Homework 04 – Nonparametric Regression

Importing Data

- After importing NumPy, Pandas, and Matplotlib libraries, I've imported eruption the data set with Pandas' read_csv() function.

Train-Test Split

- Then, I have divided data into train and test sets, and further divided them into X_train, y_train, X_test, and y_test sets. I also created an "N" variable for the number of samples.

```
train = data_set[:150]
test = data_set[150:]
```

```
X_train = train['eruptions']
y_train = train['waiting']
X_test = test['eruptions']
y_test = test['waiting']

N = data_set.shape[0]
```

Regressogram

- I have set the bin width parameter to 0.37 and origin parameter to 1.5; also created a minimum value parameter equal to the origin and maximum value parameter with the value 5.2, to create left and right borders. Using the following formula (from section 8.8.1 of textbook), I have calculated $\hat{g}(x)$ values.

$$\hat{g}(x) = \frac{\sum_{t-1}^{N} b(x,x^t)r^t}{\sum_{t-1}^{N} b(x,x^t)}$$

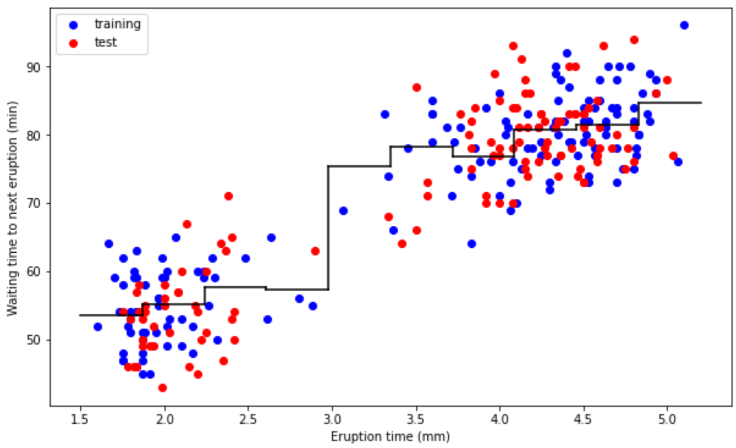
```
Left Borders:
[1.5  1.87 2.24 2.61 2.98 3.35 3.72 4.09 4.46 4.83]
Length of Left Borders: 10

Right Borders:
[1.87 2.24 2.61 2.98 3.35 3.72 4.09 4.46 4.83 5.2 ]
Length of Right Borders: 10

g_hat:
[53.48, 55.09090909090909, 57.6, 57.25, 75.33333333333333, 78.25, 76.76470588235294, 80.69565217391305, 81.36363636363636, 84.
6]
```

Drawing Regressogram

- Using left borders, right borders and $\hat{g}(x)$ values, I drew training and testing data points and regressogram:



Root Mean Squared Error (RMSE) of Regressogram

- I have written a function to calculate Root Mean Squared Error (RMSE) using the formula provided in the guide:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N_{test}} (y_i - \hat{y}_i)^2}{N_{test}}}$$

```
def calculate_rmse(X_test, y_test, left_borders, right_borders):
    total_error = sum([((y_test[(left_borders[i] < X_test) & (X_test <= right_borders[i])] - g_hat[i])**2).sum()
                       for i in range(len(left_borders))])
    return np.sqrt(total_error / len(X_test))
```

```
rmse = calculate_rmse(X_test, y_test, left_borders, right_borders)
```

```
print(f"Regressogram => RMSE is {rmse} when h is {bin_width}")
```

```
Regressogram => RMSE is 5.962617204275405 when h is 0.37
```

Running Mean Smoother

- I have set the bin width parameter to 0.37, and using NumPy's linspace() function, created a data interval array composed of equally separated 1601 points between the minimum (1.5) and maximum values (5.2) I have defined earlier.

```
bin_width = 0.37
data_interval = np.linspace(minimum_value, maximum_value, 1601)
```

- Using the following formula (from section 8.8.1 of textbook), I have calculated $\hat{g}(x)$ values and created a running mean smoother. I also created a w() function for the w function in the formula:

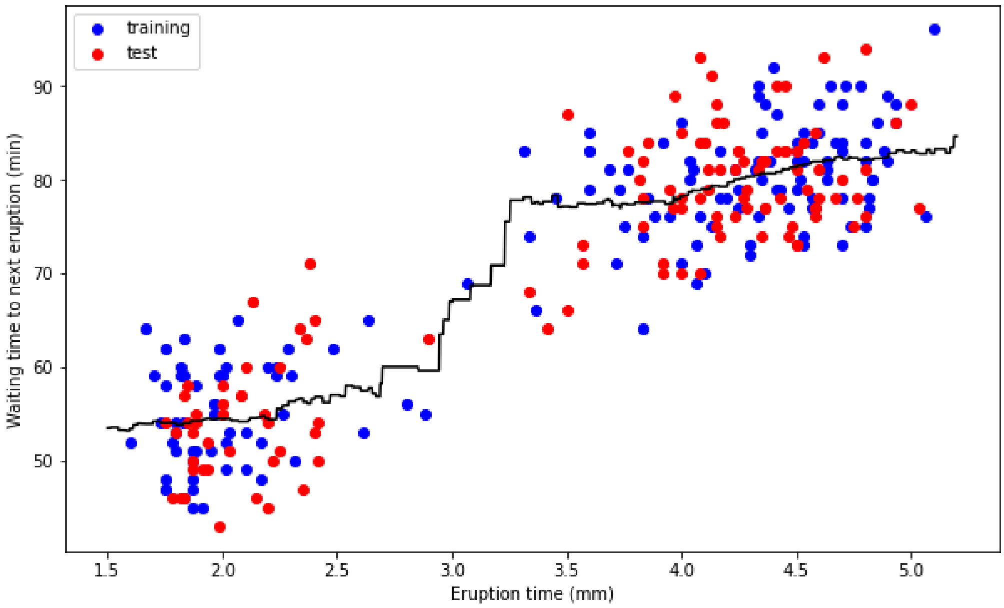
$$\hat{g}(x) = \frac{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right) r^t}{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right)}$$

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

```
def w(u):
    if np.abs(u) < 1:
        return 1
    else:
        return 0
```

Drawing Running Mean Smoother

- Using data interval array and $\hat{g}(x)$ values, I drew training and testing data points and running mean smoother:



Root Mean Squared Error (RMSE) of Running Mean Smoother

- Using the same calculate_rmse function I have created earlier, I have calculated the RMSE for my running mean smoother, with the new left and right borders drawn from the data interval:

```
left_borders = data_interval[:-1]
right_borders = data_interval[1:]
```

```
rmse = calculate_rmse(X_test, y_test, left_borders, right_borders)
```

```
print(f"Running Mean Smoother => RMSE is {rmse} when h is {bin_width}")
```

```
Running Mean Smoother => RMSE is 5.9587977786114354 when h is 0.37
```

Kernel Smoother

- Again, I have set the bin width parameter to 0.37, and created the same data interval I have created earlier.
- This time, using the following formula (from section 8.8.2 of textbook), I have calculated $\hat{g}(x)$ values and created a kernel smoother. I also created a K() function for the K function in the formula:

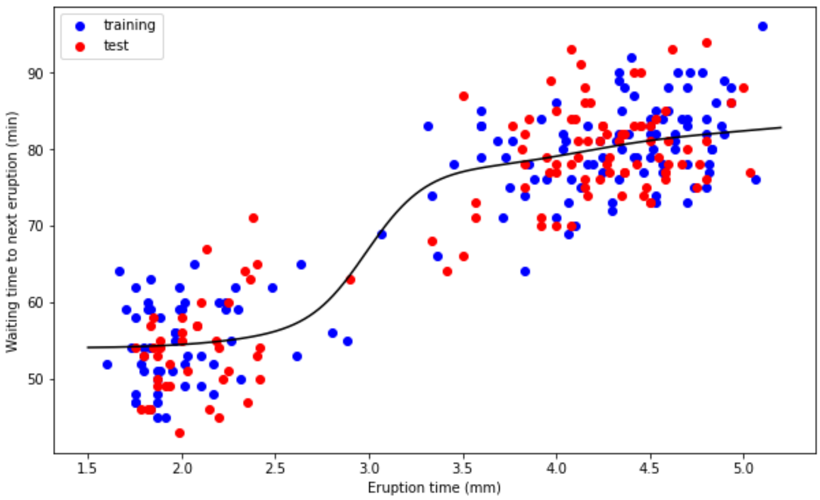
$$\hat{g}(x) = \frac{\sum_t K\left(\frac{x-x^t}{h}\right) r^t}{\sum_t K\left(\frac{x-x^t}{h}\right)}$$

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right]$$

```
def K(u):
    return 1/np.sqrt(2*np.pi) * np.exp(-(u**2)/2)
```

Drawing Kernel Smoother

- Using data interval array and $\hat{g}(x)$ values, I drew training and testing data points and kernel smoother:



Root Mean Squared Error (RMSE) of Kernel Smoother

- Lastly, I have calculated RMSE of kernel smoother using my calculate_rmse function.

```
left_borders = data_interval[:-1]
right_borders = data_interval[1:]
```

```
rmse = calculate_rmse(X_test, y_test, left_borders, right_borders)
```

```
print(f"Kernel Smoother => RMSE is {rmse} when h is {bin_width}")
```

```
Kernel Smoother => RMSE is 5.874042666597442 when h is 0.37
```