PERCEPTRON XOR

AIM:-The primary aim of this code is to implement and demonstrate a simple Perceptron model learning the XOR logical function.

PROBLEM DESCRIPTION:- The Perceptron, a simple neural network unit, is employed to learn the XOR function. The problem involves training the Perceptron to correctly classify input patterns and visualize its learning process.

ALGORITHM:-

1) Imports and Setup:
   - The code begins by importing necessary libraries such as matplotlib.pyplot, ListedColormap from matplotlib.colors, MLPClassifier from sklearn.neural_network, accuracy_score from sklearn.metrics, and numpy as np.
   - The plot_decision_boundary function is defined to visualize the decision boundary of the classifier.
2) plot_decision_boundary Function:
   a) This function takes the input data X, corresponding labels y, the trained model, and a title as parameters.
   b) It sets up a meshgrid of points covering the input space.
   c) The trained model is used to predict the output for each point in the meshgrid.
   d) The decision boundary is then plotted along with the input data points using ‘contourf’ for the decision boundary and scatter for data points.
3) Data Preparation:
   a) The XOR input data (inputs) is created as a NumPy array, with four combinations of binary values.
   b) The corresponding output data (outputs) is created, representing the XOR logic.
4) MLPClassifier Initialization:
   a) An instance of MLPClassifier is created. It's a multi-layer perceptron model with specified parameters:
      i) hidden_layer_sizes=(3,): A single hidden layer with 3 neurons.

ii) activation='relu': Rectified Linear Unit (ReLU) activation function is used.
iii) max_iter=10000: Maximum number of iterations for training the model.
iv) random_state=42: Sets the random seed for reproducibility.

5) Training the Model:
   a) The MLPClassifier (mlp) is trained using the fit method with the input-output pairs (inputs, outputs).
6) Testing the Model:
   a) Test data (test_data) with the same XOR combinations is created.
   b) Predictions are made using the trained model on this test data.
7) Accuracy Calculation:
   a) The accuracy of the model is calculated by comparing the predicted outputs with the actual outputs using the accuracy_score function from sklearn.metrics.
8) Printing Predictions and Accuracy:
   a) The predictions and accuracy are printed.
9) Visualizing Decision Boundary:
   a) The plot_decision_boundary function is called to visualize the decision boundary of the trained model using the input data points.

PSEUDOCODE:-

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import numpy as np

def plot_decision_boundary(X, y, model, title):
    h = 0.01
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, cmap=ListedColormap(['#FFAAAA', '#AAAAFF']),
alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['#FF0000', '#0000FF']),
edgecolors='k', marker='o')
    plt.title(title)
    plt.xlabel('Input 1')
    plt.ylabel('Input 2')
    plt.show()


inputs = np.array([[0, 0],
                   [0, 1],
                   [1, 0],
                   [1, 1]])

outputs = np.array([0, 1, 1, 0])

mlp = MLPClassifier(hidden_layer_sizes=(3,), activation='relu', max_iter=10000,
random_state=42)

mlp.fit(inputs, outputs)


test_data = np.array([[0, 0],
                      [0, 1],
                      [1, 0],
                      [1, 1]])

predictions = mlp.predict(test_data)
```

```
accuracy=accuracy_score(outputs, predictions)
print("Predictions after training:")
print(predictions)
print("Accuracy:",accuracy)
plot_decision_boundary(inputs, outputs, mlp, 'XOR Gate Decision Boundary')
```

RESULT:- The trained perceptron model correctly predicts the XOR gate OUTPUT.

```
Predictions after training:
[0 1 1 0]
Accuracy: 1.0
```