Logistic Regression

[Logistic Regression](https://www.geeksforgeeks.org/understanding-logistic-regression/) models the likelihood that an instance will belong to a particular class. It uses a linear equation to combine the input information and the sigmoid function to restrict predictions between 0 and 1. Gradient descent and other techniques are used to optimize the model’s coefficients to minimize the [log loss](https://www.geeksforgeeks.org/ml-log-loss-and-mean-squared-error/). These coefficients produce the resulting decision boundary, which divides instances into two classes. When it comes to binary classification, logistic regression is the best choice because it is easy to understand, straightforward, and useful in a variety of settings. Generalization can be improved by using regularization.

**Implementation of Logistic Regression using Python**

**Import Libraries**

# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_diabetes

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_curve, auc

**Read and Explore the data**

# Load the diabetes dataset

diabetes = load\_diabetes()

X, y = diabetes.data, diabetes.target

# Convert the target variable to binary (1 for diabetes, 0 for no diabetes)

y\_binary = (y > np.median(y)).astype(int)

**Splitting The Dataset: Train and Test dataset**

# Split the data 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)

**Feature Scaling**

# Standardize features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**Train The Model**

# Train the Logistic Regression model

model = LogisticRegression()

model.fit(X\_train, y\_train)

**Evaluation Metrics**

# Evaluate the model

y\_pred = model.predict(X\_test)

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

print("Accuracy: {:.2f}%".format(accuracy \* 100))

**Output:**

Accuracy: 73.03%

**Confusion Matrix and Classification Report**

# evaluate the model

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

**Output:**

Confusion Matrix:

[[36 13]

[11 29]]

Classification Report:

precision recall f1-score support

0 0.77 0.73 0.75 49

1 0.69 0.72 0.71 40

accuracy 0.73 89

macro avg 0.73 0.73 0.73 89

weighted avg 0.73 0.73 0.73 89