Computer lab 3, group 29

Machine Learning, TDDE01

Oscar Hoffmann, oscho091
Martin Sörme, masor844
Hjalmar Öhman, hjaoh082

1. Statement of contribution

Oscar Hoffmann was responsible for assignment 1. In the second assignment, Martin Sörme was responsible and Hjalmar Öhman was responsible for assignment 3. For each assignment this included the code writing and analysis. However, all members discussed the problems and took help from the other members in the tasks during the coding. After finishing the code writing and analysis, all three members sat down and discussed the problems and presented the solutions. This was done to ensure that all members understood the code, solutions and analyses for the tasks.

2. Assignments

2.1 Kernel Methods

```
#libraries
library(geosphere)

rm(list = ls())

set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

# The point to predict (adress in vallastaden, Linköping)
a <- 58.393379 #latitud
b <- 15.584957 #longitud

date <- "2010-12-24" # The date to predict
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")

#Filter out times and dates that should not be included
specified_date = as.POSIXct(date, format = "%Y-%m-%d")
st$date = as.POSIXct(st$date, format = "%Y-%m-%d")
st$time = format(strptime(st$time, format = "%H:%M:%S"), "%H:%M:%S")
st = st[st$date <= specified_date & st$time %in% times,]
```

This assignment was about implementing a kernel method to predict the hourly temperatures for a date and place in Sweden. To do this, we first used the provided template to load the data from the two files “stations.csv” and “temps50k.csv” and merge them by station number. Then, we chose an address in Vallastaden, Linköping, as the place for the temperature prediction and assigned “a" and “b” the coordinates of the address. Following that, we selected the date to predict to be 2010-12-24 (Christmas Eve) and specified a times vector with times between 04.00 and 24.00 in two-hour intervals, which was the different times to predict the temperature for. We then converted the dates and times in the 'st' dataset to be in a comparable format. Following that, we filtered out the dates after 2010-12-24 and the times that were not in our times vector. We did this in order to make a correct and unbiased prediction as per the instruction.

```
#kernel function
gaussianKernel = function(x_diff, h) {
  return(exp(-(x_diff / h)^2))
}

#function to calculate the distance between the point to predict and the station coordinates
calc_distance_diff = function() {
  stations_coordinates = matrix(c(st$longitude, st$latitude), ncol = 2)
  POI = matrix(c(b,a), ncol = 2)
  dist_diff = distHaversine(stations_coordinates, POI)
  return(dist_diff)
}

#Function to calculate the difference between the day to predict and the day of the measurement
calc_date_diff = function() {
  day_diff = as.numeric(difftime(date, st$date, units = "days")) %% 365
  return(day_diff)
}

#Function to calculate the difference between hours to predict and the hours the measurements were taken
calc_hour_diff = function(hour) {
  hour_diff = abs(as.numeric(difftime(strptime(times[hour], format = "%H:%M:%S"),strptime(cbind(st$time), forma
  return(hour_diff)
}
```

The next step was to implement the gaussian kernel formula as specified in the lecture slides. We then created three functions:

- **calc_distance_diff:** calculates the difference in distance between the point of interest and the stations using the distHaversine() function from the geosphere package.
- **calc_date_diff:** calculates the difference between the day to predict and the day of the measurement. It uses modulo 365 to account for the annual cyclical nature of the dates.
- **calc_hour_diff:** calculates the difference between hours to predict and the hours when the measurements were taken.

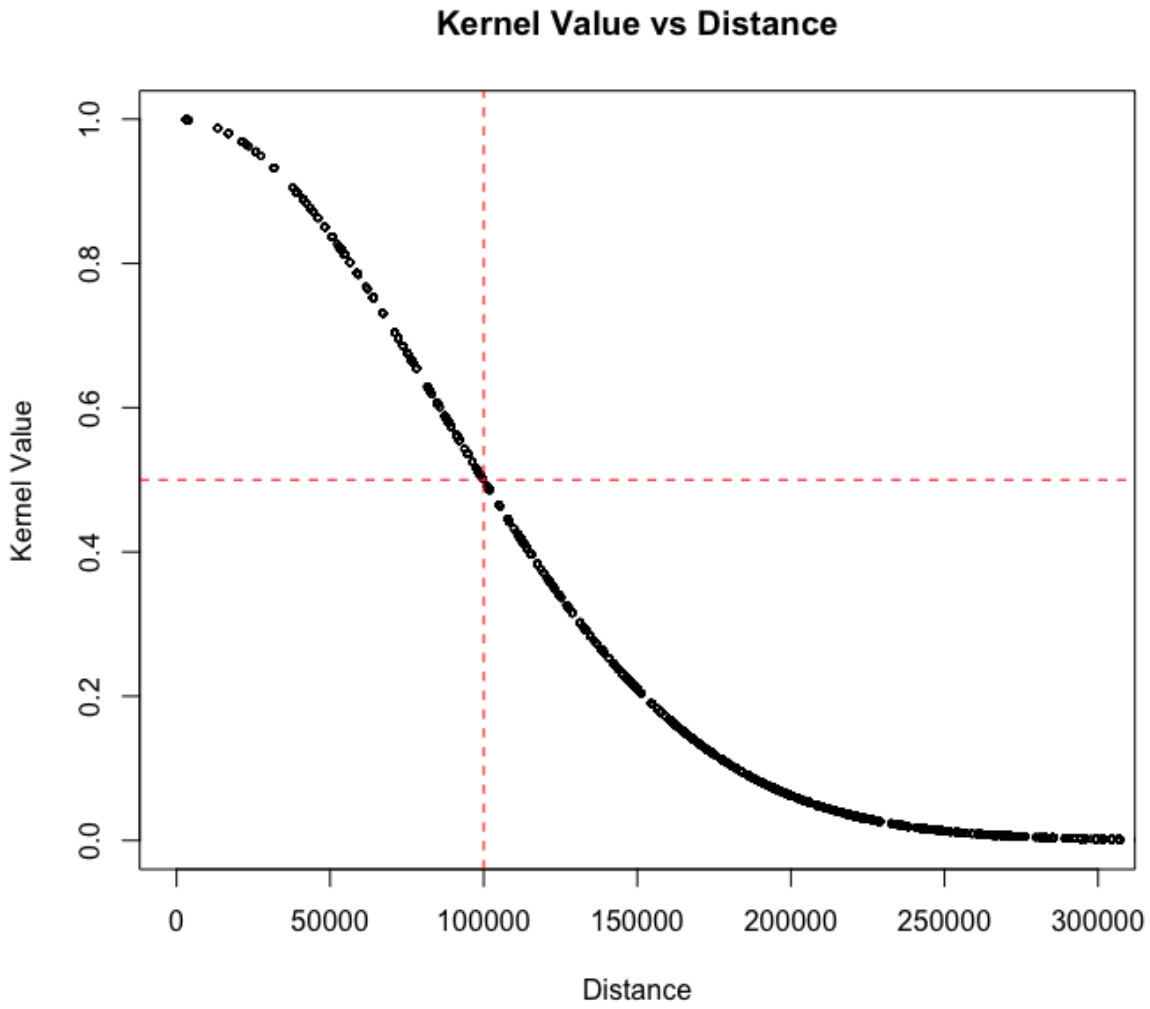
```
#function to plot kernel values against distance/days/hours
plot_kernel_vs_distance = function(h_value, against, xlim_value, v_value) {
  kernel_value = gaussianKernel(against, h_value)
  plot(against, kernel_value, type = "p", main = "Kernel Value vs Distance", xlab = "Distance", ylab = "Kernel Value", col = "bla
  abline(h = 0.5, col = 'red', lty = 2)
  abline(v = v_value, col = 'red', lty = 2)
}

#some h_values to test for distance
h_values = c(10000, 20000, 50000, 80000, 100000, 120000)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_distance_diff(), 300000, 100000)
}

#some h_values to test for days
h_values = c(2, 5, 7, 8, 9, 10)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_date_diff(), 20, 7)
}

#some h_values to test for hours
h_values = c(2, 4, 6, 8, 10, 12)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_hour_diff(), 15, 5)
}

#h_values final
h_distance = 120000
h_date = 8
h_time = 6
```

We were instructed to choose appropriate smoothing coefficients for each of the three kernels. To do this we created a function that plotted the kernel value against the distance/day/hour for different h_values. We then created vectors with some suitable h_vaules for each of the three kernels. We looped through this vector to produce six plots for each kernel. This resulted in the final smoothing coefficients being 120 000 for distance, 8 for date and 6 for time. These smoothing coefficients ensured large kernel values for closer points and small values for distant points. Below is our reasoning for choosing the different smoothing coefficients for each of the three kernels.

Physical distance

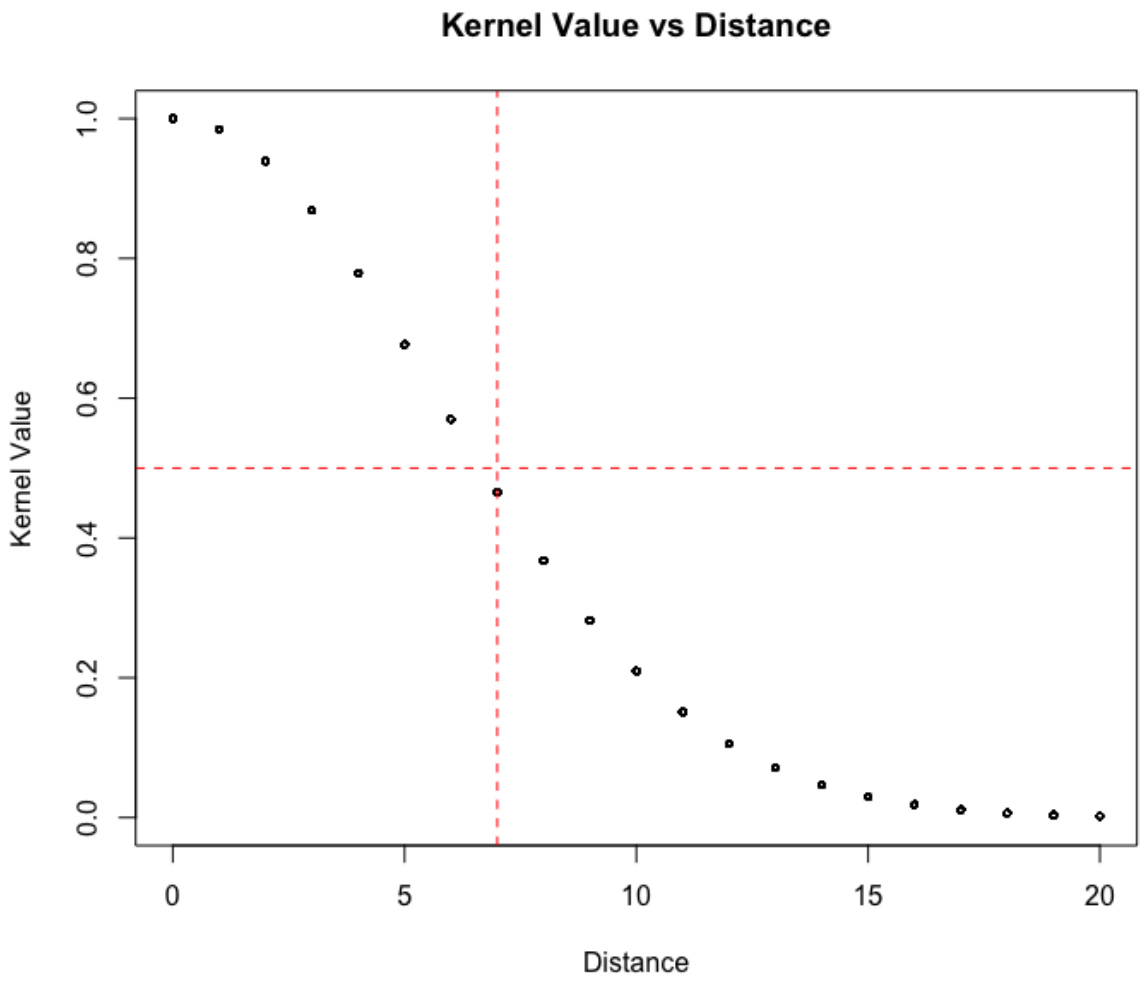
When choosing a smoothing coefficient for the distance we reasoned that a measurement 100 km away from the point of interest should be 1/2 as important as a measurement on the exact location. As can be seen in the graph, the kernel value is 0.5 when the distance is 100

km for h_value 120 000, which is why we chose it.



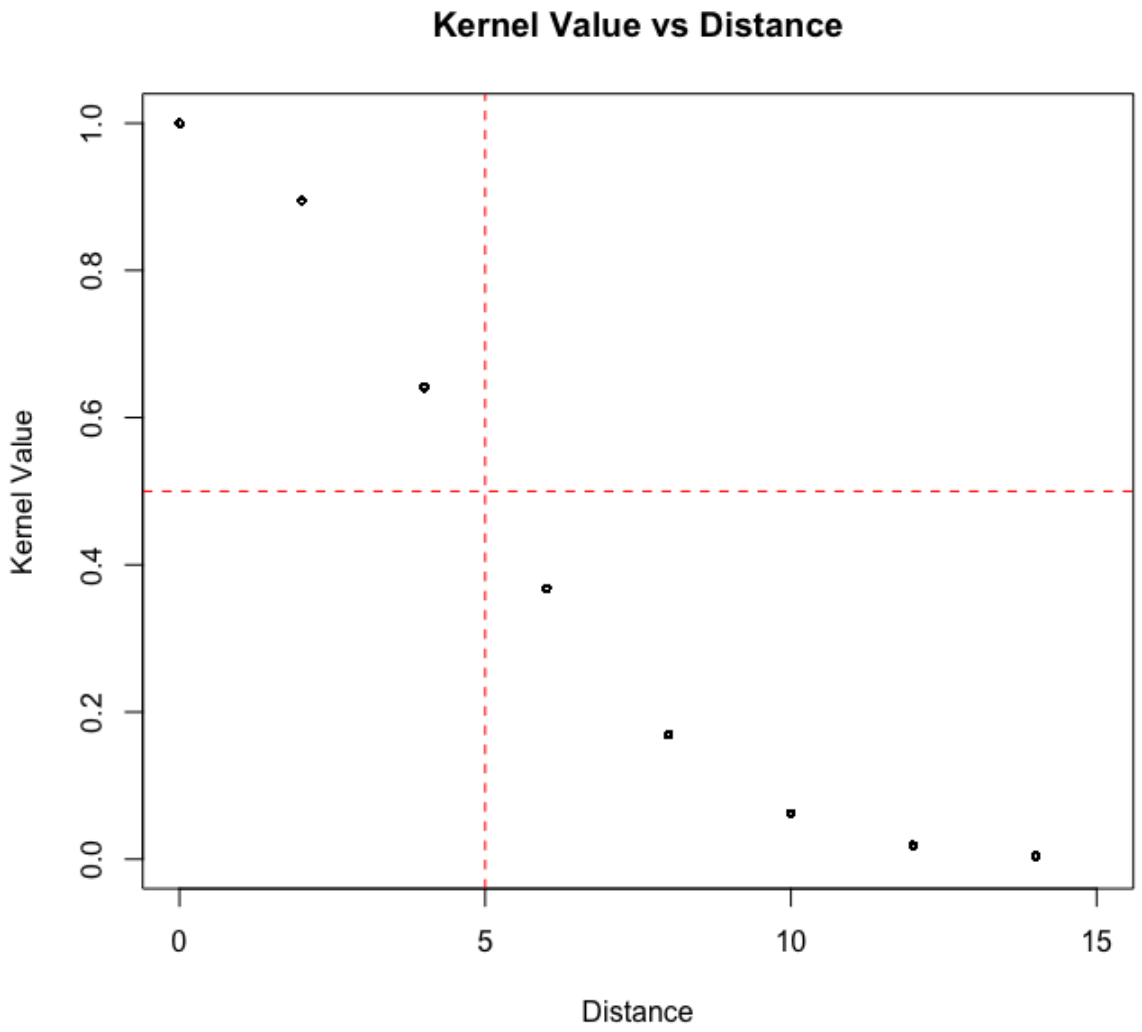
Days difference

When choosing the smoothing coefficient for the difference in days we reasoned that a measurement that differs a week (7 days) should be 1/2 as important as a measurement made the same day. This resulted in 8 as the h-value. As can be seen in the graph, the kernel value is 0.5 approximately when 7 days have passed.



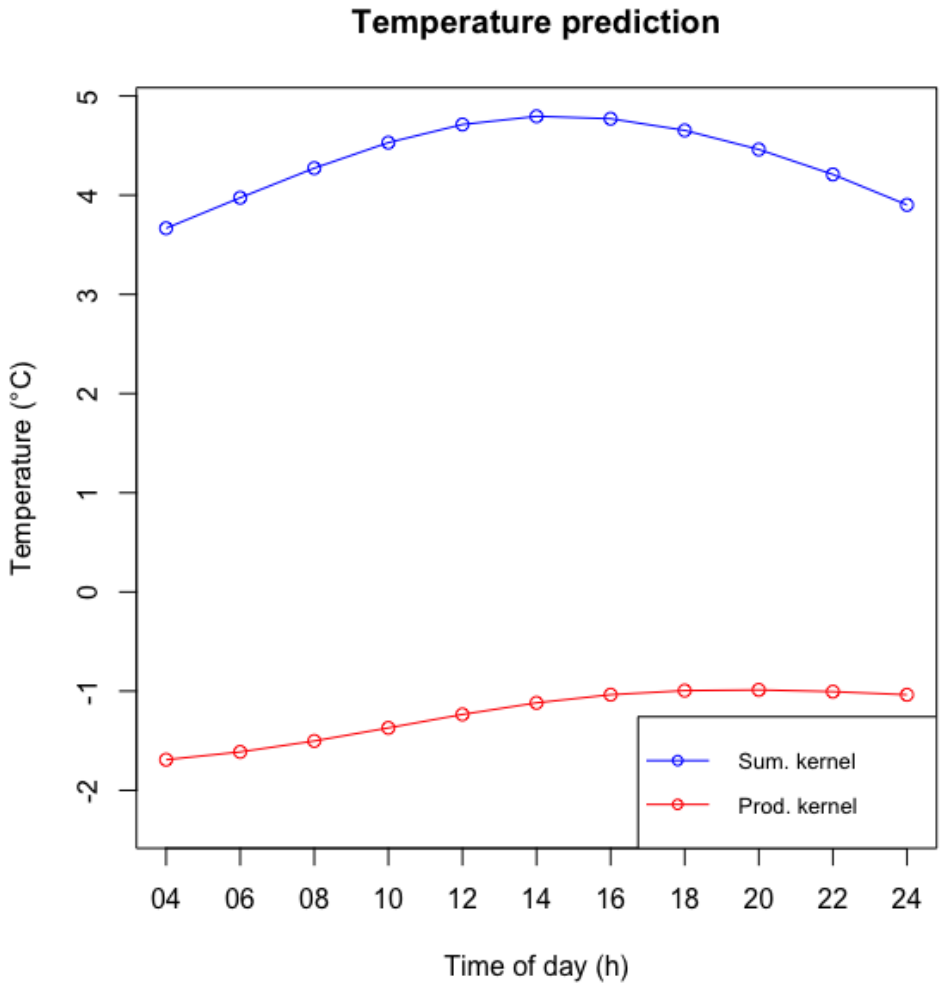
Hours distance

When choosing a smoothing coefficient for the distance in hours we reasoned that a measurement that differs 5 hours should be 1/2 as important as a measurement made the same hour. This resulted in 6 as the h-value. This can be seen in the graph where the kernel value is 0.5 when 5 hours have passed.




```r
#date & distance Kernels
date_kernel = gaussianKernel(calc_date_diff(), h_date)
dist_kernel = gaussianKernel(calc_distance_diff(), h_distance)

#temperature vectors
temp_sum = vector(length=length(times))
temp_prod = vector(length=length(times))

#looping through the hours in the times vector
for(hour in seq_along(times)) {
  hour_diff = calc_hour_diff(hour)
  hours_kernel = gaussianKernel(hour_diff, h_time)

  #Sum and product of the kernels
  kernel_sum = dist_kernel + date_kernel + hours_kernel
  kernel_prod = dist_kernel * date_kernel * hours_kernel

  #Add the temperatures to the temperature vectors
  temp_sum[hour] = sum(kernel_sum %*% st$air_temperature) / sum(kernel_sum)
  temp_prod[hour] = sum(kernel_prod %*% st$air_temperature) / sum(kernel_prod)
}

#Plotting temperatures for both sum and prod kernels
plot(temp_sum, type="o", main = "Temperature prediction", xlab = "Time of day (h)", ylab = "Temperatur
lines(temp_prod, type="o", col = "red")
axis(1, at = seq_along(times),  labels=substring(times, 1, 2))
legend("bottomright", c("Sum. kernel", "Prod. kernel"), col = c("blue", "red"),  pch=1, lty=1, box.lty
```

When we had chosen the h_values we could create the date- and distance kernels. After this we created two empty vectors to store the temperatures for the summation and multiplication of the kernels. Following that, we looped through the elements in the “times” vector and calculated an hour kernel for each hour in the vector. For each time we summarized and multiplied the kernels and then stored the predicted temperatures in the temp_sum and temp_prod vectors. Finally we plotted the summarized- and multiplied vectors in a graph showing how the predicted temperature varied from 04:00 to 24:00.



Analysis of the plot

The graph shows that the sum kernel method tends to forecast higher temperatures compared to the product kernel method. This difference can be explained by the way each method processes the input parameters. In the sum kernel method, a high value in one parameter can compensate for a lower value in another since the gaussian kernels are added. This can lead to higher overall weights and thus higher predicted temperatures.

In contrast, the product kernel method multiplies the values of the kernels together, which means that a low value in any single parameter can drastically reduce the overall kernel value, leading to lower weights and lower predicted temperatures. For example, a temperature reading taken at the correct location and time, but during a different season (large difference in days), still significantly influences the prediction in the sum method.

However, in the product kernel method, the result is more cautious. If any single parameter (such as the season) is mismatched, it substantially reduces the influence of that reading on the final prediction. This makes the product kernel method more conservative since it only gives significant weight to temperature readings when all parameters closely align with the target conditions.

2.2 Support vector machines

2.2.1 TASK 1

The code trains four different support vector machine filters, each trained on different subsets of the dataset. For example, filter1 is trained on the training set. Regarding which filter should be recommended to the user, we should choose filter3. The reasoning for choosing this one is because it is trained on the entire dataset, allowing it to learn from a wider range of data. This could lead to better generalization. The other filters only use subset, and will therefore not be as efficient models. Since it uses the entire dataset, it could also be trained on any potential outlier data points in the dataset.

2.2.2 TASK 2

For this, we choose err2 as the generalization error returned to the user. This is because it has the largest coverage of the dataset while having separation between the predictions and training. For example, filter3 uses the entire dataset and then calculates the generalization error (err3) on the test data. Here, it has already been trained on this data, leading to a misleading answer for the error. The data for the error is not unseen and therefore misleading. Err2 is chosen since it has the largest coverage of training the model and separation, the error is calculated from test and the model is trained on trainva:

```
trainva <- spam[1:3800, ]
test <- spam[3801:4601, ]
```

2.2.3 TASK 3

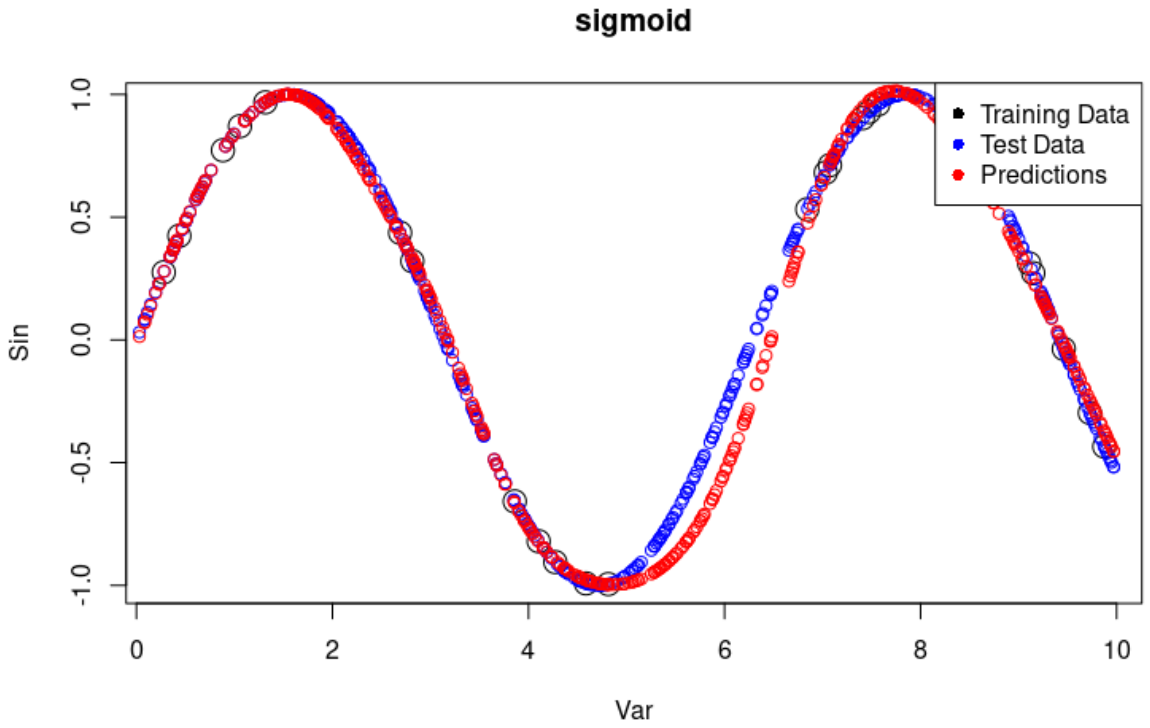
The implementation of Task 3 is done in the following way:

```r
# 3. Implementation of SVM predictions.

sv <- alphaindex(filter3)[[1]]
support.vectors <- spam[sv, -58]
co <- coef(filter3)[[1]]
inte <- - b(filter3)

#  Smaller sigma value -> narrower Gaussian function
kernel.function <- rbfdot(0.05)

k <- NULL
dot.products <- NULL

# Predictions for the first 10 points
for (i in 1:10) {

  k2 <- NULL

  for (j in seq_along(sv)) {
    k2 <- unlist(support.vectors[j, ])
    sample <- unlist(spam[i, -58])
    # Add the dot product to the existing ones
    dot.products <- c(dot.products, kernel.function(sample, k2))
  }
  # Start and end index
  start <- 1 + length(sv) * (i - 1)
  end <- length(sv) * i

  # Calculate predictions and add them
  prediction <- co %*% dot.products[start:end] + inte
  k <- c(k, prediction)
}

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

2.3 Neural Networks

2.3.1 TASK 1

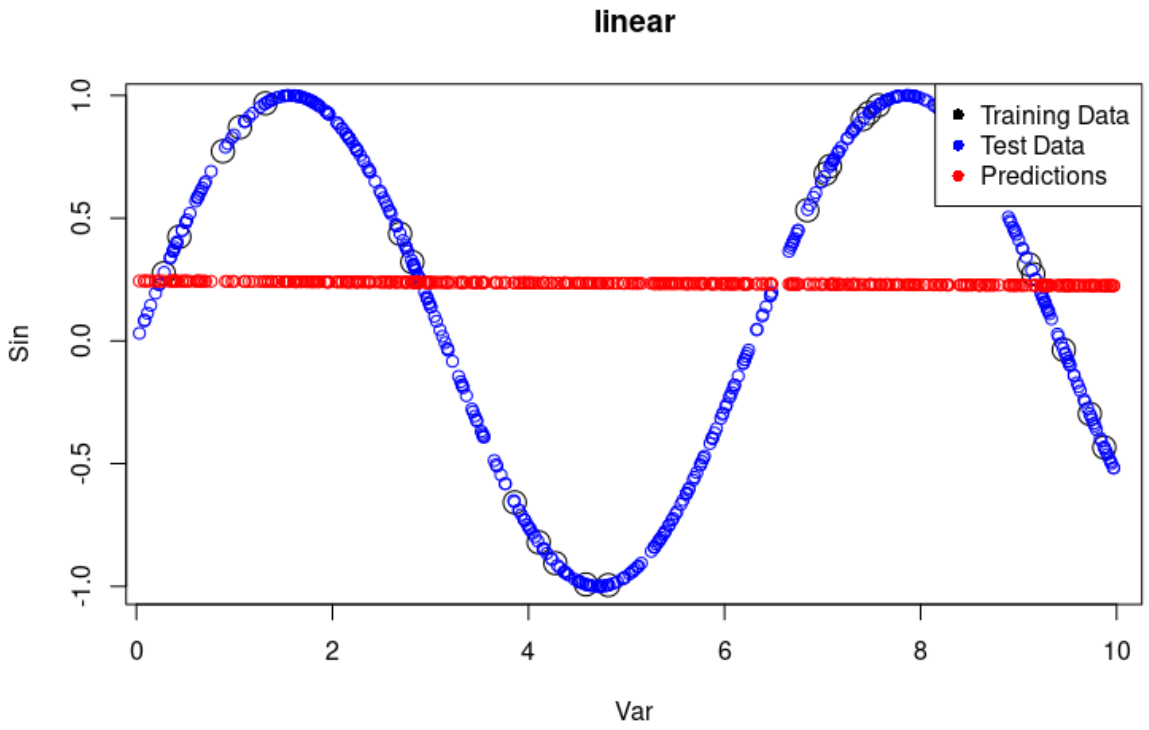
Plot the training and test data, and the predictions of the learned NN on the test data. Comment your results.

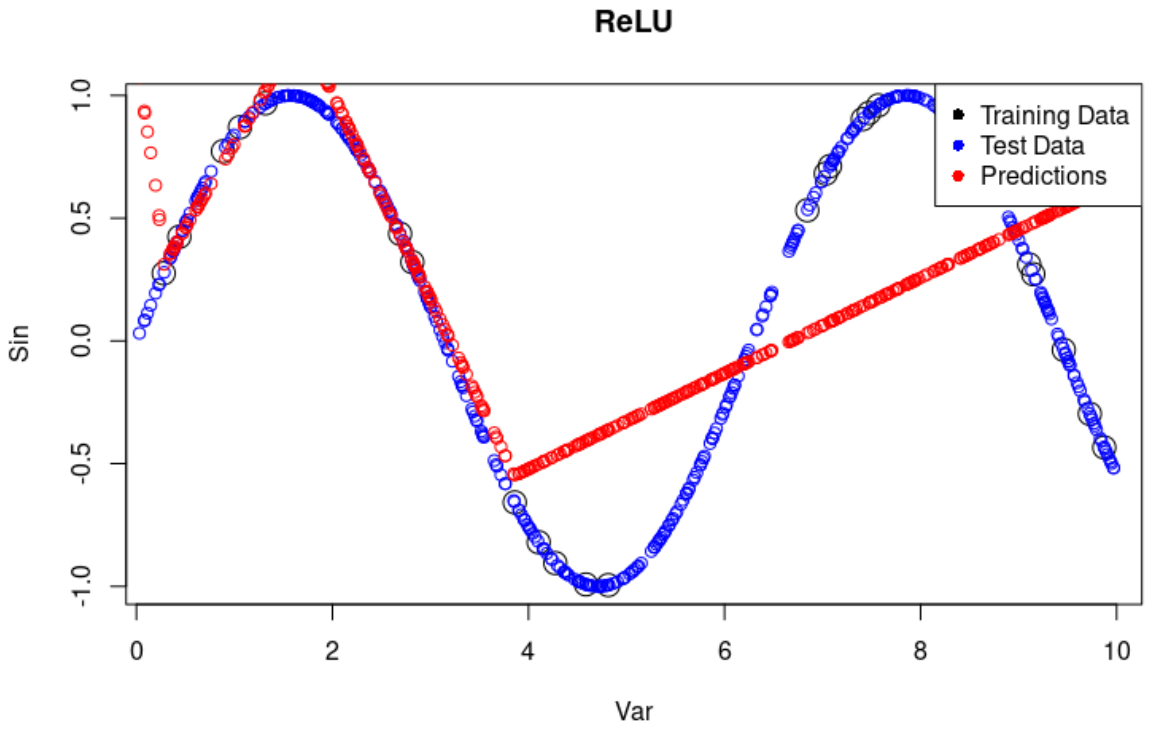


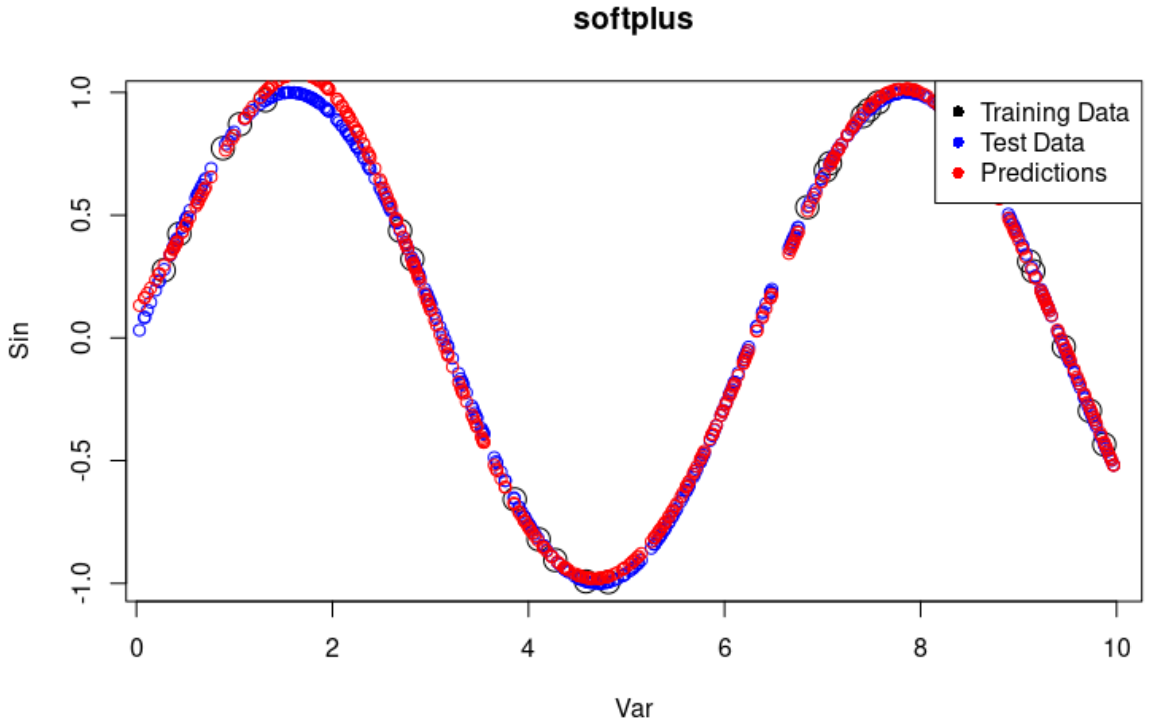
Struggles between where it lacks training data, otherwise looks good.

2.3.2 TASK 2

Plot and comment your result



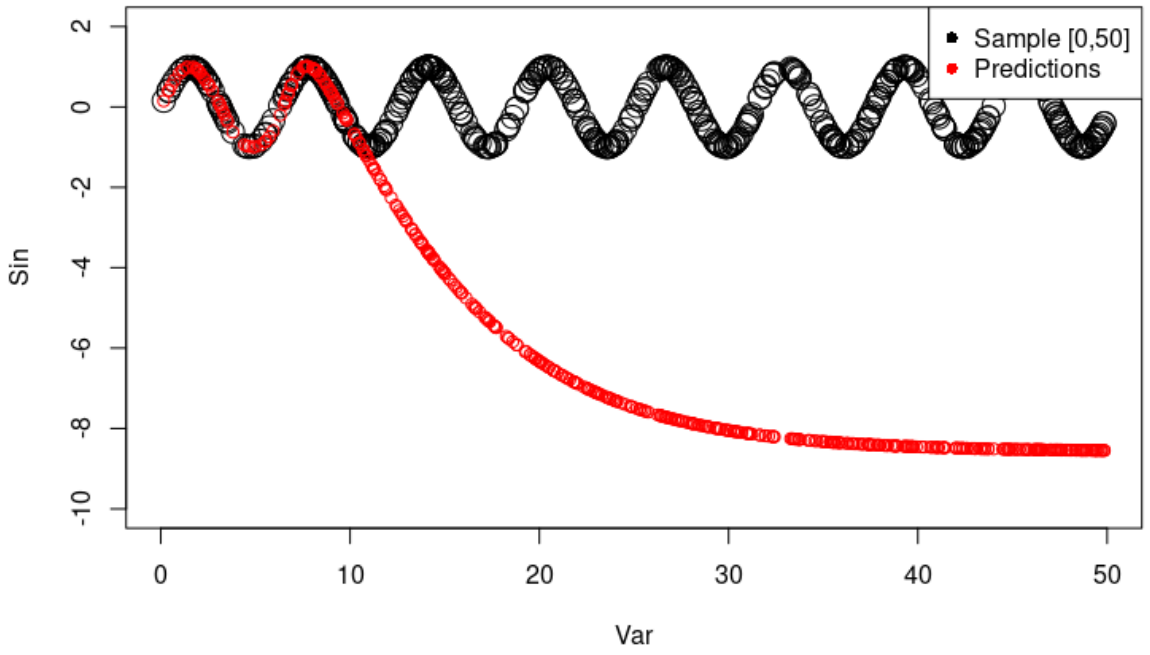




linear: Having a linear activation function essentially means that our neural network is a linear regression, which is unable to capture the non-linear pattern of the sinus function.
ReLU: ReLU is kind of a linear activation function (0 or x), so it would require a lot more lines than the 4 generated in this case to capture the sinus function.
softplus: performs very well where there is no training data when Var is around 5, but performs worse than the others when Var is around 2.

2.3.3 TASK 3

Plot and comment your results.

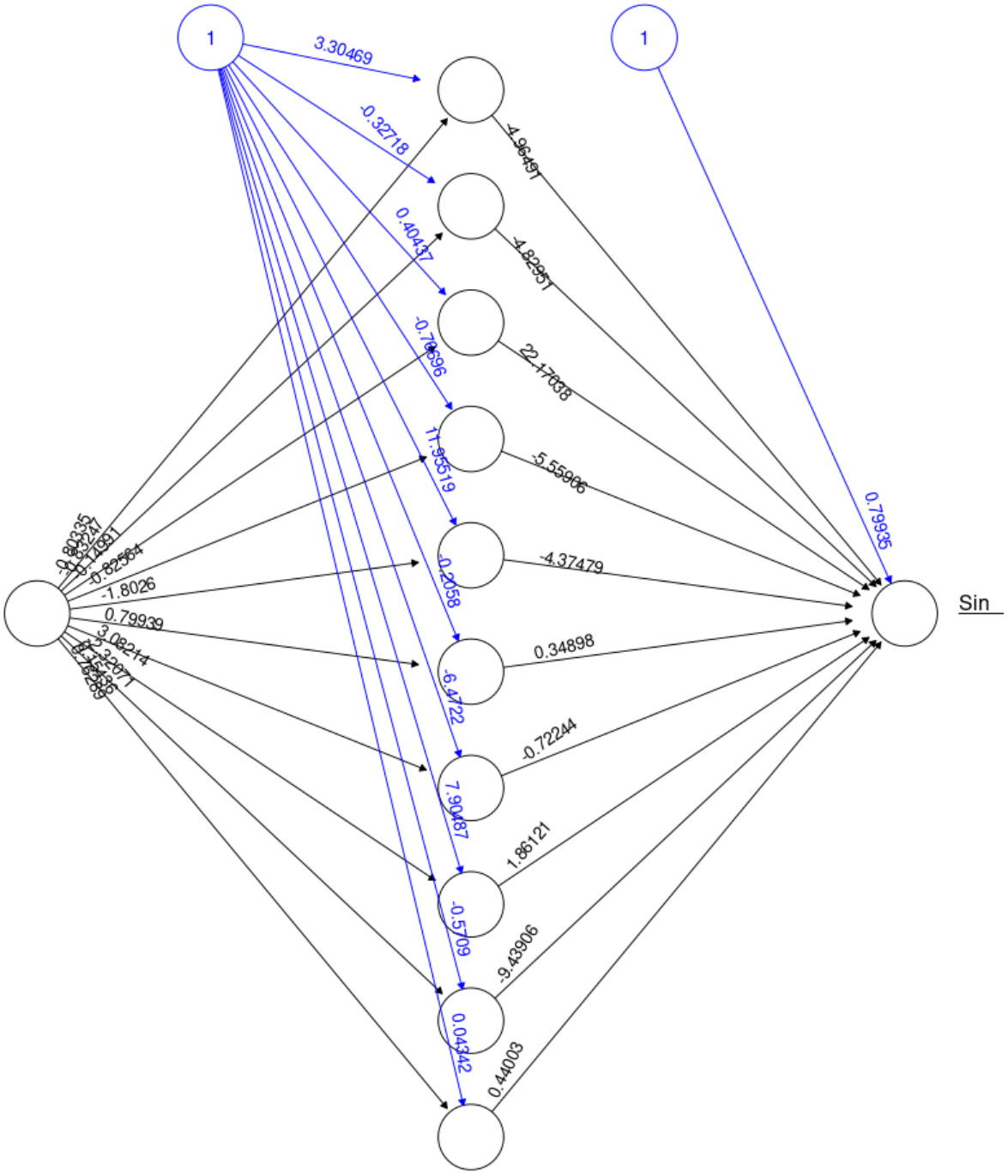


The neuralnet has never seen any values of var greater than 10 earlier, and thus has no clue what to output.

2.3.3 TASK 4

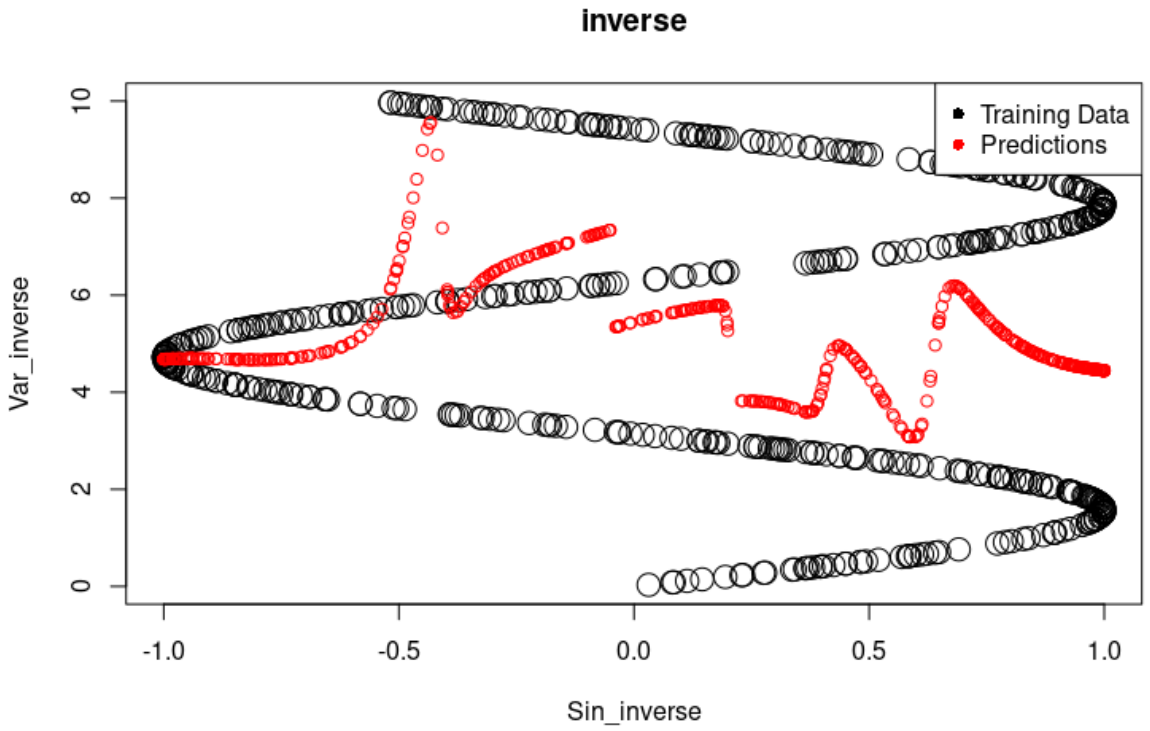
In question (3), the predictions seem to converge to some value. Explain why this happens.

It seems to converge to -8. For greater and greater inputs X, the hidden layer nodes will converge to either 0 or 1. The output of the hidden layer to the final node is therefore a constant, **C**. sigmoid(**C**) + 0.79935 = -8


1
3.30469
-0.32718
0.40437
-0.70696
11.95519
-0.2058
-6.4722
7.90487
-0.5709
0.04342
1
-4.96497
-4.82951
22.17038
-5.55906
-4.37479
0.34898
-0.72244
1.86121
-9.43906
0.44003
0.79935
-1.8026
0.79939
3.08214
Sin

2.3.3 TASK 5

Plot and comment your results.



One input may have multiple outputs. For example, the output can be either x=0 or x=pi, for input sin(x) = 0. It is therefore impossible for the model to know if the input sin(x) = 0 should output pi or 0.

Appendix

——————————————————Assignment 1——————————————————

```
#libraries

library(geosphere)


rm(list = ls())


set.seed(1234567890)

stations <- read.csv("stations.csv", fileEncoding = "latin1")

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

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


# The point to predict (adress in vallastaden, Linköping)

a <- 58.393379 #latitud

b <- 15.584957 #longitud


date <- "2010-12-24" # The date to predict

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")


#Filter out times and dates that should not be included
```

```
specified_date = as.POSIXct(date, format = "%Y-%m-%d")

st$date = as.POSIXct(st$date, format = "%Y-%m-%d")

st$time = format(strptime(st$time, format = "%H:%M:%S"), "%H:%M:%S")

st = st[st$date <= specified_date & st$time %in% times,]


#kernel function

gaussianKernel = function(x_diff, h) {

  return(exp(-(x_diff / h)^2))

}


#function to calculate the distance between the point to predict and the station coordinates

calc_distance_diff = function() {

  stations_coordinates = matrix(c(st$longitude, st$latitude), ncol = 2)

  POI = matrix(c(b,a), ncol = 2)

  dist_diff = distHaversine(stations_coordinates, POI)

  return(dist_diff)

}


#Function to calculate the difference between the day to predict and the day of the
measurement

calc_date_diff = function() {

  day_diff = as.numeric(difftime(date, st$date, units = "days")) %% 365

  return(day_diff)
```

```
}


#Function to calculate the difference between hours to predict and the hours the
measurements were taken

calc_hour_diff = function(hour) {

  hour_diff = abs(as.numeric(difftime(strptime(times[hour], format =
"%H:%M:%S"),strptime(cbind(st$time), format = "%H:%M:%S"), units = "hours")))

  return(hour_diff)

}



#function to plot kernel values against distance/days/hours

plot_kernel_vs_distance = function(h_value, against, xlim_value, v_value) {

  kernel_value = gaussianKernel(against, h_value)

  plot(against, kernel_value, type = "p", main = "Kernel Value vs Distance", xlab = "Distance",
ylab = "Kernel Value", col = "black", cex = 0.5, xlim =c(0,xlim_value))

  abline(h = 0.5, col = 'red', lty = 2)

  abline(v = v_value, col = 'red', lty = 2)

}



#some h_values to test for distance

h_values = c(10000, 20000, 50000, 80000, 100000, 120000)

for(h_value in h_values) {

  plot_kernel_vs_distance(h_value, calc_distance_diff(), 300000, 100000)

}
```

```
#some h_values to test for days
h_values = c(2, 5, 7, 8, 9, 10)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_date_diff(), 20, 7)
}

#some h_values to test for hours
h_values = c(2, 4, 6, 8, 10, 12)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_hour_diff(), 15, 5)
}

#h_values final
h_distance = 120000
h_date = 8
h_time = 6

#date & distance Kernels
date_kernel = gaussianKernel(calc_date_diff(), h_date)
dist_kernel = gaussianKernel(calc_distance_diff(), h_distance)

#temperature vectors
temp_sum = vector(length=length(times))
```

```
temp_prod = vector(length=length(times))


#looping through the hours in the times vector

for(hour in seq_along(times)) {

  hour_diff = calc_hour_diff(hour)

  hours_kernel = gaussianKernel(hour_diff, h_time)


  #Sum and product of the kernels

  kernel_sum = dist_kernel + date_kernel + hours_kernel

  kernel_prod = dist_kernel * date_kernel * hours_kernel


  #Add the temperatures to the temperature vectors

  temp_sum[hour] = sum(kernel_sum %*% st$air_temperature) / sum(kernel_sum)

  temp_prod[hour] = sum(kernel_prod %*% st$air_temperature) / sum(kernel_prod)

}


#Plotting temperatures for both sum and prod kernels

plot(temp_sum, type="o", main = "Temperature prediction", xlab = "Time of day (h)", ylab =
"Temperature (°C)", col = "blue", ylim = c(-4, 6), xaxt="n")

lines(temp_prod, type="o", col = "red")

axis(1, at = seq_along(times),  labels=substring(times, 1, 2))

legend("bottomright", c("Sum. kernel", "Prod. kernel"), col = c("blue", "red"),  pch=1, lty=1,
box.lty=1, cex=0.8)
```

—----------Assignment 2—----------

```
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])
train <- spam[1:3000, ]
val <- spam[3001:3800, ]
trainva <- spam[1:3800, ]
test <- spam[3801:4601, ]

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

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

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

filter2 <-
ksvm(type~.,data=trainva,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scal
ed=FALSE)
mailtype <- predict(filter2,test[,-58])
t <- table(mailtype,test[,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,scale
d=FALSE)
mailtype <- predict(filter3,test[,-58])
t <- table(mailtype,test[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3

# Questions

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

# We return filter3 since it uses all of the data in the model.

# 2. What is the estimate of the generalization error of the filter returned to the user? err0,
err1, err2 or err3? Why?

# The error that should be returned to the user should be error2. This model uses trainva as
training
# and test for the predicitons. These two are separated. In filter3 for example the data and
prediction
# is interconnected, which is not optimal.


# 3. Implementation of SVM predictions.

sv <- alphaindex(filter3)[[1]]
support.vectors <- spam[sv, -58]
co <- coef(filter3)[[1]]
inte <- - b(filter3)

#  Smaller sigma value -> narrower Gaussian function
kernel.function <- rbfdot(0.05)

k <- NULL
dot.products <- NULL

# Predictions for the first 10 points
for (i in 1:10) {

  k2 <- NULL

  for (j in seq_along(sv)) {
    k2 <- unlist(support.vectors[j, ])
    sample <- unlist(spam[i, -58])
    # Add the dot product to the existing ones
```

```
    dot.products <- c(dot.products, kernel.function(sample, k2))
  }
  # Start and end index
  start <- 1 + length(sv) * (i - 1)
  end <- length(sv) * i

  # Calculate predictions and add them
  prediction <- co %*% dot.products[start:end] + inte
  k <- c(k, prediction)
}

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

——————————————————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(10, -1, 1)

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

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, main="sigmoid", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)

### TASK 2 ###

# Custom activation function: h1(x) = x
linear <- function(x) x
nn_linear <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights = winit, act.fct = linear)
```

```
# Plot same way as earlier
plot(tr, main = "linear", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_linear,te), col="red", cex=1)
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)

# Custom activation function: h2(x) = max{0, x}
ReLU <- function(x) ifelse(x>0, x, 0)
nn_ReLU <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights = winit, act.fct = ReLU)

# Plot same way as earlier
plot(tr, main = "ReLU", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_ReLU,te), col="red", cex=1)
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)

# Custom activation function: h3(x) = ln(1 + exp x)
softplus <- function(x) log(1 + exp(x))
nn_softplus <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights = winit, act.fct =
softplus)

# Plot same way as earlier
plot(tr, main = "softplus", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_softplus,te), col="red", cex=1)
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)

### TASK 3 ###

set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata_50 <- data.frame(Var, Sin=sin(Var))

plot(mydata_50, cex = 2, ylim = c(-10, 2))
points(mydata_50[, 1], predict(nn, mydata_50), col = "red", cex = 1)
legend("topright", legend = c("Training", "Predictions"),
       col = c("black", "red"), pch = 16)

### TASK 4 ###
plot(nn)
nn$weights

### TASK 5 ###
```

```
set.seed(1234567890)

Var_inverse <- runif(500, 0, 10)

mydata_inverse <- data.frame(Sin_inverse=sin(Var_inverse), Var_inverse)

nn_inverse <- neuralnet(Var_inverse ~ Sin_inverse, data = mydata_inverse, hidden = 10,
threshold = 0.1)

plot(mydata_inverse, main="inverse", cex=2)
points(mydata_inverse[, 1],predict(nn_inverse, mydata_inverse), col="red", cex=1)

legend("topright", legend = c("Training Data", "Predictions"),
       col = c("black", "red"), pch = 16)
```