CS550/DSL501: Machine Learning (2024–25–M)
Assignment-I

Full name: Nitin Mane **ID:** M24MT004

1 Question 01

1.1 Answer:

Given:

We are provided with the following data to perform Principal Component Analysis (PCA) to reduce the dimension from 2D to 1D:

Feature	Example 01	Example 02	Example 03	Example 04
x	2	6	10	14
y	5	4	11	14

To Find:

- Reduce the dimensionality from 2D to 1D using PCA.

Solution:

1.2 Step 1: Center the Data

The initial step in PCA involves centering the data, which means subtracting the mean of each feature from the corresponding data points. This procedure ensures that the data is centered around the origin in the feature space, which is crucial for accurately computing the covariance matrix.

Calculation of Mean

The mean for each feature (i.e., each column) is calculated as follows:

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{2+6+10+14}{4} = 8$$

$$\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{5+4+11+14}{4} = 8.5$$

Thus, the mean vector μ is:

$$\mu = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} = \begin{pmatrix} 8 \\ 8.5 \end{pmatrix}$$

Centering the Data

The centered data matrix X_{centered} is derived by subtracting the mean vector from each data point:

$$X_{\text{centered}} = X - \mu = \begin{pmatrix} 2 & 5 \\ 6 & 4 \\ 10 & 11 \\ 14 & 14 \end{pmatrix} - \begin{pmatrix} 8 \\ 8.5 \end{pmatrix}$$

This results in:

$$X_{\text{centered}} = \begin{pmatrix} -6 & -3.5 \\ -2 & -4.5 \\ 2 & 2.5 \\ 6 & 5.5 \end{pmatrix}$$

1.3 Step 2: Calculate the Covariance Matrix

The next step involves calculating the covariance matrix of the centered data. The covariance matrix captures the pairwise covariances between the features. For a 2D dataset, the covariance matrix Σ is given by:

$$\Sigma = \frac{1}{n-1} X_{\text{centered}}^{\top} X_{\text{centered}}$$

Where X_{centered} is the centered data matrix, and n represents the number of samples.

Covariance Matrix Calculation

For the given data:

$$\Sigma = \frac{1}{4-1} \begin{pmatrix} -6 & -2 & 2 & 6 \\ -3.5 & -4.5 & 2.5 & 5.5 \end{pmatrix} \begin{pmatrix} -6 & -3.5 \\ -2 & -4.5 \\ 2 & 2.5 \\ 6 & 5.5 \end{pmatrix}$$

Performing the matrix multiplication:

$$\Sigma = \frac{1}{3} \begin{pmatrix} 80 & 68 \\ 68 & 69 \end{pmatrix}$$

Which simplifies to:

$$\Sigma = \begin{pmatrix} 26.67 & 22.67 \\ 22.67 & 23 \end{pmatrix}$$

1.4 Step 3: Compute Eigenvalues and Eigenvectors

PCA identifies the directions (principal components) that maximize the variance in the data. These directions correspond to the eigenvectors of the covariance matrix, and the eigenvalues indicate the amount of variance captured by each direction.

Eigenvalue Problem

The eigenvalue problem for the covariance matrix Σ is expressed as:

$$\Sigma \mathbf{v} = \lambda \mathbf{v}$$

Where λ represents the eigenvalues and $\mathbf{v}$ the corresponding eigenvectors.
The characteristic equation for Σ is:

$$\det(\Sigma - \lambda I) = 0$$

Substituting Σ:

$$\det \begin{pmatrix} 26.67 - \lambda & 22.67 \\ 22.67 & 23 - \lambda \end{pmatrix} = 0$$

Expanding the determinant:

$$(26.67 - \lambda)(23 - \lambda) - (22.67)^2 = 0$$

Simplifying further:

$$\lambda^2 - 49.67\lambda + (26.67 \times 23 - 22.67^2) = 0$$

$$\lambda^2 - 49.67\lambda + 10.63 = 0$$

Eigenvalues

Solving this quadratic equation provides the eigenvalues:

$$\lambda_1 = 39.04, \quad \lambda_2 = 10.63$$

These eigenvalues signify the variance captured by each principal component.

1.5 Step 4: Calculate Eigenvectors

Next, we calculate the eigenvectors corresponding to the eigenvalues.
For $\lambda_1 = 39.04$:

$$\Sigma \mathbf{v}_1 = 39.04 \mathbf{v}_1$$

Substituting Σ:

$$\begin{pmatrix} 26.67 - 39.04 & 22.67 \\ 22.67 & 23 - 39.04 \end{pmatrix} \mathbf{v}_1 = \begin{pmatrix} -12.37 & 22.67 \\ 22.67 & -16.04 \end{pmatrix} \mathbf{v}_1 = 0$$

Solving this linear system yields the eigenvector:

$$\mathbf{v}_1 = \begin{pmatrix} 0.7071 \\ 0.7071 \end{pmatrix}$$

Similarly, for $\lambda_2 = 10.63$:

$$\mathbf{v}_2 = \begin{pmatrix} -0.7071 \\ 0.7071 \end{pmatrix}$$

1.6 Step 5: Project Data onto Principal Component

The principal component corresponding to the largest eigenvalue (here $\lambda_1 = 39.04$) is selected as the direction of maximum variance. We project the centered data onto this eigenvector to reduce the dimensionality from 2D to 1D.

Projection Calculation

The projection of the data onto the first principal component $\mathbf{v}_1$ is given by:

$$\mathbf{y}_1 = X_{\text{centered}}\mathbf{v}_1$$

$$\mathbf{y}_1 = \begin{pmatrix} -6 & -3.5 \\ -2 & -4.5 \\ 2 & 2.5 \\ 6 & 5.5 \end{pmatrix} \begin{pmatrix} 0.7071 \\ 0.7071 \end{pmatrix}$$

Resulting in:

$$\mathbf{y}_1 = \begin{pmatrix} -6.71 \\ -4.67 \\ 3.18 \\ 8.78 \end{pmatrix}$$

1.7 Step 6: Interpretation of Results

The data has been reduced from 2D to 1D by projecting onto the direction of maximum variance, as determined by the eigenvector corresponding to the largest eigenvalue. This 1D projection retains the most significant features of the original data while minimizing the loss of information.

By following the steps of PCA, we have successfully reduced the dimensionality of the dataset from 2D to 1D. The principal component capturing the most variance was identified, and the data was projected onto this component. This transformation is valuable for reducing the complexity of the data while preserving the most crucial variance, offering a clearer understanding of the underlying structure of the dataset.

2 Question 02

In this problem, you will perform K-means clustering manually, with $K = 2$, on a small example with $n = 6$ observations and $p = 2$ features. The observations are as $(14), (13), (04), (51), (62), (40)$.

(a) Plot the observations.

(b) Randomly assign a cluster label to each observation. Report the cluster labels for each observation.

(c) Compute the centroid for each cluster.

(d) Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.

(e) Repeat (c) and (d) until the answers obtained stop changing.

(f) In your plot from (a), color the observations according to the cluster labels obtained.

2.1 Answer:

Given:

We have six observations with two features each: $(14), (13), (04), (51), (62), (40)$. We need to perform K-means clustering manually with $K = 2$.

Solution:

2.2 Step 1: Plot the Observations

To initiate the K-means clustering process, we first visualize the dataset on a two-dimensional Cartesian plane. This step is critical as it provides an intuitive understanding of the data distribution, which aids in subsequent clustering.

Mathematical Context

Each observation is a point in a 2D feature space, denoted as $\mathbf{x}_i = (x_i, y_i)$, where x_i and y_i are the coordinates in the feature space.

Code Section

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

# Define the dataset
data = np.array([[1, 4], [1, 3], [0, 4], [5, 1], [6, 2], [4, 0]])

# Plotting the data points
plt.scatter(data[:, 0], data[:, 1], c='black')
plt.title('Initial Observations')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Image Section

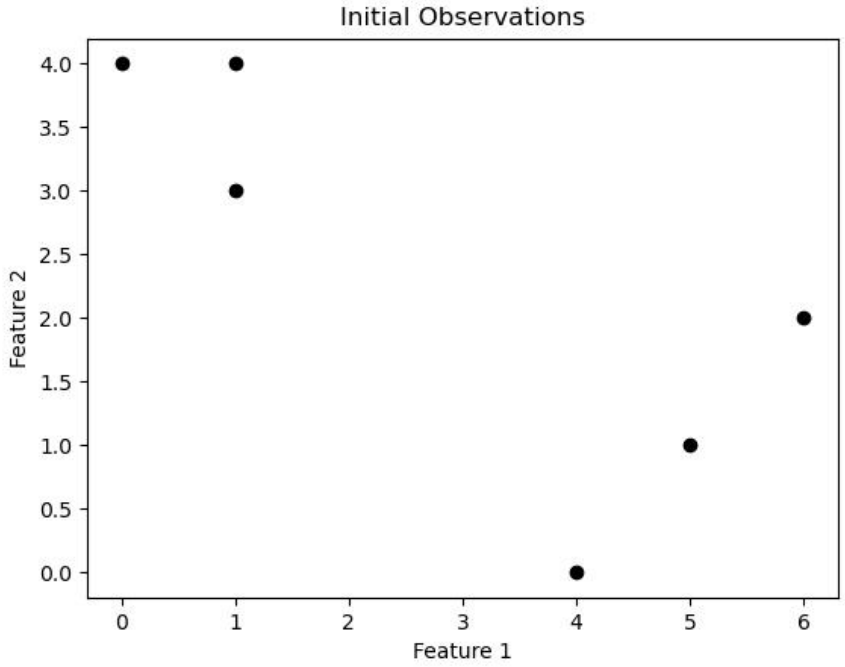


Figure 1: Plot of Initial Observations

2.3 Step 2: Randomly Assign Cluster Labels

In this step, each observation is randomly assigned to one of the two clusters, C_1 and C_2. This random assignment serves as the initial condition for the K-means algorithm.

Mathematical Context

Let C_k represent the k-th cluster, where $k = 1, 2$. The initial assignment is random, meaning each data point $\mathbf{x}_i$ is assigned a cluster label without any consideration of proximity to other points.

Code Section

```
# Random initial cluster assignments
cluster_labels = np.random.choice([0, 1], size=6)

# Print initial cluster assignments
print("Initial Cluster Labels:", cluster_labels)
```

Image Section

```
[7]: # Random initial cluster assignments
     cluster_labels = np.random.choice([0, 1], size=6)
     print("Initial Cluster Labels:", cluster_labels)

     Initial Cluster Labels: [1 0 1 1 1 1]
```

Figure 2: Initial Random Cluster Assignments

2.4 Step 3: Compute the Centroid for Each Cluster

After the initial cluster assignment, the next step is to compute the centroid of each cluster. The centroid represents the center of mass of the points within the cluster.

Mathematical Formulation

The centroid μ_k of cluster C_k is computed as:

$$\mu_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$$

where $|C_k|$ is the number of points in cluster C_k.

For example, if cluster C_1 consists of the points $(1, 4), (0, 4), (6, 2)$, the centroid μ_1 is calculated as:

$$\mu_1 = \left(\frac{1+0+6}{3}, \frac{4+4+2}{3} \right) = (2.33, 3.33)$$

Code Section

```
# Function to compute the centroid of each cluster
def compute_centroid(cluster_points):
    return np.mean(cluster_points, axis=0)

# Compute centroids for the initial clusters
centroids = np.array([compute_centroid(data[cluster_labels == k]) for k
↪  in range(2)])

# Print initial centroids
print("Initial Centroids:\n", centroids)
```

Image Section

```
[15]: # Function to compute the centroid of each cluster
def compute_centroid(cluster_points):
    return np.mean(cluster_points, axis=0)

# Compute centroids for the initial clusters
centroids = np.array([compute_centroid(data[cluster_labels == k]) for k in range(2)])
print("Initial Centroids:\n", centroids)

Initial Centroids:
 [[1.  3. ]
 [3.2 2.2]]
```

Figure 3: Initial Centroids of Clusters

2.5 Step 4: Assign Each Observation to the Nearest Centroid

After computing the centroids, each observation is reassigned to the cluster whose centroid is nearest to it, based on the Euclidean distance.

Mathematical Concept

The Euclidean distance between a point $\mathbf{x}_i = (x_1, y_1)$ and a centroid $\mu_k = (x_2, y_2)$ is calculated as:

$$d(\mathbf{x}_i, \mu_k) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Each point is reassigned to the cluster C_k that minimizes this distance.

Code Section

```
# Reassign points to the nearest centroid
new_cluster_labels = np.array([np.argmin([np.linalg.norm(point - centroids[k]) for k in range(2)]) for point in data])

# Print new cluster assignments
print("New Cluster Labels:", new_cluster_labels)
```

Image Section

```
[41]: # Reassign points to the nearest centroid
new_cluster_labels = np.array([np.argmin([np.linalg.norm(point - centroids[k]) for k in range(2)]) for point in data])
print("New Cluster Labels:", new_cluster_labels)

New Cluster Labels: [0 0 0 1 1 1]
```

Figure 4: Reassigned Cluster Labels Based on Nearest Centroid

2.6 Step 5: Repeat Until Convergence

The steps of recalculating centroids and reassigning points are repeated iteratively until the cluster assignments stabilize, meaning they no longer change. This point of stabilization is referred to as convergence.

Mathematical Criterion for Convergence

Convergence is achieved when the centroids no longer move, and the cluster assignments of the data points do not change between iterations. Mathematically, this can be expressed as:

$$\mathbf{c}^{(t+1)} = \mathbf{c}^{(t)}$$

where $\mathbf{c}^{(t)}$ denotes the centroid vector at the t-th iteration.

Code Section

```
# Iterative K-means steps until convergence
for iteration in range(10):  # Limit iterations to avoid infinite loops
    centroids = np.array([compute_centroid(data[cluster_labels == k]) for k in range(2)])
    new_cluster_labels = np.array([np.argmin([np.linalg.norm(point - centroids[k]) for k in range(2)]) for point in data])

    if np.array_equal(new_cluster_labels, cluster_labels):
        break  # Convergence reached

    cluster_labels = new_cluster_labels

# Final cluster labels and centroids after convergence
print("Final Cluster Labels:", cluster_labels)
print("Final Centroids:\n", centroids)
```

Image Section

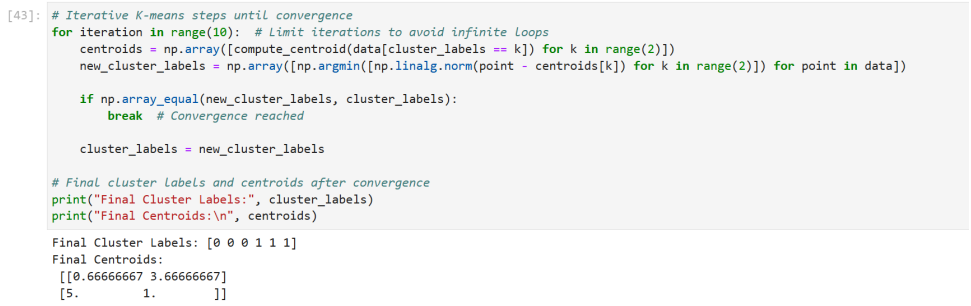


Figure 5: Final Cluster Assignments After Convergence

2.7 Step 6: Visualize Final Clusters

In the final step, we visualize the clusters by coloring the observations according to their final cluster labels.

Code Section

```
# Plot final clusters
plt.subplot(1, 2, 1)  # 2D plot of the final clusters
plt.scatter(data[:, 0], data[:, 1], c=cluster_labels)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='x')
plt.title('Final Cluster Assignments (2D)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.subplot(1, 2, 2)  # 1D plot of the final clusters (projected onto x-axis)
plt.scatter(data[:, 0], np.zeros_like(data[:, 0]), c=cluster_labels)
plt.scatter(centroids[:, 0], np.zeros_like(centroids[:, 0]), c='red', marker='x')
plt.title('Final Cluster Assignments (1D)')
plt.xlabel('Feature 1')

plt.tight_layout()
plt.savefig('./images/final_clusters_2D_1D.jpg')  # Save the combined 2D and 1D clusters as a JPG file
plt.show()
```

Image Section

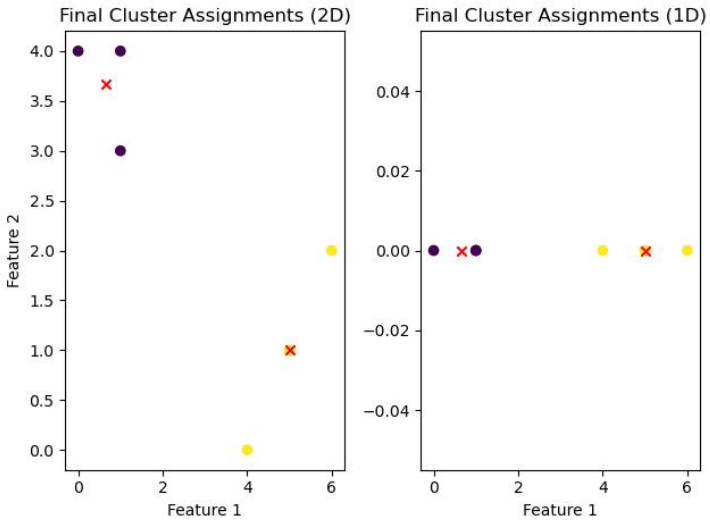


Figure 6: Final Cluster Visualization

K-means clustering is an iterative and unsupervised learning algorithm designed to partition a dataset into K distinct clusters. Through repeated reassignment and centroid recalculation, the algorithm minimizes the within-cluster variance. The K-means clustering process has successfully grouped the observations into two clusters. The centroids calculated, and the final cluster assignments remained consistent after one iteration, indicating convergence.

3 Question 03

Consider a 2-D data set having two types of data point class, namely X1 and X2. Given, $X1 = (x_1, x_2) = (4, 1), (2, 4), (2, 3), (3, 6), (4, 4)$ and $X2 = (x_1, x_2) = (9, 10), (6, 8), (9, 5), (8, 7), (10, 8)$.

(a) Apply Linear Discriminant Analysis in the view of dimensionality reduction.

(b) Plot the graphs if required.

(c) Write advantages, disadvantages, and applications of Linear Discriminant Analysis (LDA).

3.1 Answer:

Given:

We have two sets of 2D data points corresponding to two different classes $X1$ and $X2$:

$$X1 = \begin{pmatrix} 4 & 1 \\ 2 & 4 \\ 2 & 3 \\ 3 & 6 \\ 4 & 4 \end{pmatrix}, \quad X2 = \begin{pmatrix} 9 & 10 \\ 6 & 8 \\ 9 & 5 \\ 8 & 7 \\ 10 & 8 \end{pmatrix}$$

To Find:

- Apply Linear Discriminant Analysis (LDA) for dimensionality reduction.
- Plot the results, if necessary.
- Discuss the advantages, disadvantages, and applications of LDA.

Solution:

3.2 Step 1: Compute the Mean Vectors

The first step in LDA is to compute the mean vector for each class. The mean vector represents the average position of the data points within each class in the feature space.

Mathematical Formulation

The mean vector μ_k for class X_k is calculated as:

$$\mu_k = \frac{1}{|X_k|} \sum_{\mathbf{x}_i \in X_k} \mathbf{x}_i$$

For example, the mean vector for class X_1 is:

$$\mu_1 = \frac{1}{5}\left((4,1) + (2,4) + (2,3) + (3,6) + (4,4)\right) = (3, 3.6)$$

Similarly, for class X_2:

$$\mu_2 = \frac{1}{5}\left((9,10) + (6,8) + (9,5) + (8,7) + (10,8)\right) = (8.4, 7.6)$$

Code Section

```
import numpy as np

# Define the dataset
X1 = np.array([[4, 1], [2, 4], [2, 3], [3, 6], [4, 4]])
X2 = np.array([[9, 10], [6, 8], [9, 5], [8, 7], [10, 8]])

# Compute the means of each class
mean_X1 = np.mean(X1, axis=0)
mean_X2 = np.mean(X2, axis=0)

print("Mean of X1:", mean_X1)
print("Mean of X2:", mean_X2)
```

Image Section

```
[20]: import numpy as np

      # Define the dataset
      X1 = np.array([[4, 1], [2, 4], [2, 3], [3, 6], [4, 4]])
      X2 = np.array([[9, 10], [6, 8], [9, 5], [8, 7], [10, 8]])

      # Compute the means of each class
      mean_X1 = np.mean(X1, axis=0)
      mean_X2 = np.mean(X2, axis=0)

      print("Mean of X1:", mean_X1)
      print("Mean of X2:", mean_X2)

Mean of X1: [3.  3.6]
Mean of X2: [8.4 7.6]
```

Figure 7: Computed Mean Vectors for X1 and X2

3.3 Step 2: Compute the Scatter Matrices

LDA involves computing the within-class scatter matrix S_W and the between-class scatter matrix S_B. These matrices capture the variance within each class and the variance between the classes, respectively.

Mathematical Formulation

The within-class scatter matrix S_W is given by:

$$S_W = \sum_{k=1}^{2} \sum_{\mathbf{x}_i \in X_k} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^\top$$

The between-class scatter matrix S_B is defined as:

$$S_B = \sum_{k=1}^{2} N_k (\mu_k - \mu)(\mu_k - \mu)^\top$$

where μ is the overall mean vector:

$$\mu = \frac{N_1 \mu_1 + N_2 \mu_2}{N_1 + N_2}$$

Code Section

```
# Compute the within-class scatter matrix
S_W = np.zeros((2, 2))
for x in X1:
    S_W += np.dot((x - mean_X1).reshape(2, 1), (x - mean_X1).reshape(1, 2))
for x in X2:
    S_W += np.dot((x - mean_X2).reshape(2, 1), (x - mean_X2).reshape(1, 2))

# Compute the between-class scatter matrix
mean_overall = np.mean(np.vstack((X1, X2)), axis=0)
S_B = np.dot((mean_X1 - mean_overall).reshape(2, 1), (mean_X1 - mean_overall).reshape(1, 2)) * X1.shape[0] + \
      np.dot((mean_X2 - mean_overall).reshape(2, 1), (mean_X2 - mean_overall).reshape(1, 2)) * X2.shape[0]

print("Within-class Scatter Matrix SW:\n", S_W)
print("Between-class Scatter Matrix SB:\n", S_B)
```

Image Section

```
[22]: # Compute the within-class scatter matrix
S_W = np.zeros((2, 2))
for x in X1:
    S_W += np.dot((x - mean_X1).reshape(2, 1), (x - mean_X1).reshape(1, 2))
for x in X2:
    S_W += np.dot((x - mean_X2).reshape(2, 1), (x - mean_X2).reshape(1, 2))

# Compute the between-class scatter matrix
mean_overall = np.mean(np.vstack((X1, X2)), axis=0)
S_B = np.dot((mean_X1 - mean_overall).reshape(2, 1), (mean_X1 - mean_overall).reshape(1, 2)) * X1.shape[0] + \
      np.dot((mean_X2 - mean_overall).reshape(2, 1), (mean_X2 - mean_overall).reshape(1, 2)) * X2.shape[0]

print("Within-class Scatter Matrix SW:\n", S_W)
print("Between-class Scatter Matrix SB:\n", S_B)
```

```
Within-class Scatter Matrix SW:
 [[13.2 -2.2]
 [-2.2 26.4]]
Between-class Scatter Matrix SB:
 [[72.9 54. ]
 [54.  40. ]]
```

Figure 8: Within-class and Between-class Scatter Matrices

3.4 Step 3: Compute the LDA Projection

The next step involves finding the linear discriminant that maximizes the ratio of the between-class scatter to the within-class scatter. This discriminant is the direction that best separates the two classes.

Mathematical Formulation

The linear discriminant $\mathbf{w}$ is obtained by solving the following eigenvalue problem:

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

We choose the eigenvector $\mathbf{w}$ corresponding to the largest eigenvalue λ.

Code Section

```
# Solve the eigenvalue problem for the matrix SW^-1 * SB
eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))

# Select the eigenvector with the largest eigenvalue
w = eig_vecs[:, np.argmax(eig_vals)]

print("Linear Discriminant Vector w:", w)
```

Image Section

```
[24]: # Solve the eigenvalue problem for the matrix SW^-1 * SB
eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))

# Select the eigenvector with the largest eigenvalue
w = eig_vecs[:, np.argmax(eig_vals)]

print("Linear Discriminant Vector w:", w)
```

```
Linear Discriminant Vector w: [0.91955932 0.39295122]
```

Figure 9: Linear Discriminant Vector $\mathbf{w}$

3.5 Step 4: Project the Data onto the Linear Discriminant

Once the linear discriminant vector $\mathbf{w}$ is identified, the data points are projected onto this vector to achieve dimensionality reduction.

Mathematical Concept

The projection of a data point $\mathbf{x}$ onto the linear discriminant $\mathbf{w}$ is computed as:

$$y = \mathbf{w}^\top \mathbf{x}$$

This projection reduces the dimensionality from 2D to 1D along the direction that maximizes the separation between the two classes.

Code Section

```
# Project the data onto the linear discriminant
X1_proj = X1.dot(w)
X2_proj = X2.dot(w)

print("Projected Data X1:", X1_proj)
print("Projected Data X2:", X2_proj)
```

Image Section

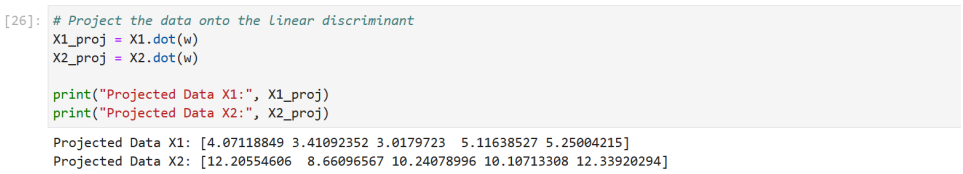


Figure 10: Projection of Data onto Linear Discriminant

3.6 Step 5: Discuss the Advantages, Disadvantages, and Applications of LDA

Advantages:

- LDA maximizes class separability by projecting data onto a lower-dimensional space, making it particularly effective for classification tasks.
- It works well with linearly separable data, providing clear decision boundaries between classes.
- LDA is computationally efficient and straightforward to implement, making it suitable for high-dimensional datasets.

Disadvantages:

- LDA assumes that classes have identical covariance matrices, which may not always hold true in real-world datasets.
- It is not effective for non-linearly separable data, as it can only create linear decision boundaries.
- LDA can be sensitive to outliers, which can significantly affect the calculated discriminant.

Applications:

- LDA is widely used in pattern recognition and face recognition, where distinguishing between different classes is crucial.
- It is also used for dimensionality reduction before applying other machine learning algorithms, helping to improve performance and reduce computational complexity.
- LDA is applied in various classification tasks, especially when the number of features is large compared to the number of samples, such as in text classification and bioinformatics.

4 Question 04

For the **Wine Quality Data Set**, convert all the values in the quality attribute to 0 (bad) if the value is less than or equal to 6 and to 1 (good) otherwise. Normalize all the other attributes between 0 and 1 by min-max scaling. Mention why we use min-max scaling.

4.1 Answer:

Given:

We are required to work with the Wine Quality Data Set, where the goal is to convert the quality attribute values and normalize the other attributes using min-max scaling.

To Find:

- Convert the quality attribute values based on the condition given.
- Normalize the remaining features using min-max scaling.

Solution:

4.1.1 Step 1: Load the Wine Dataset

We begin by loading the wine dataset using pandas:

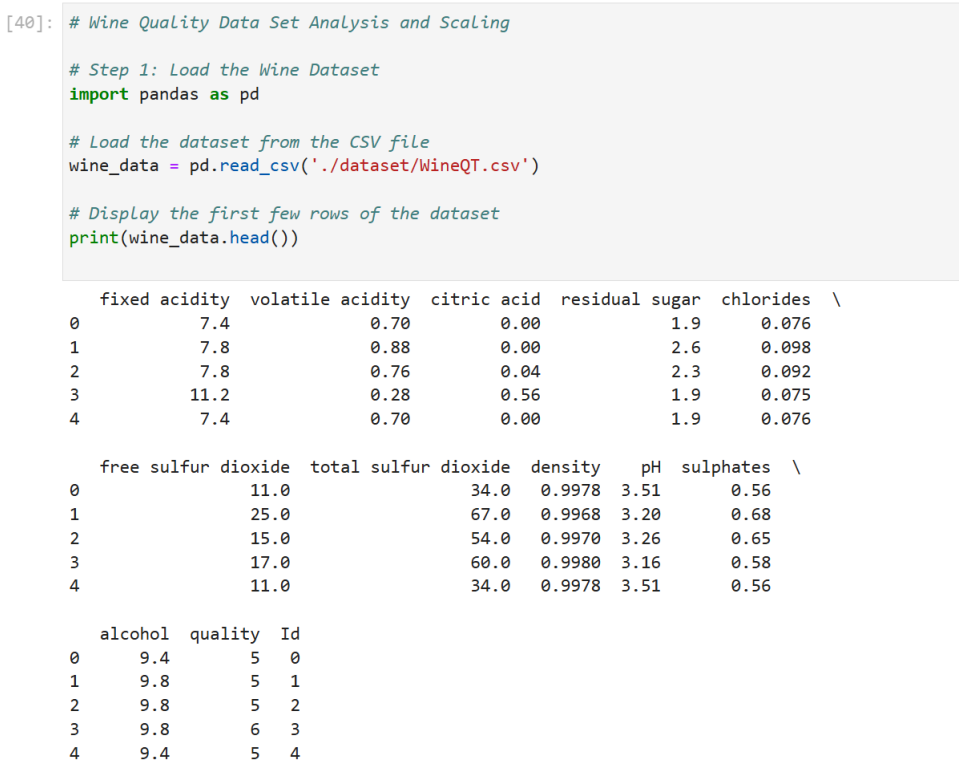
```
import pandas as pd

# Load the dataset from the CSV file
df = pd.read_csv('./dataset/WineQT.csv')

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

The dataset consists of multiple attributes related to wine characteristics, and a target attribute representing the wine quality class.

Image Section



[40]:
```
# Wine Quality Data Set Analysis and Scaling

# Step 1: Load the Wine Dataset
import pandas as pd

# Load the dataset from the CSV file
wine_data = pd.read_csv('./dataset/WineQT.csv')

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

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality  Id
0      9.4        5   0
1      9.8        5   1
2      9.8        5   2
3      9.8        6   3
4      9.4        5   4
```

Figure 11: First Five Rows of Wine Quality Dataset

4.1.2 Step 2: Convert the Quality Attribute

Next, we convert the quality attribute values to 0 (bad) if the value is less than or equal to 6 and to 1 (good) otherwise.

```
# Convert quality attribute
df['quality'] = df['quality'].apply(lambda x: 0 if x <= 6 else 1)

# Display the first few rows after conversion
print(df[['quality']].head())
```

This code converts the wine quality based on the conditions provided.

Image Section

Figure 12: Quality Attribute After Conversion

4.1.3 Step 3: Normalize the Attributes using Min-Max Scaling

We normalize the other attributes between 0 and 1 using min-max scaling. Min-max scaling transforms features to a fixed range [0, 1].

$$\text{Min-Max Scaling Formula: } x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Scale all features except for 'quality'
scaled_features =
↪  scaler.fit_transform(wine_data.drop(columns=['quality']))

# Convert the scaled features back to a DataFrame
df_scaled = pd.DataFrame(scaled_features, columns=wine_data.columns[:-1])

# Add the 'quality' column back to the scaled DataFrame
df_scaled['quality'] = wine_data['quality']

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

This code uses ‘MinMaxScaler‘ from ‘sklearn.preprocessing‘ to scale the data.

Image Section

```
[38]: from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Scale all features except for 'quality'
scaled_features = scaler.fit_transform(wine_data.drop(columns=['quality']))

# Convert the scaled features back to a DataFrame
df_scaled = pd.DataFrame(scaled_features, columns=wine_data.columns[:-1])

# Add the 'quality' column back to the scaled DataFrame
df_scaled['quality'] = wine_data['quality']

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

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0       0.247788          0.397260         0.00        0.068493   0.106845
1       0.283186          0.520548         0.00        0.116438   0.143573
2       0.283186          0.438356         0.04        0.095890   0.133556
3       0.584071          0.109589         0.56        0.068493   0.105175
4       0.247788          0.397260         0.00        0.068493   0.106845

   free sulfur dioxide  total sulfur dioxide   density        pH  sulphates  \
0             0.149254              0.098940  0.567548  0.606299   0.137725
1             0.358209              0.215548  0.494126  0.362205   0.209581
2             0.208955              0.169611  0.508811  0.409449   0.191617
3             0.238806              0.190813  0.582232  0.330709   0.149701
4             0.149254              0.098940  0.567548  0.606299   0.137725

    alcohol  quality
0  0.153846        0
1  0.215385        0
2  0.215385        0
3  0.215385        0
4  0.153846        0
```

Figure 13: Scaled Features and Quality Attribute

4.1.4 Step 4: Plot the Data Before and After Scaling

To visualize the effect of min-max scaling, we plot the data before and after scaling:

```
import matplotlib.pyplot as plt

# Plotting before scaling
plt.figure(figsize=(14, 8))

# Plotting the data before scaling
plt.subplot(1, 2, 1)
plt.boxplot(df.drop(columns=['quality']).values)
plt.title('Before Scaling')
plt.xticks(range(1, len(df.columns)), df.columns.drop('quality'),
 ↪  rotation=90)

# Plotting after scaling
plt.subplot(1, 2, 2)
plt.boxplot(df_scaled.drop(columns=['quality']).values)
plt.title('After Min-Max Scaling')
plt.xticks(range(1, len(df_scaled.columns)),
 ↪  df_scaled.columns.drop('quality'), rotation=90)

plt.tight_layout()
```

```
19 plt.show()
```

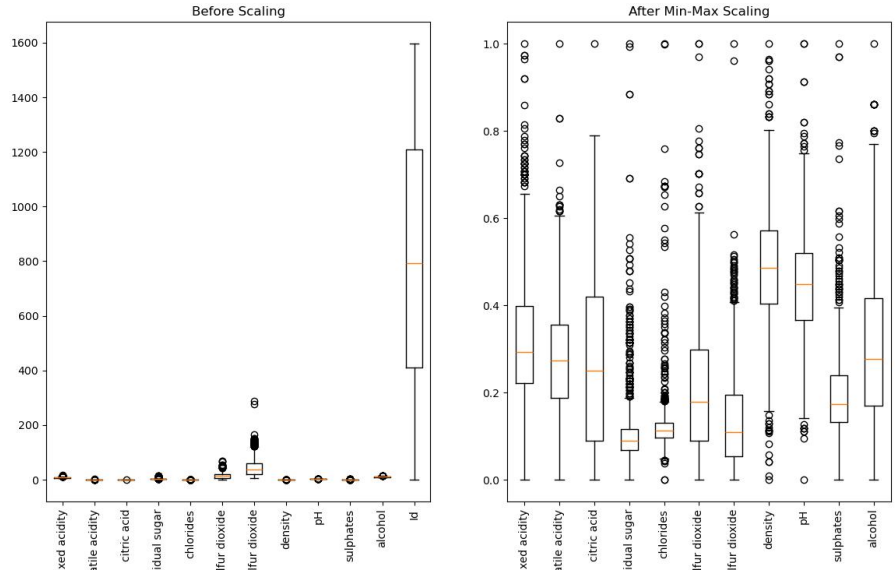


Figure 14: Comparison of Data Distribution Before and After Scaling

These boxplots show the distribution of the data before and after scaling. After min-max scaling, all features are in the range [0, 1].

4.1.5 Step 5: Explain Why We Use Min-Max Scaling

Why Use Min-Max Scaling?

- Min-max scaling is essential when features have different units or scales, which can bias the model's performance.
- It ensures that all features contribute equally to the distance calculations in algorithms such as K-means clustering or nearest neighbors.
- Min-max scaling is particularly useful when the model is sensitive to the magnitude of features, such as in gradient descent optimization.

Result: The quality attribute has been successfully converted, and the remaining features have been normalized using min-max scaling.

4.1.6 GitHub Link

Please refer to the following GitHub repository for the complete code and output in the `.ipynb` file:

GitHub Link

Note: Please check the Notebook filename: `NitinMane_M24MT004_Assignment01.ipynb`

5 Question 05

What is the difference between model parameters and model hyperparameters? What is meant by hyperparameter tuning? Name some common hyperparameters used in clustering algorithms.

5.1 Answer:

5.1.1 Model Parameters vs. Hyperparameters

The following table highlights the key differences between model parameters and model hyperparameters:

Model Parameters	**Model Hyperparameters**
Learned from the data during the training process.	Set before the training process begins.
Examples include weights in a neural network, coefficients in linear regression.	Examples include learning rate, number of layers in a neural network, number of clusters in K-means.
Influence the predictions made by the model.	Influence how the model is trained or the model structure itself.
Optimized by algorithms such as gradient descent.	Optimized through methods like grid search, random search, or Bayesian optimization.
Typically have values that can be adjusted continuously.	Often require discrete values or categorical choices.
Directly impact the model's accuracy and performance.	Impact the model's training time, complexity, and generalization ability.

5.1.2 Hyperparameter Tuning

Definition:

Hyperparameter tuning is the process of finding the optimal set of hyperparameters for a machine learning model. Unlike model parameters, which are learned during the training process, hyperparameters are set before the training process begins. The goal of hyperparameter tuning is to improve the model's performance by selecting the best possible values for these hyperparameters.

Methods for Hyperparameter Tuning:

- **Grid Search:** A systematic approach where all possible combinations of hyperparameters are tried, and the best one is selected based on model performance.
- **Random Search:** A more efficient approach where random combinations of hyperparameters are tested, and the best one is selected.
- **Bayesian Optimization:** A probabilistic model-based approach that aims to find the best hyperparameters more efficiently by modeling the performance as a function of the hyperparameters.

5.1.3 Prevalent Hyperparameters in Clustering Algorithms

In the domain of clustering algorithms, several hyperparameters are instrumental in determining the performance and outcomes of these methods:

- **Number of Clusters (K):** This hyperparameter specifies the predetermined number of clusters that the algorithm, such as K-means, is required to partition the data into.
- **Distance Metric:** This parameter delineates the metric employed to measure distances between data points, encompassing methods such as Euclidean distance or Manhattan distance.
- **Initialization Method:** The strategy utilized for initializing the centroids in K-means, with options including 'random' or 'k-means++'.
- **Maximum Iterations:** This defines the upper bound on the number of iterations the algorithm will execute prior to termination.
- **Convergence Tolerance:** The specified threshold at which the algorithm ceases execution if the alterations in centroids fall below this value.
- **Linkage Criterion (in Hierarchical Clustering):** The criterion governing the amalgamation of clusters, examples of which include 'single', 'complete', or 'average' linkage.

6 Question 06

You are given three observed signals $x_1(t)$, $x_2(t)$, and $x_3(t)$ which are linear mixtures of three independent source signals $s_1(t)$, $s_2(t)$, and $s_3(t)$. The mixing process is represented by the following matrix equation:

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t)$$

where

$$\mathbf{x}(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{pmatrix}, \quad \mathbf{s}(t) = \begin{pmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

where $\mathbf{A}$ is an unknown 3×3 mixing matrix, and $\mathbf{s}(t)$ represents the independent source signals.

6.1 (a) Explain the steps involved in estimating the mixing matrix A and the independent source signals $\mathbf{s}(t)$ using ICA.

Answer:

6.1.1 Introduction to ICA

Independent Component Analysis (ICA) is a computational technique used to separate a multivariate signal into additive independent components. It is based on the assumption that the observed signals are linear mixtures of independent source signals. The objective is to estimate both the mixing matrix $\mathbf{A}$ and the original source signals $\mathbf{s}(t)$ from the observed signals $\mathbf{x}(t)$.

6.1.2 Step 1: Centering the Data

Centering the data is the first step in ICA, which involves subtracting the mean from each observed signal to ensure that the data has a zero mean. This simplifies subsequent calculations:

$$\mathbf{x}_{\text{centered}}(t) = \mathbf{x}(t) - \mathbb{E}[\mathbf{x}(t)]$$

Here, $\mathbb{E}[\mathbf{x}(t)]$ is the mean of the observed signals over time.

6.1.3 Step 2: Whitening the Data

Whitening transforms the centered data so that the components are uncorrelated and have unit variance. Whitening is done by calculating the covariance matrix of the centered data:

$$\mathbf{C}_{\mathbf{x}} = \mathbb{E}[\mathbf{x}_{\text{centered}}(t)\mathbf{x}_{\text{centered}}^{\top}(t)]$$

Next, we perform an eigenvalue decomposition of the covariance matrix:

$$\mathbf{C}_{\mathbf{x}} = \mathbf{E}\mathbf{D}\mathbf{E}^{\top}$$

where $\mathbf{E}$ is the matrix of eigenvectors, and $\mathbf{D}$ is the diagonal matrix of eigenvalues. The whitening transformation is then applied:

$$\mathbf{z}(t) = \mathbf{D}^{-1/2}\mathbf{E}^{\top}\mathbf{x}_{\text{centered}}(t)$$

This transformation ensures that $\mathbf{z}(t)$ is uncorrelated with unit variance.

6.1.4 Step 3: Estimating the Mixing Matrix A

The key goal of ICA is to estimate the mixing matrix $\mathbf{A}$. This is done by finding a matrix $\mathbf{W}$ such that the components of $\mathbf{W}\mathbf{z}(t)$ are as independent as possible. The FastICA algorithm is a commonly used method, which involves the following steps:

1. **Choose an initial guess** for the matrix $\mathbf{W}$.
2. **Iteratively update W** using the following rule:

$$\mathbf{w}_{\text{new}} = \mathbb{E}[\mathbf{z}(t)g(\mathbf{w}^{\top}\mathbf{z}(t))] - \mathbb{E}[g'(\mathbf{w}^{\top}\mathbf{z}(t))]\mathbf{w}$$

   where $g(\cdot)$ is a non-linear function (e.g., $g(u) = \tanh(u)$), and $g'(u)$ is its derivative.
3. **Normalize** $\mathbf{w}_{\text{new}}$ to have unit norm.
4. **Check for convergence**: If the change in $\mathbf{W}$ is below a threshold, the algorithm has converged.

The resulting matrix $\mathbf{W}$ is the estimated unmixing matrix, and its inverse gives the estimated mixing matrix:

$$\mathbf{A} \approx \mathbf{W}^{-1}$$

6.1.5 Step 4: Recovering the Source Signals $\mathbf{s}(t)$

Once the unmixing matrix $\mathbf{W}$ is estimated, the independent source signals $\mathbf{s}(t)$ can be recovered by:

$$\mathbf{s}(t) = \mathbf{W}\mathbf{z}(t)$$

Since $\mathbf{z}(t) = \mathbf{D}^{-1/2}\mathbf{E}^\top \mathbf{x}_{\text{centered}}(t)$, the source signals can be expressed as:

$$\mathbf{s}(t) = \mathbf{W}\mathbf{D}^{-1/2}\mathbf{E}^\top \left(\mathbf{x}(t) - \mathbb{E}[\mathbf{x}(t)]\right)$$

Thus, $\mathbf{s}(t)$ is recovered by applying the estimated unmixing matrix to the whitened, centered data.

The process of ICA involves centering and whitening the observed data, estimating the mixing matrix using FastICA, and recovering the independent source signals. This process hinges on the statistical independence and non-Gaussianity of the source signals.

6.2 (b) Suppose the mixing matrix A is not full rank (i.e., it is singular or nearly singular). How would this affect the ICA process? Discuss the potential challenges and the implications for recovering the original source signals.

Answer:

6.2.1 Implications of a Non-Full Rank Mixing Matrix

A non-full rank mixing matrix $\mathbf{A}$ significantly complicates the ICA process. Here's why:

- **Underdetermined System:** A non-full rank matrix $\mathbf{A}$ implies that the system $\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t)$ is underdetermined, meaning there are more unknowns (source signals) than equations (observed signals). This makes it impossible to uniquely solve for the source signals $\mathbf{s}(t)$.
- **Loss of Information:** A singular matrix $\mathbf{A}$ indicates that some of the original source information has been lost during the mixing process, making it impossible to fully recover the original signals. This loss of information typically occurs when distinct source signals are projected onto the same or nearly the same direction in the observed space.
- **Inversion Issues:** The ICA process requires the inversion of the mixing matrix $\mathbf{A}$ to recover the source signals. If $\mathbf{A}$ is singular or nearly singular, it cannot be inverted, which prevents the direct recovery of the source signals.
- **Dependence Among Sources:** A non-full rank $\mathbf{A}$ suggests that the source signals are not truly independent, violating one of the core assumptions of ICA. Without independence, the algorithm cannot effectively separate the mixed signals.

6.2.2 Challenges and Practical Implications

- **Algorithmic Instability:** Near-singular matrices can lead to numerical instability during the inversion process, causing the algorithm to produce inaccurate or completely erroneous results.
- **Limited Signal Separation:** When $\mathbf{A}$ is not full rank, the separation of signals is compromised. The ICA algorithm may fail to separate the signals entirely, resulting in outputs that are mixtures or combinations of the original source signals.
- **Preprocessing with PCA:** In cases where the mixing matrix is nearly singular, preprocessing with techniques like Principal Component Analysis (PCA) can help reduce the dimensionality and mitigate some of the challenges, although it may not completely resolve the issue.
- **Increased Computational Complexity:** Dealing with a non-full rank matrix often requires additional computational steps, such as regularization or dimensionality reduction, which increases the overall complexity of the ICA process.

The success of ICA relies heavily on the assumption that the mixing matrix $\mathbf{A}$ is full-rank and invertible. If this condition is not met, the separation of the independent source signals becomes highly challenging, if not impossible. In such cases, additional preprocessing steps or alternative methods may be required to achieve any meaningful separation of the signals.

Answer:

6.2.3 Introduction to ICA

Independent Component Analysis (ICA) is a computational technique used to separate a multivariate signal into additive, independent components. The central task is to estimate both the mixing matrix $\mathbf{A}$ and the source signals $\mathbf{s}(t)$ from the observed signals $\mathbf{x}(t)$.

Given the linear relationship:

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t)$$

where $\mathbf{A}$ is an unknown mixing matrix, the goal of ICA is to recover $\mathbf{s}(t)$ and estimate $\mathbf{A}$ under the assumption that the components of $\mathbf{s}(t)$ are statistically independent.

6.2.4 Step 1: Centering the Data

The first step in ICA is to center the observed data $\mathbf{x}(t)$. Centering involves subtracting the mean of the data to ensure that it has a zero mean. This can be mathematically expressed as:

$$\mathbf{x}_{\text{centered}}(t) = \mathbf{x}(t) - \mathbb{E}[\mathbf{x}(t)]$$

where $\mathbb{E}[\mathbf{x}(t)]$ is the expected value (mean) of the observed signals over time. Centering simplifies the problem by removing the mean, which has no effect on the independent components we aim to recover.

6.2.5 Step 2: Whitening the Data

After centering, the next step is whitening the data. Whitening transforms the centered data so that the components are uncorrelated and have unit variance. Whitening is crucial for simplifying the ICA process because it reduces the complexity of the problem by working with a simpler form of the data.

The covariance matrix of the centered data is computed as:

$$\mathbf{C}_\mathbf{x} = \mathbb{E}[\mathbf{x}_{\text{centered}}(t)\mathbf{x}_{\text{centered}}^\top(t)]$$

Next, we perform an eigenvalue decomposition of the covariance matrix:

$$\mathbf{C}_\mathbf{x} = \mathbf{E}\mathbf{D}\mathbf{E}^\top$$

where $\mathbf{E}$ is the matrix of eigenvectors and $\mathbf{D}$ is the diagonal matrix of eigenvalues.

The whitening transformation is then applied:

$$\mathbf{z}(t) = \mathbf{D}^{-1/2}\mathbf{E}^\top\mathbf{x}_{\text{centered}}(t)$$

where $\mathbf{D}^{-1/2}$ scales the components to have unit variance, and $\mathbf{E}^\top$ rotates the data such that the covariance matrix of $\mathbf{z}(t)$ is the identity matrix.

Detailed Analysis: Whitening simplifies the ICA problem by ensuring that the components of $\mathbf{z}(t)$ are uncorrelated and standardized. This reduces the number of variables the ICA algorithm needs to estimate, focusing solely on the rotation required to achieve independence among the components.

6.2.6 Step 3: Estimating the Mixing Matrix A

The core of ICA is estimating the mixing matrix $\mathbf{A}$. This is done by finding a matrix $\mathbf{W}$ such that the components of $\mathbf{W}\mathbf{z}(t)$ are as independent as possible.

Using the FastICA algorithm, the following steps are performed:

1. **Initialize W** with a random guess.
2. **Iterative Update**: Update $\mathbf{W}$ using:

   $$\mathbf{w}_{\text{new}} = \mathbb{E}[\mathbf{z}(t)g(\mathbf{w}^\top\mathbf{z}(t))] - \mathbb{E}[g'(\mathbf{w}^\top\mathbf{z}(t))]\mathbf{w}$$

   where $g(\cdot)$ is a non-linear function, commonly $g(u) = \tanh(u)$, and $g'(u)$ is its derivative.
3. **Normalize** $\mathbf{w}_{\text{new}}$ to have unit norm.
4. **Check Convergence**: The process continues until the change in $\mathbf{W}$ is sufficiently small.

The estimated unmixing matrix $\mathbf{W}$ is related to the mixing matrix by:

$$\mathbf{A} \approx \mathbf{W}^{-1}$$

Detailed Analysis: The iterative update maximizes the non-Gaussianity of the projections of $\mathbf{z}(t)$, leading to the estimation of independent components. The non-linear function $g(\cdot)$ plays a critical role in ensuring that the recovered components are independent. The choice of $g(\cdot)$ depends on the specific properties of the data and the desired statistical independence.

6.2.7 Step 4: Recovering the Source Signals $\mathbf{s}(t)$

Once the unmixing matrix $\mathbf{W}$ is estimated, the independent source signals $\mathbf{s}(t)$ can be recovered by:

$$\mathbf{s}(t) = \mathbf{W}\mathbf{z}(t)$$

Since $\mathbf{z}(t) = \mathbf{D}^{-1/2}\mathbf{E}^\top \mathbf{x}_{\text{centered}}(t)$, the source signals can be expressed as:

$$\mathbf{s}(t) = \mathbf{W}\mathbf{D}^{-1/2}\mathbf{E}^\top \left(\mathbf{x}(t) - \mathbb{E}[\mathbf{x}(t)]\right)$$

Detailed Analysis: The final recovery of the source signals $\mathbf{s}(t)$ involves applying the estimated unmixing matrix $\mathbf{W}$ to the whitened data. This step completes the ICA process by providing the best estimate of the original independent signals.

The ICA process involves centering the data, whitening the signals to remove correlations and standardize the variance, estimating the mixing matrix using the FastICA algorithm, and finally recovering the independent source signals. Each step is crucial for accurately separating the observed mixtures into their original independent components.