COMP/INDR 421/521 HW04: Nonparametric Regression

Asma Hakouz 0063315

November 10, 2017

The purpose of this assignment was to implement three nonparametric regression algorithms in R.

The problem consisted of the implementation and evaluation of the following three regression algorithms:

1- **Regressogram**
2- **Running mean smoother**
3- **Kernel smoother**

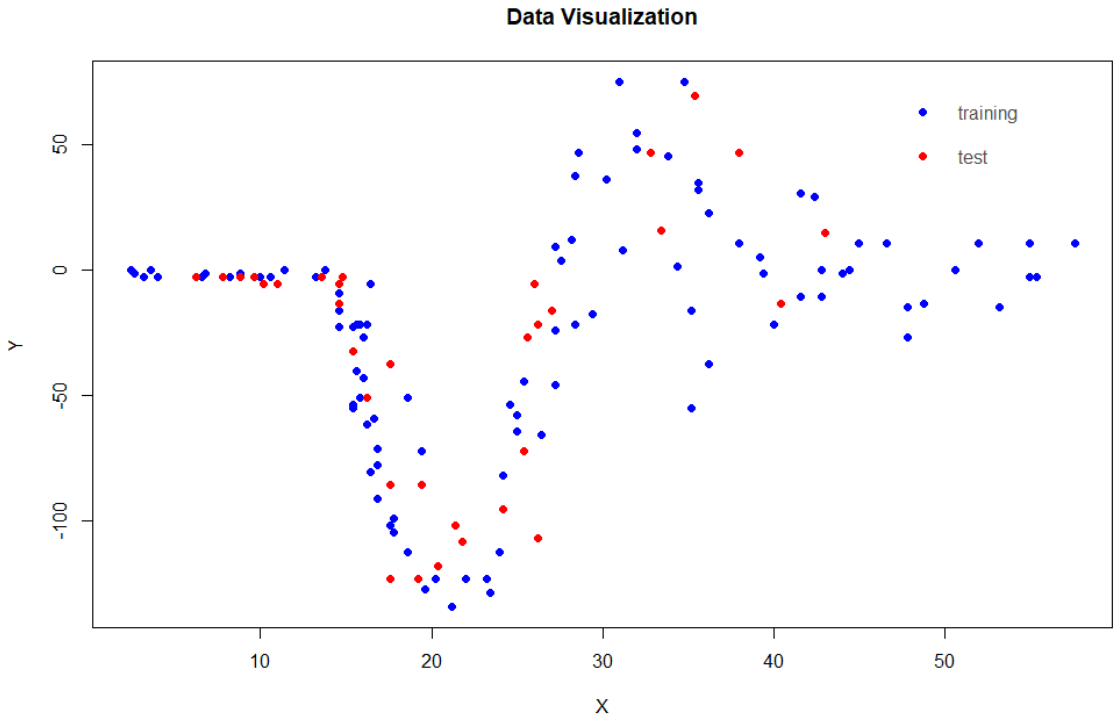
Following, more details will be discussed about each part accompanied by snippets from my source code.

- **Data Pre-processing and visualization**

```
# read data into memory
data_set <- read.csv("hw04_data_set.csv")

# get x and y values
x <- data_set$x
y <- data_set$y

# get number of samples
N <- length(y)

train_sample_count = 100

# randomly dividing the data set to training and test samples
train_indices <-  c(sample(1:N, floor(train_sample_count)))
X_train <- x[train_indices]
y_train <- y[train_indices]
X_test <- x[-train_indices]  # - means excluding
y_test <- y[-train_indices]

plot(X_train, y_train, type = "p", pch = 19, col = "blue")
points(X_test, y_test, type = "p", pch = 19, col = "red")

minimum_value <- min(x) - 2.4
maximum_value <- max(x) + 2.4
data_interval <- seq(from = minimum_value, to = maximum_value, by = 0.01)
```

This step includes reading data provided in the csv file, then randomly picking 100 out of 133 samples as the training set while the remaining 33 samples are considered as the test set.

* Note: data range is assumed to be [0, 60], i.e., the left border of the first bin will be at value 0.



- **Regressogram**

```
############## 1 - regressogram estimator ########################
### Initialization
bin_width <- 3

left_borders <- seq(from = minimum_value, to = maximum_value - bin_width, by = bin_width)
right_borders <- seq(from = minimum_value + bin_width, to = maximum_value, by = bin_width)

### Calculation
p_head <- sapply(1:length(left_borders), function(b)
    {sum(y_train[(left_borders[b] < X_train & X_train <= right_borders[b])]) /
      (sum(left_borders[b] < X_train & X_train <= right_borders[b]) + 1e-03)})
```

This small (epsilon) value was added to avoid division by zero error.

```
### Visualization
plot(X_train, y_train, type = "p", pch = 19, col = "blue", ylab = "density", xlab = "x")
points(X_test, y_test, type = "p", pch = 19, col = "red")

for (b in 1:length(left_borders)) {
  lines(c(left_borders[b], right_borders[b]), c(p_head[b], p_head[b]), lwd = 2, col = "black")
  if (b < length(left_borders)) {
    lines(c(right_borders[b], right_borders[b]), c(p_head[b], p_head[b + 1]), lwd = 2, col = "black")
  }
}

legend("topright",
       legend = c("training", "test"),
       col = c("blue",
               "red"),
       pch = c(19, 19), bty = "n", pt.cex = 1, cex = 1,
       text.col = rgb(0.3,0.3,0.3,1), horiz = F, inset = c(0.01, 0.01))
```

```
### Evaluation - RMSE calculation
rmse = sqrt(mean((y_test - p_head[ceiling(abs(X_test - minimum_value) / bin_width)]) ** 2))

sprintf("Regressogram => RMSE is %f when h is 3", rmse)
```

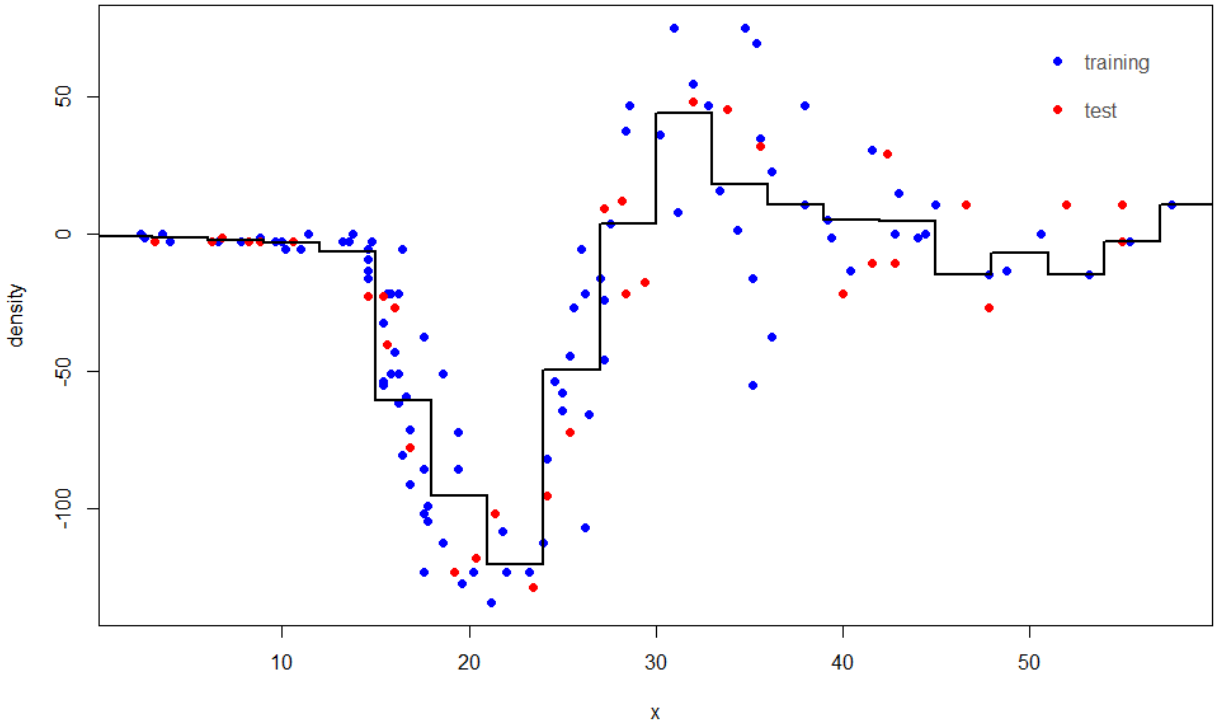
1- Implementation

This part is an implementation of the regressogram model, which is what we get if we define an origin and a bin width while averaging the label values, as in the following equation:

$$\hat{g}(x) = \frac{\sum_{i=1}^{N} b(x,\, x_i) \cdot y_i}{\sum_{i=1}^{N} b(x,\, x_i)}$$

Where $b(x, x_i) = \begin{cases} 1, & if\ x_i\ is\ in\ the\ same\ bin\ as\ x \\ 0, & otherwise \end{cases}$

2- Visualization



3- Evaluation

The following was the rmse value for one of the trials:

```
> sprintf("Regressogram => RMSE is %f when h is 3", rmse)
[1] "Regressogram => RMSE is 20.179300 when h is 3"
>
```

But since in the first step we are sampling randomly to divide the data set into training and testing sets, I took the average of multiple trials to calculate the average RMSE value for this implementation of regressogram model. Following are the code and the result of calculating average RMSE for regressogram.

```
sum_rmse <- 0
for(f in 1:1000){
  # randomly dividing the data set to training and test samples
  train_indices <-  c(sample(1:N, floor(train_sample_count)))
  X_train <- x[train_indices]
  y_train <- y[train_indices]
  X_test <- x[-train_indices]  # - means excluding
  y_test <- y[-train_indices]

  ### Calculation
  p_head <- sapply(1:length(left_borders), function(b)
  {sum(y_train[(left_borders[b] < X_train & X_train <= right_borders[b])]) /
      (sum(left_borders[b] < X_train & X_train <= right_borders[b]) + 1e-03)})
  ### Evaluation - RMSE calculation
  rmse = sqrt(mean((y_test - p_head[ceiling(abs(X_test - minimum_value) / bin_width)]) ** 2))
  sum_rmse <- sum_rmse + rmse
}
average_rmse <- sum_rmse/1000;
print(average_rmse)
```

```
> print(average_rmse)
[1] 26.69672
```

- **Running Mean Smoother**

```
############## 2 - running mean smoother ##########################
### Initialization
bin_width <- 3
p_head <- sapply(data_interval, function(x)
  {sum(y_train[(x - 0.5 * bin_width) < X_train & X_train <= (x + 0.5 * bin_width)]) /
    (sum((x - 0.5 * bin_width) < X_train & X_train <= (x + 0.5 * bin_width))+ 1e-03)})

### Visualization
plot(X_train, y_train, type = "p", pch = 19, col = "blue", ylab = "density", xlab = "x")
points(X_test, y_test, type = "p", pch = 19, col = "red")
lines(data_interval, p_head, type = "l", lwd = 2, col = "black")

legend("topright",
       legend = c("training", "test"),
       col = c("blue",
               "red"),
       pch = c(19, 19), bty = "n", pt.cex = 1, cex = 1,
       text.col = rgb(0.3,0.3,0.3,1), horiz = F, inset = c(0.01, 0.01))

### Evaluation - RMSE calculation
rmse = sqrt(mean((y_test - p_head[ceiling(X_test/data_step)]) ** 2))
sprintf("Running mean smoother => RMSE is %f when h is 3", rmse)
```

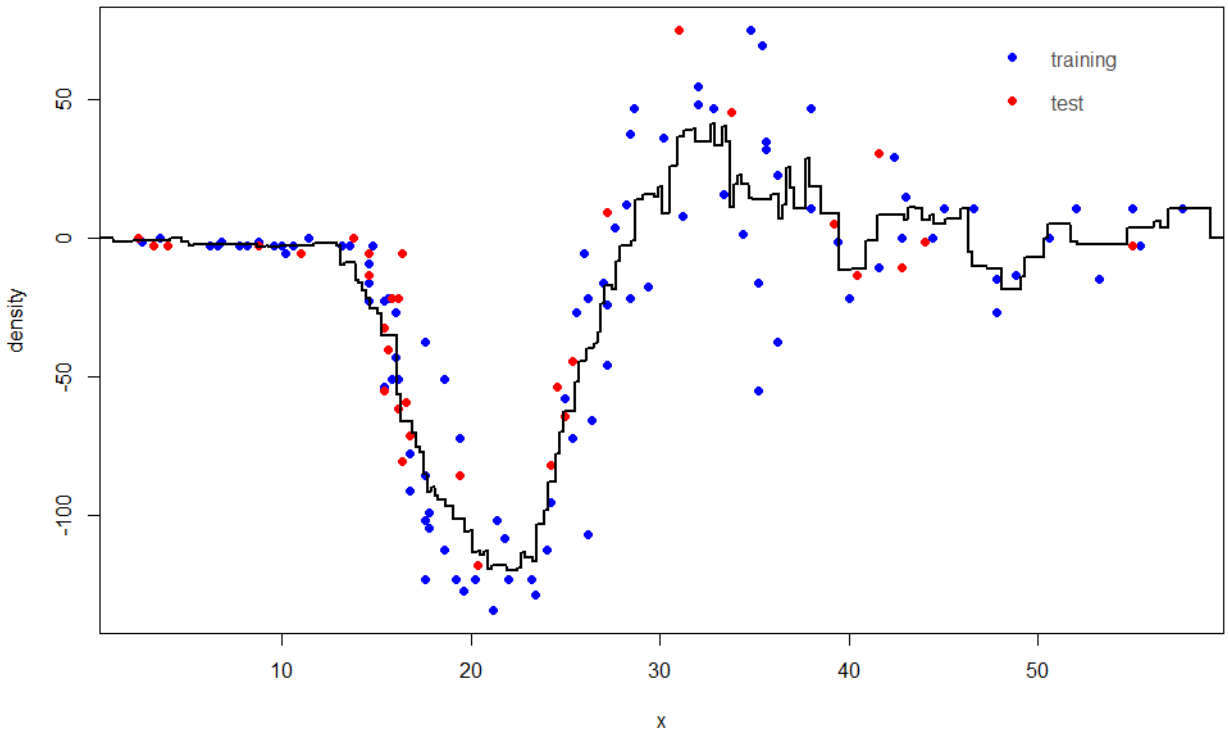
1- <u>Implementation</u>

This part is an implementation of the running mean smoother model, which is what we get if we define a bin symmetric around x while averaging the label values, as in the following equation:

$$\hat{g}(x) = \frac{\sum_{i=1}^{N} w\left(\frac{x - x_i}{h}\right) \cdot y_i}{\sum_{i=1}^{N} w\left(\frac{x - x_i}{h}\right)}$$

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

2- Visualization



3- Evaluation

The following was the rmse value for one of the trials:

```
> sprintf("Running mean smoother => RMSE is %f when h is 3", rmse)
[1] "Running mean smoother => RMSE is 18.899807 when h is 3"
```

But since in the first step we are sampling randomly to divide the data set into training and testing sets, I took the average of multiple trials to calculate the average RMSE value for this implementation of running mean smoother model. Following are the code and the result of calculating average RMSE for running mean smoother.

```
> sprintf("Running mean smoother => Average RMSE is %f when h is 3", print(average_rmse))
[1] 25.04194
[1] "Running mean smoother => Average RMSE is 25.041943 when h is 3"
```

- **Kernel Smoother**

```
############## 3 - Kernel smoother########################################
bin_width <- 1
p_head <- sapply(data_interval, function(x)
  {sum((1 / sqrt(2 * pi)) * exp(-0.5 * ((x - X_train)/ bin_width) ** 2) * y_train) /
    (sum((1 / sqrt(2 * pi)) * exp(-0.5 * ((x - X_train)/ bin_width) ** 2)))})

plot(X_train, y_train, type = "p", pch = 19, col = "blue", ylab = "density", xlab = "x")
points(X_test, y_test, type = "p", pch = 19, col = "red")
lines(data_interval, p_head, type = "l", lwd = 2, col = "black")

legend("topright", legend = c("training", "test"),
       col = c("blue", "red"), pch = c(19, 19), bty = "n", pt.cex = 1, cex = 1,
       text.col = rgb(0.3,0.3,0.3,1), horiz = F, inset = c(0.01, 0.01))

rmse = sqrt(mean((y_test - p_head[ceiling(X_test/0.01)]) ** 2))

## RMSE calculation
sprintf("Kernel Smoother => RMSE is %f when h is 1", rmse)
```

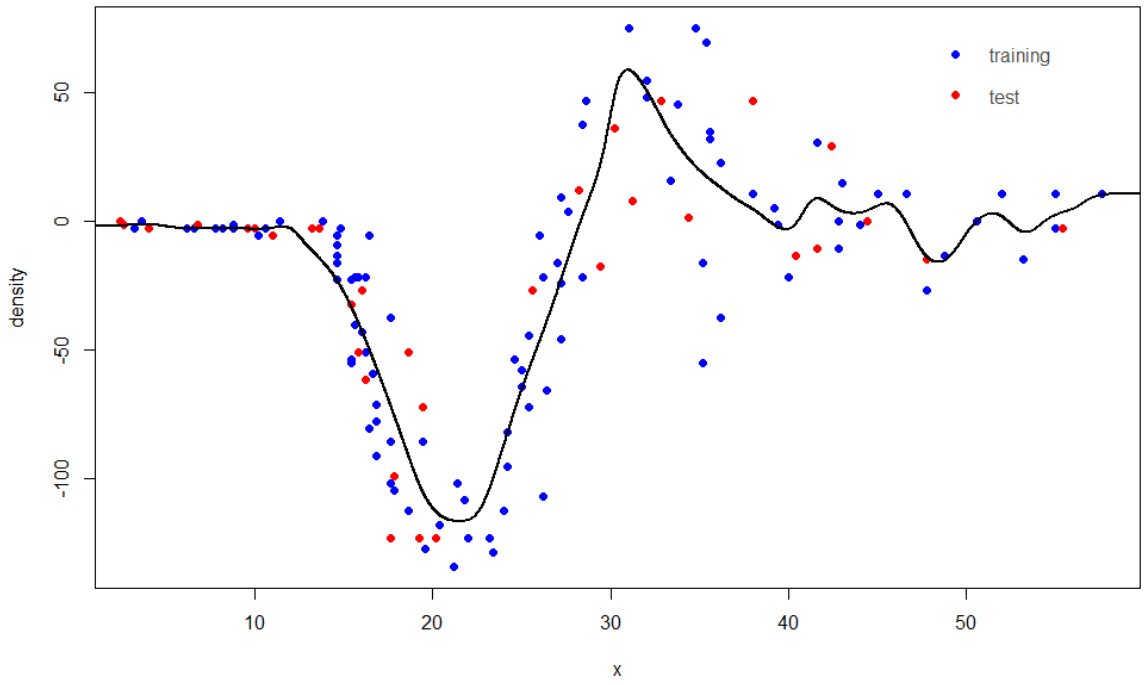
1- Implementation

This part is an implementation of the kernel smoother model as in the following equation:

$$\hat{g}(x) = \frac{\sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right) \cdot y_i}{\sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right)}$$

Where $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(u)^2}{2}}$

2- Visualization



3- Evaluation

The following was the rmse value for one of the trials:

```
> sprintf("Kernel Smoother => RMSE is %f when h is 1", rmse)
[1] "Kernel Smoother => RMSE is 21.979471 when h is 1"
>
```

But since in the first step we are sampling randomly to divide the data set into training and testing sets, I took the average of multiple trials to calculate the average RMSE value for this implementation of kernel smoother model. Following are the code and the result of calculating average RMSE for kernel smoother.

```
> sprintf("Kernel smoother => Average RMSE is %f when h is 1", print(average_rmse))
[1] 25.75503
[1] "Kernel smoother => Average RMSE is 25.755028 when h is 1"
>
```