This report is written by Group B6:

Erik Andersson

Oscar Hubertsson

Isak Jansson

The answers in this report were discussed in the group and then written down in this document through the efforts of all members. Code in appendix of assignment 1 is then taken from Oscar, while assignment 3 is taken from Isak.

Assignment 1. Kernel Methods

In this assignment the task is to predict the temperature for a given point in Sweden on a given date. The prediction is to be given bi-hourly, for times: 04:00 through 24:00. As the base for prediction, data from SMHI was provided.

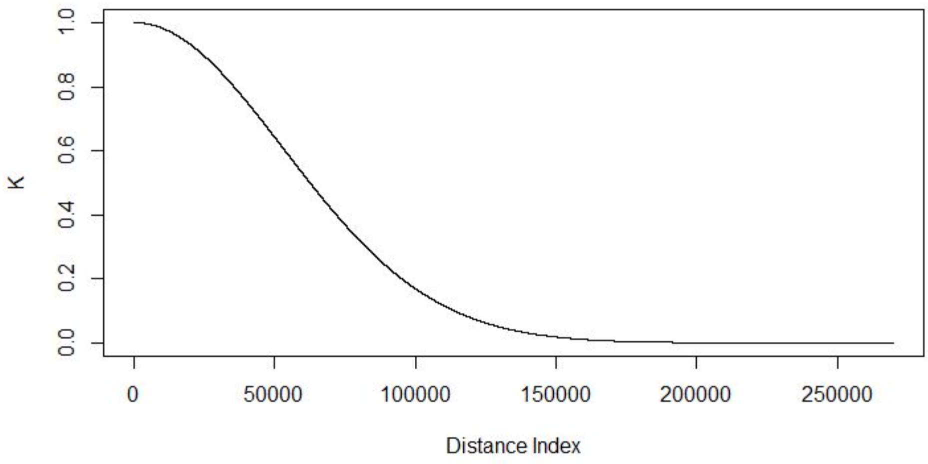
A Gaussian kernel, which is the sum of three kernels, was used. The Gaussian kernel consists of one kernel for distance from measurement station to point of interest, one for difference in time of day between measurement and prediction time and one for the difference in days.

Deciding the width

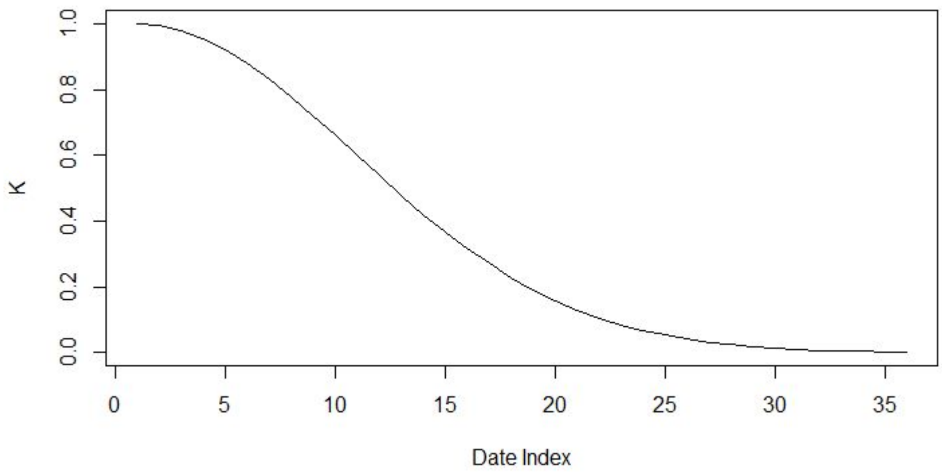
We decided for the following values:

- *Distance* - width 75000, which is equivalent to 75 km.
- *Date* - width 14, which is equivalent to 14 days.
- *Time* - width 4, which is equivalent to 4 hours.

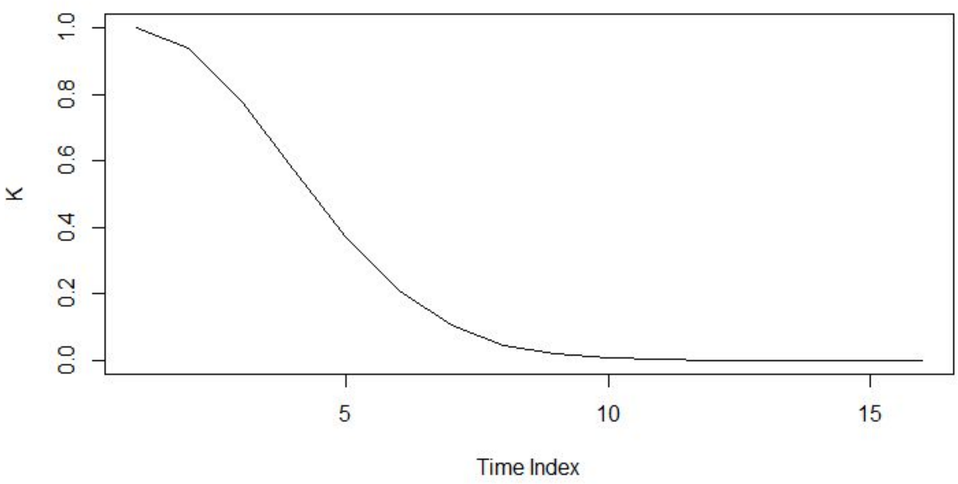
From analysing the graphs on the following page for different widths, we decided that these would be best values. With this we only take into consideration measurements within 75 km of our point of interest, taken within a range of 14 days from the desired date and within 4 hours of the time of interest.



1.0
0.8
0.6
0.4
0.2
0.0
K
0
50000
100000
150000
200000
250000
Distance Index



1.0
0.8
0.6
0.4
0.2
0.0
K
0
5
10
15
20
25
30
35
Date Index

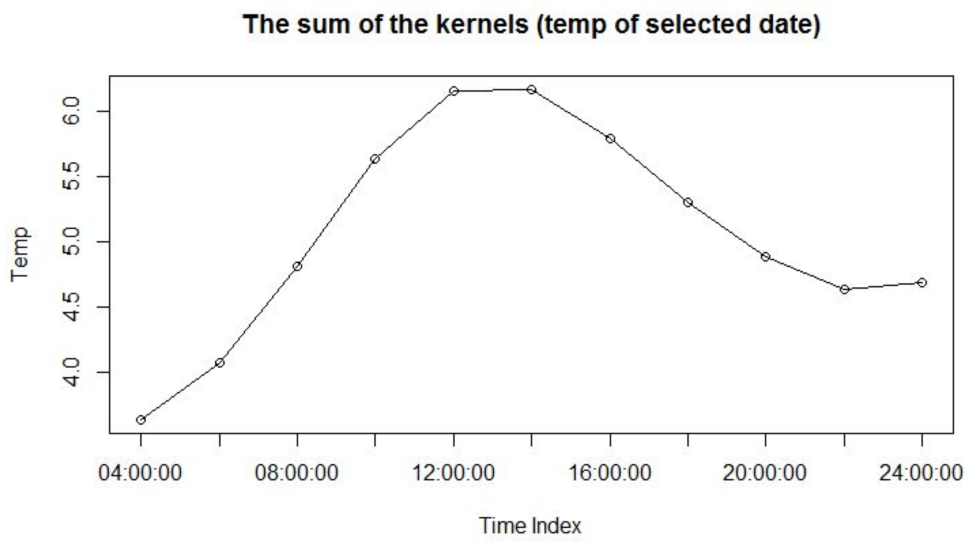


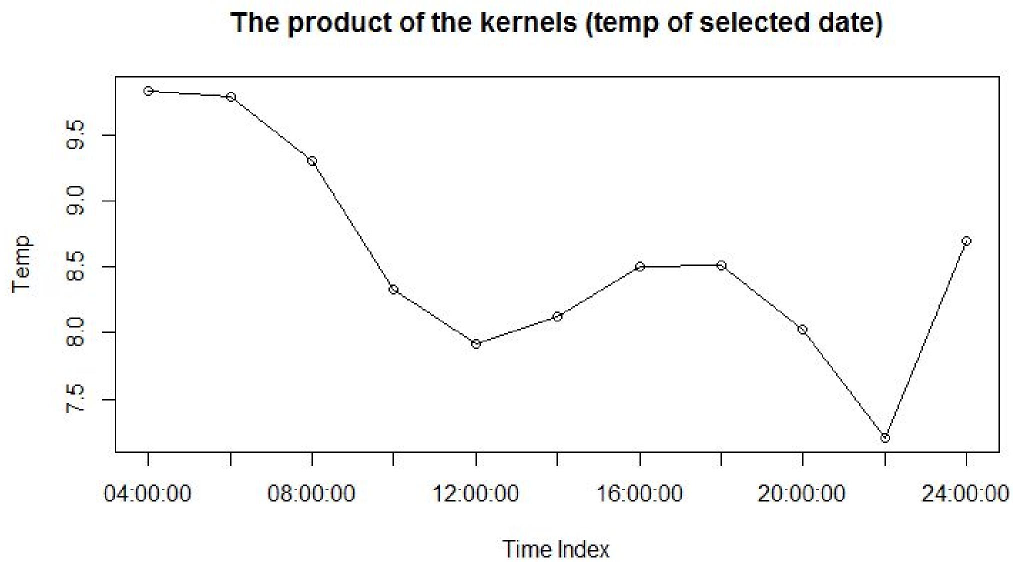
1.0
0.8
0.6
0.4
0.2
0.0
K
5
10
15
Time Index

Results

We then used the kernels to help predict the temperatures of a specific date 2013-11-04. These predictions can bee seen in the plots below where we either sum or multiply the kernels with each other to produce the different plots.

Adding the independent kernels provided a worse estimate than when they were multiplied. This is because when multiplying the kernels their weights become dependent on each other. Strong values enforce each other, giving a stronger result, while smaller values in one kernel weaken the result from the others.

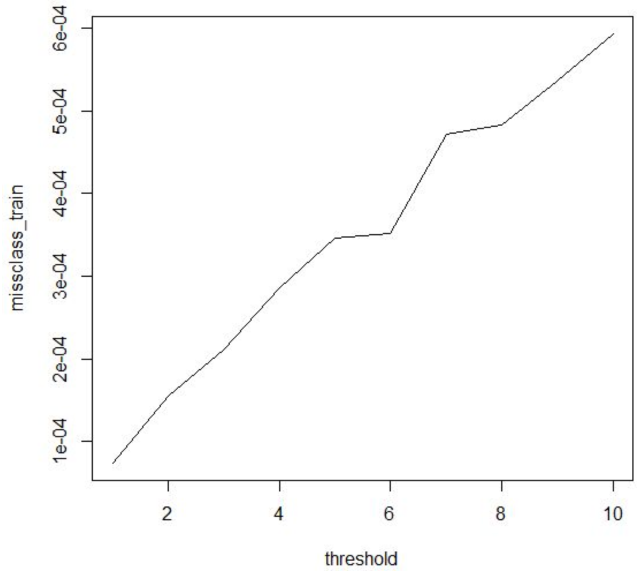


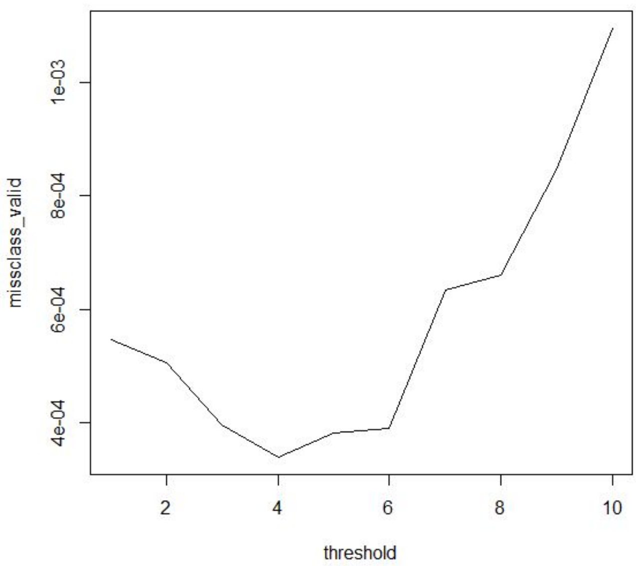


Assignment 3 Neural Networks

In this assignment we were tasked with training a neural network (NN) to learn the Sin function. Using 25 random random values for training and 25 for validation. A different NN is to be created for thresholds 1/1000 to 10/1000.

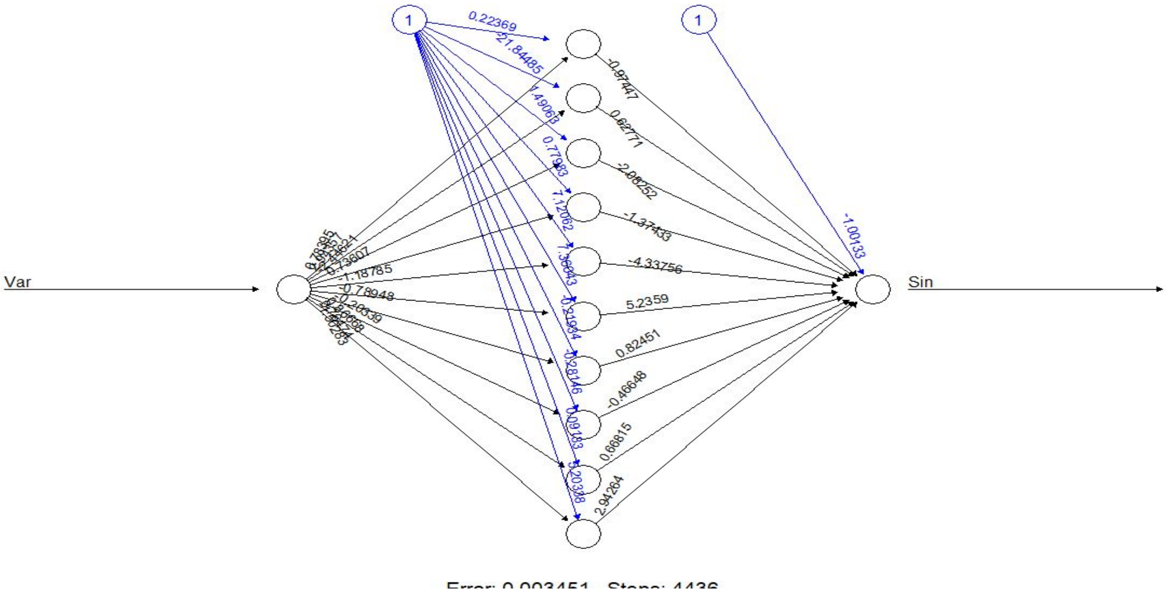
We used MSE to examine which would be the best threshold value. We chose the threshold 4 / 1000 because it gave the smallest MSE for the validation data. This was done to make sure the model is not overfitted.





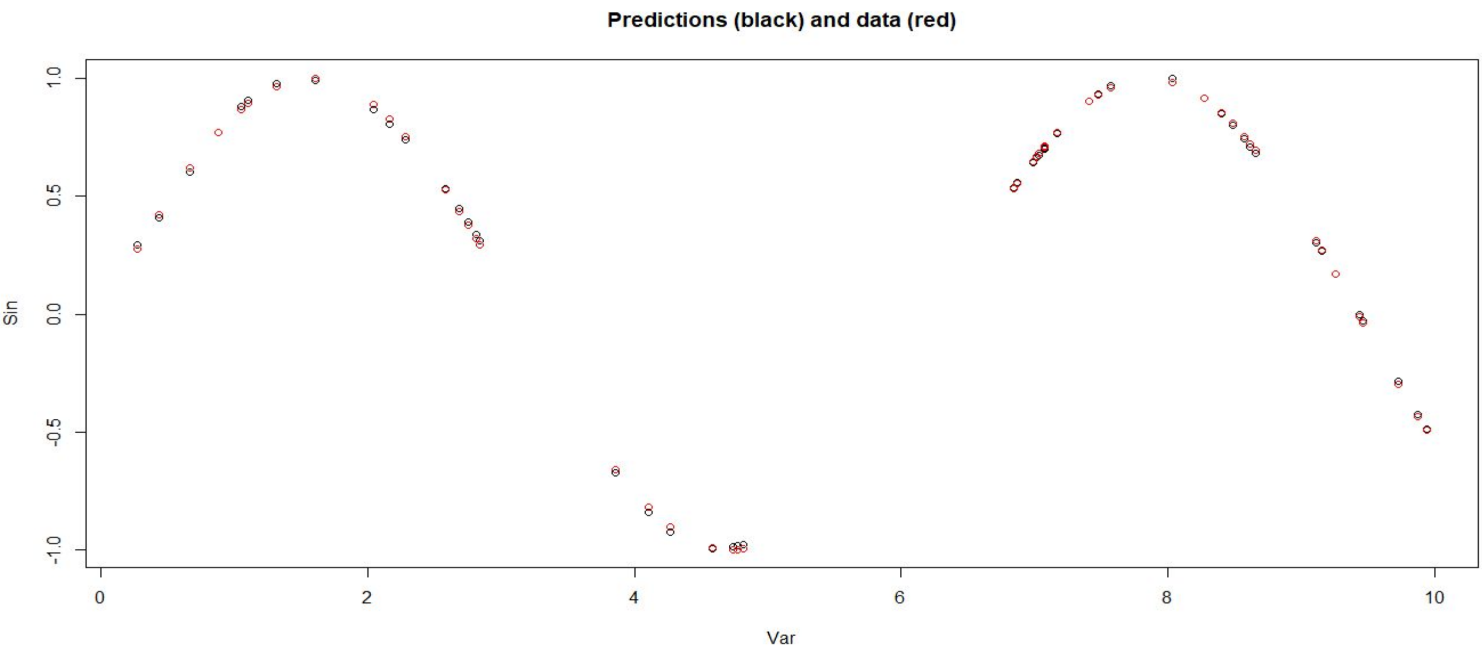
Weights

We were tasked with choosing the appropriate weights (starting weights). We did this by counting the connections/synaptic constants from the plot below of the neural network. We then created a vector of 31 values between -1 and 1.



Result

Below we can see the plot of the result of the sin function that we created with the neural net as compared to the data.



Appendix - Code

Assignment 1

```
#Should reset variables at beginning of run (eases tracability and rerun when solving the problems
with code)

rm(list=ls())

#Remove all plots(eases tracability and rerun when solving the problems with code)

dev.off()

RNGversion("3.5.1")

#-----------------------------------------------------------------------------------------------------------------------

#Set up dataset

set.seed(1234567890)

library(geosphere)

stations <- read.csv("stations.csv", header = TRUE, fileEncoding = "ISO-8859-1")

temps <- read.csv("temps50k.csv", header = TRUE, fileEncoding = "ISO-8859-1")

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

#-----------------------------------------------------------------------------------------------------------------------
```

```r
#Fillter out measurements posterior to the day and hour of my forcast (labinstruction)


# Posterior date filter (data posterior to our target date)

filterdate = function(dataset, date) {

  filtered_data = dataset[!as.Date(dataset$date) >= as.Date(date),]

  return (filtered_data)

}


#-----------------------------------------------------------------------------------------------------------------------


# Create the different parameters

# These three values are up to the students

      h_distance <- 75000

      h_date <- 14

      h_time <- 4

      a <- 58.4274 # The point to predict (up to the students)

      b <- 14.826


position <- c(b, a) # Using given point from lab


date <- "2013-11-04" # 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))


#-------------------------------------------------------------------------------------------------------------------


#Find decent h-values (smoothing coefficient aka width)


h_plot_k_against_ddt = function(sequence, h_value, tag) {

  u <- sequence/h_value

  k_of_u <- exp(-u^2)

  plot(k_of_u, type="l", xlab = tag, ylab = "K")

}


#To find decent h values plot K against distance/date/time

#Try different sequences to see which seems most suitable

distance_sequence <- seq(0,270000, 1)

date_sequence <- seq(0,35,1)

time_sequence <- seq(0,15,1)


#Plots of K against distance/date/time

h_plot_k_against_ddt(distance_sequence, h_distance, "Distance Index")
```

```
h_plot_k_against_ddt(date_sequence,h_date, "Date Index")

h_plot_k_against_ddt(time_sequence,h_time, "Time Index")


#----------------------------------------------------------------------------------------------------------------------


# Create the functions necessary for Gaussian Kernels

# Gaussian Kernels and sum of three kernels

# Lecture 3A: Gaussian kernel: k(u)=exp(−||u||^2)


#Instruction Part 1 of kernel

# Calculate the gaussian kernel for distance between point of interest and station

calculate_gaussian_kernel_for_distance = function(n_longlat_of_points, longlat_of_point, h_value) {

  u = distHaversine(data.frame(n_longlat_of_points$longitude, n_longlat_of_points$latitude), longlat_of_point)/h_value

  k_of_u = exp(-u^2)

  return(k_of_u)

}


#Instruction Part 2 of kernel

# Calculate the gaussian kernel for distance between day of interest and day

calculate_gaussian_kernel_for_day = function(n_temp_of_dates, temp_of_a_date, h_value) {

  u  =  (as.numeric(as.Date(n_temp_of_dates$date) - as.Date(temp_of_a_date), unit="days"))/h_value
```

```
  k_of_u  =  exp(-u^2)

  return(k_of_u)

}



#Instruction Part 3 of kernel

# Calculate the gaussian kernel for distance between hour of interest and hour

calculate_gaussian_kernel_for_hour = function(n_time_of_day, time_of_day, h_value) {



  difference_in_time = difftime(strptime(n_time_of_day$time , format = "%H:%M:%S"),
strptime(time_of_day , format = "%H:%M:%S"))



  difference_in_time = as.numeric(difference_in_time/(3600))



  u = difference_in_time/h_value

  k_of_u = exp(-u^2)

  return(k_of_u)

}



#Instruction Part 4 of kernel (SUM)

# Calculate the gaussian kernel for distance between day of interest and day, sum of kernels



estimate_temperature_sum = function (stations, date_input, time_input, pos, h_value_date,
h_value_distance, h_value_hour) {
```

```
}

  data_filtered_by_date = filterdate(stations, date_input)

  k_of_u_distance = calculate_gaussian_kernel_for_distance(data_filtered_by_date, pos,
h_value_distance)

  k_of_u_day =calculate_gaussian_kernel_for_day(data_filtered_by_date,date_input, h_value_date)

  temperature = vector(length = length(time_input))

  estimated_temperature_sum = vector(length = length(time_input))

  for (i in 1:length(temperature)){

        k_of_u_hour = calculate_gaussian_kernel_for_hour(data_filtered_by_date, time_input[i],
h_value_hour)

        total_sum_of_k = k_of_u_day + k_of_u_distance + k_of_u_hour

        estimated_temperature_sum[i] = sum(total_sum_of_k %*%
data_filtered_by_date$air_temperature)/sum(total_sum_of_k)

  }

  return(estimated_temperature_sum)

}

# Calculate the gaussian kernel for distance between day of interest and day, multiplication of kernels
```

```
estimate_temperature_multiyply = function (stations, date_input, time_input, pos, h_value_date,
h_value_distance, h_value_hour) {

  data_filtered_by_date = filterdate(stations, date_input)



  k_of_u_distance = calculate_gaussian_kernel_for_distance(data_filtered_by_date, pos,
h_value_distance)

  k_of_u_day =calculate_gaussian_kernel_for_day(data_filtered_by_date,date_input, h_value_date)



  temperature = vector(length = length(time_input))

  estimated_temperature_multiplication = vector(length = length(time_input))




  for (i in 1:length(temperature)){



        k_of_u_hour = calculate_gaussian_kernel_for_hour(data_filtered_by_date, time_input[i],
h_value_hour)

        total_multiplication_of_k = k_of_u_day * k_of_u_distance * k_of_u_hour

        estimated_temperature_multiplication[i] = sum(total_multiplication_of_k %*%
data_filtered_by_date$air_temperature)/sum(total_multiplication_of_k)



  }



  return(estimated_temperature_multiplication)
```

```r
}

#------------------------------------------------------------------------------------------------------------------------

# Returns vectors of temperature estimates

temperature_with_sum = estimate_temperature_sum(st, date, times, position, h_date, h_distance, h_time)

temperature_with_multiplication = estimate_temperature_multiyply(st, date, times, position, h_date, h_distance, h_time)

# Time to plot the results

plot(temperature_with_sum, type="o", xlab="Time Index", ylab="Temp", xaxt = "n", main = "The sum of the kernels (temp of selected date)")

axis(1, at=1:length(times), labels=times)

plot(temperature_with_multiplication, type="o", xlab="Time Index", ylab="Temp", xaxt = "n", main = "The product of the kernels (temp of selected date)")

axis(1, at=1:length(times), labels=times)
```

Assignment 3

```
dev.off()
rm(list=ls())
RNGversion('3.5.1')
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

#missclassification function to meassure which is best
missclassfunc=function(X,X1){
  n = length(X)
  diff_total=0
  for (i in 1:n) {
          diff_total = diff_total + (X[i]-X1[i])^2
  }
  diff_total = diff_total/n
  return (diff_total)
}

# Random initialization of the weights in the interval [-1, 1]
winit = runif(31, -1, 1)
threshold = seq(1,10)
missclass_train = vector("numeric", length=0)
missclass_valid = vector("numeric", length=0)

# Set up NN for all thresholds and calculate MSE for them
  for(i in seq(1, length(threshold))) {
          nn <- neuralnet(Sin ~ Var, data= tr, hidden = c(10), startweights = winit, threshold =
threshold[i]/ 1000)
          train_pred <- compute(nn, covariate=tr)$net.result
          MSE = missclassfunc(train_pred, tr$Sin)
          missclass_train = c(missclass_train, MSE)
          valid_pred <- compute(nn, covariate=va)$net.result
          MSE = missclassfunc(valid_pred, va$Sin)
          missclass_valid = c(missclass_valid, MSE)
  }

# Plot the missclassifications to see which threshold is best
plot(threshold, missclass_train, type="l")
```

```
plot(threshold, missclass_valid, type="l")

# 4/1000 gives lowest SME on valid and train
nn = neuralnet(Sin ~ Var, data= trva, hidden = c(10), startweights = winit, threshold = 4 /
1000)

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