LDA Projection

Table of contents

Import Notebook Funcs

```
from notebookfuncs import *
```

Source: Understanding Linear Discriminant Analysis (LDA) in Python Programming

Linear Discriminant Analysis (LDA) is a powerful technique in the field of pattern recognition, machine learning, and statistics. It is used for dimensionality reduction, feature extraction, and classification. LDA is particularly useful when you have a labeled dataset and want to find the linear combinations of features that best separate different classes. In this article, we'll delve into the principles of LDA and demonstrate its implementation in Python.

LDA is a supervised dimensionality reduction technique. Unlike Principal Component Analysis (PCA), which focuses on capturing the maximum variance in the data, LDA emphasizes class separability. It aims to project the data points onto a lower-dimensional subspace while maximizing the distance between the class means and minimizing the spread within each class.

The main steps involved in LDA are as follows

1. Compute the mean vectors for each class

Calculate the mean vector for each class by averaging the feature vectors of the data points belonging to that class.

2. Compute the scatter matrices

Within-class scatter matrix (S_w): It measures the spread of data within each class and is calculated as the sum of the covariance matrices for individual classes.

Between-class scatter matrix (S_B): It quantifies the separation between classes and is computed as a weighted sum of the covariance matrices between the classes.

3. Eigendecomposition of $S_W^{-1} * S_B$

To find the linear discriminants, we perform an eigendecomposition of the matrix $S_W^{-1} * S_B$. The resulting eigenvalues and eigenvectors help us determine the optimal directions in the new feature space.

4. Select the top k eigenvectors

Sort the eigenvalues in descending order and choose the top k eigenvectors corresponding to the k largest eigenvalues. These eigenvectors will define the new subspace.

5. Create a projection matrix

Form a projection matrix by stacking the selected eigenvectors as columns. This matrix will be used to project the data onto the new subspace.

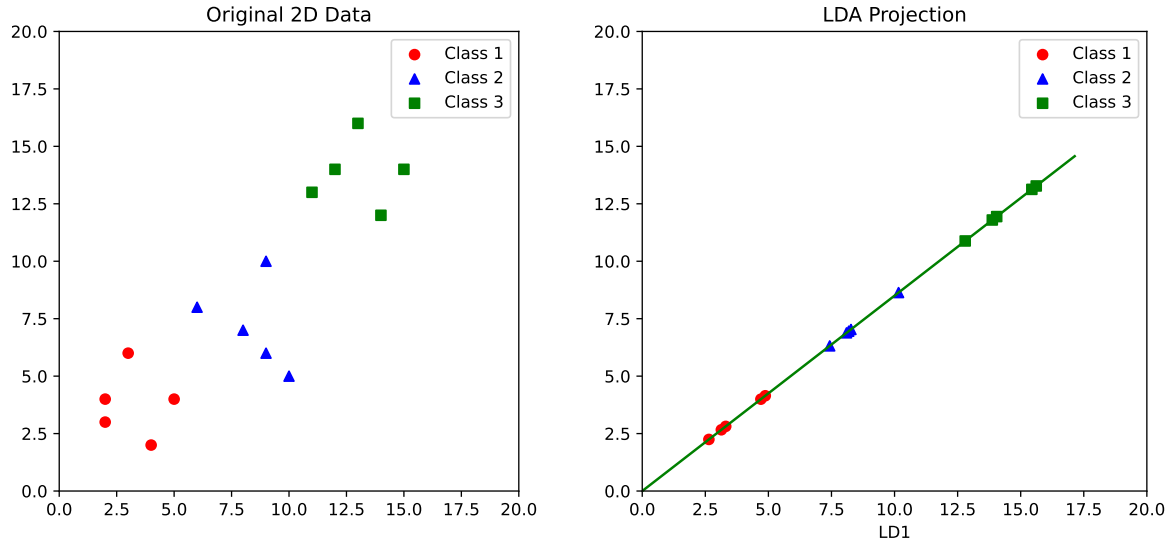
6. Project data onto the new subspace:

Finally, the data points are projected onto the new subspace created by the projection matrix.

Let's take a practical look at implementing Linear Discriminant Analysis in Python. We'll use the scikit-learn library, which provides a convenient interface for LDA.

```
import numpy as np
import matplotlib.pyplot as plt

# Data for three classes
x1 = np.array([[4, 2], [2, 4], [2, 3], [3, 6], [5, 4]])
x2 = np.array([[9, 10], [6, 8], [9, 6], [8, 7], [10, 5]])
x3 = np.array([[14, 12], [12, 14], [11, 13], [13, 16], [15, 14]])

# Scatter plot of the three classes (Original Data)
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(x1[:, 0], x1[:, 1], color='red', marker='o', label='Class 1')
plt.scatter(x2[:, 0], x2[:, 1], color='blue', marker='^', label='Class 2')
plt.scatter(x3[:, 0], x3[:, 1], color='green', marker='s', label='Class 3')
plt.title('Original 2D Data')
plt.xlim(0, 20)
plt.ylim(0, 20)
plt.gca().set_aspect('equal', adjustable='box')
plt.legend()

# Class means
mu1 = np.mean(x1, axis=0)
mu2 = np.mean(x2, axis=0)
mu3 = np.mean(x3, axis=0)

# Covariance matrices
```

```
s1 = np.cov(x1, rowvar=False)
s2 = np.cov(x2, rowvar=False)
s3 = np.cov(x3, rowvar=False)

# Within-class scatter matrix
sw = s1 + s2 + s3

# Between-class scatter matrix
sb = np.outer((mu1 - mu2), (mu1 - mu2)) + np.outer((mu1 - mu3), (mu1 - mu3)) +
 ↪  np.outer((mu2 - mu3), (mu2 - mu3))

# Compute LDA projection
invsw_by_sb = np.dot(np.linalg.inv(sw), sb)

# Get projection vectors
eigenvalues, eigenvectors = np.linalg.eig(invsw_by_sb)

# First projection vector
w = eigenvectors[:, 0]

# Project data to 1D
y1 = np.dot(x1, w)
y2 = np.dot(x2, w)
y3 = np.dot(x3, w)

# Plot the projected data in the original 2D space showing projection direction
theta = np.arctan2(w[1], w[0])
z1x, z1y = np.multiply(np.cos(theta), y1), np.multiply(np.sin(theta), y1)
z2x, z2y = np.multiply(np.cos(theta), y2), np.multiply(np.sin(theta), y2)
z3x, z3y = np.multiply(np.cos(theta), y3), np.multiply(np.sin(theta), y3)
z4x, z4y = np.multiply(np.cos(theta), max([max(y1), max(y2), max(y3)]) + 2),
 ↪  np.multiply(np.sin(theta), max([max(y1), max(y2), max(y3)]) + 2)

plt.subplot(1, 2, 2)
plt.plot([0, z4x], [0, z4y], 'g-')
plt.scatter(z1x, z1y, color='red', marker='o', label='Class 1')
plt.scatter(z2x, z2y, color='blue', marker='^', label='Class 2')
plt.scatter(z3x, z3y, color='green', marker='s', label='Class 3')
plt.title('LDA Projection')
plt.xlabel('LD1')
plt.xlim(0, 20)
plt.ylim(0, 20)
plt.legend()
plt.show()
```



References: https://www.sjsu.edu/faculty/guangliang.chen/Math253S20/lec11lda.pdf

```
allDone();
```

```
<IPython.lib.display.Audio object>
```