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]}")
```

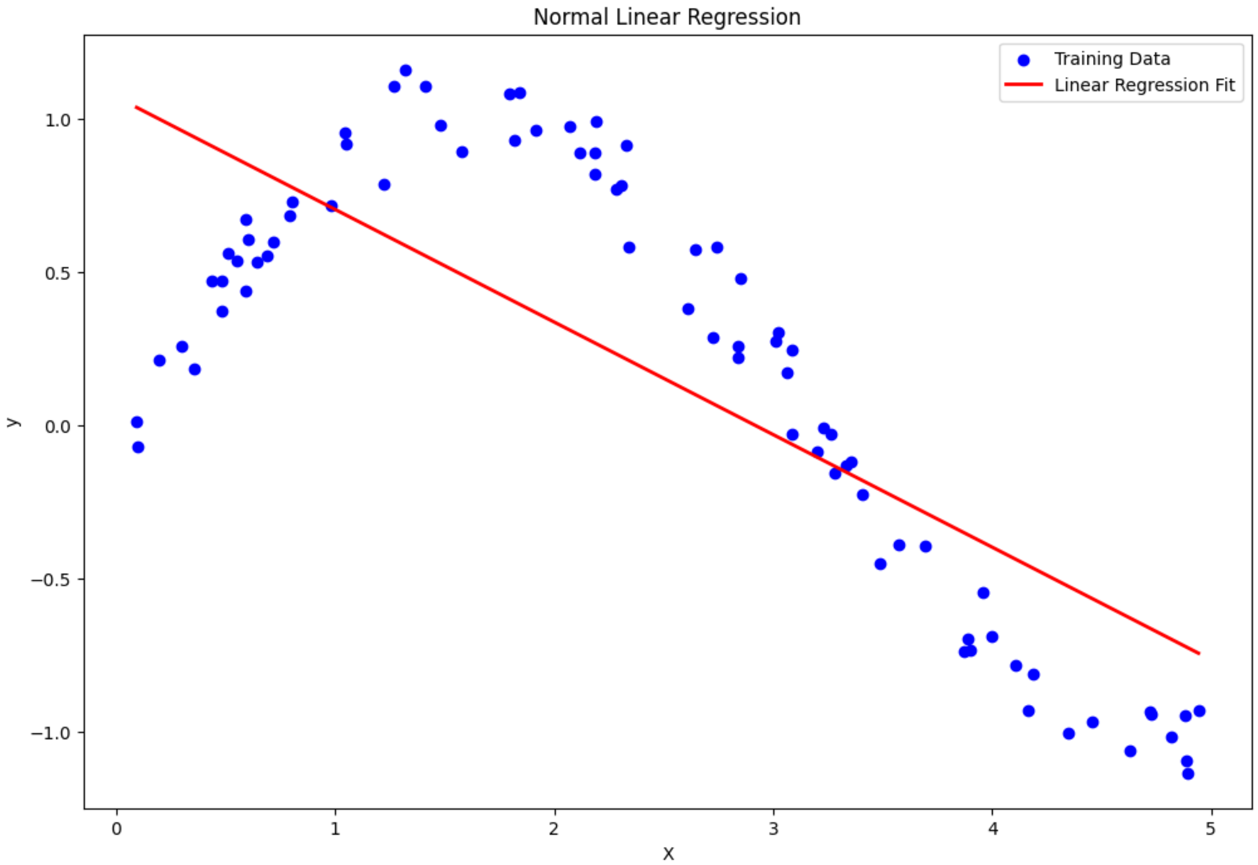
 # Implementation of Locally Weighted Regression algorithm

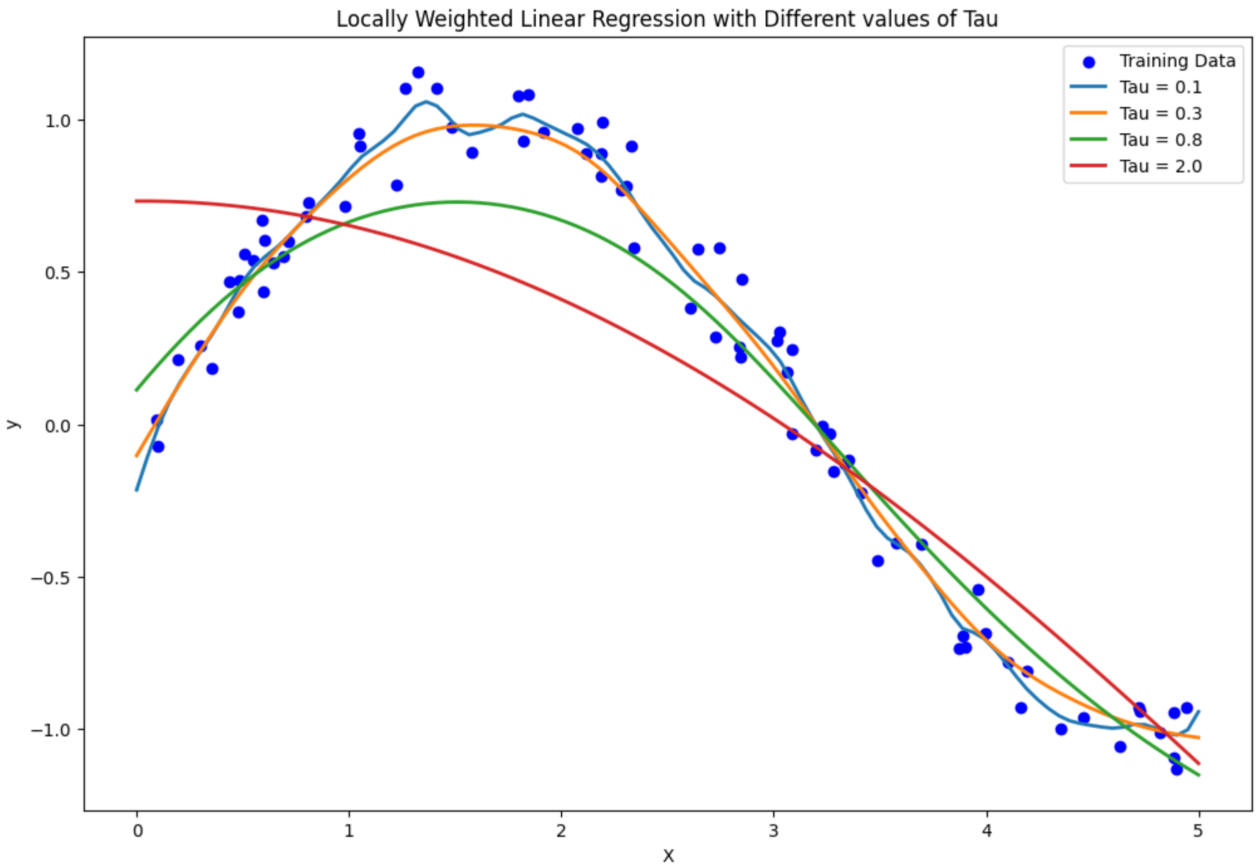
$$w^{(i)} = \exp\left(\frac{-\left(x^{(i)} - x\right)^2}{2\tau^2}\right) \qquad J(\theta) = \frac{1}{2}\sum_{i=1}^{m} w^{(i)}\left(\boldsymbol{\theta}^{\boldsymbol{T}} x^{(i)} - y^{(i)}\right)^2$$

```
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


Normal Linear Regression
Training Data
Linear Regression Fit
1.0
0.5
0.0
−0.5
−1.0
y
0
1
2
3
4
5
x


Locally Weighted Linear Regression with Different values of Tau
Training Data
Tau = 0.1
Tau = 0.3
Tau = 0.8
Tau = 2.0
1.0
0.5
0.0
−0.5
−1.0
y
0
1
2
3
4
5
x