**LAB ASSIGNMENT 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name:** | Harsh Shah | **Semester:** | VII | **Division:** | 6 |
| **Roll No.:** | 21BCP359 | **Date:** | 12-08-24 | **Batch:** | G11 |
| **Aim:** | Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs | | | | |

**Objective**

To fit data points by assigning different weights to each point based on its proximity to the query point.

**Dataset:** Synthetic sinusoidal patterned data (Generated using *numpy* library)

**Code**

import numpy as np

import matplotlib.pyplot as plt

***# Generate Dataset and splitting into features and target vars***

We generate 80 random data points X uniformly distributed between 0 and 5. The target variable y is generated by taking the sine of X and adding some random noise to simulate real-world data.

np.random.seed(0)

X = np.sort(5 \* np.random.rand(80, 1), axis=0)

y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])

***# Normal Linear Regression***

*def* normal\_equation\_linear\_regression(*X*, *y*):

    m = X.shape[0]

    X\_augmented = np.hstack([np.ones((m, 1)), X])

    XTX = np.dot(X\_augmented.T, X\_augmented)

    XTX\_inv = np.linalg.pinv(XTX)

    theta = np.dot(XTX\_inv, np.dot(X\_augmented.T, y))

    return theta

theta = normal\_equation\_linear\_regression(X, y)

*def* predict(*X*, *theta*):

    m = X.shape[0]

    X\_augmented = np.hstack([np.ones((m, 1)), X])

    y\_pred = np.dot(X\_augmented, theta)

    return y\_pred

y\_pred = predict(X, theta)

plt.figure(*figsize*=(12, 8))

plt.scatter(X, y, *label*="Training Data", *color*="blue")

plt.plot(X, y\_pred, *color*="red", *linewidth*=2, *label*="Linear Regression Fit")

plt.xlabel("X")

plt.ylabel("y")

plt.title("Normal Linear Regression")

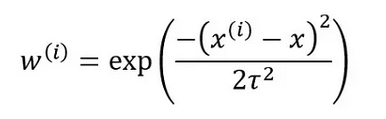
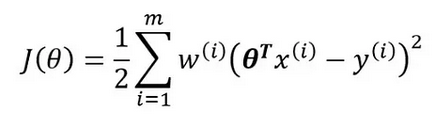
plt.legend()

plt.show()

print(*f*"Intercept: {theta[0]}")

print(*f*"Coefficient: {theta[1]}")

***# Implementation of Locally Weighted Regression algorithm***

** **

*def*  lwlr(*X*, *y*, *x\_query*, *tau*):

    m = X.shape[0]

    W = np.exp(-np.sum((X - x\_query) \*\* 2, *axis*=1) / (2 \* tau\*\*2))

    W = np.diag(W)

    X\_augmented = np.hstack([np.ones((m, 1)), X])

    x\_query\_augmented = np.array([1, x\_query.item()]).reshape(1, 2)

    XTWX = np.dot(np.dot(X\_augmented.T, W), X\_augmented)

    XTWy = np.dot(np.dot(X\_augmented.T, W), y)

    theta = np.linalg.solve(XTWX, XTWy)

    y\_query = np.dot(x\_query\_augmented, theta)

    return y\_query

*def* predict\_lwlr(*X*, *y*, *X\_query*, *tau*):

    y\_pred = np.array([lwlr(X, y, x\_query, tau) for x\_query in X\_query])

    return y\_pred

***# Using Multiple Query Points***

X\_query = np.linspace(0, 5, 100).reshape(-1, 1)

taus = [0.1, 0.3, 0.8, 2.0]

predictions = {}

for tau in taus:

    predictions[tau] = predict\_lwlr(X, y, X\_query, tau)

***# Plotting the Graph***

plt.figure(*figsize*=(12, 8))

plt.scatter(X, y, *label*="Training Data", *color*="blue")

for tau in taus:

    plt.plot(X\_query, predictions[tau], *linewidth*=2, *label*=*f*"Tau = {tau}")

plt.xlabel("X")

plt.ylabel("y")

plt.title("Locally Weighted Linear Regression with Different values of Tau")

plt.legend()

plt.show()

**Output**

