Lab Report in Machine Learning

Laboration 3

TDDE01

David Grönberg
Davgr686

16-12-2019

Contents

List of Figures

1. Assignments

1.1 Assignment 1

The task of this assignment is to predict hourly temperatures for a given date and place (defined by latitude and longitude coordinates) in Sweden. SMHI has provided us with temperature measurements for previous dates and places. This is done by the use of a kernel that is the sum of three Gaussian kernels:

- One that accounts for the euclidean distance between two stations. (denoted *distance_kernel*)
- One that accounts for the difference in days between two temperature measurements. (denoted *date_kernel*)
- One that accounts for the difference in time between two temperature measurements. (denoted *time_kernel*)

The chosen smoothing coefficients for each Gaussian kernel are presented below:

- *distance_kernel*: $h = 50,000$
- *date_kernel*: $h = 7$
- *time_kernel*: $h = 6$

To confirm we have chosen appropriate smoothing coefficients, the kernels width are plotted. Here, we use *2011-01-05* and *12:00* as the target date and time with coordinates representing a place in Stockholm.

Figure 1.1, 1.2, 1.3 shows a plot representing how sensible each kernel is. In figure 1.2 - where the x-axis is the day of the year and y-axis is the weight - the kernel gives points that are close to the target date higher values (i.e. days of the year that are close to 5). This smoothing coefficient seems reasonable to have because the weather can drastically change in a month and we would like to capture that change and take it into account in our model.

Figure 1.3 shows the width of the *time_kernel*. Here, the x-axis represents the hour of day, ranging from 0 to 23 and our target time is *12*. We can see that the kernel assigns higher values to times that are closer to our target time. As previously stated, we chose a smoothing coefficient of *6* here. This is because we want to catch the smaller change in temperature between day time and night time, but still believe all hours has some kind of relevance.

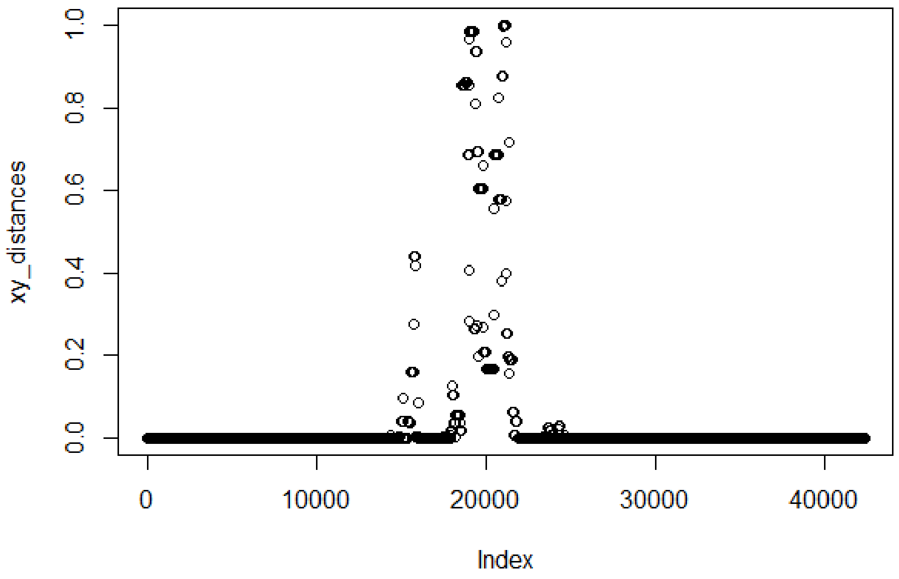

Figure 1.1: Width of *distance_ kernel*

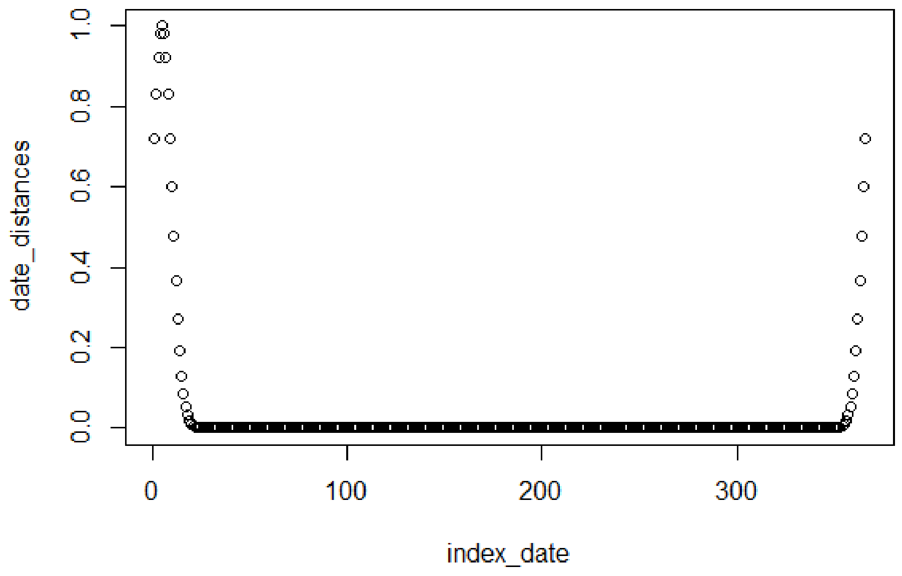

Figure 1.2: Width of *date_kernel*

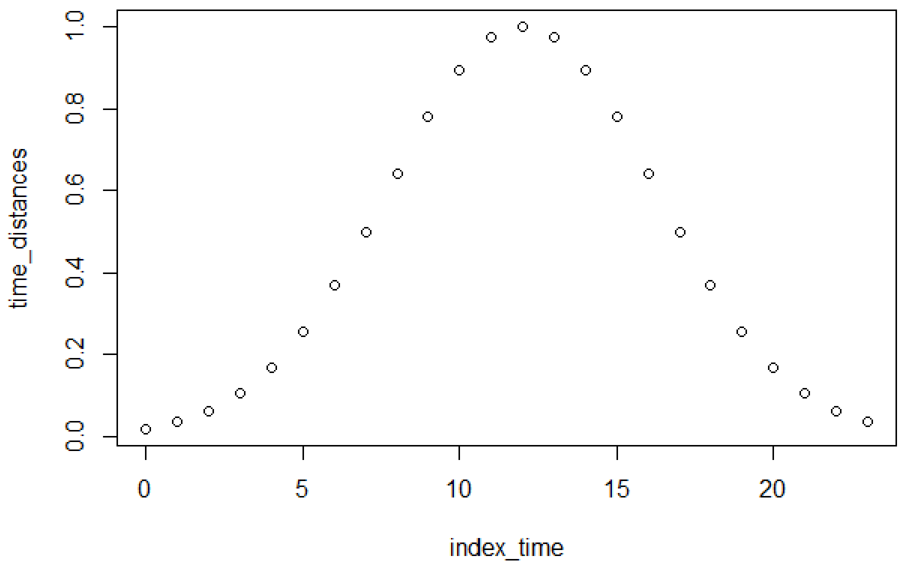

Figure 1.3: Width of *time_kernel*

Figure 1.4 shows the predicted temperature for different times on the target date by summing the kernels. The model predicts temperatures between 3.5 and 5.0. These predictions can be plausible temperatures in the beginning of January, even though they seem rather high. We can see that the model predicts lower temperatures at night, compared to day time.

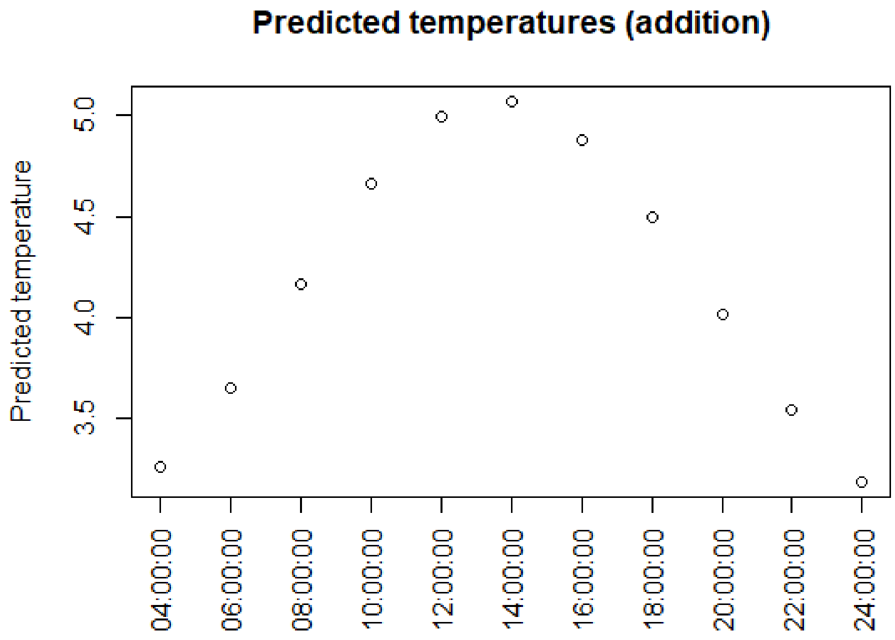

Figure 1.4: Temperature predictions using addition method

Next, instead of summing the kernels, we now multiply them and plot the results. Figure 1.5 shows the predicted temperatures for each time on the target date. Now we observe values ranging between -3 and -2, which seem more plausible for the target date. However, when inspecting the change in predicted temperature depending on time of day - the results seem rather strange. The model predicts that the lowest temperature will be observed around 08.00 and the highest temperature observed around 20.00 in the evening.

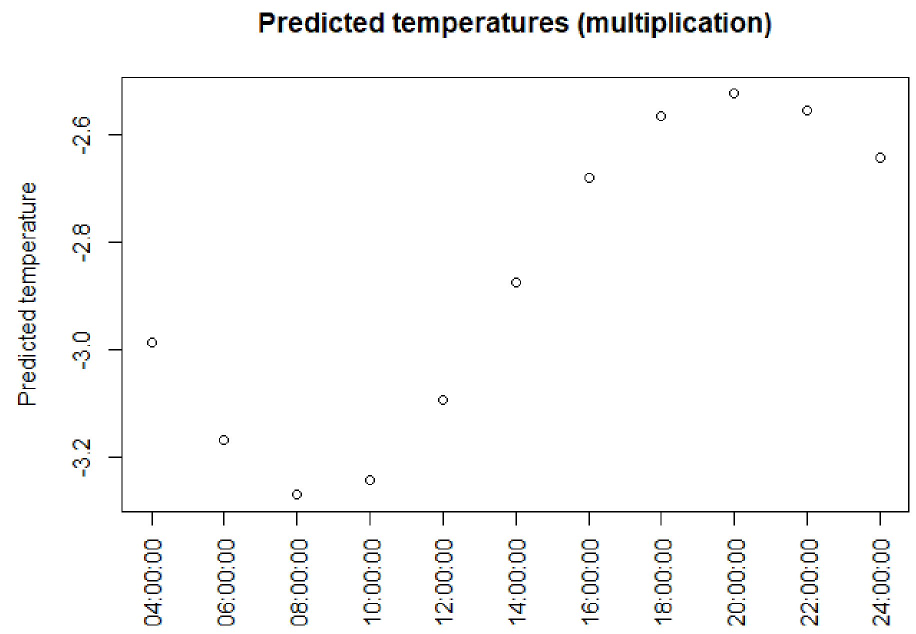

Figure 1.5: Temperature predictions using multiplication method

To verify that the model takes the time of year into account, we make the model predict temperatures for every month of the year. Figure 1.6 shows that the model using addition predicts temperatures ranging only between 5-6, while figure 1.7 shows that the model using multiplication predicts temperatures with a larger range, yielding more plausible predictions.

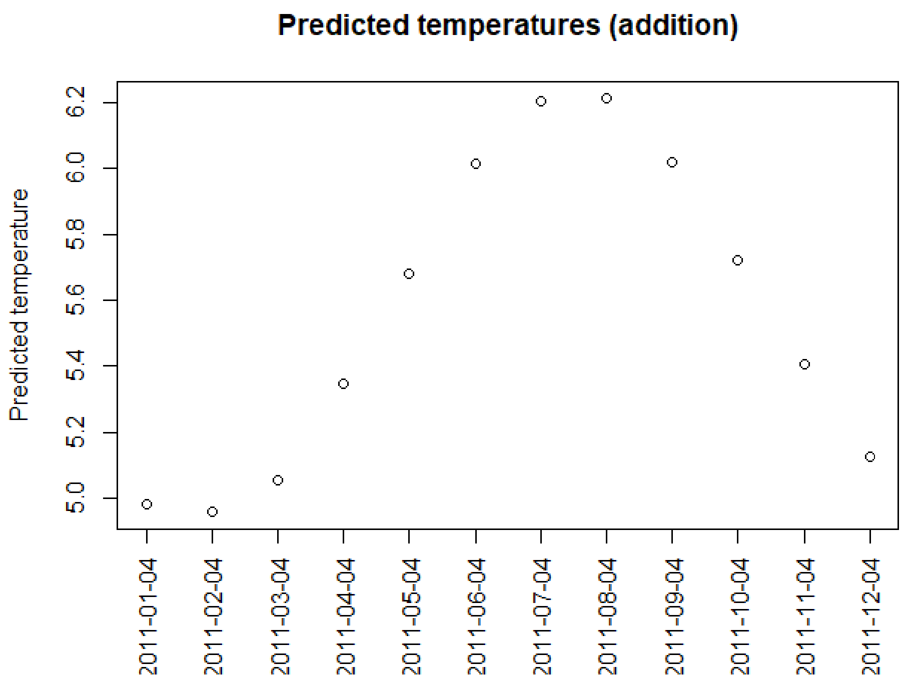

Figure 1.6: Temperature predictions on dates using addition method

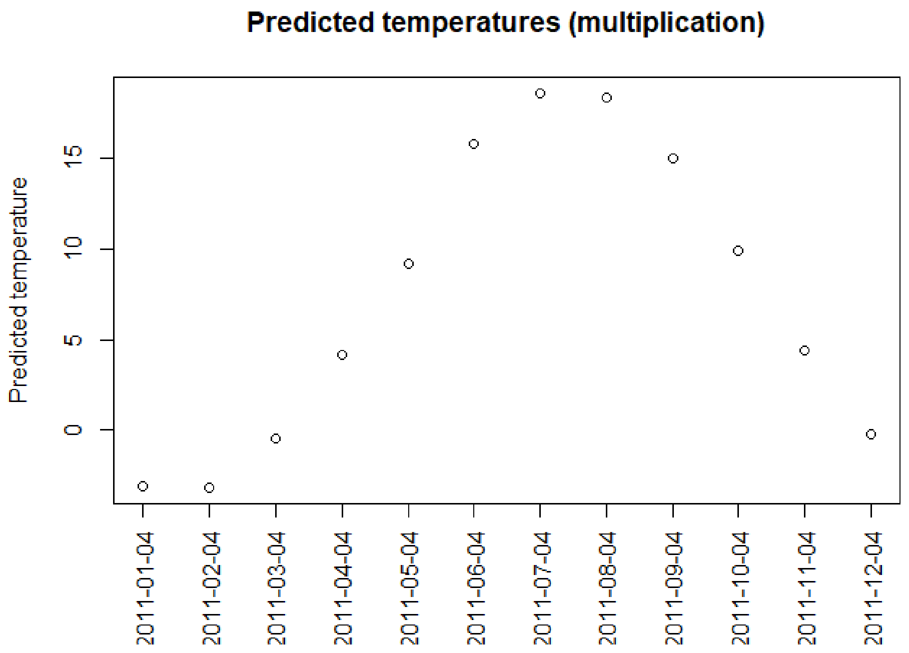

Figure 1.7: Temperature predictions on dates using multiplication method

1.2 Assignment 3

The task of this assignment is to train a neural network to learn the trigonometric sine function. First, a dataset containing mappings of variables and their corresponding sine output is divided into a train and validation set. Then, the *neuralnet* function is used to train the neural network, with a single hidden layer with 10 units.

To find the optimal threshold, we consider threshold values: $i/1000$ where $i = 1,...,10$ and then train a neural network with each threshold value using the train set and compute the Mean Square Error (MSE) on the validation set.

Figure 1.8 shows the Mean Squared Error on the validation set for each threshold value considered. The plot shows the lowest observed MSE when the threshold is 0.001, 0.003 and 0.008. After further investigation we find the lowest MSE when the threshold is 0.001.

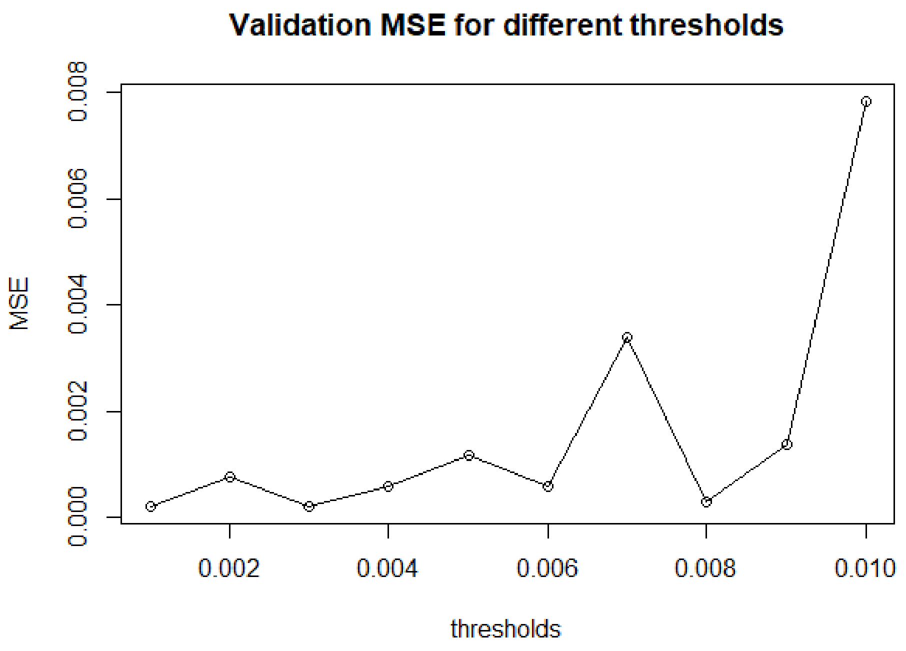

Figure 1.8: MSE for different threshold values

Next, we fit a neural network with the optimal threshold and report its structure.

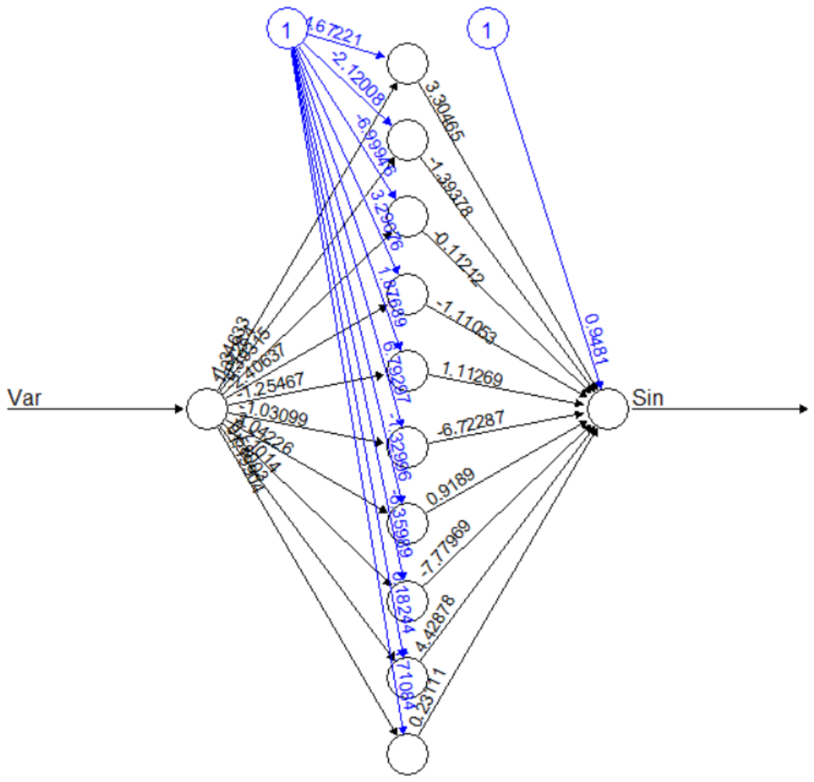

Figure 1.9: Optimal neural network

Lastly, the optimal neural network's predictions are plotted with the generated dataset in figure 1.10, where the predictions are coloured blue and the data is coloured red. As the plot shows, the majority of the points overlap, indicating that the neural network has high accuracy when predicting the sine function.

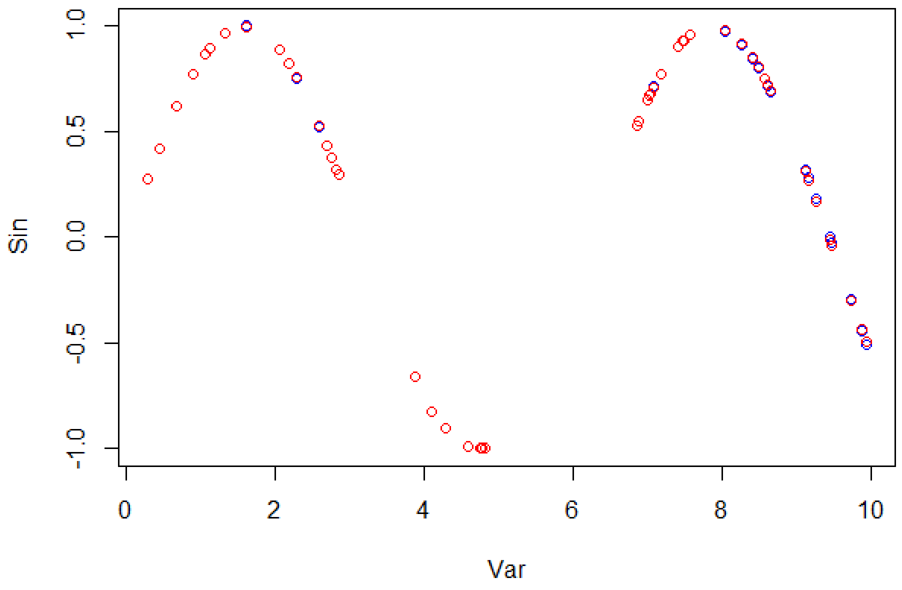

Figure 1.10: Predicted values and data

2. Code Appendix

2.1 Assignment 1

```r
library(geosphere)
library(lubridate)
require(stats)

set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")


h_distance <- 50000
h_date <- 7
h_time <- 6

target_point = c(18.0574, 59.3420) # sthlm
pred_date <- "2011-01-05" # The date to predict
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00", "16:00:00",
     "18:00:00", "20:00:00", "22:00:00", "24:00:00")
temp <- vector(length=length(times))
# Students code here
df_before <- subset(st, as.Date(date) < as.Date(pred_date))


get_date_diff <- function(day1, day2, h_date) {
  diff <- abs(yday(day1) - yday(day2))
  if (diff > 182) {
    return (abs(diff - 365)/h_date)
  }
  return (diff/h_date)
}

get_time_diff <- function( time2, time1, h_time) {
  time1.numeric <- as.numeric(substring(time1,1,2))
  diff <- abs(time1.numeric - time2)
  if (diff > 12) {
    return (abs(diff - 24)/h_time)
  }
  return (diff/h_time)
}

get_xy_diff <- function(target, x, h_distance) {
  x_lat <- as.numeric(x[4])
  x_long <- as.numeric(x[5])
  dist <- distHaversine(c(x_long, x_lat), target)
  return (dist[1]/h_distance)
}
```

```r

euc_norm <- function(x) return (sqrt(sum(x^2)))


date_kernel <- function(X, target, h_date) {
  u <- abs(sapply(X$date, function(x) get_date_diff(target, x, h_date)))
  k <- exp(-u^2)
  return (k)
}


time_kernel <- function(X, target, h_time) {
  u <- abs(sapply(X$time, function(x) get_time_diff(target, x, h_time)))
  k <- exp(-u^2)
  return (k)
}

distance_kernel <- function(X, target, h_distance) {
  u <- abs(apply(X, 1, function(x) get_xy_diff(target, x, h_distance)))
  k <- exp(-u^2)
  return (k)
}

xy_distances <- distance_kernel(df_before, target_point, h_distance)
plot(xy_distances)

index_date <- c(yday(df_before$date))
date_distances <- date_kernel(df_before, pred_date, h_date)
plot(y = date_distances, x = index_date)

index_time <- as.numeric(substring(df_before$time,1,2))
time_distances <- time_kernel(df_before, 12, h_time)
plot(y = time_distances, x = index_time)

denom = c()
nomi = c()


pred_dates <- c("2011-01-04", "2011-02-04", "2011-03-04", "2011-04-04", "2011-05-04", "
    2011-06-04", "2011-07-04", "2011-08-04", "2011-09-04", "2011-10-04","2011-11-04", "
    2011-12-04")

pred = c()
for (x in 1:length(pred_dates))
{
  temp_date_distances <- date_kernel(df_before, pred_dates[x], h_date)
  for (i in 1:length(df_before$date)) {
    denom[i] <- (temp_date_distances[i] + time_distances[i] + xy_distances[[i]])
    nomi[i] <- (denom[i] * df_before$air_temperature[i])
  }
  pred[x] <- sum(nomi) / sum(denom)

}


plot(pred, axes=FALSE, xlab="", ylab="Predicted temperature", main="Predicted temperatures (
    addition)")
axis(2)
axis(1, at=seq_along(pred),labels=as.character(pred_dates), las=2)
box()'


```

```r

pred = c()
for (x in 1:length(times))
{
  temp_time <- as.numeric(substring(times[x], 1, 2))
  temp_time_distances <- time_kernel(df_before, temp_time, h_time)
  for (i in 1:length(df_before$date)) {
    denom[i] <- (date_distances[i] * temp_time_distances[i] * xy_distances[[i]])
    nomi[i] <- (denom[i] * df_before$air_temperature[i])
  }
  pred[x] <- sum(nomi) / sum(denom)

}

plot(pred, axes=FALSE, xlab="", ylab="Predicted temperature", main="Predicted temperatures (
    multiplication)")
axis(2)
axis(1, at=seq_along(pred),labels=as.character(times), las=2)
box()
```

2.2 Assignment 3

```r
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
#plot(tr)
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(10, min=-1, max=1)# Your code here
MSE = c()
thresholds = c()


for(i in 1:10) {
  nn <- neuralnet(Sin ~ Var, data=tr, startweights = winit, threshold = i/1000, hidden = c(10)
    )
  pred <- predict(nn, va)
  MSE[i] <- mean((pred - va$Sin)^2)
  thresholds[i] <- (i/1000)
}

plot(y=MSE, x=thresholds, type = 'o', main="Validation MSE for different thresholds")
best_threshold <- thresholds[which.min(MSE)]

nn <- neuralnet(Sin ~ Var, data=trva, startweights = winit, threshold = best_threshold, hidden
     = c(10))
plot(nn)
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1,  col = "blue", main="Predicions")
points(trva, col = "red")
```