Name : Satwik Shirpurwar

Batch : A3

Roll No: 52

Practical 2

Aim: Design and implement a Python program to build a binary classifier using the Perceptron Learning Algorithm. The Iris dataset will be used to train and test the Perceptron Learning Algorithm. Focus on two classes of the Iris dataset (e.g., "Iris-setosa" and "Iris-versicolor") to create a binary classification problem. Use only two features (e.g., "sepal length" and "sepal width") for simplicity and visualization. Measure performance using Accuracy on the test set. Also Visualize the learned decision boundary.

In [5]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

In [6]:

```
# Load the Iris Dataset
df = pd.read_csv("Iris.csv")
```

In [7]:

```
df.head()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [8]:
```
df.tail()
```

Out[8]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [9]:
```
df.shape
```

Out[9]:
```
(150, 6)
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
Id                0
SepalLengthCm     0
SepalWidthCm      0
PetalLengthCm     0
PetalWidthCm      0
Species           0
dtype: int64
```

In [11]:
```
df.describe()
```

Out[11]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
```

In [13]:
```
X = X[y != 2]
y = y[y != 2]
```

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
```

In [15]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Perceptron Algorithm

In [16]:
```
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = 0

    def fit(self, X, y):
        # Initialize weights and bias
        self.weights = np.zeros(X.shape[1])

        # Training loop
        for _ in range(self.n_iterations):
            for xi, target in zip(X, y):
                # Calculate the model output (y_pred)
                update = self.learning_rate * (target - self.predict(xi))

                # Update weights and bias
                self.weights += update * xi
                self.bias += update

    def predict(self, X):
        # Linear combination: weights * inputs + bias
        linear_output = np.dot(X, self.weights) + self.bias
        # Step function to classify
        return np.where(linear_output >= 0.0, 1, 0)

    def score(self, X, y):
        # Calculate accuracy
        predictions = self.predict(X)
        return np.mean(predictions == y)
```

In [17]:
```
model = Perceptron(learning_rate=0.01, n_iterations=1000)
model.fit(X_train, y_train)
```

In [18]:
```
accuracy = model.score(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Test Accuracy: 96.67%
```

In [19]:

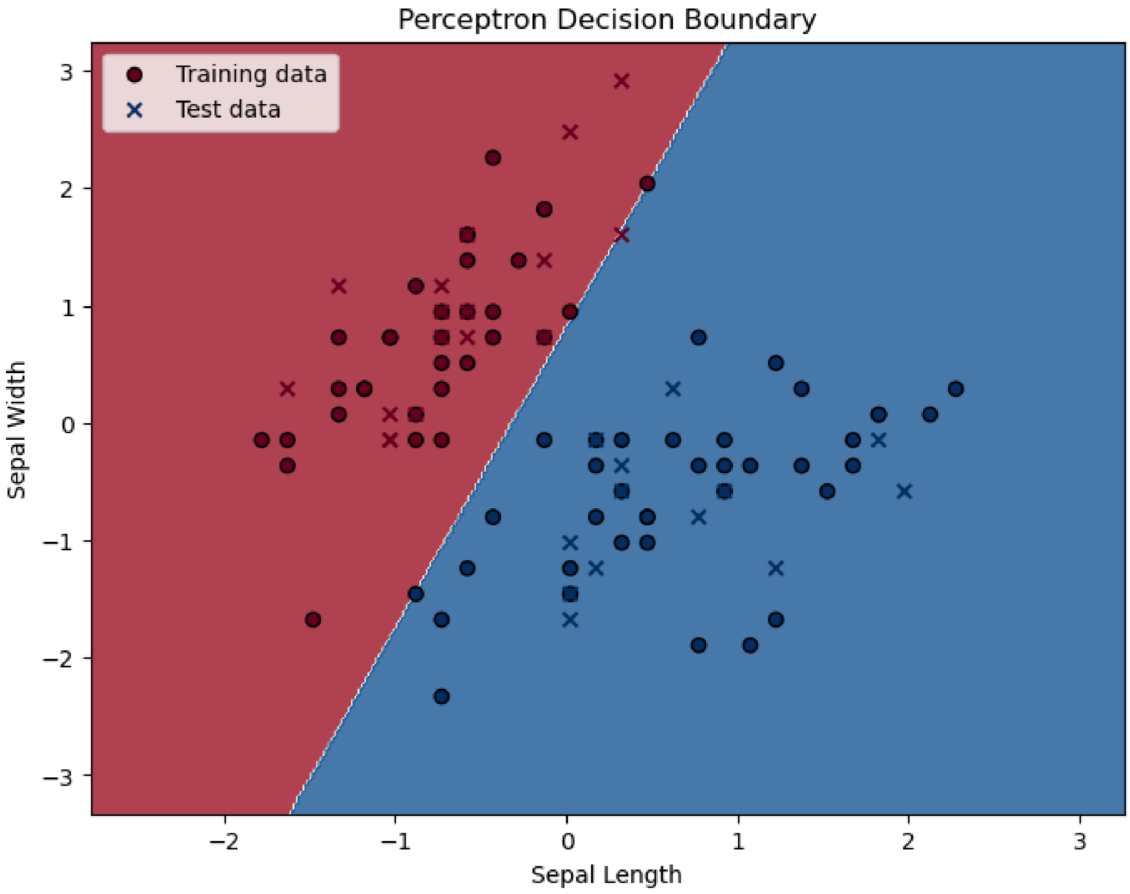
```
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Make predictions across the grid
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary and the data points
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.RdBu)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', cmap=pl
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, edgecolors='k', cmap=plt.

plt.title("Perceptron Decision Boundary")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\cse\AppData\Local\Temp\ipykernel_5108\2155796798.py:14: UserWarni
ng: You passed a edgecolor/edgecolors ('k') for an unfilled marker ('x').
Matplotlib is ignoring the edgecolor in favor of the facecolor.  This beha
vior may change in the future.
  plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, edgecolors='k', cmap=p
lt.cm.RdBu, marker='x', label='Test data')
```



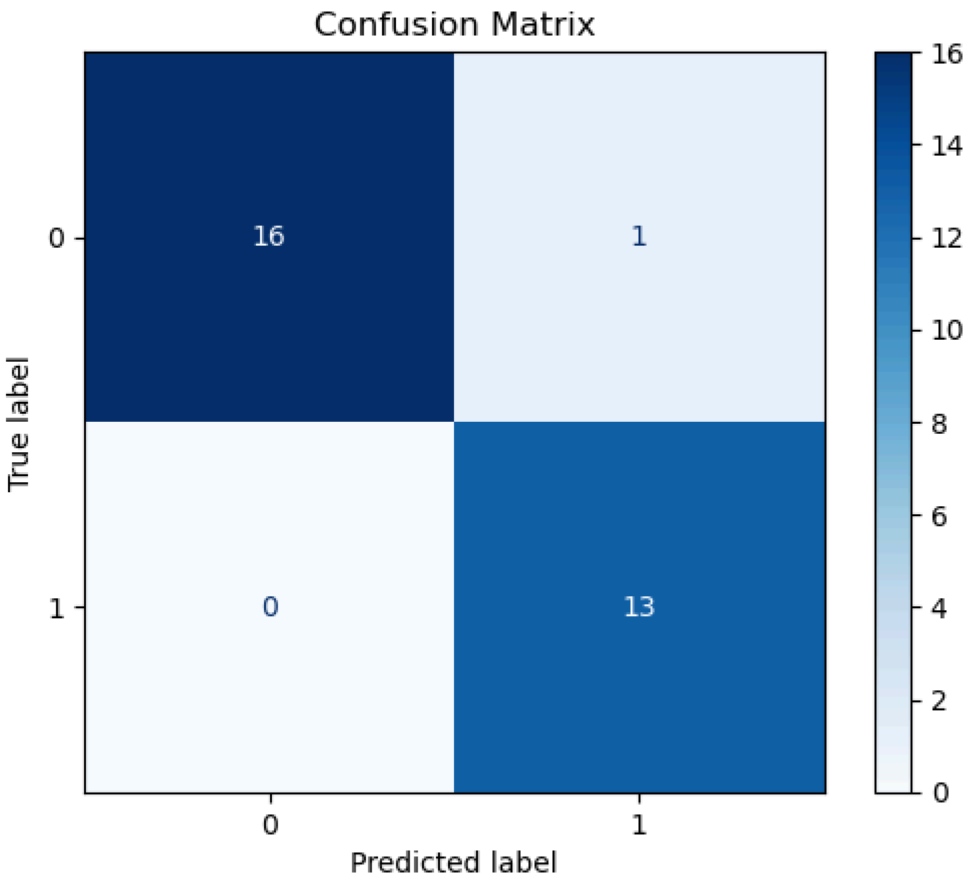
In [20]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Get predictions
y_pred = model.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Use the unique values from y_train as display labels
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:

In [ ]: