Lab6

November 19, 2024

0.0.1 Question 1

[1]:

```python
import math

def get_coordinates():
    """Gets 3D coordinates from the user."""
    x1 = float(input("Enter x-coordinate of point 1: "))
    y1 = float(input("Enter y-coordinate of point 1: "))
    z1 = float(input("Enter z-coordinate of point 1: "))
    x2 = float(input("Enter x-coordinate of point 2: "))
    y2 = float(input("Enter y-coordinate of point 2: "))
    z2 = float(input("Enter z-coordinate of point 2: "))
    return ((x1, y1, z1), (x2, y2, z2))

def euclidean_distance(point1, point2):
    """Calculates the Euclidean distance between two 3D points."""
    x1, y1, z1 = point1
    x2, y2, z2 = point2
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
    return distance

def manhattan_distance(point1, point2):
    """Calculates the Manhattan distance between two 3D points."""
    x1, y1, z1 = point1
    x2, y2, z2 = point2
    distance = abs(x2 - x1) + abs(y2 - y1) + abs(z2 - z1)
    return distance

def minkowski_distance(point1, point2, p):
    """Calculates the Minkowski distance between two 3D points."""
    x1, y1, z1 = point1
    x2, y2, z2 = point2
    distance = (abs(x2 - x1)**p + abs(y2 - y1)**p + abs(z2 - z1)**p) ** (1/p)
    return distance

# Get coordinates from the user
point1, point2 = get_coordinates()
```

```
# Calculate and print distances
print('\n==========')
print(f"Euclidean distance: {euclidean_distance(point1, point2):.2f}")
print(f"Manhattan distance: {manhattan_distance(point1, point2):.2f}")
p = float(input("Enter the value of p for Minkowski distance: "))
print(f"Minkowski distance (p = {p}): {minkowski_distance(point1, point2, p):.
 ↪2f}")
```

```
Enter x-coordinate of point 1:  5
Enter y-coordinate of point 1:  7
Enter z-coordinate of point 1:  9
Enter x-coordinate of point 2:  3
Enter y-coordinate of point 2:  6
Enter z-coordinate of point 2:  8


==========
Euclidean distance: 2.45
Manhattan distance: 4.00

Enter the value of p for Minkowski distance:  2

Minkowski distance (p = 2.0): 2.45
```

0.0.2 Question 2

[4]:

```
import pandas as pd
from sklearn.datasets import load_iris
from scipy.spatial.distance import pdist, squareform

# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target
# Calculate the Euclidean distances using pdist and squareform
distances = squareform(pdist(df, metric='euclidean'))

# Print the distance matrix
print("The distance matrix\n==================\n",distances)
```

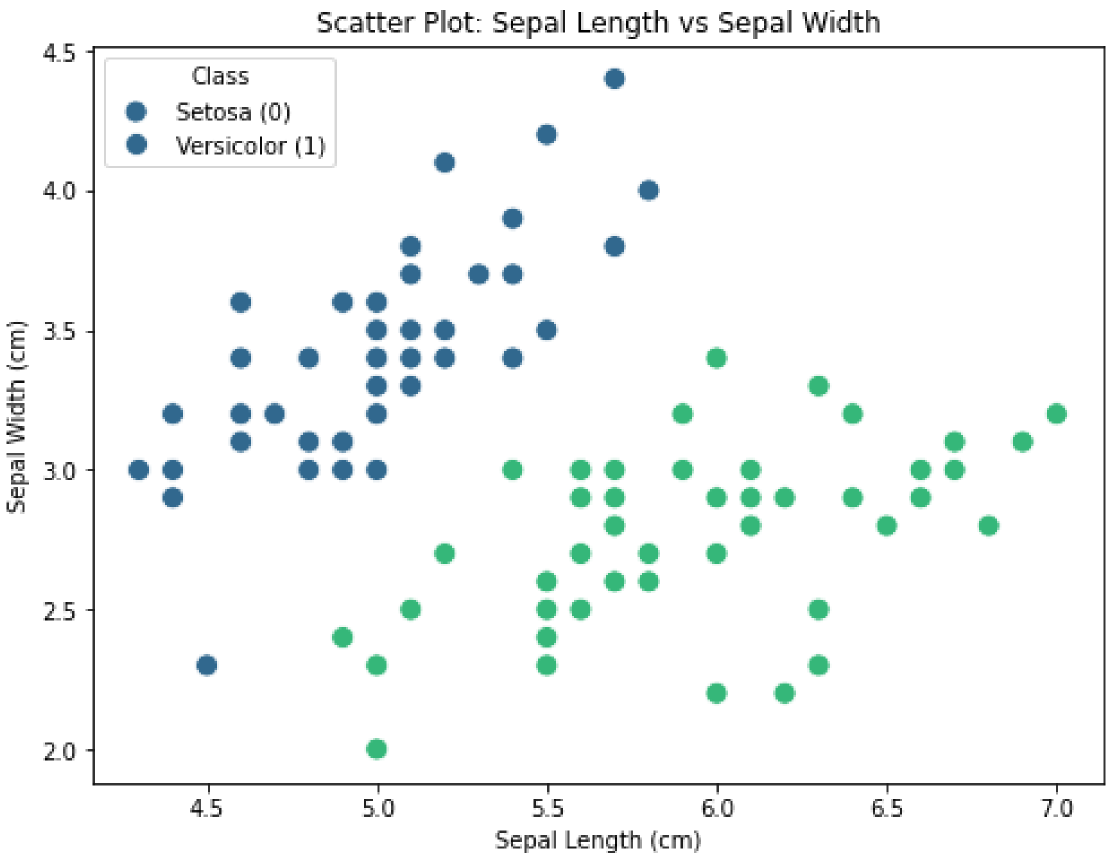
```
The distance matrix
==================
 [[0.         0.53851648 0.50990195 … 4.88773976 5.06260802 4.59782557]
 [0.53851648 0.         0.3        … 4.92341345 5.12445119 4.60977223]
 [0.50990195 0.3        0.         … 5.07247474 5.24499762 4.74130784]
 …
 [4.88773976 4.92341345 5.07247474 … 0.         0.6164414  0.64031242]
 [5.06260802 5.12445119 5.24499762 … 0.6164414  0.         0.76811457]
 [4.59782557 4.60977223 4.74130784 … 0.64031242 0.76811457 0.        ]]
```

0.0.3 Question 3

Method 1

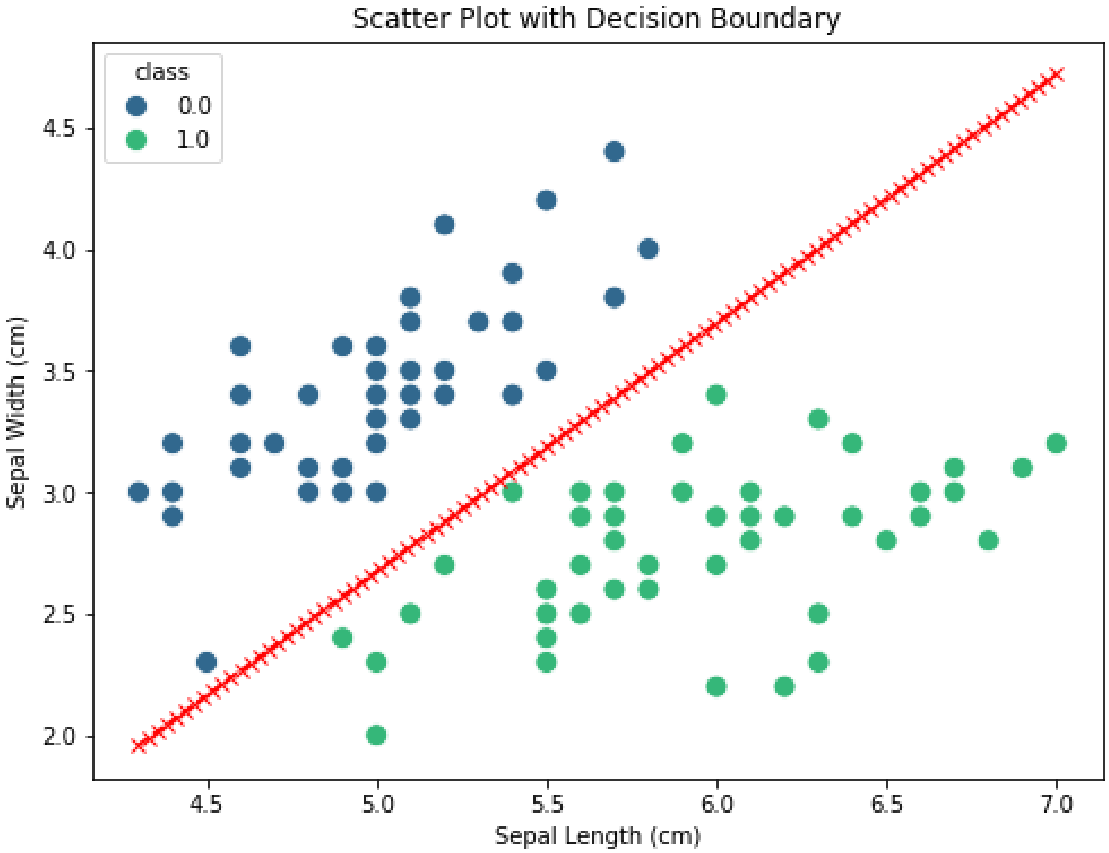
[35]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import seaborn as sns
import pandas as pd

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (input attributes)
y = iris.target  # Labels (output classes)

# Select only the samples from class 0 (setosa) and class 1 (versicolor)
class_0 = X[y == 0]
class_1 = X[y == 1]

# Combine the selected samples
X_binary = np.vstack((class_0, class_1))
y_binary = np.hstack((np.zeros(len(class_0)), np.ones(len(class_1))))

# Create a DataFrame for better visualization
df = pd.DataFrame(X_binary, columns=iris.feature_names)
df['class'] = y_binary

# Plot the scatter plot
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x='sepal length (cm)', y='sepal width (cm)',␣
 ↪hue='class', palette='viridis', s=100)
plt.title('Scatter Plot: Sepal Length vs Sepal Width')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend(title='Class', loc='upper left', labels=['Setosa (0)', 'Versicolor␣
 ↪(1)'])
plt.show()
```



```
[41]: from sklearn.linear_model import LogisticRegression

      # Fit a Logistic Regression model to the data
      model = LogisticRegression()
      model.fit(X_binary[:, :2], y_binary)  # We use only the first two features for␣
       ↪this example

      # Get the slope and intercept of the decision boundary
      slope = -model.coef_[0][0] / model.coef_[0][1]
      intercept = -model.intercept_[0] / model.coef_[0][1]

      # Plot the decision boundary
      xx = np.linspace(X_binary[:, 0].min(), X_binary[:, 0].max(), 100)
      yy = slope * xx + intercept

      # Plot the scatter plot again along with the decision boundary
      plt.figure(figsize=(8,6))
      sns.scatterplot(data=df, x='sepal length (cm)', y='sepal width (cm)',␣
       ↪hue='class', palette='viridis', s=100)
```

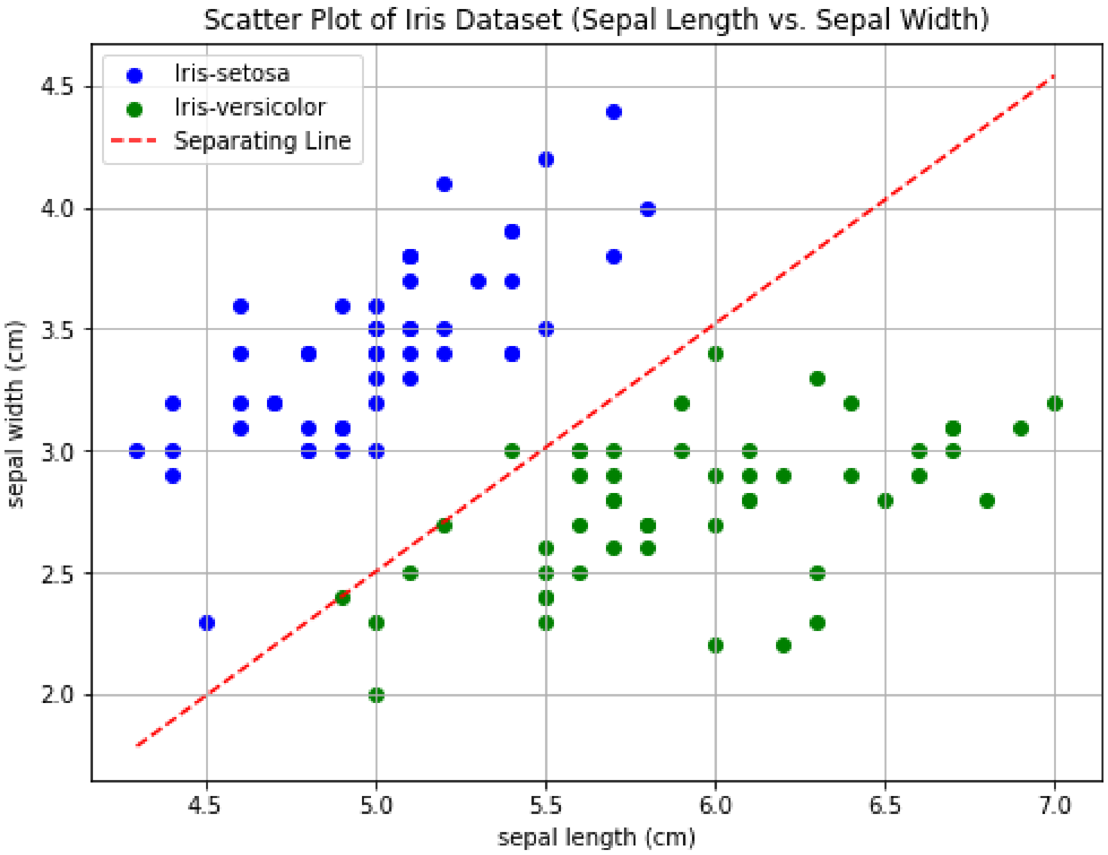
```
plt.plot(xx, yy, color='red', linestyle='--', label=f'Decision Boundary: y =␣
↪{slope:.2f}x + {intercept:.2f}',marker = 'x')
plt.title('Scatter Plot with Decision Boundary')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
# plt.legend(title='Class', loc='upper left', labels=['Setosa (0)', 'Versicolor␣
↪(1)', 'Decision Boundary'])
plt.show()
```



Method 2

[56]:
```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target
```

```
# Select samples belonging to two classes (e.g., Iris-setosa and␣
 ↪Iris-versicolor)
new_df = df[df['target'].isin([0, 1])]

# Select two input attributes (e.g., sepal length and sepal width)
x_attr = 'sepal length (cm)'
y_attr = 'sepal width (cm)'

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(new_df[x_attr][new_df['target'] == 0],␣
 ↪new_df[y_attr][new_df['target'] == 0], label='Iris-setosa', color='blue')
plt.scatter(new_df[x_attr][new_df['target'] == 1],␣
 ↪new_df[y_attr][new_df['target'] == 1], label='Iris-versicolor',␣
 ↪color='green')
plt.xlabel(x_attr)
plt.ylabel(y_attr)
plt.title('Scatter Plot of Iris Dataset (Sepal Length vs. Sepal Width)')
plt.legend()
plt.grid(True)

# Manually find a line to separate the classes (visual estimation)
# Example: A line with equation y = -0.5x + 4
x_line = [new_df[x_attr].min(), new_df[x_attr].max()]
y_line = [1.02* x + -2.6 for x in x_line]
plt.plot(x_line, y_line, color='red', linestyle='--', label='Separating Line')
plt.legend()

plt.show()
```


Scatter Plot of Iris Dataset (Sepal Length vs. Sepal Width)
Iris-setosa
Iris-versicolor
Separating Line
sepal width (cm)
sepal length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0

```
[ ]:
```