Lab 3

Rolf Sievert (rolsi701)

2018-12-17

Contents

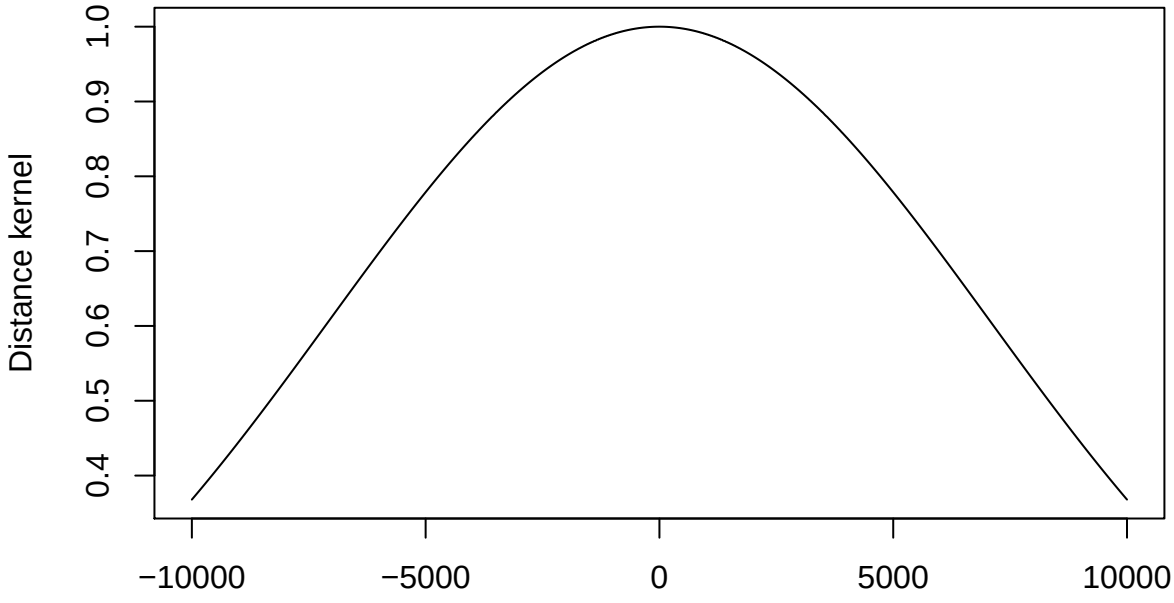
Assignment 1

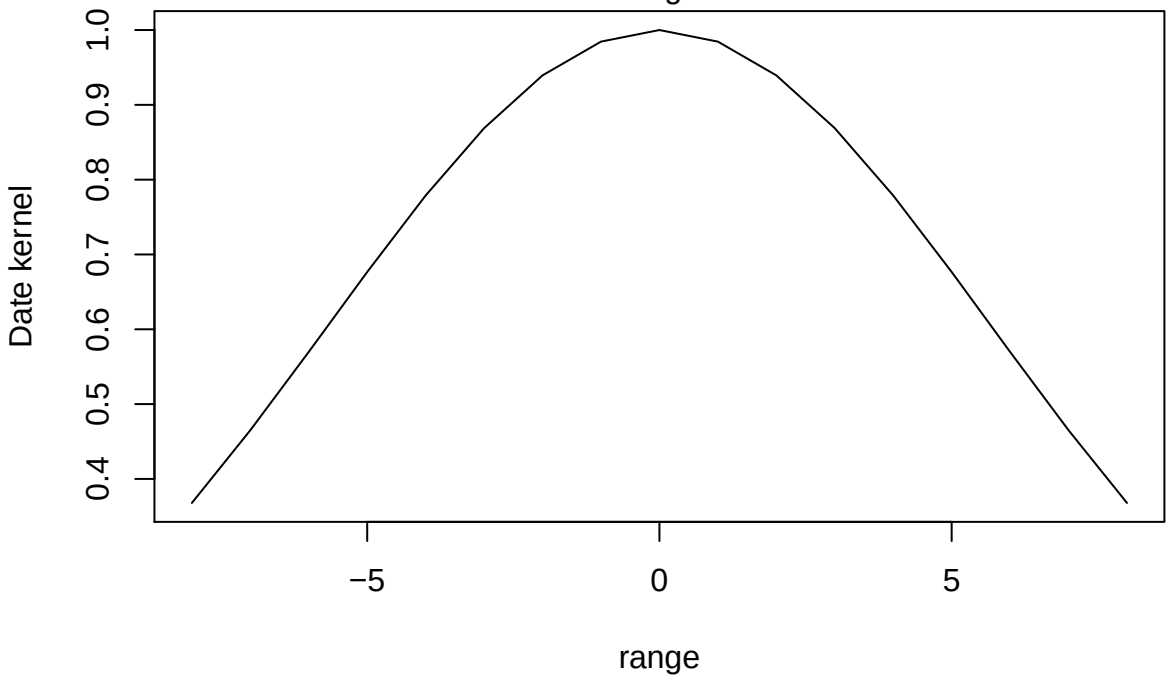
Part 1

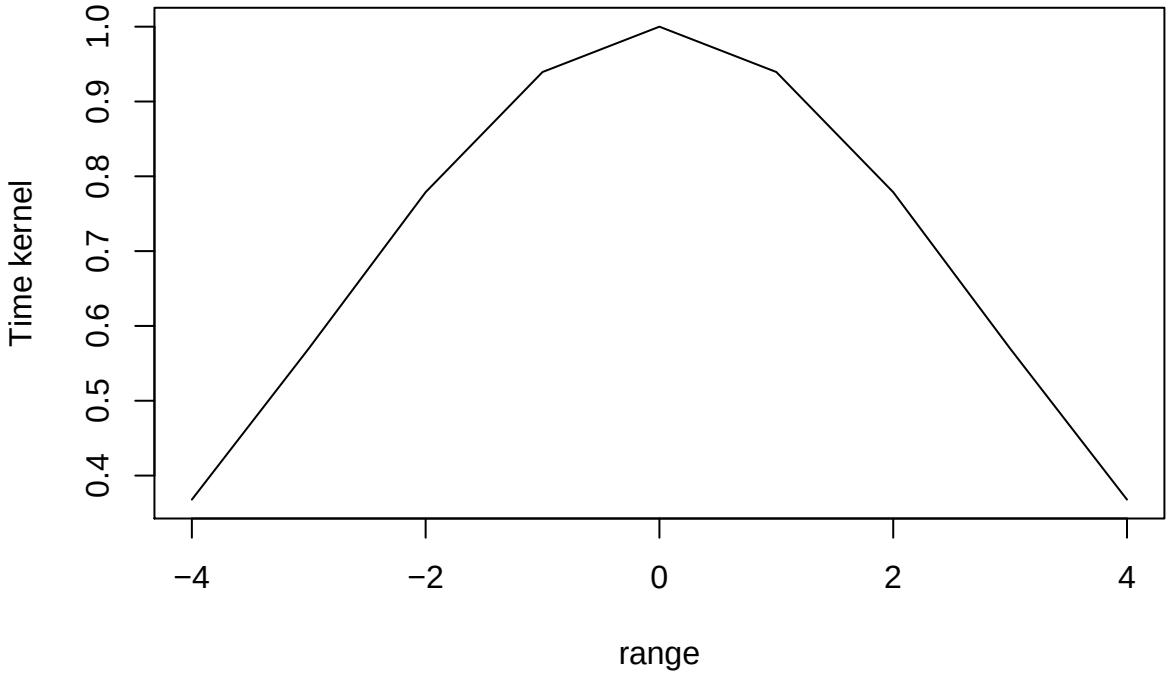
The chosen width for each kernel's is:

- h_distance: 10000 m
- h_date: 8 days
- h_time: 4 hours

The plots below show that the kernels gives more weight to closer points.

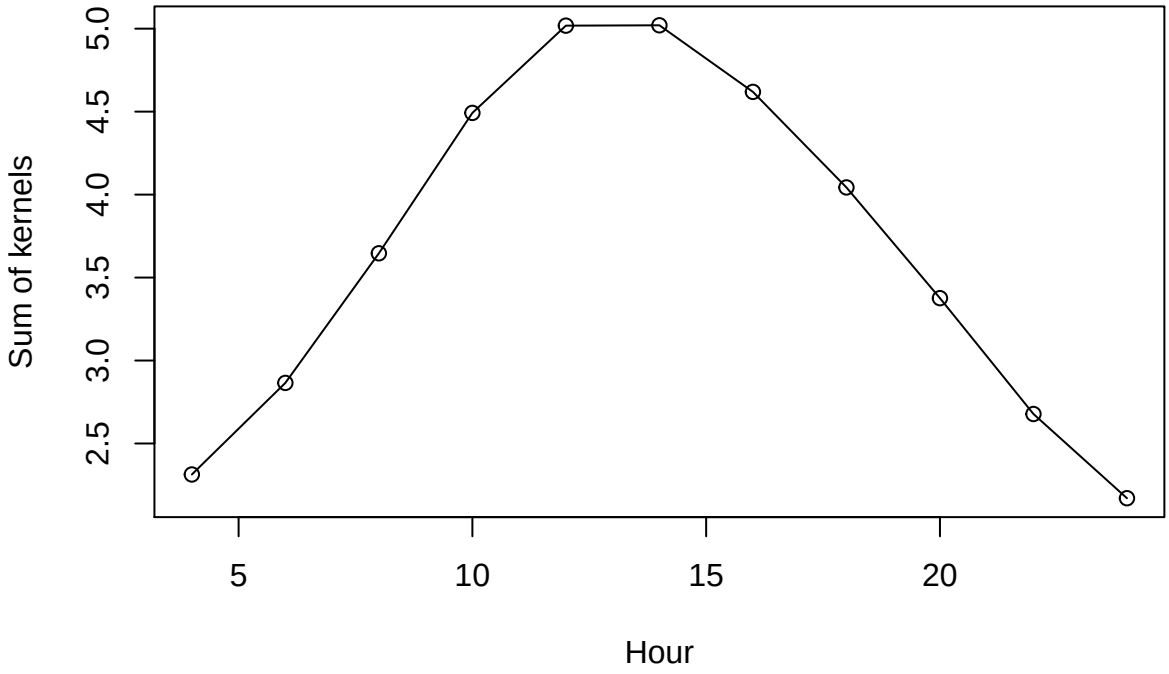

Distance kernel
0.4
0.5
0.6
0.7
0.8
0.9
1.0
−10000
−5000
0
5000
10000
range

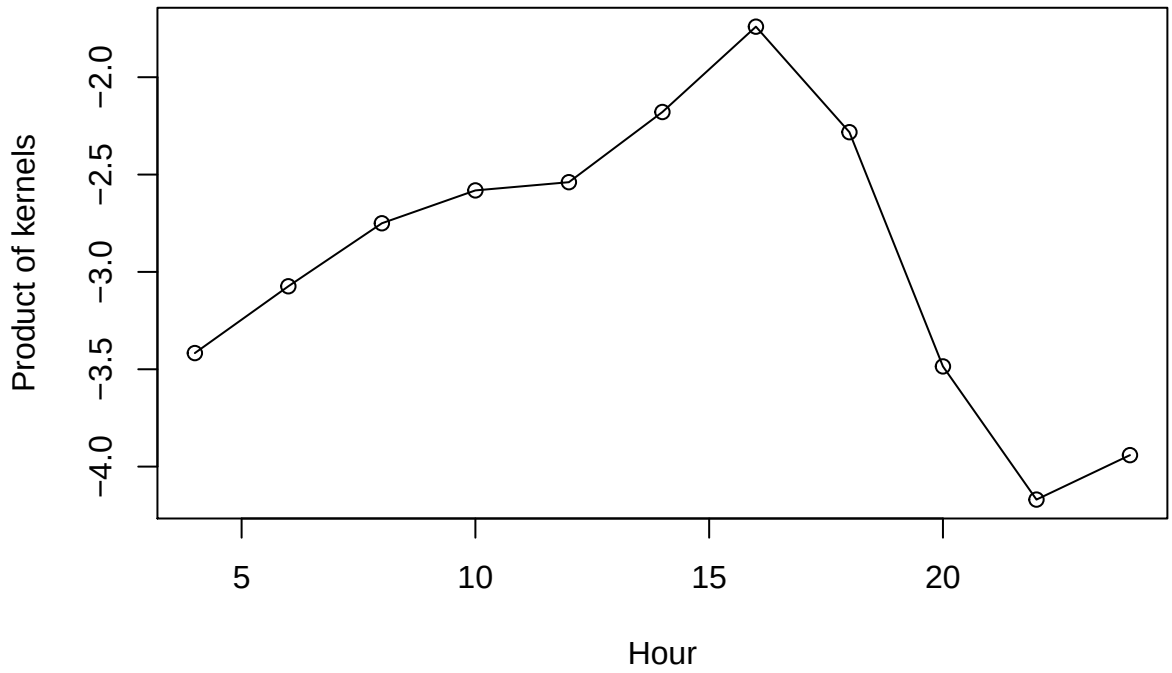

Date kernel
0.4
0.5
0.6
0.7
0.8
0.9
1.0
−5
0
5
range



Part 2

The plots below show a temperature prediction on Christmas Eve. The first one summarizing each kernel, the other multiplying them.



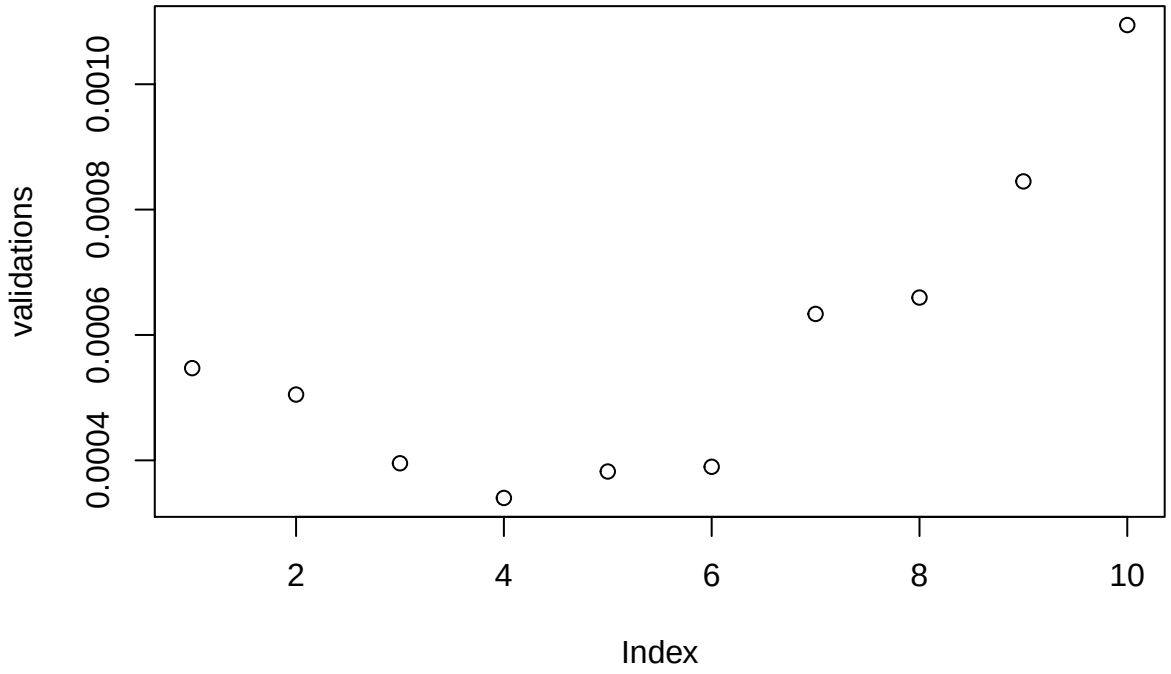


As can be seen, the product of kernels give preffered results (since it's Christmas). It's also more probable that it's minus degrees in end of December.

Assignment 3

Validation

The threshold was chosen to the one that ended up resulting in the lowest mean square error. The MSE for thresholds in the interval [1/1000, 10/1000] are plotted below.



As seen here, the best threshold in this case is 4/1000.

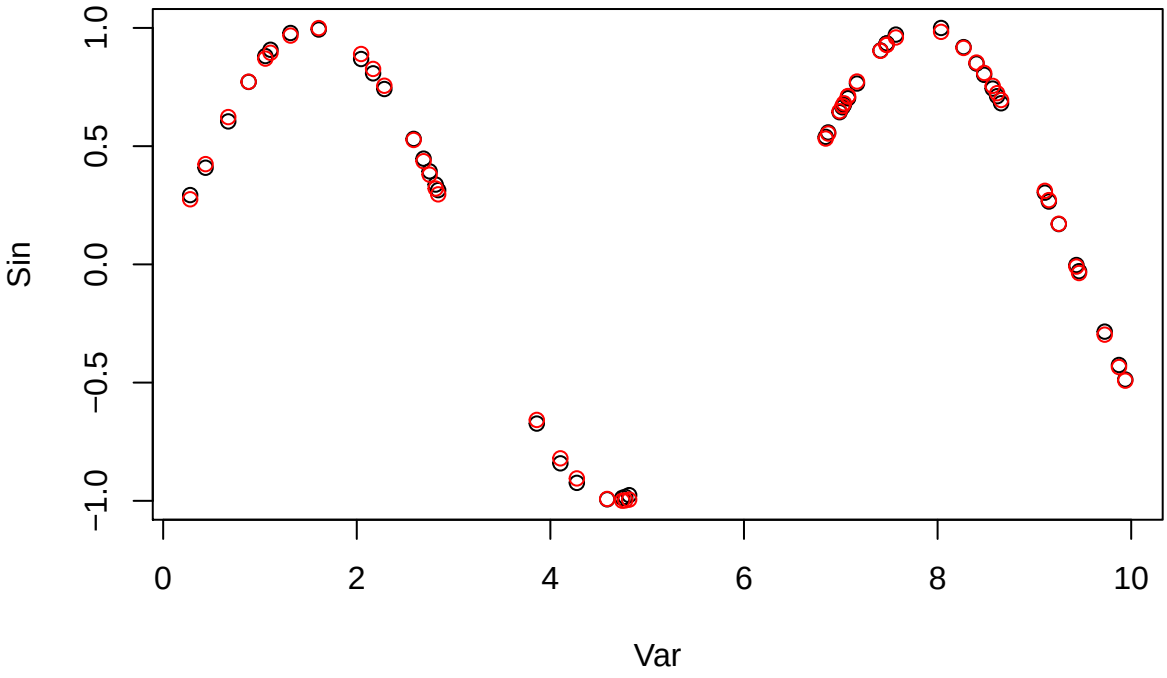
The final neural network

Test?

Prediction of Sin

Black dots are prediction, red ones are correct values.

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



Appendix

Assignment 1

```
library("geosphere")

setwd("~/courses/tdde01/lab3/")

stations = read.csv("stations.csv", fileEncoding = "Latin1")
temps = read.csv("temps50k.csv")
st = merge(stations, temps, by="station_number")

gauss_kernel = function(u) {
    return (exp(-(u^2)))
}

h_distance = 10000
```

```
h_date = 8
h_time = 4

# Station is station number, interest is vector(long, lat)
kernel_distance = function(station, interest) {
    distance = distHaversine(station, interest)
    res = gauss_kernel(distance/h_distance)
    return (res)
}

# Calculates difference in days from day to interest
kernel_date = function(dates, interest) {
    dates = as.Date(dates)
    interest = as.Date(interest)
    distance = as.numeric(difftime(dates, interest, units=c("days")))
    leap_years = floor(floor(distance/365)/4)
    distance = (distance - leap_years) %% 365
    distance = sapply(distance, function(d) min(d, abs(365 - d)))
    res = gauss_kernel(distance/h_date)
    return (res)
}

kernel_time = function(hour, interest) {
    h1 = as.numeric(substr(hour, 1, 2))
    h2 = as.numeric(substr(interest, 1, 2))
    distance = abs(((h2-h1) + 12) %% 24 - 12)
    return (gauss_kernel(distance/h_time))
}

# Plot kernels
range = -h_distance:h_distance
plot(x=range, y=gauss_kernel(range/h_distance), type="l", ylab = "Distance kernel")

range = -h_date:h_date
plot(x=range, y=gauss_kernel(range/h_date), type="l", ylab = "Date kernel")

range = -h_time:h_time
plot(x=range, y=gauss_kernel(range/h_time), type="l", ylab = "Time kernel")

kernel_sum = function(long, lat, date, time, product=FALSE) {
    longlat = data.frame(st$longitude, st$latitude)
    distances = kernel_distance(longlat, c(long, lat))

    datediffs = kernel_date(st$date, date)

    timediffs = kernel_time(st$time, time)

    sum = sum((distances + datediffs + timediffs) * st$air_temperature)
    sum = sum / sum(distances + datediffs + timediffs)

    prod = sum((distances * datediffs * timediffs) * st$air_temperature)
    prod = prod / sum(distances * datediffs * timediffs)

    if (product) {
```

```
        return(prod)
    }
    else {
        return(sum)
    }
}

latitude = 58.4274 # The point to predict
longitude = 14.826
date = "2018-12-24" # 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",
          "00:00:00")
# Remove posterior dates
st = st[as.Date(st$date) < as.Date(date),]

temp = lapply(times, function(t) kernel_sum(longitude, latitude, date, t, FALSE))

plot(temp, x=seq(4, 24, 2), type="o", ylab="Sum of kernels", xlab="Hour")

temp = lapply(times, function(t) kernel_sum(longitude, latitude, date, t, TRUE))

plot(temp, x=seq(4, 24, 2), type="o", ylab="Product of kernels", xlab="Hour")
```

Assignment 3

```
library("neuralnet")
set.seed(1234567890)

# Random data from 0-10, uniformly distributed
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)
validations = c()
for (i in 1:10) {
    nn = neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, startweights = winit, threshold = i/100
    comp = compute(nn, va$Var)

    mse = sum((comp$net.result - va$Sin)^2) / nrow(va)
    validations = c(validations, mse)
}
plot(validations)
best_threshold = which.min(validations)/1000

# Optimal nn
nn = neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit, threshold = best_th

plot(nn, ylab="Neural network")
```

```
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1, ylim = c(-1, 1))
points(trva, col = "red")
```