PCA Introduction

Outline

Introduction

The steps

Python Implementation

Introduction

What is PCA?

- Unsupervised dimensionality reduction technique
- Transforms original variables into uncorrelated **principal components**
- Captures the directions of highest variance
- Common uses: visualization, noise reduction, speeding up ML

When to Use PCA

- When you have **many features** and want to reduce complexity
- Features are **correlated**
- You want to **visualize** high-dimensional data
- To help **improve performance** or reduce overfitting

Key Concepts

- **Variance:** Spread of data
- **Covariance matrix:** Relationship between features
- **Eigenvectors:** Principal directions
- **Eigenvalues:** Amount of variance explained

PCA Step-by-Step

1. **Standardize** the data
2. Compute **Covariance Matrix**
3. Compute **Eigenvectors and Eigenvalues**
4. Select top **k components**
5. **Transform** the data

The steps

Step 1: Standardize the Data

What:
Center each feature (mean $= 0$), scale to unit variance.

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

Why:
PCA is scale-sensitive. Without standardization, features with larger magnitudes will dominate the principal components.

Step 2: Compute the Covariance Matrix

What:
Calculate the covariance between each pair of features.

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

Why:
The covariance matrix reveals the linear relationships between features — essential to find directions of greatest variance.

Step 3: Compute Eigenvectors and Eigenvalues

What:
Solve the equation:

$$\mathbf{C}\mathbf{w} = \lambda\mathbf{w}$$

Eigenvectors = directions; Eigenvalues = variance explained.

Why:
To discover the axes (principal components) along which the data varies the most.

Step 4: Select Top k Components

What:
Rank eigenvalues in descending order and pick the top k.

Retain the first k eigenvectors

Why:
To reduce dimensionality while preserving most of the variance (information) in the dataset.

Step 5: Transform the Data

What:
Project the original data onto the new basis:

$$Z = X \cdot W_k$$

Why:
This gives a new set of uncorrelated features in a lower-dimensional space — useful for visualization or model input.

Python Implementation

PCA in Python

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

data = load_iris()
X = StandardScaler().fit_transform(data.data)

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

plt.scatter(X_pca[:,0], X_pca[:,1], c=data.
    target)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("PCA on Iris Dataset")
plt.show()
```

Choosing Number of Components

```
pca = PCA().fit(X)
plt.plot(np.cumsum(pca.explained_variance_ratio_
    ))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid(True)
plt.show()
```

Pros and Cons

Pros:

- Reduces overfitting
- Speeds up training
- Useful for visualization
- Removes multicollinearity

Cons:

- Loss of interpretability
- Assumes linearity
- Sensitive to feature scaling
- May discard useful info

Applications of PCA

- Face recognition and image compression
- Noise filtering and denoising
- Gene expression analysis
- Financial modeling and portfolio risk
- Data visualization in 2D/3D

Summary

- PCA is a powerful tool for reducing dimensionality
- It transforms data into uncorrelated, variance-maximizing components
- Useful for improving model performance, visualizing data, and removing noise

Thank you!