Practical 4 –

Code –

```
import numpy as np

import matplotlib.pyplot as plt


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 = None


    def fit(self, X, y):

        n_samples, n_features = X.shape


        # Initialize weights and bias

        self.weights = np.zeros(n_features)

        self.bias = 0


        # Convert labels to {0, 1} format if necessary

        y_ = np.where(y <= 0, -1, 1)


        for _ in range(self.n_iterations):

            for idx, x_i in enumerate(X):

                linear_output = np.dot(x_i, self.weights) + self.bias

                y_pred = np.sign(linear_output)


                # Update rule

                if y_[idx] * y_pred <= 0:

                    update = self.learning_rate * y_[idx]
```

```python
                self.weights += update * x_i
                self.bias += update

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

# Manually create linearly separable data
X = np.array([
    [2, 3], [1, 1], [2, 1], [3, 3], [2, 2],  # Class 0
    [6, 5], [7, 7], [8, 6], [7, 5], [9, 7]   # Class 1
])
y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])

# Train perceptron
perceptron = Perceptron(learning_rate=0.1, n_iterations=10)
perceptron.fit(X, y)

# Plotting decision regions
def plot_decision_regions(X, y, model):
    # Set axis boundaries
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # Create meshgrid
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Predict on grid points
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    Z = model.predict(grid_points)
```

```python
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolor='k')
    plt.title("Perceptron Decision Regions")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

plot_decision_regions(X, y, perceptron)
```

Output –