No, it only works well for linear data patterns	Yes, it captures complex non-linear relationships
Interpretability	Easier to interpret as eigenvectors represent original feature directions	Harder to interpret because transformed features exist in an implicit high-dimensional space
Usage Scenario	Suitable for cases where data has a linear structure (e.g., image compression, noise reduction)	Useful when data has non-linear relationships (e.g., facial recognition, anomaly detection)

13

Kernel Principal Component Analysis (KPCA)

Kernel Principal Component Analysis (KPCA) is a non-linear extension of Principal Component Analysis (PCA) that leverages the kernel trick to map data into a higher-dimensional space before applying PCA. This allows it to handle complex, non-linear structures in the data.

Why KPCA?

PCA is limited to capturing linear structures in the data. However, many real-world datasets have non-linear relationships that PCA fails to capture. KPCA overcomes this by implicitly transforming data into a high-dimensional space where linear separation is possible.

Example of PCA vs KPCA

PCA: If data is shaped like a circle (e.g., concentric circles), PCA cannot effectively separate the points.

KPCA: Maps the data into a higher dimension (e.g., 3D space), where it becomes linearly separable.

The Mathematics Behind KPCA

KPCA extends PCA using the kernel trick, which allows computations in a high-dimensional space without explicitly transforming the data.

14

Step 1: Map Data to a Higher-Dimensional Space

Instead of working in the original space $\mathbb{R}^n$, we define a **non-linear** mapping function:

$$\Phi : \mathbb{R}^n \to \mathbb{R}^m \quad (\text{where } m \gg n)$$

This function maps input data x_i to a higher-dimensional space.

Step 2: Compute the Kernel Matrix

Since explicitly computing $\Phi(x)$ is computationally expensive, KPCA uses the **kernel trick**. A kernel function $K(x_i, x_j)$ computes the dot product in high-dimensional space:

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$$

Common kernel functions:

1. **Polynomial Kernel:** $K(x, y) = (x \cdot y + c)^d$
2. **Radial Basis Function (RBF) Kernel:** $K(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$
3. **Sigmoid Kernel:** $K(x, y) = \tanh(\alpha x \cdot y + c)$

15

Step 3: Center the Kernel Matrix

Since PCA requires the data to be centered, we compute a centered kernel matrix:

$$K' = K - 1_N K - K 1_N + 1_N K 1_N$$

where 1_N is an $N \times N$ matrix with all elements equal to $\frac{1}{N}$.

Step 4: Compute Eigenvalues and Eigenvectors

Perform eigenvalue decomposition on the kernel matrix K':

$$K' v_i = \lambda_i v_i$$

where:

- v_i are the eigenvectors (principal components),
- λ_i are the eigenvalues.

Step 5: Transform Data

Finally, the transformed data in the new feature space is obtained as:

$$z_i = \sum_{j=1}^{N} v_j K(x_j, x_i)$$

16

Advantages of KPCA

- **Captures non-linear patterns** that PCA cannot.
- **More flexibility** through different kernel choices.
- **Better feature extraction** for tasks like image recognition, anomaly detection, and clustering.

Limitations of KPCA

- **Computationally expensive**: Kernel computation requires storing an NxN matrix, making it slow for large datasets.
- **Choice of kernel matters**: The right kernel function must be selected for optimal performance.
- **Hard to interpret**: The transformed features exist in an implicit space, making interpretation difficult.

When to Use?

- **Use PCA** when the dataset is linearly separable, and reducing dimensionality without significant information loss is the goal.
- **Use KPCA** when the dataset exhibits complex, non-linear relationships, and capturing these relationships is crucial for better representation.

17

Applications of KPCA

- **Face recognition**: Extracts non-linear features from images.
- **Anomaly detection**: Identifies non-linear structures in data.
- **Data clustering**: Improves clustering performance for non-linearly separable data.
- **Bioinformatics**: Used for protein classification and gene expression analysis.

18