Introduction to Machine Learning

TDDE01 – Lab 3

Author:
Christoffer Eduards - chred146

I. Introduction

This report is part 3 of the laborations in the course TDDE01 – Introduction to Machine Learning. This report consists of two mandatory exercises. The code required for solving these assignments is attached to this report and can be viewed under Appendix.

II. Assignment 1

Kernel Methods
In the first task the goal is to implement a Kernel Method to predict the hourly temperatures for a date and place in Sweden. The exercise is based on data from the files "stations.csv" and "temps50k.csv" that are weather data provided by SMHI.

Method
The method that is used is to create two kernels, one is a sum of three Gaussian kernels and the other one is a product of the same three Gaussian kernels. The first of the three "sub-kernels", k_x, takes into account the distance from a weather station to a point of interest. The second one, k_y, account for the distance between the day a temperature measurement was made and the day of interest. This one does not account for the difference in years, i.e. it can never be more than 182,5 days. Lastly, the third one, k_z, account for the distance between the hour of the day a temperature measurement was made and the hour of interest. All the "sub-kernels" use a smoothing coefficient which is to be chosen.

Constants
Time points:
11 time points are used, from 04.00-00.00 with 2 hour intervals.

Dates
To be able to draw more accurate conclusions from the results two dates will be tested. These are **2013-01-01** and **2013-08-01.**

Smoothing coefficients
For the geographical distance the chosen smoothing coefficient is **10 Swedish miles**. The model should not give weight to data points that geographically are too far away from our POI to be interesting. Therefore 10 miles were chosen since the temperatures do not vary that much within that span.
For the difference in dates **15 days** were chosen. If this coefficient is too large, data from periods that are not representable for the date of interest will influence the result too much. If it is too small on the other hand, it will be over-smoothened and the fluctuations between similar dates from different years will not be considered.
For the difference in time of day, **2.5 hours** were chosen. This value was simply chosen by how the temperature fluctuates during the day. For example, for a larger value there can be a significant difference in temperature. For example, imagine 5 hours. What the temperature is at 08.00 is not that representative for what the temperature is at 13.00 that day. On the other hand, a smaller value will over-smoothen and too much weight will be put on a very short time span.

Math

To create the "sub-kernels" k_x, k_y and k_z, the following expression for a Gaussian kernel has been used:

$$k(u) = e^{-||\frac{u}{h}||^2}$$

The parameters are u: geographical distance/days difference/time of day difference which are calculated with separate functions, and h: smoothing coefficients. To create the two main kernels the following two expressions are used:

$$t_{sum}(\boldsymbol{x}) = \frac{\sum_n (k_x + k_y + k_z) t_n}{\sum_n k(k_x + k_y + k_z)}$$

$$t_{prod}(\boldsymbol{x}) = \frac{\sum_n (k_x k_y k_z) t_n}{\sum_n k(k_x k_y k_z)}$$

where t_n are the temperature measurements from the data (excluding the posterior measurements).

Results

Figure 1 and 2 show the predictions for 2013-01-01 for the two different methods. According to SMHI´s historical data, a normal average temperature for January in Linköping is approximately -2 °C. This can be used to validate and analyse the results.

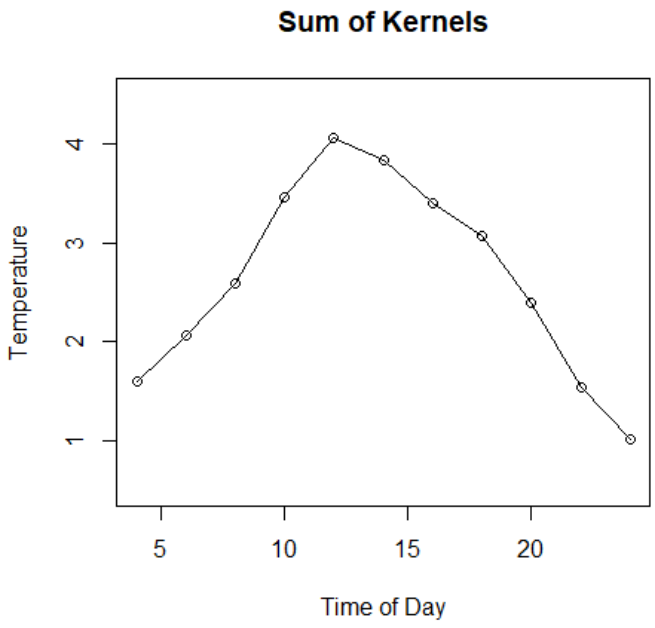

Figure 1: Predictions using sum for 2013-01-01

Using the sum of kernels, the predictions are clearly too high. Looking at the curve itself, the pattern seems to be correct. The highest temperature is predicted around 13.00 which is a clear peak on the curve.

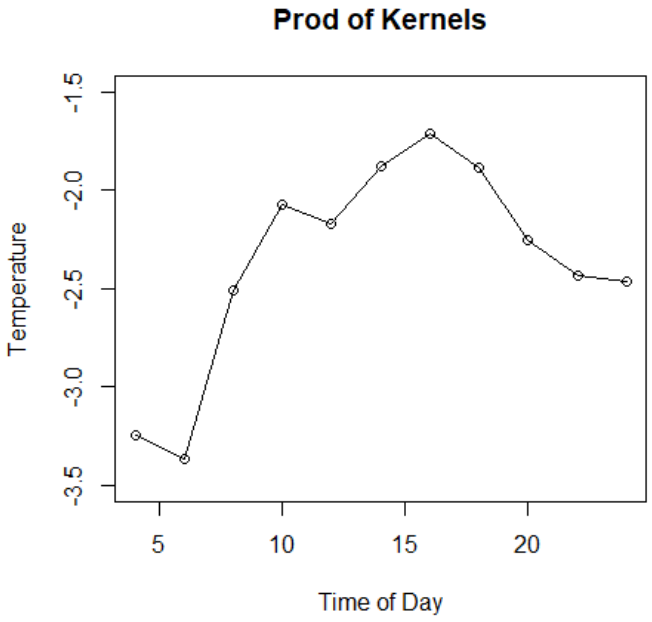

Figure 2: Predictions using product for 2013-01-01

With the product of the kernels, the temperature predictions are much more reasonable since they are close to the actual value. The curve still has a defined peak although the time of day for the warmest temperature may be a bit too late.

Figure 3 and 4 show the predictions for 2013-08-01 for the two different methods. According to SMHI´s historical data, a normal average temperature for August is approximately 16 °C.

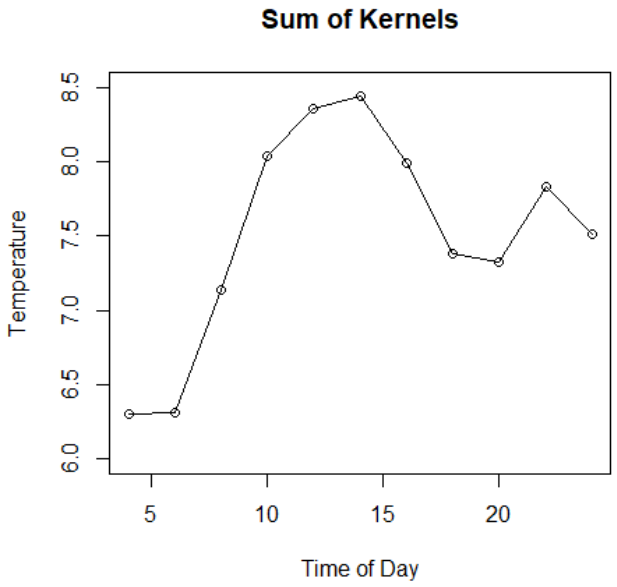

Figure 3: Predictions using sum for 2013-08-01

When using the sum of the kernels, the predictions are too low. If we compare with Figure 1, we can see that this way of constructing the kernels lower the variations during the year and smoothens out the temperature predictions.

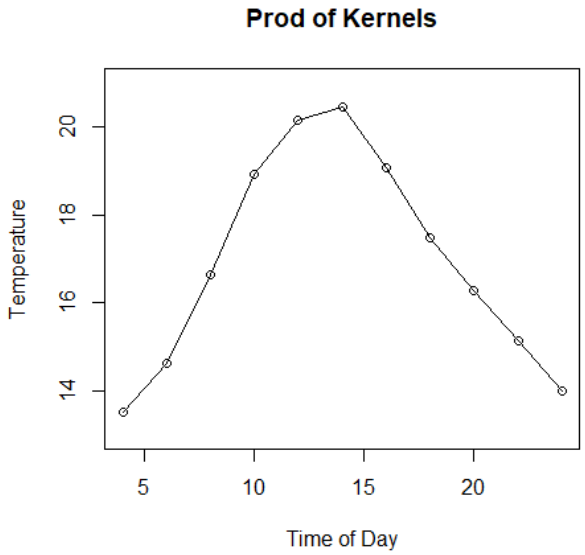

Figure 4: Predictions using product for 2013-08-01

Just as with Figure 2, this prediction is more reasonable. It corresponds well with the average temperatures for August.

The reason that the method where the sum is used smoothen outs the temperatures of the year can be explained by looking at the mathematical expressions. For the sum, the weight will have a value between 0 and 3 (each one between 0 and 1 individually). Therefore, two way different dates of the year can have similar temperatures if the time of day and location are similar. On the other hand, when the product is used, this will not happen. The weight factor will then have a value between 0 and 1 which will heavily limit the previous problem described with the summation kernel.

III. Assignment 3

Neural Networks

In this assignment the task was to train a neural network to learn the trigonometric sine function. The data used for this task was generated by sampling 50 points on the interval [0,10] and applying the sine function to these points. These points of x and sin(x) were then divided into training and validation data.

Next, a neural network was trained by using the package neuralnet in R. This NN consists of 1 layer of 10 hidden dimensions. The initial weights for the network were randomly generated in the interval [-1,1]. The optimal threshold was found by finding the lowest MSE for 10 different threshold values on the formula $\frac{i}{1000}, i \in 1,2, \dots ,10$.
The optimal threshold on the interval can be seen in Figure 5 below:

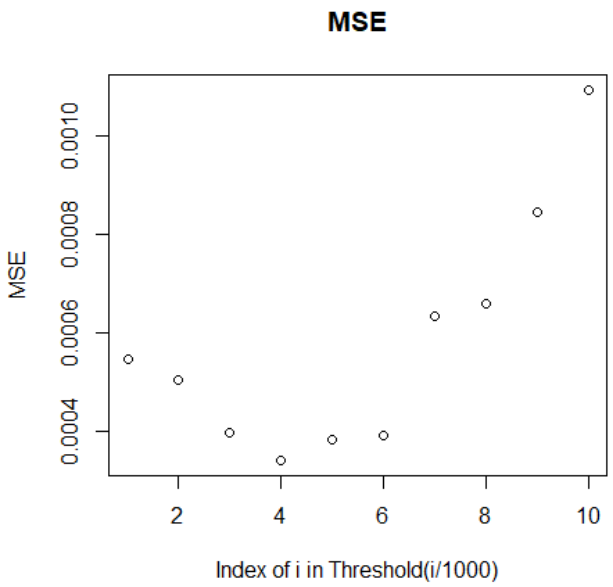

Figure 5: MSE for different i

From the figure we see that i = 4 is the value we shall chose for the threshold.

The optimal threshold could also have been found by plotting the different iterations vs the actual values. The problem with this, which is also shown by the different MSE, is that the different thresholds barely have an effect. This small difference can be seen by comparing i = 4 to i = 10 which is shown in Figure 6 and 7:

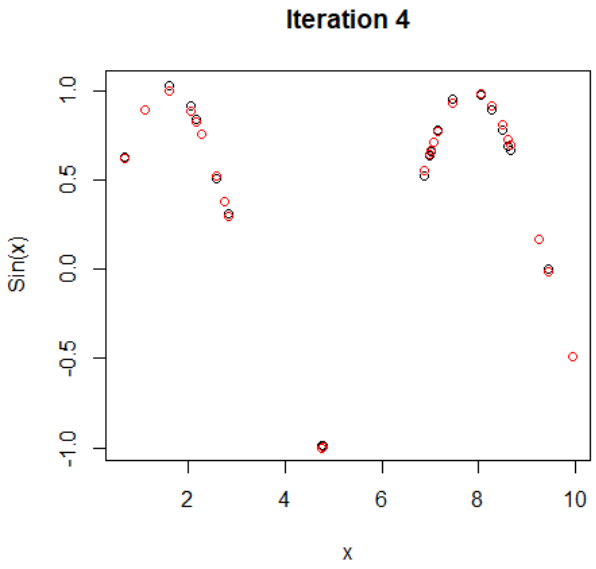

Figure 6: Predictions (blue) vs actual values (red) for i = 4

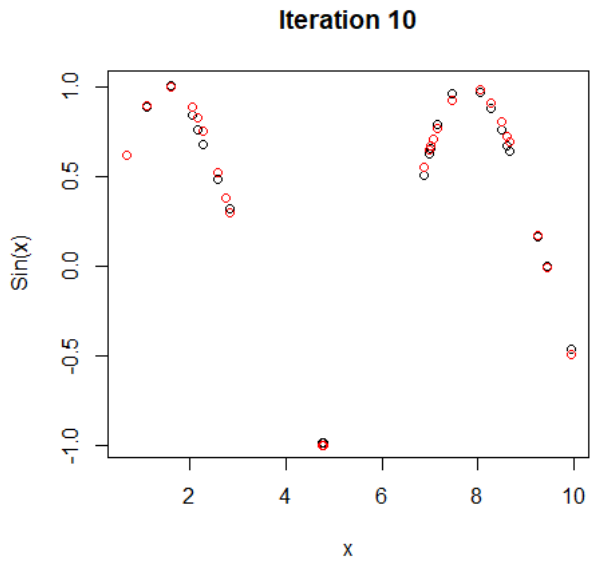

Figure 7: Predictions (blue) vs actual values (red) for i = 10

Lastly, the final network with the optimal threshold can be seen in Figure 8:

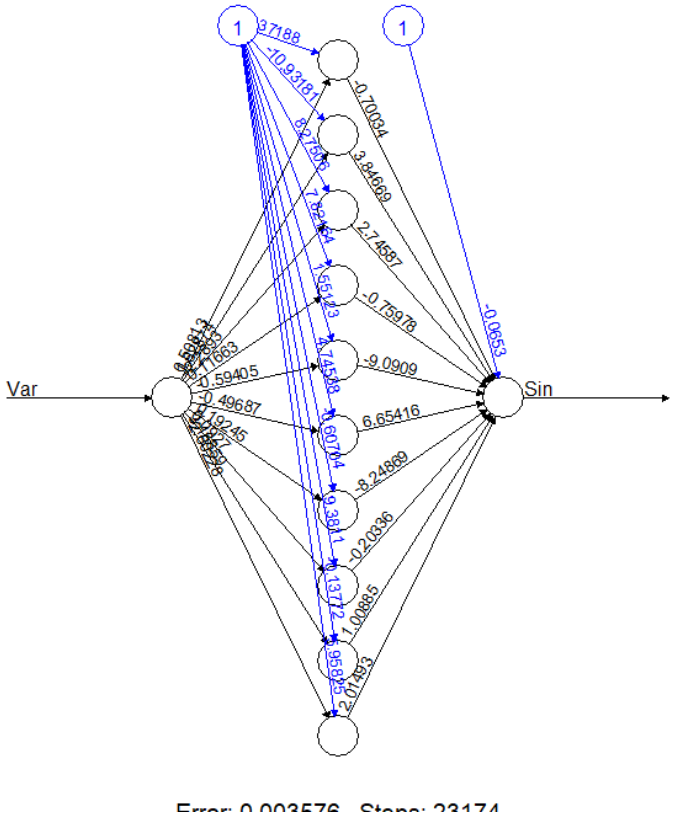

Figure 8: NN with optimal threshold

IV. Appendix

```
Assignment 1:
#### Import and divide data ####
setwd("C:/Users/Christoffer Eduards/OneDrive/R/Lab 3/Assignment 1")
set.seed(1234567890)
library(geosphere)
stations = read.csv("stations.csv")
temps = read.csv("temps50k.csv")
st = merge(stations, temps, by = "station_number")
#### Constants ####
h_distance = 100000 #Smoothing coefficient [meters]
h_date = 15 #Smoothing coefficient [days]
h_time = 2.5 #Smoothing coefficient [hours]
#POI (Linköping)
a = 58.401
b = 15.577
date = "2013-08-01" #The date to predict (one out of two used)
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",
    "00:00:00"
  ) #The times to predict
tempPredSum = vector(length = length(times)) #Used for predicted
temperatures with sum method
tempPredProd = vector(length = length(times)) #Used for predicted
temperatures with prod method
st = st[(difftime(st$date, date)) < 0, ] #Filter out posterior data
stPositions = cbind(c(st[, 4]), c(st[, 5])) #Get latitudes and longitudes
from data
##### Functions ####
#All methods implement this general Gaussian method
gaussianKernel = function(u, h) {
  return(exp(-abs(u / h) ^ 2))
}
#Measures the distance from the stations to a POI
stationToPoi = function(poi) {
  return(gaussianKernel(distHaversine(stPositions, poi), h_distance))
}
#Measures the days from the measurments to a Day of interest
dateToDoi = function(doi) {
  daysDiff = vector(length = length(st$date))
  mod = vector(length = length(st$date))
  u = vector(length = length(st$date))
  #Number of days difference
  for (i in 1:length(st$date)) {
    daysDiff[i] = as.numeric(as.Date(doi) - as.Date(st$date[i]))
  }
  #Modulus to get rid of the years
  for (i in 1:length(mod)) {
    mod[i] = daysDiff[i] %% 365
  }
```

```
  #Maximum half a year
  for (i in 1:length(daysDiff)) {
    u[i] = min(mod[i], 365 - mod[i])
  }
  return(gaussianKernel(u, h_date))
}
#Measures the time of day from the measurements to an hour of interest
hourToHoi = function(hoi) {
  hourDistance = as.numeric(abs(difftime(
    strptime(hoi, "%H:%M:%S"), strptime(st$time, "%H:%M:%S")
  ))) / 3600
  ifelse(hourDistance > 12, 24 - hourDistance, hourDistance) #Maximum 12
hours
  return(gaussianKernel(hourDistance, h_time))
}
#### Main ####
#### Temperature predictions ####
#Lastly all methods are utilized as described in the report and the
predictions are calculated using the three "sub-kernels"
stDist = stationToPoi(c(a, b))
daDist = dateToDoi(date)
for (i in 1:length(times)) {
  tiDist = hourToHoi(times[i])
  kernelSum = stDist + daDist + tiDist
  kernelProd = stDist * daDist * tiDist
  tempPredSum[i] = (kernelSum %*% st$air_temperature) / sum(kernelSum)
  tempPredProd[i] = (kernelProd %*% st$air_temperature) / sum(kernelProd)
}
#Plot the kernel utilizing a summation (boundaries for y-values need to be
changed for different dates)
plot(
  seq(4, 24, 2),
  tempPredSum,
  type = "o",
  ylim = c(6, 8.5),
  xlab = "Time of Day",
  ylab = "Temperature",
  main = "Sum of Kernels"
)
#Plot the kernel utilizing a product (boundaries for y-values need to be
changed for different dates)
plot(
  seq(4, 24, 2),
  tempPredProd,
  type = "o",
  ylim = c(13,21),
  xlab = "Time of Day",
  ylab = "Temperature",
  main = "Prod of Kernels"
)

Assignment 3:
#### Import and divide data ####
setwd("C:/Users/Christoffer Eduards/OneDrive/R/Lab2/Assignment 1")
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
#### Main ####
```

```
#Randomly generating the initial weights
winit = runif(31, min = -1, max = 1)
MSE = c() #Vector for the MSE
#Training nn different i
for (i in 1:10) {
  nn = neuralnet(
    Sin ~ Var,
    data = tr,
    hidden = 10,
    threshold = i / 1000,
    startweights = winit
  )
  #Prediction for validation data
  pred = compute(nn, va$Var)$net.result
  #Plotting iterations for nn training
  plot(
    va$Var,
    pred,
    main = paste("Iteration", i),
    ylab = "Sin(x)",
    xlab = "x"
  )
  #Plot actual data
  points(va, col = "red")
  #Mean squared error vs iteration
  MSE[i] = mean((va$Sin - pred) ^ 2) / nrow(va)
}
#Show mean squared error for different i
plot(MSE, xlab = "Index of i in Threshold(i/1000)", main = "MSE")
#Train optimal nn
nn = neuralnet(
  Sin ~ Var,
  data = tr,
  hidden = 10,
  threshold = which.min(MSE) / 1000,
  startweights = winit
)
plot(nn)#Picture of the final nn
```