Memory updated

Principal Component Analysis (PCA) – A Deep Dive

1. What is Principal Component Analysis (PCA)?

Principal Component Analysis (PCA) is a **dimensionality reduction technique** that transforms high-dimensional data into a lower-dimensional space while preserving as much variance as possible.

Why Use PCA?

- High-dimensional data is **hard to visualize and analyze**.
- PCA helps by **projecting the data onto a smaller number of dimensions**.
- It finds new **orthogonal axes** (principal components) that capture **maximum variance**.

Example Intuition

- Imagine you have **3D data** (height, weight, age).
- PCA finds the **best 2D plane** that captures most of the variation.
- This allows for a **lower-dimensional representation** without losing critical information.

2. When to Use & Avoid PCA

✅ Use PCA When:

- You want to **reduce dimensionality** while keeping most of the variance.
- You need to **visualize high-dimensional data** (e.g., projecting to 2D or 3D).
- You want to remove **correlated features** (PCA removes redundancy).

❌ Avoid PCA When:

- You need **interpretable features** (PCA transforms features into abstract principal components).
- The data is **not linearly separable** (PCA is a **linear** method).
- The dataset contains **categorical variables** (PCA is designed for continuous data).

3. PCA Step-by-Step Algorithm

The PCA process follows these steps:

1. **Standardize the Data** – Center and scale the data (subtract mean, divide by std dev).
2. **Compute Covariance Matrix** – Measure how features vary together.
3. **Compute Eigenvalues & Eigenvectors** – Find principal axes of variation.
4. **Sort & Select Principal Components** – Choose top components that capture most variance.
5. **Transform Data** – Project original data onto the new axes.

5. Pseudo Code for PCA

plaintext

```plaintext
# Step 1: Standardize the Data
1. Compute the mean of each feature.
2. Subtract the mean from the data.
3. Divide by standard deviation.

# Step 2: Compute Covariance Matrix
1. Compute covariance matrix.

# Step 3: Compute Eigenvalues & Eigenvectors
1. Solve the eigenvalue equation.
2. Sort eigenvalues in descending order.

# Step 4: Select Principal Components
1. Choose top k eigenvectors based on explained variance.

# Step 5: Transform Data
1. Project data onto selected principal components.
```

4. Mathematical Breakdown

Let's go through each step **explicitly**:

4.1 Step 1: Standardize the Data

PCA requires the data to be **zero-centered** (mean = 0) and **scaled**:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

where:

- X = original data matrix (size $n \times d$, where n is samples, d is features).
- μ = mean of each feature.
- σ = standard deviation of each feature.

4.2 Step 2: Compute Covariance Matrix

The **covariance matrix** measures relationships between features:

$$C = \frac{1}{n-1} X^T X$$

where:

- C is a $d \times d$ matrix.
- C_{ij} measures the **covariance between feature i and feature j**.

4.3 Step 3: Compute Eigenvalues & Eigenvectors

We solve the **eigenvalue equation**:

$$Cv = \lambda v$$

where:

- v are the **eigenvectors** (principal component directions).
- λ are the **eigenvalues** (variance captured by each component).

Sorting **eigenvalues in descending order** gives the **principal components in order of importance**.

4.4 Step 4: Select Top Principal Components

Choose the **top k eigenvectors** that explain the most variance:

$$\text{Explained Variance} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{d} \lambda_j}$$

where:

- k is the number of components selected.
- λ_i are the eigenvalues of selected components.

4.4 Step 4: Select Top Principal Components

Choose the **top k eigenvectors** that explain the most variance:

$$\text{Explained Variance} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{d} \lambda_j}$$

where:

- k is the number of components selected.
- λ_i are the eigenvalues of selected components.

4.5 Step 5: Transform Data

Project data onto the new principal component space:

$$X_{\text{PCA}} = X_{\text{scaled}} V_k$$

where:

- V_k contains the **top k eigenvectors**.
- X_{PCA} is the transformed data in the reduced space.

6. Fully Commented Manual Implementation (From Scratch)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

class PCA:
    """
    Implementation of Principal Component Analysis (PCA) from scratch.
    """
    def __init__(self, n_components):
        """
        Initialize PCA.
        :param n_components: Number of principal components to retain.
        """
        self.n_components = n_components
        self.mean = None
        self.components = None

    def fit(self, X):
        """
        Fit PCA to the dataset.
        :param X: Data matrix (n_samples, n_features).
        """
        # Step 1: Standardize the Data
        self.mean = np.mean(X, axis=0)
        X_centered = X - self.mean  # Centering

        # Step 2: Compute Covariance Matrix
        covariance_matrix = np.cov(X_centered, rowvar=False)

        # Step 3: Compute Eigenvalues & Eigenvectors
        eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)

        # Step 4: Sort Eigenvalues & Select Components
        sorted_indices = np.argsort(eigenvalues)[::-1]
        self.components = eigenvectors[:, sorted_indices[:self.n_components]]

    def transform(self, X):
        """
        Transform the dataset using learned principal components.
        :param X: Data matrix (n_samples, n_features).
        :return: Transformed data.
        """
        X_centered = X - self.mean
        return np.dot(X_centered, self.components)
```

```python

# Load sample dataset (Digits dataset from sklearn)
digits = load_digits()
X, y = digits.data, digits.target

# Apply PCA manually
pca = PCA(n_components=2)
pca.fit(X)
X_pca = pca.transform(X)

# Visualize PCA Results
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', alpha=0.7)
plt.colorbar(label='Digit Label')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Projection of Digits Dataset')
plt.show()

```

7. Scikit-Learn Implementation

python

Copy Edit

```python
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits

# Load dataset
X, y = load_digits(return_X_y=True)

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Explained Variance
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

8. Advantages & Disadvantages

✅ Advantages

- Reduces **dimensionality** while retaining most variance.
- Removes **correlation** between features.
- Improves **visualization** of high-dimensional data.

❌ Disadvantages

- Can lose **interpretability** (original features transformed into abstract components).
- Sensitive to **scaling** (PCA assumes features are standardized).
- Assumes **linear relationships** (fails for non-linear data).

Final Takeaways

✅ PCA is a fundamental tool for dimensionality reduction, feature extraction, and visualization.
✅ It finds the directions (principal components) that explain the most variance in data.
✅ PCA works best when data is standardized and when features are correlated.