10: Locally Weighted Regression algorithm in order to fit data points

Algorithm of "Locally Weighted Regression"

1. Read the Given data sample to X and the curve to Y

2. Set the value for smoothening parameter say τ (tau)

1. Set the bias point of interest set X_0 which is a subset of X

1. Determine the weight matrix using :

$$w(x, x_0) = e^{\frac{(x - x_0)^2}{-2\tau^2}}$$

1. Determine the value of model term parameter $\hat{\beta}$ using :

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W Y$$

2. Prediction = $X_0 * \hat{\beta}$

a) Read the given data sample to X and the curve to Y

```
import numpy as np
from math import pi


number_of_datapoints = 1000


# our data samples are  f(x) = sin(x),  x ∈ [0, 2π ]

X = np.linspace(0, 2 * pi, number_of_datapoints)                         # generating a evenly space thousand datapoints in range 0 to 2π


# output  = sin(x) + noise, because output will be a narrow curve without noise

Y = np.sin(X)   +   0.1 *  np.random.randn( number_of_datapoints )          # generating the output
```

b) Set the value for smoothening parameter say τ

```
tau = 10
```

c) Set the bias point of interest set X_0 which is a subset of X

```
X0 = X[200]  # any point you like
```

d) Determine the weight matrix using :

$$w(x, x_0) = e^{\frac{(x - x_0)^2}{-2\tau^2}}$$

```python
def get_weights(X0, tau):


    squared_difference = (X - X0) ** 2

    denominator =  -2 * (tau ** 2)

    W = np.exp( squared_difference / denominator)

    return W
```

e) Determine the value of model term parameter $\hat{\beta}$ using :

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W Y$$

```python
def calc_beta(W, inputs, outputs):

    X = inputs.copy()    # since we are modifying X else it is dangerous
    Y = outputs

   #-------------------------------------------

    # variable set up, for matrix multiplication
    X = np.c_[np.ones(len(X)), X]                      # essential

   # +----------------------------------------

    XTW = X.T * W                                      # X_Transpose * W

    XTWX_inverse = np.linalg.pinv( XTW  @ X)

    beta =  XTWX_inverse @ XTW @ Y                    # as per the equation

    return beta
```

f) Prediction = $X_0 * \hat{\beta}$

```python
def predict(beta, X0):


    # variable set up, for matrix multiplication
    X0 = np.r_[1, X0]

    return beta @ X0
```

Putting things together

```python
def local_weighted_regression(X0, tau):



    W = get_weights(X0,tau)



    global X,Y         # accessing X and Y which are generated in step 1

    beta = calc_beta(W, X, Y)

    prediction = predict(beta, X0)

    return prediction
```

Create a domain, and a helper function for plotting

```python
domain =  np.linspace(0, 2*pi, num=300)  # same as X but only few points


def plotter(tau):

    # get all predictions
    predictions = [ local_weighted_regression(X0, tau) for X0 in domain]

    plot = figure(width=400, height=400,title = f'tau={tau}')    #f-string title (python 3.5+)

    plot.scatter(X, Y, alpha=.3)                                 #plot datapoints

    plot.line(domain, predictions, line_width=2, color='red')    #plot the regression line

    return plot
```

Plot for different values of tau

```python
# essential imports  and setup

from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
output_notebook()


first_row_plots = [plotter(10), plotter(1)]

second_row_plots = [plotter(0.1), plotter(0.01)]

grid = gridplot([ first_row_plots, second_row_plots])
show(grid)
```

(https://bokeh.org) BokehJS 1.4.0 successfully loaded.