****

**Green University of Bangladesh**

**Department of Computer Science and Engineering (CSE)**

**Faculty of Sciences and Engineering**

**Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)**

**Home Work**

**Course Title: Machine Learning Lab**

**Course Code: CSE 412 Section: 213\_D2**

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | **Md. Rabby Khan** | **213902037** |

**Lab Date : 01-10-2024**

**Submission Date : 08-10-2024**

**Course Teacher’s Name : Sadia Afroze**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature:.....................**  **Comments:.............................................. Date:..............................** |

**Home Work**

**Problem:**

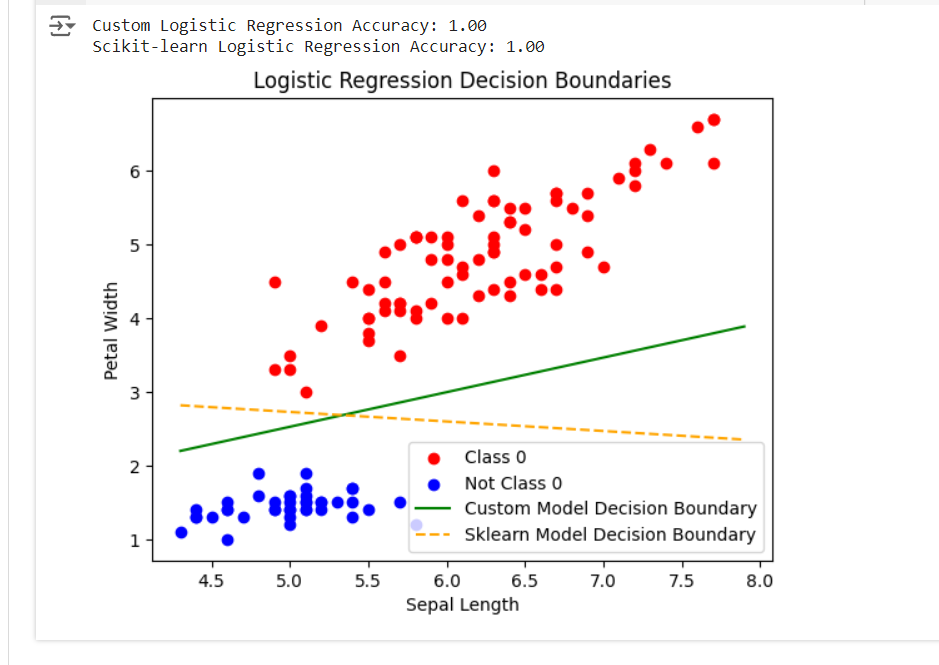
1. Implement logistic regression from scratch (i.e., without using any machine learning library) and compare the performance with the one of scikit-learn.

2. Evaluate the model performance using Sepal Length vs Petal Width only.

**Solution:**

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.datasets import load\_iris  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score  from sklearn.linear\_model import LogisticRegression  # Step 1: Data Preparation  # Load the Iris dataset  data = pd.read\_csv('/content/sample\_data/Data\_Set/iris.csv')  iris = load\_iris()  X = iris.data[:, [0, 2]]  # Sepal Length vs Petal Width  y = iris.target  # Convert to binary classification (class 0 vs. class 1 and 2)  y\_binary = (y == 0).astype(int)  # Class 0 vs. Not Class 0  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_binary, test\_size=0.2, random\_state=42)  # Step 2: Logistic Regression Implementation from Scratch  class LogisticRegressionScratch:      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):          num\_samples, num\_features = X.shape          self.weights = np.zeros(num\_features)          self.bias = 0          # Gradient Descent          for \_ in range(self.num\_iterations):              linear\_model = np.dot(X, self.weights) + self.bias              y\_predicted = self.sigmoid(linear\_model)              # Compute gradients              dw = (1 / num\_samples) \* np.dot(X.T, (y\_predicted - y))              db = (1 / num\_samples) \* np.sum(y\_predicted - y)              # Update weights and bias              self.weights -= self.learning\_rate \* dw              self.bias -= self.learning\_rate \* db      def predict(self, X):          linear\_model = np.dot(X, self.weights) + self.bias          y\_predicted = self.sigmoid(linear\_model)          y\_predicted\_class = [1 if i > 0.5 else 0 for i in y\_predicted]          return np.array(y\_predicted\_class)  # Step 3: Model Training  # Train the custom logistic regression model  custom\_model = LogisticRegressionScratch(learning\_rate=0.1, num\_iterations=1000)  custom\_model.fit(X\_train, y\_train)  # Train and predict using scikit-learn's logistic regression  sklearn\_model = LogisticRegression()  sklearn\_model.fit(X\_train, y\_train)  y\_pred\_sklearn = sklearn\_model.predict(X\_test)  # Step 4: Model Evaluation  # Predict and evaluate custom model  y\_pred\_custom = custom\_model.predict(X\_test)  accuracy\_custom = accuracy\_score(y\_test, y\_pred\_custom)  accuracy\_sklearn = accuracy\_score(y\_test, y\_pred\_sklearn)  # Print the accuracies  print(f"Custom Logistic Regression Accuracy: {accuracy\_custom:.2f}")  print(f"Scikit-learn Logistic Regression Accuracy: {accuracy\_sklearn:.2f}")  # Step 5: Optional Visualization  plt.scatter(X\_train[y\_train == 0][:, 0], X\_train[y\_train == 0][:, 1], color='red', label='Class 0')  plt.scatter(X\_train[y\_train == 1][:, 0], X\_train[y\_train == 1][:, 1], color='blue', label='Not Class 0')  # Decision boundary for the custom model  x\_values = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)  y\_values = -(custom\_model.weights[0] \* x\_values + custom\_model.bias) / custom\_model.weights[1]  plt.plot(x\_values, y\_values, color='green', label='Custom Model Decision Boundary')  # Decision boundary for the sklearn model  y\_values\_sklearn = -(sklearn\_model.coef\_[0][0] \* x\_values + sklearn\_model.intercept\_[0]) / sklearn\_model.coef\_[0][1]  plt.plot(x\_values, y\_values\_sklearn, color='orange', linestyle='dashed', label='Sklearn Model Decision Boundary')  plt.xlabel('Sepal Length')  plt.ylabel('Petal Width')  plt.title('Logistic Regression Decision Boundaries')  plt.legend()  plt.show() |

**Output:**

****