Experiment 10 - Non-Parametric Locally Weighted Regression

May 1, 2023

1 Experiment Details

1.1 Submitted By

Desh Iyer, 500081889, Year III, AI/ML(H), B5

1.2 Problem Statement

Implement the non-parametric Locally Weighted Regression algorithm to fit a curve to a set of data points. Select an appropriate dataset for your experiment and draw graphs to visualize the results.

1.3 Theory

Locally Weighted Regression (LWR) is a non-parametric algorithm used to fit a curve to a set of data points. Unlike parametric algorithms, LWR does not assume any specific functional form for the underlying data. Instead, it fits a curve to the data by using a weighted linear regression. The weights are determined by a kernel function that assigns higher weights to points that are closer to the target point and lower weights to points that are farther away.

The basic idea behind LWR is to use a linear regression to fit a curve to the data points that are closest to the target point. The weights are determined by a kernel function that assigns higher weights to points that are closer to the target point and lower weights to points that are farther away. This allows the algorithm to capture the local structure of the data and to fit a curve that is more flexible than a simple linear regression.

1.3.1 Parametric vs Non-Parametric Learning Algorithms

Parametric — In a Parametric Algorithm, we have a fixed set of parameters such as theta that we try to find(the optimal value) while training the data. After we have found the optimal values for these parameters, we can put the data aside or erase it from the computer and just use the model with parameters to make predictions. Remember, the model is just a function.

Non-Parametric — In a Non-Parametric Algorithm, you always have to keep the data and the parameters in your computer memory to make predictions. And that's why this type of algorithm may not be great if you have a really really massive dataset.

1.3.2 Need for NPLW Regression

We specifically apply this regression technique when the data to fit is non-linear. In Linear Regression we would fit a straight line to this data but that won't work here because the data is non-linear and our predictions would end up having large errors. We need to fit a curved line so that our error is minimized.

1.3.3 Under the Hood

In Locally weighted linear regression, we give the model the `x` where we want to make the prediction, then the model gives all the `x(i)`'s around that `x` a higher weight close to one, and the rest of `x(i)`'s get a lower weight close to zero and then tries to fit a straight line to that weighted `x(i)`'s data.

This means that if want to make a prediction for the green point on the x-axis (see figure below), the model gives higher weight to the input data i.e. `x(i)`'s near or around the circle above the green point and all else `x(i)` get a weight close to zero, which results in the model fitting a straight line only to the data which is near or close to the circle. The same goes for the purple, yellow, and grey points on the x-axis.

1.3.4 Calculating Error

In the loss function, it translates to error terms for the `x(i)`'s which are far from `x` being multiplied by almost zero and for the `x(i)`'s which are close to `x` get multiplied by almost 1. In short, it only sums over the error terms for the `x(i)`'s which are close to `x`.

1.4 Steps

Here are the steps to implement the Locally Weighted Regression algorithm:

1. Load the dataset.
2. Define the kernel function.
3. For each point in the dataset:
    1. Calculate the weights for the neighbouring points using the kernel function.
    2. Fit a linear regression to the neighbouring points using the weights.
    3. Predict the target value using the linear regression.
4. Plot the predicted values along with the original data points to visualize the results.

1.5 Advantages

- LWR is a non-parametric algorithm, which means it can fit a curve to the data without assuming any specific functional form for the underlying data.
- LWR can capture the local structure of the data and fit a curve that is more flexible than a simple linear regression.

1.6 Limitations

- LWR can be computationally expensive, especially for large datasets.
- LWR can be sensitive to the choice of kernel function and its parameters.

1.7 Pseudocode

Here's the pseudocode for the Locally Weighted Regression algorithm:

1. Load the dataset.
2. Define the kernel function.

3. For each point in the dataset:
    1. Calculate the weights for the neighbouring points using the kernel function.
    2. Fit a linear regression to the neighbouring points using the weights.
    3. Predict the target value using the linear regression.
4. Plot the predicted values along with the original data points to visualize the results.

2 Import Libraries

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

3 Define the Parameters of the Model

[ ]:
```
# Define global variables
numberSamples = 500

# Lambda function to de-linearize input function
deLinearize = lambda X: np.cos(1.5 * np.pi * X) + np.cos(5 * np.pi * X)

# Define X and y
X = np.sort(np.random.rand(numberSamples)) * 2
y = deLinearize(X) + np.random.randn(numberSamples) * 0.1

X = X.reshape(X.shape[0], 1)
y = y.reshape(y.shape[0], 1)

# Define tau
tauList = np.arange(0, 0.1, step=0.01)
```

4 Define a Function to Calculate the Weight Matrix

[ ]:
```
# Function to calculate weight matrix
def calculateWeightMatrix(point, X, tau):
    '''
    The parameters of this function are,
    tau --> bandwidth
    X --> Training data.
    point --> the x where we want to make the prediction.
    '''

    # m is the number of training examples.
    m = X.shape[0]
```

```
    # Initializing W as an identity matrix.
    w = np.mat(np.eye(m))

    # Calculating weights for all training examples [x(i)'s].
    for i in range(m):
        xi = X[i]
        d = (-2 * tau * tau)
        w[i, i] = np.exp(np.dot((xi - point), (xi - point).T) / d)

    return w
```

5 Define a Function to Predict for a Point in the Input Vector

```
[ ]: # Function to predict for a single point in the input vector
def predictSinglePoint(X, y, point, tau):
    # Calculating the weight matrix using the wm function we wrote earlier.
    w = calculateWeightMatrix(point, X, tau)

    # Calculating parameter theta using the formula.
    theta = np.linalg.pinv(X.T * (w * X)) * (X.T * (w * y))

    # Calculating predictions.
    pointPrediction = np.dot(point, theta)

    # Returning the theta and predictions
    return theta, pointPrediction
```

6 Define a Function to Predict all Points for a Single Value of Tau

```
[ ]: # Function to predict for a single tau value for all the points in the input␣
 ↪vector
def predictSingleTau(XTest, tau):
    # Empty list for storing predictions.
    predictionForSingleTau = []

    # Predicting for all numberPredictions values and storing them in␣
 ↪predictions.
    for point in XTest:
        _, pointPrediction = predictSinglePoint(X, y, point, tau)
        predictionForSingleTau.append(pointPrediction)

    # Reshaping predictions
    predictionForSingleTau = np.array(predictionForSingleTau).
 ↪reshape(numberSamples, 1)
```

```
    return predictionForSingleTau
```

7 Define Test Data

[ ]:
```
# Define testing data to predict
XTest = np.sort(np.random.rand(numberSamples)) * 2
XTest = np.array(XTest).reshape(numberSamples, 1)
```

8 Plot Predictions for Multiple Tau Values

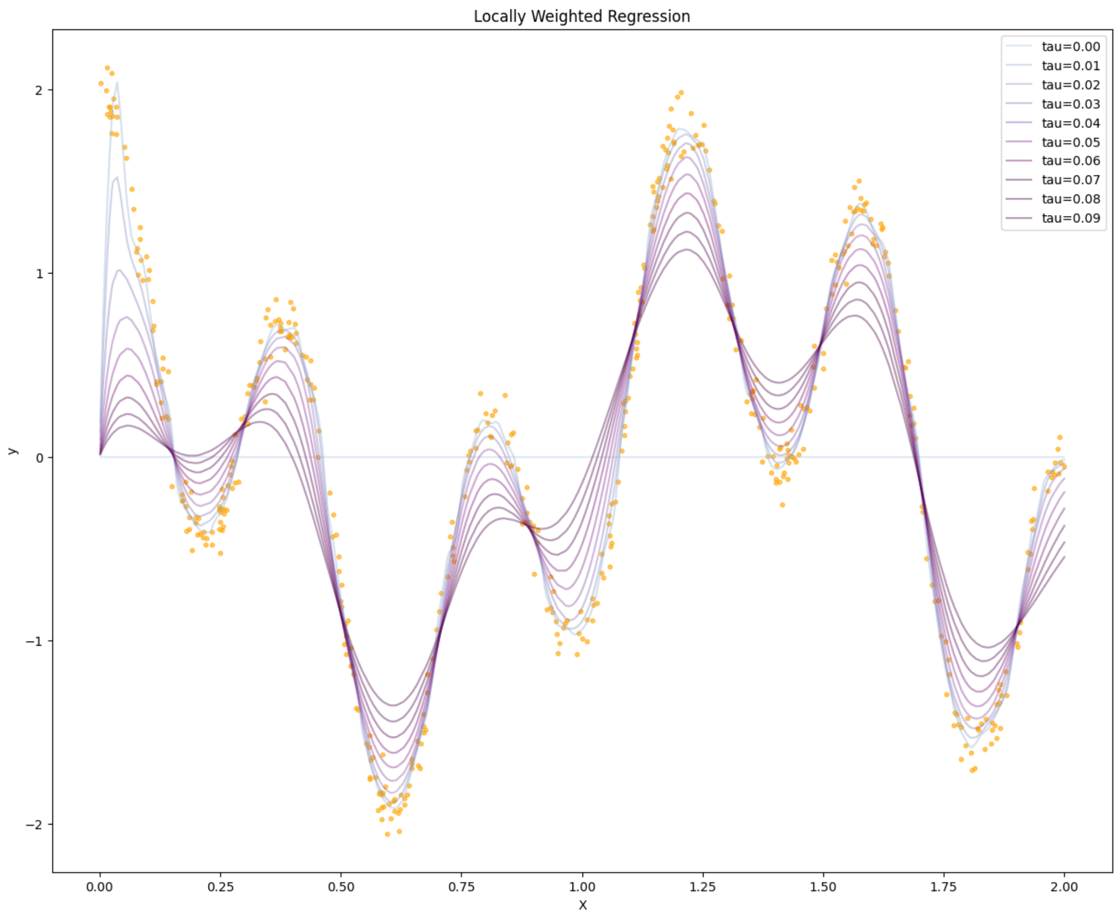
[ ]:
```
# Plot the training data
plt.figure(figsize=(15, 12))
# plt.figure(figsize=(20, 15), dpi=80)

plt.scatter(X, y, s=10, c='orange', marker='o', alpha=0.6)
# plt.plot(X, y, c='orange', marker='o', alpha=0.6)

# Lower value means that the higher tau values are darker in the gradient
colorDelta = 0.3

# Predict for each tau value and plot
for i, tau in enumerate(tauList):
    prediction = predictSingleTau(XTest, tau)
    color = plt.cm.BuPu(colorDelta + (i / len(tauList)))
    plt.plot(XTest, prediction, color=color, alpha=0.4, label=f'tau={tau:.2f}')

# Set plot attributes
plt.title("Locally Weighted Regression")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```



9 Conclusion

Implementing the Locally Weighted Regression algorithm is a powerful technique for fitting a curve to a set of data points. With the right dataset and parameters, LWR can capture the local structure of the data and fit a curve that is more flexible than a simple linear regression. Visualizing the results using plots can help to gain insights into the underlying data and the quality of the fit.