Lab3 TDDE01

Philip Palapelas Kantola. Elliot Magnusson, Arun Uppugunduri

Assignment 1

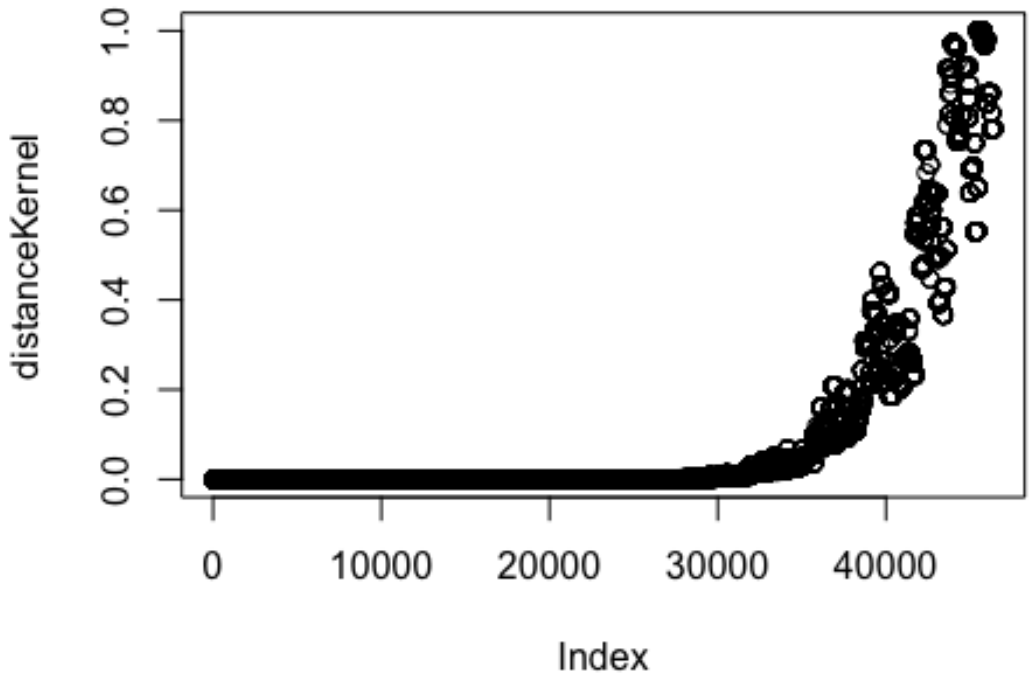
You are asked to provide a temperature forecast for a date and place in Sweden. The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels: ● The first to account for the distance from a station to the 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 mea- surement was made and the hour of interest. Choose an appropriate smoothing coefficient or width for each of the three kernels above.

Weather for the following time and location is to be predicted.

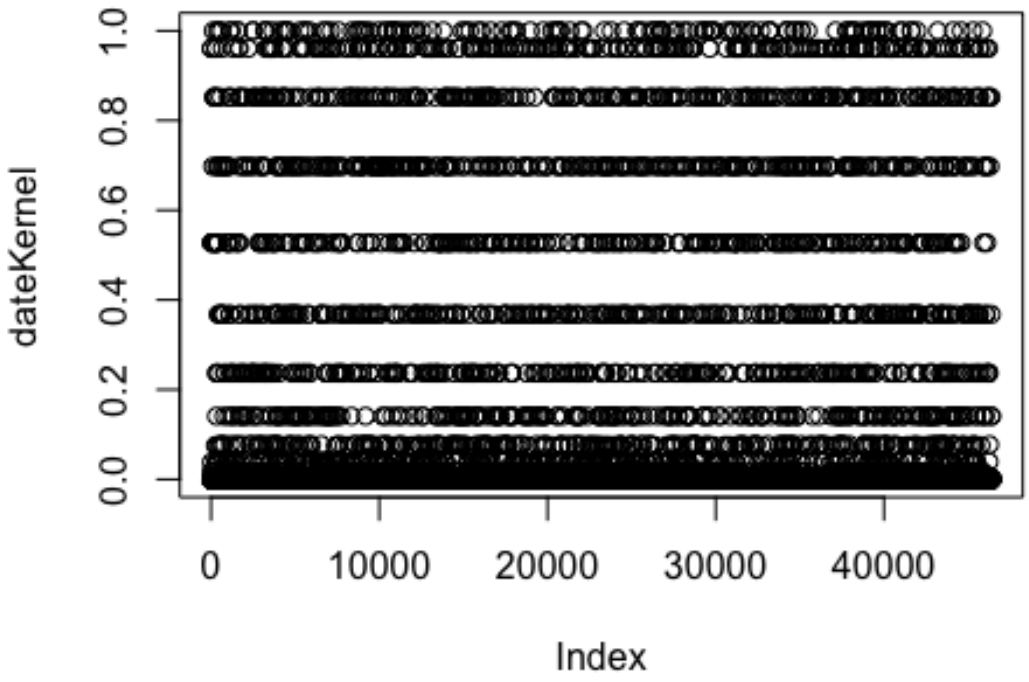
```
# coordinates for Abisko
latitude = 68.3557
longitude = 18.8206

# date to predict the weather for
myDate <- "2013-11-04"
```

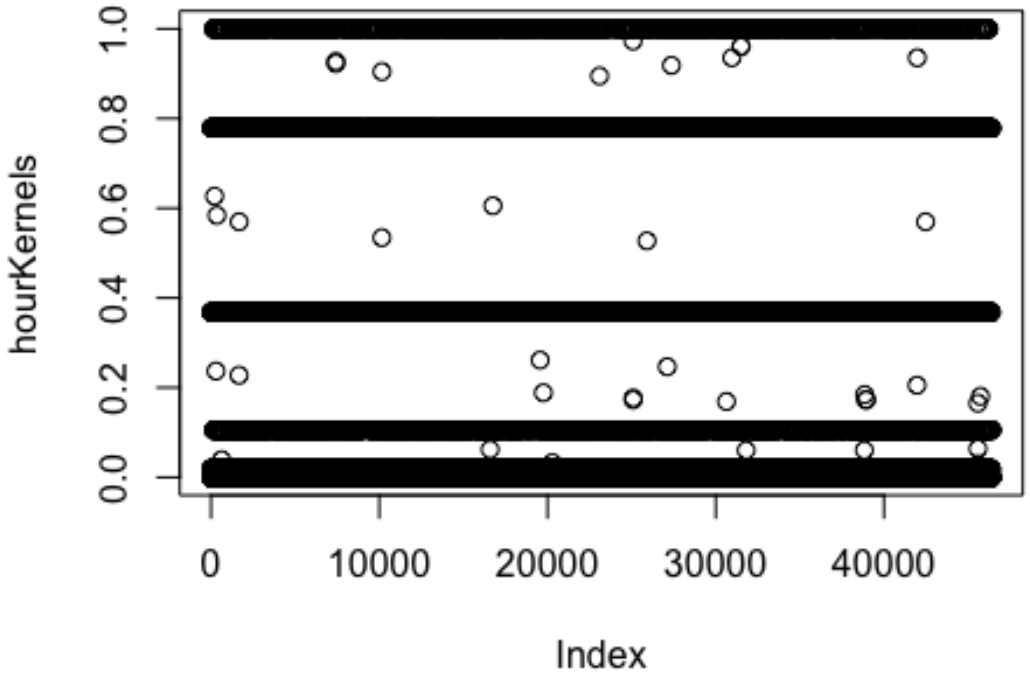
For the distance kernel a bandwidth of 300 000 was chosen. As seen in the following graph it is easy to differ between data points which are close to the chosen location and which are not, and locations far away from abisko gets a kernel value of 0, and are not considered in the prediction. H is set relatively high, since the weather usually is similar in a big area of the country. Since our location is very far north the locations which are closest are in the end of the data set. If a greater width is chosen all data points goes to the kernel value of 1, and if a smaller width is chosen the data points goes to a kernel value of 0.



For the date kernel a bandwidth of 5 was chosen. As seen in the graph below this means that about 10 preceding days is taken into account in the prediction. The rest of the days have a kernel value of 0 and does not affect the prediction. With a greater width it is very hard to differentiate between the data points, and more data points are considered. Now the majority of data points are collected in the bottom of the graph, with the data points closer in time (date) in the top of the graph, with the closest (the same day) having a kernel value of 1. The data points are weighted independent of year, so to say only the month and day matter.



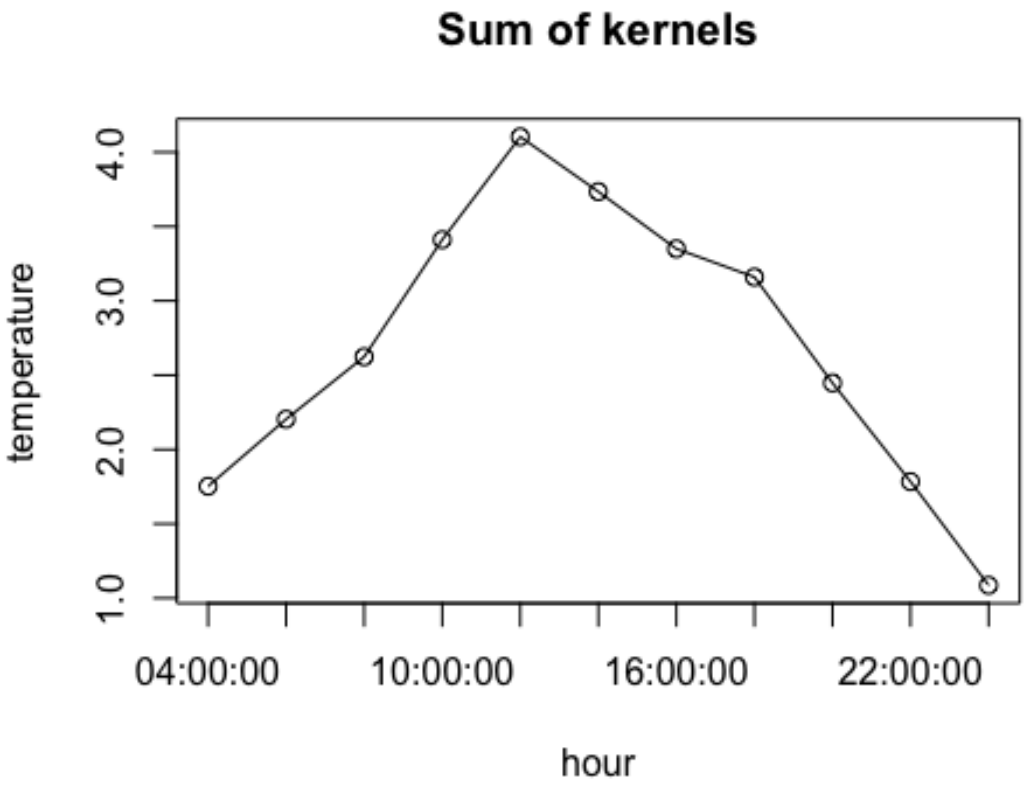
For the time kernel a bandwidth of 2 was chosen. Since the time is discrete with only a few different options of time most of the data points stack up in thick lines. With this wisth only the 3-4 preceding hours have an impact on the prediction. With a bigger value of h the lines move upwards until all data points have a kernel value of 1, and the opposite with an increasingly lower h. In this graph the time 14:00:00 is used as example.

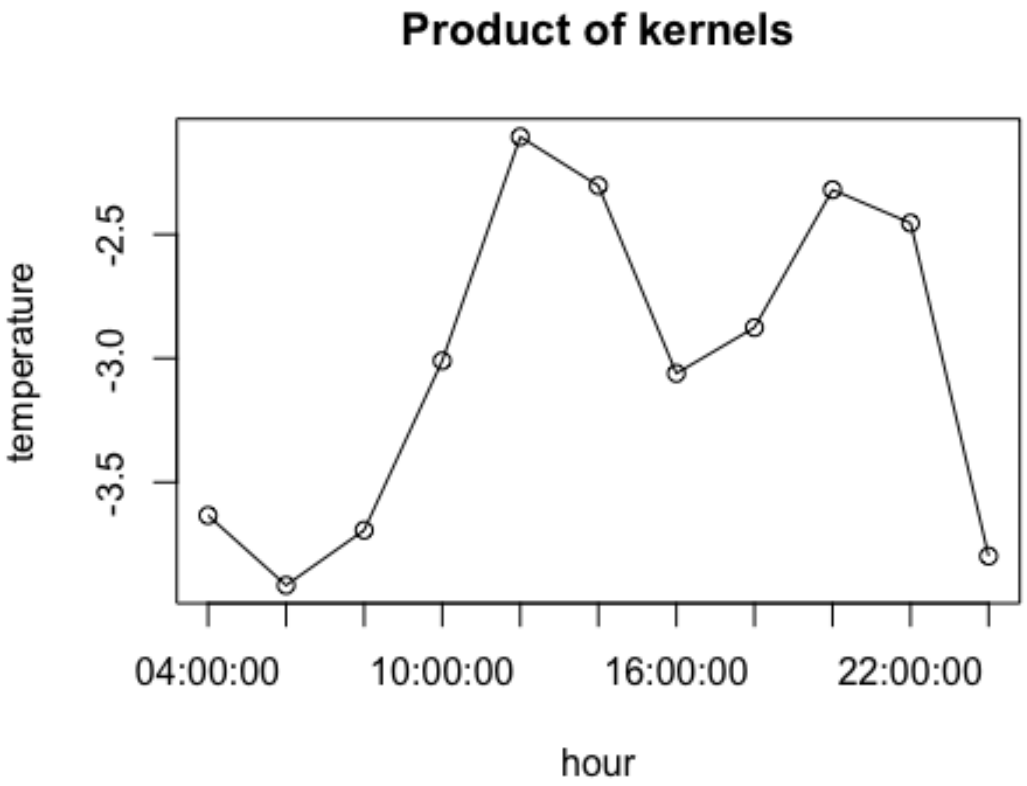


For prediction the following function is used;

$$y_k(\boldsymbol{x}) = \frac{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) t_n}{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)}$$

Below are the plots for summarized and multiplied kernels. The predicted temperatures from the sum of kernels seems to be kind of accurate with reality. The predicted temperatures from the product of kernels are lower and are not as realistic. The temperature rises by midnight and it also predicts a wierd dip in the middle of the day.


Sum of kernels
temperature
1.0
2.0
3.0
4.0
04:00:00
10:00:00
16:00:00
22:00:00
hour



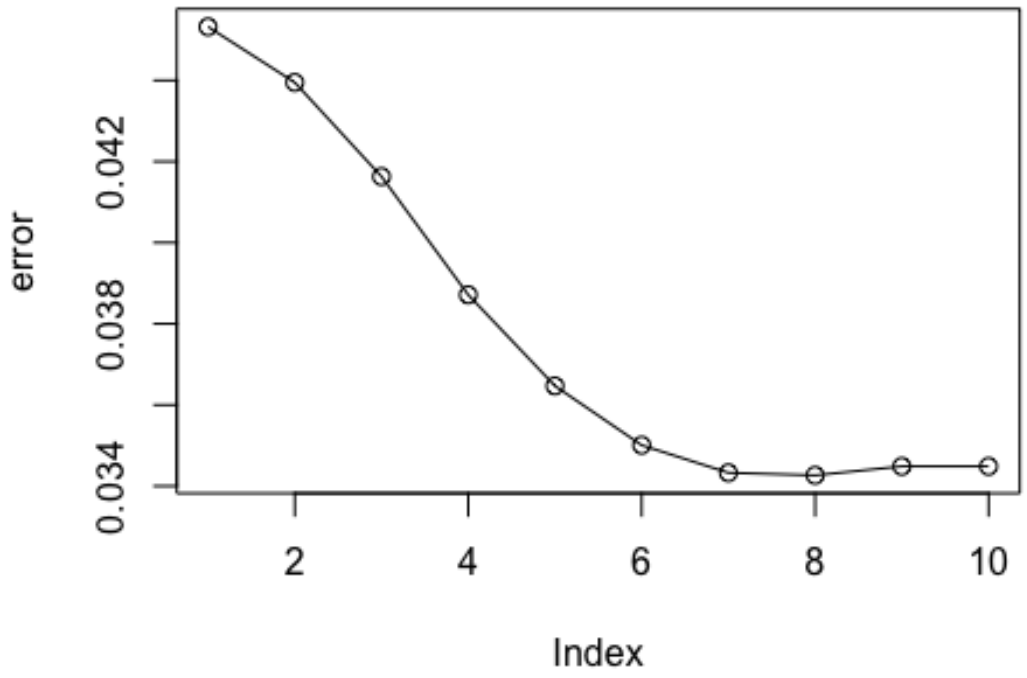
Assignment 3

Train a neural network to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting pairs are the data available to you. Use 25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent. That is, you should use the validation set to detect when to stop the gradient descent and so avoid overfitting. Stop the gradient descent when the partial derivatives of the error function are below a given threshold value.

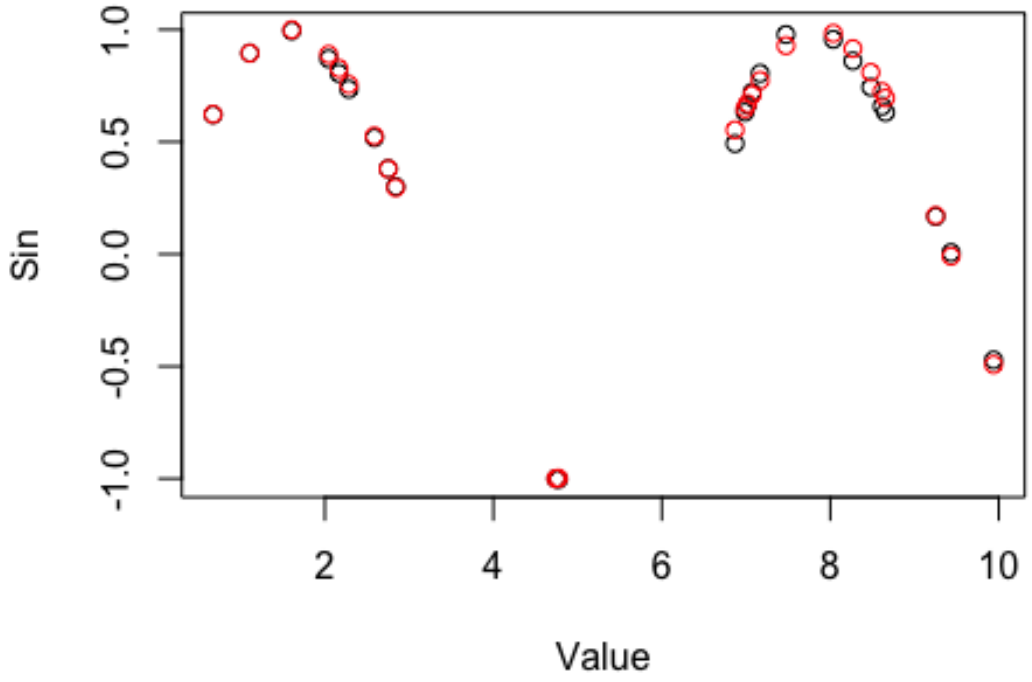
Threshold values between 1/1000 and 10/1000 is tried. The best result is given with 8/1000, using which.min(abs(error)) function. Below is a graph of the error values. Weights are randomized for all 31 vertices.

```
# Train model with different thresholds
for(i in 1:10) {

  nn <- neuralnet(Sin ~ Variables, threshold = i/1000, data = train , hidden
= 10, startweights = winit)
  prediction <- compute(nn, validate)
  error[i] <- mean(prediction$net.result^2-validate$Sin)
}
```



Here is a graph where the black dots represent the predicted values using threshold 8/1000, and the red dots represents the actual validation data, pretty accurate.



Appendix

Assignment 1 code

```
# somehow my working directory changed, so had to redirect it


set.seed(12345)
library(geosphere)
stations <- read.csv('stations.csv', fileEncoding="latin1")
temps <- read.csv('temps50k.csv')
stFull <- merge(stations,temps,by="station_number")

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")



# coordinates for Abisko
latitude = 68.3557
longitude = 18.8206
```

```r
# date to predict the weather for
myDate <- "2013-11-04"

st = subset(stFull, as.Date(stFull$date) <= as.Date(myDate))

help(subset)

coordinates = c(longitude, latitude)

# converts date from string to Date type
dateTemp <- as.Date(myDate)
# take out only month and day from date
dateFormatted = substr(dateTemp, 6, 10)


# width constant for distance kernel, needs to be big since the distance is m
easured in meters
h_distance = 200000
# width constant for date kernel
h_date = 5
# width constant for hour kernel
h_time = 2

distanceVector = 1:length(st[,1])

# create kernel vector which transposes data points to higher dimension depen
ding
# on how close they are to abisko (the chosen location)
distanceKernel = 1:length(st[,1])
for (i in 1:length(st[,1])) {
  # distHaversine is function which calculates shortest distance between
  # two coordinates taking in account the curvature of earth.
  # x is distance from chosen location to data point, divided by a width cons
tant
  x = distHaversine(coordinates,c(st$longitude[i], st$latitude[i]))/h_distanc
e
  # gaussian kernel function
  distanceKernel[i] = exp(-x^2)
}
# plot to see if h is reasonable, should be able to diffirentiate between dat
a points
plot(distanceKernel)

# kernel vector transposing data points depending on the difference in days
dateKernel = 1:length(st[,1])
for (i in 1:length(st[,1])) {
  # difftime gives difference in time between the two dates, independent of y
ear
```

```
  diffDays = abs(difftime(strptime(substr(st$date[i], 6, 10), format = "%m-%d
"),
                  strptime(dateFormatted, format = "%m-%d"), units = "days"))
  # if days are past half a year away, the counting reverts
  if(isTRUE(diffDays > 183)) {
    diffDays = 365 - diffDays
  }
  x = diffDays/h_date
  # gaussian kernel function
  dateKernel[i] = exp(-as.numeric(x)^2)
}
plot(dateKernel)


hourKernel <- function(st, h, hour) {
# kernel vector transposing data points depending on the difference in hours
of the day
hourKernelVector = 1:length(st[,1])
# converts time from string to time type
hourFormatted = as.POSIXct(hour, format = "%H:%M:%S")
for (i in 1:length(st[,1])) {
  x = difftime(as.POSIXct(st$time[i], format = "%H:%M:%S"),hourFormatted, uni
ts="hours")/h
  # gaussian kernel function
  hourKernelVector[i] = exp(-as.numeric(x)^2)
}
return(hourKernelVector)
}
# vector to store the sum kernel for each hour
tempSum <- vector(length=length(times))
tempProduct <- vector(length=length(times))
# sum the kernels to one
finalKernelSum = 1:length(st[,1])
finalKernelProduct = 1:length(st[,1])
for (i in 1:length(times)) {

  finalKernelSum = distanceKernel + dateKernel + hourKernel(st,h_time,times[i
])
  finalKernelProduct = distanceKernel * dateKernel * hourKernel(st, h_time, t
imes[i])
  #na.rm=T is used, seems like some value(s) in data set is NA , this will av
oid the problem
  tempSum[i] = sum(finalKernelSum*st$air_temperature, na.rm=T)/sum(finalKerne
lSum, na.rm=T)
  tempProduct[i] = sum(finalKernelProduct*st$air_temperature, na.rm=T)/sum(fi
nalKernelProduct, na.rm=T)
}


#xaxt removes labels from x axis
```

```r
plot(tempSum,  type="o", xlab = "hour",xaxt = "n", ylab = "temperature", main
= "Sum of kernels")
# axis puts labels to axis
axis(1, at=1:length(times), labels=times)
plot(tempProduct,  type = "o",  xlab = "hour",xaxt = "n", ylab="temperature",
main = "Product of kernels")
# axis puts labels to axis
axis(1, at=1:length(times), labels=times)
```

Assignment 3 code

```r
# neuralnet package requires numeric inputs and does not play nicely with fac
tor variables
library(neuralnet)
set.seed(1234567890)
# generate 50 random values (uniformly distributed) between 0 and 10.
Variables <- runif(50, 0, 10)
trva <- data.frame(Variables, Sin=sin(Variables))
train <- trva[1:25,] # Training
validate <- trva[26:50,] # Validation

# Random initialization of the weights in the interval [-1, 1]
# 31 weights, one for every  vertex (båg?)
set.seed(1234567890)
winit <- runif(31,-1,1)

error <- 1:10
# Train model with different thresholds
for(i in 1:10) {

  nn <- neuralnet(Sin ~ Variables, threshold = i/1000, data = train , hidden
= 10, startweights = winit)
  prediction <- compute(nn, validate)
  error[i] <- mean(prediction$net.result^2-validate$Sin)
}
# returns smallest value of a vector (7/1000)
which.min(abs(error))
plot(error,type="o")

# final neuralnet, the best one
nn <- neuralnet(Sin ~ Variables, threshold = 8/1000, data = train , hidden =
10, startweights = winit)
prediction <- compute(nn, validate)

plot(validate$Variables, prediction$net.result,  ylab = "Sin", xlab = "Value"
)
points(validate$Variables, validate$Sin, col = "red")
```