Dimensionality Reduction

Reading Material

PW SKILLS

Topics Covered

1. The Curse of Dimensionality
   - Definition and Significance
   - Effects of High Dimensionality
   - Strategies to Mitigate the Curse

2. Dimensionality Reduction Techniques
   - Introduction to Dimensionality Reduction
   - Types of Dimensionality Reduction Techniques
   - Applications of Dimensionality Reduction

3. PCA (Principal Component Analysis)
   - Introduction to PCA
   - Algorithm Steps
   - Applications of PCA
   - Advantages and Limitations
   - Practical Implementation

4. LDA(Linear Discriminant Analysis)
   - Introduction to LDA
   - Algorithm Steps
   - Applications of LDA
   - Practical Implementation

5. PCA V/s LDA

6. Interview Questions.(Easy/Medium/Hard)
   Theoretical Questions (20 Questions)
   Practical Questions (20 Questions)

7. Multiple Choice Questions.(Minimum Question 25-30 MCQ)

The Curse of Dimensionality

Definition and Significance

The term "Curse of Dimensionality" describes the different problems and complexities occurring with data analysis and organization in high-dimensional spaces. In machine learning, understanding this concept is crucial because as the number of features or dimensions increases in some dataset, the amount of data we need in order to generalize accurately grows exponentially.

Effects of High Dimensionality

High dimensionality can give rise to a number of problems including:
Data sparsity: the more the number of dimensions increases, the sparser the data gets, and hence, any clustering or classification tasks are not easy.

- **Increased computation:** The extra number of dimensions requires greater computational resources and longer times to process data.
- **Overfitting:** The higher the dimension, the more complex models can turn out. They end up fitting to the noise rather than the pattern underlying the noise and hence reduce the generality of the model to new data.
- **Distances cease to mean anything:** The relative distances of pairs of data points tend to converge to a single distance in high dimensions, making measures such as Euclidean distance much less meaningful.
- **Performance Degradation:** Distance measurement-based algorithms, such as k-nearest neighbors, will degrade in performance. Challenges of visualization: High-dimensional data does not allow for easy visualization, hence making the process of exploratory data analysis tricky.

Strategies to attenuate the curse

Dimensionality reduction mainly remedies the curse of dimensionality. It is the process of reducing the number of random variables considered by getting a set of principal variables. By reducing the dimensions, we are retaining the most important information in data and discarding the redundant or less important features. Some common techniques used to reduce dimensionality are:

Principal Component Analysis: This is a statistical procedure that changes the original variables into a completely new set of variables-not only a different number, but also linear combinations of the original variables.

LDA (Linear Discriminant Analysis): This is a method that works to find which attributes best explain the differences between classes, proving useful in classification tasks.

t-Distributed Stochastic Neighbor Embedding-t-SNE: A non-linear technique of dimensionality reduction and is especially good for visualization of high-dimensional datasets.

Autoencoder: These are models of neural networks used in a process called the reduction of dimensionality. This neural network process works by compressively shrinking the input into a compact representation and then reconstructing the original input from this representation.

Consequently, these dimensionality reduction methods reduce the effect of high dimensionality and thereby enhance the performance of machine learning algorithms.

Dimensionality Reduction

Introduction to Dimensionality Reduction

In fact, dimensionality reduction is a statistical approach to lowering data complexity. It is one of the techniques of variable reduction applied to a reduced training dataset in which machine learning model development is undertaken. The process keeps track of the dimensionality of the data by projecting high-dimensional data into a lower dimension where it captures the best core essence of the data.

Types of Dimensionality Reduction Techniques:

The dimensionality reduction techniques can be broadly categorized basically on the two bases: feature selection and feature extraction. Feature selection retains relevant (optimal) features and discards irrelevant features to ensure high model accuracy. Feature extraction refers to the process related to conversion of multidimensional space into lower dimensional space.

Various common dimensionality reduction techniques are as follows:

It can roughly be divided into two categories of dimensionality reduction techniques: Feature Selection and Feature Extraction.

1. Feature Selection

Feature selection involves choosing a subset from among the original features, and this can be achieved by one of the following strategies:

- **Filter Strategy:** Features are to be selected based upon the statistical relationship between the input and output variable. Correlation coefficients, chi-square tests, and mutual information are some of the techniques used.

- **Wrapper Strategy:** This approach gauges the performance of a model for a variety of subsets of features and selects the subset giving the best performance of the model. This is the leading approach to wrappers, known as Recursive Feature Elimination. Embedded Strategy: Feature selection happens within the model training process in this strategy. Some approaches, such as LASSO and decision trees, have packed feature selection into the model constructs, which is enacted during training.

2. Feature Extraction

Feature extraction is a means of transforming the data into another feature space, typically reducing the number of features by combining them:

1. **Principal Component Analysis:** PCA is the method whereby data is linearly transformed into a new coordinate system in such a way that the greatest variance on the first coordinate-so-called the first principal component-the second greatest variance on the second coordinate, and so forth. PCA is vastly used as a technique of reducing dimensionality for high-dimensional data sets in such a way that the variance could be preserved as much as possible.

2. **Nonnegative Matrix Factorization:** This method represents a linear technique for decomposing the non-negative matrix into two non-negative matrices in such a way that it still captures the additive nature of the information. This technique is utilized and applied in many domains where non-negativity must be required, including image and text data.

3. **Linear Discriminant Analysis-LDA:** The LDA is both a feature reduction technique and a classification technique. It projects the data into a lower dimension while maximizing the separation between different classes. It finds its application in situations where the classes are well separated.

4. **Generalized Discriminant Analysis:** GDA extends LDA to nonlinear relationships using appropriate kernel functions. It projects the data into a higher-dimensional space so that linear separability becomes possible and performs dimensionality reduction while maximising class separability.

5. **Ratio of Missing Values:** This will just filter out the features that contain a high percentage with missing values according to a threshold ratio. It helps when some of the features have more rather than the required ratio of missing values. Perhaps none of the useful information is held in those features.

6. **Low Variance Filter:** This technique removes those features that are of low variance. That means they provide very little information to the model. Thus, all those features that have a variance less than a particular threshold are removed, thereby simplifying the dataset.

7. **High Correlation Filter:** This method removes the features that are highly correlated with each other. In this way, it diminishes redundant information contributed by each feature pair. Typically, one feature for every high level of correlation is retained, at which multicollinearity is reduced.

8. **Backward Feature Elimination:** It is an iterative method that starts with all the features. At each step, it removes the least important feature. Then it runs retraining of the model. This process is continued until there is no more improvement in the model performance.

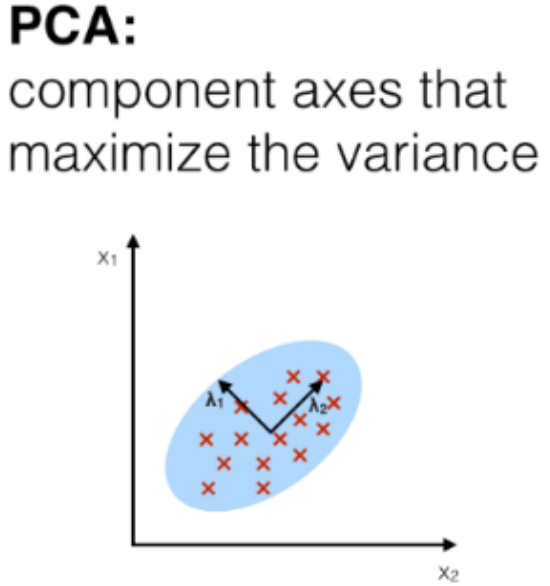
9. **Forward Feature Construction:** The opposite of backward elimination, this technique starts with no features and adds the most important feature at a time, each time retraining the model. It proceeds until the performance of the model plateaus. Random Forests: While commonly used for classification and regression tasks, the random forest is yet another ensemble method which has happened to perform feature selection. A forest of a large number of decision trees is constructed and the importance of each feature evaluated depending on how often it is used to split the data, keeping only the most informative features.ng only the most informative features.

Dimensionality Reduction Applications

Dimensionality reduction techniques play an important role in the following real-life applications:

1. **Text Categorization:** Automatically classifying newly-added documents based on predefined categories.
2. **Image Retrieval:** Indexing images according to their visual contents, such as colors, textures, or shapes.
3. **Gene Expression Analysis:** This involves the determination and analysis of the expression levels of several thousand genes in a single experiment.
4. **Intrusion Detection:** A system to protect network-based computers in cyberspace against outside threats by determining some optimum features that act as a check-point to suspicious activities.
5. **Neuroscience:** This is widely used in neuroscience for performing statistical analysis on neural responses. These are dimensionality reduction methodologies that will help reduce the effect of high dimensionality and improve the performance of machine learning algorithms.\

Principal Component Analysis-PCA



Introduction to PCA

Principal Component Analysis is a dimensionality reduction and machine learning method that summarizes a gigantic data set into a smaller set, while having retained most of the pattern and trend in the original data. It does this by changing a gigantic set of variables into a smaller set of variables that retains most of the information that exists within the gigantic set.

Algorithm Steps

1. **Standardization of the range of continuous initial variables:** This step is important in that PCA is sensitive to the variances of the initial variables. Standardizing the range of continuous initial variables ensures each of them will contribute equally to the analysis.
2. **The covariance matrix calculation to identify correlations:** The covariance matrix is basically a table that summarizes the correlation between all the possible pairs of variables. This helps to understand how the variable of the input dataset is varying from mean with respect to each other.
3. To identify the principal components, calculate the covariance matrix eigenvectors and eigenvalues. The eigenvector and eigenvalue are concepts from linear algebra that help in finding the main directions of data variability. Eigenvectors of the Covariance matrix are the directions of the axes where there is most variance, and eigenvalues are coefficients attached to eigenvectors that give the amount of variance carried in each Principal Component.
4. **Formation of the feature vector to determine PCs to retain:** Ranked eigenvectors by their eigenvalues yield the principal components ordered by importance. Then form the feature vector with the eigenvectors of the components that are decided to keep.

Project the data along the axes of principal components: The feature vector that has been developed by using the eigenvectors of the covariance matrix acts as a means to reorient data from the original axes to those represented by the principal components.

Applications of PCA

PCA is widely used in various fields, including:

1. **Data compression:** in this case, one may want to reduce a number of variables for a dataset via PCA during data storage and processing.
2. **Data Visualization:** PCA facilitates the process of visualizing data in a low-dimensional space by reducing the dimensionality of a data set.
3. **Noise filtering:** PCA works toward the filtering-out of noise in a dataset by deleting the components of low variance.
4. **Genetics:** PCA also finds its applications in the analysis of genetic data, such as gene expression levels, to find out patterns or trends.
5. **Image Processing:** PCA finds its application in image processing, such as noise reduction, image compression, and feature extraction.

Advantages and Limitations

Positive features: PCA is a potent tool in the areas of dimensionality reduction and filtering out noise. It is not as complex to implement, and it also enjoys wide applicability to many kinds of data.

Limitations:
The limitations are that most of the methods of PCA mentioned here assume that the principal components are linear combinations of the original variables. Therefore, it may not work well in conditions where the underlaying relationships are nonlinear. Another limitation is that PCA does not consider prior knowledge regarding either data or the problem being solved.

Practical Implementation of Principal Component Analysis (PCA)

Implementing PCA in practice involves several steps, from data preprocessing to applying PCA and interpreting the results. Below is a step-by-step guide on how to implement PCA using Python and its popular libraries such as numpy, pandas, and scikit-learn.

```
#Step 1: Importing the Necessary Libraries
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Step 2:  Load the Iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Display the first few rows of the dataset
print(df.head())
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

#Step 3: Data Preprocessing

Before applying PCA, it is crucial to standardize the data. PCA is sensitive to the scale of the data, so features should be standardized to have a mean of 0 and a variance of 1.

```
# Separate features and target
X = df.drop(columns=['target'])
y = df['target']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Step 4: Applying PCA

Now, we can apply PCA to reduce the dimensionality of the dataset. Let's reduce the data to 2 principal components for visualization purposes.

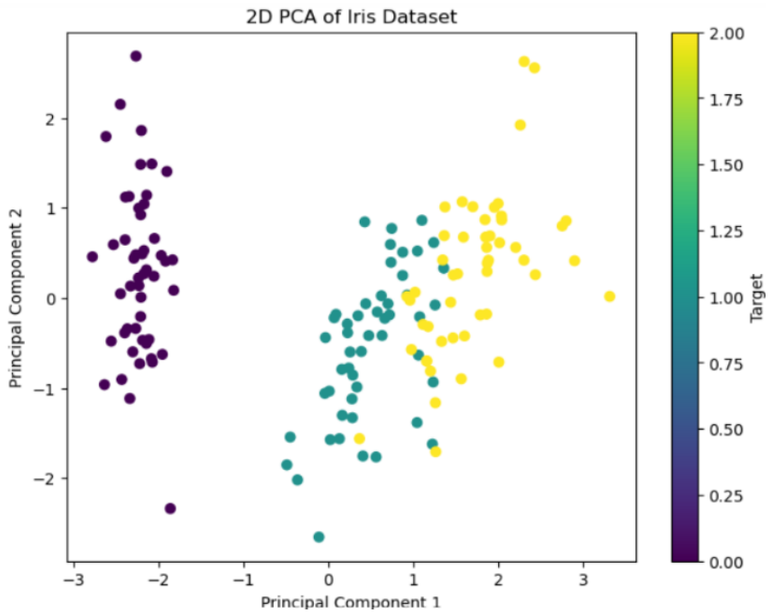
```
# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Convert the result to a DataFrame
pca_df = pd.DataFrame(data=X_pca, columns=['Principal Component 1',
'Principal Component 2'])
pca_df['target'] = y

# Display the first few rows of the PCA-transformed data
print(pca_df.head())

#The Output:
```

	Principal Component 1	Principal Component 2	target
0	-2.264703	0.480027	0
1	-2.080961	-0.674134	0
2	-2.364229	-0.341908	0
3	-2.299384	-0.597395	0
4	-2.389842	0.646835	0

Step 5: Visualizing the PCA Results

PCA can be particularly useful for visualizing high-dimensional data in 2D or 3D. Here, we'll plot the two principal components.

```
# Plot the principal components
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['Principal Component 1'], pca_df['Principal Component
2'], c=pca_df['target'], cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('2D PCA of Iris Dataset')
plt.colorbar(label='Target')
plt.show()
```



Step 6: Interpreting the Explained Variance

Understanding how much variance is captured by each principal component is essential. The explained_variance_ratio_ attribute of the PCA object provides this information.

```
# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f'Explained variance by each component: {explained_variance}')
print(f'Total variance explained by the first 2 components:
{np.sum(explained_variance)}')
```

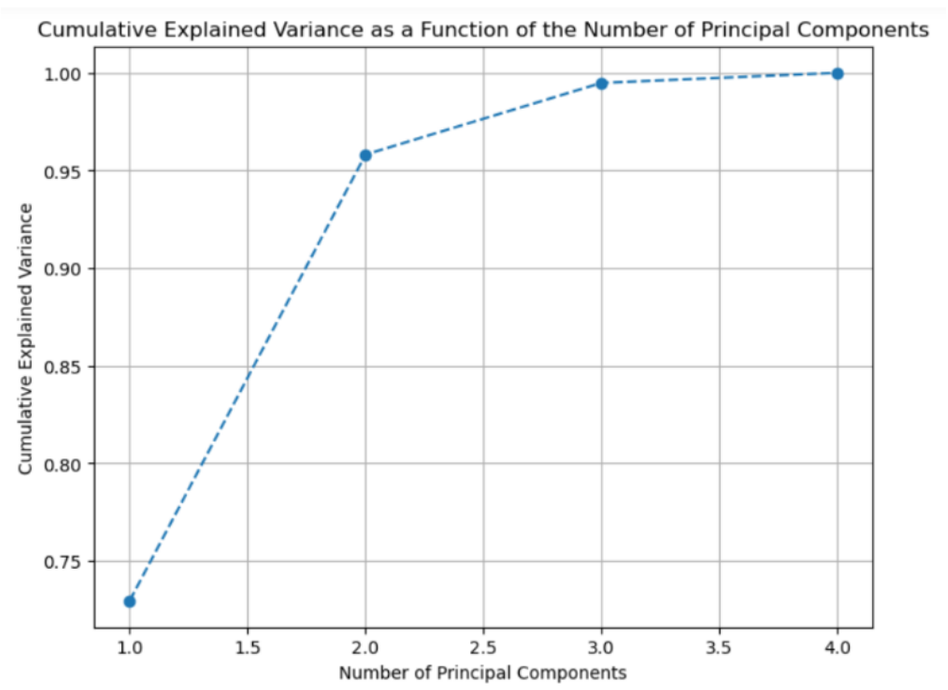
The Output:

Explained variance by each component: [0.72962445 0.22850762]
Total variance explained by the first 2 components: 0.9581320720000164
This output tells you the proportion of the dataset's variance that is captured by each principal component.

Step 7: Selecting the Optimal Number of Components

In practice, you might want to retain more than two principal components. A common approach is to plot the cumulative explained variance to determine the optimal number of components.

```
# Apply PCA with all components
pca_full = PCA()
pca_full.fit(X_scaled)

# Plot cumulative explained variance
cumulative_variance = np.cumsum(pca_full.explained_variance_ratio_)
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
marker='o', linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance as a Function of the Number of
Principal Components')
plt.grid()
plt.show()
```



This plot helps you decide how many components are needed to explain a desired amount of variance (e.g., 95%).

Step 8: Reconstructing Data from Principal Components (Optional)

If you need to reconstruct the original data from the principal components, you can use the inverse transform.

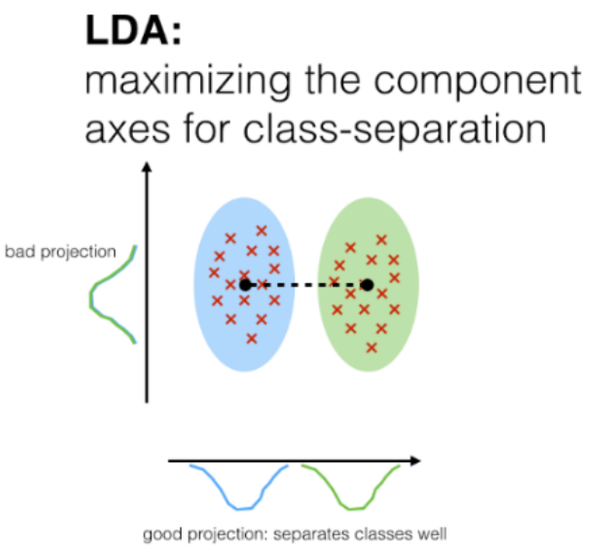
```
# Reconstruct data from the first two principal components
X_reconstructed = pca.inverse_transform(X_pca)

# Convert to DataFrame and inspect
reconstructed_df = pd.DataFrame(X_reconstructed,
columns=iris.feature_names)
print(reconstructed_df.head())
```

Reconstructing the data can help you understand how much information is lost when reducing dimensions.

The Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.998889	1.053198	-1.302707	-1.247098
1	-1.338748	-0.061923	-1.224328	-1.220572
2	-1.360961	0.321117	-1.380603	-1.358338
3	-1.423598	0.067762	-1.349224	-1.338813
4	-1.001138	1.240918	-1.371254	-1.306618

Linear Discriminant Analysis (LDA)



Definition and Overview

Linear Discriminant Analysis (LDA) is a classification and dimensionality reduction technique that aims to find a linear combination of features that characterizes or separates two or more classes of objects. It transforms features into a lower-dimensional space, making it easier to perform classification while retaining as much class-discriminating information as possible.

LDA differs from other dimensionality reduction techniques, such as Principal Component Analysis (PCA), because LDA explicitly accounts for class labels and seeks to maximize the separability between known classes, making it a supervised learning method.

Historical Context

LDA was developed by the statistician Ronald A. Fisher in 1936. Initially intended for distinguishing between two classes (binary classification), LDA has since been extended to handle multiple classes and has become a key tool in many classification tasks.

Significance in Machine Learning

LDA is particularly useful when the classes are linearly separable or nearly so. Its application spans various domains, including image recognition, bioinformatics, and financial modeling, where the goal is not only dimensionality reduction but also improving classification accuracy by projecting data into a lower-dimensional space while maintaining class separability.

How LDA Works?

Intuition Behind LDA

The core idea behind LDA is to find a new axis (or axes) in such a way that when data is projected onto this axis, the classes are as separable as possible. LDA works by finding the direction(s) that maximize the difference between the means of different classes (between-class variance) while minimizing the spread of each class (within-class variance). In other words, it maximizes the ratio of between-class variance to within-class variance, which ensures good class separability.

Mathematical Formulation

The mathematical foundation of LDA lies in its objective function, which maximizes the ratio of the determinant of the between-class scatter matrix Sb to the determinant of the within-class scatter matrix SwS_wSw. The formula is:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_b \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}$$

Where:

- S_b represents the between-class scatter matrix.
- S_w represents the within-class scatter matrix.
- $\mathbf{w}$ is the projection vector that defines the new axis for the transformed data.

The goal is to find the projection vector w\mathbf{w}w that maximizes this objective function, ensuring the classes are well-separated in the lower-dimensional space.

Linear Discriminant

The linear discriminant is the line (or hyperplane in higher dimensions) along which the data will be projected. Once the data is projected onto this line, we use standard classification techniques to separate the classes.

3. Steps in LDA Algorithm

Step 1: Compute the Mean Vectors for Each Class

For each class in the dataset, calculate the mean vector. This represents the average value for each feature within that class. Let mk represent the mean vector for class k, and m be the overall mean vector.

Step 2: Compute the Scatter Matrices

- **Within-Class Scatter Matrix (S_w):** Measures the spread of data points within each class. It captures how dispersed the data is within individual classes.

$$S_w = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} (\mathbf{x}_i - \mathbf{m}_k)(\mathbf{x}_i - \mathbf{m}_k)^T$$

Where Nk is the number of samples in class k.

Step 4: Sort the Eigenvalues and Select the Top Eigenvectors
Sort the eigenvalues in descending order and select the top kkk eigenvectors (where kkk is the number of dimensions you want to reduce the data to).

Step 5: Project the Data onto the New Subspace
Transform the original dataset by projecting it onto the new kkk-dimensional subspace formed by the selected eigenvectors.

4. Applications of LDA

Use Cases in Real-World Scenarios

LDA has broad applications in various fields, including:

- **Facial Recognition:** LDA is commonly used in computer vision for tasks like face recognition. By reducing the dimensionality of facial features while retaining the class information (i.e., different individuals), LDA improves the accuracy of recognition algorithms.
- **Bioinformatics:** In genomics and proteomics, LDA is used to identify patterns and classify biological samples based on gene expression data.
- **Marketing and Customer Segmentation:** LDA can classify customers into distinct groups based on their purchasing behaviors or demographic data, enabling more targeted marketing efforts.

Comparative Analysis
In many classification problems, LDA is preferred over PCA when class labels are available, as LDA focuses on maximizing class separability, whereas PCA focuses only on variance without considering class information.

5. Practical Implementation of LDA

In this section, we'll demonstrate how to implement Linear Discriminant Analysis (LDA) using Python and scikit-learn. We will also visualize the results using matplotlib to better understand how LDA projects data onto a lower-dimensional space while maintaining class separability.

Step-by-Step Implementation

First, let's import the necessary libraries:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

Next, we'll load the Iris dataset, a classic dataset in machine learning, and prepare it for LDA:

```
# Load dataset
data = load_iris()
X = data.data
y = data.target

# Define label dictionary for visualization
label_dict = {1: 'Setosa', 2: 'Versicolor', 3: 'Virginica'}
```

We'll then apply LDA to reduce the dimensionality of the data:

```
# LDA
sklearn_lda = LDA(n_components=2)
X_lda_sklearn = sklearn_lda.fit_transform(X, y)
```

Now, let's define a function to visualize the LDA results:

```
def plot_scikit_lda(X, title):
    ax = plt.subplot(111)
    for label, marker, color in zip(range(1, 4), ('^', 's', 'o'), ('blue',
'red', 'green')):
        plt.scatter(x=X[:, 0][y == label],
                    y=X[:, 1][y == label] * -1,  # flip the figure
                    marker=marker,
                    color=color,
                    alpha=0.5,
                    label=label_dict[label])

    plt.xlabel('LD1')
    plt.ylabel('LD2')


    leg = plt.legend(loc='upper right', fancybox=True)
    leg.get_frame().set_alpha(0.5)
    plt.title(title)

    # Hide axis ticks
    plt.tick_params(axis="both", which="both", bottom=False, top=False,
labelbottom=True, left=False, right=False, labelleft=True)

    # Remove axis spines
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

    plt.grid()
    plt.tight_layout()
    plt.show()
```

This function creates a 2D scatter plot of the data after applying LDA. Each point is colored according to its class label, and the axes represent the first two linear discriminants (LD1 and LD2).

Let's also define another plotting function to visualize the step-by-step LDA process:

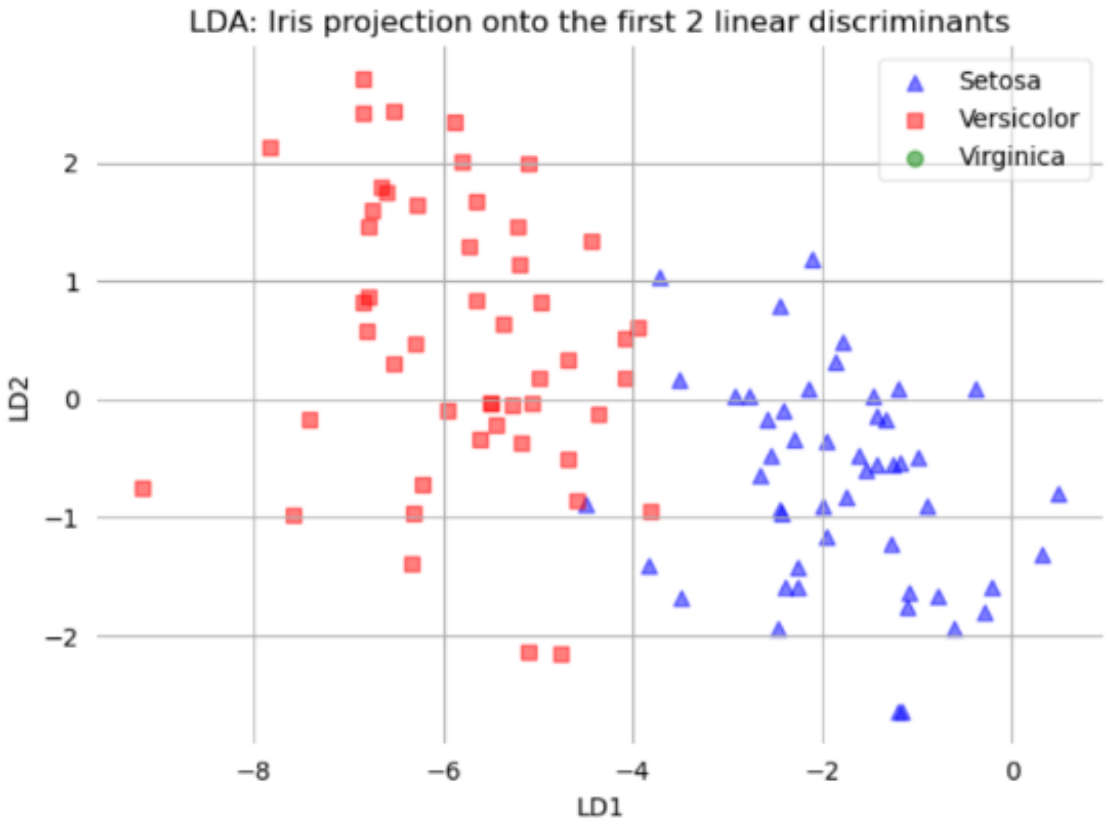
```
def plot_step_lda():
    ax = plt.subplot(111)
    for label, marker, color in zip(range(1, 4), ('^', 's', 'o'), ('blue',
'red', 'green')):
        plt.scatter(x=X_lda_sklearn[:, 0].real[y == label],
                    y=X_lda_sklearn[:, 1].real[y == label],
                    marker=marker,
                    color=color,
                    alpha=0.5,
                    label=label_dict[label])

    plt.xlabel('LD1')
    plt.ylabel('LD2')

    leg = plt.legend(loc='upper right', fancybox=True)
    leg.get_frame().set_alpha(0.5)
    plt.title('LDA: Iris projection onto the first 2 linear discriminants')

    # Hide axis ticks
    plt.tick_params(axis="both", which="both", bottom=False, top=False,
labelbottom=True, left=False, right=False, labelleft=True)

    # Remove axis spines
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

    plt.grid()
    plt.tight_layout()
    plt.show()
```

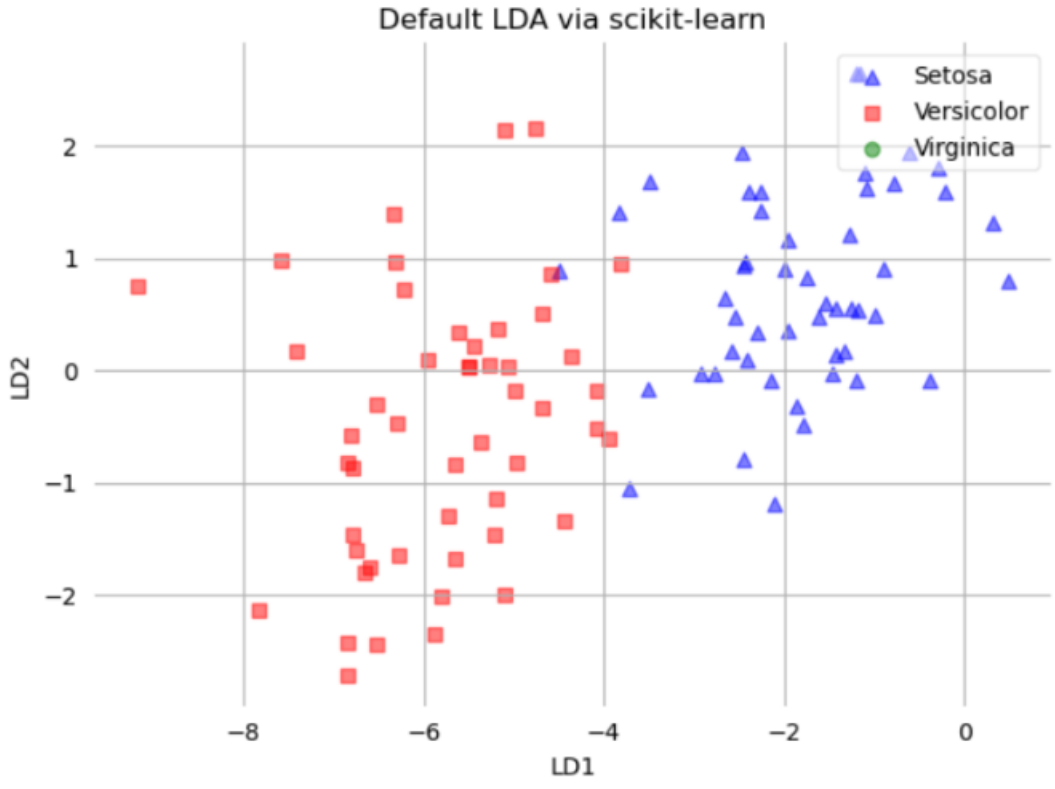
This function visualizes the projection of the Iris dataset onto the first two linear discriminants using LDA.

Plotting the Results

Finally, let's plot the LDA results using the functions defined above:

```
# Plot the step-by-step LDA visualization
plot_step_lda()

# Plot the final LDA result using scikit-learn
plot_scikit_lda(X_lda_sklearn, title='Default LDA via scikit-learn')
```


LDA: Iris projection onto the first 2 linear discriminants
Setosa
Versicolor
Virginica
LD1
LD2
2
1
0
−1
−2
−8
−6
−4
−2
0


Default LDA via scikit-learn
Setosa
Versicolor
Virginica
LD1
LD2
2
1
0
−1
−2
−8
−6
−4
−2
0

Explanation of the Output

- **LDA Projection:** The scatter plots generated by the functions above show how the LDA algorithm projects the original 4-dimensional Iris data onto a
- 2-dimensional space defined by the first two linear discriminants. In these plots, each point corresponds to an iris sample, and the color represents the species (Setosa, Versicolor, Virginica).
- **Class Separation:** The plots clearly show that LDA effectively separates the different iris species in this 2D space, which demonstrates the strength of LDA in creating class-discriminative features.

Why LDA?

- LDA is particularly useful when dealing with a high-dimensional dataset as it provides a way to reduce the number of dimensions while preserving as much of the variation in the original data as possible.

- It also helps identify the most important variables in a dataset, which is useful for further analysis. LDA can be used for exploratory data analysis and for predictive modeling.

- LDA is also used in face detection algorithms. In Fisherfaces, LDA is used to extract useful data from different faces. Coupled with eigenfaces, it produces effective results.

Drawbacks of Linear Discriminant Analysis (LDA)

1. It assumes that the data is normally distributed, which may not always be the case in real-world data sets.
2. It assumes that the variables are statistically independent, which is also not always the case in real-world data sets.
3. It is sensitive to outliers, which can affect the accuracy of the model.
4. It can only be used for two-class classification problems, and does not work well with multi-class problems.
5. It is not suitable for data sets with a large number of features, as it can become computationally expensive.

Difference between Linear Discriminant Analysis and Principal Component Analysis

Feature	Linear Discriminant Analysis (LDA)	Principal Component Analysis (PCA)
Nature	Supervised learning technique	Unsupervised dimensionality reduction technique
Objective	Classify objects into categories based on predictor variables	Reduce dimensionality and retain most data variability
Applicability	Commonly used in classification problems, especially for binary classification	Widely used for simplifying datasets with many variables

Linear Combination	Linear Discriminant Analysis (LDA)	Extracts uncorrelated components capturing data variability
Classes Separation	Maximizes the separation between classes	Maximizes variance along principal components
Data Requirement	Requires labeled data for training	Works well with unlabeled data
Complexity Reduction	Focuses on separating classes, less emphasis on reducing complexity	Primarily used to reduce the complexity of a dataset
Ordered Components	Components are ordered based on their contribution to class separation	Components are ordered by the amount of total variance they explain
Use Case Example	Face recognition, medical diagnosis	Image compression, feature extraction