Logistic Regression

AIM:

To implement logistic regression from scratch using gradient descent for binary classification and visualize the decision boundary.

ALGORITHM:

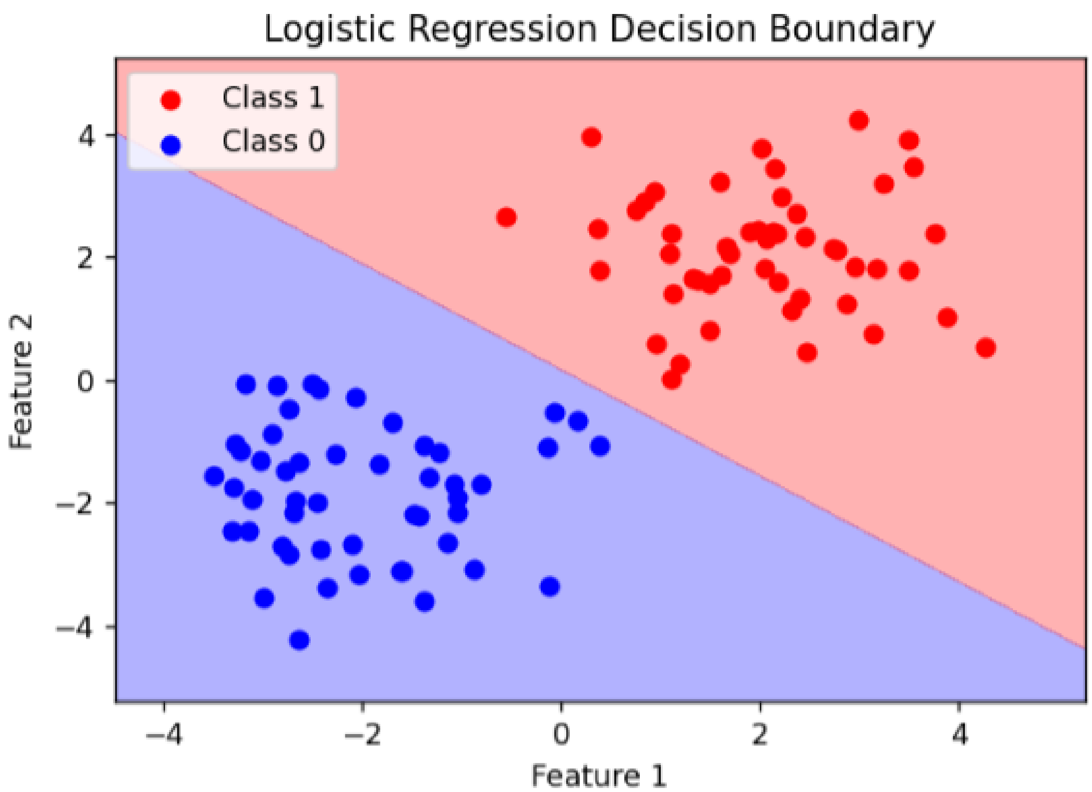
Step 1: Generate synthetic 2D data for two classes.
Step 2: Add a bias term to the feature matrix.
Step 3: Define the sigmoid activation function.
Step 4: Define the binary cross-entropy loss function.
Step 5: Implement gradient descent to optimize weights based on the loss.
Step 6: Train the logistic regression model on the data.
Step 7: Predict class labels using the learned weights.
Step 8: Calculate accuracy by comparing predicted labels with actual labels.
Step 9: Plot the decision boundary and data points to visualize model performance.

SOURCE CODE:

```
import numpy as np
import matplotlib.pyplot as plt

# 1. Simulate Data (2D binary classification)
np.random.seed(0)
X1 = np.random.randn(50, 2) + np.array([2, 2])
X2 = np.random.randn(50, 2) + np.array([-2, -2])
X = np.vstack((X1, X2))
y = np.hstack((np.ones(50), np.zeros(50)))

# 2. Add bias term (intercept)
X_b = np.c_[np.ones((X.shape[0], 1)), X]  # shape: (100, 3)

# 3. Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# 4. Loss Function (Binary Cross Entropy)
```

```
def loss(y, y_pred):
    return -np.mean(y * np.log(y_pred + 1e-10) + (1 - y) * np.log(1 - y_pred + 1e-10))

# 5. Gradient Descent
def train(X, y, lr=0.1, epochs=1000):
    weights = np.zeros(X.shape[1])
    for epoch in range(epochs):
        z = X @ weights
        y_pred = sigmoid(z)
        gradient = X.T @ (y_pred - y) / y.size
        weights -= lr * gradient
        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Loss = {loss(y, y_pred):.4f}")
    return weights

# 6. Train the model
weights = train(X_b, y)

# 7. Predict
def predict(X, weights):
    return sigmoid(X @ weights) >= 0.5

y_pred = predict(X_b, weights)
accuracy = np.mean(y_pred == y)
print(f"\nFinal Accuracy: {accuracy * 100:.2f}%")

# 8. Plot Decision Boundary
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.linspace(x1_min, x1_max, 100),
                       np.linspace(x2_min, x2_max, 100))
grid = np.c_[np.ones(xx1.ravel().shape), xx1.ravel(), xx2.ravel()]
probs = sigmoid(grid @ weights).reshape(xx1.shape)

plt.figure(figsize=(6,4))
plt.contourf(xx1, xx2, probs, levels=[0, 0.5, 1], alpha=0.3, colors=['blue', 'red'])
plt.scatter(X1[:, 0], X1[:, 1], color='red', label='Class 1')
plt.scatter(X2[:, 0], X2[:, 1], color='blue', label='Class 0')
plt.title("Logistic Regression Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Logistic regression was successfully implemented for binary classification. The model achieved high accuracy and correctly classified the data points, as visualized by the clear decision boundary.