Machine Learning Lab Report 03

Daisuke Ronald Ssegwanyi Yamashita (ronss490)
Greeshma Jeev Koothuparambil(greko370) Johan Lundin (johlu023)
Sangeeth Sankunny Menon(sansa237)"

2023-12-16

Statement of Contribution

Assignment 1. Kernel Methods was done by Sangeeth Sankunny Menon(sansa237)

Assignment 2. Support Vector Machines - Johan Lundin (johlu023) was responsible for the code, answers and layout for assignment 2 in collaboration with Daisuke Ronald Ssegwanyi Yamashita (ronss490).

Assignment 3. Neural Networks was done by Greeshma Jeev Koothuparambil(greko370)

Assignment 1. KERNEL METHODS

Implement a kernel method to predict the hourly temperatures for a date and place in Sweden.To do so, you are provided with the files stations.csv and temps50k.csv. These files contain information about weather stations and temperature measurements in the stations at different days and times. The data have been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI). You are 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. Use a kernel that is the sum of three Gaussian kernels:

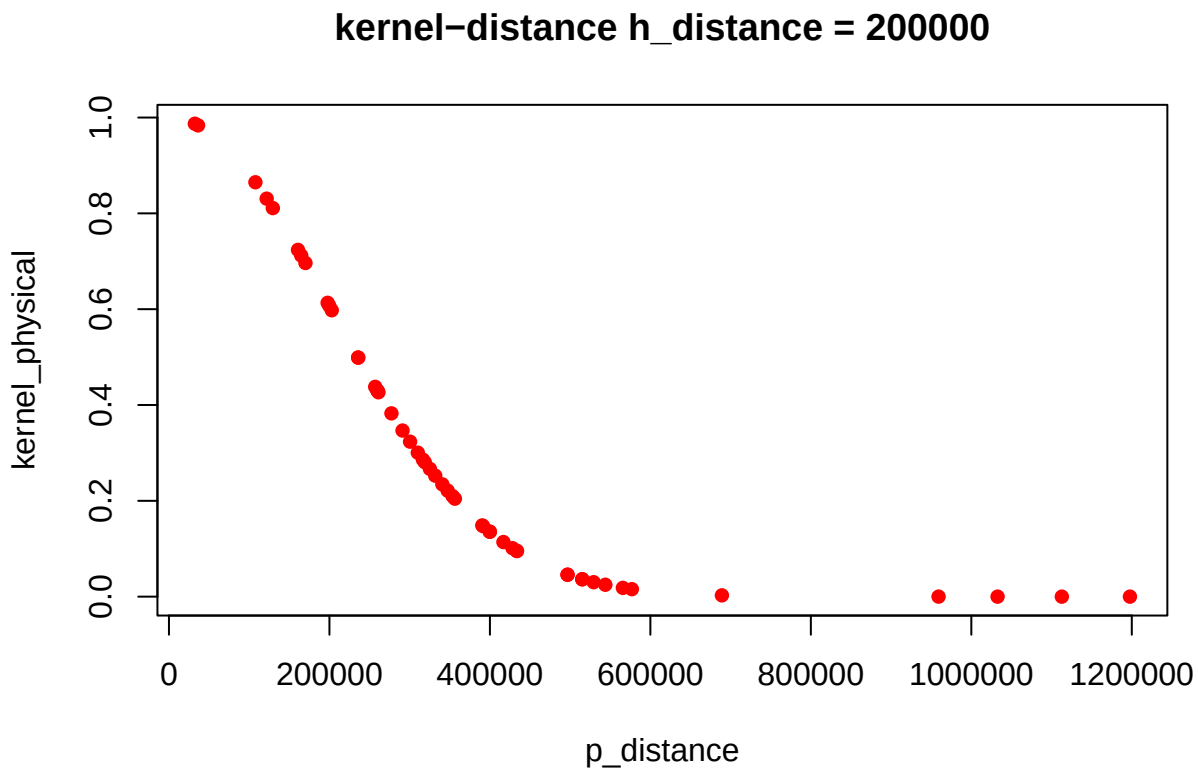
1.The first to account for the physical distance from a station to the point of interest. For this purpose, use the function distHaversine from the R package geosphere.
2. The second to account for the distance between the day a temperature measurement was made and the day of interest.
3. The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Choose an appropriate smoothing coefficient or width for each of the three kernels above. No cross-validation should be used. Instead, choose manually a width that gives large kernel values to closer points and small values to distant points. Show this with a plot of the kernel value as a function of distance. Help: Note that the file temps50k.csv may contain temperature measurements that are posterior to the day and hour of your forecast. You must filter such measurements out, i.e. they cannot be used to compute the forecast. Finally, repeat the exercise above by combining the three kernels into one by multiplying them, instead of summing them up. Compare the results obtained in both cases and elaborate on why they may differ. The only R package that is allowed to solve this assignment is the geosphere package (specifically, the function distHaversine).

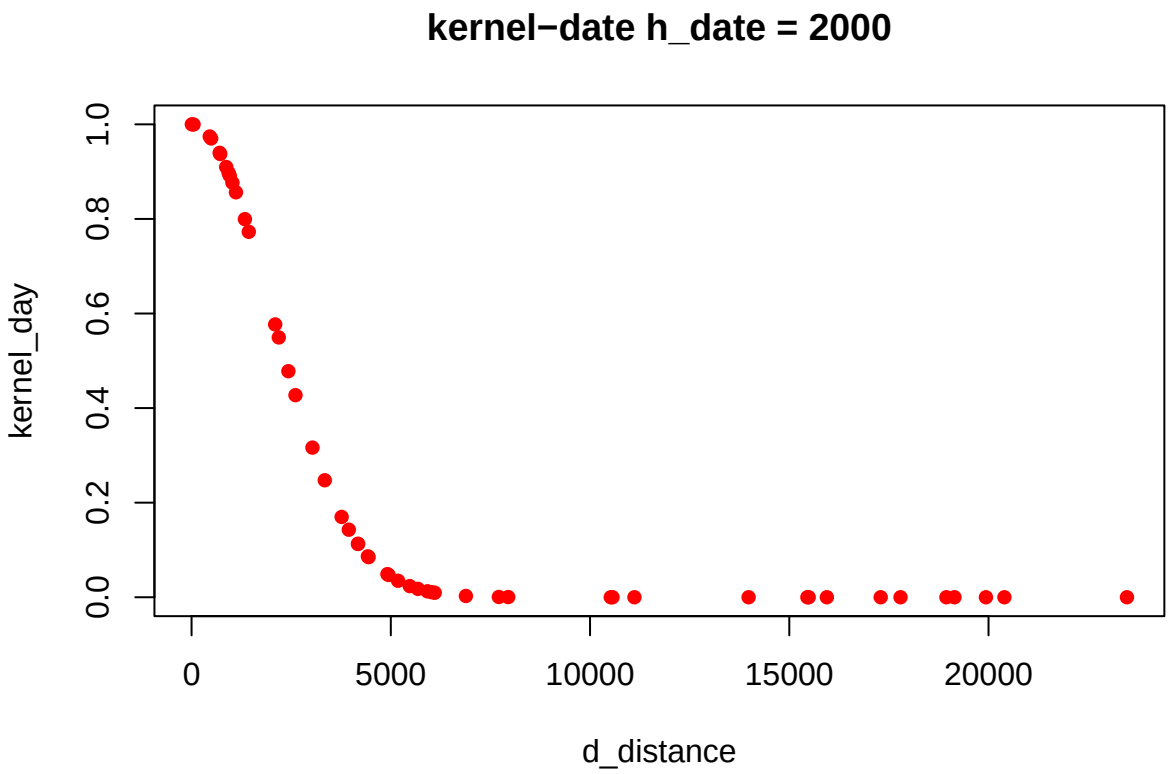
The widths chosen are 200000 for h_distance , 2000 for h_date and 2 for h_time for each of the kernels after trail and error.

Plots of the three kernel value as a function of respective distances

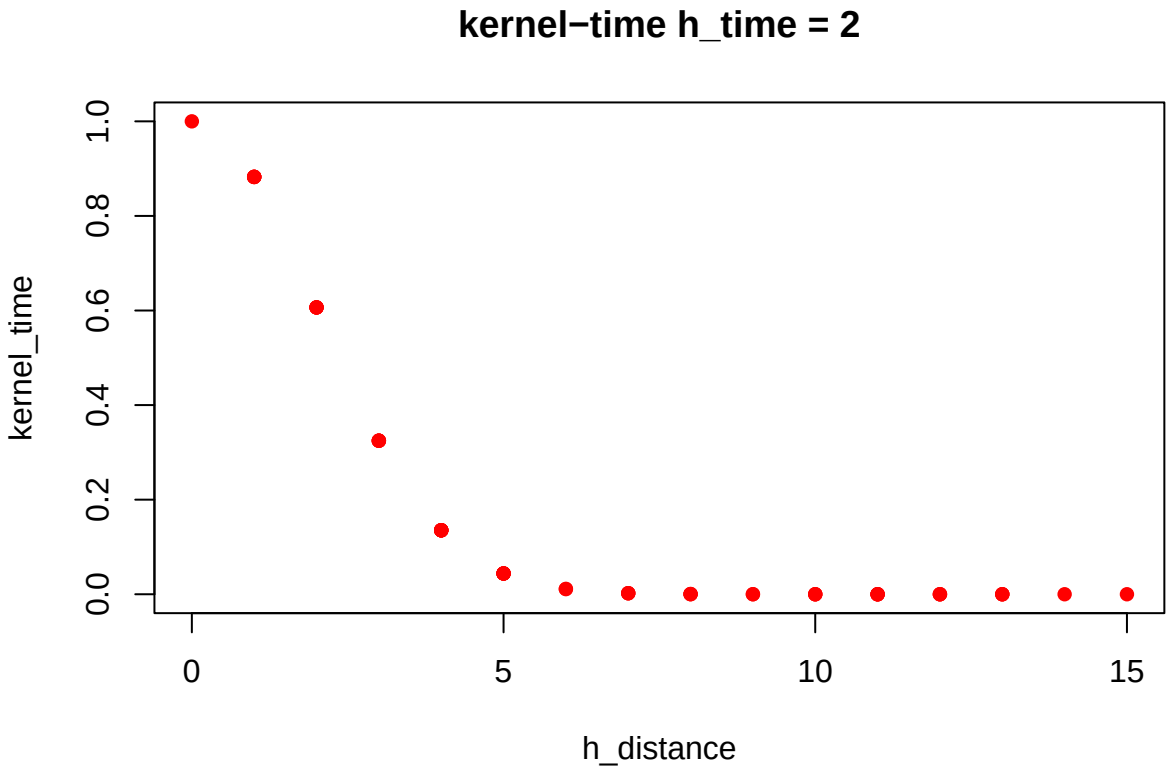
```
plot(p_distance,kernel_physical,col = "red",pch = 16,main = "kernel-distance h_distance = 200000")
```



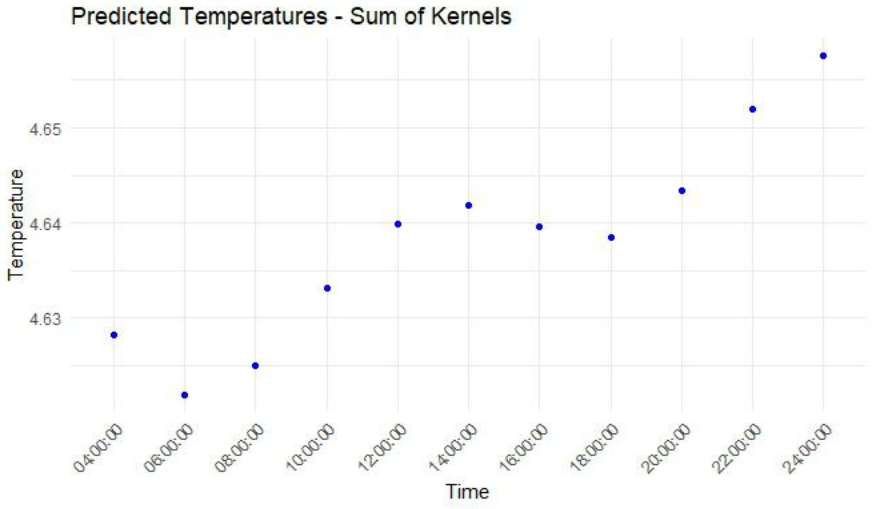

```
plot(d_distance,kernel_day,col = "red",pch = 16,main = "kernel-date h_date = 2000")
```

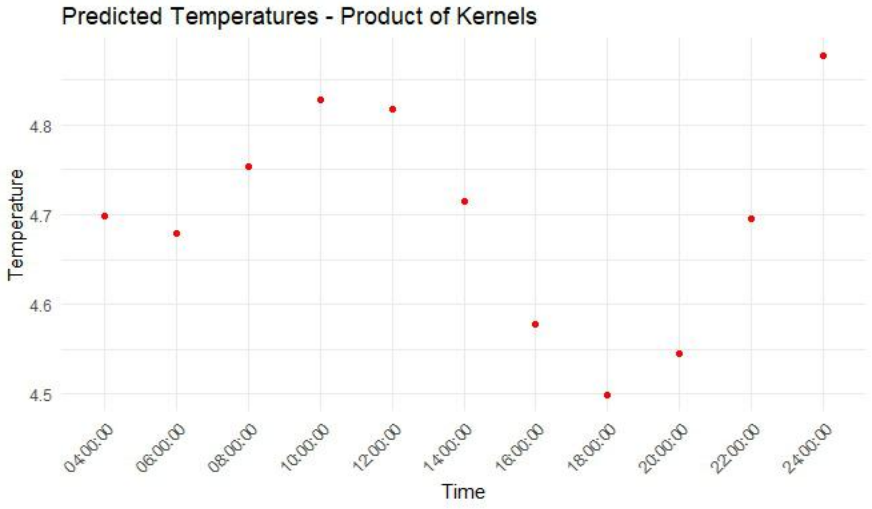



```
plot(h_distance,kernel_time,col = "red",pch = 16,main = "kernel-time h_time = 2")
```


kernel−time h_time = 2
kernel_time
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
h_distance

Results obtained in both cases - Summing Kernels vs Multiplying Kernels





The comparison between the approaches of summing kernels and multiplying kernels in the context of predicting temperatures using Gaussian kernels:

Summation of Kernels Approach assumes an additive relationship among the kernels, where they contribute independently to the final prediction without considering potential interactions or dependencies between them. However, Multiplication of Kernels Approach emphasizes joint effects by multiplying the kernel weights, giving higher significance to cases where all three factors align with lower values.

Assignment 2. SUPPORT VECTOR MACHINES

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

It seems like all four filters are using the same model and settings, the only thing that differs between the models is how much of the data is used for training. So if the user is going to get one of these filters why not use all the data (assuming all of the data is of the same quality), i.e. selecting filter3.

However, if we would just take all the training data (filter3) we would not get an idea on how the model performs on unseen data.

2) What is the estimate of the generalization error of the filter returned to the user? Why?

Filter2 uses the training and validation data and then predict values for the test data (unseen data). So it uses as many data points as possible without using data used for training for testing. So err2 would probably be a good approximation how filter3 (all datapoints) would perform on unseen data.

3) Once a SVM has been fitted to the training data, a new point is essentially classified according to the sign of a linear combination of the kernel function values between the support vectors and the new point. You are asked to implement this linear combination for filter3. You should make use of the functions alphaindex, coef and b that return the indexes of the support vectors, the linear coefficients for the support vectors, and the negative intercept of the linear combination.

```
# 3. Implementation of SVM predictions.

n.points = 10      # Number of Data-points to compare (10 according to the instructions)
l = 0.05           # Width of kernel (given in lab instructions)

# Vectors generated from kusm-model filter3
sv<-alphaindex(filter3)[[1]]  # The index of the resulting support vectors in the data matrix
co<-coef(filter3)[[1]]        # Corresponding coefficients times the training labels
inte<- - b(filter3)           # The negative intercept
k<-NULL

# We produce predictions for n.points (first  10 points) in the data-set

for(i in 1:n.points){
```

```
  k2<-NULL
  #print(i)
  for(j in 1:length(sv)){

    x =   spam[ sv[j],-58]
    x.prim =  spam[i,-58]

    distance = sum( (x-x.prim)^2)   #Distance between x and x.prim

    # Gaussian kernel (this is what was used in ksvm-model-function according to
    # the assignment instructions, since we are to compare the results it makes
    # sense to use the same kernel function (?)

    # k2.new = exp(- distance / (2*l^2))
    k2.new = exp(- distance * 1 )

    k2 = c(k2, k2.new)
  }

  k<-c(k, t(co) %*% k2 + inte)

}

k
```

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

```
k.m = predict(filter3,spam[1:n.points,-58], type = "decision")

t(k.m)
```

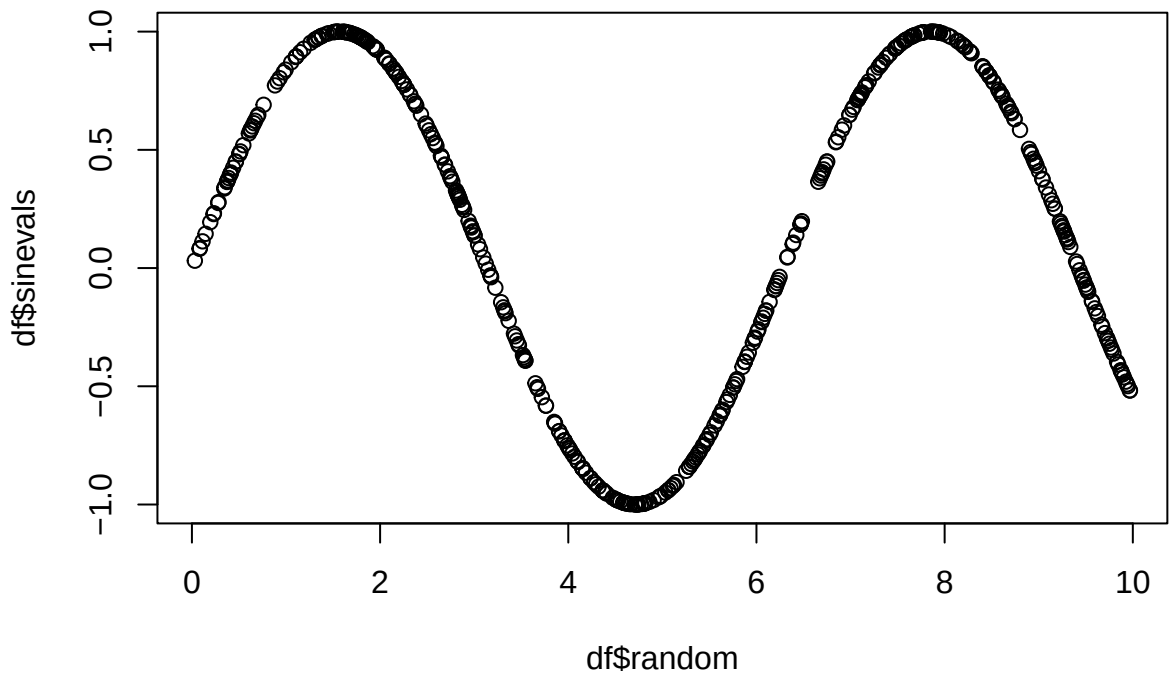
```
##           [,1]     [,2]     [,3]      [,4]      [,5]     [,6]      [,7]
## [1,] -1.998999 1.560584 1.000278 -1.756815 -2.669577 1.291312 -1.068444
##           [,8]     [,9]     [,10]
## [1,] -1.312493 1.000184 -2.208639
```

Assignment 3. NEURAL NETWORK

1. Train a neural network to learn the trigonometric sine function. To do so, sample 500 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting value pairs are the data points available to you. Use 25 of the 500 points for training and the rest for test. Use one hidden layer with 10 hidden units. You do not need to apply early stopping. Plot the training and test data, and the predictions of the learned NN on the test data. You should get good results. Comment your results.

```
library(neuralnet)
library(ggplot2)
```

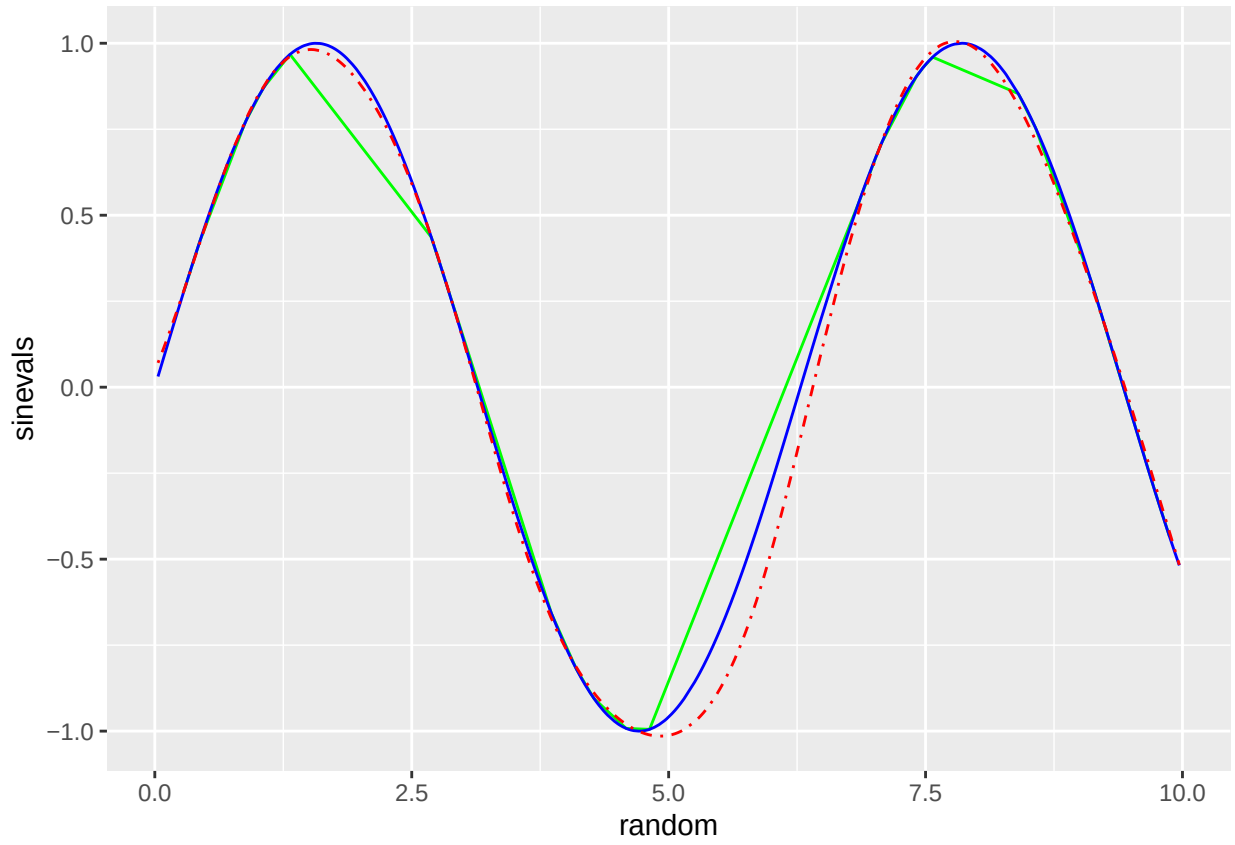
```r
#1.a

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

#Applying sine function
sinevals <- sin(random)

df <- data.frame(random, sinevals)
plot(df$random, df$sinevals)
```




```r
#generating train and test data
train <- df[1:25,]
test <- df[-(1:25),]

predictiondf <- test

netmodel <- neuralnet(sinevals~ random, data = train, hidden = 10)
predictiondf$predictnet1 <- predict(netmodel, newdata = test)

meansquarederror <- mean((predictiondf$sinevals-predictiondf$predictnet1)^2)
```

The plot looks as follow:

Green line represents the train data, blue represents the test data while red dotdash line represents the predictions



Analysis

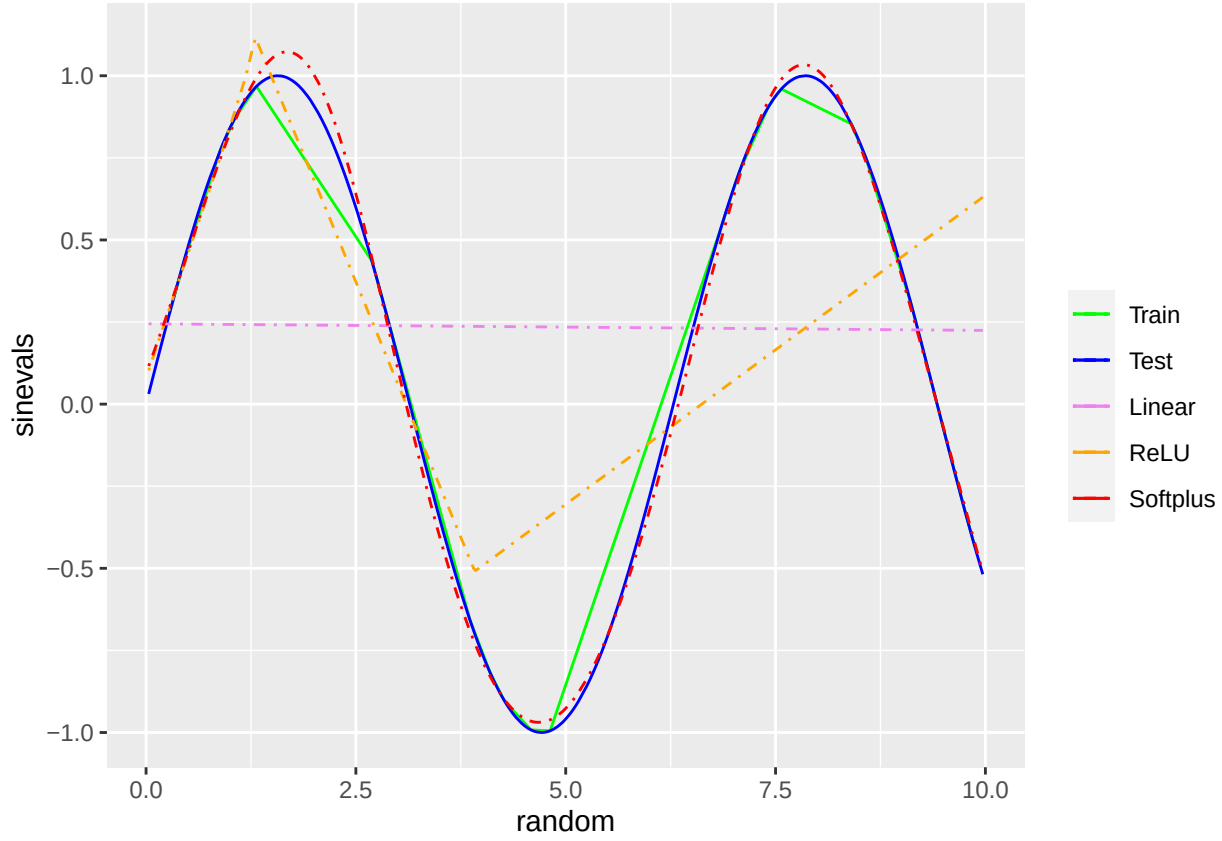
The test data is not smooth enough to be a sinusoidal curve. It can be seen that it deviates from the natural curve. This is due to the lack of number of observations. But it is clear that even with small number of training data the model was able to predict a good result on the test data. The predictions look almost similar to the test data. It follows the sinusoidal curve efficiently. The mean squared error comes upto 0.0045079 .

2. In question (1), you used the default logistic (a.k.a. sigmoid) activation function, i.e. act.fct = "logistic". Repeat question (1) with the following custom activation functions: h 1 (x) = x, h 2 (x) = max{0, x} and h 3 (x) = ln(1 + exp x) (a.k.a. linear, ReLU and softplus). See the help file of the neuralnet package to learn how to use custom activation functions. Plot and comment your results.

```
set.seed(12345)
linear <- function(x){
  x
}

ReLU <- function(x){
  ifelse(x>0, x, 0)
}

softplus <- function(x){
    log(1+exp(x))
```

```
}

netmodel_linear <- neuralnet(sinevals~ random, data = train, hidden = 10, act.fct = linear)
predictiondf$predictnetlinear <- predict(netmodel_linear, newdata = test)

netmodel_ReLU <- neuralnet(sinevals~ random, data = train,  hidden = 10, act.fct = ReLU)
predictiondf$predictnetReLU <- predict(netmodel_ReLU, newdata = test)

netmodel_softplus <- neuralnet(sinevals~ random, data = train, hidden = 10, act.fct = softplus)
predictiondf$predictnetsoftplus <- predict(netmodel_softplus, newdata = test)


meansquarederrorL <- mean((predictiondf$sinevals-predictiondf$predictnetlinear)^2)
meansquarederrorR <- mean((predictiondf$sinevals-predictiondf$predictnetReLU)^2)
meansquarederrorS <- mean((predictiondf$sinevals-predictiondf$predictnetsoftplus)^2)
```

The mixed plot is as given below:



Analysis

The resulting graph is quite interesting as they give insights on the variation in predictions of the neural networks according to the activation algorithm. The network which used the linear function as the activation function gives a straight line which is similar to the nature of the activation function. The ReLU activation made an interesting trend to the prediction. Till random values to be almost 3.75 it follows the train data

curve but after that it shows an increasing linear trend. While the softplus function gives a similar trend as the test data.

The mean squared error was calculated for the models for further analysis and is as follows:

models	errors
logistic	0.0045079
linear	0.4467241
ReLU	0.2049228
Softplus	0.0015883

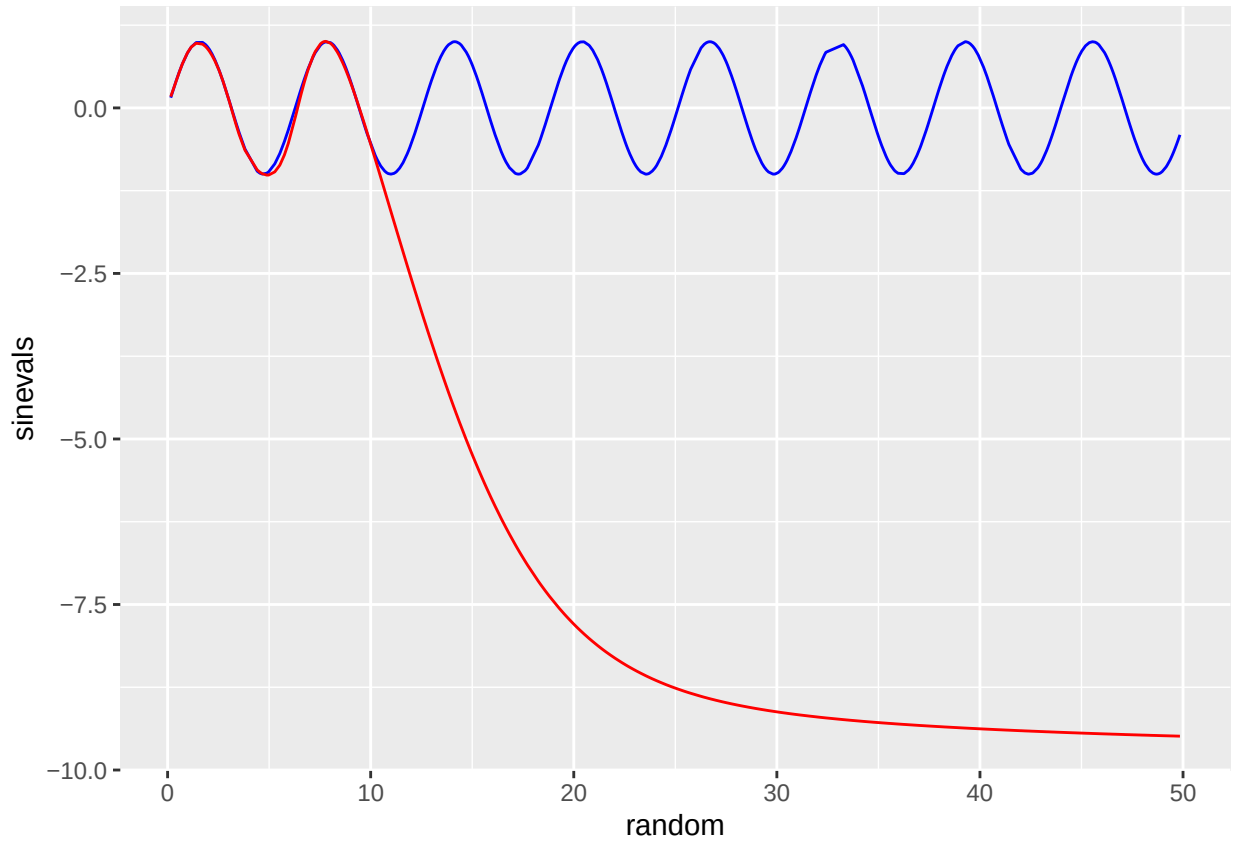
The table is giving means square error for all the models trained. It can also be seen that the MSE for Softplus activation function is less than for the logistic activation function

3. Sample 500 points uniformly at random in the interval [0, 50], and apply the sine function to each point. Use the NN learned in question (1) to predict the sine function value for these new 500 points. You should get mixed results. Plot and comment your results.

```
set.seed(1234567890)
random <- runif(500, 0, 50)

#Applying sine function
sinevals <- sin(random)
df2 <- data.frame(random, sinevals)
df2$predictnet1 <- predict(netmodel, newdata = df2)
```

The predicted values can be projeted as follows:

Blue is is the original data and red is the predicted data



Analysis

The predictions follow the test curve till the values for which it was trained for that is in the range [0,10]. After that the predictions went wrong with values following the downward trend of the test data and eventually summing up to be logistic curve.

4.In question (3), the predictions seem to converge to some value. Explain why this happens. To answer this question, you may need to get access to the weights of the NN learned. You can do it by running nn or nn$weights where nn is the NN learned.

The weights to first hidden layer is given as:

3.965406, -1.8484966, -1.2313394, -1.10816, -2.7255077, 0.2616825, 7.3872163, -2.2981544, -0.4392372, -0.0561888, 0.8143266, -1.9677321, 16.9497283, -2.5102522, -0.8458926, -1.1041865, 2.4703284, -0.7615488, -11.6184909, 1.7558853

and the weights to the second hidden layer is given as :

-0.8272835, 0.6261725, -1.3652234, -15.2546738, 1.4724743, 4.0194898, -2.370571, 1.522523, 1.3549643, -1.9815383, 6.4412888

Analysis

The network weights are distributed according to the activation function which is logistic function. The function tends to reach its optima when the initial weights are positives. Since the random values greater

than 10 is unknown for the network is will tend to converge itself to the optimum value. the optimum can be obtained through the following formula:

```
sum(netmodel$weights[[1]][[2]][which(netmodel$weights[[1]][[1]][2,]>0)+1])+
netmodel$weights[[1]][[2]][1]
```

```
## [1] -9.640669
```

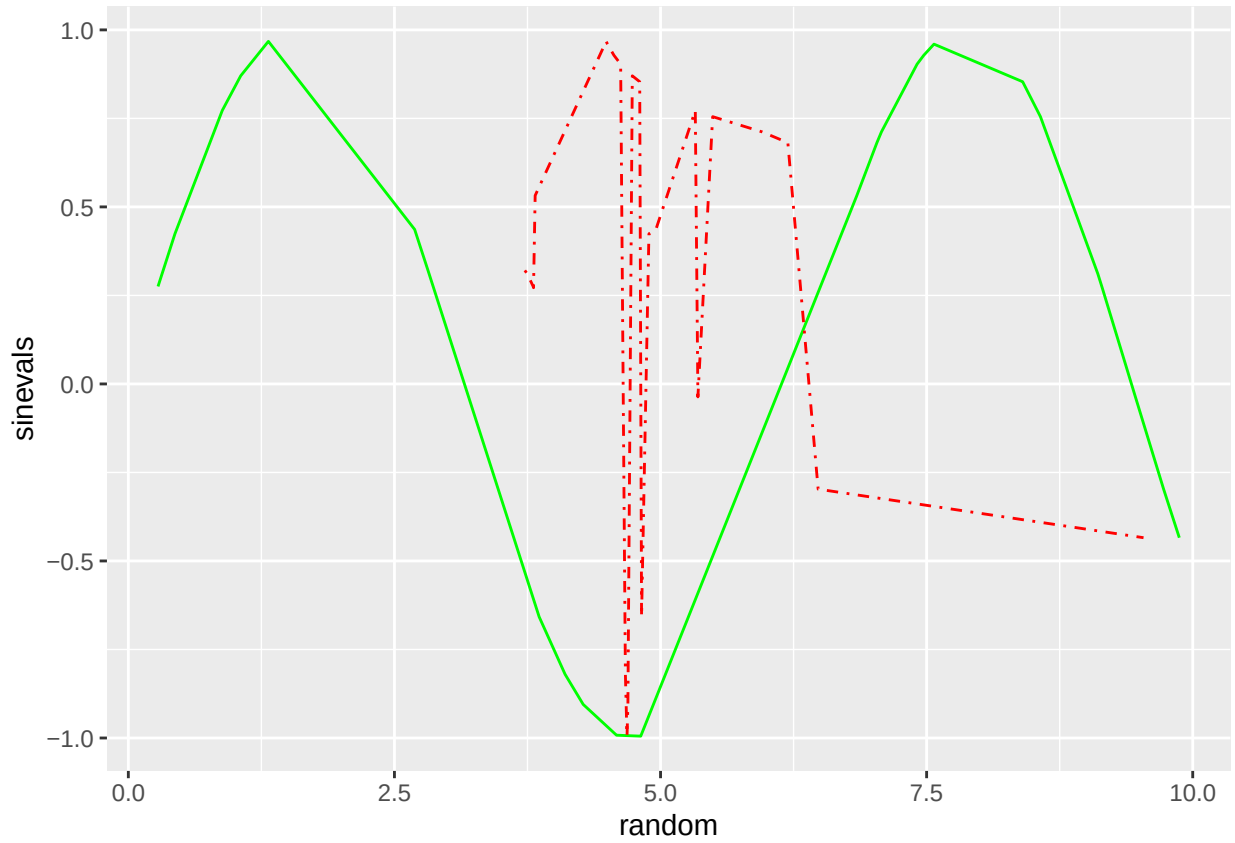
We can also see from the graph that the function tends to converge to its optima around the value -9.6.

5. Sample 500 points uniformly at random in the interval [0, 10], and apply the sine function to each point. Use all these points as training points for learning a NN that tries to predict x from sin(x), i.e. unlike before when the goal was to predict sin(x) from x. Use the learned NN to predict the training data. You should get bad results. Plot and comment your results. Help: Some people get a convergence error in this question. It can be solved by stopping the training before reaching convergence by setting threshold = 0.1.

```
#1.5
set.seed(1234567890)
random <- runif(500, 0, 10)

#Applying sine function
sinevals <- sin(random)

df3 <- data.frame(random, sinevals)


#generating train and test data

predictiondf2 <- train

netmodel_reverse <- neuralnet(random~ sinevals, data = df3, hidden = 10, threshold = 0.1)
predictiondf2$predictnetreverse <- predict(netmodel_reverse, newdata = train)

meansquarederrorreverse <- mean((predictiondf2$random-predictiondf2$predictnetreverse)^2)
```

The reverse modeling resulted in a graph as follows:

Green is the train data and red represents the predicted random values.



Analysis

The prediction is very poor. The random values predicted ranges from 3.72 to 9.53 instead of 0 to 10. The predicted random values do not give a good sine curve as well. The mean squared error is 9.0801808.

APPENDIX

Assignment 1

```
set.seed(1234567890)
library(geosphere)
library(ggplot2)
# Read data
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")

# Merge station and temperature data
st <- merge(stations, temps, by = "station_number")

# Set bandwidths for the three Gaussian kernels
h_distance <- 200000  # Adjust this value based on distance considerations
h_date <- 2000        # Adjust this value based on date differences
h_time <- 2        # Adjust this value based on hour differences

# Point to predict
a <- 58.4274  # Latitude of the point
b <- 14.826   # Longitude of the point

# Date to predict
date <- "2013-11-04"  # Date for prediction

# Times for prediction
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")

# Initialize empty vector for predicted temperatures
temp <- vector(length = length(times))

# Filter out measurements posterior to the forecast date and time
forecast_date <- as.Date("2013-11-04")  # Convert the forecast date to Date format
forecast_time <- "04:00:00"

st_filtered <- subset(st, as.Date(date) < forecast_date | (as.Date(date) ==
                                forecast_date & time <= forecast_time))
st_filtered[,4:5]

new_index <- sample(nrow(st_filtered))
new_data <- st[new_index,]
data_shown <- new_data[1:50,]

physical_dist <- function(pt1,pt2){
  return(distHaversine(pt1,pt2))
}

num<- nrow(data_shown)

new_data_1 <-  c(" 58.4274", "14.826", "2013-11-04","08:00:00")
```

```r
p_distance <- NULL
for(i in 1:50){
  distance_temp <- physical_dist(as.numeric(new_data_1[1:2]),data_shown[i,4:5])
  print(distance_temp)
  p_distance <- c(p_distance,distance_temp)
}
length(p_distance)

order_ascend <- order(p_distance)
data_distance_1 <- data_shown[order_ascend,]

kernel_physical <- NULL

for (i in 1:50) {
  kernel_distance_1_temp <- exp(-p_distance^2 / (2 * h_distance^2))

  kernel_physical <- c(kernel_distance_1_temp)

}

plot(p_distance,kernel_physical,col = "red",pch = 16,main = " h_distance")


# for second width

# Function to calculate the difference in days between two dates
days_distance <- function(date1, date2) {
  diff_days <- as.numeric(difftime(date2, date1, units = "days"))
  return(abs(diff_days))
}

d_distance <- NULL
for(i in 1:50){
  day_temp <- days_distance(new_data_1[3],data_shown[i,9])
  d_distance <- c(d_distance,day_temp)
}

order_ascend <- order(d_distance)
data_distance_2 <- data_shown[order_ascend,]

kernel_day <- NULL

for (i in 1:50) {
  kernel_distance_2_temp <- exp(-d_distance^2 / (2 * h_date^2))
  kernel_day <- c(kernel_distance_2_temp)
}

plot(d_distance,kernel_day,col = "red",pch = 16,main = "h_date")


# for third width

# Function to calculate the difference in hours between two times
```

```r
hours_distance <- function(time1, time2) {
  # Extract hour components from the times
  hour1 <- as.numeric(substr(time1, 1, 2))
  hour2 <- as.numeric(substr(time2, 1, 2))

  # Calculate the absolute difference in hours
  diff_hours <- abs(hour2 - hour1)
  return(diff_hours)
}

h_distance <- NULL
for(i in 1:50){
  time_temp <- hours_distance(new_data_1[4],data_shown[i,10])
  h_distance <- c(h_distance,time_temp)
}
length(h_distance)
order_ascend <- order(abs(h_distance))
data_distance_3 <- data_shown[order_ascend,]

kernel_time <- NULL

for (i in 1:50) {
  kernel_distance_3_temp <- exp(-h_distance^2 / (2 * h_time^2))
  kernel_time <- c(kernel_distance_3_temp)
}

plot(h_distance,kernel_time,col = "red",pch = 16,main = "h_time")



predicted_temps_summed <- vector(length = length(times))
predicted_temps_multiplied <- vector(length = length(times))

# Calculate the weights for each kernel for the given point, date, and times
for (j in 1:length(times)) {
  # Calculate the weights for each kernel for the current time
  weights_sum <- NULL
  weights_mul <- NULL

  for (i in 1:nrow(st_filtered)) {
    # Calculate distances for each kernel
    distance_physical <- physical_dist(c(a, b), st_filtered[i, c("latitude", "longitude")])
    distance_date <- days_distance(date, st_filtered[i, "date"])
    distance_time <- hours_distance(times[j], st_filtered[i, "time"])

    # Calculate weights for each kernel using Gaussian kernel formula
    weight_distance <- exp(-distance_physical^2 / (2 * h_distance^2))
    weight_date <- exp(-distance_date^2 / (2 * h_date^2))
    weight_time <- exp(-distance_time^2 / (2 * h_time^2))

    # Store weights for each kernel
    weights_sum <- c(weights_sum, weight_distance + weight_date + weight_time)
    weights_mul <- c(weights_mul, weight_distance * weight_date * weight_time)
```

```
  }

  # Use weights to predict temperature by summing and multiplying kernels
  predicted_temps_summed[j] <- sum(weights_sum * st_filtered[,"air_temperature"])/ sum(weights_sum)

  predicted_temps_multiplied[j] <- sum(weights_mul*st_filtered[,"air_temperature"]) / sum(weights_mul)

}

day_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")


# Create data frames for plotting
data_summed <- data.frame(Time = day_times, Temperature = predicted_temps_summed)
data_multiplied <- data.frame(Time = day_times, Temperature = predicted_temps_multiplied)

# Convert 'day_times' to a factor to maintain order on the x-axis
data_summed$Time <- factor(data_summed$Time, levels = day_times)
data_multiplied$Time <- factor(data_multiplied$Time, levels = day_times)

# Plot using ggplot for summed kernels
plot_summed <- ggplot(data_summed, aes(x = Time, y = Temperature)) +
  geom_point(color = "blue") +
  labs(x = "Time", y = "Temperature", title = "Predicted Temperatures - Sum of Kernels") +


# Plot using ggplot for multiplied kernels
plot_multiplied <- ggplot(data_multiplied, aes(x = Time, y = Temperature)) +
  geom_point(color = "red") +
  labs(x = "Time", y = "Temperature", title = "Predicted Temperatures - Product of Kernels") +


# Show the plots
plot_summed
plot_multiplied
```

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 = FALSE)
mailtype <- predict(filter3, te[, -58])
t <- table(mailtype, te[, 58])
err3 <- (t[1, 2] + t[2, 1])/sum(t)
err3
```

```
# 3. Implementation of SVM predictions.

n.points = 10  # Number of Data-points to compare (10 according to the instructions)
l = 0.05  # Width of kernel (given in lab instructions)

# Vectors generated from kvsm-model filter3
sv <- alphaindex(filter3)[[1]]  # The index of the resulting support vectors in the data matrix
co <- coef(filter3)[[1]]  # Corresponding coefficients times the training labels
inte <- -b(filter3)  # The negative intercept
k <- NULL

# We produce predictions for n.points (first 10 points) in the data-set
```

```
for (i in 1:n.points) {
    k2 <- NULL
    # print(i)
    for (j in 1:length(sv)) {

        x = spam[sv[j], -58]
        x.prim = spam[i, -58]

        distance = sum((x - x.prim)^2)  #Distance between x and x.prim

        # Gaussian kernel (this is what was used in ksvm-model-function
        # according to the assignment instructions, since we are to compare the
        # results it makes sense to use the same kernel function (?)

        # k2.new = exp(- distance / (2*l^2))
        k2.new = exp(-distance * l)

        k2 = c(k2, k2.new)
    }

    k <- c(k, t(co) %*% k2 + inte)

}

k

k.m = predict(filter3, spam[1:n.points, -58], type = "decision")

t(k.m)
```

Assignment 3

```
library(neuralnet)
library(ggplot)


#1.a
set.seed(1234567890)
random <- runif(500, 0, 10)

#Applying sine function
sinevals <- sin(random)

df <- data.frame(random, sinevals)
plot(df$random, df$sinevals)

#generating train and test data
train <- df[1:25,]
test <- df[-(1:25),]

predictiondf <- test
```

```
netmodel <- neuralnet(sinevals~ random, data = train, hidden = 10)
predictiondf$predictnet1 <- predict(netmodel, newdata = test)

meansquarederror <- mean((predictiondf$sinevals-predictiondf$predictnet1)^2)

ggplot()+
  geom_line(train, mapping = aes(x= random, y= sinevals), colour = "green")+
  geom_line(test, mapping = aes(x= random, y= sinevals), colour = "blue")+
  geom_line(predictiondf, mapping = aes(x= random, y= predictnet1),
            colour = "red", linetype = "dotdash")

#1.b

set.seed(12345)
linear <- function(x){
  x
}

ReLU <- function(x){
  ifelse(x>0, x, 0)
}

softplus <- function(x){
    log(1+exp(x))

}

netmodel_linear <- neuralnet(sinevals~ random, data = train, hidden = 10, act.fct = linear)
predictiondf$predictnetlinear <- predict(netmodel_linear, newdata = test)

netmodel_ReLU <- neuralnet(sinevals~ random, data = train,  hidden = 10, act.fct = ReLU)
predictiondf$predictnetReLU <- predict(netmodel_ReLU, newdata = test)

netmodel_softplus <- neuralnet(sinevals~ random, data = train, hidden = 10, act.fct = softplus)
predictiondf$predictnetsoftplus <- predict(netmodel_softplus, newdata = test)

ggplot()+
  geom_line(train, mapping = aes(x= random, y= sinevals, color = "Train"))+
  geom_line(test, mapping = aes(x= random, y= sinevals, color = "Test"))+
  geom_line(predictiondf, mapping = aes(x= random, y= predictnetlinear,
                                        color = "Linear"), linetype = "dotdash")+
  geom_line(predictiondf, mapping = aes(x= random, y= predictnetReLU,
                                        color = "ReLU"), linetype = "dotdash")+
  geom_line(predictiondf, mapping = aes(x= random, y= predictnetsoftplus,
                                        color = "Softplus"), linetype = "dotdash")+
  scale_colour_manual("", breaks = c("Train", "Test", "Linear", "ReLU", "Softplus"),
                      values = c( "green", "blue", "violet", "orange", "red"))

#1.3
set.seed(1234567890)
random <- runif(500, 0, 50)

#Applying sine function
```

```r
sinevals <- sin(random)
df2 <- data.frame(random, sinevals)
df2$predictnet1 <- predict(netmodel, newdata = df2)
ggplot()+
  geom_line(df2, mapping = aes(x= random, y= sinevals), colour = "blue")+
  geom_line(df2,mapping = aes(x= random, y= predictnet1), colour = "red")


#1.4
netmodel$weights[[1]][[1]][1,]
netmodel$weights[[1]][[2]]

sum(netmodel$weights[[1]][[2]][which(netmodel$weights[[1]][[1]][2,]>0)+1])+
netmodel$weights[[1]][[2]][1]

#1.5
set.seed(1234567890)
random <- runif(500, 0, 10)

#Applying sine function
sinevals <- sin(random)

df3 <- data.frame(random, sinevals)


#generating train and test data

predictiondf2 <- train

netmodel_reverse <- neuralnet(random~ sinevals, data = df3, hidden = 10, threshold = 0.1)
predictiondf2$predictnetreverse <- predict(netmodel_reverse, newdata = train)

meansquarederrorreverse <- mean((predictiondf2$random-predictiondf2$predictnetreverse)^2)
ggplot()+
  geom_line(train, mapping = aes(x= random, y= sinevals), colour = "green")+
  geom_line(predictiondf2, mapping = aes(x= predictnetreverse, y = sinevals),
            colour = "red", linetype = "dotdash")
```
