12.11.2019
Emre Uludağ
50209 Hw04

Homework 4 was about defining non-parametric regressions. For this purpose we had to implement 3 algorithms.

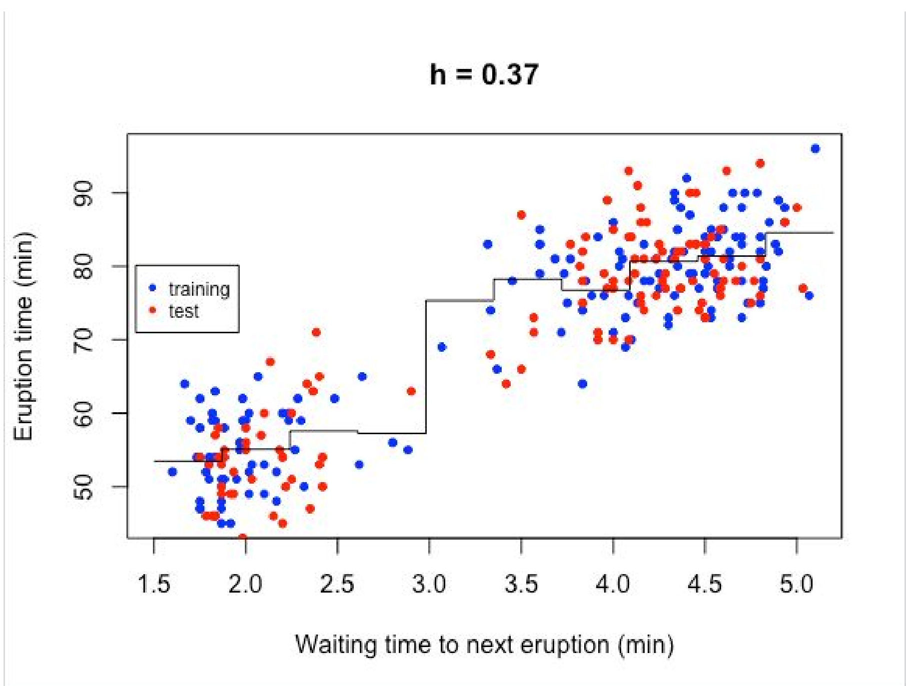
- First I read data from the “hw04_data_set.csv” file and separated the data into training and test data sets.

- Then I assigned the bin_width, origin and x_maximum values as:
bin-width <- 0.37
origin <- 1.5
x_maximum <- max(data$eruptions)

-I implemented regressogram function with above three arguments along with the data argument.
The implementation of the regressogram function applies following procedure:

- I calculated the bin_value of each data point
- I grouped them in accordance with the bin value.
- Then I calculated the mean of grouped y values in every group and if there is no value in that group the mean of that group I assigned to 0.
- Lastly, I returned a data frame sorted according to their bin_value.

-Afterwards I applied the above created formula to training data using bin-width, origin and x_max parameters. When I applied the formula and I got the **training_regressogram** data frame back. When I plotted the **training_regressogram** following diagram showed up:



- I added a dummy point with the same eruption time as the lowest in order to start plotting from the origin
- When I calculated the root mean squared of test data points with the regressogram data I got the following output:

```
> cat('Regressogram => RMSE is', test_regressogram_r
Regressogram => RMSE is 5.962617 when h is 0.37
>
```

- On later stages, I defined values again for mean smoother and following that I created a sequence in order to detect mean smoothed data point more concisely. Here are my parameters:

rms_bin_width <- 0.37
rms_origin <- 1.5
rms_x_max <- max(data$eruptions) #this is actually 5.1

The sequence I created:
rms_data_interval <- seq(from = rms_origin, to = rms_x_max, by = 0.01)

The implementation of the regressogram function applies following procedure:

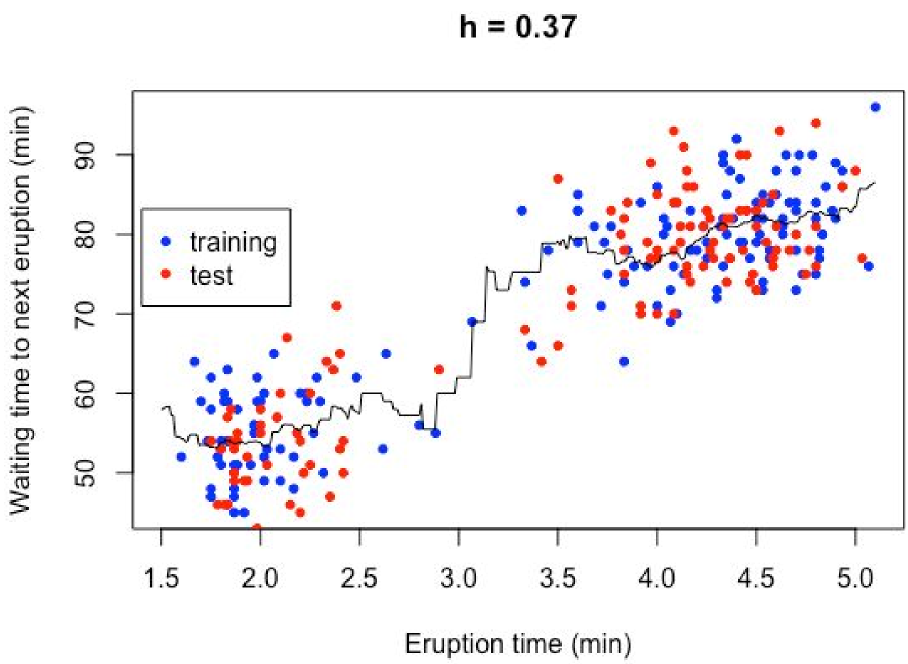
- I again classified data points in w function, it classifies data points a value smaller than 0.5.
- And then implemented rms function
- Both methods applies the logic of the following two formulas:

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

where

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

- When I applied the training_data along with above created data sequence to the rms function I got a data frame back. When I plotted it I got the following figure:



-Also when I calculated the test data root mean squared error with the resulting line I got the following output:

```
Running Mean Smoother => RMSE is 6.069909 when h is 0.37>
```

-For the kernel smoother part I used following parameters and the following sequence:

variables:
ks_bin_width <- 0.37
ks_origin <- 1.5
ks_x_max <- max(data$eruptions) #this is actually 5.1

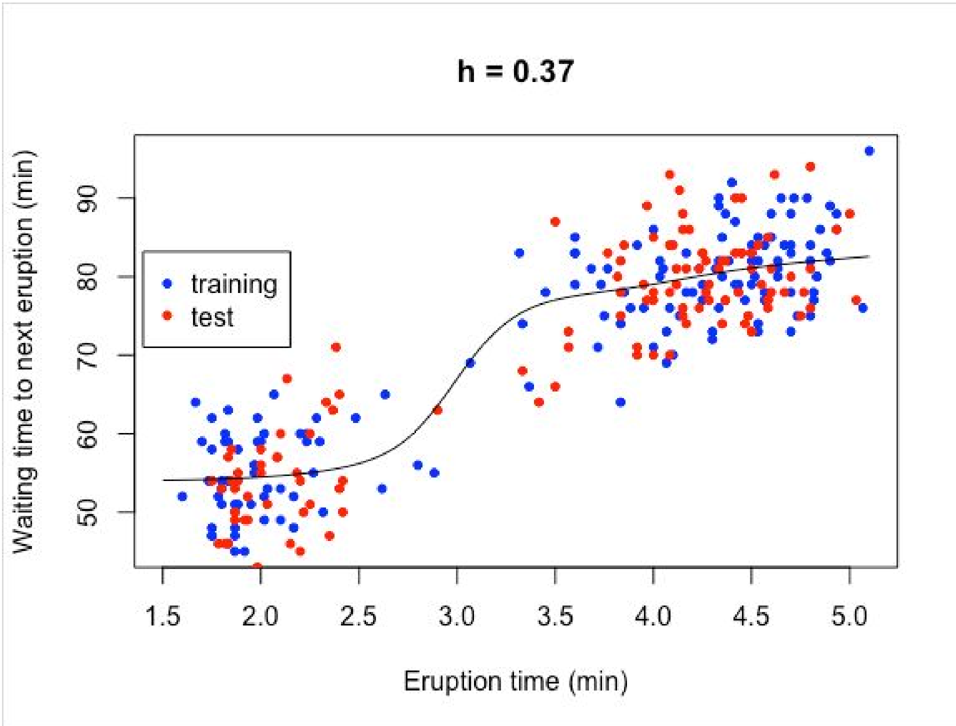
sequence:
ks_data_interval <- seq(from = ks_origin, to = ks_x_max, by = 0.01)

- Finally when I implemented kernel smoother with the following procedure:
  - -I defined two functions in the following figures:
  - -First function is defined as kernel_smth and it applies the formula in first figure.
  - -Second function is defined as apply_kernel.

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

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

- I applied training_data to apply_kernel function and the return data frame. When I ploted it I got the following figure:



-Lastly I calculated the rmse of test data waining times with kernel smoothed training data prediction I get the following rmse value:

```
Kernel Smoother => RMSE is 5.87213 when h is 0.37
```