| **CO2** | **Apply pre-processing techniques and explore the using visualization techniques** |
| --- | --- |
| Task 3: | Apply PCA and LDA algorithm to select the appropriate features from the given data set.  Platform: Orange, Google co-lab, Anaconda navigator, Language: Python |

**Use Case: Classification on Iris Species using Principal Component Analysis (PCA)**

**Objective:**

Many research attempts aim to classify Iris Species through behavior-based dimensionality reduction using Principal Component Analysis (PCA) to enhance its classification.

**Background:**

The Iris Dataset has been collecting diverse sets of features of Iris Species. The platform recognizes the need to understand behavior more deeply to classify it from other species.

**Algorithm: Classification on Iris Species through Behavior-Based Dimensionality Reduction**

1.1 Collect data on Iris Species.

1.2 Preprocess the data:

- Handle missing values.

- Address outliers.

2.1 Apply Principal Component Analysis (PCA):

- Mathematically represented as Xpca = X . eigenvectors.

2.2 Determine the optimal number of principal components:

- Calculate the explained variance ratio.

- Choose the number of components based on objectives.

3.1 Employ KMeans clustering:

- Mathematically represented as assigning data points Xi to clusters Ck based on the minimization of the within-cluster sum of squares.

3.2 Choose an appropriate number of clusters:

- Utilize methods like the elbow method to determine the optimal number of clusters.

4.1 Examine segment characteristics:

- Analyze the original features for each customer segment.

4.2 Interpret principal components:

- Understand the impact of principal components on segment characteristics.

5.1 Summarize findings and insights.

**Program:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.decomposition import PCA

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Load Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Apply PCA for dimensionality reduction

pca = PCA(n\_components=2)

X\_train\_pca = pca.fit\_transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

# Plot the PCA-transformed training data

plt.figure(figsize=(8, 6))

colors = ['red', 'green', 'blue']

for i in range(3):

plt.scatter(X\_train\_pca[y\_train == i, 0], X\_train\_pca[y\_train == i, 1], label=f'Class {i}', color=colors[i])

plt.title('PCA-transformed Training Data')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend()

plt.show()

# Train a classifier on the PCA-transformed data

knn\_classifier = KNeighborsClassifier(n\_neighbors=3)

knn\_classifier.fit(X\_train\_pca, y\_train)

# Make predictions on the PCA-transformed test data

y\_pred = knn\_classifier.predict(X\_test\_pca)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy on the test set: {accuracy:.2f}')

# Plot the PCA-transformed test data with predictions

plt.figure(figsize=(8, 6))

for i in range(3):

plt.scatter(X\_test\_pca[y\_test == i, 0], X\_test\_pca[y\_test == i, 1], label=f'Actual Class {i}', color=colors[i], alpha=0.7)

plt.scatter(X\_test\_pca[:, 0], X\_test\_pca[:, 1], c=y\_pred, marker='x', cmap='viridis', label='Predictions')

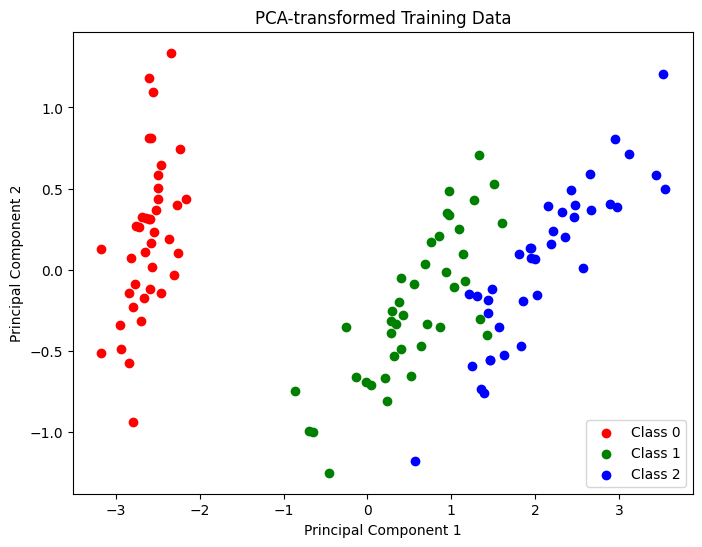
plt.title('PCA-transformed Test Data with Predictions')

plt.xlabel('Principal Component 1')

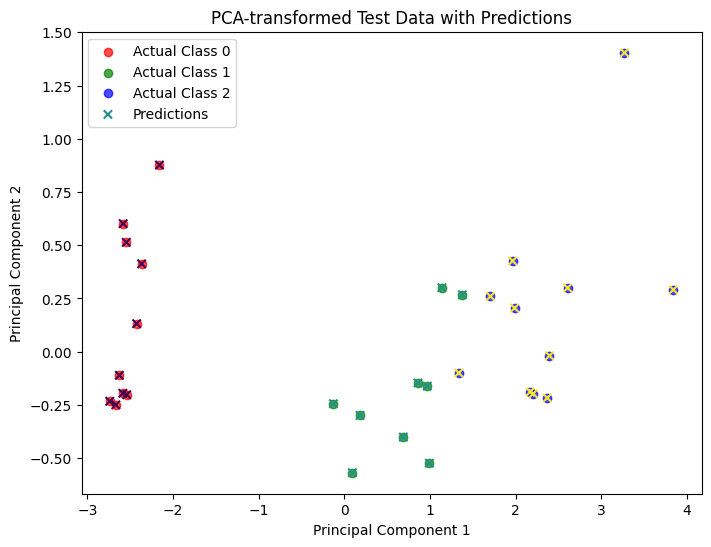
plt.ylabel('Principal Component 2')

plt.legend()

plt.show()



Accuracy on the test set: 1.00



**Use Case: Classification on Iris Species using Linear Discriminant Analysis (LDA)**

**Objective:**

Many research attempts aim to classify Iris Species through behavior-based dimensionality reduction using Linear Discriminant Analysis (LDA) to enhance its classification.

**Algorithm: Classification on Iris Species through Behavior-Based Dimensionality Reduction**

1. Input:

- Iris data with features.

2. Preprocessing the data

3. Linear Discriminant Analysis (LDA) on Training Set:

a. Initialize an `LinearDiscriminantAnalysis` object with the desired number of components (2 in this case).

b. Fit and transform the training set using LDA.

4.Visualization (Training Set):

- Plot the customer segments in 2D space based on the first two components of LDA for the training set.

5. Fit LDA Model on Training Data

- Fit the LDA model on the entire training data.

6. Transform Test Set Using Trained LDA Model:

- Transform the test set using the trained LDA model.

7. Output LDA Results:

a. Print the coefficients and intercept of the LDA model.

b. Plot the customer segments in 2D space based on the first two components of LDA for the test set.

**Program:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Load Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Apply Linear Discriminant Analysis (LDA) for dimensionality reduction

lda = LinearDiscriminantAnalysis(n\_components=2)

X\_train\_lda = lda.fit\_transform(X\_train, y\_train)

X\_test\_lda = lda.transform(X\_test)

# Plot the LDA-transformed training data

plt.figure(figsize=(8, 6))

colors = ['red', 'green', 'blue']

for i in range(3):

    plt.scatter(X\_train\_lda[y\_train == i, 0], X\_train\_lda[y\_train == i, 1], label=f'Class {i}', color=colors[i])

plt.title('LDA-transformed Training Data')

plt.xlabel('LDA Component 1')

plt.ylabel('LDA Component 2')

plt.legend()

plt.show()

# Train a classifier on the LDA-transformed data

knn\_classifier = KNeighborsClassifier(n\_neighbors=3)

knn\_classifier.fit(X\_train\_lda, y\_train)

# Make predictions on the LDA-transformed test data

y\_pred = knn\_classifier.predict(X\_test\_lda)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy on the test set: {accuracy:.2f}')

# Plot the LDA-transformed test data with predictions

plt.figure(figsize=(8, 6))

for i in range(3):

    plt.scatter(X\_test\_lda[y\_test == i, 0], X\_test\_lda[y\_test == i, 1], label=f'Actual Class {i}', color=colors[i], alpha=0.7)

plt.scatter(X\_test\_lda[:, 0], X\_test\_lda[:, 1], c=y\_pred, marker='x', cmap='viridis', label='Predictions')

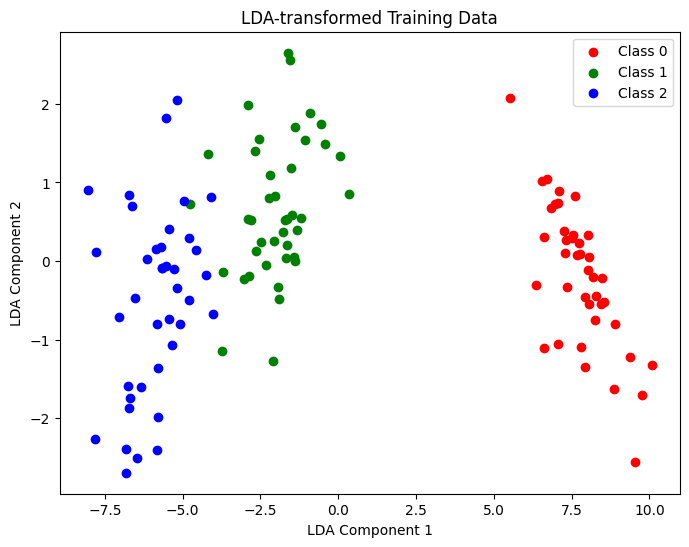
plt.title('LDA-transformed Test Data with Predictions')

plt.xlabel('LDA Component 1')

plt.ylabel('LDA Component 2')

plt.legend()

plt.show()



Accuracy on the test set: 1.00

