ML_Lab03

Question 1:

Loading the required data and library and the stations at different d ays a nd t imes. T he d ata h ave been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI).

0.0.1 defining s oothing factor

```
h_distance <- 1000000
h_date <- 10
h_time <- 3
H_vals <- data.frame(h_distance, h_date, h_time)
```

point of interests

```
a <- 55.3836 # latitude of interest
b <- 12.8203 # longitude of interest
```

The date to predict for which we want to predict

```
date <- "2004-05-28"
targets <- data.frame(a, b, date)
```

Combining both set of data

```
st = merge(stations,temps,by="station_number")
```

Required intervals to predicts

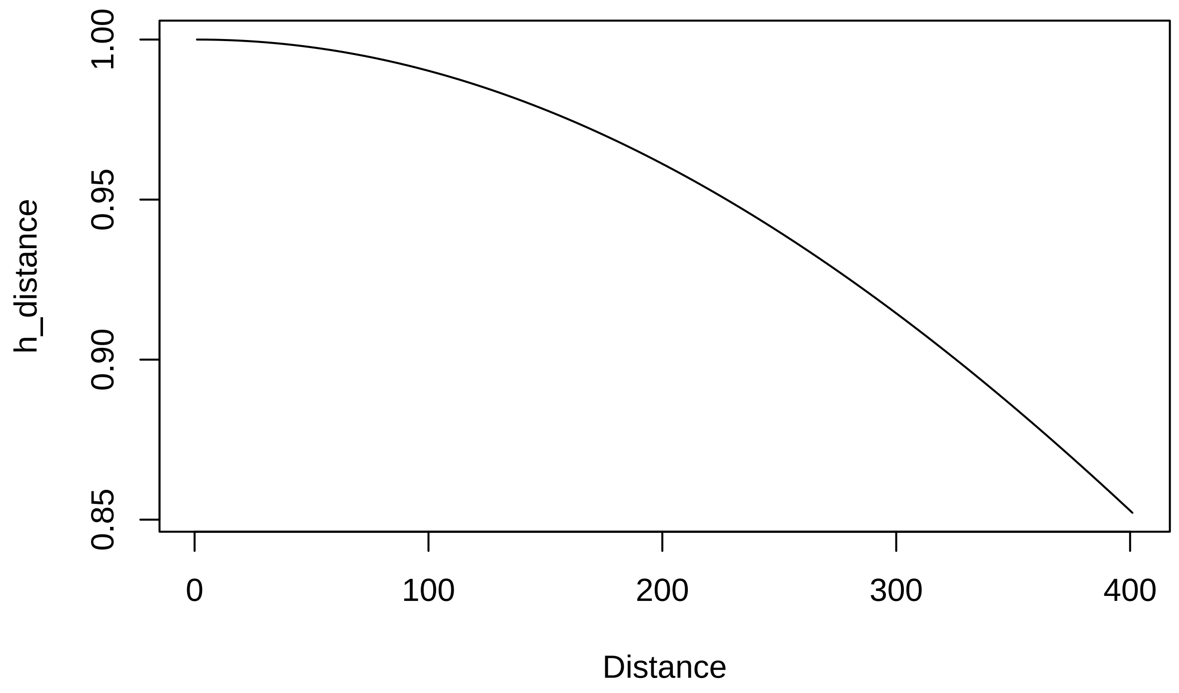
```
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00",
           "14:00:00", "16:00:00", "18:00:00", "20:00:00", "22:00:00", "00:00:00")
```

distance to find good h-values, our parameters that are returned from distHaversine are returned in meter format

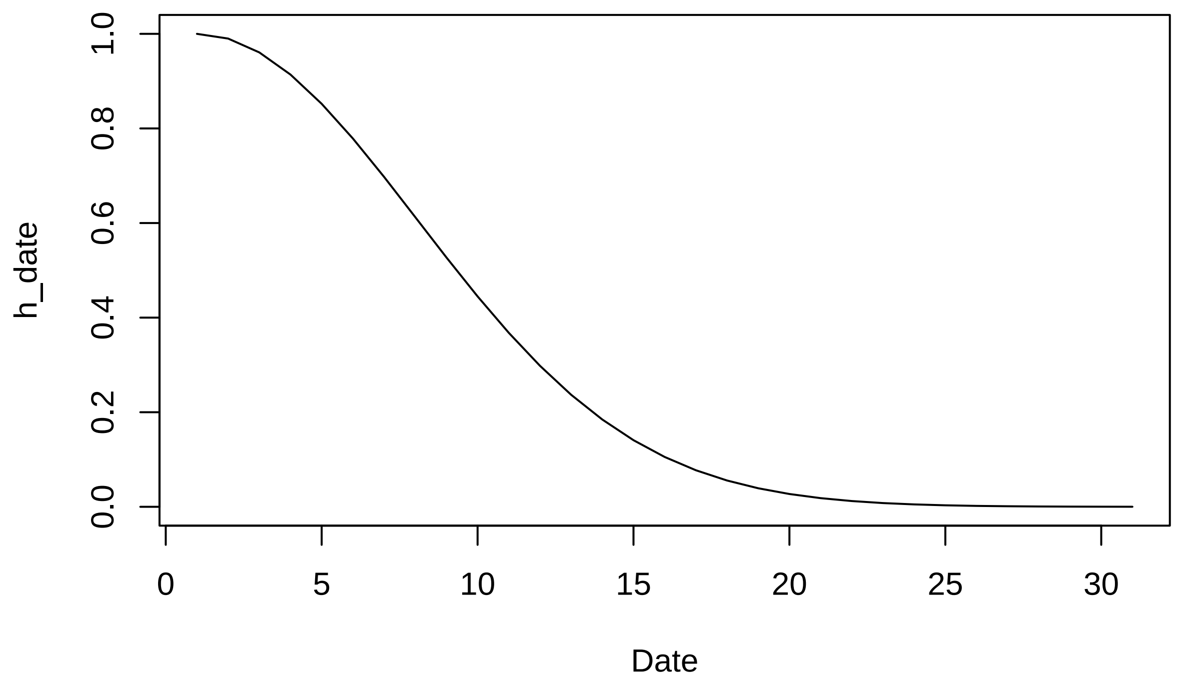
```
distance <- seq(0,400000,1000)
date_diff <- seq(0,30,1)
time_diff <- seq(0,12,1)
```

Calculating guassian kernels

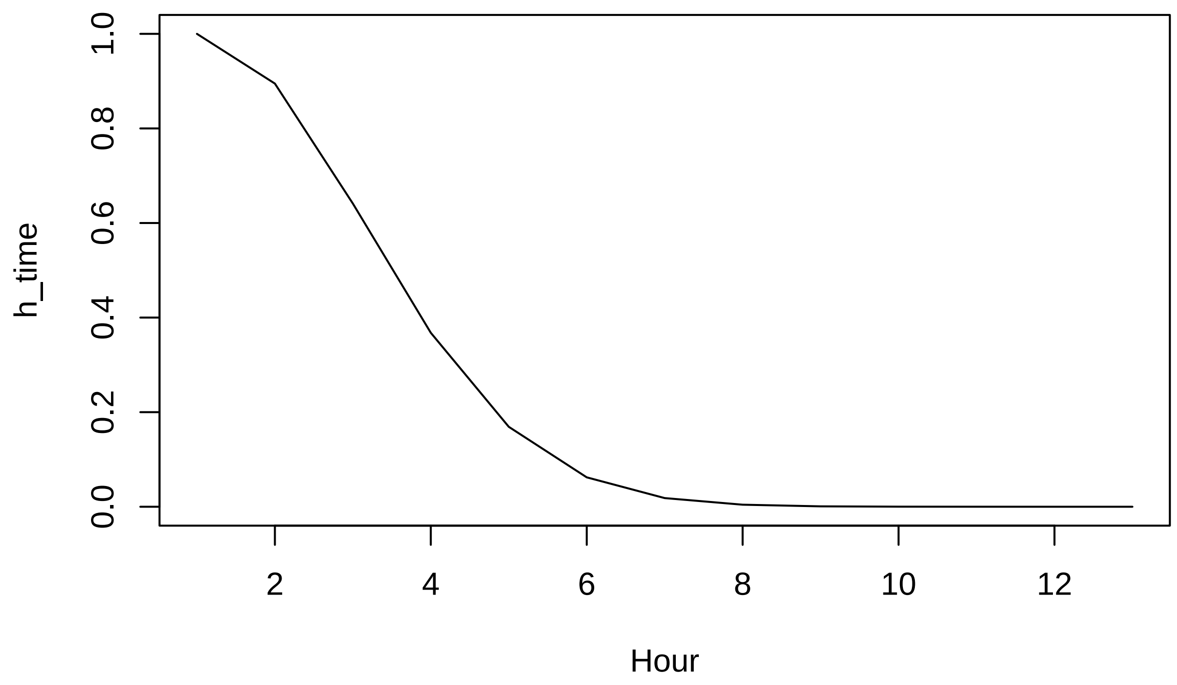
```
h_distance <- exp(-(distance/h_distance)^2)
# u <- distance/h_distance
  # k <- 1/exp(u^2)
 plot(h_distance, type="l", xlab = "Distance")
```



```
h_date <- exp(-(date_diff/h_date)^2)
 plot(h_date, type="l", xlab = "Date")
```



```
h_time <- exp(-(time_diff/h_time)^2)
plot(h_time, type="l", xlab = "Hour")
```



Filter dates in data posterior to target date and Drop rows with date greater than selected date

```r
dateData <- function(data, date) {
  res <-data[!(as.Date(data$date) > as.Date(date)),]
  return (res)
}
```

Filter data to use for prediction

```r
filter_date_data=dateData(st, date)

# Gaussian Kernel for distance
dist_kernel <- function(data, target, h) {
  dist_dif <- distHaversine(data.frame(data$longitude, data$latitude), target)
  k <- 1/exp((dist_dif/h)^2)
  return(k)
}
```

Calculate Gaussian Kernel for day

```r
d_kernerl <- function(data, date1, h) {
  dif_day <-as.numeric(as.Date(data$date) - as.Date(date1), unit="days")
  dif_day <- abs(dif_day)
  k <- 1/exp((dif_day/h)^2)
  return(k)
}
```

Calculate Hours kernel

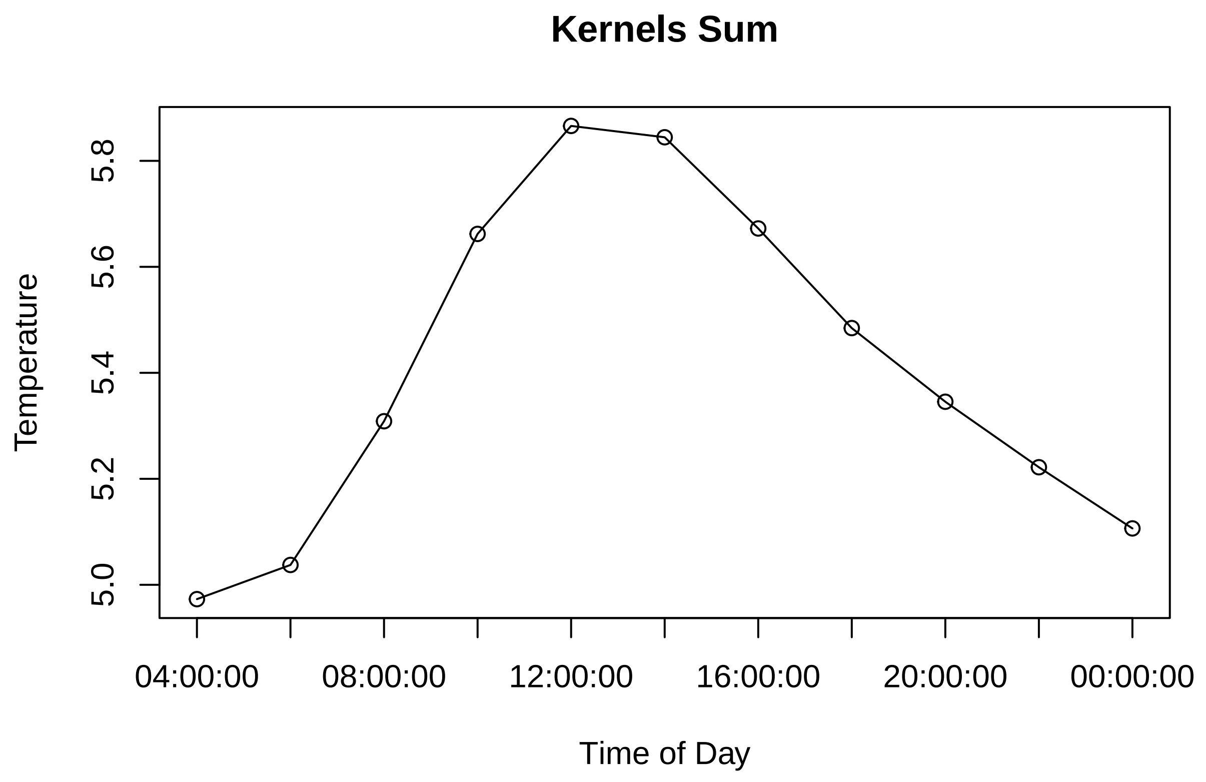
```r
h_kernel <- function(data, time, h) {
  t_diff <- as.numeric(difftime(strptime(data$time , format = "%H:%M:%S"),
  strptime(time , format = "%H:%M:%S")))

  t_diff = t_diff/3600
  for(i in 1:length(t_diff)){
    if (t_diff[i]<(-12)){
      t_diff[i]<-t_diff[i] %% 24
    }
    else if (t_diff[i]>12){
      t_diff[i]<-abs(t_diff[i] - 24)
    }
  }
  u <- t_diff/h
  k <- 1/exp(u^2)
  return(k)
}
```

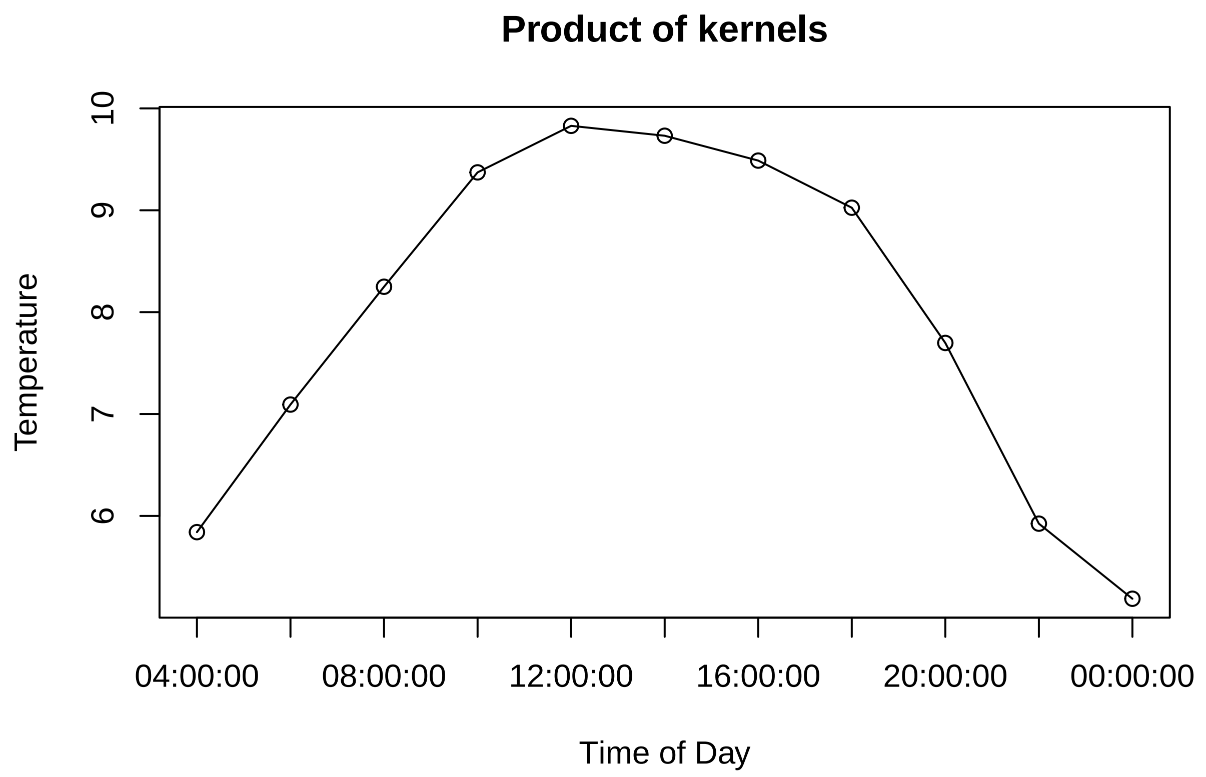
1 Multiplication and Summation of kernels

In below prediction one is based on the sum and other one is on the product of the three different kernels. Temperature predictions is worse in the kernel summation of the compared to the product of kernels. As the sum of the kernels basically need one kernel. to have relatively high values in order to consider the measurement relevant however the product of the kernels consider all the values of measurement, for the kernels need to have a relatively large value in order to make a difference in the prediction.

```r
plot(estimated_temp$temp_sum, xaxt = "n",
            xlab="Time of Day",
            ylab="Temperature",
            type="o", main = "Kernels Sum", axis=FALSE)
axis(1, at=1:length(times), labels=times)
```



```
plot(estimated_temp$temp_mult,
            xaxt = "n",
            xlab="Time of Day",
            ylab="Temperature", type="o",
            main = "Product of kernels")
axis(1, at=1:length(times), labels=times)
```


Product of kernels
Temperature
10
9
8
7
6
04:00:00
08:00:00
12:00:00
16:00:00
20:00:00
00:00:00
Time of Day

Question2:

1.

We have 4 models with diffrent errors. error of filter0 is equal to 0.0675, error of filter1 is equal to 0.08489388, error of filter2 is equal to 0.082397 and error of filter3 is equal to 0.02122347. Although, error of filter3 is minimum among the others, the point is that, in filter3, we train our model through the whole data including test part and we calculate error on test dataset. so we choose filter0, which is minimum after filter3, which train the model on train data and run the model on validation data.

2.

Generalization error is the error obtained by applying a model to data it has not seen before. So, if we want to measure generalization error, we need to remove a subset from your data and don't train our model on it. According to above concept, all of errors except error3 can estimate generalization error, cause as above mentioned, filter3 trained on the whole data and the erro is on test data which is not unseen, but cause we chose filter0, our estimate of generalization error would be error0.

3.

The code has been written based on the template provided and can be seen in Appendix part.

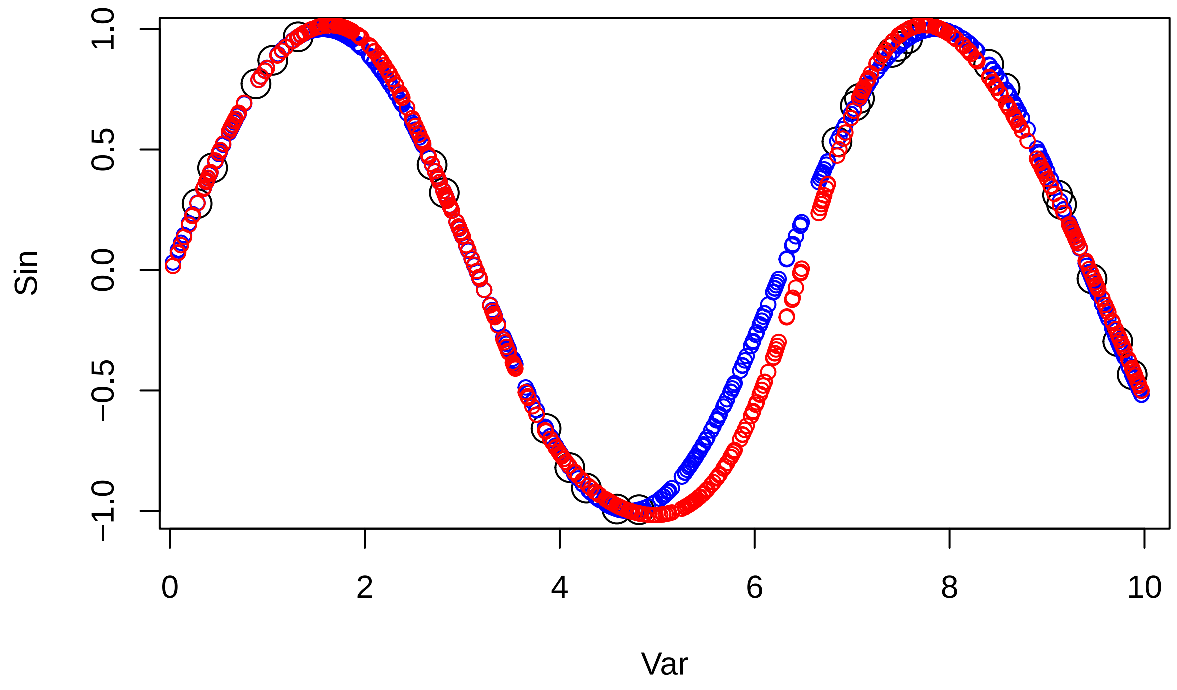
Question3:

1.

Train the neural network model with neuralnet(), and plot the result with training data in black, test data in blue, and predicted test data in red.

```r
winit <- runif(501, min=-1, max=1)
nn <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit)
```

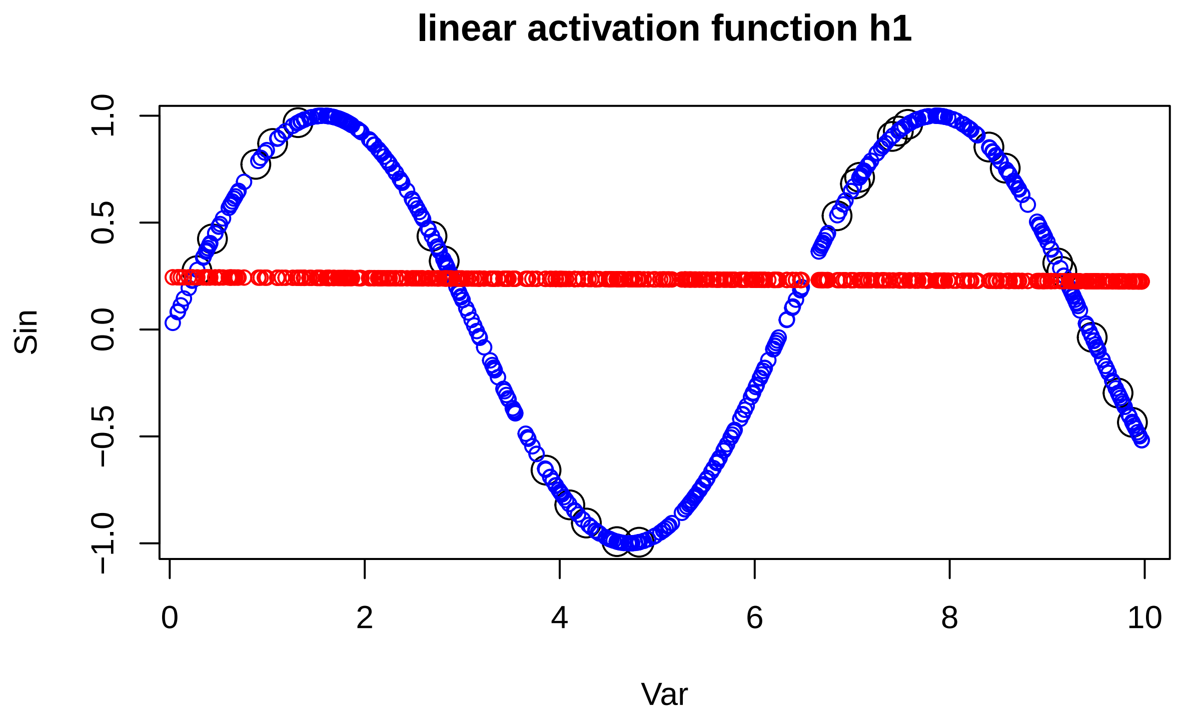
```
## Warning: package 'neuralnet' was built under R version 4.1.2
```

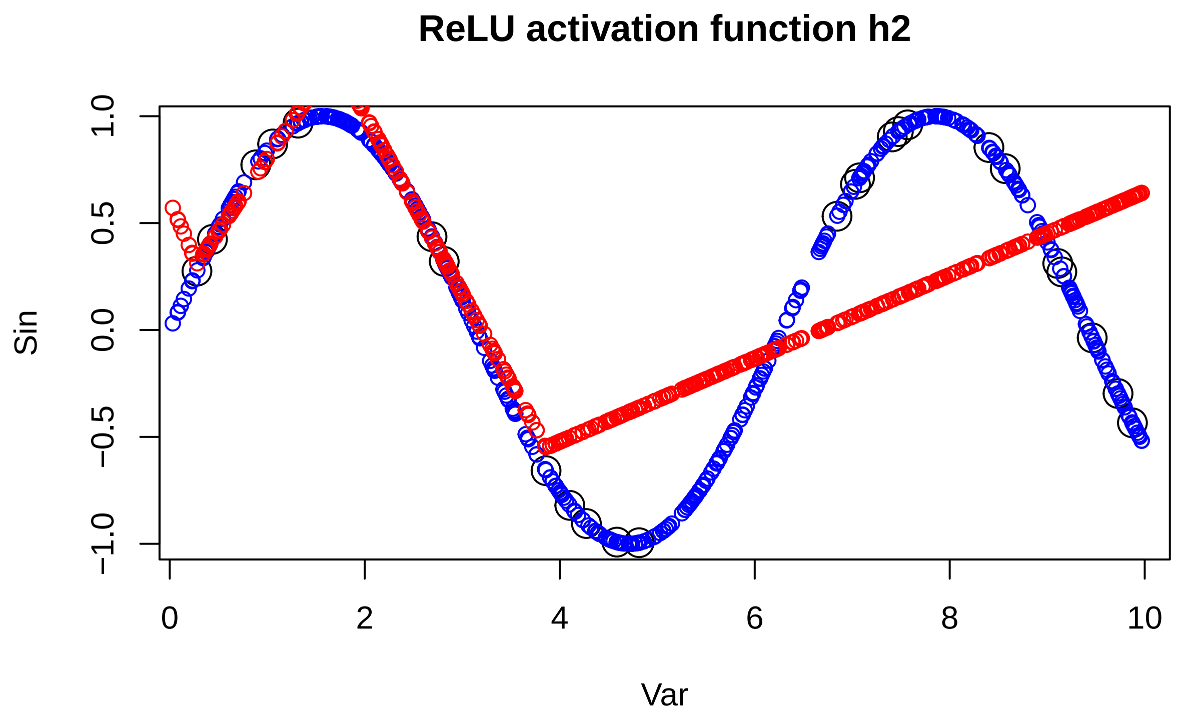


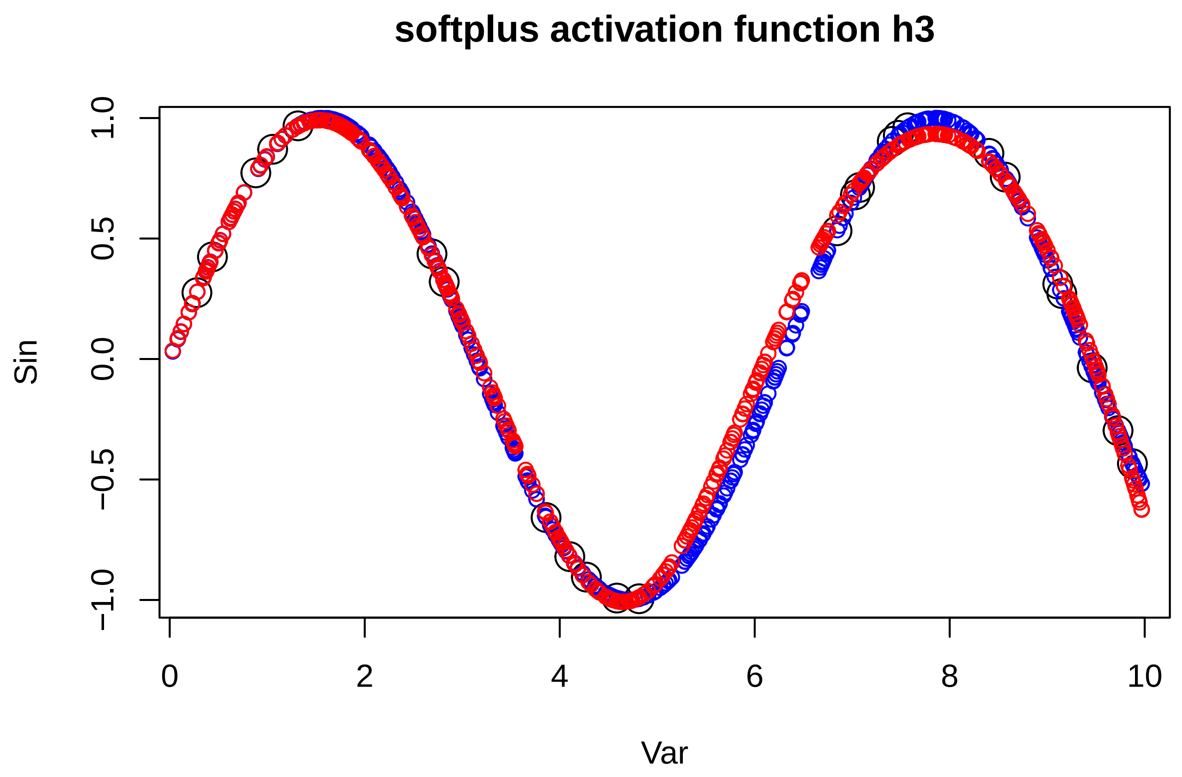
From the plot, the model fits quite well. Error can be observed for Var in [5,6] where there is no training data in this range.

2.

The below three figures shows the result with linear, ReLU and softplus activation function.

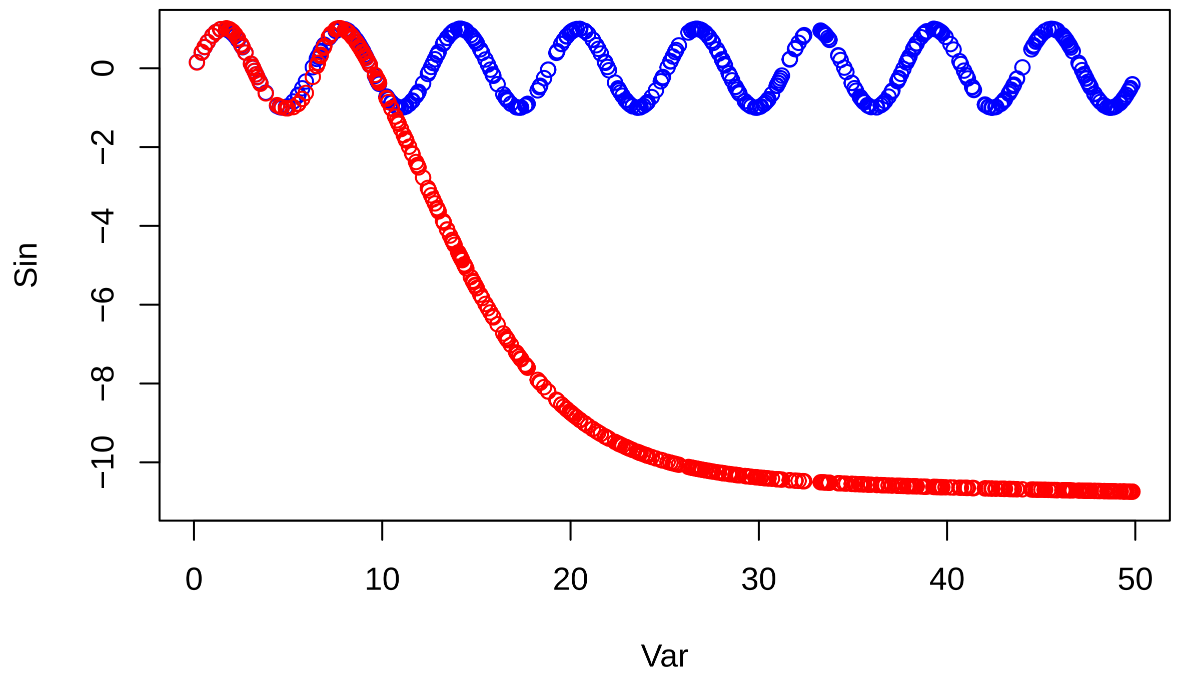

linear activation function h1
Sin
Var
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10


ReLU activation function h2
Sin
Var
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10



The models with linear and ReLU activation function predict the data quite badly, while the model with softplus activation function fit the test data very well.

3.



The model fits well in the [0,10] range, which is trained data range, fits badly outside this range.

4.

The weights of the NN learned is printed below.

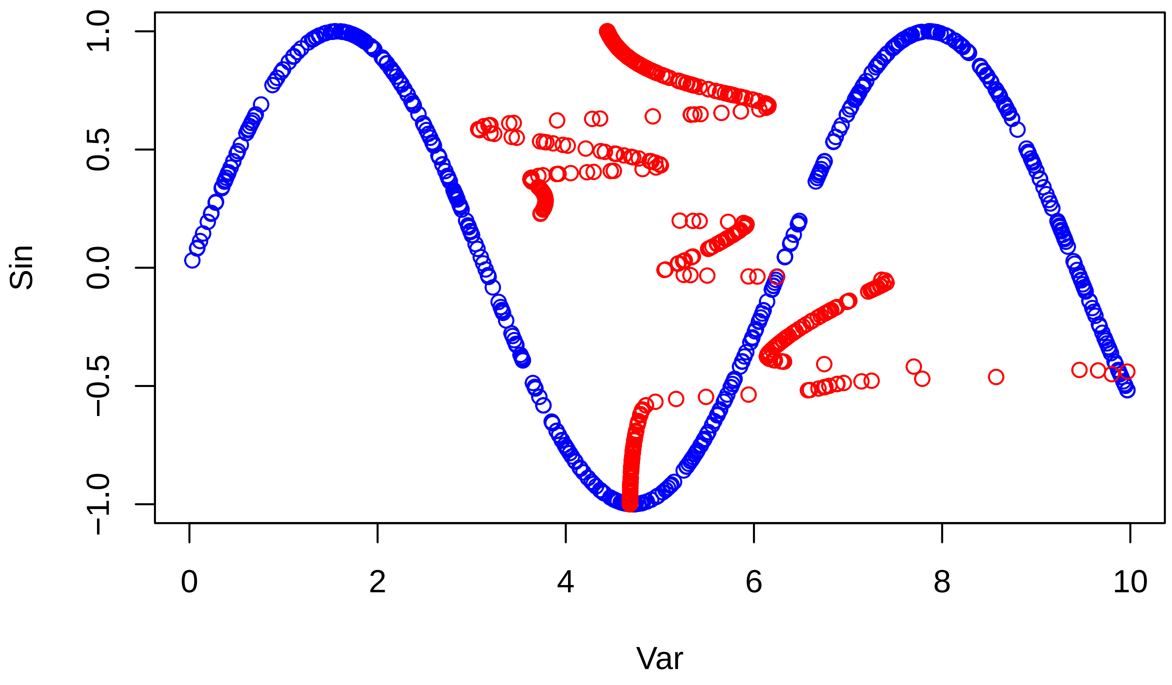
```
## [[1]]
## [[1]][[1]]
##            [,1]       [,2]       [,3]       [,4]      [,5]       [,6]      [,7]
## [1,] -11.870831 -0.9153178 -3.0316259  1.5313105  6.555437 -11.768525  1.397325
## [2,]   4.042494 -0.5368677  0.2603355 -0.5487656 -2.233839   1.787862 -1.259834
##            [,8]       [,9]       [,10]
## [1,]  0.2032402  0.2948804 -0.21326487
## [2,] -2.1974036 -0.5886005 -0.03177278
##
## [[1]][[2]]
##              [,1]
##  [1,]  -0.1088032
##  [2,]   0.7716196
##  [3,]  -1.0436754
##  [4,] -16.0529801
##  [5,]  -1.2718986
##  [6,]   3.2805756
##  [7,]   4.3076898
##  [8,]   0.1220703
##  [9,]  -1.2234185
## [10,]  -2.7618205
## [11,]   2.3696931
```

From the result we can see the weights (W1) of for the hidden units is quite big, while b1 is not very large in value. In this way, when the value of test data feature Var increases, the value of hidden units also increases, and after the the activation function, the output value of each hidden unit becomes saturated (either close to 1 or 0 for logical activation function), and then the final result converge to some value.

5.

Train the neural network model with neuralnet(), and plot the result

```r
winit <- runif(10001, min=-1, max=1)
nn_2 <- neuralnet(Var ~.,data = mydata, hidden = 10, startweights = winit, threshold=0.1)
```



The model fits very badly, as when predict x from sin(x), the same feature sin(x) value can map to different x values, which leads to the bad result.

Appendix:

```r
knitr::opts_chunk$set(echo = TRUE)
set.seed(1234567890)
library(geosphere)

stations <- read.csv('stations.csv')

temps <- read.csv('temps50k.csv') h_distance <- 1000000
h_date <- 10
h_time <- 3
H_vals <- data.frame(h_distance, h_date, h_time)

a <- 55.3836 # latitude of interest
b <- 12.8203 # longitude of interest
date <- "2004-05-28"
targets <- data.frame(a, b, date)
st = 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", "00:00:00")
distance <- seq(0,400000,1000)
date_diff <- seq(0,30,1)
time_diff <- seq(0,12,1)
h_distance <- exp(-(distance/h_distance)^2)
# u <- distance/h_distance
  # k <- 1/exp(u^2)
plot(h_distance, type="l", xlab = "Distance")
h_date <- exp(-(date_diff/h_date)^2)
 plot(h_date, type="l", xlab = "Date")
h_time <- exp(-(time_diff/h_time)^2)
plot(h_time, type="l", xlab = "Hour")
dateData <- function(data, date) {
  res <-data[!(as.Date(data$date) > as.Date(date)),]
  return (res)
}
filter_date_data=dateData(st, date)

# Gaussian Kernel for distance
dist_kernel <- function(data, target, h) {
  dist_dif <- distHaversine(data.frame(data$longitude, data$latitude), target)
  k <- 1/exp((dist_dif/h)^2)
  return(k)
}
d_kernerl <- function(data, date1, h) {
  dif_day <-as.numeric(as.Date(data$date) - as.Date(date1), unit="days")
  dif_day <- abs(dif_day)
  k <- 1/exp((dif_day/h)^2)
  return(k)
}
h_kernel <- function(data, time, h) {
  t_diff <- as.numeric(difftime(strptime(data$time , format = "%H:%M:%S"),
  strptime(time , format = "%H:%M:%S")))

  t_diff = t_diff/3600
  for(i in 1:length(t_diff)){
    if (t_diff[i]<(-12)){
      t_diff[i]<-t_diff[i] %% 24
    }
    else if (t_diff[i]>12){
      t_diff[i]<-abs(t_diff[i] - 24)
    }
  }
  u <- t_diff/h
  k <- 1/exp(u^2)
  return(k)
}
estimat_temp <- function(data, target, smoothH, times) {
  #Basically a vector of values between 0-1 depending on the
  #location of the data we are #comparing to.
  #The closer the location -> The bigger the value
  k_distance <- dist_kernel(data, c(target$b[1], target$a[1]), smoothH$h_distance[1])

  # Basically a vector of values between 0-1 depending on the date of the data we are
  # comparing to. The closer the date -> The bigger the value
  k_day <- d_kernerl(data, target$date[1], smoothH$h_date[1])

  # Constructing vectors that we can store the temperature
   # values based on the summation and the product of the kernels
  temp_sum <- vector(length = length(times))
  temp_mult <- vector(length = length(times))

  for (i in 1:length(times)){
    # vector of values between 0-1 depending on the time of the data we are
    # comparing to. The closer the time -> The bigger the value
    k_hour <- h_kernel(data, times[i], smoothH$h_time[1])

    #Summation of kernels
    kernel_sum <- k_distance + k_day + k_hour

    #Product of kernels
    kernel_multiplication <- k_distance * k_day * k_hour

    #temperature values between 4am to 24pm
    #Kernel-weighted average over all the points slide 8, 3a
    temp_sum[i] <- sum(kernel_sum %*% data$air_temperature)/sum(kernel_sum)
    temp_mult[i] <- sum(kernel_multiplication %*% data$air_temperature)/sum(kernel_multiplication)
  }
  return(list(temp_sum = temp_sum, temp_mult = temp_mult))
}


estimated_temp <- estimat_temp(filter_date_data, targets, H_vals, times)

plot(estimated_temp$temp_sum, xaxt = "n",
            xlab="Time of Day",
            ylab="Temperature",
            type="o", main = "Kernels Sum", axis=FALSE)
```

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

plot(estimated_temp$temp_mult,
            xaxt = "n",
            xlab="Time of Day",
            ylab="Temperature", type="o",
            main = "Product of kernels")
axis(1, at=1:length(times), labels=times)
winit <- runif(501, min=-1, max=1)
nn <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit)
library(neuralnet)
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test
winit <- runif(501, min=-1, max=1)
nn <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit)
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
h1 <- function(x) x                   # linear
h2 <- function(x) ifelse(x<0,0,x)  # ReLU
h3 <- function(x) log(1 + exp(x))    # softplus
#### linear activation function h1
nn_h1 <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit, act.fct = h1)
plot(tr, cex=2, main="linear activation function h1")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_h1,te), col="red", cex=1)
# the fit is quite bad
#### ReLU activation function h2
nn_h2 <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit, act.fct = h2)
plot(tr, cex=2, main="ReLU activation function h2")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_h2,te), col="red", cex=1)
# the fit is also bad
#### softplus activation function h3
nn_h3 <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit, act.fct = h3)
plot(tr, cex=2, main="softplus activation function h3")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_h3,te), col="red", cex=1)

set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata <- data.frame(Var, Sin=sin(Var))
plot(mydata, col = "blue", cex=1, ylim=c(-11,1))
points(mydata[,1],predict(nn,mydata), col="red", cex=1)

nn$weights

winit <- runif(10001, min=-1, max=1)
nn_2 <- neuralnet(Var ~.,data = mydata, hidden = 10, startweights = winit, threshold=0.1)
set.seed(1234567890)
```

```r
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
winit <- runif(10001, min=-1, max=1)
nn_2 <- neuralnet(Var ~.,data = mydata, hidden = 10, startweights = winit, threshold=0.1)
plot(mydata, col = "blue", cex=1)
points(predict(nn_2,mydata),mydata[,2], col="red", cex=1)
# Question1

set.seed(1234567890)
library(geosphere)

stations <- read.csv('stations.csv')


temps <- read.csv('temps50k.csv')


##defining soothing factor
h_distance <- 1000000
h_date <- 10
h_time <- 3
H_vals <- data.frame(h_distance, h_date, h_time)

#point of interests

a <- 55.3836 # latitude of interest
b <- 12.8203 # longitude of interest
#The date to predict for which we want to predict

date <- "2004-05-28"
targets <- data.frame(a, b, date)

#Combining both set of data

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

#Required intervals to predicts

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")
#distance to find good h-values, our parameters that are returned
#from distHaversine are returned in meter format

distance <- seq(0,400000,1000)
date_diff <- seq(0,30,1)
time_diff <- seq(0,12,1)

#Calculating guassian kernels

h_distance <- exp(-(distance/h_distance)^2)
# u <- distance/h_distance
# k <- 1/exp(u^2)
plot(h_distance, type="l", xlab = "Distance")
h_date <- exp(-(date_diff/h_date)^2)
plot(h_date, type="l", xlab = "Date")
```

```r
h_time <- exp(-(time_diff/h_time)^2)
plot(h_time, type="l", xlab = "Hour")

#Filter dates in data posterior to target date and Drop rows with date greater than selected date

dateData <- function(data, date) {
  res <-data[!(as.Date(data$date) > as.Date(date)),]
  return (res)
}

#Filter data to use for prediction

filter_date_data=dateData(st, date)

# Gaussian Kernel for distance
dist_kernel <- function(data, target, h) {
  dist_dif <- distHaversine(data.frame(data$longitude, data$latitude), target)
  k <- 1/exp((dist_dif/h)^2)
  return(k)
}

#Calculate Gaussian Kernel for day


d_kernerl <- function(data, date1, h) {
  dif_day <-as.numeric(as.Date(data$date) - as.Date(date1), unit="days")
  dif_day <- abs(dif_day)
  k <- 1/exp((dif_day/h)^2)
  return(k)
}

#Calculate Hours kernel

h_kernel <- function(data, time, h) {
  t_diff <- as.numeric(difftime(strptime(data$time , format = "%H:%M:%S"),
                              strptime(time , format = "%H:%M:%S")))

  t_diff = t_diff/3600
  for(i in 1:length(t_diff)){
    if (t_diff[i]<(-12)){
      t_diff[i]<-t_diff[i] %% 24
    }
    else if (t_diff[i]>12){
      t_diff[i]<-abs(t_diff[i] - 24)
    }
  }
  u <- t_diff/h
  k <- 1/exp(u^2)
  return(k)
}


# Multiplication and Summation of kernels
```

```r
estimat_temp <- function(data, target, smoothH, times) {
  #Basically a vector of values between 0-1 depending on the
  #location of the data we are #comparing to.
  #The closer the location -> The bigger the value
  k_distance <- dist_kernel(data, c(target$b[1], target$a[1]), smoothH$h_distance[1])

  # Basically a vector of values between 0-1 depending on the date of the data we are
  # comparing to. The closer the date -> The bigger the value
  k_day <- d_kernerl(data, target$date[1], smoothH$h_date[1])

  # Constructing vectors that we can store the temperature
  # values based on the summation and the product of the kernels
  temp_sum <- vector(length = length(times))
  temp_mult <- vector(length = length(times))

  for (i in 1:length(times)){
    # vector of values between 0-1 depending on the time of the data we are
    # comparing to. The closer the time -> The bigger the value
    k_hour <- h_kernel(data, times[i], smoothH$h_time[1])

    #Summation of kernels
    kernel_sum <- k_distance + k_day + k_hour

    #Product of kernels
    kernel_multiplication <- k_distance * k_day * k_hour

    #temperature values between 4am to 24pm
    #Kernel-weighted average over all the points slide 8, 3a
    temp_sum[i] <- sum(kernel_sum %*% data$air_temperature)/sum(kernel_sum)
    temp_mult[i] <- sum(kernel_multiplication %*% data$air_temperature)/sum(kernel_multiplication)
  }
  return(list(temp_sum = temp_sum, temp_mult = temp_mult))
}


estimated_temp <- estimat_temp(filter_date_data, targets, H_vals, times)

plot(estimated_temp$temp_sum, xaxt = "n",
     xlab="Time of Day",
     ylab="Temperature",
     type="o", main = "Kernels Sum", axis=FALSE)
axis(1, at=1:length(times), labels=times)

plot(estimated_temp$temp_mult,
     xaxt = "n",
     xlab="Time of Day",
     ylab="Temperature", type="o",
     main = "Product of kernels")
axis(1, at=1:length(times), labels=times)


# Question2
library(kernlab)
```

```r
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58])
tr <- spam[1:3000, ]
va <- spam[3001:3800, ]
trva <- spam[1:3800, ]
te <- spam[3801:4601, ]

by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){
  filter <- ksvm(type~.,
                 data=tr,
                 kernel="rbfdot",
                 kpar=list(sigma=0.05),
                 C=i,
                  scaled=FALSE)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}

filter0 <- ksvm(type~.,data=tr,
                kernel="rbfdot",
                kpar=list(sigma=0.05),
                C=which.min(err_va)*by,
                scaled=FALSE)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0

filter1 <- ksvm(type~.,data=tr,
                kernel="rbfdot",
                kpar=list(sigma=0.05),
                C=which.min(err_va)*by,
                scaled=FALSE)
mailtype <- predict(filter1,te[,-58])
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
err1

filter2 <- ksvm(type~.,data=trva,
                kernel="rbfdot",
                kpar=list(sigma=0.05),
                C=which.min(err_va)*by,
                scaled=FALSE)
mailtype <- predict(filter2,te[,-58])
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
```

```r
err2

filter3 <- ksvm(type~.,data=spam,
                kernel="rbfdot",
                kpar=list(sigma=0.05),
                C=which.min(err_va)*by,
                scaled=FALSE)
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3

sv<-alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)

rbf <- rbfdot(sigma = 0.05)

zx <- as.matrix(spam[1:10,-58])
zy <- as.matrix(spam[,-58])

k <- NULL
for(i in 1:10){
  k2<-NULL
  for(j in 1:length(sv)){
    k2[j] <- co[j] * rbf(c(zx[i,]), c(zy[sv[j],]))
    #print(k2[j])
    ss <- sum(k2)
  }
  k<-c(k,ss+inte)
}

as.matrix(k)
predict(filter3,spam[1:10,-58], type = "decision")

# Question3

library(neuralnet)
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test
winit <- runif(501, min=-1, max=1)
nn <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit)
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
# The model fits quite well. The error is bigger around [5,6] where
#there is no training data in this range.

# 2
h1 <- function(x) x                  # linear
```

```r
h2 <- function(x) ifelse(x<0,0,x)  # ReLU
h3 <- function(x) log(1 + exp(x))    # softplus
#### linear activation function h1
nn_h1 <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit, act.fct = h1)
plot(tr, cex=2, main="linear activation function h1")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_h1,te), col="red", cex=1)
# the fit is quite bad

#### ReLU activation function h2
nn_h2 <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit, act.fct = h2)
plot(tr, cex=2, main="ReLU activation function h2")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_h2,te), col="red", cex=1)
# the fit is also bad

#### softplus activation function h3
nn_h3 <- neuralnet(Sin ~.,data = tr, hidden = 10, startweights = winit, act.fct = h3)
plot(tr, cex=2, main="softplus activation function h3")
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_h3,te), col="red", cex=1)
# the softplus activation function fits quite well

# 3
set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata <- data.frame(Var, Sin=sin(Var))
plot(mydata, col = "blue", cex=1, ylim=c(-11,1))
points(mydata[,1],predict(nn,mydata), col="red", cex=1)
#plot(mydata[,1],predict(nn,mydata), col="red", cex=1)
# fits well in the [0,10] range, which is trained data range, fit badly outside this range.

# 4
nn$weights

# 5
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
winit <- runif(10001, min=-1, max=1)
nn_2 <- neuralnet(Var ~.,data = mydata, hidden = 10, startweights = winit, threshold=0.1)
plot(mydata, col = "blue", cex=1)
points(predict(nn_2,mydata),mydata[,2], col="red", cex=1)
```