Unsupervised Machine Learning

Assignment 2

SAI KUMAR MURARSHETTI

LEWIS ID: L30079224

1. Technical Description:

Principal Component Analysis (PCA): It is an useful statistical and machine learning technique that reduces dataset dimensionality while keeping data variance. It performs this by changing the original variables into a new set of transverse variables known as principle components. These principle components are linear combinations of the original variables sorted by the amount of variance they explain in the data. PCA works to identify the directions (principal components) in which the data shifts the most. It achieves this by calculating the eigenvectors of the data's covariance matrix and projecting the data on these eigenvectors.

Linear Discriminant Analysis (LDA): It is an unsupervised technique that decreases dimensionality and improves class separation. It performs this by identifying the linear combinations of variables (discriminants) that most accurately identify the classes in the data. With PCA, which aims to maximize variance, LDA is focused on maximizing between-class variation while decreasing within-class scatter.

```
#Importing numpy,pandas,matplotlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import files
from mpl_toolkits.mplot3d import Axes3D
```

```
# Loading the dataset
data = pd.read_csv('Traffic.csv')
```

```
# Performing PCA on the count of columns (CarCount, BikeCount, BusCount, TruckCount)
X = data[['CarCount', 'BikeCount', 'BusCount', 'TruckCount']]
# 'Traffic Situation' is the target variable
y = data['Traffic Situation']
```

2. Algorithm Design:

- The covariance matrix for the standard data is determined. The covariance matrix contains information about the relationships between variables.
- The eigenvectors and eigenvalues of the covariance matrix are determined. Eigenvectors denote the directions of maximal variation while eigenvalues define the quantity of variance in these directions.

```
# Computing covariance matrix for PCA
cov_matrix = np.cov(X, rowvar=False)
# Computing eigenvectors and eigenvalues for PCA
eig_values_pca, eig_vectors_pca = np.linalg.eig(cov_matrix)
```

The eigenvectors corresponding to the largest eigenvalues (principal components) are selected. These important factors explain a significant portion of the data variance.

- Sorted_indices_pca = np.argsort(eig_values_pca)[::-1]. This line determines the numbers that are used for sorting the eigenvalues in descending order. np.argsort returns the index that was used to sort the array, and [::-1] reverses the order to sort in descending order. These indices are kept in sorted_indices_pca.
- Eig_values_pca = eig_values_pca[sorted_indices_pca] This line sorts the eigenvalues based on the sorted indices gained in the following phases. So, follow this line, eig_values_pca will have the eigenvalues in descending order.
- eig_vectors_pca is equal to[:, ordered_indices_pca]: This line arranges the eigenvectors correctly. Because each column in eig_vectors_pca represents an eigenvector, it rearranges them to correspond to the sorted eigenvalues. The [:, sorted_indices_pca] component ensures that each column is reordered accordingly to the sorted_indices_pca.

```
# Sorting the eigenvalues and eigenvectors for PCA
sorted_indices_pca = np.argsort(eig_values_pca)[::-1]
eig_values_pca = eig_values_pca[sorted_indices_pca]
eig_vectors_pca = eig_vectors_pca[:, sorted_indices_pca]
```

PCA_1d = np.dot(X, eig_vectors_pca[:,:1]): calculates the projection of the original data, X, into the first principal component. eig_vectors_pca[:,:1] selects only the first eigenvector (which corresponds to the direction of maximum variance), and np.dot(X, eig_vectors_pca[:,:1]) computes the dot product of the original data matrix X with this eigenvector.

```
# Select top 1, 2, and 3 principal components for PCA
pca_1d = np.dot(X, eig_vectors_pca[:, :1])
pca_2d = np.dot(X, eig_vectors_pca[:, :2])
pca_3d = np.dot(X, eig_vectors_pca[:, :3])
```

This NumPy function returns the array's unique elements while maintaining their order. In this context, y is taken to represent an array or list of class labels for each sample in a binary classification issue. For instance, if y has the labels [0, 1, 1, 0, 1], then np.unique(y) returns [0, 1], indicating that the dataset contains two unique class labels. In a binary classification situation, it is usual to have two class labels, which are typically represented as 0 and 1, with each label representing one of the two classes being predicted.

```
# Assuming binary classification for simplicity
class_labels = np.unique(y)
```

Generate the mean vector for each class in the dataset.

Determine the within-class scatter matrix, which represents the variability within each class.

Evaluate the between-class scatter matrix, which shows the variation between classes.

```
# Calculating class means for LDA
class_means = [np.mean(X[y == label], axis=0) for label in class_labels]
# Computing within-class scatter matrix for LDA
within_class_scatter_matrix = np.sum([np.cov(X[y == label], rowvar=False) for label in class_labels], axis=0)
# Computing between-class scatter matrix for LDA
overall_mean = np.mean(X, axis=0)
between_class_scatter_matrix = np.sum([np.outer(class_means[i] - overall_mean, class_means[i] - overall_mean) for i in range(len(class_labels))], axis=
```

Figure out the eigenvectors and eigenvalues for the matrix Sw^-1 * Sb, where Sw is the within-class scatter matrix and Sb is the between-class scatter matrix.

For creating the projection matrix, select the eigenvectors that correspond to the biggest eigenvalues.

```
# Computing eigenvectors and eigenvalues of (inverse of within-class scatter matrix) times (between-class scatter matrix)
eig_values_lda, eig_vectors_lda = np.linalg.eig(np.linalg.inv(within_class_scatter_matrix).dot(between_class_scatter_matrix))
# Sorting eigenvalues and eigenvectors for LDA
sorted_indices_lda = np.argsort(eig_values_lda)[::-1]
eig_values_lda = eig_values_lda[sorted_indices_lda]
eig_vectors_lda = eig_vectors_lda[:, sorted_indices_lda]
# Selecting 1st discriminant component for LDA
lda_1d = np.dot(X, eig_vectors_lda[:, :1])
```

3. Results of the Algorithms:

Performing Principal Component Analysis (PCA) on a dataset. The PCA observations are shown in three subplots that indicate the data's projection into 1D, 2D, and 3D spaces.

```
# Ploting results for PCA
fig, axs = plt.subplots(1, 3, figsize=(20, 5))
```

- In the first subplot:

The data is based on the first main component (1D). Each point's position along the x-axis represents its projection into this one dimension. Because it is a one-dimensional projection, the y-values are set to zero. This graphic shows how the data is distributed when reduced to a single dimension.

- In the second subplot:

The data is projected on the first two main components (2D). Each point's position on the plot corresponds to its projection into these two dimensions. This 2D scatter plot visualizes relationships and patterns that may exist in the data in a reduced dimensional context.

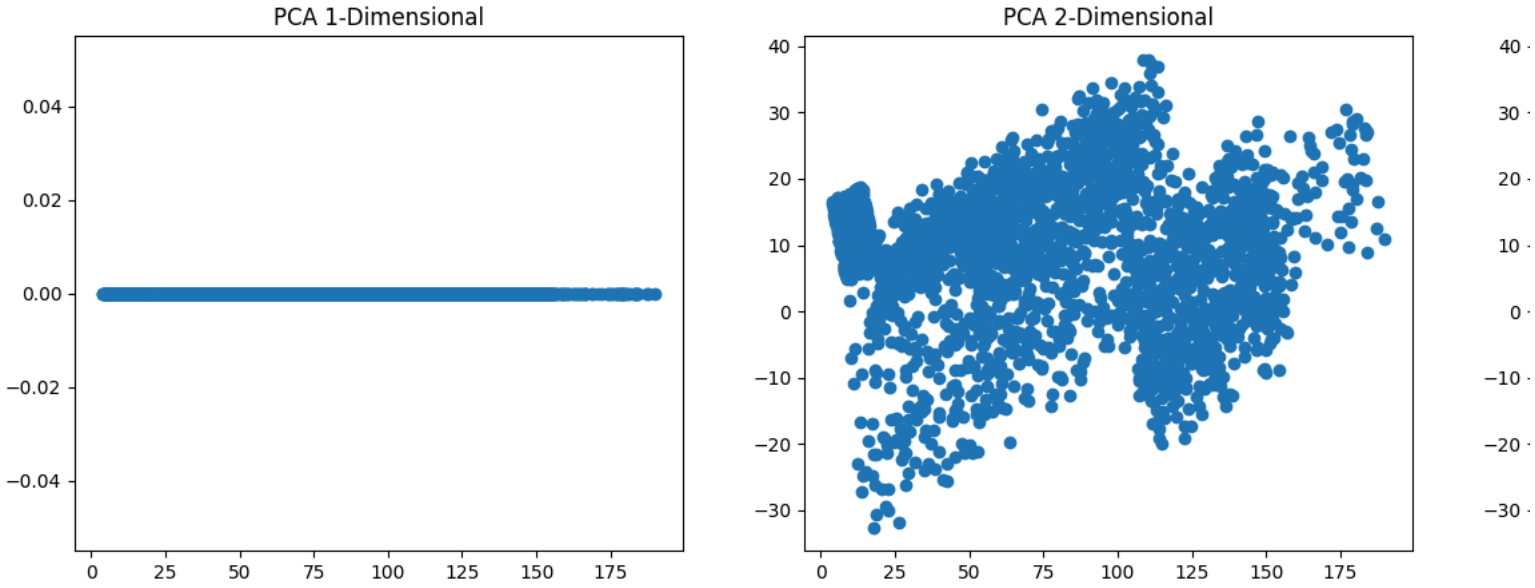
- In the third subplot:

The data is projected on the first three main parts (3D). Each point's position in this 3D space represents its projection into these three dimensions. When data is reduced to three dimensions, it becomes easy to understand its structure and distribution.

Overall, these graphs make it easier to determine the efficiency of PCA in lowering dataset dimensionality while keeping significant information. They aid in spotting clusters, trends, and outliers within data in a reduced dimensional space.

```
# PCA plots
axs[0].scatter(pca_1d, np.zeros_like(pca_1d))
axs[0].set_title('PCA 1-Dimensional')
axs[1].scatter(pca_2d[:, 0], pca_2d[:, 1])
axs[1].set_title('PCA 2-Dimensional')
axs[2].scatter(pca_3d[:, 0], pca_3d[:, 1])
axs[2].set_title('PCA 3-Dimensional')
```

```
Text(0.5, 1.0, 'PCA 3-Dimensional')
```



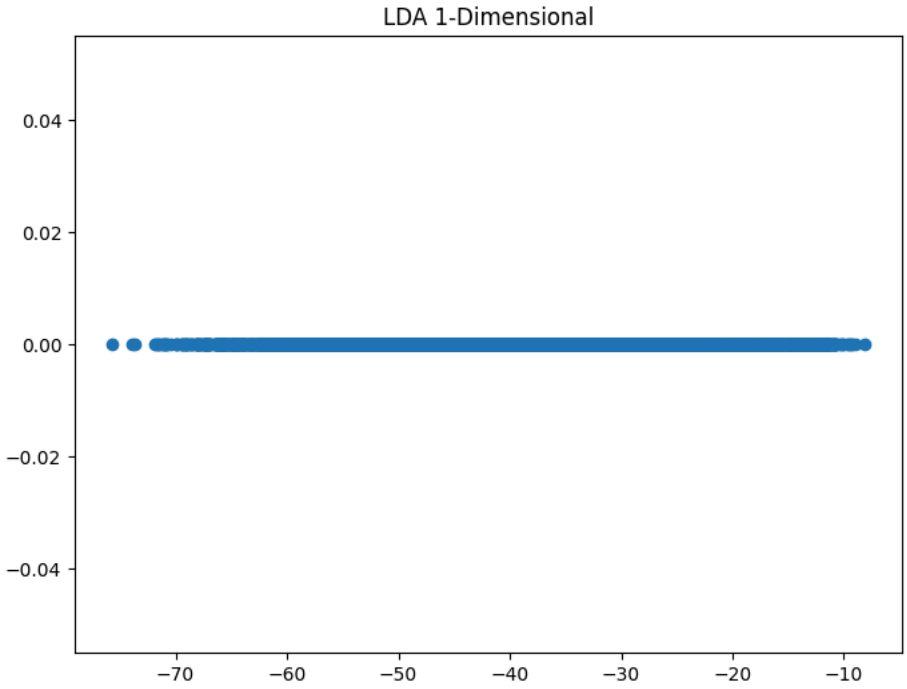
The results include reduced-dimensional representations of data generated by PCA and LDA. These representations are evaluated using their capacity to keep important information from the original data while lowering its dimensionality.

4. Analysis of the Results:

The analysis comprises determining if the reduced-dimensional representations generated by PCA and LDA complete the requirements. This involves evaluating the algorithms' performance in terms of data structure integrity and class separation.

The objective of this plot is to visualize the data after using LDA and projecting it into a single classification axis. It improves in understanding the way data can be separated and discriminated in a smaller 1-dimensional space using **class** labels or groupings.

```
# Ploting result for LDA
plt.figure(figsize=(8, 6))
plt.scatter(lda_1d, np.zeros_like(lda_1d))
plt.title('LDA 1-Dimensional')
plt.show()
```



5. **Conclusion:**

- Overall, these visualizations give information about the way the dimensionality reduction approaches (PCA and LDA) represent the underlying structure of the data. They help in understanding the correlations, patterns and separation of data in reduced-dimensional spaces, which is useful in tasks such as classification, clustering and feature selection.
- PCA and LDA are efficient dimensionality reduction algorithms that maintain data's important structure and class separability. By carefully analyzing the output of these algorithms I understood useful insights into the underlying structure of the data and improve the