# Lab 3 TDDE01

Daniel Kouznetsov (danko376), Liv Kåreborn (livka967), Erik Johansson (erijo073)

2022-12-18

## Contributions

Assignment 1: Liv

Assignment 2: Erik

Assignment 3: Daniel

## Assignment 1 - Kernel Methods

## Assignment 1

The assignment consists of implementing a kernel method to predict the hourly temperatures for a date and place in Sweden. It will be solved using the r library "geosphere".

**Read data and initialize variables and vectors.**

```
stations = read.csv("stations.csv", fileEncoding="latin1")
temps = read.csv("temps50k.csv")
st = merge(stations,temps,by="station_number")
st$measurement_height=c()
st$elevation=c()
st$quality=c()

h_distance = 30  # distance more than 5000km
h_date = 15 # more than 10 days difference
h_time = 2 # more than 2 hours difference

# point to predict
a = 58.4274 # latitud
b = 14.826 #longitud

# the date to predict
date = "2013-08-04"
# remove all observations after date to predict
st.new <- st[which(st$date < date),]
# create a vector with all relevant time stamps
times = c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "11:00:00", "12:00:00", "13:00:00", "14:00:00
temp = vector(length=length(times))
```

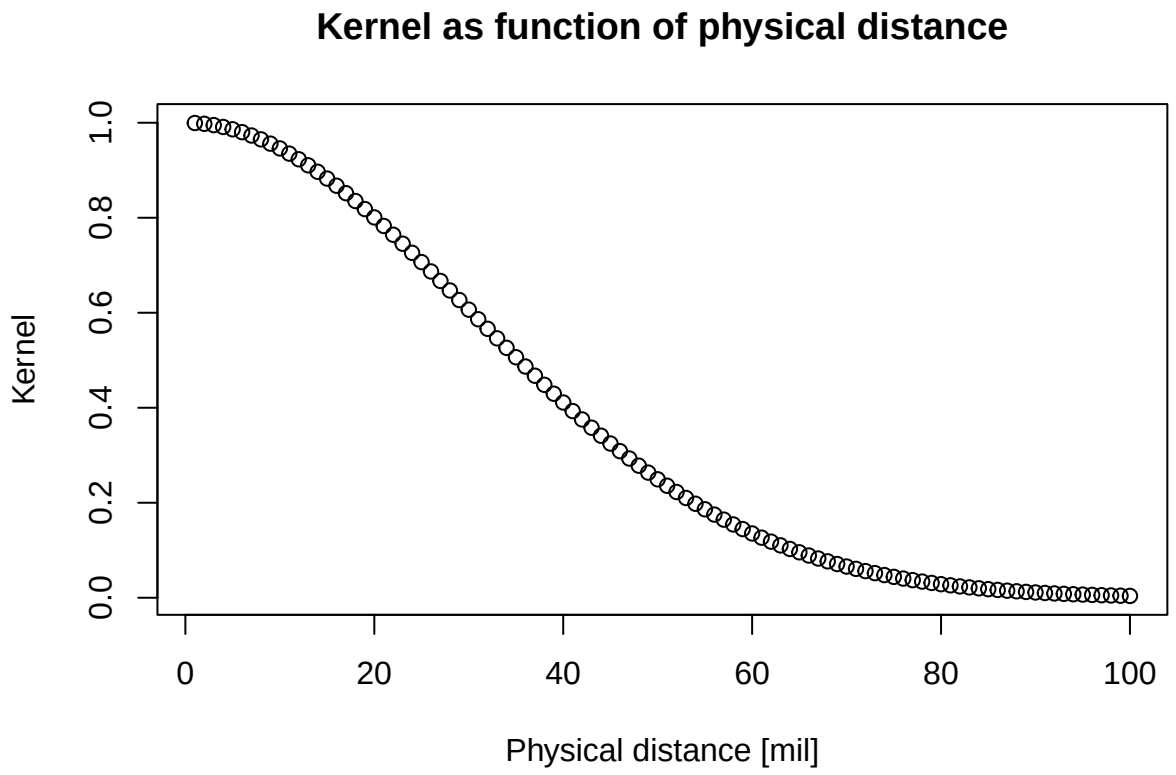
**Implement a function that generates a kernal from distance and smoothing factor.**

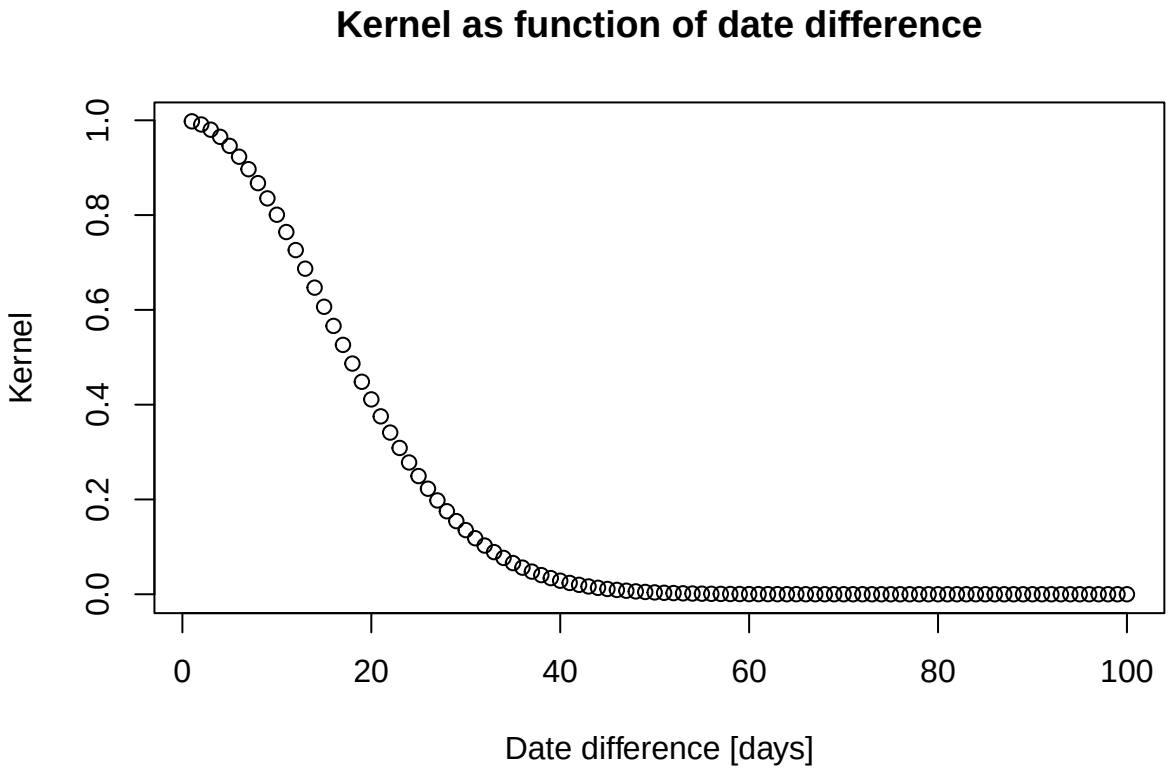
```
# function that takes a vector with all distance values and the smoothing coefficient as input and retu
getKernal = function(dist, h) {
  k = exp(-(dist^2)/(2*h^2))
  return(k)
}
```

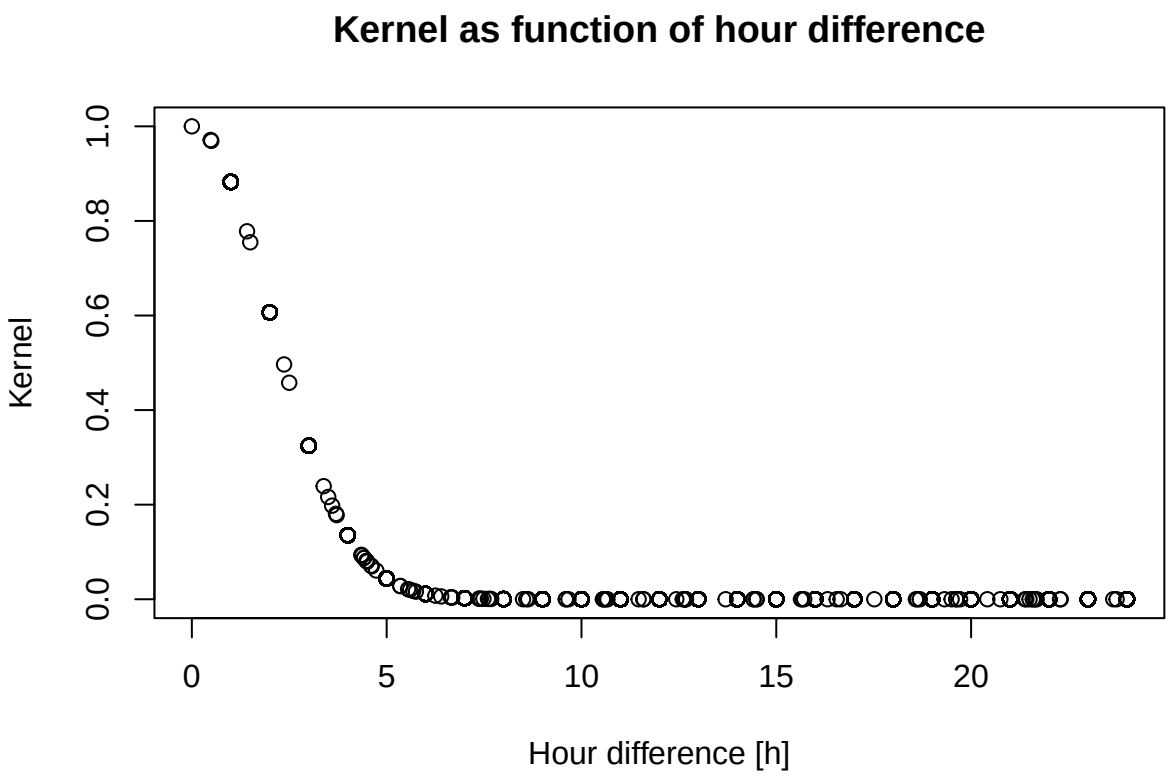
**Implement functions to calculate the distances, physical, date and hour. Then calculate their corresponding kernel.**

```
# function calculating psysical distance
getPhysicalDist = function(data) {
  dist = vector(length=nrow(data))
    for (i in 2:nrow(data)-1) {
      # dist[i] = distHaversine(c(data[i,]$longitude, data[i,]$latitude),c(a,b), r=637.8137)
      dist[i] = distHaversine(c(data[i,]$latitude, data[i,]$longitude),c(a,b), r=637.8137)
    }
  return(dist)
}

# function calculating date differences and returns a vector with all differences in days
getDateDist = function(data, d) {
  dist = vector(length=nrow(data))
    for (i in 2:nrow(data)-1) {
      dist[i] = abs(as.numeric(as.Date(data[i,]$date))-as.numeric(as.Date(d)))
      # look at similar dates previous years
      while(dist[i]>365) {
        dist[i] = dist[i] - 365
      }
    }
  return(dist)
}

# # function calculating hour differences and return vector containing the all hour differences.
getHourDist = function(data, t) {
  dist = vector(length=nrow(data))
  for (i in 2:nrow(data)-1) {
      dist[i] = abs(difftime(strptime(data[i,]$time, format = "%H:%M:%S"),
                        strptime(t, format = "%H:%M:%S"), units = "hours"))
      # look at similar hours previous days
      while(dist[i] > 24) {
        dist[i] = dist[i] - 24
      }
  }
  return(dist)
}


# check physical distance kernel
dist.dist.test = 1:100 # alternative for checking kernel
dist.k.test = getKernal(dist.dist.test, h_distance)

# check date difference kernel
```

```
date.dist.test = 1:100 # alternative for checking kernel
date.k.test = getKernal(date.dist.test, h_date)

#get distances
dist.dist = getPhysicalDist(st.new)
date.dist = getDateDist(st.new, date)
hour.dist = getHourDist(st.new, "24:00:00")

# get kernels
hour.k = getKernal(hour.dist, h_time)
date.k = getKernal(date.dist, h_date)
dist.k = getKernal(dist.dist, h_distance)
```




Kernel as function of date difference
Kernel
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Date difference [days]



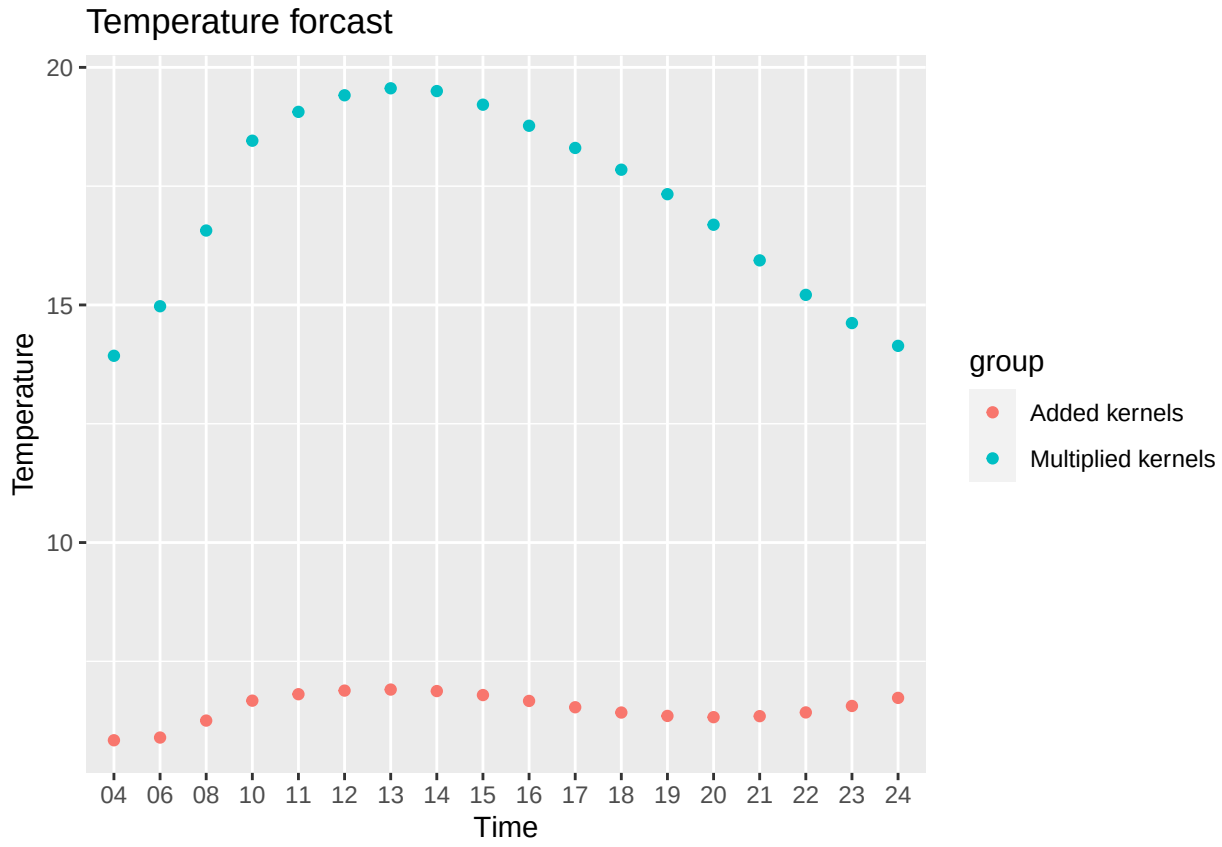

From the plots shown above one can observe that all kernels follow the Gaussian structure and only take relevant data into consideration. Meaning that data outside the span specified by the smoothing factor are off less importance for predictions.

The smoothing factors chosen are: - h_distance = 30 # distance more than 5000km - h_date = 15 # more than 10 days difference - h_time = 2 # more than 2 hours difference.

**Implement a function that predicts the temperature in two ways, by adding the kernels and by multipling the kernels.**

```
# function to get forecast prediction for all relevant times
getForcast = function(k.dist, k.date, c, times) {
  temp = vector(length=length(times))
  i = 1
  # loop over all relevant times
  for(t in times) {
    # update hour kernel
    hour.dist = getHourDist(st.new, t)
    k.hour = getKernal(hour.dist, h_time)
    # update kernel for new time
    if(c=='+') {
      k = k.dist+k.date+k.hour
    }
    else if (c=='*') {
      k = as.matrix(k.dist)*as.matrix(k.date)*as.matrix(k.hour)
    }
```

```
    else {
      return(3)
    }
    # make prediction
    y.hat = (t(as.matrix(k))%*%as.matrix(st.new$air_temperature))/sum(k)
    temp[i] = y.hat
    i=i+1
  }
  return(temp)
}
# forecast by summing up the kernels
forcast.add = getForcast(dist.k,date.k, '+', times)
# forecast by multiplying the kernels
forcast.mult = getForcast(dist.k,date.k, '*', times)
```




From the above plots one can observe the temperature forecasts for the chosen date, 2013-08-04. The plots show one prediction where all the kernels have been summed up, and one where the kernels have been multiplied.

From the plots it's clear to see that the prediction corresponding to summing up the kernels seem a bit off, since it feels implausible that the temperature would start to increase in the evening. However the curve for the rest of the ay seems reasonable with the exception that the temperature might be a bit low for a day in the beginning of august.

The prediction corresponding to multiplying the kernels seems more reasonable It follows a curve where the temperature gets larger towards the middle of the day, to then decrease towards the evening and night. The temperatures also seem more reasonable for a day in august.

## Assignment 2 - Support Vector Machines

```
# Lab 3 block 1 of 732A99/TDDE01/732A68 Machine Learning
# Author: jose.m.pena@liu.se
# Made for teaching purposes

library(kernlab)
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
```

```
## [1] 0.0675
```

```
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
```

```
## [1] 0.08489388
```

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

```
## [1] 0.082397
```

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

```
## [1] 0.02122347
```

```
# Questions

# 1. Which filter do we return to the user ? filter0, filter1, filter2 or filter3? Why?

# 2. What is the estimate of the generalization error of the filter returned to the user? err0, err1, e

# 3. Implementation of SVM predictions.
```

### 2.1

It is filter1 that should be returned to the user as it is the one which uses the correct data for the model and correct data for the prediction. Filter0 predicts on validation data, filter2 makes the model on training and validation data and filter3 models on the entire data set of spam.

### 2.2

The error returned to the user is err3 as it is the model with the lowest error.

### 2.3

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

support_vector <- spam[sv,-58]
f<-rbfdot(0.05)

k<-NULL
d<-NULL
l<-NULL
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  k2<-NULL
  for(j in 1:length(sv)){
    k2<- unlist(support_vector[j,])# Your code here
    n <- unlist(spam[i,-58])
    s<- f(n,k2)
    k<-c(k, s)
  }
  a<- 1 + length(sv)*(i-1)
  z<- length(sv)*i
  d<- co %*% k[a:z]
  l<- c(l, d + inte)
}
l
```

```
##  [1] -1.998999  1.560584  1.000278 -1.756815 -2.669577  1.291312 -1.068444
##  [8] -1.312493  1.000184 -2.208639
```

```
predict(filter3,spam[1:10,-58], type = "decision")
```

```
##           [,1]
##  [1,] -1.998999
##  [2,]  1.560584
```

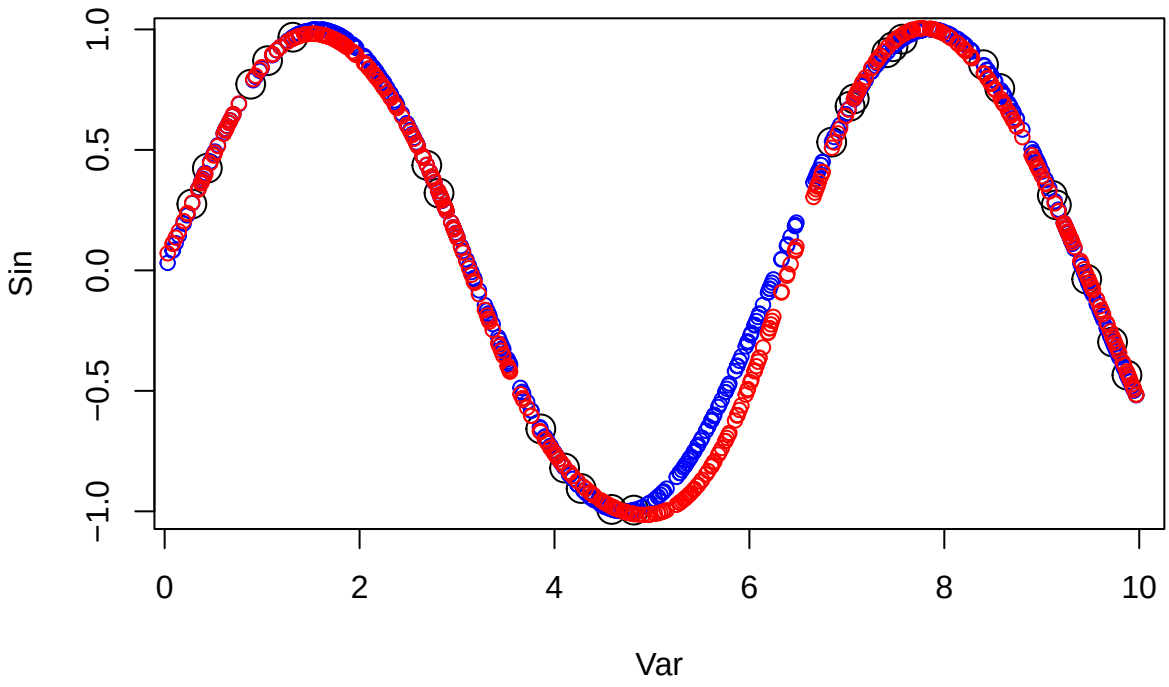
```
##  [3,]  1.000278
##  [4,] -1.756815
##  [5,] -2.669577
##  [6,]  1.291312
##  [7,] -1.068444
##  [8,] -1.312493
##  [9,]  1.000184
## [10,] -2.208639
```

Comparing the above prediction using only the first 10 points in the dataset and the prediction done using the predict function it shows that only using the first 10 points can yield the same predictions as a more computationally costly function.

# Assignment 3 - Neural Networks

## 3.1

```
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

nn <- neuralnet(tr$Sin ~., tr, hidden=c(10))

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2); points(te, col = "blue", cex=1); points(te[,1],predict(nn,te), col="red", cex=1)
```

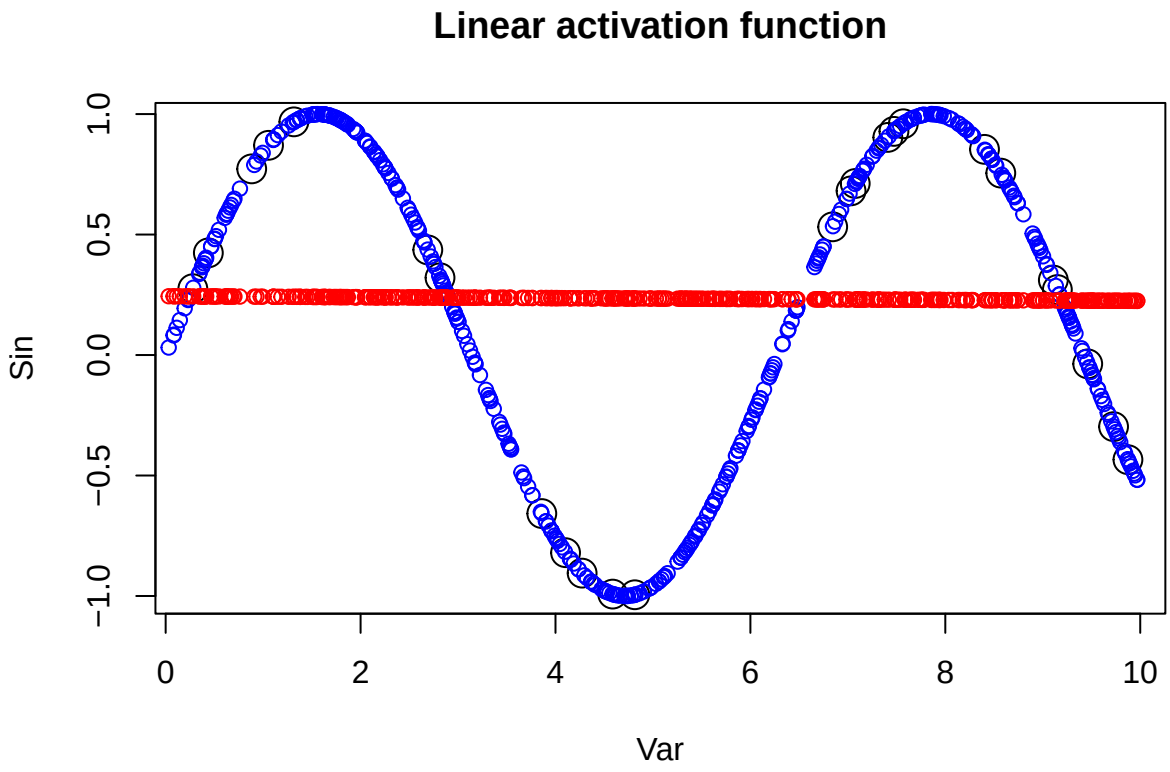



We get a good result. For the Var interval between 5 to 6 we get not so good results, this is because our training data does not have any points here which gives us a worse result for this span.

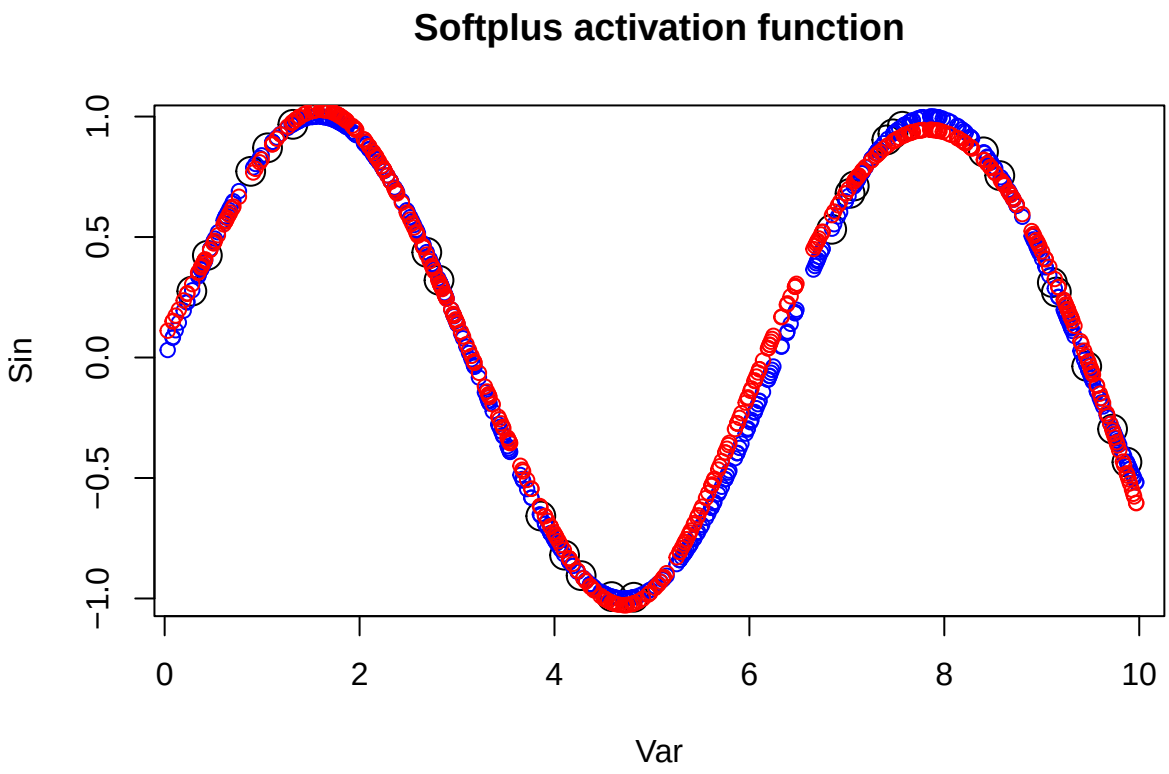
## 3.2

```
# h1(x) = x
linear <- function(x) x
nn <- neuralnet(tr$Sin ~., tr, hidden=c(10), act.fct = linear)

# Plot of the training data (black), test data (blue), and predictions (red)
```

```
plot(tr, cex=2, main="Linear activation function"); points(te, col = "blue", cex=1); points(te[,1],predi
```




```
# h2(x) = max{0, x}
#relu <- function(x) {x * (x > 0)}
#nn <- neuralnet(tr$Sin ~., tr, hidden=c(10), act.fct = relu)

# Plot of the training data (black), test data (blue), and predictions (red)
#plot(tr, cex=2); points(te, col = "blue", cex=1); points(te[,1],predict(nn,te), col="red", cex=1)

# h3(x) = ln(1 + exp(x))
softplus <- function(x) log(1 + exp(x))
nn <- neuralnet(tr$Sin ~., tr, hidden=c(10), act.fct = softplus)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2, main="Softplus activation function"); points(te, col = "blue", cex=1); points(te[,1],pr
```



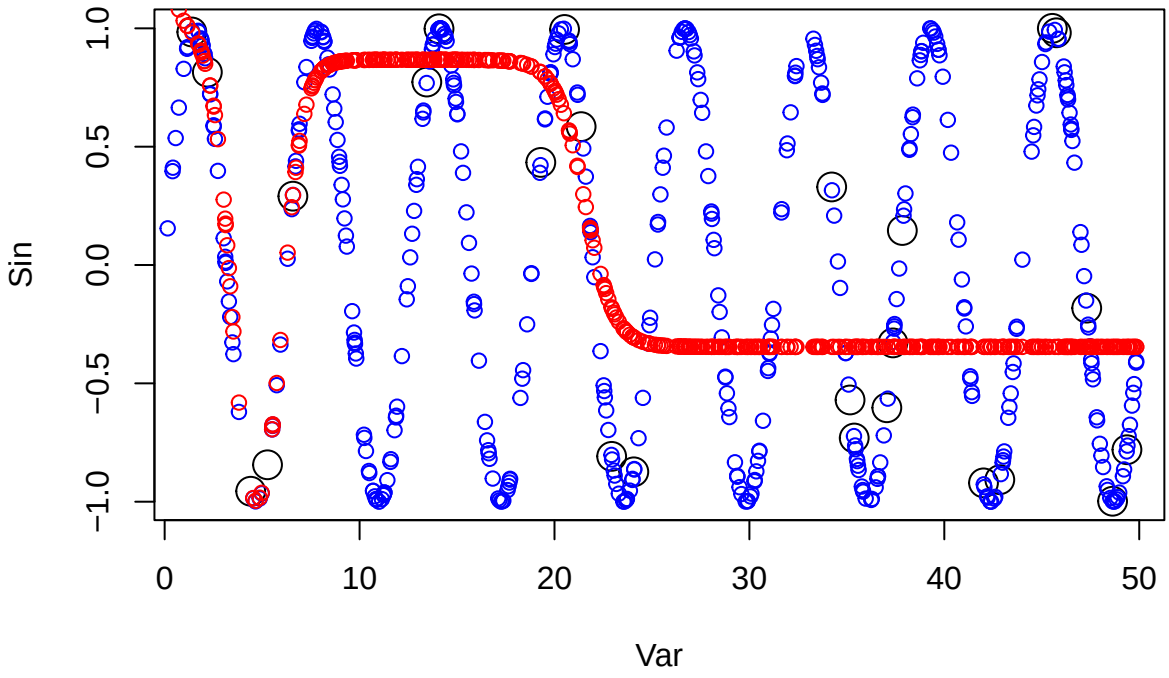

With the linear activation function the results are terrible. This is because this function does not allow for non-linear behavior, i.e., problems such as approximating a sine function. The output from a linear activation function is always a straight line.

The RelU activation function does not work as the NN-package also calculates the derivative of the function. RelU is non-differentiable at 0 therefore the function does not work.

The third activation function, the softplus function, is working well, especially for the first few numbers up to 5 then it does not predict the data quite as well but still pretty good.

### 3.3

```
set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test

nn <- neuralnet(tr$Sin ~., tr, hidden=c(10))

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2); points(te, col = "blue", cex=1); points(te[,1],predict(nn,te), col="red", cex=1)
```



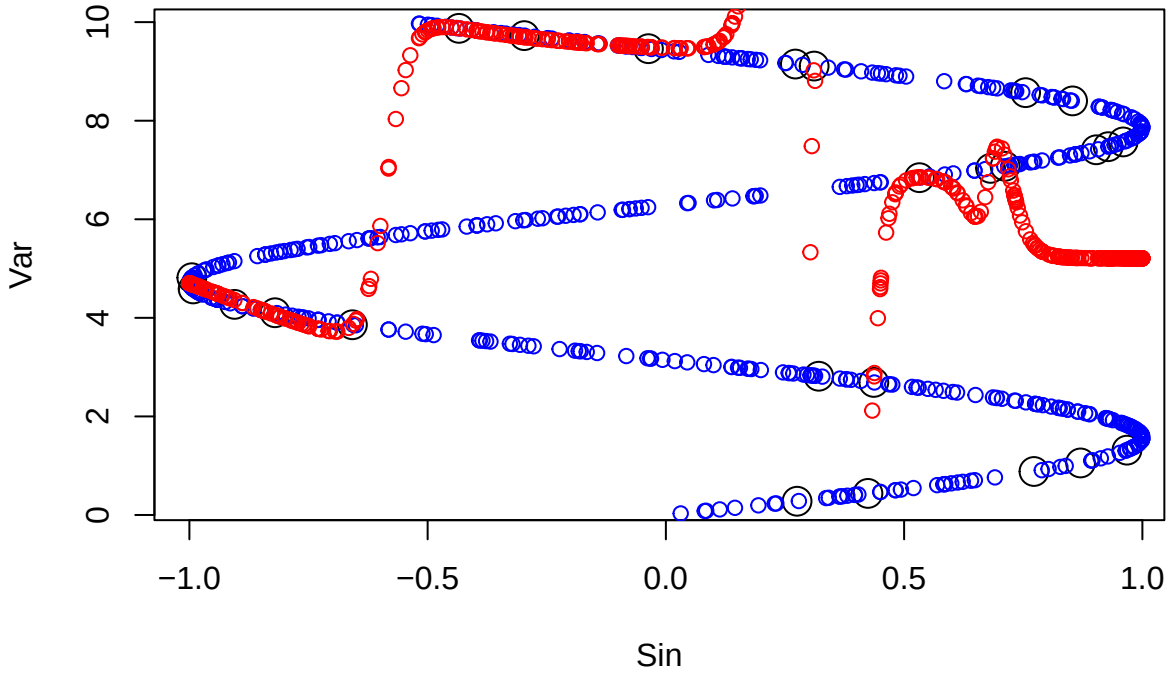

When the interval increases from [0, 10] to [0, 50] the NN is unable to generalize its understanding of the sine curve for this larger interval. That's why we get a wacky result that does not resemble a sine curve for our predicted line (the red line).

### 3.4

The predictions converge because the neural network adjusts the weights to minimize the error of its predictions. As the network adjusts the weights it will eventually reach a point where the error is minimized and the predictions converge to some value. In our case the value it converges towards is approximately -0.3.

### 3.5

```
set.seed(1234567890)
Var <- runif(500, 0, 10)
Sin <- sin(Var)
mydata <- data.frame(Sin, Var)
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test

nn <- neuralnet(tr$Var ~., tr, hidden=c(10))

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2); points(te, col = "blue", cex=1); points(te[,1],predict(nn,te), col="red", cex=1)
```




We get a bad result which is not weird because we get several y-values for the same x-value, then it's not easy to decide what x-value corresponds to which y-value since it has more than one answer.