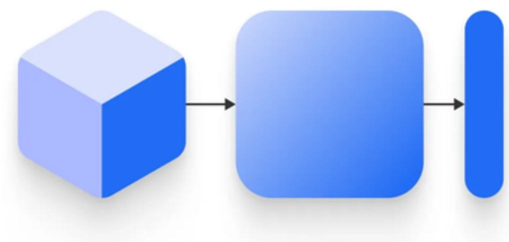
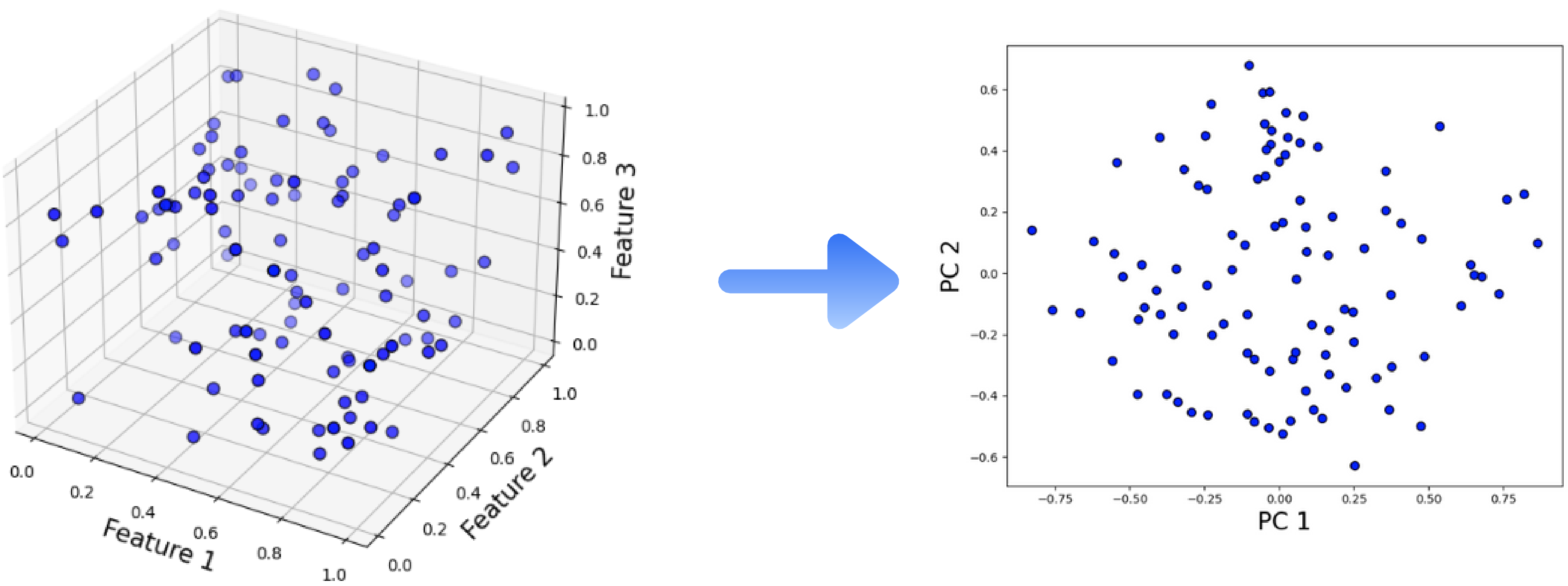
@shreyapatilu

# Dimensionality Reduction Techniques



## PCA, LDA, and KPCA Compared



# Why do we need dimensionality reduction techniques?

- **Data simplification:** Reduces high-dimensional data to a more manageable form.
- **Sparsity reduction:** Creates a denser, more informative feature space.
- **Overfitting prevention:** Fewer features can lead to more generalizable models.
- **Improved visualization:** Allows high-dimensional data to be visualized in 2D or 3D.
- **Computational efficiency:** Reduces processing time and resource requirements.
- **Mitigates curse of dimensionality:** Addresses issues related to high-dimensional spaces.
- **Feature decorrelation:** Often produces less correlated features.
- **Noise reduction:** Can filter out less important variations in the data.
- **Enables advanced techniques:** Makes data more suitable for certain algorithms (e.g., kernel methods).

## Potential drawbacks:

- New features may not have clear real-world meanings.
- Some methods can be intensive to compute.
- Some data characteristics may be lost in the process.

# Three Fundamental Techniques

| Feature | PCA | LDA | KPCA |
|---|---|---|---|
| **Supervised/Unsupervised** | Unsupervised | Supervised | Unsupervised |
| **Linearity** | Linear | Linear | Non-linear |
| **Goal** | Maximize variance | Maximize class separability | Maximize variance in higher-dimensional space |
| **Class information** | Not used | Used | Not used |
| **Scalability** | Moderate | Poor for high-dimensional data | Poor for large datasets |
| **Interpretability** | High | High | Low |
| **Handles multicollinearity** | Yes | Yes | Yes |
| **Optimal for classification** | No | Yes | No |
| **Captures non-linear relationships** | No | No | Yes |
| **Requires parameter tuning** | No | No | Yes (kernel selection) |
| **Sensitive to feature scaling** | Yes | Less sensitive | Depends on kernel |

# Principal Component Analysis (PCA):

PCA is an unsupervised dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving as much variance as possible.

**Steps of PCA :**

- Standardize the data
- Compute the covariance matrix
- Calculate eigenvectors and eigenvalues of the covariance matrix
- Sort eigenvectors by decreasing eigenvalues
- Choose top k eigenvectors as the new feature space
- Project the original data onto the new feature space

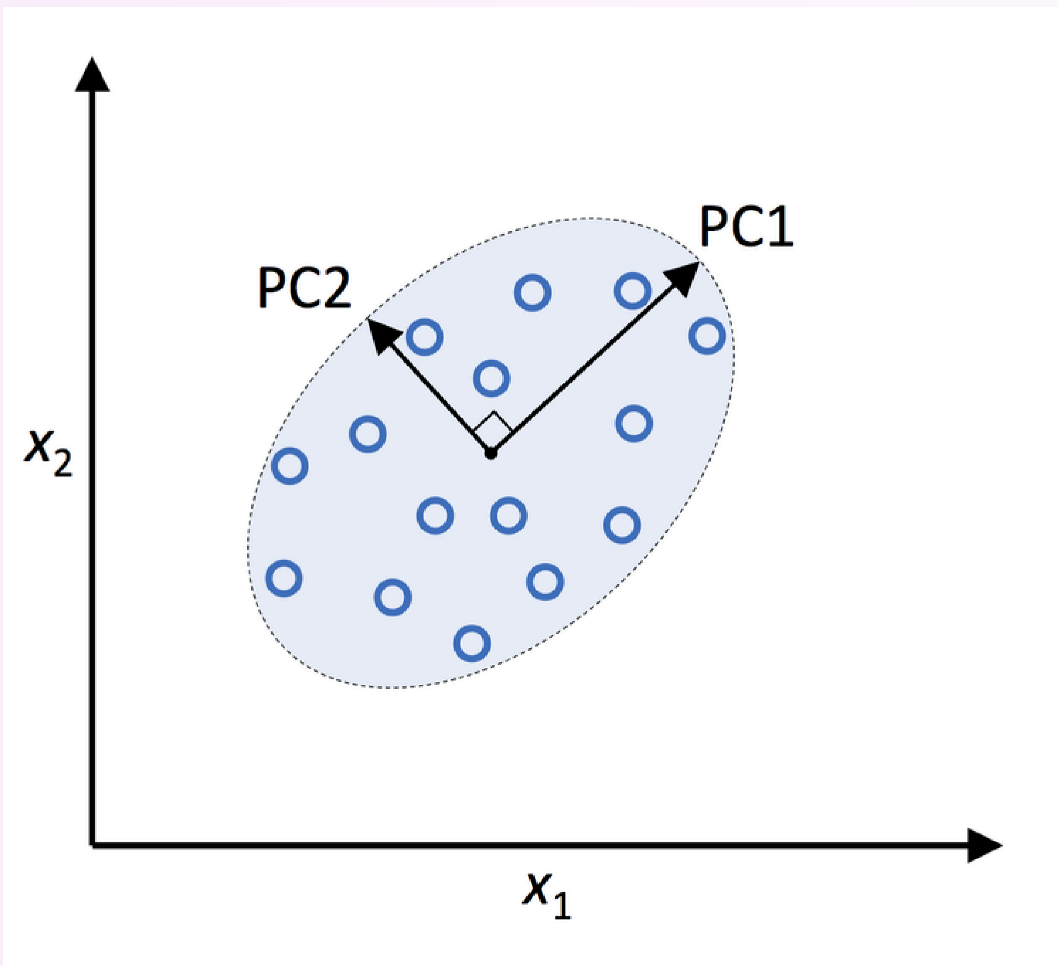
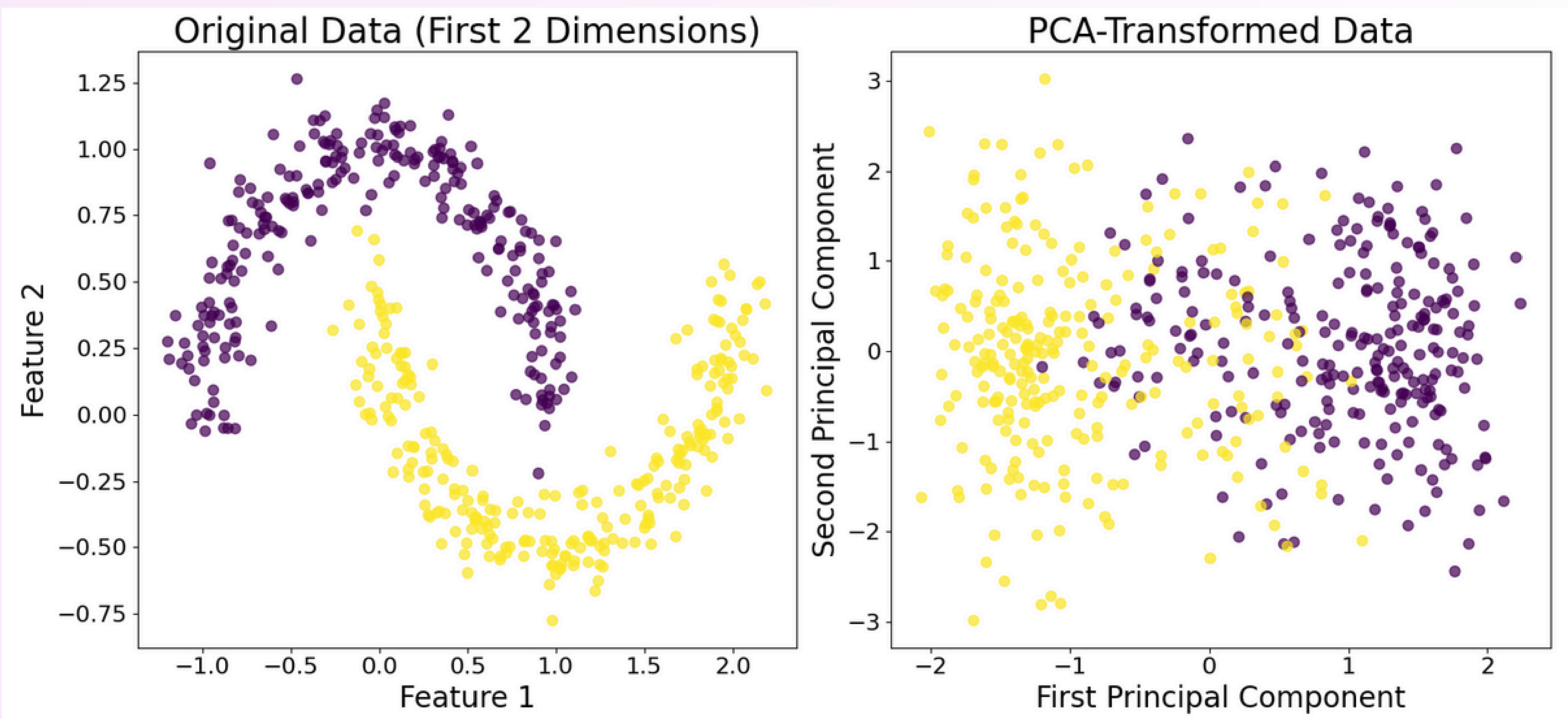



Image Source

PCA finds orthogonal directions (principal components) in the feature space that capture the maximum variance in the data. It's particularly useful for visualization, noise reduction, and feature extraction.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler

# Generate complex dataset
def generate_data(n_samples=500):
    # Generate two interleaving moons
    X1, y1 = make_moons(n_samples=n_samples, noise=0.1)

    # Add some random noise dimensions
    noise_dims = np.random.randn(n_samples, 3) * 0.1
    X = np.hstack((X1, noise_dims))

    return X, y1

# Generate the data
X, y = generate_data(n_samples=500)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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



# Linear Discriminant Analysis (LDA)

LDA is a supervised dimensionality reduction technique that aims to find a linear combination of features that best separates two or more classes.

**Steps of LDA:**

- Compute the mean vectors for each class
- Calculate the within-class and between-class scatter matrices
- Compute the eigenvectors and eigenvalues of the matrix product of the inverse within-class scatter matrix and the between-class scatter matrix
- Sort eigenvectors by decreasing eigenvalues
- Choose top k eigenvectors as the new feature space
- Project the original data onto the new feature space

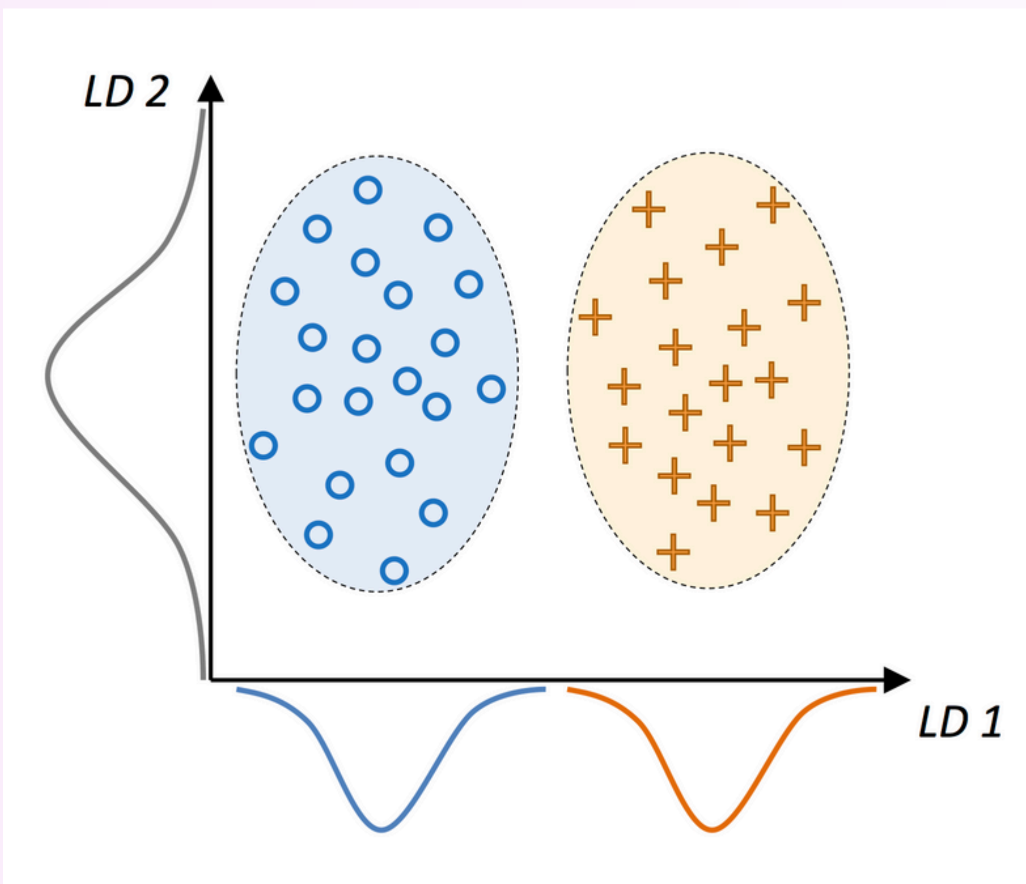
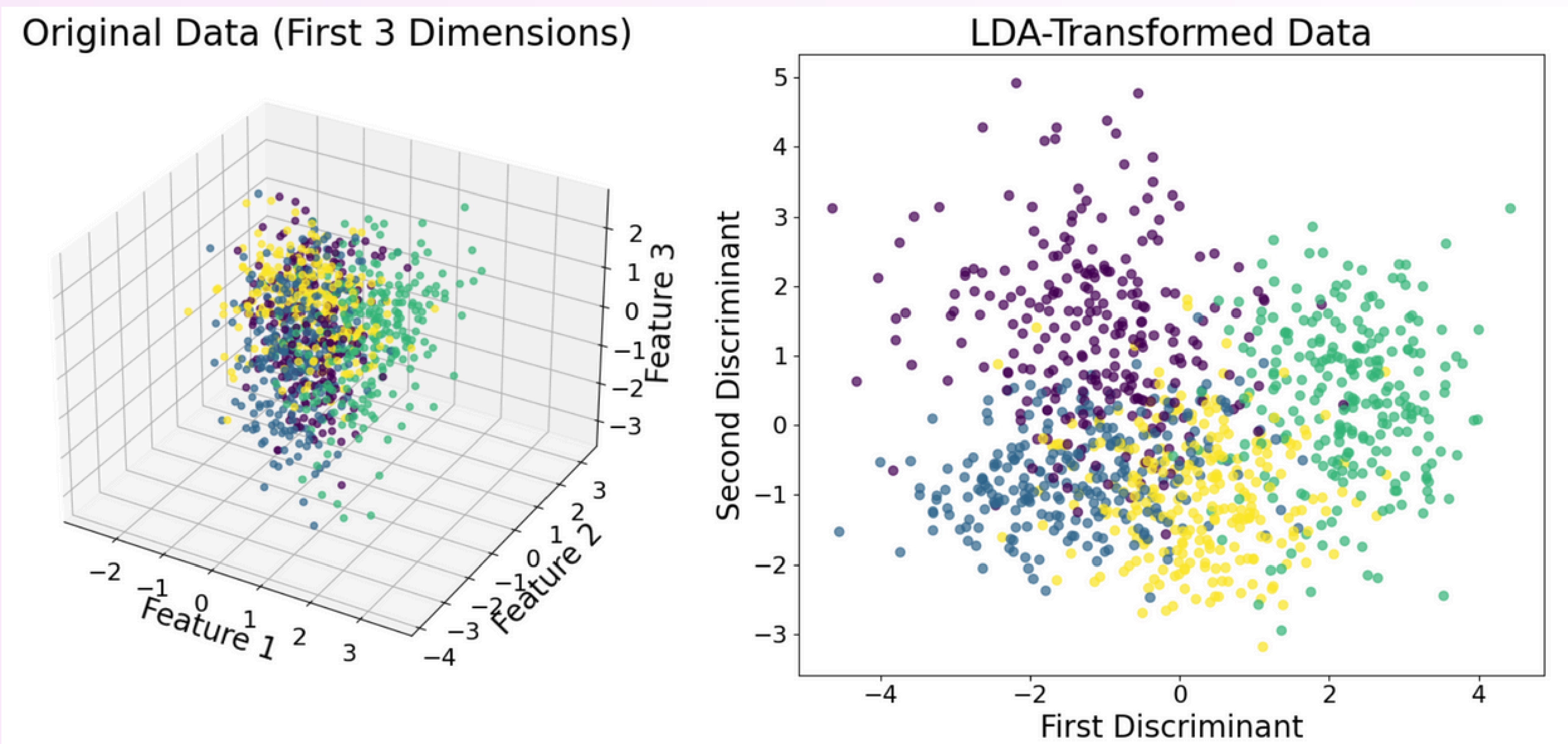



Image Source

LDA maximizes the ratio of between-class variance to within-class variance, making it particularly useful for classification tasks.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler

# Generate complex dataset
def generate_complex_data_lda(n_samples=1000, n_features=20, n_classes=4):
    X, y = make_classification(n_samples=n_samples, n_features=n_features, n_classes=n_classes,
                               n_informative=10, n_redundant=5, n_repeated=3,
                               n_clusters_per_class=2, class_sep=1.5, random_state=42)
    return X, y

# Generate the complex data
X, y = generate_complex_data_lda(n_samples=1000)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)
```



# Kernel Principal Component Analysis (KPCA)

KPCA is a non-linear extension of PCA that uses kernel methods to perform dimensionality reduction in high-dimensional feature spaces.

**Steps of KPCA:**

- Choose a kernel function (e.g., Gaussian, polynomial)
- Compute the kernel matrix
- Center the kernel matrix
- Compute eigenvectors and eigenvalues of the centered kernel matrix Sort eigenvectors by decreasing eigenvalues
- Choose top k eigenvectors as the new feature space
- Project the original data onto the new feature space using the kernel trick

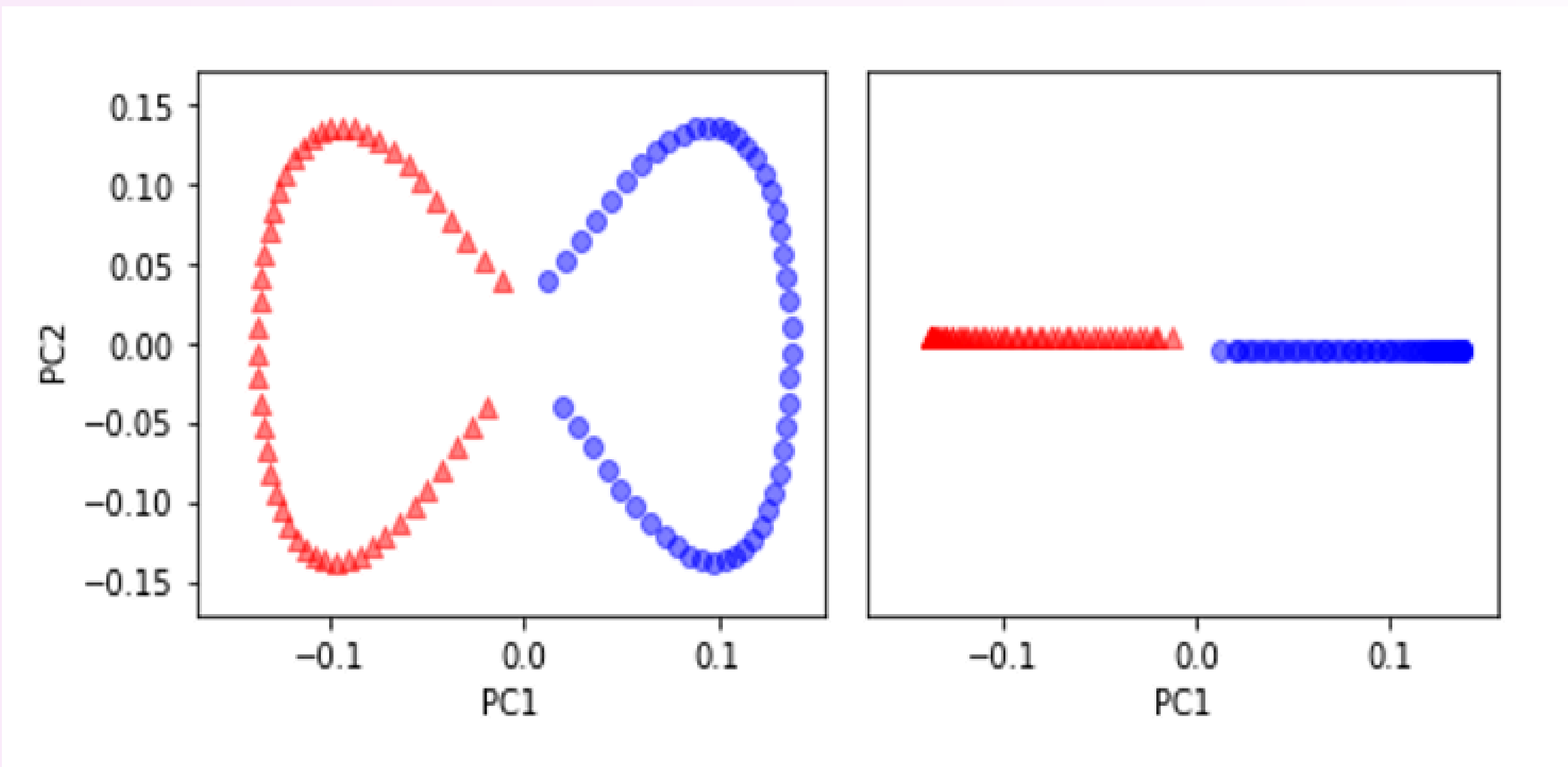
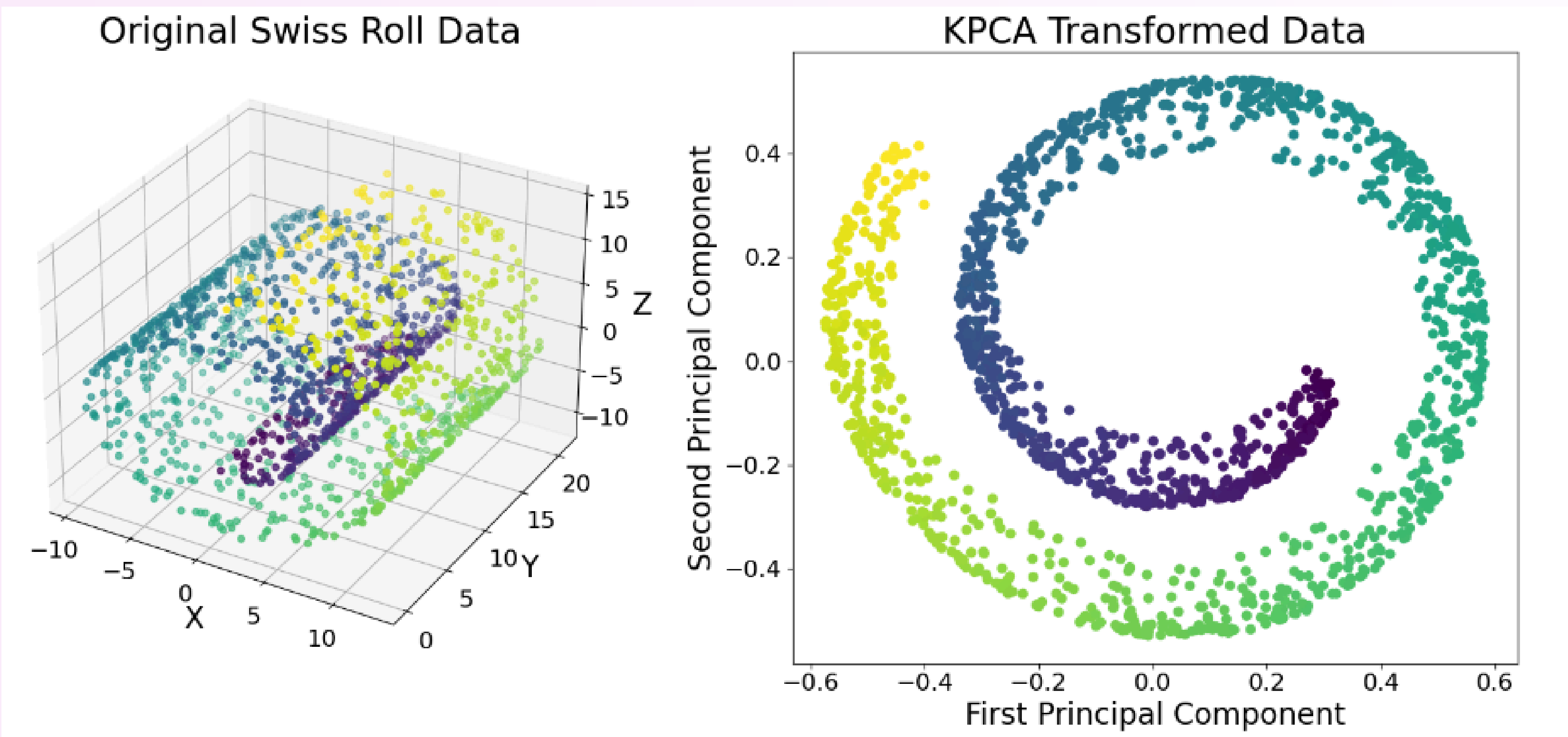



Image Source

KPCA can capture non-linear relationships in the data, making it useful for datasets with complex structures that PCA might miss.

```python
import numpy as np
from sklearn.decomposition import KernelPCA
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from mpl_toolkits.mplot3d import Axes3D

# Generate Swiss roll data
n_samples = 1500
noise = 0.05
X, color = make_swiss_roll(n_samples, noise=noise)

# Perform KPCA
kpca = KernelPCA(n_components=2, kernel='rbf', gamma=0.002)
X_kpca = kpca.fit_transform(X)
```



@shreyapatilu

# Resources

- Machine Learning with Python 3rd Edition, Chapter 5
- Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, Chapter 8
- kernel-pca
- sklearn.decomposition.PCA
- GitHub Full Code