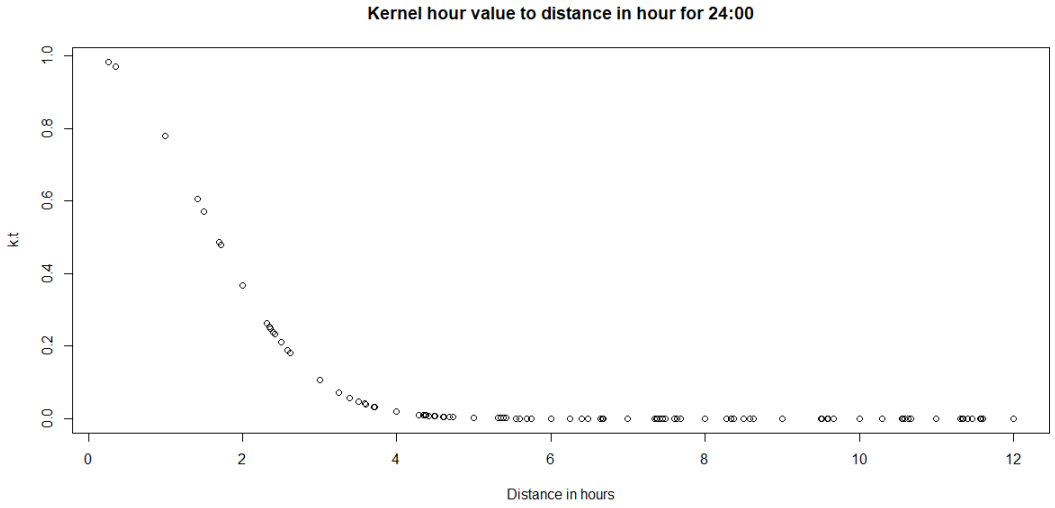
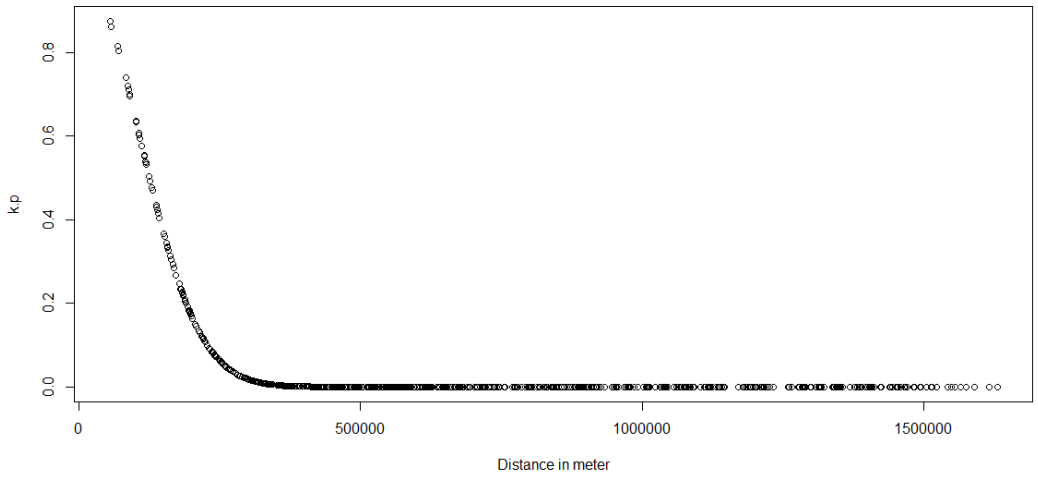
Assignment 1

The physical distance in meters was calculated, as was the distance in days. The h-values were after some reasoning, testing and study of the plots below chosen to be 2 hours for h_time, 10 days for h_date, and 150 000 meters for h_distance. The h-values act as sigma for data inclusion.

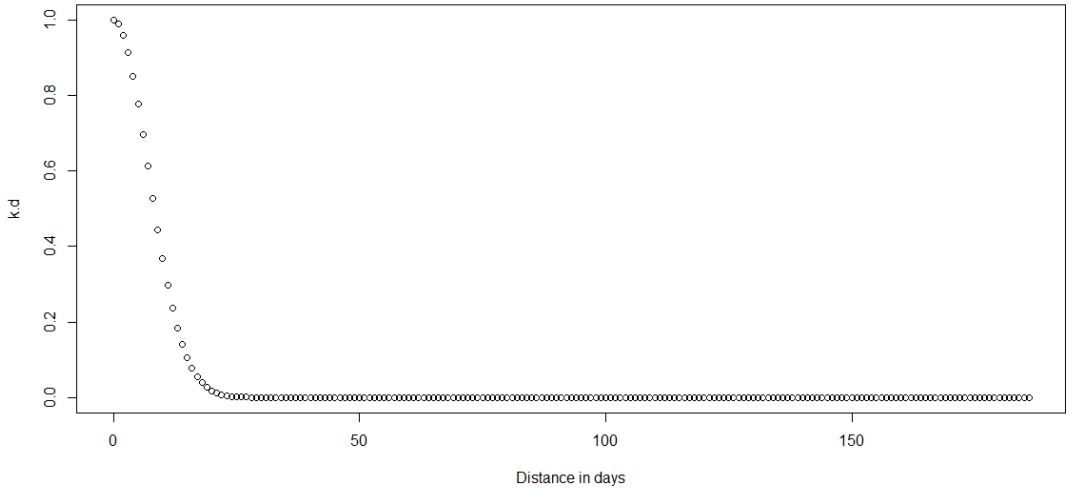
2 hours was chosen for h_time since we are giving predictions for every two hours and therefore do not want the predictions to become too similar, which they would risk if they consider data points past the neighbouring bi-hourly predictions. The plot for kernel distance for hours against distance in hours became the following plot which has the wanted effect of mainly considering points at two hours away (use ~34.1%) and completely rejecting measurements further than four hours away.



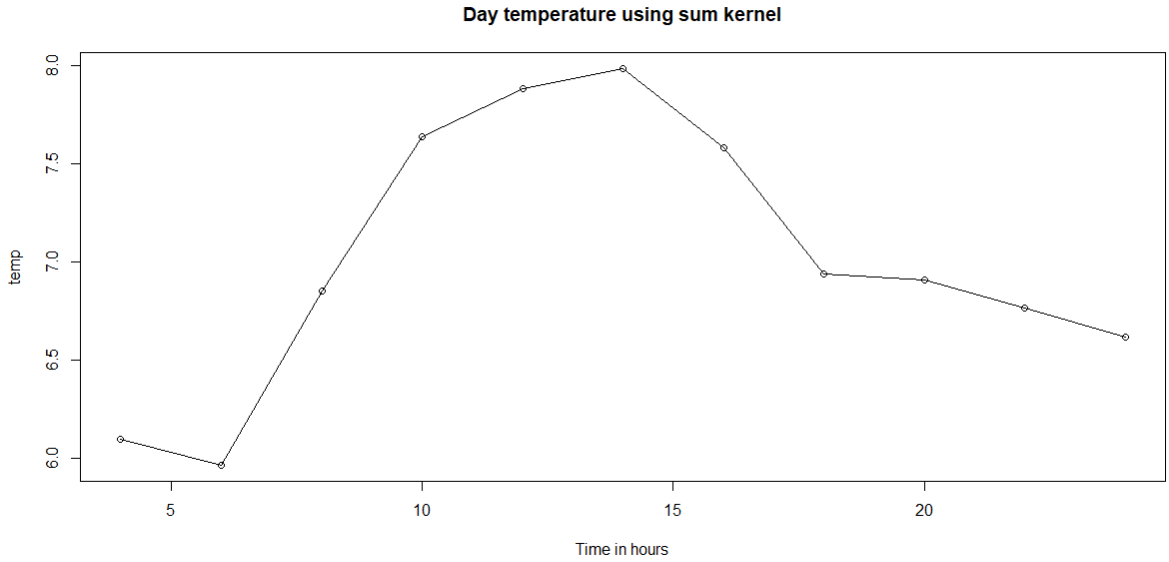
A similar reasoning was made for the physical distance h_distance and distance in days h_date. 150 000 m was chosen to reflect the nature of weather, in that weather often is similar over larger areas. With that said, this means that the predictions will most likely not be able to forecast the regional differences which can occur and instead provides an overview since data of regional differences would be diluted away in the regional “mean value”. In the kernel plot below, we see that 1 standard deviation (34.1%) is achieved at around 150 000 meters, and that data beyond 200 000 meters is accounted for in very little degree.



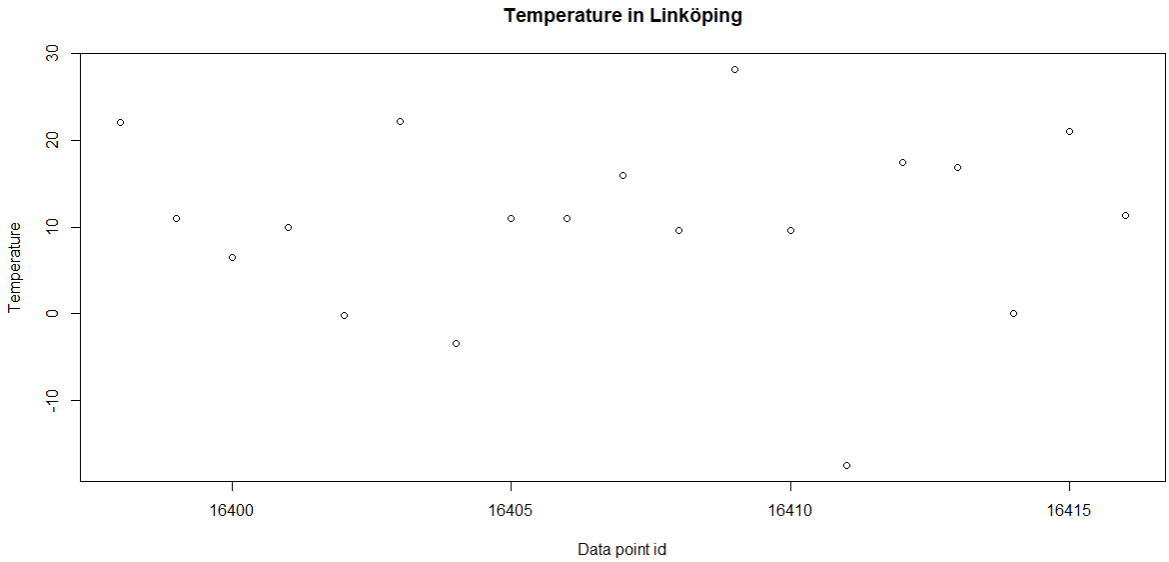
h_time was chosen to be 10 days, since weather often takes time to change. This means that several quick weather changes over a week would result in none of the extreme values but instead a “mean value”. The plots for the kernel values below showed an adequate result, having the wanted inclusion with data in a radius of 10 days accounting for in total 60% of the temperature (34.1% at +10)



Using the summed gaussian kernel: k = k_distance + k_date + k_time, the following result was achieved for the place 58.3836N, 14.826E in august.

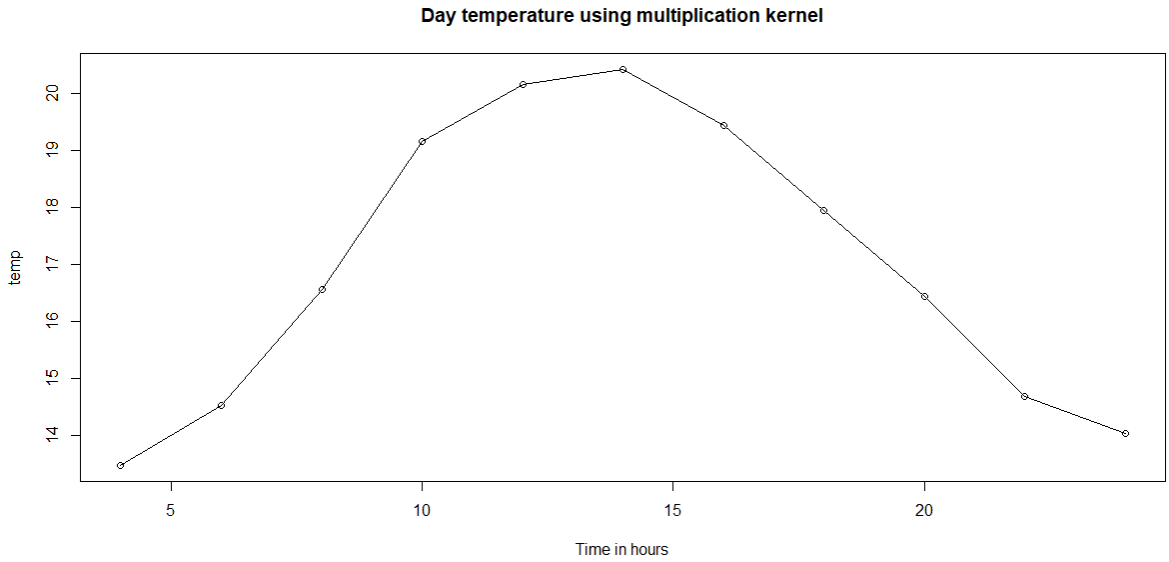


The summed kernel is quite bad when predicting. We can see a shape which seems reasonable for the temperature change over a day, with peaking temperature at noon and colder in the night, but since the place is close to Linköping which probably has warmer weather in the summer it seemed off. This observation was confirmed by plotting the temperature for the weather station in Linköping, where we can see that many of the values were between 10 and 20 degrees and above, assumably summer temperatures. See plot below.



The sum kernel is problematic since it can give high weights to points which are completely irrelevant and instead should have gotten a very small weight. For example, a point which is far off in physical distance will have a distance kernel value close to zero, but the summed kernel can still be large if the date and time is close to the point studied, thus the weight will still become relatively large and the temperature will thus have a significant impact. This becomes very problematic since all data points are used to predict the temperature, only that they are given different weights to have different impact on the prediction. The same happens for date and time as well, where a relatively high weight still will be given to a point which is close in distance and time but is on the opposite side of the year. Winter values will thus affect the prediction for the summer prediction and the other way around.

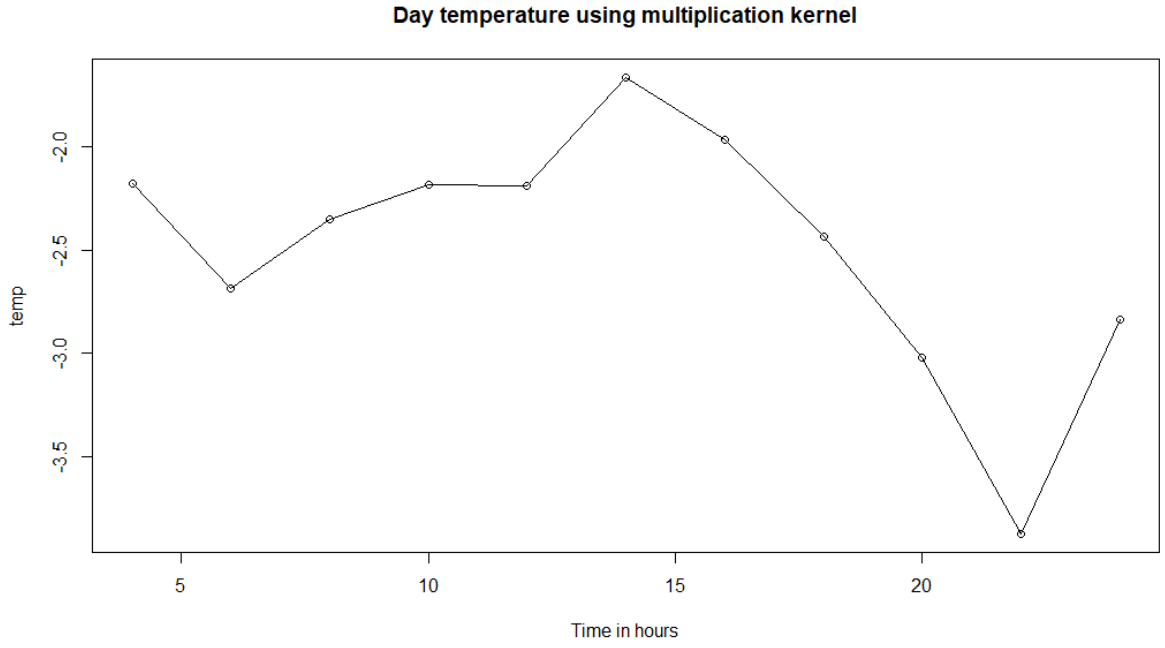
A better method would be to make sure that if one of the parameters, physical distance, date or time is off, then the weight will be very small. This result is possible through using a kernel of multiplied gaussian kernels, since the product is sensitive to changes in the factors: if one factor is close to zero, then the product becomes close to zero, regardless of the values of the two other factors. This means that in order to get a higher weight, then all factors will have to be of a certain relevance.



With the multiplication kernel we now see prediction values which better adhere to the warmer Linköping data, at around 15-20 degrees in summer.

This method still depends on the values of the h:s, and are thus still inheriting the generalizing qualities they provide which were discussed above. Hence the predictions will still be generalizing and not able to predict local variances.

The model was also tested on February winter weather, and a clear difference was obtained with temperatures below zero predicted. See plot below.



This prediction seems to be in line with the Linköping data, even though it has a behaviour in at four in the morning, which is less reasonable, one would imagine that it is cold both early in the morning and in the late night in the general case. One reason might be that there are fewer data points for winter weather, thus making the model more sensitive to outliers. To get around this, the h-values could be changed, but then the model might end up overfitting the specific data and what we need instead are more data points.

Assignment 2

General reasoning before answering the questions

The code first creates several kinds of filters all with different values of C, to identify which value of C results in the smallest misclassification error for predictions with the validation data. In this process the training data was used, and the model was tested on the validation data.

After this the four different filters, filter0, filter1, filter2 and filter3 are produced, using the C which minimizes the validation misclassification error identified above. The following data was used for each model:

	filter0	**filter1**	**filter2**	**filter3**
In model	Training	Training	Train and validation	All data
In predict	Validation	Testing	Testing	Testing

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

The best hyperparameter was calculated in the first step ("inner loop" (but no cross validation)) and is which.min(err_va)*by. Since the best hyperparameter has been found, then the next question is what the optimal model with the optimal parameter is.

Using as much data as possible to train the model makes it more robust, thus, using filter0 or the two other models that only use a part of the data for training would be a waste of data.

A better model would be filter3, since it has been trained on the whole data set and therefore becomes more robust, though it must be noted that err3 does not tell us anything about its estimated performance since it was produced by predicting on data used to train the model. The real error (generalization error) is instead obtained as described below in question 2.

-> filter3 is the filter we should return to the user.

 # 2. What is the estimate of the generalization error of the model selected? err0, err1, err2 or err3? Why?

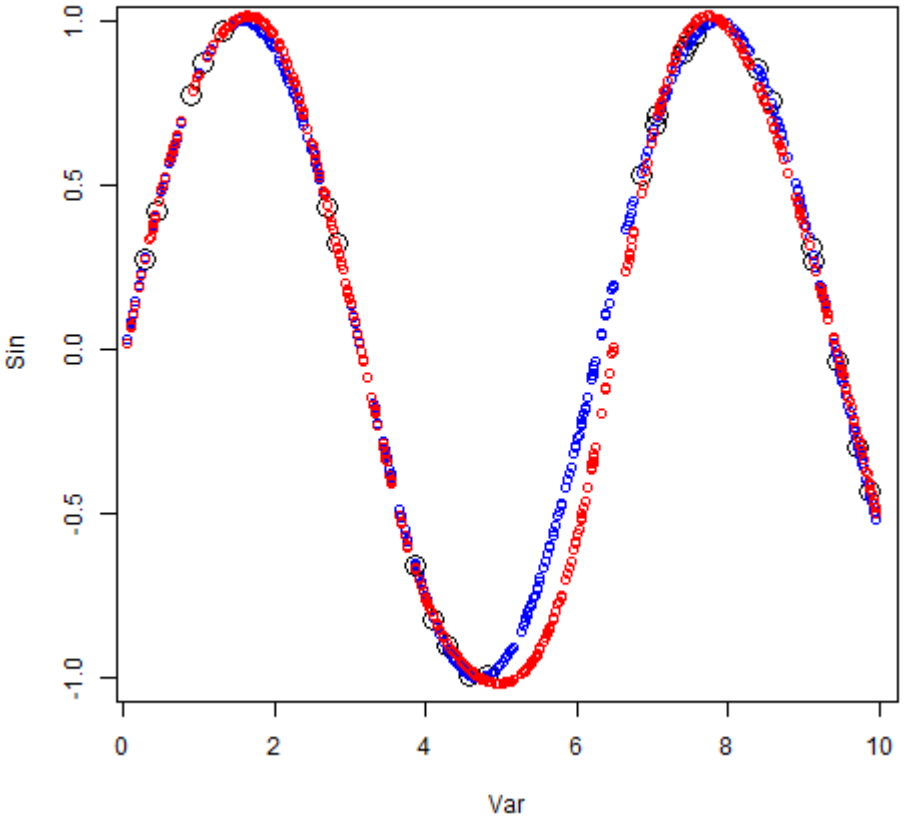
Generalization error usually refers to the error estimated on data the model has not seen, which means that since the training and validation data was used to tune the hyperparameters, thus the unseen data is the testing data. Hence, the generalization error (like calculating the error for the outer loop) is obtained by training the model on training and validation data and then testing on the testing data

-> error2 is the generalization error = 0.08364544

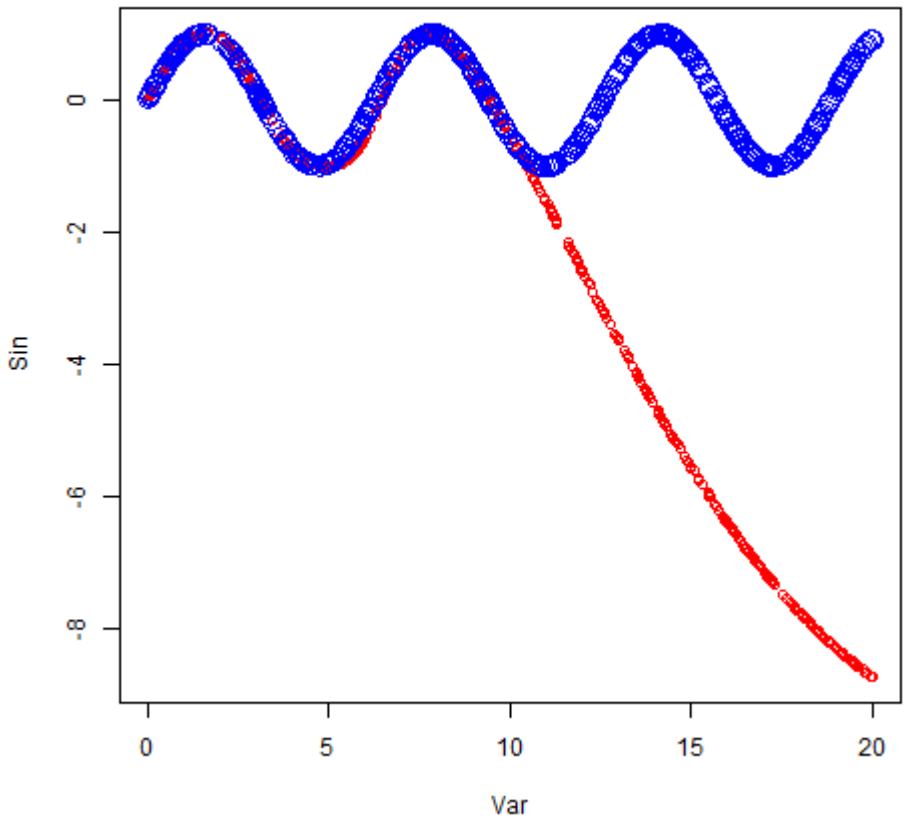
Here we can see that the regular cross-validation is over optimistic, since err0 < err2

Assignment 3

1. For the first task, 10 hidden nodes in a single layer were chosen as a reasonable number. The neural network was generated based on the training data, and applying that model to the test data resulted in the following plot:

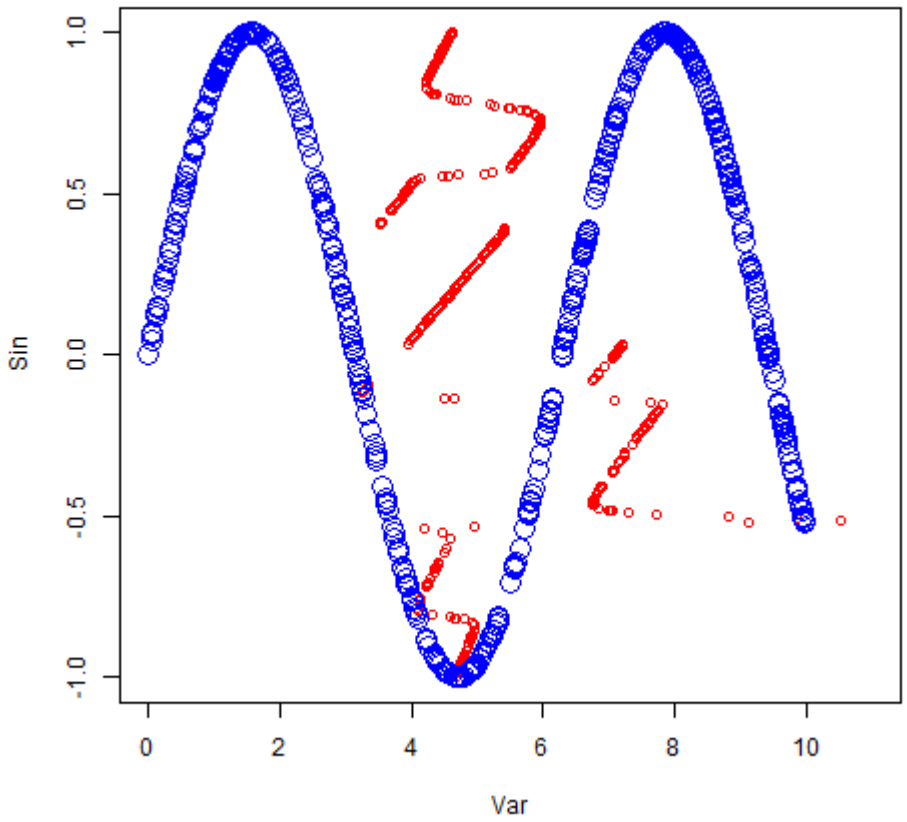


   This shows that the model fits rather well to the test data, which could be quite expected as sinus acts predictable according to its input. There might be some slight overfitting to the data, but overall, it is satisfactory.
2. When predicting data in the 0-20 range using the previous model, it results in the following plot:



It shows that the model predicts values up to 10 quite well, but for values higher than that it fails to predict any points correctly. This seems reasonable. As the neural network is only trained for the span of 0-10, it can not predict fluctuations in the actual function after that. Instead, it would rather assume that the function would continue in the same general direction as the derivate of the function around the point of 10, which can be seen clearly in the plot above. As a neural network (At least in this case) simply linearly adds different components to predict the model, it could not reasonably predict that the sine function will progress in a repeated pattern.

3. For the final task, a new neural network was calculated, this time by trying to predict x from sin(x) with all 500 datapoints as a training set. When trying to predict the same data using the model, we got the following plot:



As can be seen, the model was not able to predict these values at all. This is not really surprising, as the sine function's repeating pattern makes it very difficult to predict x from sin(x). Only in this short span , most values of sin(x) have 2-4 corresponding x-values, which makes it almost impossible to correctly predict. Instead, we see that predicted values, in general, correspond to the mean of the possible values for that sine value, i.e. for sin(x) > 0, which has 4 different x-values, predicted values are around 5, for 0 > sin(x) > -0.5, x-values are predicted around 7 as three different values are possible, and for sin(x) < -0.5, which have two possible x-values, predicted values are also around 5.

Statement of Contribution

- Assignment 1: Was solved by Per Bark (perba583)
- Assignment 2: Was solved by Ingrid Wendin (ingwe018)
- Assignment 3: Was solved by David Åström (davas593)

The report was written by all three group members, who wrote their respective part, and then control read the whole report.

Appendix

Code - Assignment 1

```
set.seed(1234567890)
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

h_distance <- 150000 # These three values are up to the students
h_date <- 10
h_time <- 2

a <- 58.3836  # The point to predict (up to the students) 58.3836  14.826
b <- 14.826
date <- "2013-08-04" # 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))

# Help functions
get.date.dist <- function(old.date, new.date){
  d <- NULL
  # Old month
  o.month <- as.matrix(as.numeric(strftime(old.date,"%m")))
  o.day <- as.matrix(as.numeric(strftime(old.date, "%d")))
  # New month
  n.month <- as.numeric(strftime(new.date,"%m"))
  n.day <- as.numeric(strftime(date, "%d"))

  #Calculate distance
  for(i in 1:length(old.date)) {
    if(o.month[i]==n.month) {
      d = c(d,abs(o.day[i]-n.day))
    } else {
      #distance months and days - assumes 31 days each month
      # Direction 1
      d1.month <- ifelse(n.month-o.month[i]<0, (n.month-o.month[i])%%12, n.month-o
.month[i])
      d1.day <- (31-o.day[i]+n.day)
      d1 <- (d1.month-1)*31+d1.day

      # Direction 2
      d2.month <- ifelse(o.month[i]-n.month<0, (o.month[i]-n.month)%%12, o.month[i
]-n.month)
      d2.day <- (31-n.day+o.day[i])
      d2 <- (d2.month-1)*31+d2.day

      d=c(d,min(d1,d2))
    }
  }

  return (d)
}

gaus.kernel.p <- function(x) {
  return(exp(-(norm(x/h_distance, "2"))^2))
}
```

```r
gaus.kernel.d <- function(x) {
  return(exp(-(norm(x/h_date, "2"))^2))
}

gaus.kernel.t <- function(x) {
  return(exp(-(norm(x/h_time, "2"))^2))
}


#filter out values after the chosen date and convert to date
st_filtered <- st[as.Date(st$date) < as.Date(date),]
st_filtered[9] <- as.Date(st_filtered[,9])

#physical distance in meter
target.loc <- c(a,b)
p.dist <- distHaversine(st_filtered[4:5], target.loc)
k.p <- as.matrix(sapply(p.dist, gaus.kernel.p)) #exp(-norm(x/h_distance, "2")^2)


# date distance in days - year is irrelevant for distance
date <- as.Date(date)
d.dist <- get.date.dist(st_filtered[,9], date)
k.d <- as.matrix(sapply(d.dist, gaus.kernel.d)) #exp(-norm(x/h_date, "2")^2)



# Control for kernel values physical distance and distance in days
plot(p.dist, k.p, xlab = "Distance in meter")

plot(d.dist, k.d, xlab = "Distance in days")

# The temperature = (sum((k1+k2+k3)*t))/sum(k1+k2+k3) - sapply(time, function(x) d
ifftime(n.time,x))

k.tot <- k.p + k.d
time <- strptime(st_filtered[,10], "%H:%M:%S")

#04:00:00
t <- NULL
n.time <- strptime(times[1], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[1] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[1]
```

```
## [1] 6.095466
```

```r
#06:00:00
t <- NULL
n.time <- strptime(times[2], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
```

```
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[2] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[2]

## [1] 5.964336

#08:00:00
t <- NULL
n.time <- strptime(times[3], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[3] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[3]

## [1] 6.853378

#10:00:00
t <- NULL
n.time <- strptime(times[4], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[4] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[4]

## [1] 7.638957

#12:00:00
t <- NULL
n.time <- strptime(times[5], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[5] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[5]

## [1] 7.886145

#14:00:00
t <- NULL
n.time <- strptime(times[6], "%H:%M:%S")
```

```
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[6] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[6]
```

```
## [1] 7.985989
```

```
#16:00:00
t <- NULL
n.time <- strptime(times[7], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[7] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[7]
```

```
## [1] 7.583366
```

```
#18:00:00
t <- NULL
n.time <- strptime(times[8], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[8] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[8]
```

```
## [1] 6.938525
```

```
#20:00:00
t <- NULL
n.time <- strptime(times[9], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[9] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[9]
```

```
## [1] 6.905982
```

```
#22:00:00
t <- NULL
n.time <- strptime(times[10], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[10] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[10]

## [1] 6.766571

#00:00:00
t <- NULL
n.time <- strptime(times[11], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot + k.t
temp[11] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[11]

## [1] 6.616189

temp

##  [1] 6.095466 5.964336 6.853378 7.638957 7.886145 7.985989 7.583366 6.938525
##  [9] 6.905982 6.766571 6.616189

# Students? code here
plot(seq(4,24, 2), temp, type="o", main = "Day temperature using sum kernel", xlab
="Time in hours")

# The temperature = (sum((k1*k2*k3)*t))/sum(k1*k2*k3)
# Multiplication makes the weight of perfect targets disappear
# Thus the overrepresentation of physical place is reduced/removed

k.tot <- k.p * k.d
time <- strptime(st_filtered[,10], "%H:%M:%S")



#04:00:00
t <- NULL
n.time <- strptime(times[1], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
```

```
k.all <- k.tot * k.t
temp[1] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[1]

## [1] 13.48286

#06:00:00
t <- NULL
n.time <- strptime(times[2], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[2] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[2]

## [1] 14.53164

#08:00:00
t <- NULL
n.time <- strptime(times[3], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[3] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[3]

## [1] 16.55993

#10:00:00
t <- NULL
n.time <- strptime(times[4], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[4] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[4]

## [1] 19.15393

#12:00:00
t <- NULL
n.time <- strptime(times[5], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
```

```
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[5] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[5]

## [1] 20.15522

#14:00:00
t <- NULL
n.time <- strptime(times[6], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[6] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[6]

## [1] 20.4151

#16:00:00
t <- NULL
n.time <- strptime(times[7], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[7] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[7]

## [1] 19.43271

#18:00:00
t <- NULL
n.time <- strptime(times[8], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[8] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[8]

## [1] 17.95306

#20:00:00
t <- NULL
```

```
n.time <- strptime(times[9], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[9] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[9]
```

```
## [1] 16.44087
```

```
#22:00:00
t <- NULL
n.time <- strptime(times[10], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[10] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[10]
```

```
## [1] 14.69059
```

```
#00:00:00
t <- NULL
n.time <- strptime(times[11], "%H:%M:%S")
for(j in 1:length(time)) {
  t <- c(t, abs(difftime(n.time,time[j])))
}
t <- ifelse(t>12, 12-t%%12, t)
k.t <- as.matrix(sapply(t, gaus.kernel.t))

# Predicted temperature for time
k.all <- k.tot * k.t
temp[11] <- sum(k.all*st_filtered[,11])/sum(k.all)
temp[11]
```

```
## [1] 14.03858
```

```
temp
```

```
##  [1] 13.48286 14.53164 16.55993 19.15393 20.15522 20.41510 19.43271 17.95306
##  [9] 16.44087 14.69059 14.03858
```

```
# Students? code here
plot(seq(4,24, 2),temp, type="o", main="Day temperature using multiplication kerne
l", xlab = "Time in hours")

#Control of h_t: for 24:00
plot(t, k.t, main="Kernel hour value to distance in hour for 24:00", xlab ="Distan
ce in hours")

# Control Linköping
```

```
Linköping <- which(st_filtered[2]=="Linköping" )
Linköping <- st_filtered[Linköping,]

plot(row.names(Linköping), Linköping$air_temperature, xlab ="Data point id", ylab=
"Temperature", main ="Temperature in Linköping")
```

Code - Assignment 2

The file provided by José

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

library(kernlab)
set.seed(1234567890)

data(spam)

index <- sample(1:4601)
tr <- spam[index[1:3000], ]
va <- spam[index[3001:3800], ]
trva <- spam[index[1:3800], ]
te <- spam[index[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)
  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(e
rr_va)*by)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0

## [1] 0.07

filter1 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(e
rr_va)*by)
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)
mailtype <- predict(filter2,te[,-58])
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
err2

## [1] 0.08364544

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

```
## [1] 0.03370787

# 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 ? err
0, err1, err2 or err3 ? Why ?
```

Code - Assignment 3

```
library(neuralnet)
set.seed(1234567890)

# Part 1, generate neuralnet from Var to Sin
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(31, -1, 1)
nn <- neuralnet(Sin ~ Var, data = tr, startweights = winit, hidden = 10)

# Plot of the training data (black), test data (blue), and predictions (red)
png("part1.png")
plot(tr, cex=2)

points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
dev.off()

## png
##   2

# Part 2, test model on new set
Var <- runif(500, 0, 20)
mydata.2 <- data.frame(Var, Sin=sin(Var))

#Plot of the test data (blue), and predictions (red)
png("part2.png")
plot(mydata.2[,1], predict(nn, mydata.2), col="red", cex=1, ylab = "Sin", xlab = "
Var")
points(mydata.2, col = "blue", cex=2)
dev.off()

## png
##   2

# Part 3, generate neuralnet from Sin to Var
Var <- runif(500, 0, 10)
mydata.3 <- data.frame(Var, Sin=sin(Var))
tr <- mydata.3 # Test
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
nn <- neuralnet(Var ~ Sin, data = tr, startweights = winit, hidden = 10)

# Plot of the training data (blue), and predictions (red)
png("part3.png")
plot(predict(nn,tr), tr[,2], col="red", cex=1, ylab = "Sin", xlab = "Var")
points(tr, col = "blue", cex=2)
dev.off()

## png
##   2
```