TDDE01 MACHINE LEARNING

Lab 3 - B18 Group Report

Lawrence Thanakumar Rajappa (lawra776)
Grégoire Vola (grevo149)
Kyriakos Domanos (kyrdo817)

December 17, 2019

Assignment - 1

The objective of this assignment is to find the temperature readings between 4 AM to 24 PM in an interval of 2 hours with independent variables date and place in Sweden.

The following data are provided as input for forecasting the temperature

- **Date** : 2019-12-19
- **place**: **Longitude** :58.4108, **Latitude** :15.6214

The prediction is done using 3 kernels which are gaussian kernels and given below

- The first to account for the distance from a station to the point of interest.
- The second to account for the distance between the day a temperature measurement was made and the day of interest.
- The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

We use the below gaussian function for creating above-mentioned kernels,

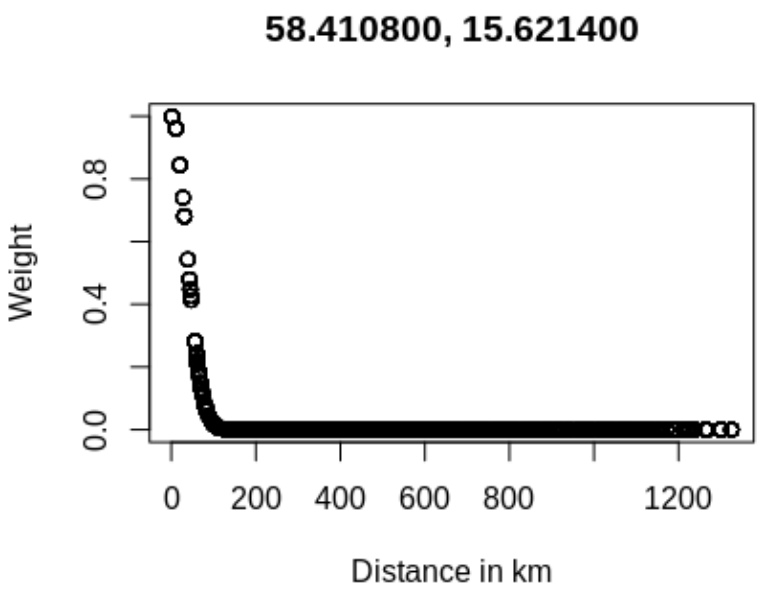
```
gaussian_kernel = function(dist, smoothing){
  return (exp(-(dist/smoothing)^2))
}
```

3 Smoothing coefficients are selected for the above kernels

- h_distance = 50 # km
- h_date = 7 # days
- h_time = 3 # hours

Geographical Kernel

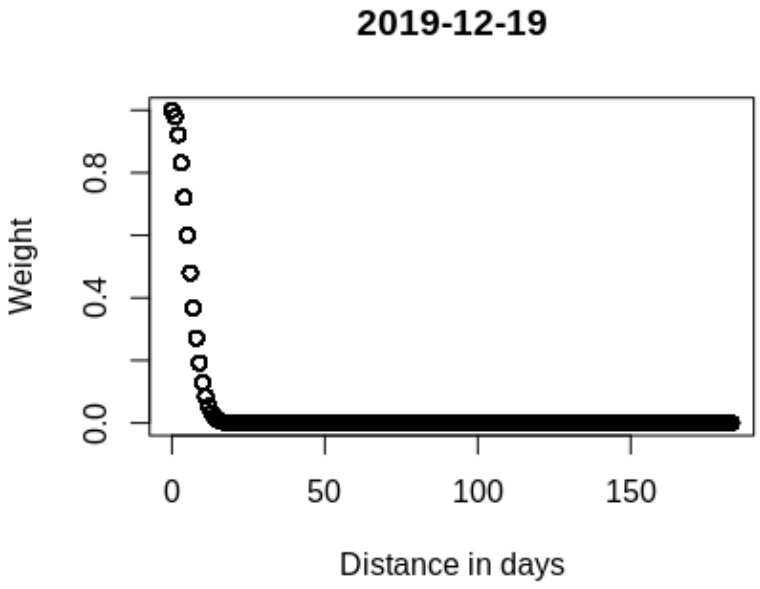
The plot below describe the weights which is provided by the 1st kernel for each of the data points based on the distance between two places



We can see from the above plot that as we move from the location we want to predict, the weights are getting reduced until they reach 0. This is the expected behaviour, as one moves away from a location, the temperature calculated becomes irrelavant for the prediction.

Day Kernel

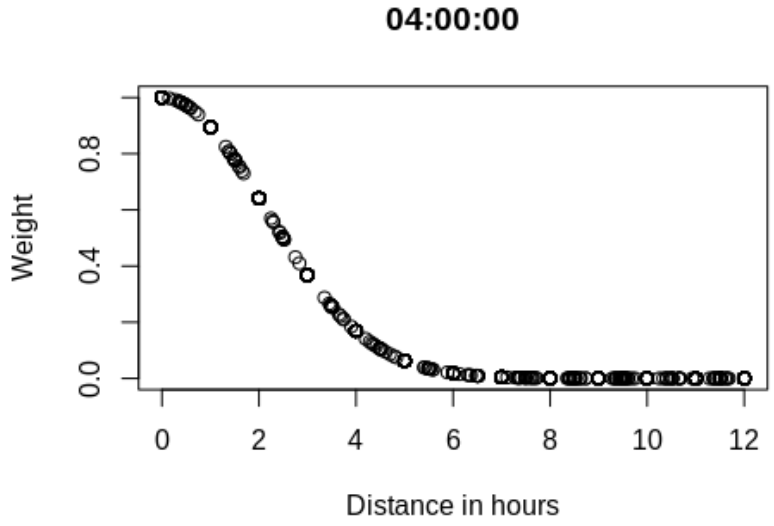
The plot below describe the weights which is provided by the 2nd kernel for each of the data points based on the difference between prediction date and temperature measured dates (*in days*).



From the above plot, we could see that as we move away from the date to predict, the weights are getting reduced until they reach 0. This is the expected behaviour, as one moves away from a day, the temperature measured becomes irrelavant for the prediction.

Hour Kernel

The plot below describe the weighs which is provided by the 3rd kernel for each of the data points based on the time difference (*in hours*)

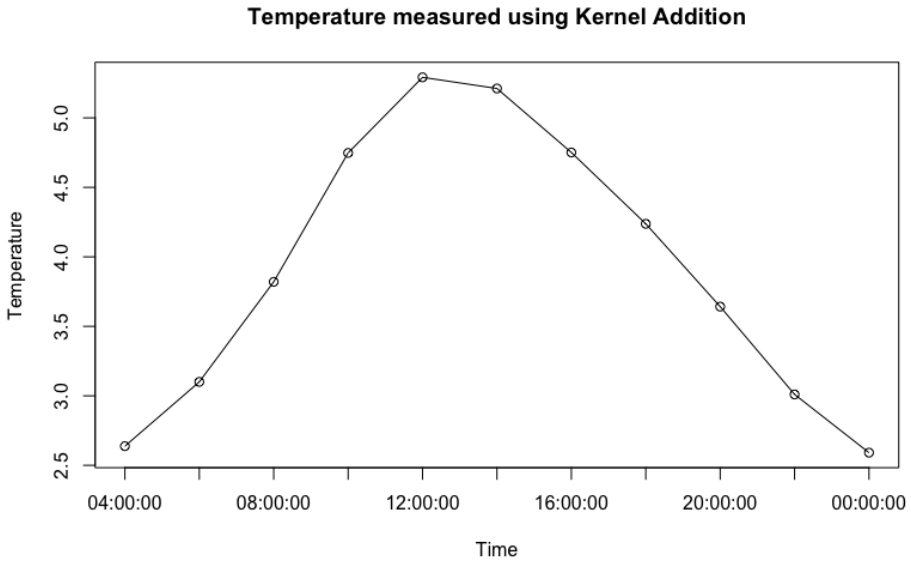


We can see from the above plot that, as we move away from the time we need to predict, the weights are reducing until they reach 0. This is the expected behaviour, as one moves away from a hour, the temperature measured becomes irrelvant for the prediction.

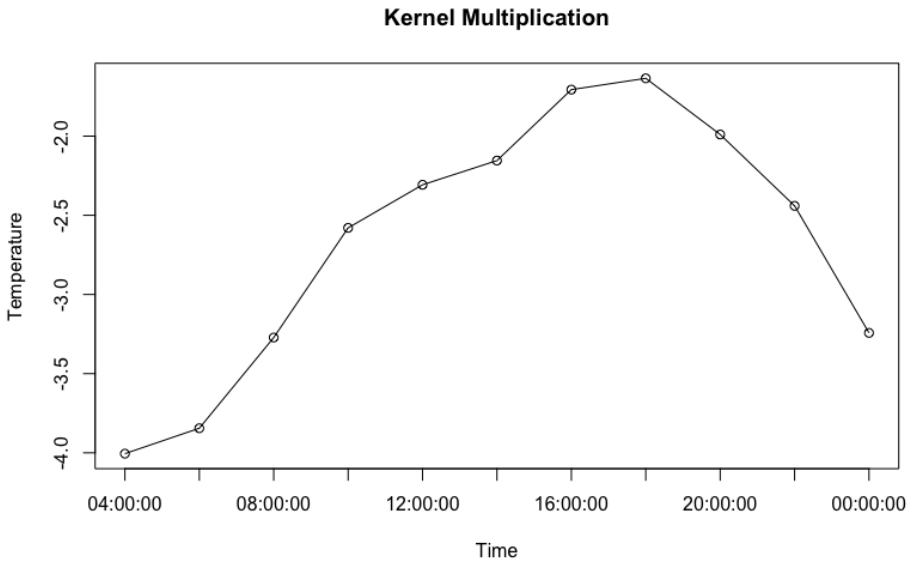
We use the below mean function to measure the temperature,

```
return(sum(dist * obs$air_temperature) / sum(dist))
```

Temperature Prediction for 2019-12-19-Kernel Addition



Temperature Prediction for 2019-12-19-Kernel Multiplication



We predict the temperature by using the weighted mean of all observations temperatures. By looking the above two plots, we can see that multiplied kernel is better than summed kernel, because summed kernel provides higher MSE error rate when compared with multiplied kernel.

Assignment - 1 Annex

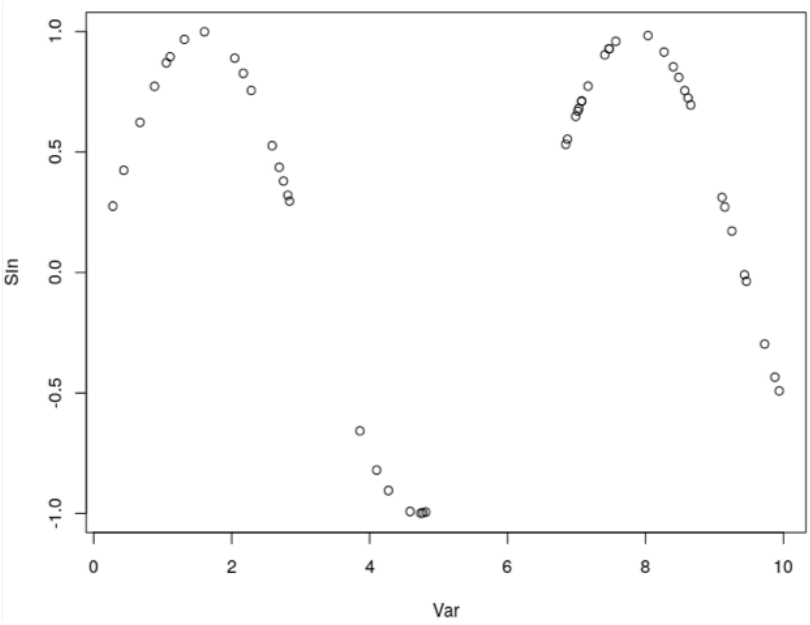
We have performed cross-validation in order to find the best smoothing factors. please find attached code below for your reference.

```
c_h_distance = c(10000,20000,50000,100000)
c_h_date = c(3,7,15,30)
c_h_time = c(2,7,11)
for (di in 1:length(c_h_distance)) {
  h_distance = c_h_distance[di]
  for (da in 1:length(c_h_date)) {
    h_date = c_h_date[da]
    for (ti in 1:length(c_h_time)) {
      h_time = c_h_time[ti]
      mse_val = 0
      #for all example in the val
      for(i in 1:n){
        temp_mult[i]  = mean_temp(to_pred[i,], st, i, "mult")
        #temp_mult[i]  = mean_temp(to_pred[i,], st, i, "sum")
        #compute the total mse
        mse_val = mse_val+(1/n)*((temp_mult[i]-to_pred[i,]$air_temperature)^2)
      }
      #print(mse_val)

      if (mse_val < best_mse_val) {
        best_mse_val = mse_val
        best_h_distance = c_h_distance[di]
        best_h_time = c_h_time[ti]
        best_h_date = c_h_date[da]
```

Assignment - 3

In this assignment, we wish to train a neural network to learn the trigonometric sine function. We generate uniformly 50 randoms points in the interval [0,10]. Given below is the graph of ***sine*** function applied on these 50 data points



We then split the data into 2 sets: 25 points for both the training and the validation set.The training set will be used to train the neural network to learn this sine function. The validation set will be used to detect when to stop the gradient descent in order to avoid overfitting.We are going to stop the gradient descent when the error function is below a given threshold value. We calculate threshold by iterating 10 times i.e. i = 1,2,3,..,10 using the below expression,

$$\frac{\boldsymbol{i}}{\mathbf{1000}}$$

We will then choose the threshold value that minimized the error on the validation set.

our neural network consists of:

- 1 input neuron
- 1 hidden layer with 10 neuron units
- 1 output neuron

We will randomly initialize all the weights of the neural network in the interval [-1,1]. The total weight is calculated is by,

- 1 bias * 10 neurons (hidden layer)
- 1 intercept * 10 neurons (hidden layer)
- 1 bias * 1 neuron (output neuron)

- 10 intercept* 1 neuron (from hidden layer to output neuron)

We create the neural network by using the ***neuralnet*** function,

```
nn = neuralnet(Sin~Var,data=tr,threshold = i/1000,hidden = 10,startweights = winit)
```

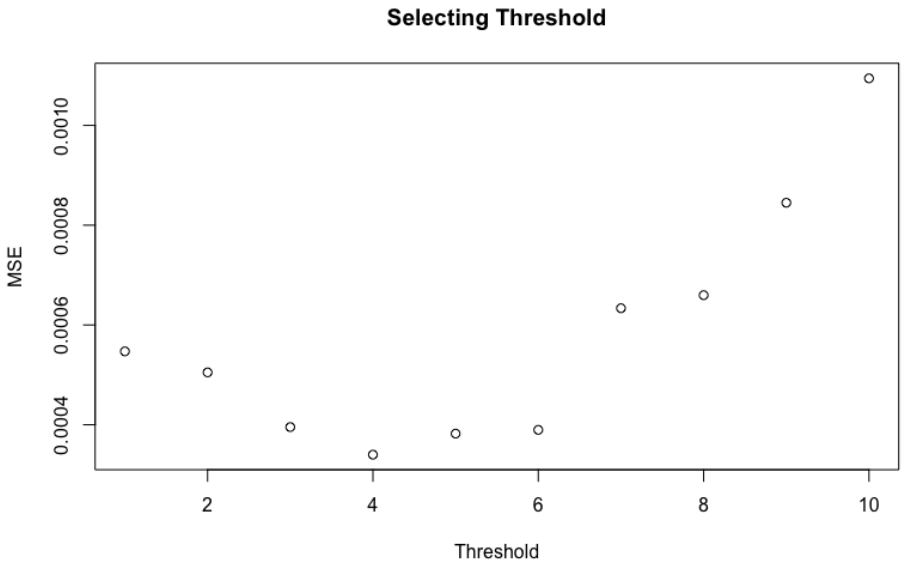
where,

- "Sin Var" : we try to predict the values in the Sin column from Var column values
- "data=tr": we use tr as a training set for the NN to learn the sine function
- "hidden=10": we have one hidden layer consisting of 10 hidden neurons units
- "startweights=winit": we use the values we randomly generated to initialize the weights

We predict over the validation set and calculte the MSE for each value of the threshold. The threshold with the least square error is chosen as the best threshold and best neural network.

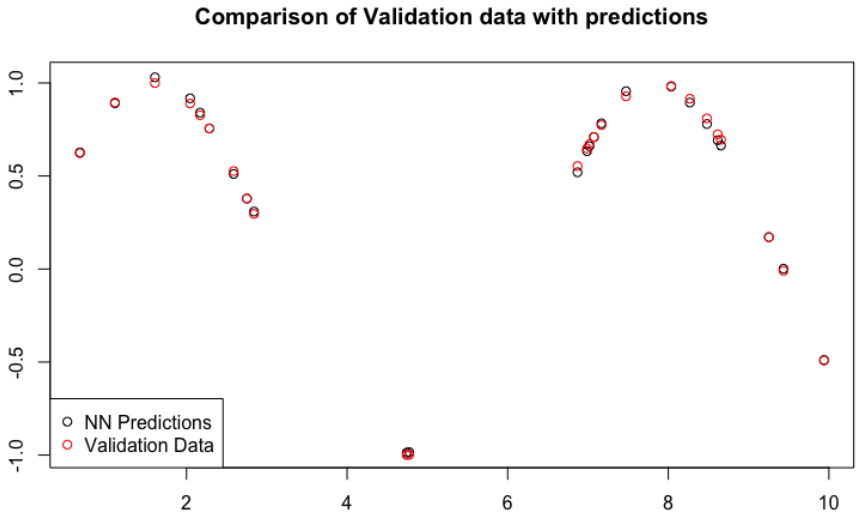
```
mse_test[i] = sum((va$Sin-pred)^2)/nrow(va) #va = validation_data
best_Threshold = which(mse_test == min(mse_test))
```

We can also select best threshold from plotting as well.After computing the error for all threshold values, we can plot the MSE Value of the validation data, which is given below.



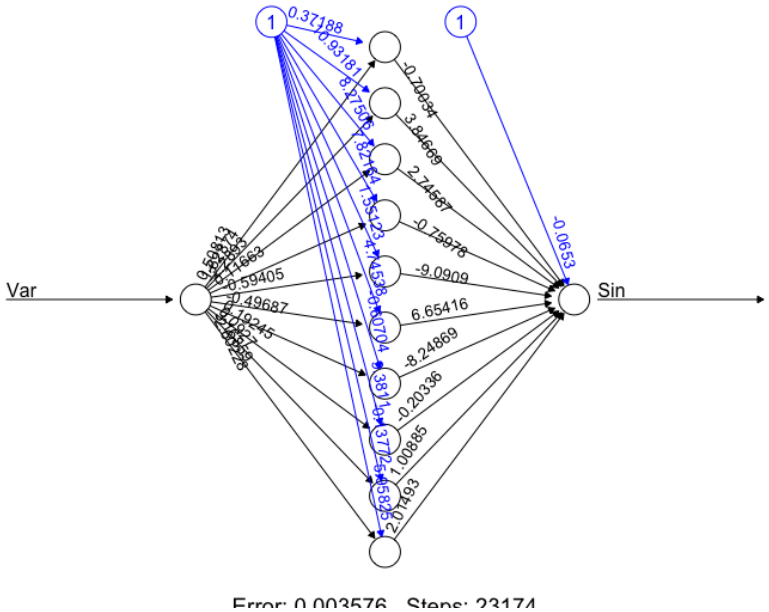
By comparing the graph as well as value from the variable ***best_threshold*** we chose the $threshold = 0.004$ where $i = 4$. Our minimum MSE value is **0.0003400358**

We plotted both the real values and predicted values which is given below,



The above plot is confirming what we see from the MSE value. The predictions are almost perfectly matching with the real values of the sine function

In order to visualize the network topology with weights and biases, we are plotting the nerual network and it is given below



Code Appendix

Assignment 1

```
options(scipen=999) #To avoid scientific notations
RNGversion('3.5.1')

set.seed(1234567890)
library(geosphere)

get_gaussian_kernel = function(a_smoothenedDistances) {
  return(exp(-a_smoothenedDistances^2))
}

get_temp_prediction = function(a_longitude, a_latitude, a_date, a_kernelType) {

  \#Calculating geo kernel
  distanceDifference = distHaversine(c(a_longitude, a_latitude), st[4:5])/1000 #Calc in km
  geoDistance = get_gaussian_kernel((distanceDifference) / h_distance)
  plot(distanceDifference, geoDistance, main = sprintf("%f, %f",a,b), ylab = "Weight", xlab = "Distance in
      km")

  \#Calculating date kernel
  dateDifference = abs(as.numeric(difftime(st$date, a_date, units = "days")))%%365
  dateDifference[dateDifference>182] = 365 - dateDifference[dateDifference>182]
  dateDistance = get_gaussian_kernel((dateDifference)/ h_date)
  plot(dateDifference, dateDistance, main = prediction_date, ylab = "Weight", xlab = "Distance in days")

  \#Calculating time kernel
  temp_predictions = c()
  for(i in 1:length(times)) {
    timeDifference = abs(as.numeric(difftime(strptime(st$time, "%H:%M:%S"), strptime(times[i],
        "%H:%M:%S")), units = "hours"))
    timeDifference[timeDifference > 12] = 24 - timeDifference[timeDifference > 12]
    timeDistance = get_gaussian_kernel(timeDifference / h_time)
    plot(timeDifference, timeDistance, main = times[i], ylab = "Weight", xlab = "Distance in hours")

    if(identical(a_kernelType, "sum")) {
      kernel = geoDistance + dateDistance + timeDistance
    } else {
      kernel = geoDistance * dateDistance * timeDistance
    }
    temp_predictions[i] = sum(kernel * st$air_temperature) / sum(kernel)
  }
  return(temp_predictions)
}

\#Read Data
stations <- read.csv("stations.csv", fileEncoding="latin1")
temps <- read.csv("temps50k.csv")

st <- merge(stations,temps,by="station_number")

h_distance <- 50 #km # These three values are up to the students
h_date <- 7 # days
h_time <- 3 # hours

a <- 58.4108 # The point to predict (up to the students)
```

```
b <- 15.6214
prediction_date <- "2019-12-19" # The date to predict (up to the students)
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

\#Filter out posterior dates
st = subset(st, as.Date(st$date) < as.Date(prediction_date))

\#Prediction using kernel sum
predictionKernelSum = get_temp_prediction(a, b, prediction_date, "sum")
print(predictionKernelSum)
plot(predictionKernelSum, type="o", main = "Kernel Sum",
     xlab = "Prediction point in temperature vector", ylab = "Temperature")

\#Prediction using kernel multiplication
predictionKernelMultiply = get_temp_prediction(a, b, prediction_date, "multiply")
print(predictionKernelMultiply)
plot(predictionKernelMultiply, type="o", main = "Kernel Multiplication",
     xlab = "Prediction point in temperature vector", ylab = "Temperature")
```

Assignment 3

```
options(scipen=999) #To avoid scientific notations
RNGversion('3.5.1')
library(neuralnet)
library(ggplot2)

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
ggplot(trva,aes(x=Var,y=Sin))+geom_line()
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31,-1,1)
#Since we are dealing with regression, best way to choose the model is to use MSE
mse_test = rep(0,10)
mse_train = rep(0,10)

for(i in 1:10)
{
  nn = neuralnet(Sin~Var,data=tr,linear.output = TRUE,threshold = i/1000,hidden = 10,startweights = winit)
  pred = predict(nn,va)
  mse_test[i] = sum((va$Sin-pred)^2)/nrow(va)
  mse_train[i] = sum((tr$Sin-pred)^2)/nrow(tr)
}

best_Threshold = which(mse_test == min(mse_test))
plot(1:10,mse_test,xlab="Threshold",ylab = "MSE",main="Selecting Threshold")
#1st model is having lowest test error when compared with other models.
nn = neuralnet(Sin~Var,data=tr,linear.output = TRUE,threshold = best_Threshold/1000,hidden =
    10,startweights = winit)
```

```
plot(prediction(nn)$rep1,col="black",xlab = "",ylab = "")
points(trva,col="red")
legend(x="top",legend = c("NN Predictions","Sin"),col=c("Black","Red"),pch = c(1,1))

plot(nn,rep="best")
```
