TDDE01. Lab3. Group B24 report.

Statement of contribution

Firstly, general analysis was performed teamwise. Approaches and strategies of solving the tasks were elaborated teamwise as well.

Student Elena was responsible for code writing and problem solving for the Assignment 1. Student Elham was responsible for code writing and problem solving for the Assignment 2. Student Anton was responsible for code writing and problem solving for the Assignment 3.

After completion of coding stage group analyzed the results together and peer reviewed the results of each other's work. Finally, each student corrected their solutions according to the received reviews from groupmates.

Assignment 1. 1. KERNEL METHODS.

We were provided with the raw data regarding weather stations and temperature measurements in the stations in Sweden at different days and times. We were 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.

First, three Gaussian kernels were implemented for three different types of distances:

- Geographical distance between weather stations in the given data and the point of interest
- Distance in days between dates in the given data and date of interest (relative distance in days on a distance within one year)
- Distance in hours between hours in the given data and hour of interest (in our case, a set of hours from 4 am to 24 pm with the interval of 2 hours)

The following formula for the Gaussian kernel was used:

$$\text{Gaussian kernel: } k(u) = exp(-||u||^2)$$

Second, we compared different smoothing factors and chose the optimal ones so the kernels would have the largest values at the points close to the point of interest and small values at the distant points.

a) Choosing h_distance=100 000 smoothing factor for the physical distance:

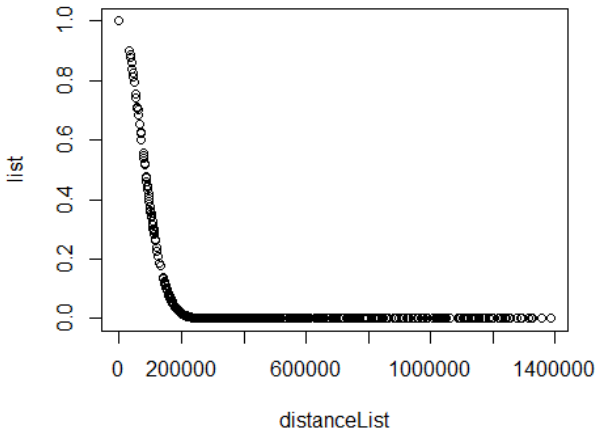


Figure 1. Geographical smoothing coefficient.

b) Choosing h_date=10 smoothing factor for the date distance:

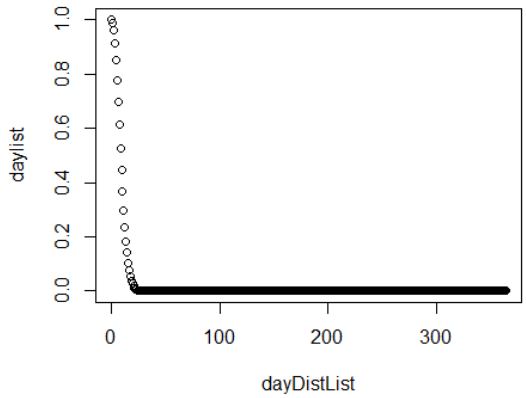


Figure 2. Days smoothing coefficient.

c) Choosing h_time=2 smoothing factor for the hour distance:

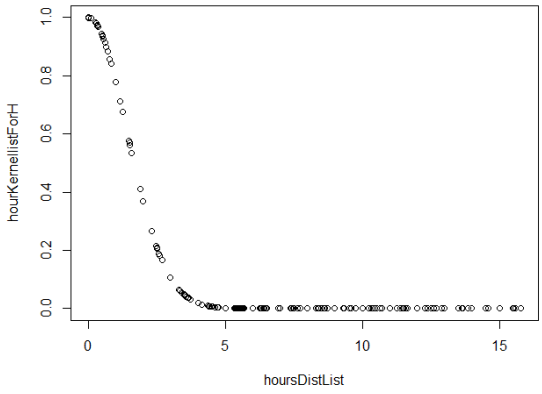


Figure 3. Hours smoothing coefficient.

Then, the forecast was computed. The dates and hours posterior to the point of interest were discarded before the forecast computation.

There were types of forecasts generated for Stockholm (lat= 59.335, long= 18.063) on August 2013:

- First based on the sum three Gaussian kernels
- Second based on the multiplication of three Gaussian kernels

The results are given on the following plot:

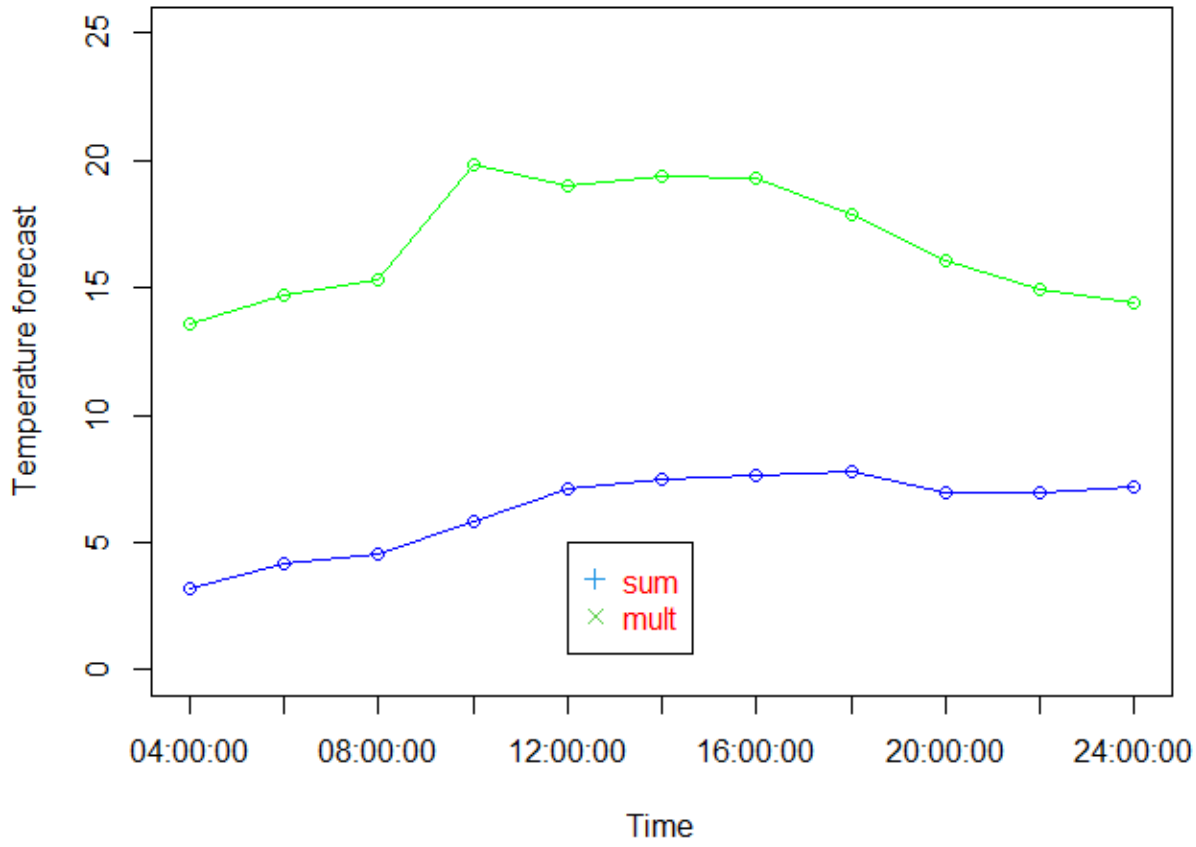


Figure 4. Predictions of Stockholm weather for the year of 2013.

As it can be seen from the plot, the result of the prediction through the multiplication of the kernels is very realistic. As a reference point any weather web-site could be used (for example https://www.weather-atlas.com/en/sweden/stockholm-weather-august):

Weather in August

The last month of the summer, August, is also an agreeable month in Stockholm, Sweden, with an average temperature fluctuating between 12.7°C (54.9°F) and 20.4°C (68.7°F).

But the results of the summation of the kernels is far from good, in fact very unplausible.

This is happening because the sum of the kernels isn't an optimal choice for this task as it is unable to discard those kernels that shouldn't be taken into account. For example, this kernel will take into account a very distant point (say, far North), if the date and time are close to those of interest. Values of kernels for day and time will be high, value for kernel of distance very low, but the resulting value of the summation of the kernels will be high. Thus, we risk obtaining irrelevant result.

Quite the opposite behavior can be seen in multiplication kernel. It will not take into account the result for the point if it is too far in any of three measurements, the resulting kernel for this point will be very small and won't affect the accuracy of the prediction.

As a conclusion, we suggest using the multiplication of the kernels for obtaining accurate weather forecasts.

Assignment 2. Support Vector Machines

We used Lab3Block1-2021-SVMs-St.R from lab material that performs SVM to classify the R spam data set by KSVM function. All SVM models use Gaussian kernel function (rbfdot) with the width of 0.05 and the C parameter is different in each model.

Four different filters were constructed with the C parameter which gives the lowest MCR.

Task1: Which filter do you return to the user? filter0, filter1, filter2 or filter3? Why?

All filters are same except in data that they use in KSVM. You can see their differences in table below.

	Filter 0	**Filter 1**	**Filter 2**	**Filter 3**
Data set in KSVM	Data= Train	Data= Train	Data= Train+ Valid	Data= Spam
Obs of 58 variables	3000	3000	3800	4601
Date set for Predict	Data= valid	Data= Test	Data= Test	Data= Test
Error	0.0675	0.08489388	0.082397	0.02122347
Accuracy	93.25 %	91.51061 %	91.7603 %	97.87765 %

Filter 0: Has smaller data sets than filter 2 and 3 and it does not use test data set for prediction.

Filter1: Such as filter 0, this filter also has smaller data sets than filter 2 and 3, but did not use the validation data set at all.

Filter2: We prefer return this filter to user because it has the largest usage of available data while still keeping some separate for testing.

Filter3: In this filter training dataset contained test dataset and it cannot be a good filter.

Task2: What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why? err2

To answer this question, we considered unseen data used, unseen data for C (reg term) and size of data set. At first step we reject err0, because both training and validation datasets were seen before. We reject err3 because it uses test data set in training.

Between err1 or err 2, err1 is obtained on a smaller dataset (tr) and is a bit bigger than err2 while both of them use unseen data (which is test), so we prefer err2.

Task3: Implementation of SVM predictions:

In this task we are asked to implement the linear combination for filter 3. We used given code to get indexes of support vectors, linear coefficients for support vectors and the negative intercept. We produce prediction for the first 10 point of data set by a nested loop and compare it with the result of function predict where the type is "decision". As you can see the prediction results are identical.

	k	pred
1	-1	-1.998999
2	1	1.560584
3	1	1.000278
4	-1	-1.756815
5	-1	-2.669577
6	1	1.291312
7	-1	-1.068444
8	-1	-1.312493
9	1	1.000184
10	-1	-2.208639

We used those formulae to calculate our prediction while k is the RBF kernel:

$$f(x) = \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

$$\widehat{y}(\mathbf{x}_\star) = \text{sign}\left(\widehat{\alpha}^\mathsf{T} \boldsymbol{K}(\mathbf{X}, \mathbf{x}_\star)\right) \qquad (8.35c)$$

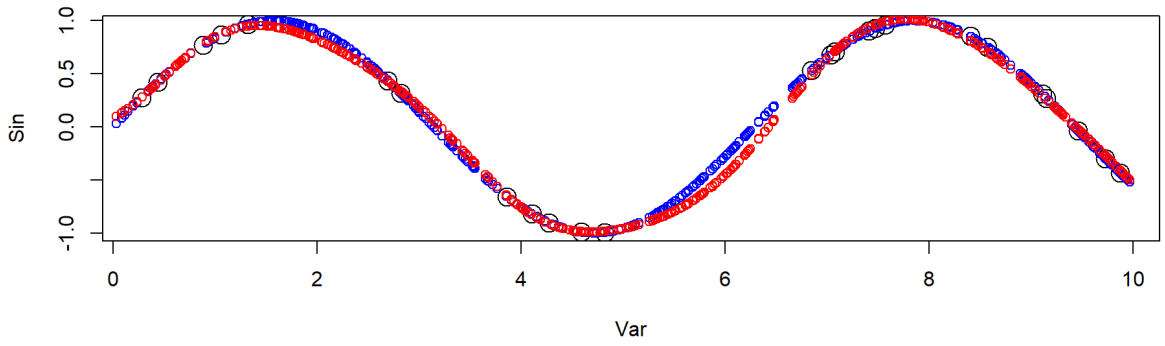
Assignment 3 NEURAL NETWORKS

Task 1.

A Neural Network (NN) model was trained to learn the sine function (sin(x)). The data set consisted of 500 samples with two columns, 'var' and 'sin'. Var is the value of x which is a value in a range from 1 to 10 and sin is sin(x), sin of x.

25 samples were used for training and 475 for test. Starting weights were in the range -1 to 1 and random.

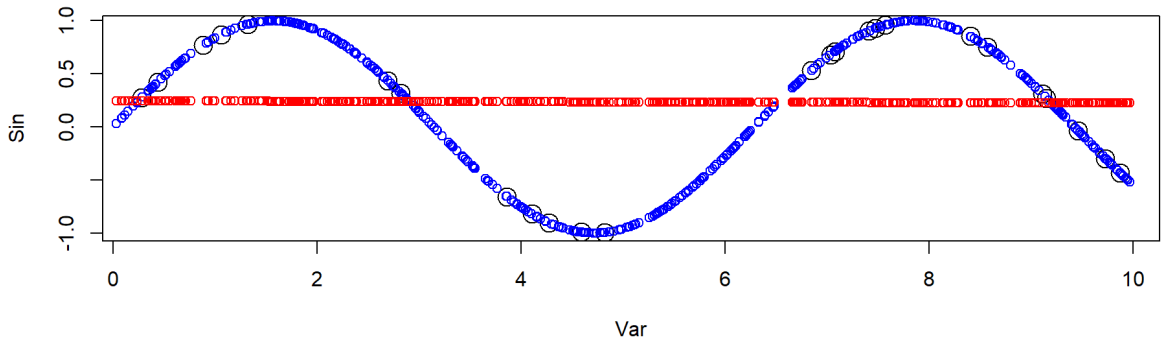
Plot of the training data (black), the test data (blue) and the predictions (red) of the NN model.



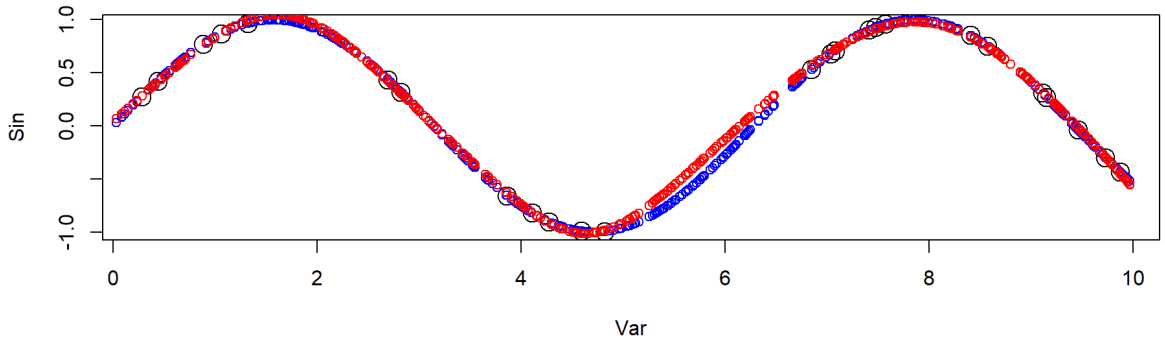
The results look good. But as seen in the plot there are some error, the prediction doesn't follow the data perfectly. But very well.

Task 2.

Three new neural networks were trained, with different custom activation functions. One linear (h1=x), one ReLU (h2=max{0,x}) and one softplus (h3=ln(1+exp(x))). The previous task used the default sigmoid function, also known as the "logistic". Beyond the activation function, nothing was changed from task 1.



Linear activation function. As seen, it gives a straight line. Also, it can't approximate a non-linear function.



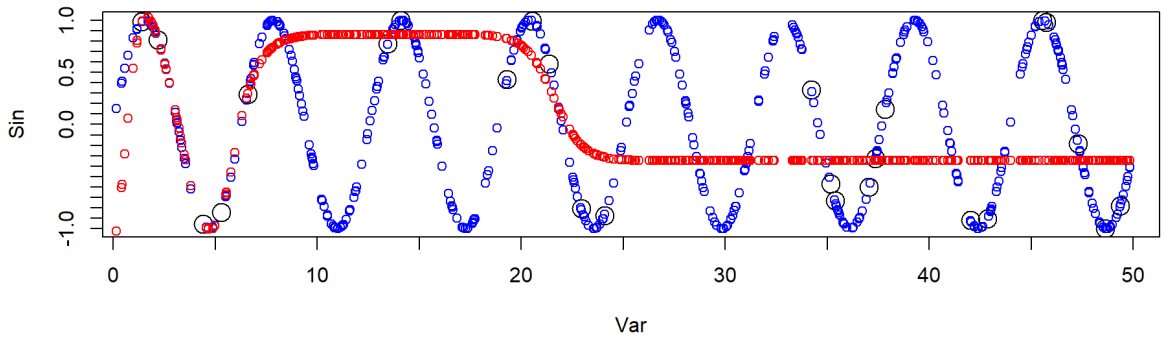
Softplus activation function. Fits the best, because to the difference of a linear activation function it considers unknown variables.

ReLU was not implementable in the same way as the previous two custom activation functions. Because {0,x} is not a differentiable function, it cannot be derived at zero. In other words the left side and right side of the derivative is not identical and this is not something rstudio could handle. There are workarounds, however we were not tasked with implementing the workarounds.

Task 3.

Same steps as in task 1, however the range for x was increased to 0 to 50.

Black is training, blue is test and red is prediction.



The model looks good up to x=10, after that it starts to look worse, so mixed results. Then it seems to converge to some value.

Task 4.

The prediction in the previous plot seems to converge to some value. NN and NN$weights were looked at to explain why.

The activation function for the NN was the default sigmoid function, 1/(1 + exp(-x))=$\frac{1}{1+\frac{1}{e^x}}$.

Looking at the graph the value being converged to is approximately somewhere between -0.3 and -0.4 or -0.5.

The prediction for a large x value, 50, was calculated with the bias, so a vector of 1 and 50. The activation function was used on the bias and x*weights which looked like this 1/(1 + exp(-x*weights)). The result was a vector of 1s and almost zeroes, first position in the vector corresponds to bias and the

rest to each hidden node in turn. Based on this result it is possible to see which hidden units were used in the output, those that had the value of 1. Those with value of zero were not used. Those that had a value of 1 were the bias and input variable 4, 6, 7, 9 and 10.

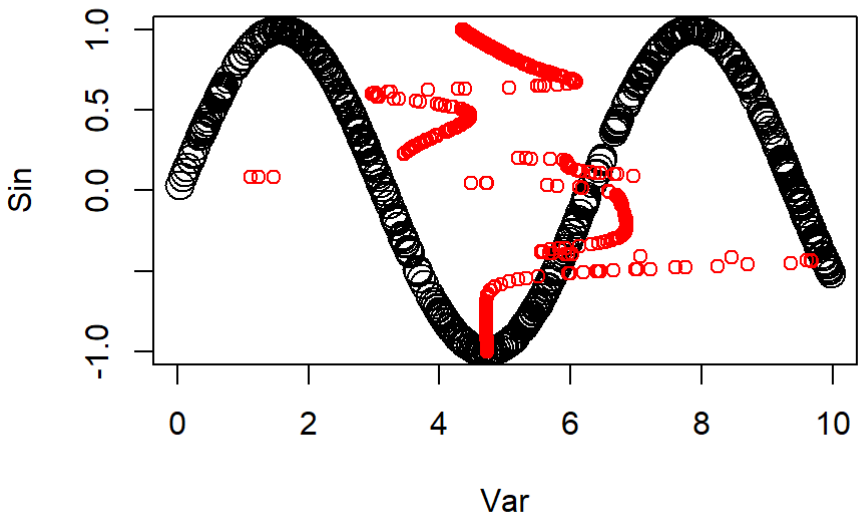
Summing up these values from nn$weights, or multiplying the previously mentioned vector with the vector for nn$weights [[1]][[2]] we got the value the models prediction is converging towards, -0.3474075. nn$weights [[1]][[2]] is the values of weights for the output layer in the NN.

This happens because as x grows the sigmoid function approaches 1 or 0 depending on if x is positive or negative, positive x goes towards 1 and negative x towards 0.

Task 5.

Same as task 1 but with a twist. So, the range for x is 1 to 10 again. Instead of trying to predict sin(x) based on x, it is the other way around, predicting x based on sin(x).

Black is training data, test is blue and predictions red.



Result is bad, prediction doesn't match the data. Because the model is trying to get a linear prediction, x, from a non-linear function, sin(x). For example for sin(0) there can be 4 different values for x in the range [1,10] (zero, pi, 2*pi and 3*pi for the curious). However, the model can't see that.

Appendix. Code.

Assignment 1.

```
##################Assignment 1#########################333
```

```
#install.packages("geosphere")
library(geosphere)
#install.packages("dplyr")
library(dplyr)
#################preparations#########################
set.seed(1234567890)
library(geosphere)
stations <- read.csv("stationsutf8.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
h_distance <- 100000
h_date <- 10
h_time <-2

##The point to predict (up to the students)- Stockholm
# a=latitude  b= longitude
a <- 59.335
b <- 18.063
# date
date <- "2013-08-04" # The date to predict (up to the students)
# times of interest
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")
# Students' code here



###################function to calculate Gaussian kernel for distance ############
#########takes a row from st table as an input
StarStation= c(a,b)
GKerneldistance <- function(station2) {
  coordinatesStation2=c(station2$latitude,station2$longitude)
  distance <- distHaversine(StarStation,coordinatesStation2)
  result = exp(- (distance/h_distance)^2 )
```

```
  return(result)
}

############picking nice smoothing factor h_distance###########
distanceList=c()
list<-c()
for(i in 1 : nrow(st)){
  distanceList <- append(distanceList,distHaversine(StarStation,c(st[i,]$latitude,st[i,]$longitude)))
  list[i]= GKerneldistance(st[i,])
}
#plotting the dependency of distance kernels values on physical distance
plot(x=distanceList,y=list)

###################function for getting vector of distance kernels for given data########
##returns a vector
filldist <-function(data)
{
  list=c()
  for(i in 1 : nrow(data)){
    list[i]= GKerneldistance(data[i,])
  }
  return(list)
}

#########################################################Gaussian kernel for Day
#################
StarDay=as.Date(date)
GKernelDay <- function(Day2) {
  DateDay2=as.Date(Day2)
  daysdiff=((as.numeric(difftime(DateDay2,StarDay, units = "days")))%%365)
  result = exp((- (daysdiff)^2)/(h_date)^2)
  return(result)
}
```

```
############picking nice smoothing factor h_day###########
dayDistList=c()
daylist<-c()
for(i in 1 : nrow(st)){
  dayDistList <- append(dayDistList,(as.numeric(difftime(as.Date(st[i,]$date),StarDay, units = "days"))%%365))
  daylist<-append(daylist,GKernelDay(st[i,]$date))
}
#plotting the dependency of day kernels values on the distance between days
plot(x=dayDistList,y=daylist)
##########################################


###############function for getting vector of day kernels for given data and day
##returns a vector
fillDay <-function(data)
{
  daylist=c()
  for(i in 1 : nrow(data)){
    daylist[i]= GKernelDay(data[i,]$date)
  }
  return(daylist)
}
##################



############Gaussian kernel for Hours ################
GKernelHours <- function(Hour2,StarHour) {
  first <- as.POSIXct(paste("2022-01-01 ",StarHour))
  second <- as.POSIXct(paste("2022-01-01 ",Hour2))
  result = exp((- (as.numeric( difftime(first, second,units = "hours")))^2)/(h_time)^2)
  return(result)
}
```

```
##################This code to pick smoothing factor for hours#########

############picking nice smoothing factor h_day###########

hoursDistList=c()

hourKernellistForH<-c()

StarHourH=times[3]

StarHourHTime <- as.POSIXct(paste("2022-01-01 ",StarHourH))


for(i in 1 : nrow(st)){

  currentTime <- as.POSIXct(paste("2022-01-01 ",st[i,]$time))

  hoursDistList <- append(hoursDistList,(as.numeric( abs(difftime(currentTime, StarHourHTime,units =
"hours")))))

  hourKernellistForH<-append(hourKernellistForH,GKernelHours(st[i,]$time,StarHourH))

}

#plotting the dependency of day kernels values on the distance between days

plot(x=hoursDistList,y=hourKernellistForH)

########################################


####################Forecast for Sum and Multiply ####################


finalsum= c() #sum of kernels

mult1= list() #multiplication of kernels

forecastsum<-c()

forecastmult<-c()

daylist<-c()


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

  StarHour=times[i]

  #filtering

  stfiltered <- filter(st, date < StarDay & time<StarHour)

  daylist=fillDay(stfiltered)

  distlist=filldist(stfiltered)

  timelist=c(1:nrow(stfiltered))
```

```
 for(k in 1 : nrow(stfiltered)){
  timelist[k]= GKernelHours(stfiltered[k,]$time,StarHour)
 }

 finalsum<-distlist+timelist+daylist
 mult1=distlist*timelist*daylist
 forecastsum[i]=sum(finalsum*stfiltered$air_temperature)/sum(finalsum)
 forecastmult[i]=sum(mult1*stfiltered$air_temperature)/sum(mult1)
}
print(forecastsum)
print(forecastmult)

plot(forecastsum, type="o", xlab = "Time", ylab = "Temperature forecast",
xaxt="n",ylim=c(0,25),col="blue")
axis(1, at=1:length(times), labels = times)
points(forecastmult, type="o", xlab = "Time", xaxt="n",col="green")
legend(5, 5, c("sum", "mult"), col = c(4,3), text.col = "red", pch = c(3, 4))
```

Assignment 2.

```
# 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

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)
err2

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



library(kernlab)

sv<-alphaindex(filter3)[[1]]

co<-coef(filter3)[[1]]

inte<- - b(filter3)

k<-0

rbfkernel <- rbfdot(sigma = 0.05)

kpar(rbfkernel)

for(i in 1:10){

  k2<-0
```

```
  for(j in 1:length(sv)){
    x= as.vector(spam[sv[j],-58])
    names(x)<-NULL
    x<-as.numeric(x)
    y= as.vector(spam[i,-58])
    names(y)<-NULL
    y<-as.numeric(y)
    RBFkernel= rbfkernel(y,x)
    k2 <- k2+co[j]*RBFkernel
  }
  k2<-k2+inte
  sig<-sign(k2)
  k<-c(k,sig)
}
k=k[2:11]
pred=predict(filter3,spam[1:10,-58], type = "decision")
data.frame(k,pred)
```

Assignment 3.

```
library(neuralnet)

#####Task 1#####
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
```

```
# Random initialization of the weights in the interval [-1, 1]
winit <-  runif(25, -1, 1)

# Training model
nn <- neuralnet(Sin~Var, data=tr, hidden=10, startweights=winit)

# 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)

#####Task 2#####
winit <-  runif(25, -1, 1)
linear <- function(x) x
nnLinear <- neuralnet(Sin~Var, data=tr, hidden=10, act.fct = linear, startweights=winit)

winit <-  runif(25, -1, 1)
softplus <- function(x) log(1 + exp(x))
nnSoftPlus <- neuralnet(Sin~Var, data=tr, hidden=10, act.fct = softplus)

#install.packages('sigmoid')
library(sigmoid)

#relu()
winit <-  runif(25, -1, 1)
nnReLu <- neuralnet(Sin~Var, data=tr, hidden=10, act.fct =relu, startweights=winit)

#plotting
#linear
plot(tr, cex=2)
points(te, col = "blue", cex=1)
```

```
points(te[,1],predict(nnLinear,te), col="red", cex=1)

#softplus
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nnSoftPlus,te), col="red", cex=1)

#ReLu
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nnReLu,te), col="red", cex=1)

##### Task 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

# Random initialization of the weights in the interval [-1, 1]
winit <-  runif(25, -1, 1)

nntask3 <- neuralnet(Sin~Var, data=tr, hidden=10, startweights=winit)

# 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(nntask3,te), col="red", cex=1)
library(Hmisc)
minor.tick(ny = 5,  tick.ratio = 1) #line seems to be exactly at -0.347

##### Task 4 #####
```

```
print(nntask3$weights)

# Look at some (large) x value (1 is the bias)
x <- c(1, 50)
print(x)

# Print the hidden units
hidden <- c(1, 1/(1+exp(-x %*% nntask3$weights[[1]][[1]])))
print(hidden)

# Print the prediction
prediction <- hidden %*% nntask3$weights[[1]][[2]]
print(prediction)

# Print the weights
print(nntask3$weights)
#Plot of neural network
plot(nntask3)

# [4,] -1.2142028
#[6,]  0.6218254
#[7,] -1.2120181
#[9,] -1.2250588
#[10,]  2.1300140
summm<- sum(-1.2142028, 0.6218254,-1.2120181,-1.2250588,2.1300140,0.5520328)
#exactly -0.3474075

##### Task 5 #####
#predict x from sin(x)

set.seed(1234567890)
Var <- runif(500, 0, 10)
```

```
mydata <- data.frame(Var, Sin=sin(Var))

# Random initialization of the weights in the interval [-1, 1]
winit <-  runif(25, -1, 1)

#NN that predicts x from sin(x) on the whole dataset (all point are training)
nnreversed <- neuralnet(Var~Sin, data=mydata, hidden=10, startweights=winit,threshold = 0.1)
reversedprediction=predict(nnreversed,mydata)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(mydata, cex=2) #black dots = training data
points(reversedprediction,mydata[,2],col="red", cex=1)
```