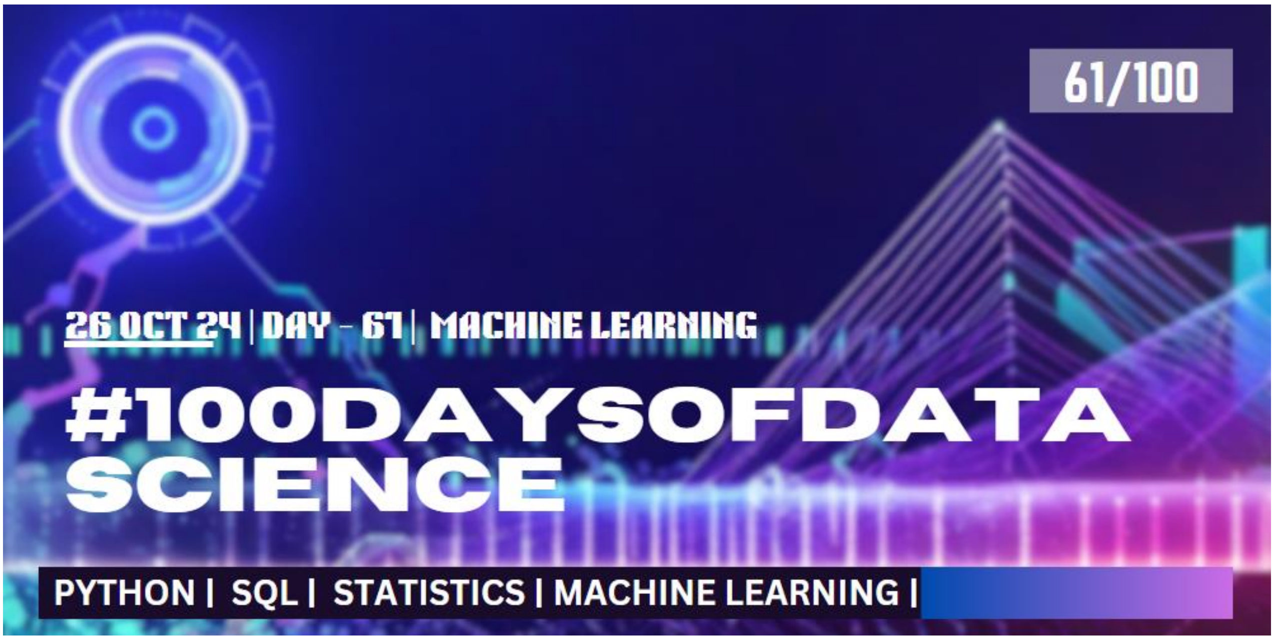


Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA): Maximizing Class Separation in Supervised Learning
Linear Discriminant Analysis (LDA) is a popular technique in machine learning and statistics used for both classification and dimensionality reduction. Unlike PCA, which is unsupervised and focuses on variance, LDA is a supervised method that emphasizes maximizing the separation between different classes. LDA is particularly useful in scenarios where you need to classify data points while reducing the number of features, all while retaining critical class-related information.

Key Features of LDA:

1. **Dimensionality Reduction for Classification:**
    - LDA reduces the number of dimensions by projecting the data onto a lower-dimensional space, aiming to preserve class separability. This is particularly useful when working with datasets that have multiple features but relatively few observations.
2. **Class Separation:**
    - Unlike PCA, which preserves variance, LDA explicitly maximizes the distance between class means while minimizing the variance within each class. This ensures that data points from different classes are projected as far apart as possible.
3. **Linear Decision Boundaries:**
    - LDA assumes that the different classes in your dataset can be separated by linear decision boundaries. This makes it a powerful tool for problems where linear classification is appropriate.
4. **Scatter Matrices:**
    - LDA uses two scatter matrices: one representing the within-class scatter (how data points spread within the same class) and another for the between-class scatter (how class means differ from each other). By maximizing the ratio of these matrices, LDA identifies the best linear discriminants.

How LDA Works:

1. **Compute Class Means:**
    - The first step in LDA is to calculate the mean of each class in the dataset. This allows LDA to measure how different the classes are from each other.
2. **Compute Scatter Matrices:**
    - The within-class scatter matrix captures how data points in each class are dispersed around the class mean. The between-class scatter matrix reflects how different the class means are from one another. LDA aims to minimize the former and maximize the latter.

3. **Solve Eigenvalue Problem:**
    - Using the scatter matrices, LDA solves an eigenvalue problem to find the directions (discriminants) that provide the best class separation. These directions form the axes onto which the data will be projected.
4. **Project Data:**
    - The original dataset is projected onto the new axes (linear discriminants), forming a lower-dimensional representation where class separability is maximized.

Advantages of LDA:

- **Improved Class Separability:**
    - LDA excels at enhancing the separation between different classes, which can significantly improve the performance of classification algorithms.
- **Dimensionality Reduction:**
    - LDA reduces the feature space while preserving important class information, making it ideal for simplifying high-dimensional datasets and preventing overfitting.
- **Robust for Small Datasets:**
    - LDA is particularly effective when the dataset has many features but relatively few observations, helping to combat the curse of dimensionality.

Limitations:

- **Linearity Assumption:**
    - LDA assumes that classes are linearly separable. In cases where the classes follow non-linear patterns, LDA may not perform well, and other methods like Quadratic Discriminant Analysis (QDA) might be better.
- **Equal Covariance Assumption:**
    - LDA assumes that all classes have the same covariance structure, which might not hold true in real-world datasets.
- **Sensitivity to Outliers:**
    - LDA can be sensitive to outliers because it tries to maximize class separability based on means and variances, which can be affected by extreme values.

Applications of LDA:

- **Face Recognition:**
    - LDA is widely used in face recognition tasks to reduce the dimensionality of facial features while maintaining the ability to distinguish between different individuals.
- **Text Classification:**
    - LDA can be applied to text data for tasks like sentiment analysis or topic modeling by reducing the number of features (words) while ensuring that the classifications (positive/negative sentiment, topic labels) are maintained.

Linear Discriminant Analysis (LDA) is a powerful and intuitive tool for dimensionality reduction in supervised learning tasks. By focusing on maximizing the separation between classes, it not only simplifies high-dimensional data but also enhances the performance of classification models. Whether applied to tasks like face recognition, medical diagnosis, or text classification, LDA offers a robust framework for balancing simplicity and predictive accuracy. However, it’s crucial to remember the method’s assumptions of linearity and equal covariance when selecting LDA for real-world applications.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import seaborn as sns
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline

# Import the 3 dimensionality reduction methods
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
as LDA

# Import the pandas library for data manipulation and analysis
import pandas as pd

# Load the training dataset from a CSV file
# The file path is specified as a raw string to avoid issues with
backslashes
train = pd.read_csv(r"C:\Users\Zahid.Shaikh\100days\63\train.csv")

# Print the shape of the DataFrame to see the number of rows and
columns
print(train.shape)

# Display the DataFrame to inspect its contents
train

(42000, 785)

       label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6
pixel7  \
0          1       0       0       0       0       0       0       0
0
1          0       0       0       0       0       0       0       0
0
2          1       0       0       0       0       0       0       0
0
3          4       0       0       0       0       0       0       0
0
4          0       0       0       0       0       0       0       0
0
...      ...     ...     ...     ...     ...     ...     ...     ...
...
41995      0       0       0       0       0       0       0       0
0
```

```
41996      1       0       0       0       0       0       0       0
0
41997      7       0       0       0       0       0       0       0
0
41998      6       0       0       0       0       0       0       0
0
41999      9       0       0       0       0       0       0       0
0

       pixel8  ...  pixel774  pixel775  pixel776  pixel777
pixel778  \
0           0  ...         0         0         0         0         0

1           0  ...         0         0         0         0         0

2           0  ...         0         0         0         0         0

3           0  ...         0         0         0         0         0

4           0  ...         0         0         0         0         0

...       ...  ...       ...       ...       ...       ...       ...

41995       0  ...         0         0         0         0         0

41996       0  ...         0         0         0         0         0

41997       0  ...         0         0         0         0         0

41998       0  ...         0         0         0         0         0

41999       0  ...         0         0         0         0         0


       pixel779  pixel780  pixel781  pixel782  pixel783
0             0         0         0         0         0
1             0         0         0         0         0
2             0         0         0         0         0
3             0         0         0         0         0
4             0         0         0         0         0
...         ...       ...       ...       ...       ...
41995         0         0         0         0         0
41996         0         0         0         0         0
41997         0         0         0         0         0
41998         0         0         0         0         0
41999         0         0         0         0         0

[42000 rows x 785 columns]

# Save the labels to a Pandas Series named 'target'
target = train['label']
```

```python
# Drop the label feature from the training dataset as it will not be
used for PCA
train = train.drop("label", axis=1)

# Import the StandardScaler class from sklearn for data
standardization
from sklearn.preprocessing import StandardScaler

# Convert the training DataFrame to a NumPy array
X = train.values

# Standardize the data by removing the mean and scaling to unit
variance
X_std = StandardScaler().fit_transform(X)

# Calculate the mean vector of the standardized data
mean_vec = np.mean(X_std, axis=0)

# Compute the covariance matrix of the standardized data
cov_mat = np.cov(X_std.T)

# Calculate the eigenvalues and eigenvectors of the covariance matrix
eig_vals, eig_vecs = np.linalg.eig(cov_mat)

# Create a list of (eigenvalue, eigenvector) tuples for further
processing
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:, i]) for i in
range(len(eig_vals))]

# Sort the eigenvalue-eigenvector pairs from high to low based on
eigenvalues
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Calculate the total sum of eigenvalues
tot = sum(eig_vals)

# Calculate individual explained variance for each principal component
var_exp = [(i / tot) * 100 for i in sorted(eig_vals, reverse=True)]

# Calculate cumulative explained variance to see the total variance
explained by the first k components
cum_var_exp = np.cumsum(var_exp)

import plotly.graph_objs as go  # Import necessary Plotly graphing
library
from plotly.subplots import make_subplots  # Import to create subplots

# Define the first trace for cumulative explained variance
trace1 = go.Scatter(
    x=list(range(784)),  # X-axis values representing feature columns
```

```
    y=cum_var_exp,  # Y-axis values representing cumulative explained
variance
    mode='lines+markers',  # Display both lines and markers on the
plot
    name="Cumulative Explained Variance",  # Name for the legend
    line=dict(
        shape='spline',  # Smooth line shape for better visual
representation
        color='goldenrod'  # Set the line color
    )
)

# Define the second trace for individual explained variance
trace2 = go.Scatter(
    x=list(range(784)),  # X-axis values representing feature columns
    y=var_exp,  # Y-axis values representing individual explained
variance
    mode='lines+markers',  # Display both lines and markers on the
plot
    name="Individual Explained Variance",  # Name for the legend
    line=dict(
        shape='linear',  # Straight line shape for clarity
        color='black'  # Set the line color
    )
)

# Create the figure using make_subplots to enable subplots and insets
fig = make_subplots(
    insets=[{'cell': (1, 1), 'l': 0.7, 'b': 0.5}],  # Define inset
layout parameters
    print_grid=True  # Print grid for better visualization
)

# Add the first trace (cumulative explained variance) to the main plot
fig.add_trace(trace1, 1, 1)

# Add the second trace (individual explained variance) to the main
plot
fig.add_trace(trace2, 1, 1)

# Configure layout settings for the main plot
fig.update_layout(
    title='Explained Variance Plots - Full and Zoomed-in',  # Set the
title of the plot
    xaxis=dict(range=[0, 80], title='Feature Columns'),  # Configure
X-axis properties
    yaxis=dict(range=[0, 60], title='Explained Variance'),  #
Configure Y-axis properties
)
```

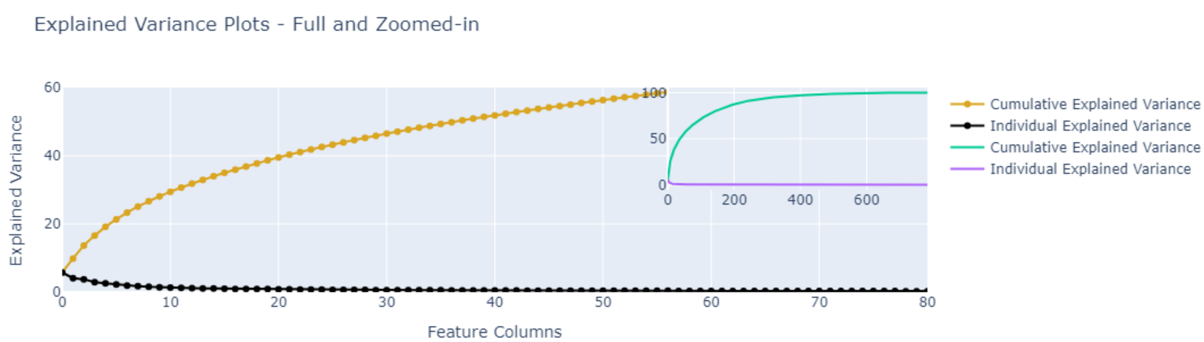
```
# Add the cumulative explained variance trace to the inset
fig.add_trace(go.Scatter(
    x=list(range(784)),  # X-axis values for the inset
    y=cum_var_exp,  # Y-axis values for cumulative explained variance
    xaxis='x2',  # Specify which X-axis to use in the inset
    yaxis='y2',  # Specify which Y-axis to use in the inset
    name='Cumulative Explained Variance'  # Name for the inset trace
))

# Add the individual explained variance trace to the inset
fig.add_trace(go.Scatter(
    x=list(range(784)),  # X-axis values for the inset
    y=var_exp,  # Y-axis values for individual explained variance
    xaxis='x2',  # Specify which X-axis to use in the inset
    yaxis='y2',  # Specify which Y-axis to use in the inset
    name='Individual Explained Variance'  # Name for the inset trace
))

# Display the plot
fig.show()

This is the format of your plot grid:
[ (1,1) x,y ]

With insets:
[ x2,y2 ] over [ (1,1) x,y ]
```




```
lda = LDA(n_components=5)
# Taking in as second argument the Target as labels
X_LDA_2D = lda.fit_transform(X_std, target.values )

traceLDA = go.Scatter(
    x = X_LDA_2D[:,0],
    y = X_LDA_2D[:,1],
#     name = Target,
#     hoveron = Target,
```

```
    mode = 'markers',
    text = target,
    showlegend = True,
    marker = dict(
        size = 8,
        color = target,
        colorscale ='Jet',
        showscale = False,
        line = dict(
            width = 2,
            color = 'rgb(255, 255, 255)'
        ),
        opacity = 0.8
    )
)
data = [traceLDA]

layout = go.Layout(
    title= 'Linear Discriminant Analysis (LDA)',
    hovermode= 'closest',
    xaxis= dict(
         title= 'First Linear Discriminant',
        ticklen= 5,
        zeroline= False,
        gridwidth= 2,
    ),
    yaxis=dict(
        title= 'Second Linear Discriminant',
        ticklen= 5,
        gridwidth= 2,
    ),
    showlegend= False
)

fig = dict(data=data, layout=layout)
py.iplot(fig, filename='styled-scatter')
```