Name: Tanishq Thuse *PRN-* 12310237 *Rollno.* 52 *Class- TY CSAI- B*

Experiment Number: 03

Title: Write a program to demonstrate the perceptron learning law with decision regions, visually showcasing how a perceptron learns and separates different classes.

Objective

To implement the **Perceptron Learning Algorithm** and visually demonstrate how the perceptron learns to classify linearly separable data. The goal is to showcase the decision boundary and decision regions as the perceptron iteratively updates its weights to separate two classes.

Theory

The **Perceptron** is the simplest form of a neural network, introduced by Frank Rosenblatt in 1958. It is a binary linear classifier that maps input features to one of two classes using a linear decision boundary.

1. Perceptron Output:

   $y = f(w \cdot x + b)$

   where:

   $w \rightarrow$ weight vector

   $x \rightarrow$ input vector

   $b \rightarrow$ bias

   $f(\cdot) \rightarrow$ step activation function

2. Weight Update Rule:

   $w_{new} = w_{old} + \eta \cdot (t - y) \cdot x$

   $b_{new} = b_{old} + \eta \cdot (t - y)$

   where:

   t = target output

   y = predicted output

   η = learning rate

The perceptron converges only when the data is **linearly separable**.

Algorithm

1. Initialize weights and bias with small random values.
2. For each training sample:

    - Compute output using step function.
    - Compare output with target label.
    - Update weights and bias using the perceptron learning law.
3. Repeat for several epochs or until convergence.
4. Plot decision boundary and regions to visualize learning.

Program (Python Implementation)

Assignment 03

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from matplotlib.colors import ListedColormap
```

In [21]:
```
# Step function
def step_function(x):
    return np.where(x >= 0, 1, 0)
```

In [23]:
```
# Perceptron class
class Perceptron:
    def __init__(self, learning_rate=0.1, n_epochs=10):
        self.lr = learning_rate
        self.n_epochs = n_epochs

    def fit(self, X, y):
        self.weights = np.zeros(X.shape[1])
        self.bias = 0
        self.errors = []

        for _ in range(self.n_epochs):
            total_error = 0
            for xi, target in zip(X, y):
                linear_output = np.dot(xi, self.weights) + self.bias
                y_pred = step_function(linear_output)
                update = self.lr * (target - y_pred)
                self.weights += update * xi
                self.bias += update
                total_error += int(update != 0.0)
            self.errors.append(total_error)
        return self

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return step_function(linear_output)
```

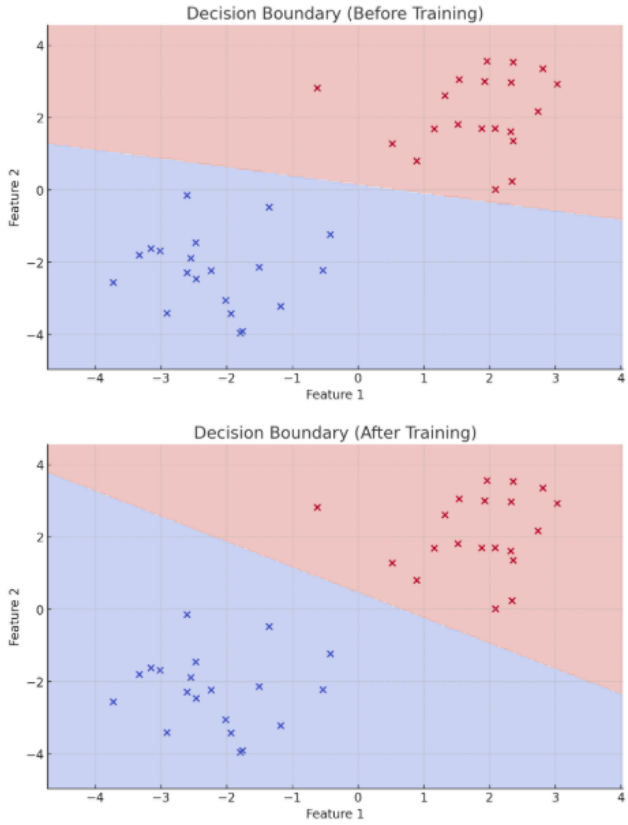
In [ ]:
```
# Generate dataset (2D linearly separable)
X, y = make_classification(n_samples=100, n_features=2,
                           n_redundant=0, n_informative=2,
                           n_clusters_per_class=1, class_sep=1.5, random_state=42)
```

In [25]:
```
# Train perceptron
ppn = Perceptron(learning_rate=0.1, n_epochs=10)
ppn.fit(X, y)
```

Out[25]:
```
<__main__.Perceptron at 0x276b9577c20>
```

In [29]:
```
def plot_decision_regions(X, y, classifier):
    cmap = ListedColormap(('red', 'blue'))
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, 0.02),
                           np.arange(x2_min, x2_max, 0.02))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
```

Sample Output



.

Conclusion

This assignment successfully demonstrated the Perceptron Learning Law. The perceptron was able to learn a linear decision boundary to separate two classes. The plotted decision regions clearly showed how the perceptron distinguishes between classes after training. Although perceptrons work well for linearly separable data, they fail to classify non-linear problems, which led to the development of advanced models like the Multi-Layer Perceptron (MLP) and Deep Neural Networks.