Name Vaibhav Patel

Scholer No 211112262

Logistic Regression

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Data Input
X = np.array([1, 2, 3, 4, 5,6,7,8])  # Example feature input
y = np.array([0, 0,0,0, 1,1, 1, 1])  # Binary labels

# Step 2: Define the Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Step 3: Define the Cost Function
def cost_function(X, y, m, c):
    m = len(X)
    h = sigmoid(m*X + c)
    error = (-y * np.log(h)) - ((1 - y) * np.log(1 - h))
    cost = (1 / m) * sum(error)
    return cost

# Step 4: Define Gradient Descent
def gradient_descent(X, y, learning_rate, iterations):
    m = 0
    c = 0
    n = len(X)
    for _ in range(iterations):
        h = sigmoid(m*X + c)
        gradient_m = (1/n)*np.dot(X.T, (h - y))
        gradient_c = (1/n)*np.sum(h - y)
        m -= learning_rate * gradient_m
        c -= learning_rate * gradient_c
    return m, c

# Step 5: Training the Model
learning_rate = 0.01
iterations = 100000
optimal_m, optimal_c = gradient_descent(X, y, learning_rate, iterations)

# Step 6: Predict the Values of m and c
print("Optimal 'm' value:", optimal_m)
print("Optimal 'c' value:", optimal_c)

# Step 7: Predict the values of 'y' based on the optimal 'm' and 'c'
plt.scatter(X, y, c=y, cmap=plt.cm.Spectral)
X1=np.linspace(min(X),max(X),100)
```

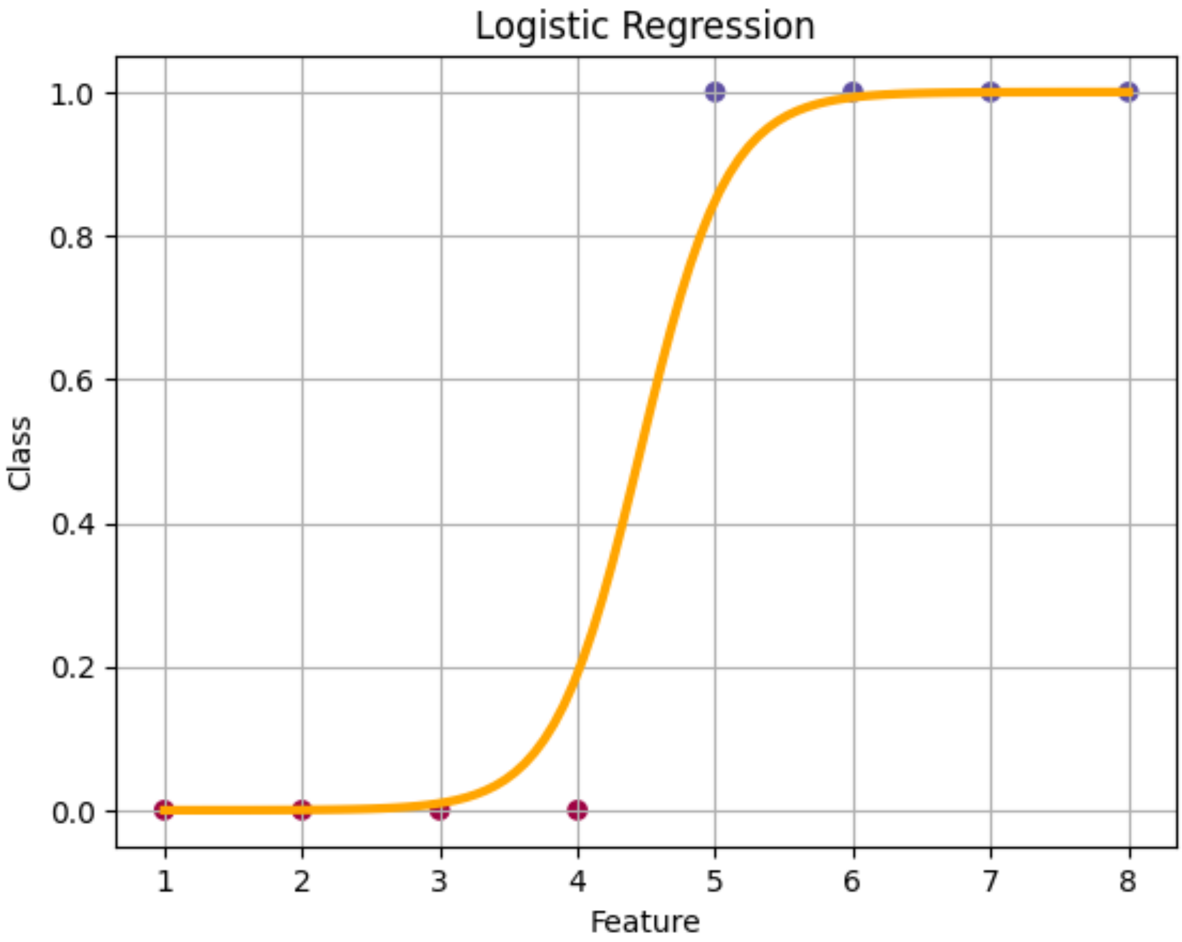
```
predicted_y = sigmoid(optimal_m*X1 + optimal_c)
print("Predicted 'y' values:", predicted_y)

pred_y= sigmoid(optimal_m*X + optimal_c)

# Step 8: Plot the Graph

# plt.scatter(X, y, c=y, cmap=plt.cm.Spectral)
# X1=np.linspace(min(X),max(X),8)
plt.plot(X1, predicted_y, color='orange', linewidth=3)
plt.xlabel('Feature')
plt.ylabel('Class')
plt.title('Logistic Regression')
plt.grid(True)
plt.show()
```

```
Optimal 'm' value: 3.1768392821276543
Optimal 'c' value: -14.155678097956308
Predicted 'y' values: [1.70586035e-05 2.13547754e-05 2.67328998e-05 3.3465
4404e-05
 4.18934649e-05 5.24439141e-05 6.56512136e-05 8.21843219e-05
 1.02880575e-04 1.28788039e-04 1.61218487e-04 2.01813682e-04
 2.52628260e-04 3.16233371e-04 3.95846225e-04 4.95491958e-04
 6.20205788e-04 7.76285334e-04 9.71605308e-04 1.21600964e-03
 1.52179947e-03 1.90433964e-03 2.38281081e-03 2.98114026e-03
 3.72915004e-03 4.66396796e-03 5.83175295e-03 7.28979164e-03
 9.10902509e-03 1.13770615e-02 1.42017178e-02 1.77151032e-02
 2.20782040e-02 2.74858360e-02 3.41716813e-02 4.24129121e-02
 5.25335843e-02 6.49055851e-02 7.99454213e-02 9.81046413e-02
 1.19851325e-01 1.45640125e-01 1.75869184e-01 2.10824312e-01
 2.50614276e-01 2.95105791e-01 3.43871452e-01 3.96166522e-01
 4.50948568e-01 5.06946096e-01 5.62769883e-01 6.17047766e-01
 6.68555972e-01 7.16321223e-01 7.59677307e-01 7.98273133e-01
 8.32040910e-01 8.61139439e-01 8.85888107e-01 9.06703776e-01
 9.24047910e-01 9.38386804e-01 9.50164672e-01 9.59787583e-01
 9.67615697e-01 9.73961262e-01 9.79090309e-01 9.83226449e-01
 9.86555655e-01 9.89231319e-01 9.91379131e-01 9.93101550e-01
 9.94481751e-01 9.95587037e-01 9.96471723e-01 9.97179555e-01
 9.97745705e-01 9.98198417e-01 9.98560345e-01 9.98849648e-01
 9.99080868e-01 9.99265647e-01 9.99413301e-01 9.99531280e-01
 9.99625544e-01 9.99700856e-01 9.99761025e-01 9.99809094e-01
 9.99847495e-01 9.99878173e-01 9.99902680e-01 9.99922258e-01
 9.99937897e-01 9.99950391e-01 9.99960371e-01 9.99968344e-01
 9.99974712e-01 9.99979800e-01 9.99983864e-01 9.99987110e-01]
```



In [ ]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Assuming y_true and y_pred are the true labels and predicted labels
y_true = np.array([0, 0,0,0, 1,1, 1, 1])  # Actual labels
y_pred = pred_y.round().astype(int)  # Predicted labels

# Calculate performance metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
gmean = np.sqrt(recall*(1-precision))
tpr = recall
fpr = 1 - recall







print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Gmean:", gmean)
print("TPR:", tpr)
print("FAR:", fpr)
```

```
Accuracy: 0.875
Precision: 0.8
Recall: 1.0
F1 Score: 0.888888888888889
Gmean: 0.44721359549995787
TPR: 1.0
FAR: 0.0
```

In [ ]:

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

e=2.718281828459045
def sigmoid(self,z):
        sig = 1/(1+e**(-z))
        return sig

# Define the data
x = np.array([1,2,3,4,5,6,7,8])
y = np.array([0,0,0,0,1,1,1,1])
# Define the learning rate and number of iterations
learning_rate = 0.02
iterations = 100
# Initialize the weights
m = 0
b = 0
# Implement gradient descent algorithm
for _ in range(iterations):
 # Calculate the predictions
 y_pred = (m * x + b )
 # Calculate the errors
 errors = y - y_pred
 # Calculate the gradients
 m_gradient = -2 * np.mean(x * errors)
 b_gradient = -2 * np.mean(errors)
 # Update the weights
 m -= learning_rate * m_gradient
 b -= learning_rate * b_gradient
# Calculate the RMSE
rmse = np.sqrt(np.mean(errors**2))
# Calculate the MAE
mae = np.mean(np.abs(errors))
# Calculate the coefficient of determination (R^2)
r_squared = 1 - np.var(errors) / np.var(y)
# Print the results

# Implement ridge regularization
lambda_value = 0.1
for _ in range(iterations):
 # Calculate the predictions
 y_pred = m * x + b
 # Calculate the errors
 errors = y - y_pred
 # Calculate the gradients with regularization
 m_gradient = -2 * np.mean(x * errors) - 2 * lambda_value * m
 b_gradient = -2 * np.mean(errors)
 # Update the weights
 m -= learning_rate * m_gradient
```

```
    b -= learning_rate * b_gradient
# Calculate the RMSE with regularization
print("m",m)
print("b",b)


plt.scatter(x, y, c=y, cmap=plt.cm.Spectral)
plt.plot(x, m * x + b, color='orange', linewidth=3)
plt.xlabel('Input Feature')
plt.ylabel('Output Class')
plt.title('Logistic Regression')
plt.grid(True)
plt.show()
```

```
m 0.17902879444701045
b -0.28895736670362904
```