EXPERIMENT-5

AIM: Implementation of perceptron networks.

THEORY

1. Introduction to Perceptron

The Perceptron is the simplest type of artificial neural network and is used for binary classification tasks. It is based on a linear model, meaning it can only classify linearly separable data.
The perceptron was introduced by Frank Rosenblatt in 1958 as a computational model inspired by biological neurons.

2. Structure of a Perceptron

A perceptron consists of the following components:

(a) Input Layer

- Takes multiple **input features** X=(x1,x2,...,xn)X = (x_1, x_2, ..., x_n)X=(x1,x2,...,xn).
- Each input has an associated **weight** w=(w1,w2,...,wn)w = (w_1, w_2, ..., w_n)w=(w1,w2,...,wn).

(b) Weights & Bias

- Weights determine the importance of each input.
- Bias **shifts** the decision boundary to improve learning.

(c) Summation Function

- Computes the weighted sum: z=w1x1+w2x2+...+wnxn+bz = w_1 x_1 + w_2 x_2 + ... + w_n x_n + bz=w1x1+w2 x2+...+wnxn+b

(d) Activation Function

- Determines the **output** based on the weighted sum.
- The **step activation function** is commonly used in a basic perceptron:
- $$u(t-a)=\begin{cases}0, & t<a\\ 1, & t>a\end{cases}$$
- In modern neural networks, **sigmoid, ReLU, or softmax** are used.

(e) Output Layer

- Produces a **binary classification** output (0 or 1).

3. Working of a Perceptron

A perceptron follows these steps:

Step 1: Initialize Weights and Bias

- Start with **random** weights and bias.

Step 2: Compute Weighted Sum

- Calculate z=w1x1+w2x2+...+wnxn+bz = w_1 x_1 + w_2 x_2 + ... + w_n x_n + bz=w1x1+w2x2+...+wnxn+b.

Step 3: Apply Activation Function

- Use **step activation** to get output: y=f(z)y = f(z)y=f(z)

Step 4: Compute Error

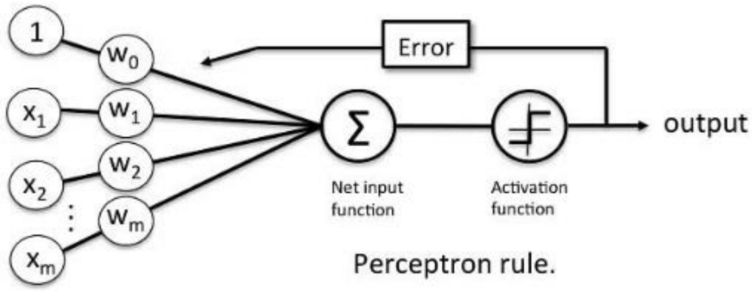
- Compare the predicted output with the **actual label**.

Step 5: Update Weights (Perceptron Learning Rule).

$w^{new} = w^{old} + \Delta w \qquad \Delta w = \eta(d - o)$

η is called the **learning rate**

$0 < \eta \leq 1$



CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Step 1: Generate a Linearly Separable Dataset
X, y = make_classification(
    n_samples=4000,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_clusters_per_class=1,
    random_state=42,
    class_sep=2.0
)

# Convert labels to -1 and 1 for Perceptron
y = np.where(y == 0, -1, 1)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 2: Implement the Perceptron
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iters=50):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
        self.train_accuracy = []

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for epoch in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = np.where(linear_output >= 0, 1, -1)

                if y_predicted != y[idx]:
                    update = self.lr * (y[idx] - y_predicted)
                    self.weights += update * x_i
                    self.bias += update

            y_train_pred = self.predict(X)
            acc = accuracy_score(y, y_train_pred)
```

```
        self.train_accuracy.append(acc)
        print(f"Epoch {epoch + 1}/{self.n_iters}, Training Accuracy: {acc * 100:.2f}%")

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return np.where(linear_output >= 0, 1, -1)

# Step 3: Train and Evaluate the Perceptron
perceptron = Perceptron(learning_rate=0.01, n_iters=50)
perceptron.fit(X_train, y_train)

y_pred = perceptron.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Step 4: Plot Training Progress
plt.figure(figsize=(10, 5))
plt.plot(range(1, perceptron.n_iters + 1), perceptron.train_accuracy, marker='o', linestyle='-')
plt.xlabel("Epochs")
plt.ylabel("Training Accuracy")
plt.title("Training Accuracy Over Epochs")
plt.grid()
plt.show()

# Step 5: Plot Decision Boundary with Sample Image
def plot_decision_boundary(X, y, model, image_path=None):
    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, 0.01), np.arange(y_min, y_max, 0.01))

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

    fig, ax = plt.subplots(figsize=(8, 6))

    # Plot Decision Boundary
    ax.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
    ax.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Paired)
    ax.set_title("Perceptron Decision Boundary")

    plt.show()

# Provide the image path (Change this to an actual image file)
sample_image_path = "sample.png"  # Replace with a valid image path

plot_decision_boundary(X_test, y_test, perceptron, image_path=sample_image_path)
```

OUTPUT:

```
Epoch 1/50, Training Accuracy: 97.56%
Epoch 2/50, Training Accuracy: 97.25%
Epoch 3/50, Training Accuracy: 98.06%
Epoch 4/50, Training Accuracy: 97.59%
Epoch 5/50, Training Accuracy: 98.03%
Epoch 6/50, Training Accuracy: 97.69%
Epoch 7/50, Training Accuracy: 97.81%
Epoch 8/50, Training Accuracy: 98.28%
Epoch 9/50, Training Accuracy: 98.06%
Epoch 10/50, Training Accuracy: 98.03%
Epoch 11/50, Training Accuracy: 97.72%
Epoch 12/50, Training Accuracy: 98.03%
Epoch 13/50, Training Accuracy: 98.06%
Epoch 14/50, Training Accuracy: 98.12%
Epoch 15/50, Training Accuracy: 97.72%
Epoch 16/50, Training Accuracy: 97.69%
Epoch 17/50, Training Accuracy: 97.97%
Epoch 18/50, Training Accuracy: 98.31%
```

Epoch 19/50, Training Accuracy: 97.88%
Epoch 20/50, Training Accuracy: 97.97%
Epoch 21/50, Training Accuracy: 97.81%
Epoch 22/50, Training Accuracy: 98.06%
Epoch 23/50, Training Accuracy: 98.03%
Epoch 24/50, Training Accuracy: 98.16%
Epoch 25/50, Training Accuracy: 98.03%
Epoch 26/50, Training Accuracy: 98.28%
Epoch 27/50, Training Accuracy: 98.25%
Epoch 28/50, Training Accuracy: 98.28%
Epoch 29/50, Training Accuracy: 98.31%
Epoch 30/50, Training Accuracy: 98.06%
Epoch 31/50, Training Accuracy: 98.25%
Epoch 32/50, Training Accuracy: 98.28%
Epoch 33/50, Training Accuracy: 98.31%
Epoch 34/50, Training Accuracy: 97.81%
Epoch 35/50, Training Accuracy: 98.25%
Epoch 36/50, Training Accuracy: 98.31%
Epoch 37/50, Training Accuracy: 98.31%
Epoch 38/50, Training Accuracy: 97.88%
Epoch 39/50, Training Accuracy: 97.97%
Epoch 40/50, Training Accuracy: 97.97%
Epoch 41/50, Training Accuracy: 98.28%
Epoch 42/50, Training Accuracy: 98.09%
Epoch 43/50, Training Accuracy: 98.03%
Epoch 44/50, Training Accuracy: 97.94%
Epoch 45/50, Training Accuracy: 98.31%
Epoch 46/50, Training Accuracy: 98.06%
Epoch 47/50, Training Accuracy: 98.12%
Epoch 48/50, Training Accuracy: 98.00%
Epoch 49/50, Training Accuracy: 98.28%
Epoch 50/50, Training Accuracy: 97.69%
Test Accuracy: 97.50%