Introduction to Machine Learning - Lab 3 Block 1 Group Report

Milda Poceviciute, Henrik Karlsson, Ugurcan Lacin

15 December 2017

Q1. KERNEL METHODS

In this part we will implement the kernel method by summing and multiplying three gaussian kernels. First of all, the data has to be prepared - posterior observations are removed and the variables initialised. We will be preparing the weather forecast for the Gothenburg's station at the date of 1987-07-28.

The overall approach to solving this task is to compute the distance for each point in the data set and then weight each point, to adjust how much each point affect the prediction. The closer distance between the point and an observation within the data, the stronger weight. The weight have a range from 0 to 1, were 0 mean no effect on the prediction (to large distance between observation and defined point) and 1 is identical as defined point (very short distance between point and observation).

Here distance is measured in 3 diffrent ways, geographically, in days and time of the day. In this task, I've not taken the fact that temperature over the year is cyclical into consideration, so the distance in days is computed in absolut numbers between two dates and not as order of days on the year that probably would give a better result.

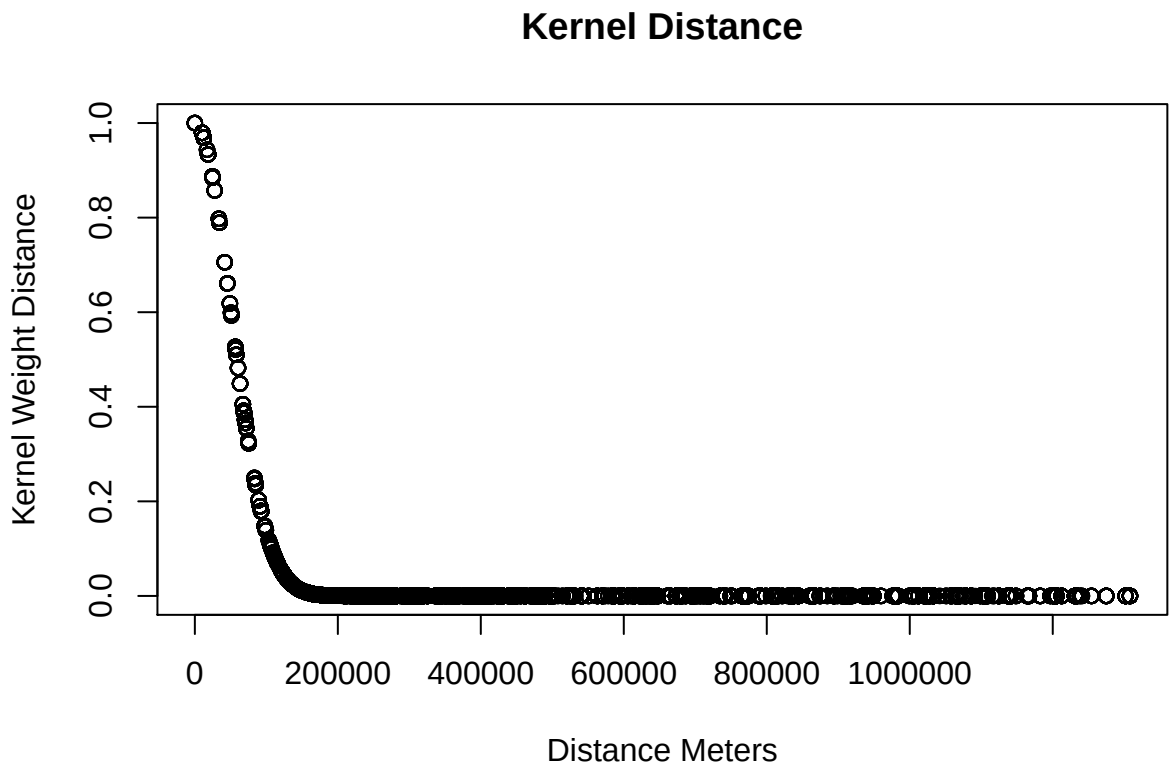
```
set.seed(1234567890)
library(geosphere)
library(lubridate)

a <-57.7084 # Latitude
b <- 11.9939 # Longitude
date1 <- as.Date("1987-07-28","%Y-%m-%d")  # The date to predict (up to the students)
times <- c(seq(0,22,2))
#times <- 4
temp <- vector(length=length(times)) # predictions by summation of kernels
temp2 <- vector(length=length(times)) # predictions by product of kernels


# Prepare data frames
stations_all <- read.csv("stations.csv",fileEncoding = "ISO-8859-1")
temps_all <- read.csv("temps50k.csv",fileEncoding = "ISO-8859-1")
temps_all$date <- as.Date(temps_all$date,"%Y-%m-%d")

# Subest data so only past data is used in the prediction
temporary <- subset(temps_all, temps_all$date  < date1)
stations <- subset(stations_all, stations_all$station_number %in% temporary$station_number)

# Add stations locations to temps data frame
temps <- merge(temporary, stations, by.x = c( "station_number"),
                  by.y = c("station_number"))
```

Next, the kernels are computed for distance, time and date:

```r
# h values
h_distance <-  50000 # var(distance) = 105282451451
h_time <- 3 # var(time) = 42.69921
h_date <-  7 # var(date) = 35975135

# gaussian kernal function
k_gaussian = function(u, h){
    result <- c()
    result <- unlist(lapply(u, function(x){ exp((-1)*((x^2)/(2*(h^2))))}))
    return(result)
}


# the distance from a station to the point of interest
longlat <- data.frame(long = temps[,9], lat = temps[,8])

u1 <- as.vector(distHaversine(p1=longlat, p2=c(b,a), r=6378137))

k1 <- k_gaussian(u1, h = h_distance)

k1_df <- data.frame(u = u1, kernel = k1)
# plot(k1_df$u,k1_df$kernel)
#k1_df[order(k1_df$u, decreasing = TRUE),]


# the distance between the day a temperature measurement was made and the day of interest

# number of days between
days <- unlist(lapply(temps$date, function(x){d <- as.numeric(date1-x)} ))

k2 <- k_gaussian(days, h = h_date)
k2_df <- data.frame(u = days, kernel = k2)



# distance between the hour of the day a temperature measurement was made and the hour of interest
hours <- as.numeric(substr(temps$time, 1, 2))
for (i in 1: length(times)){
    u3 <- unlist(lapply(hours, function(x){times[i]-x}))

    k3 <- k_gaussian(u3, h = h_time )

    if (i == length(times)){
     k3_df <- data.frame(u = u3, kernel = k3)
    }
    # plot(k3_df$u,k3_df$kernel)

    # Summing the kernels
    final_k <- (k1 + k2  + k3)/3
    t <- temps$air_temperature

    # Prediction:
    temp[i] <- sum(final_k*t)/sum(final_k)
```

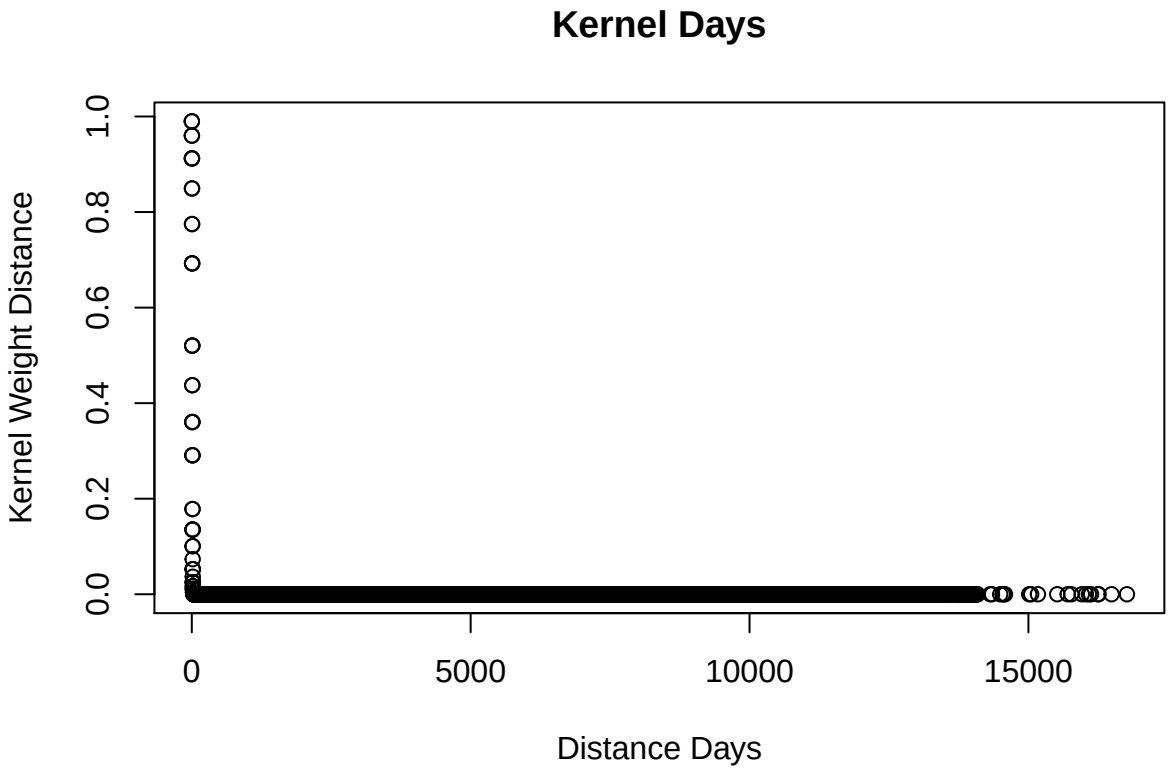
```
    # Multiplying kernels
    final2_k <- k1 * k2  * k3

    # Prediction:
    temp2[i] <- sum(final2_k*t)/sum(final2_k)

}
```

The width for each kernel was chosen after testing different values. When the widths were set to be the variances of each variable (observations of distances, times and dates), the kernel functions included most of the observations into prediction - and the predicted value of the temperature was usually close to the mean value in the database. Hence, the forecasts was heavily influenced by the average value of the temperature of the database. We chose smoothing factors such that they would only include the nearest observations and also would be intuitive. From my experience, it seems reasonable to assume that the weather conditions are similar in the range of 50 km, so h_distance = 50000. Plot below shows that only closest locations receive non-zero weights:

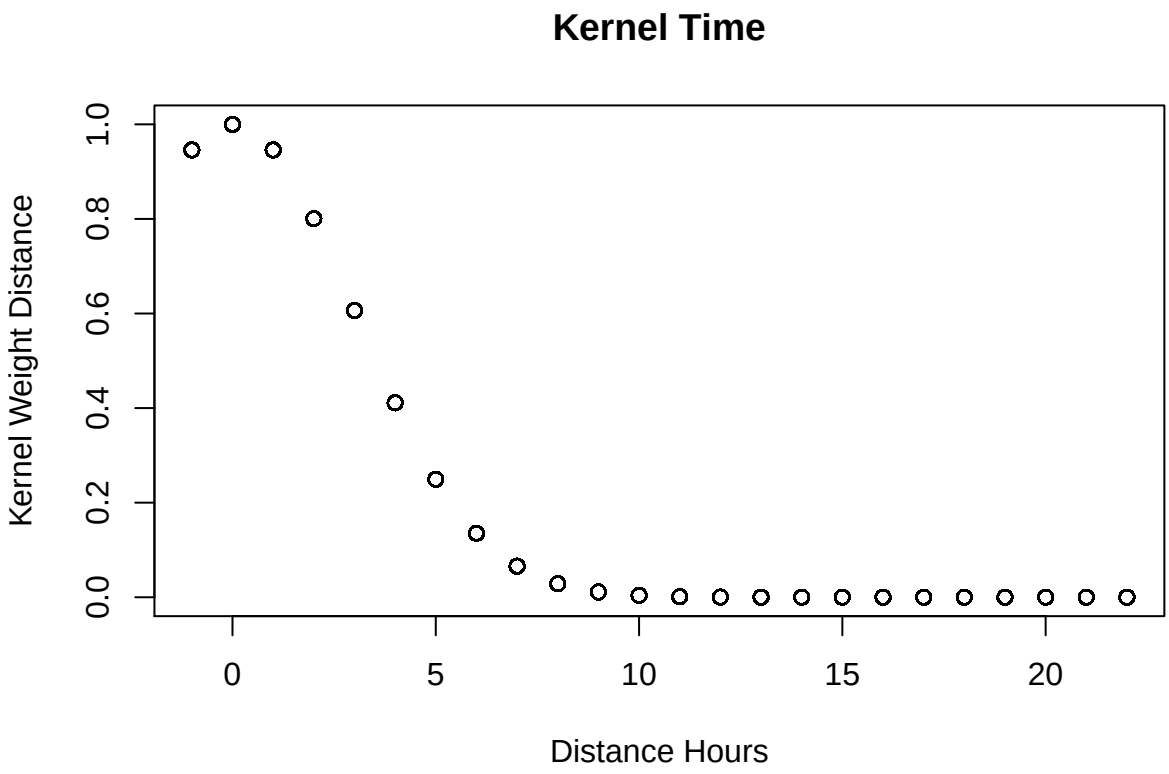


Also we assume that the weather is likely to change every 3 hours (h_time = 3) and the similar weather conditions can be observed over the period of the week (h_date = 7). The plots below show the weights estimated by the two kernels discussed above:

```
## Plot of the weights choisen for each date
```

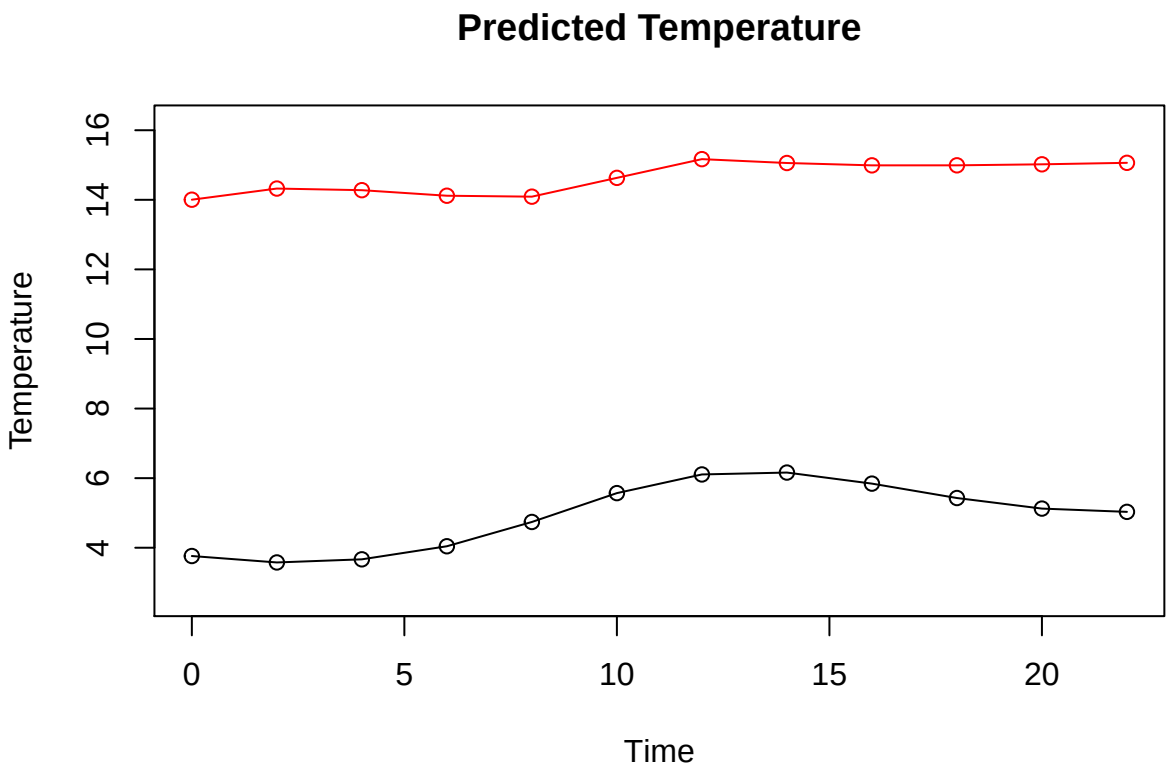



```
## Plot of the weights choisen for each time ( when predicting for hour 22
```



Finally, the forecast for the chosen date is done using summation and multiplication of the kernels. The actual temperature at this location on 1987-07-28 was 13.5 degrees at 5 am. The black line represents the predictions made by summing the kernels and the red line is the predictions made by multiplying the kernels. It is clear that the red line forecasts the temperature much closer to the actual observed temperature. The forecast by the summation of the kernels seem to be highly effected by the average temperature of the data - this means, that stations from the northen part of the Sweden has low weights for the distance kernel, but may have high weights for the time and date kernels. Hence, the summation of kernels get highly affected by the temperatures of the parts of Sweden that are actually far away from the area of forecast, but close in the respect of time and/or date. When kernels are multiplied, the low kernels have higher influence on the final product. So the observation from an area that is far away in distance but close in time and date will have lower final weight in multiplication case than in the case of suming the kernels. Hence, the multiplication of the kernels is a better way of forecasting the temperatures.

```
pred <- data.frame(Temps1 = unlist(temp), Temps2 = unlist(temp2), Time = times)
actual <- subset(temps_all, temps_all$date == date1)
plot(pred$Time, pred$Temps1, type="o", ylim=c(min(pred$Temps1)-1, max(pred$Temps2)+1), main = "Predicte
     xlab = "Time", ylab = "Temperature")
lines(pred$Time, pred$Temps2, type="o", col="red")
```



Q2. NEURAL NETWORKS

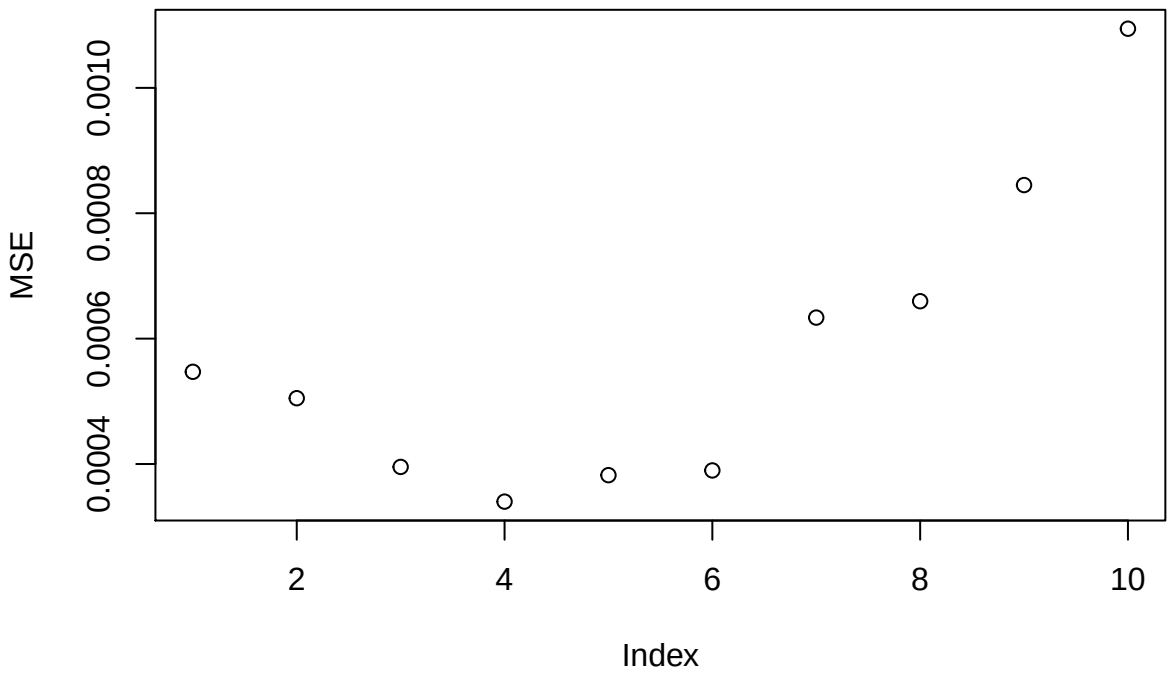
The task in this assignment is to train a neural network to predict a trigonometric sine function.

```
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 3.4.3
```

```
set.seed(1234567890)

# Create data
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31,-1,1)
MSE <- c()
for(i in 1:10) {
  nn <- neuralnet(Sin~Var, startweights = winit, hidden = 10, data=tr, threshold = i/1000)
  pred <- compute(nn,va[,1])$net.result
  MSE[i] <- (1/25) * sum((va$Sin - pred)^2)
}

opt_MSE <- which.min(MSE)
```

```
plot(MSE)
```



31 random initial weights is generated from the uniform distribution. 31 weights is needed because the neural network have 20 weights and 11 biases, when the network have 1 layer with 10 hidden nodes.

The threshold function tries to find the minimum of the partial derivative and continues to iterate until the specified threshold condition have been reached.

The method of cross-validation is used to test neural networks with different thresholds. The decision was based on computing MSE errors and choosing the model that minimises this quantity. We conlcude that the best predictions were done by the neural network with threshold = 4/1000 = 0.004. The plot of the MSE errors for different thresholds (threshold = index / 1000) is provided above. Next the optimal neural networks model is fitted on the full data set. The optimal model is:

```
nn_optimal <- neuralnet(Sin~Var, startweights = winit, hidden = 10, data=trva, threshold = 4/1000)
optimal_pred <- prediction(nn_optimal)$rep1
```

```
## Data Error:  0;
```

The predictions (black dots) vs actual values (red dots) are plotted below. The pattern of predictions seem to follow the real valus very closely, hence we conclude that the model is a good fit.

```
# Plot of the predictions (black dots) and the data (red dots)
plot(optimal_pred)
points(trva, col = "red")
```

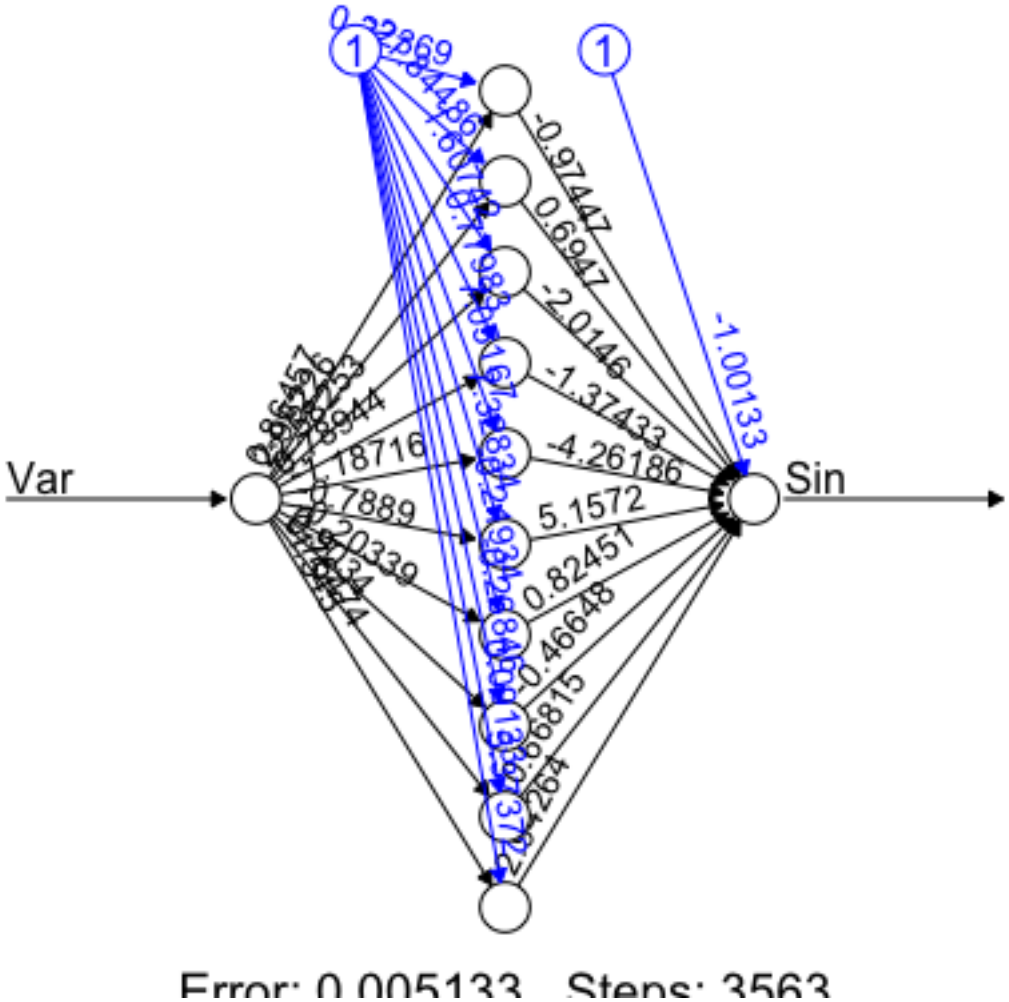
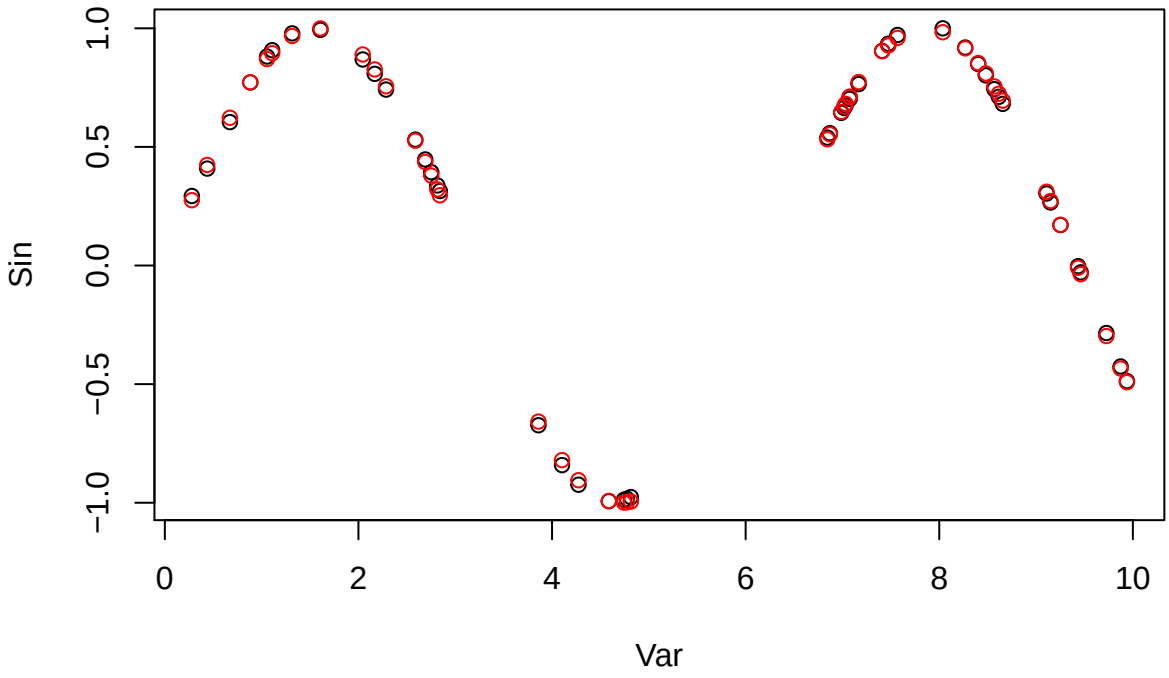


Figure 1: The final Neural Network.



Appendix

```
knitr::opts_chunk$set(echo = TRUE)
set.seed(1234567890)
library(geosphere)
library(lubridate)

a <-57.7084 # Latitude
b <- 11.9939 # Longitude
date1 <- as.Date("1987-07-28","%Y-%m-%d")  # The date to predict (up to the students)
times <- c(seq(0,22,2))
#times <- 4
temp <- vector(length=length(times)) # predictions by summation of kernels
temp2 <- vector(length=length(times)) # predictions by product of kernels


# Prepare data frames
stations_all <- read.csv("stations.csv",fileEncoding = "ISO-8859-1")
temps_all <- read.csv("temps50k.csv",fileEncoding = "ISO-8859-1")
temps_all$date <- as.Date(temps_all$date,"%Y-%m-%d")

# Subest data so only past data is used in the prediction
temporary <- subset(temps_all, temps_all$date  < date1)
```

```r
stations <- subset(stations_all, stations_all$station_number %in% temporary$station_number)

# Add stations locations to temps data frame
temps <- merge(temporary, stations, by.x = c( "station_number"),
                  by.y = c("station_number"))

# h values
h_distance <-  50000 # var(distance) = 105282451451
h_time <- 3 # var(time) = 42.69921
h_date <-  7 # var(date) = 35975135

# gaussian kernal function
k_gaussian = function(u, h){
    result <- c()
    result <- unlist(lapply(u, function(x){ exp((-1)*((x^2)/(2*(h^2))))}))
    return(result)
}


# the distance from a station to the point of interest
longlat <- data.frame(long = temps[,9], lat = temps[,8])

u1 <- as.vector(distHaversine(p1=longlat, p2=c(b,a), r=6378137))

k1 <- k_gaussian(u1, h = h_distance)

k1_df <- data.frame(u = u1, kernel = k1)
# plot(k1_df$u,k1_df$kernel)
#k1_df[order(k1_df$u, decreasing = TRUE),]


# the distance between the day a temperature measurement was made and the day of interest

# number of days between
days <- unlist(lapply(temps$date, function(x){d <- as.numeric(date1-x)} ))

k2 <- k_gaussian(days, h = h_date)
k2_df <- data.frame(u = days, kernel = k2)



# distance between the hour of the day a temperature measurement was made and the hour of interest
hours <- as.numeric(substr(temps$time, 1, 2))
for (i in 1: length(times)){
    u3 <- unlist(lapply(hours, function(x){times[i]-x}))

    k3 <- k_gaussian(u3, h = h_time )

    if (i == length(times)){
     k3_df <- data.frame(u = u3, kernel = k3)
    }
    # plot(k3_df$u,k3_df$kernel)
```

```
    # Summing the kernels
    final_k <- (k1 + k2  + k3)/3
    t <- temps$air_temperature

    # Prediction:
    temp[i] <- sum(final_k*t)/sum(final_k)

    # Multiplying kernels
    final2_k <- k1 * k2  * k3

    # Prediction:
    temp2[i] <- sum(final2_k*t)/sum(final2_k)

}
plot(k1_df$u,k1_df$kernel, main = "Kernel Distance",
     xlab = "Distance Meters", ylab = "Kernel Weight Distance")
cat("Plot of the weights choisen for each date")
cat("\n")
plot(k2_df$u,k2_df$kernel, main = "Kernel Days",
     xlab = "Distance Days", ylab = "Kernel Weight Distance")
cat("\n")
cat("Plot of the weights choisen for each time ( when predicting for hour 22")
cat("\n")
plot(k3_df$u,k3_df$kernel, main = "Kernel Time",
     xlab = "Distance Hours", ylab = "Kernel Weight Distance")

pred <- data.frame(Temps1 = unlist(temp), Temps2 = unlist(temp2), Time = times)
actual <- subset(temps_all, temps_all$date == date1)
plot(pred$Time, pred$Temps1, type="o", ylim=c(min(pred$Temps1)-1, max(pred$Temps2)+1), main = "Predicte
     xlab = "Time", ylab = "Temperature")
lines(pred$Time, pred$Temps2, type="o", col="red")

library(neuralnet)
set.seed(1234567890)

# Create data
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31,-1,1)
MSE <- c()
for(i in 1:10) {
  nn <- neuralnet(Sin~Var, startweights = winit, hidden = 10, data=tr, threshold = i/1000)
  pred <- compute(nn,va[,1])$net.result
  MSE[i] <- (1/25) * sum((va$Sin - pred)^2)
}

opt_MSE <- which.min(MSE)
plot(MSE)
nn_optimal <- neuralnet(Sin~Var, startweights = winit, hidden = 10, data=trva, threshold = 4/1000)
optimal_pred <- prediction(nn_optimal)$rep1
```

```r
# Plot of the predictions (black dots) and the data (red dots)
plot(optimal_pred)
points(trva, col = "red")
```