PRINCIPAL COMPONENTS ANALYSIS (PCA)

Machine Learning

IIIy CSE IDP and IDDMP

Refer : Stephen Marsland

Motivation

1. **High-dimensional data** can be noisy and redundant
2. We seek a **lower-dimensional representation** that preserves the essential structure
3. By rotating to a new set of axes, we often find that some directions carry **very little variance**
4. Eliminating those low-variance directions can:
    a. Reduce noise
    b. Improve learning performance
    c. Simplify visualization
5. **Illustration:** (Figure – 1)
    d. Original data lie on an ellipse at 45° to the axes
    e. After rotation, data align with the new x-axis, and the y-axis shows minimal spread

Figure - 1

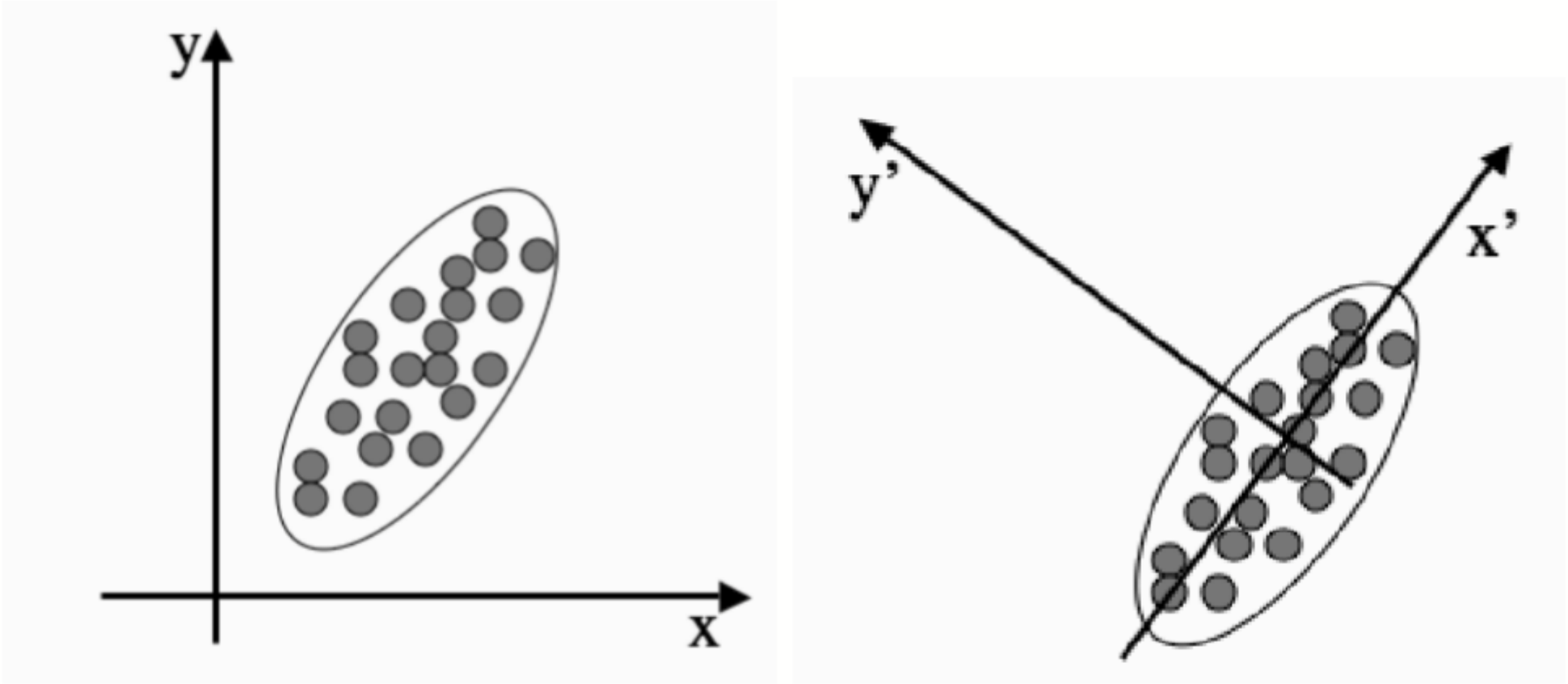

FIGURE 6.6 Two different sets of coordinate axes. The second consists of a rotation and translation of the first and was found using Principal Components Analysis.

What Is Principal Components Analysis?

1. **Center the data**
   a. Subtract the mean of each feature so that each has zero mean
2. **Find the first principal component**
   b. Identify the direction (unit vector) along which the data have **maximum variance**
3. **Extract subsequent components**
   c. Project data orthogonally to the first component
   d. Find the next direction of maximal remaining variance
4. **Iterate**
   e. Continue until you have as many components as original dimensions
5. **Result**
   f. A new coordinate system where the covariance matrix is **diagonal**
   g. Components are **uncorrelated**
   h. Components with **low variance** can be dropped for dimensionality reduction

Formalizing the Rotation

Goal: Rotate data so its covariance becomes diagonal

Data matrix: $X \in R^{NxM}$ (N features, M samples)

Rotation matrix: P (orthonormal, so $P^T=P^{-1}$)

Transformed data:

$$Y=P^T X$$

Covariance of Y :

$$\text{cov}(\mathbf{Y}) = \text{cov}(\mathbf{P}^T\mathbf{X}) = \begin{pmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \lambda_N \end{pmatrix} = \quad \mathbf{P}^T\text{cov}(\mathbf{X})\mathbf{P}$$

Key fact: We choose P so that $\mathbf{P}^T\text{cov}(\mathbf{X})\mathbf{P}$ is diagonal

Eigen-Decomposition & Principal Components

Write P in terms of its column vectors (eigenvectors):

$$\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N]$$

Since cov(Y) is a diagonal, we have

$$\mathbf{P}\mathrm{cov}(\mathbf{Y}) = [\lambda_1\mathbf{p}_1, \lambda_2\mathbf{p}_2, \ldots, \lambda_N\mathbf{p}_N]$$

Defining $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_N)^T$ and $\mathbf{Z} = \mathrm{cov}(\mathbf{X}))$, it follows that for each p_i

$$\boldsymbol{\lambda}\mathbf{p}_i = \mathbf{Z}\mathbf{p}_i$$

Interpretation:

Each p_i is an eigenvector of **Z** with eigenvalue λ_i ,representing a principal component direction and its variance.

Eigenvectors & Eigenvalues

1. Recall from $\mathbf{P}\mathrm{cov}(\mathbf{Y}) = [\lambda_1\mathbf{p}_1, \lambda_2\mathbf{p}_2, \ldots, \lambda_N\mathbf{p}_N]$ $\boldsymbol{\lambda}\mathbf{p}_i = \mathbf{Z}\mathbf{p}_i$ and

2. λ is a column vector of eigenvalues; it rescales each p_i
    - No rotation or mixing—only stretching along those directions

3. Eigenvectors (p_i) are special directions that **Z** does not rotate

4. For a square symmetric matrix **A**:
    - All eigenvectors are orthogonal
    - Normalizing them makes **E** a rotation matrix ($E^{-1} = E^{T}$)

Spectral Decomposition & Variance Interpretation

Spectral decomposition of a symmetric matrix **A**:

$$A = EDE^T$$

- E: orthonormal eigenvector matrix (rotates into eigenspace)
- D: diagonal matrix of eigenvalues (stretches along each axis)
- E^T: rotates back to original space

In the context of cov(X):

- Eigenvalues λ_i quantify how much variance lies along p_i
  - **Large λ_i :** high variance direction
  - **Small λ_i :** data tightly clustered
- Extremely small λ_i indicate directions with negligible variation

PCA Algorithm

The Principal Components Analysis Algorithm

- Write N datapoints $\mathbf{x}_i = (\mathbf{x}_{1i}, \mathbf{x}_{2i}, \ldots, \mathbf{x}_{Mi})$ as row vectors
- Put these vectors into a matrix $\mathbf{X}$ (which will have size $N \times M$)
- Centre the data by subtracting off the mean of each column, putting it into matrix $\mathbf{B}$
- Compute the covariance matrix $\mathbf{C} = \frac{1}{N}\mathbf{B}^T\mathbf{B}$
- Compute the eigenvalues and eigenvectors of $\mathbf{C}$, so $\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D}$, where $\mathbf{V}$ holds the eigenvectors of $\mathbf{C}$ and $\mathbf{D}$ is the $M \times M$ diagonal eigenvalue matrix
- Sort the columns of $\mathbf{D}$ into order of decreasing eigenvalues, and apply the same order to the columns of V
- Reject those with eigenvalue less than some η, leaving L dimensions in the data

Relation with the Multi-Layer Perceptron

Auto-associative MLP & PCA

- Hidden layer of an auto-encoder learns directions similar to principal components
- Both perform only linear transformations (rotations & scalings)

Deeper MLP as Non-linear PCA

- With four layers (input → hidden_1 → bottleneck → hidden_2 → output):
  - Layer 1: non-linear feature transform
  - Layer 2 (bottleneck): PCA on transformed features
  - Layers 3–4: reconstruct original inputs
- Results in PCA of a non-linear mapping of inputs

Implication:

- Captures variance in directions that may not be linearly separable
- Bridges PCA concepts with kernel methods (see Section 8.2)

Kernel PCA – Motivation & Formulation

Limitation of standard PCA:

- Assumes principal directions are **linear** (straight lines)

Kernel trick extension:

1. Map data non-linearly: $\Phi : x \mapsto \Phi(x)$ in feature (kernel) space
2. Compute covariance in kernel space:
$$\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N}\Phi(\mathbf{x}_n)\Phi(\mathbf{x}_n)^T$$
3. Solve eigenproblem:
$$\lambda\left(\Phi(\mathbf{x}_i)\mathbf{V}\right) = \left(\Phi(\mathbf{x}_i)\mathbf{C}\mathbf{V}\right) \;\; i = 1 \ldots N \text{ where } \mathbf{V} = \sum_{j=1}^{N}\boldsymbol{\alpha}_j\Phi(\mathbf{x}_j)$$
4. Form kernel matrix $K \in R^{N\times N}$ with $\mathbf{K}_{(i,j)} = \left(\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)\right)$
5. Reduce to $N\lambda\mathbf{K}\boldsymbol{\alpha} = \mathbf{K}^2\boldsymbol{\alpha} \longrightarrow N\lambda\boldsymbol{\alpha} = \mathbf{K}\boldsymbol{\alpha}$

Projection of new point:

$$\left(\mathbf{V}^k\cdot\Phi(\mathbf{x})\right) = \sum_{i=1}^{N}\boldsymbol{\alpha}_i^k\left(\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)\right)$$

Kernel PCA Algorithm

The Kernel PCA Algorithm

- Choose a kernel and apply it to all pairs of points to get matrix $\mathbf{K}$ of distances between the pairs of points in the transformed space
- Compute the eigenvalues and eigenvectors of $\mathbf{K}$
- Normalise the eigenvectors by the square root of the eigenvalues
- Retain the eigenvectors corresponding to the largest eigenvalues

Computational Considerations & Examples

- **Complexity:**
  - Naïve implementation: $O(N^3)$ (full eigen-decomposition of K)
  - With truncation to top eigenpairs: can reduce toward $O(N^2)$
- **Use cases:**
  - Linear PCA fails on non-linear structures (e.g., concentric circles)
  - Kernel PCA can **unfold** such structure into separable components
- **Illustrative outputs:**
  - *Iris dataset:* kernel PCA separates classes similarly to linear PCA (Fig – 2)
  - *Concentric circles:* a single kernel principal component suffices to separate rings (Fig – 3)

Figure - 2

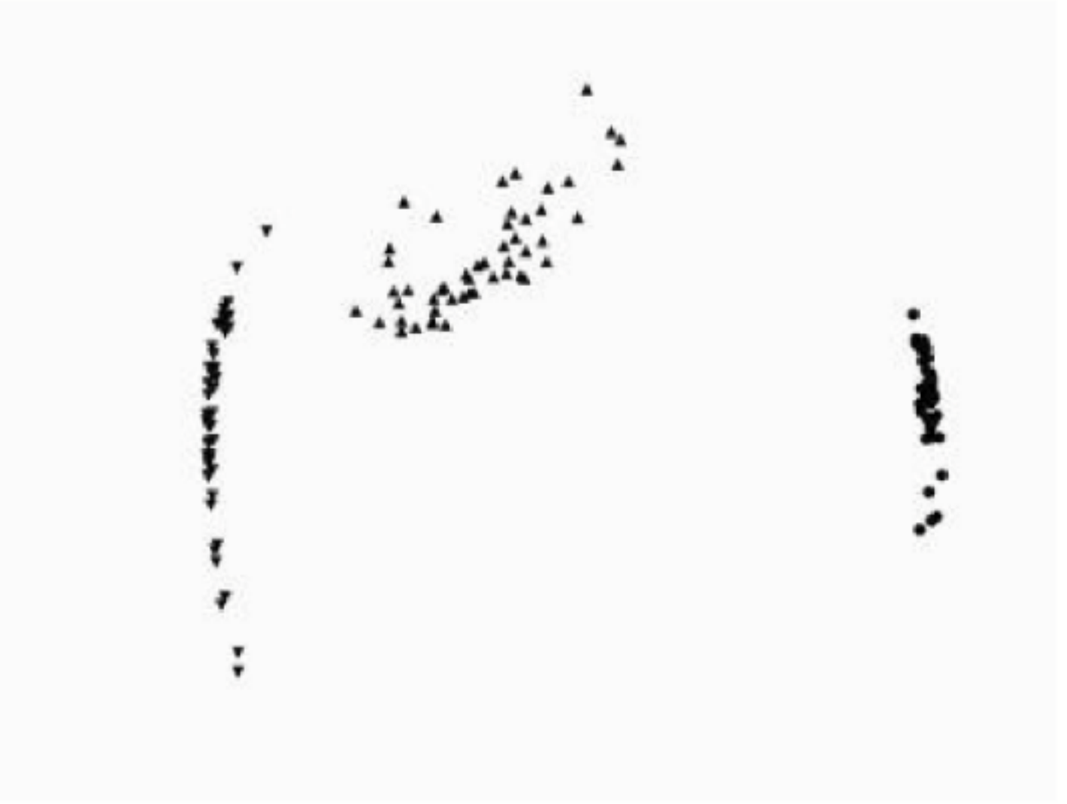

FIGURE 6.9 Plot of the first two non-linear principal components of the iris data, (using the Gaussian kernel) showing that the three classes are clearly distinguishable.

Figure - 3

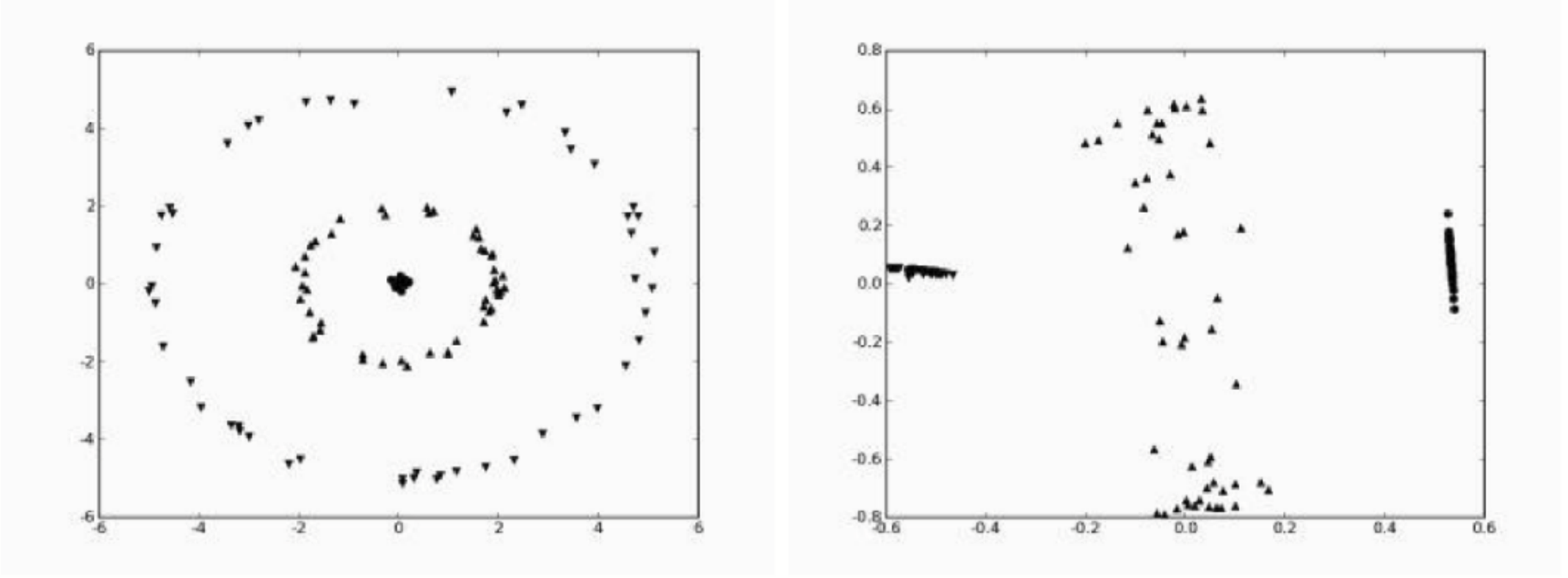

FIGURE 6.10 A very definitely non-linear dataset consisting of three concentric circles, and the (Gaussian) kernel PCA mapping of the iris data, which requires only one component to separate the data.

Summary & Conclusion

1. Principal Components Analysis (PCA)

a. Rotates and centers data to align with directions of maximum variation

2. Key Insights

b. PCA is a **linear** transform: only rotations and scalings, no distortion

c. Directions with small eigenvalues correspond to low-variance (noisy) dimensions

d. Kernel PCA generalizes PCA via a nonlinear mapping $\Phi(\cdot)$ and the kernel trick

3. Connections & Extensions

e. Auto-associative MLP hidden layer approximates linear PCA

f. Deeper MLPs perform PCA on nonlinear feature transforms

g. Kernel PCA handles curved/manifold structures beyond linear separability

4. Conclusion

h. PCA provides an efficient, orthogonal basis for understanding data variance

i. Forms the foundation for dimensionality reduction and data preprocessing

j. Kernel and neural-network–based extensions enable capturing non-linear structure