|  |  |
| --- | --- |
| **Ex. No: 1**  **Date: 28.12.23** | **PCA & K-Means Clustering** |

**Name: S Sowmithaa Sri**

**Reg no: 22011101096**

**Aim:**

To implement two unsupervised Machine learning algorithms

1. Principle Component Analysis
2. K-Means Clustering

In mathematical version and using the built-in function from sklearn.

**Dataset description:**

The Iris dataset is available under the scikit-learn library and consists of 150 samples of iris flowers, each belonging to one of three species: setosa, versicolor, or virginica. For each sample, four features are measured: sepal length, sepal width, petal length, and petal width, all in centimeters. The dataset is balanced, with 50 samples from each species, making it suitable for multiclass classification tasks.

**Code:**

1. **PCA- Mathematical Version**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load the Iris dataset

iris = load\_iris()

data = iris.data

# Step 1: Standardize the data

def standardize\_data(data):

return (data - np.mean(data, axis=0)) / np.std(data, axis=0)

# Step 2: Compute the covariance matrix

def compute\_covariance\_matrix(data):

return np.cov(data, rowvar=False)

# Step 3: Compute eigenvectors and eigenvalues

def compute\_eigenvectors\_and\_values(covariance\_matrix):

eigvals, eigvecs = np.linalg.eigh(covariance\_matrix)

return eigvals, eigvecs

# Step 4: Create a feature vector

def create\_feature\_vector(eigvals, eigvecs, target\_variance):

total\_variance = np.sum(eigvals)

variance\_ratio = eigvals / total\_variance

cumulative\_variance\_ratio = np.cumsum(variance\_ratio)

# Choose the number of components to keep based on target\_variance

num\_components = np.argmax(cumulative\_variance\_ratio >= target\_variance) + 1

# Create the feature vector with the top components

feature\_vector = eigvecs[:, :num\_components]

return feature\_vector

# Step 5: Recast the data along the principal components axes

def recast\_data(data, feature\_vector):

return np.dot(data, feature\_vector)

# Step 1: Standardize the data

standardized\_data = standardize\_data(data)

# Step 2: Compute the covariance matrix

cov\_matrix = compute\_covariance\_matrix(standardized\_data)

# Step 3: Compute eigenvectors and eigenvalues

eigvals, eigvecs = compute\_eigenvectors\_and\_values(cov\_matrix)

# Step 4: Create a feature vector

target\_variance = 0.95 # Set your desired target variance

feature\_vector = create\_feature\_vector(eigvals, eigvecs, target\_variance)

# Step 5: Recast the data along the principal components axes

pca\_result\_math = recast\_data(standardized\_data, feature\_vector)

# Visualization

plt.figure(figsize=(10, 5))

# Original Data

plt.subplot(1, 2, 1)

plt.scatter(data[:, 0], data[:, 1], c=iris.target, cmap='viridis', alpha=0.8)

plt.title('Original Data')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

# PCA Result

plt.subplot(1, 2, 2)

plt.scatter(pca\_result\_math[:, 0], pca\_result\_math[:, 1], c=iris.target, cmap='viridis', alpha=0.8)

plt.title('PCA Result (Mathematical Version)')

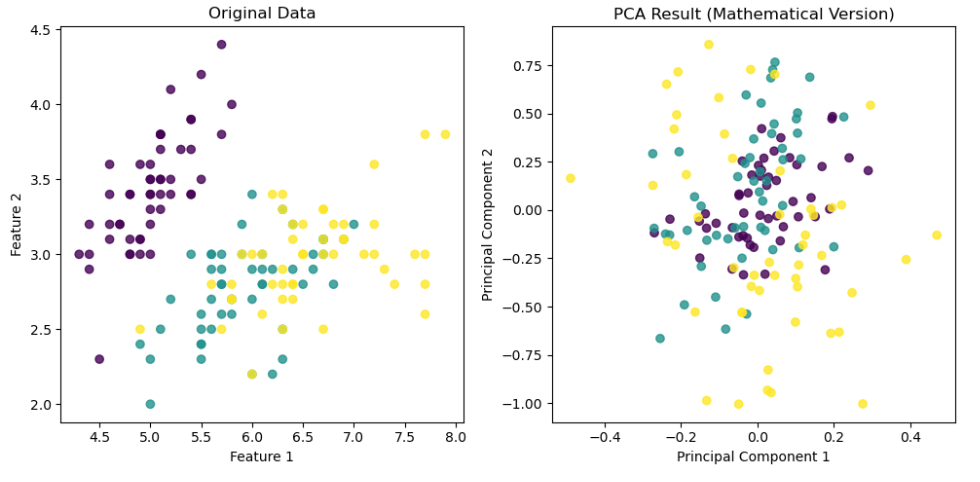
plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.tight\_layout()

plt.show()

**Result:**



**Code:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.datasets import load\_iris

# Load the Iris dataset

iris = load\_iris()

data = iris.data

# Step 1: Standardize the data (not explicitly needed in Scikit-Learn PCA)

# Step 2: No need to compute the covariance matrix separately

# Step 3: Compute eigenvectors and eigenvalues using Scikit-Learn PCA

pca\_sklearn = PCA()

pca\_sklearn.fit(data)

# Step 4: No need to create a feature vector explicitly (explained\_variance\_ratio\_ is available)

# Step 5: Recast the data along the principal components axes

pca\_result\_sklearn = pca\_sklearn.transform(data)

# Visualization

plt.figure(figsize=(10, 5))

# Original Data

plt.subplot(1, 2, 1)

plt.scatter(data[:, 0], data[:, 1], c=iris.target, cmap='viridis', alpha=0.8)

plt.title('Original Data')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

# PCA Result

plt.subplot(1, 2, 2)

plt.scatter(pca\_result\_sklearn[:, 0], pca\_result\_sklearn[:, 1], c=iris.target, cmap='viridis', alpha=0.8)

plt.title('PCA Result (Scikit-Learn Version)')

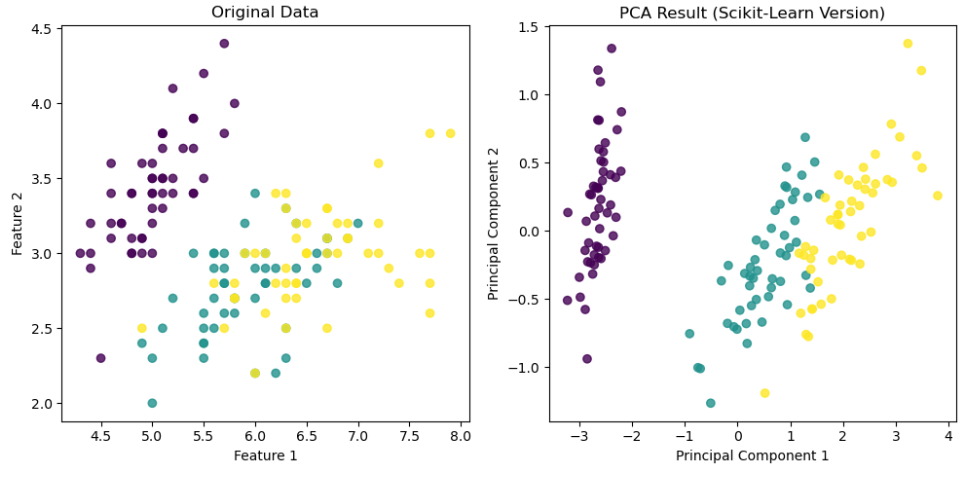
plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.tight\_layout()

plt.show()

**Result:**



**Code:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

def euclidean\_distance(point1, point2):

return np.sqrt(np.sum((point1 - point2)\*\*2))

def initialize\_centroids(data, k):

return data[np.random.choice(range(len(data)), k, replace=False)]

def assign\_to\_clusters(data, centroids):

distances = np.array([[euclidean\_distance(point, centroid) for centroid in centroids] for point in data])

return np.argmin(distances, axis=1)

def update\_centroids(data, labels, k):

new\_centroids = np.zeros((k, data.shape[1]))

for i in range(k):

cluster\_points = data[labels == i]

if len(cluster\_points) > 0:

new\_centroids[i] = np.mean(cluster\_points, axis=0)

return new\_centroids

def kmeans\_with\_visualization(data, k, max\_iters=100):

centroids = initialize\_centroids(data, k)

for iteration in range(max\_iters):

labels = assign\_to\_clusters(data, centroids)

new\_centroids = update\_centroids(data, labels, k)

# Visualize clusters for the current iteration

plt.figure(figsize=(4, 3))

plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', edgecolor='k', s=25)

plt.scatter(new\_centroids[:, 0], new\_centroids[:, 1], marker='X', c='red', s=200, label=f'Centroids (Iteration {iteration + 1})')

plt.title(f'K-means Clustering (Mathematical Version) - Iteration {iteration + 1}')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()

if np.all(centroids == new\_centroids):

break

centroids = new\_centroids

return centroids, labels

# Load the Iris dataset

iris = load\_iris()

data = iris.data

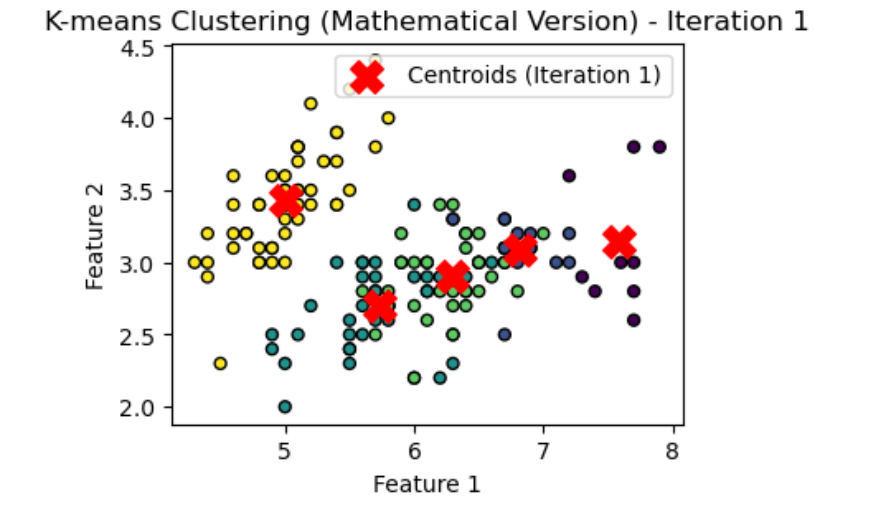
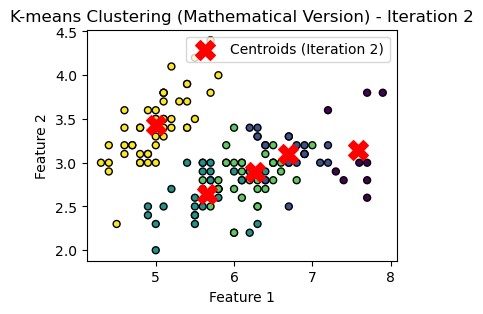
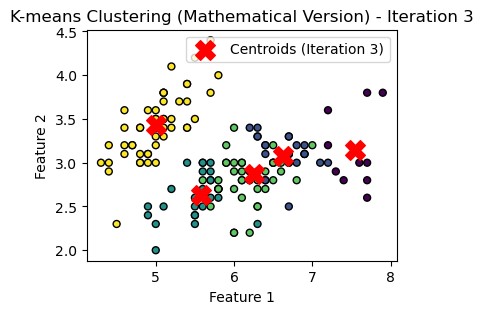
# Choose the number of clusters (k)

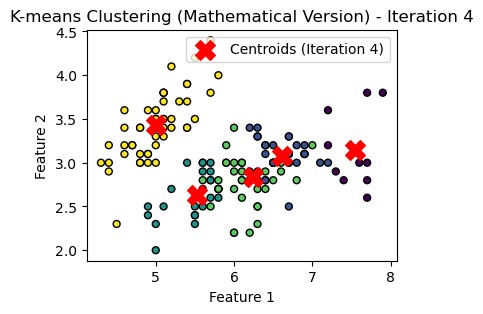
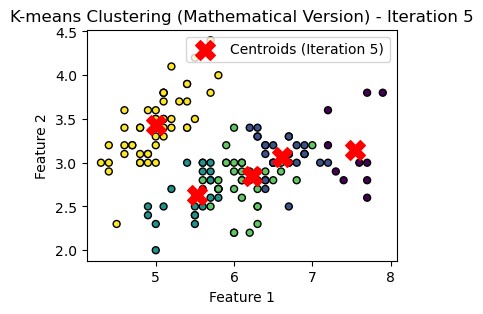
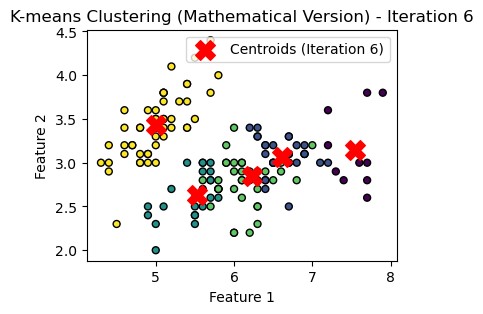
k = 5

# Apply K-means clustering with iterative visualization

centroids\_math, labels\_math = kmeans\_with\_visualization(data, k)

**Result:**

**Code:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import load\_iris

# Load the Iris dataset

iris = load\_iris()

data = iris.data

# Choose the number of clusters (k)

k = 5

# Create KMeans instance

kmeans = KMeans(n\_clusters=k, random\_state=42)

# Fit the model to the data with iterative visualization

for iteration in range(10): # Adjust the number of iterations as needed

kmeans.n\_init = 1 # Ensure only one initialization for clarity

kmeans.max\_iter = iteration + 1

kmeans.fit(data)

# Get cluster labels and centroids

labels\_sklearn = kmeans.labels\_

centroids\_sklearn = kmeans.cluster\_centers\_

# Visualize clusters for the current iteration

plt.figure(figsize=(4, 3))

plt.scatter(data[:, 0], data[:, 1], c=labels\_sklearn, cmap='viridis', edgecolor='k', s=25)

plt.scatter(centroids\_sklearn[:, 0], centroids\_sklearn[:, 1], marker='X', c='red', s=100, label=f'Centroids (Iteration {iteration + 1})')

plt.title(f'K-means Clustering (Scikit-Learn Version) - Iteration {iteration + 1}')

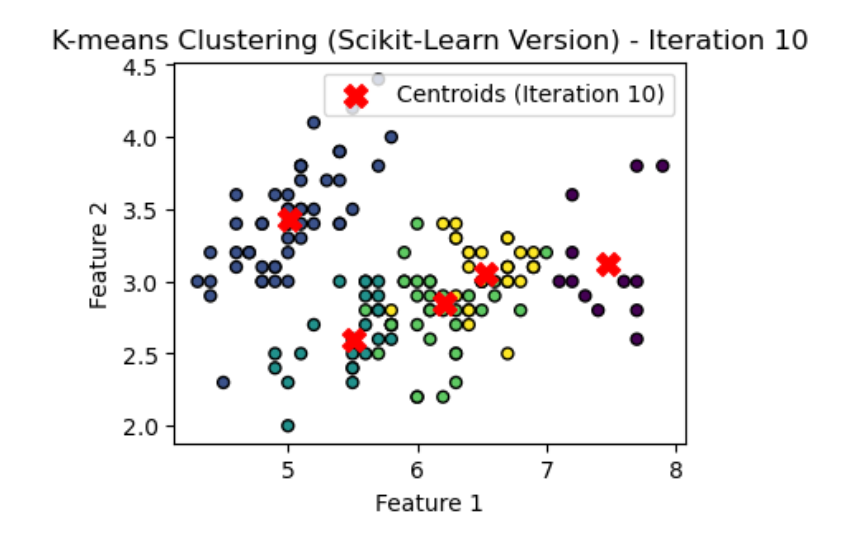
plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()

**Result:**



**Result Description:**

Mathematical PCA Result: In the mathematical PCA on the Iris dataset, we standardized the features and computed the covariance matrix. Eigenvectors and eigenvalues were derived, and a feature vector was created to achieve 95% variance. The data was then transformed along principal components, revealing clear species separation in the visualization.

Scikit-Learn PCA Result: In the Scikit-Learn PCA on Iris, features were loaded and automatically standardized, with PCA handling covariance matrix computation and data transformation. The resulting principal components effectively captured dataset variance. Visualization showcased distinct clusters corresponding to Iris species, demonstrating PCA's efficiency in dimensionality reduction.

Mathematical K-means Result: For the mathematical K-means on Iris, features were standardized, and centroids were iteratively updated based on mean assigned points. The algorithm converged to distinct clusters, visually partitioning the dataset into groups, showcasing its ability to uncover natural patterns in iris flowers.

Scikit-Learn K-means Result: The Scikit-Learn K-means on Iris simplified the process, with automatic assignment of points to clusters and centroid updates. Visualization demonstrated cluster convergence, revealing clear separation of Iris species. Scikit-Learn's K-means provided an accessible and efficient solution for clustering analysis in machine learning tasks.

**Conclusion:**

Thus the two unsupervised ML algorithms-PCA and K-means Clustering are successfully implemented and executed.