======================================================================  
 Exp no 2… Logistic Regression Classifier

======================================================================

AIM:

To implement logistic regression as a simple neural network model

Program:

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_breast\_cancer

from sklearn.metrics import accuracy\_score

X, y = load\_breast\_cancer(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train = (X\_train - np.mean(X\_train, axis=0)) / (np.std(X\_train, axis=0) + 1e-8)

X\_test = (X\_test - np.mean(X\_test, axis=0)) / (np.std(X\_test, axis=0) + 1e-8)

class LogisticRegressionModel:

def \_\_init\_\_(self, learning\_rate=0.01, num\_iterations=1000):

self.learning\_rate = learning\_rate

self.num\_iterations = num\_iterations

self.weights = None

self.bias = None

def \_sigmoid(self, z):

return 1 / (1 + np.exp(-z))

def fit(self, X, y):

n\_samples, n\_features = X.shape

self.weights = np.zeros(n\_features)

self.bias = 0

for iteration in range(self.num\_iterations):

linear\_model = np.dot(X, self.weights) + self.bias

y\_predicted = self.\_sigmoid(linear\_model)

cost = -np.mean(y \* np.log(y\_predicted + 1e-8) + (1 - y) \* np.log(1 - y\_predicted + 1e-8))

dw = (1 / n\_samples) \* np.dot(X.T, (y\_predicted - y))

db = (1 / n\_samples) \* np.sum(y\_predicted - y)

self.weights -= self.learning\_rate \* dw

self.bias -= self.learning\_rate \* db

if (iteration + 1) % 100 == 0:

print(f"Iteration {iteration + 1}/{self.num\_iterations}, Cost: {cost:.4f}")

def predict(self, X):

linear\_model = np.dot(X, self.weights) + self.bias

y\_predicted = self.\_sigmoid(linear\_model)

y\_predicted\_cls = [1 if i > 0.5 else 0 for i in y\_predicted]

return np.array(y\_predicted\_cls)

lr\_model = LogisticRegressionModel(learning\_rate=0.001, num\_iterations=5000)

lr\_model.fit(X\_train, y\_train)

y\_pred = lr\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("\nModel Accuracy on Test Set:", accuracy)

OUTPUT…

Iteration 100/5000, Cost: 0.5478

Iteration 200/5000, Cost: 0.4618

Iteration 300/5000, Cost: 0.4058

Iteration 400/5000, Cost: 0.3662

Iteration 500/5000, Cost: 0.3366

Iteration 600/5000, Cost: 0.3135

Iteration 700/5000, Cost: 0.2948

Iteration 800/5000, Cost: 0.2793

Iteration 900/5000, Cost: 0.2662

Iteration 1000/5000, Cost: 0.2550

Iteration 1100/5000, Cost: 0.2452

Iteration 1200/5000, Cost: 0.2366

Iteration 1300/5000, Cost: 0.2290

Iteration 1400/5000, Cost: 0.2221

Iteration 1500/5000, Cost: 0.2159

Iteration 1600/5000, Cost: 0.2103

Iteration 1700/5000, Cost: 0.2052

Iteration 1800/5000, Cost: 0.2004

Iteration 1900/5000, Cost: 0.1961

Iteration 2000/5000, Cost: 0.1921

Iteration 2100/5000, Cost: 0.1883

Iteration 2200/5000, Cost: 0.1848

Iteration 2300/5000, Cost: 0.1816

Iteration 2400/5000, Cost: 0.1785

Iteration 2500/5000, Cost: 0.1757

Iteration 2600/5000, Cost: 0.1730

Iteration 2700/5000, Cost: 0.1704

Iteration 2800/5000, Cost: 0.1680

Iteration 2900/5000, Cost: 0.1657

Iteration 3000/5000, Cost: 0.1635

Iteration 3100/5000, Cost: 0.1615

Iteration 3200/5000, Cost: 0.1595

Iteration 3300/5000, Cost: 0.1576

Iteration 3400/5000, Cost: 0.1559

Iteration 3500/5000, Cost: 0.1542

Iteration 3600/5000, Cost: 0.1525

Iteration 3700/5000, Cost: 0.1509

Iteration 3800/5000, Cost: 0.1494

Iteration 3900/5000, Cost: 0.1480

Iteration 4000/5000, Cost: 0.1466

Iteration 4100/5000, Cost: 0.1453

Iteration 4200/5000, Cost: 0.1440

Iteration 4300/5000, Cost: 0.1427

Iteration 4400/5000, Cost: 0.1415

Iteration 4500/5000, Cost: 0.1404

Iteration 4600/5000, Cost: 0.1393

Iteration 4700/5000, Cost: 0.1382

Iteration 4800/5000, Cost: 0.1371

Iteration 4900/5000, Cost: 0.1361

Iteration 5000/5000, Cost: 0.1351

Model Accuracy on Test Set: 0.9824561403508771