Python Programming and Basic Data Science

Shuvro Pal
Data Scientist, Markopolo.ai
AI Team Lead, Zantrik
ML Facilitator, Crowdsource by Google

PCA

What is it?

PCA is a dimensionality reduction technique. It helps simplify complex datasets by transforming them into a new set of variables called principal components.

- **Key idea:** PCA keeps the important information while reducing the number of features (dimensions).

Example: Imagine taking a 3D object (like a cube) and viewing it from the side—it becomes a 2D square. You still have much of the important information, but with fewer dimensions.

PCA

Use cases?

- Data compression
- Visualizing high-dimensional data in 2D/3D
- Speeding up machine learning algorithms by reducing feature size

PCA

Real World Examples?

Problem: In genomics, each person's DNA is represented by tens of thousands of genes. Predicting diseases from such large datasets is computationally expensive, and many genes might be irrelevant or correlated.

1. **Before PCA**: You might have 50,000 genes per individual, making it difficult to visualize or process.
2. **After PCA**: PCA reduces this to a few hundred principal components, which still explain most of the variability in gene expression.

Impact: PCA in genomics can lead to faster disease prediction models and more efficient data analysis in personalized medicine.

PCA

How it actually works?

Step-by-Step:

- **Step 1**: **Standardize** the data.

$$z = \frac{value - mean}{standard\ deviation}$$

- **Step 2**: Find the **covariance** matrix. This tells us how features vary together. If they move in the same direction, covariance is positive, and if they move in opposite directions, it's negative.

  The covariance matrix is a 3×3 data matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

PCA

How to calculate covariance?

For two features X and Y, covariance is:

$$\mathrm{Cov}(X,Y) = \frac{1}{n}\sum(X_i - \bar{X})(Y_i - \bar{Y})$$

PCA

How it actually works?

- **Step 3:** Compute the **eigenvectors** and **eigenvalues**. These give us the new directions (principal components) and their importance (variance explained).

  More simplified version:

  - Eigenvectors represent the directions of maximum variance (principal components).
  - Eigenvalues tell us how much variance each principal component explains.

- **Step 4: Transform** the data. We project the data onto the new principal components.

PCA

Some code please?

```
from sklearn.decomposition import PCA
import numpy as np
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1],
[2, 1], [3, 2]])
pca = PCA()
pca.fit(X)
print pca.components_
```

This gives an output like

[[0.83849224 0.54491354]
[0.54491354 -0.83849224]]

where every row is a principal component in the p-dimensional space (2 in this toy example). Each of these rows is an Eigenvector of the centered covariance matrix.

PCA

Some code please?

if print the explained_variance_ratio_ attribute :
[0.99244289 0.00755711]

This means that the ratio of the eigenvalue of the first principal component to the eigenvalue of the second principal component is 0.99244289:0.00755711.

PCA

More Practical example?

Imagine you have a dataset of facial images, each with thousands of pixels (features). Analyzing each pixel independently is inefficient because many features (pixels) may be correlated. PCA helps by reducing the number of pixels (features) while retaining important patterns in the data.

Apply PCA: Instead of using thousands of pixel values for classification, PCA will reduce the number of features by finding the principal components that capture most of the variance in the image data.

Result: You now have a dataset with only a few features (principal components) that still represent the important variations in the faces, making recognition faster and more efficient.

PCA

Why it's worthy?

- **Simplifies Complex Data**: PCA transforms complex datasets with many features into a smaller, more manageable set of components while retaining the key information.
- **Speeds Up Algorithms**: By reducing the number of dimensions, PCA can significantly improve the speed of machine learning algorithms.
- **Improves Visualization**: When dealing with high-dimensional data, PCA enables you to visualize the data in 2D or 3D, helping to identify patterns more easily.

Manifold Learning

What is it?

Manifold Learning is a type of dimensionality reduction technique, like PCA, but designed for non-linear data. It helps us uncover the true lower-dimensional structure (manifold) of data that lies in a higher-dimensional space.

Key idea: Real-world data often lies on a curved surface (manifold), not in a straight line or flat space. Manifold learning helps us find this underlying structure.

Example: Imagine a crumpled-up piece of paper. Although it exists in 3D, it’s essentially 2D. Manifold learning "unfolds" this paper to find its 2D structure.

Manifold Learning

Use cases?

- Speech recognition (complex vocal patterns)
- Image and video compression
- Analyzing high-dimensional biological data like brain scans

Manifold Learning

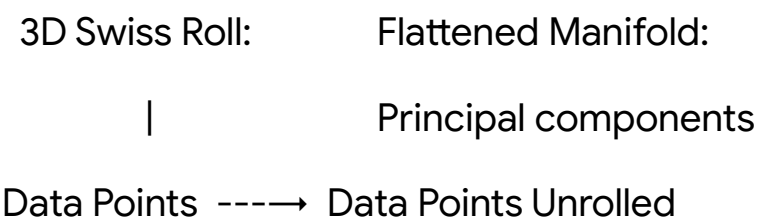
How it actually works?

Step-by-Step:

- **Step 1**: Data in High Dimensions. We start with data that seems complex and high-dimensional (many features).
- **Step 2**: Identify the Manifold. The algorithm identifies the true underlying structure (manifold) where the data actually lies.
- **Step 3**: Reduce Dimensions. We project the high-dimensional data onto this lower-dimensional manifold, keeping the essential structure.

Visual Representation:

- Imagine points on a Swiss roll (a spiral in 3D space). Manifold learning "unrolls" this Swiss roll, projecting it onto a flat 2D surface, revealing the simple underlying structure.



Manifold Learning

More theories?

High-Dimensional Data:

- Data in high dimensions is often non-linear. Manifold learning focuses on non-linear relationships between data points, unlike PCA, which handles only linear relationships.

Manifold:

- A manifold is a lower-dimensional surface embedded in a higher-dimensional space. The goal is to "flatten" the manifold while preserving the distances and relationships between points.
- Intuition: If data looks like a curved surface in 3D, manifold learning unfolds it to find its simpler structure in 2D.

Manifold Learning

More theories?

Popular Manifold Learning Algorithms:

- **Isomap (Isometric Mapping)**: Preserves the distances between points in the high-dimensional space when reducing dimensions.
- **Locally Linear Embedding (LLE):** Tries to preserve the local relationships between points, emphasizing how each point relates to its neighbors.

Key Concept: Geodesic Distance

- Unlike Euclidean distance (straight-line distance), geodesic distance measures the shortest path along the manifold. Manifold learning algorithms often rely on this to capture the true distance between points.

Manifold Learning

Any practical example?

Imagine you have a dataset with thousands of images of handwritten digits (0-9). Each image has 784 pixels (28x28), so the dataset has 784 dimensions per image. Directly analyzing this data is hard because of the high dimensionality.

Challenge: How can we visualize or analyze these images efficiently?

1. Apply Manifold Learning (LLE or Isomap):
    - The manifold learning algorithm identifies the 2D structure underlying the high-dimensional data.
    - It reduces the 784-dimensional space to a 2D representation while maintaining the relationships between images.
2. Result: We can now visualize the images in 2D space. Similar digits (e.g., 2s or 9s) will cluster together, making it easier to understand and classify the data.

Manifold Learning

Advantages?

1. **Non-Linear Dimensionality Reduction**: Manifold learning captures non-linear relationships in the data, making it more powerful than linear techniques like PCA.
2. **Helps with Visualization**: It reduces high-dimensional data to 2D or 3D, allowing us to visualize and explore complex datasets.
3. **Captures True Structure**: By uncovering the manifold, we can focus on the most meaningful aspects of the data while ignoring irrelevant dimensions or noise.

Manifold Learning

Use Case in NLP?

Problem: In NLP, words are often represented as high-dimensional vectors (using techniques like word embeddings). These vectors capture complex relationships between words, but direct analysis is difficult because of the high dimensionality.

Manifold Learning Solution:

- We can apply manifold learning (e.g., Isomap or t-SNE) to reduce the dimensionality of the word embeddings while preserving the relationships between words.

1. **Before Manifold Learning:** You have 300-dimensional word vectors, where each word is a point in this high-dimensional space. Understanding these relationships is challenging.
2. **After Manifold Learning:** The algorithm reduces the data to 2D or 3D. Now, similar words (like "king" and "queen") cluster together, and relationships between words (e.g., male/female, singular/plural) are easier to see and analyze.

Impact: This dimensionality reduction helps visualize and understand word relationships, making NLP tasks (like text clustering or classification) more interpretable and efficient.

Future of AI


সবার জন্য কৃত্তিম বুদ্ধিমত্তা
Java
আপনার সন্তানকে
মেশিন লার্নিং শিক্ষা দিন
SUFIAN SIDDIQUI