Lab 3 Report

Hugo Morvan & Marijn Jaarsma & Simon Jorstedt

2023-12-17

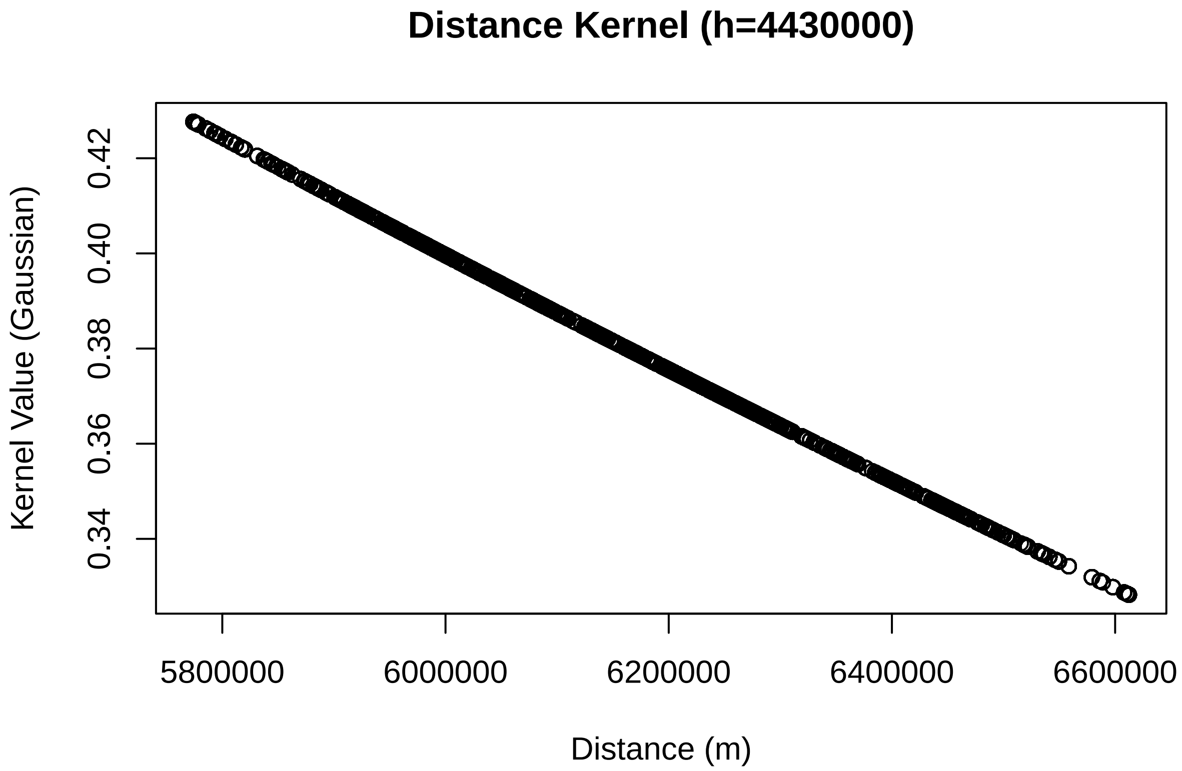
1. Kernel Methods

Setup

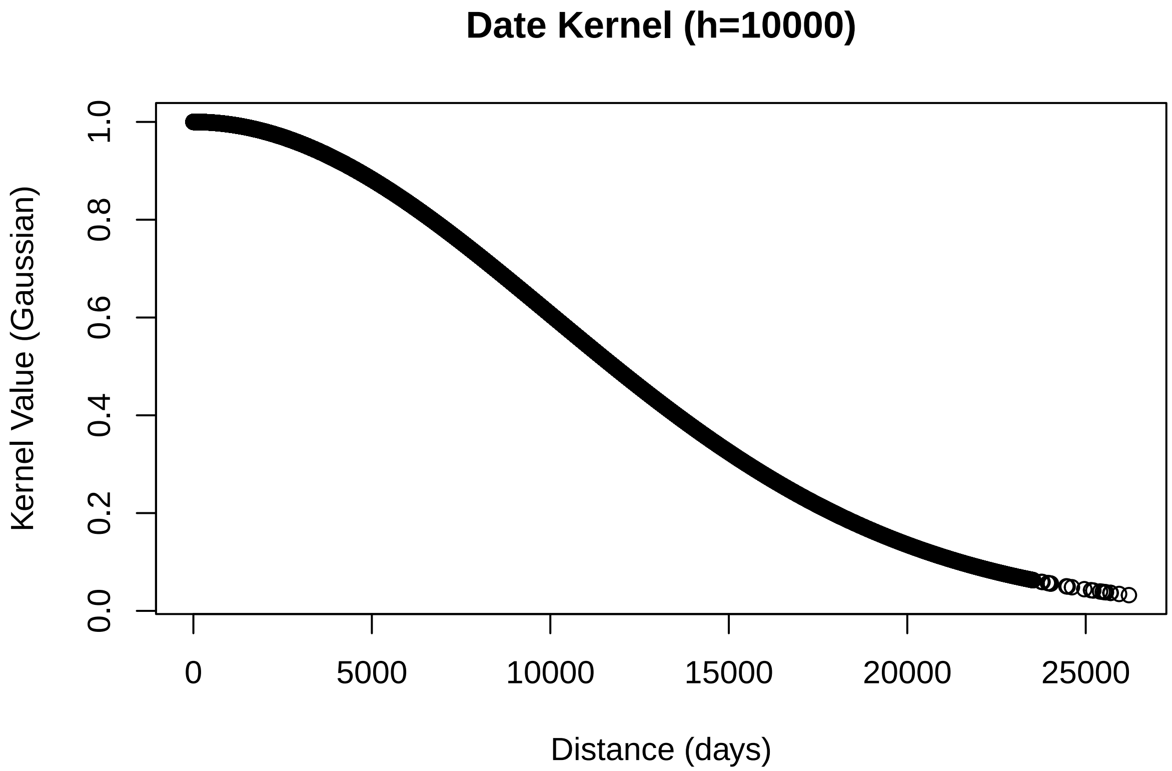
```
# Read in data and merge
df_stations <- read.csv("stations.csv", fileEncoding="latin1")
df_temps <- read.csv("temps50k.csv")
df_st <- merge(df_stations, df_temps, by="station_number")

# Define hyper parameters
h_distance <- 4430000
h_date <- 10000
h_time <- 8

# Prediction point
a <- 58.4274
b <- 14.826
# a <- 55.386246
# b <- 13.056955
date <- "2013-06-04"
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")
```

Kernel Computation (Sum of Three Gaussian Kernels)

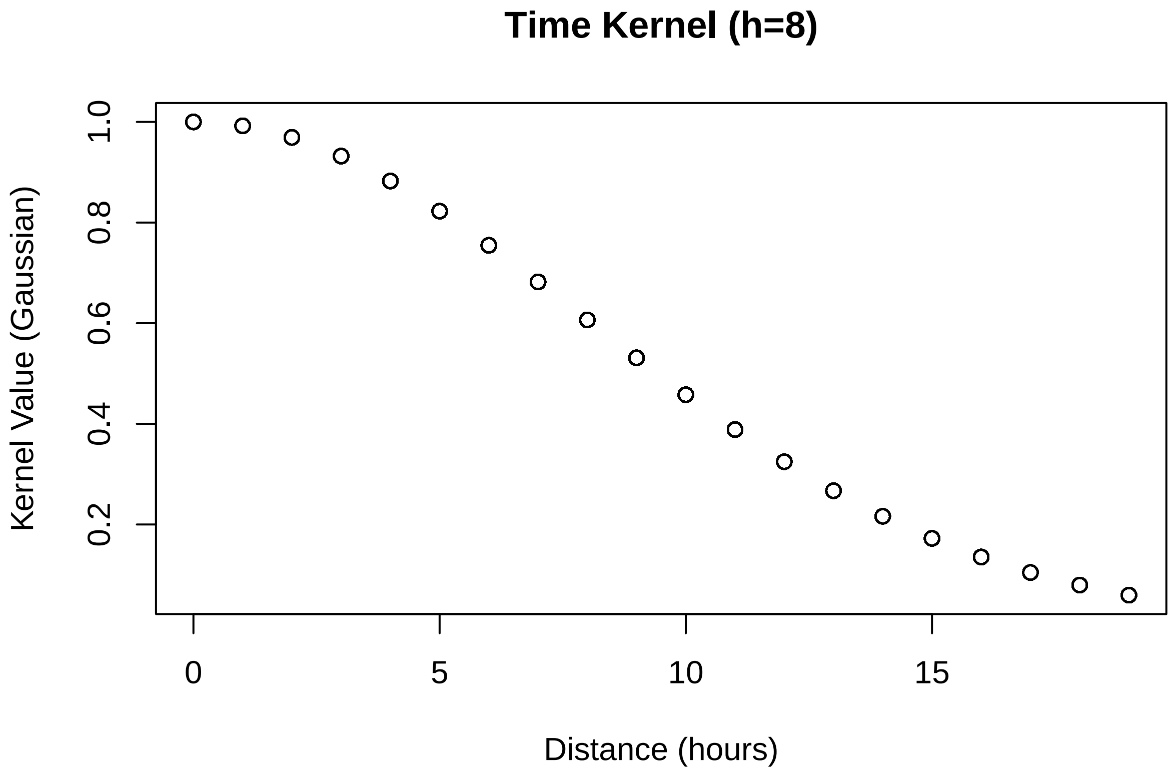
```
# Remove rows with observations posterior to wanted time
data_dates <- strptime(df_st[, "date"], "%Y-%m-%d", tz="UTC")
wanted_date <- strptime(date, "%Y-%m-%d", tz="UTC")
df_st_filt <- df_st[which(data_dates < wanted_date),]

# Distance kernel
k_dist <- exp(-(distHaversine(df_st_filt[, c("longitude", "latitude")], c(a, b
    ))^2) / (2 * h_distance^2))

plot(distHaversine(df_st_filt[, c("longitude", "latitude")], c(a, b)),
      k_dist,
      xlab="Distance␣(m)", ylab="Kernel␣Value␣(Gaussian)")
title(paste0("Distance␣Kernel␣(h=", h_distance, ")"))
```



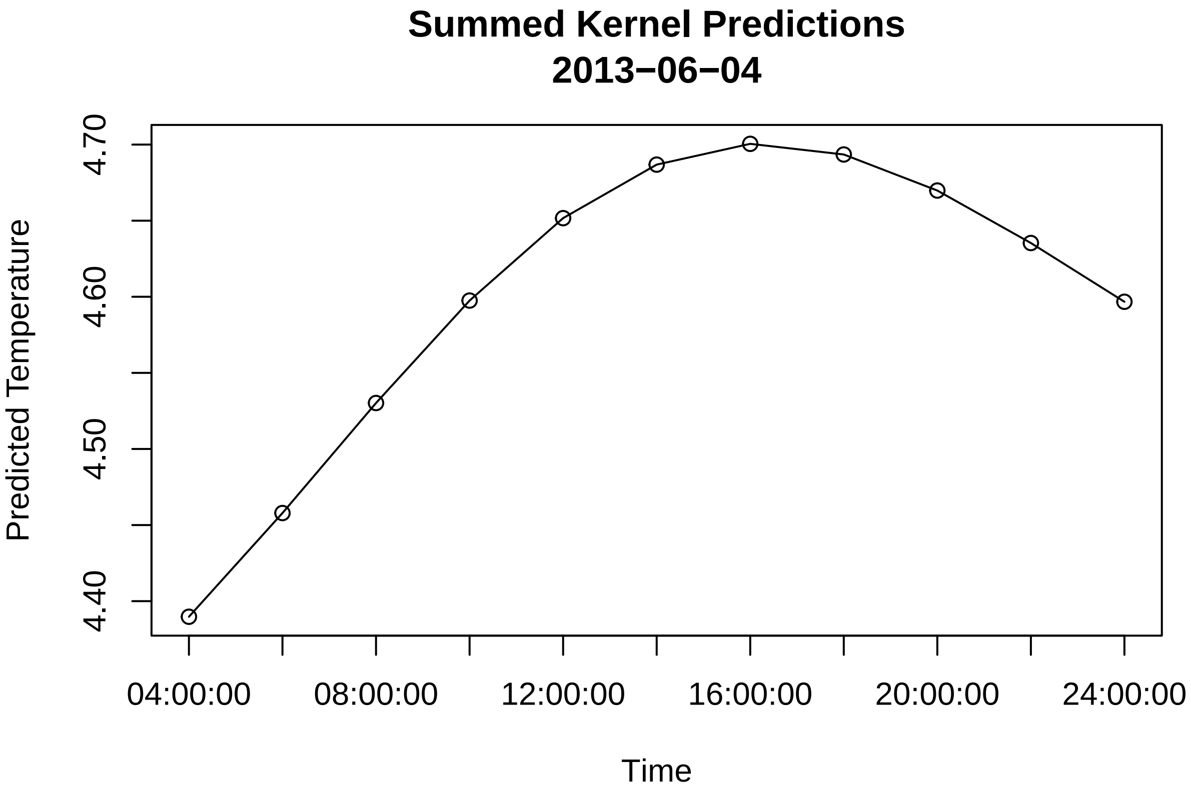
```
### FIND BEST h_distance ###
# v <- vector()
# for (i in seq(1000000, 7000000, by=10000)) {
#   k_dist <- exp(-(distHaversine(df_st_filt[, c(”longitude”, ”latitude”)], c(
    a, b))^2) / (2 * i^2))
#   v <- append(v, max(k_dist) - min(k_dist))
# }
#
# plot(seq(1000000, 7000000, by=10000), v)

# seq(1000000, 7000000, by=10000)[which(v == max(v))]
###########################

# Date kernel
v_date_differences <- as.numeric(as.Date(date, ”%Y-%m-%d”) - as.Date(df_st_
    filt[, ”date”], ”%Y-%m-%d”))
k_date <- exp(-(v_date_differences^2) / (2 * h_date^2))

plot(v_date_differences, k_date,
      xlab=”Distance␣(days)”, ylab=”Kernel␣Value␣(Gaussian)”)
title(paste0(”Date␣Kernel␣(h=”, h_date, ”)”))
```



```
# Hour kernel
# https://stackoverflow.com/questions/53818163/calculate-difference-between-
    hours-not-considering-date
data_hour <- as.numeric(substr(df_st_filt[, "time"], 1, 2))
wanted_hours <- matrix(rep(sapply(times, function(x) {as.numeric(substr(x, 1,
    2))}), nrow(df_st_filt)), ncol=length(times), byrow=TRUE)
v_time_differences <- abs(sweep(wanted_hours, 1, data_hour))
k_time <- exp(-(v_time_differences^2) / (2 * h_time^2))

plot(v_time_differences[, 1], k_time[, 1],
      xlab="Distance␣(hours)", ylab="Kernel␣Value␣(Gaussian)")
title(paste0("Time␣Kernel␣(h=", h_time, ")"))
```

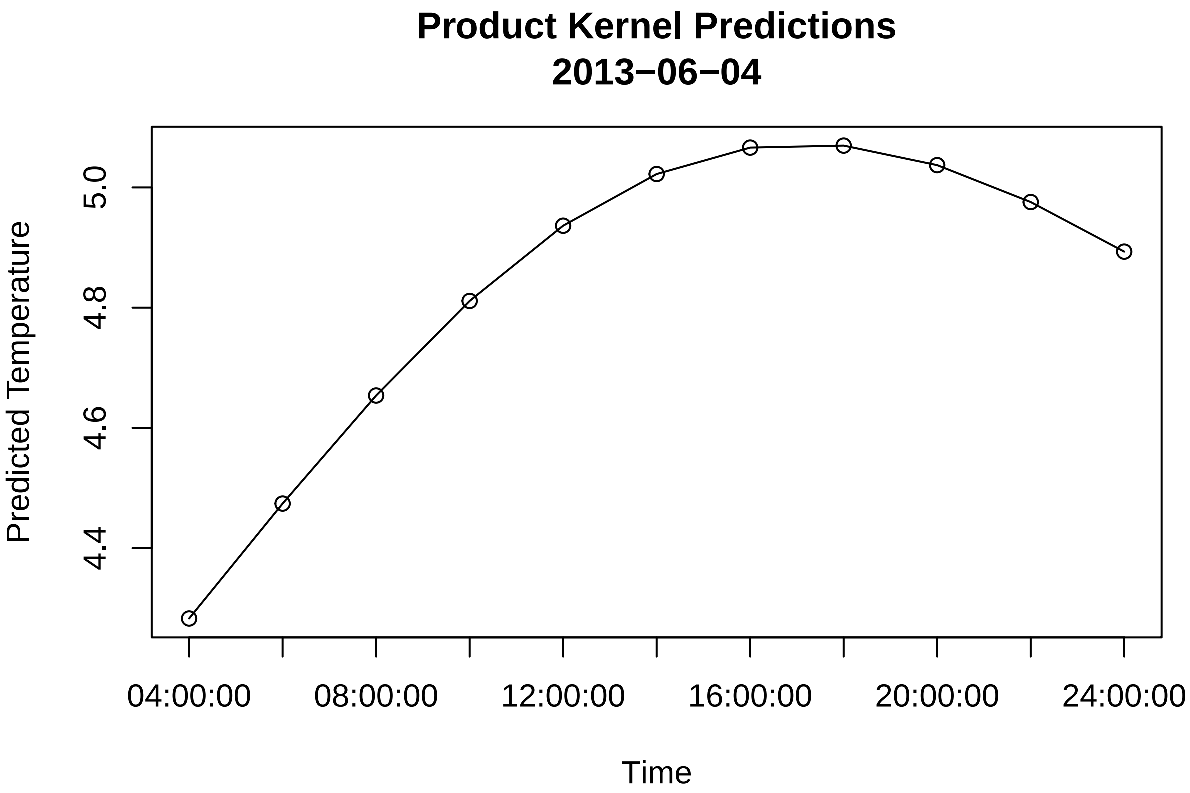


```r
# Combined kernel summed
k_comb_sum <- sweep(k_time, 1, k_dist + k_date, FUN="+")

# Non-parametric kernel regression
temp_sum <- colSums(k_comb_sum * df_st_filt$air_temperature) / colSums(k_comb_sum)
plot(temp_sum, type="o", xaxt="n", xlab="Time", ylab="Predicted Temperature")
axis(1, at=1:length(temp_sum), labels=times)
title(paste0("Summed Kernel Predictions\n", date))
```



Kernel Computation (Product of Three Gaussian Kernels)

```r
# Combined kernel product
k_comb_prod <- k_dist * k_date * k_time

# Non-parametric kernel regression
temp_prod <- colSums(k_comb_prod * df_st_filt$air_temperature) / colSums(k_comb_prod)
plot(temp_prod, type="o", xaxt="n", xlab="Time", ylab="Predicted Temperature")
axis(1, at=1:length(temp_prod), labels=times)
title(paste0("Product Kernel Predictions\n", date))
```



Kernel Comparison

The product kernel produces slightly higher values than the summed kernel. This is due to the division in the computation of the predictions. Multiplying the kernels gives smaller values than summing would, as the kernel values are all between 0 and 1 due to the fact that a Gaussian kernel function was used. However, when dividing by a smaller value this produces slightly higher values than the summed kernel produces.

The model is not very flexible to different inputs; a different location or time of year gives almost the same result. Perhaps different weights should be attached to the different kernels. The hour kernel is always the same, and has equal weight to the other two kernels. Additionally, the distance kernel is only on a range between about 0.32 and 0.42 rather than 0 and 1 as the distance values in this kernel are so large. A big difference in distance may have a big impact on the predicted temperature, but this doesn't reflect from the kernel value. A linear combination between the kernels, where the weights may be tuned may be a better option.

2. SUPPORT VECTOR MACHINES

The code in the file Lab3Block1 2021 SVMs St.R performs SVM model selection to classify the spam dataset. To do so, the code uses the function ksvm from the R package kernlab, which also includes the spam dataset. All the SVM models to select from use the radial basis function kernel (also known as Gaussian) with a width of 0.05. The C parameter varies between the models. Run the code in the file Lab3Block1 2021 SVMs St.R and answer the following questions.

```
# 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, ] # Training set, 3000 samples
va <- spam[3001:3800, ] # Validation set, 800 samples
trva <- spam[1:3800, ] # Training+validation set, 3800 samples
te <- spam[3801:4601, ]  # Test set, 800 samples

nrow(spam) # 4601

## [1] 4601

by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){ # 0.3 0.6 0.9 1.2 1.5 1.8 2.1 2.4 2.7 3.0 3.3 3.6 3.9
    4.2 4.5 4.8
  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: data=tr, C=which.min(err_va)*by
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]) # Predictions on the validation set
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0

## [1] 0.0675

# The difference between filter0 and filter1 is that filter0 is using the
    validation dataset for the prediction while filter 1 uses the test dataset
     for the prediction.

# Filter1: data=tr,
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]) # Predictions on the test set
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
err1

## [1] 0.08489388
```

```
#The difference between filter1 and filter2 is that filter1 training dataset
    for training while fitler2 is using training and validation for training.

# Filter2: data=trva, not good?, training on the validation set and the
    training set
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]) # Predictions on the test set
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
err2

## [1] 0.082397

# Filter3: data=spam, not good because training on the whole dataset, also
    predicting on the test dataset.

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]) # Predictions on the test set
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3

## [1] 0.02122347

#*** This filter cannot be used because it is using the whole dataset for the
    training, therefore the prediction on the test data, which is included in
    the training data, is not reliable.


# Questions

# 1. Which filter do we return to the user ? filter0, filter1, filter2 or
    filter3? Why?
# -> We return filter0 to the user because it is the only one using the
    validation dataset for prediction. All the other 3 filters are using the
    test dataset for prediction, which is not good because the test dataset is
     supposed to be used only for the final evaluation of 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 estimate of the generalization error of the filter returned to the
    user is err1 because it is using the same training as filter0 but the
    prediction is made on the test dataset instead of the validation dataset.
```

Questions

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

-> We return filter0 to the user because it is the only one using the validation dataset for prediction. All the other 3 filters are using the test dataset for prediction, which is not good because the test dataset is supposed to be used only for the final evaluation of 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 estimate of the generalization error of the filter returned to the user is err1 because it is using the same training as filter0 but the prediction is made on the test dataset instead of the validation dataset.

(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 that return the indexes of the support vectors, coef that returns the linear coefficients for the support vectors, and b , that returns the negative intercept of the linear combination. See the help file of the kernlab package for more information. You can check if your results are correct by comparing them with the output of the function predict where you set type = “decision”. Do so for the first 10 points in the spam dataset. Feel free to use the template provided in the Lab3Block1 2021 SVMs St.R file.

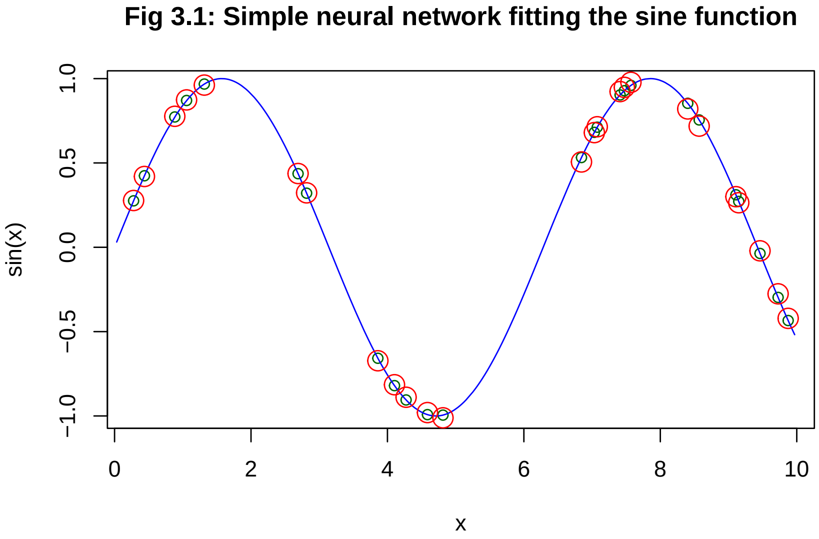
```
# 3. Implementation of SVM predictions.

sv<-alphaindex(filter3)[[1]] # indexes of the support vectors
co<-coef(filter3)[[1]] # linear coefficients for the support vectors
inte<- - b(filter3) # negative intercept of the linear combination
k<-NULL
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  # cat("point:",i,"/10","\n")
  k2<-NULL
  for(j in 1:length(sv)){
    # cat("support vector:",j,"/",length(sv),"\r")
    # a new point is classified according to the sign of a linear combination of the kernel function values between the support vectors and the new point.
    # "kernel used: "radial basis function kernel (also known as Gaussian) with a width of 0.05"
    k2 <- c(k2, co[j]*exp(-0.05*sum((spam[i,-58]-spam[sv[j],-58])^2)))
  }
  k<-c(k, sign(sum(k2)+inte))
}
k

##  [1] -1  1  1 -1 -1  1 -1 -1  1 -1

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

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

Predictions are working !

3. Neural Networks

Assignment 3.1

In Figure 3.1 below, we see the result of a neural network applied to a set of 25 datapoints uniformly spread over the interval $[0, 10]$, with the response being the sine function applied to the datapoints, with no added random error. These training points are colored green and made small. Red points are the pointwise predictions by the neural net. The blue curve represents the true sine curve.

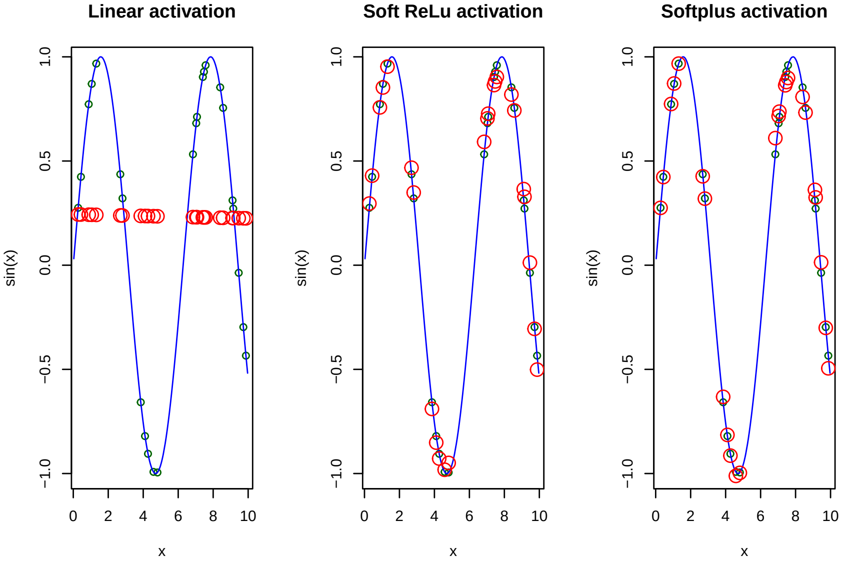


Assignment 3.2

The neural network in Assignment 3.1 used a standard sigmoid activation function. Now we are going to use a linear activation function (or the identity) $h_1(x) = x$, the ReLu $h_2(x) = \max(0, x)$, and the softplus $h_3(x) = \ln(1 + e^x)$ each in their own neural network. However, as we tried this, we ran into an issue with the ReLu. It does not appear to be preprogrammed in the neuralnet function, and neither does neuralnet seem to be able to handle the undifferentiable point in ReLu. Instead we will be using a function we call the "Soft ReLu" $h_4(x) = \ln(1 + e^x) \cdot \frac{e^x}{1+e^x}$, which is constructed by multiplying the softplus function with its derivative. That derivative happens also to be the sigmoid function. The result is a function that resembles the softplus, but takes a sharper turn around $x = 0$, and thus approximates the ReLu better than the softplus.

We train these three neural networks using the data from Assignment 3.1, and plot the resulting predictions in Figure 3.2. Again, green points are training observations, red points are predictions, and the blue curve is the true sine curve. The neural network with a linear "activation function" clearly performs terribly. This is because the linear "activation function", or rather the *lack* of a non-linear activation function, reduces the neural network to a simple linear regression model. This explains why all the predictions by this neural network follow a straight line. For the neural networks with Soft ReLu and Softplus activation functions respectively, the predictions appear to be quite good, but possibly slightly worse than for the neural network with a sigmoid activation function.

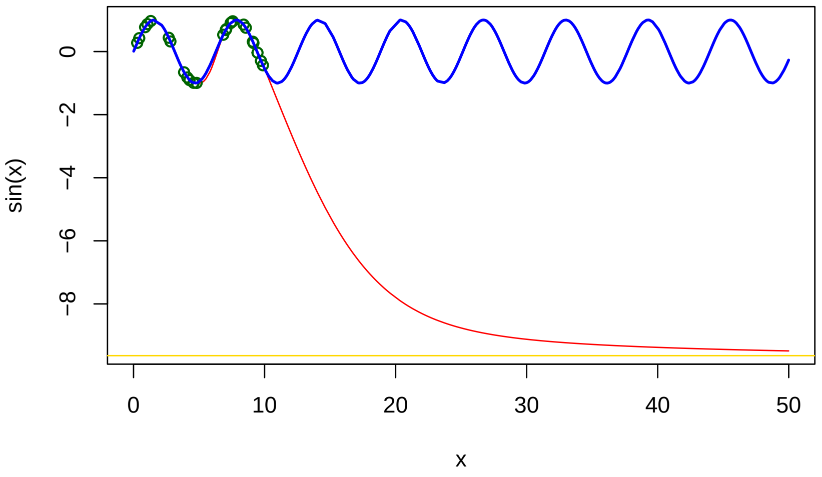
Figure 3.2: Three activation functions



Assignment 3.3

Now we generate data similarly to before in the range $[0, 50]$ and attempt to predict the value of the sine function applied to these values using the neural network that was trained in Assignment 3.1. Essentially we will use the neural network to predict the response of data from far outside the range of the training data. As a reminder, the range of this training data is $[0, 10]$. The result of this is plotted in Figure 3.3. Again the training data of the model is represented by green circles, and the predictions are represented by the red curve. The blue curve represents the entire new dataset (essentially the sine function graph). Clearly, the n eural network has a fatal flaw: It breaks away, producing terrible predictions immediately outside the range of the training data.

Fig 3.3: Neural network predictions on new data from outside the range of training data



Assignment 3.4

In Assignment 3.3 we saw that the prediction curve of the Neural Network appeared to converge to some value. To find this limit, we formulate the function $f(x)$ of the Neural Network, first in compact notation, and then as a linear combination. Here, $W^{(1)}$ is a 10 by 1 matrix carrying the weights of the first layer, $W^{(2)}$ is a 1 by 10 matrix carrying the weights of the second (output) layer, $b^{(1)}$ is a 10 by 1 vector carrying the biases for the first layer, and $b^{(2)}$ is a vector of length 1 (so a scalar in this case) carrying the second (output) layer bias term. The sigmoid activation function $\sigma(\cdot)$ is carried out element wise.

$$f(x) = W^{(2)}\sigma\Big(W^{(1)}x + b^{(1)}\Big) + b^{(2)} = W_1^{(2)}\sigma\Big(W_1^{(1)}x + b_1^{(1)}\Big) + \cdots + W_{10}^{(2)}\sigma\Big(W_{10}^{(1)}x + b_{10}^{(1)}\Big) + b^{(2)}$$

When studying this function, we find that if $W_i^{(1)}$ is positive, then $\sigma\Big(W_i^{(1)}x + b_i^{(1)}\Big)$ will approach 1 as x becomes larger. If $W_i^{(1)}$ is negative, then $\sigma\Big(W_i^{(1)}x + b_i^{(1)}\Big)$ will approach 0 as x becomes larger. If $W_i^{(1)} = 0$, then the activation of the ith hidden unit in the first layer reduces to a constant. This has the effect that as $x \to \infty$, $f(x)$ reduces to the following, where we first introduce an indicator variable δ_i.

$$\delta_i = \begin{Bmatrix} 0 & \text{if} & W_i^{(2)} < 0 \\ \sigma(b_i^{(1)}) & \text{if} & W_i^{(2)} = 0 \\ 1 & \text{if} & W_i^{(2)} > 0 \end{Bmatrix}$$

$$\lim_{x\to\infty} f(x) = W_1^{(2)} \cdot \delta_1 + \cdots + W_{10}^{(2)} \cdot \delta_{10} + b^{(2)}$$

We calculate this limit for the trained Neural Network from Assignment 3.1 to be about -9.64, and include the limit as a gold line of triumph in Figure 3.3.

Assignment 3.5

Now we are interested in training a neural network to learn the arcsine function, which is the inverse of the sine function. To do this, we first generate 500 observations in the interval $[0, 10]$ uniformly, and then apply the sine function to each observation. We then train a neural network with the same structure as previously, but with the observations as response variable, and the sine function applied to the observations as the explanatory variable. We use the sigmoid activation function. In Figure 3.4 we plot the training observations with a green line, and the predictions (on the training data) as a red line.

In Figure 3.4 we see clearly that the neural network is terrible at predicting the observations. This is not surprising, since the sine function is surjective, but not injective, and thus can not be uniquely inverted. Additionally, the range of the explanatory variable ($[-1, 1]$) is quite narrow, which makes it harder for the neural network to capture the behaviour in the response variable.

It should be noted that these issues reside mostly with the problem formulation however. By simulating x-values in the interval $[0, 10]$, rather than the true output region $[-\frac{\pi}{2}, \frac{\pi}{2}]$ of the arcsine function, the neural network is *set up* to fail, rather than failing due to an inherent flaw in the model. In fact any conceivable model would fail just as spectacularly, likely generating similar results to those in Figure 3.4. If we instead reformulate the problem, and generate (as few as 50) x-values uniformly in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, and then train a neural network on this data, we achieve a very good fit, shown in Figure 3.5.

```
## Warning in plot.xy(xy, type, ...): plot type 'lp' will be truncated to
    first
## character
```

Fig 3.4: NN trained with x−values from [0, 10] (inappropriate)

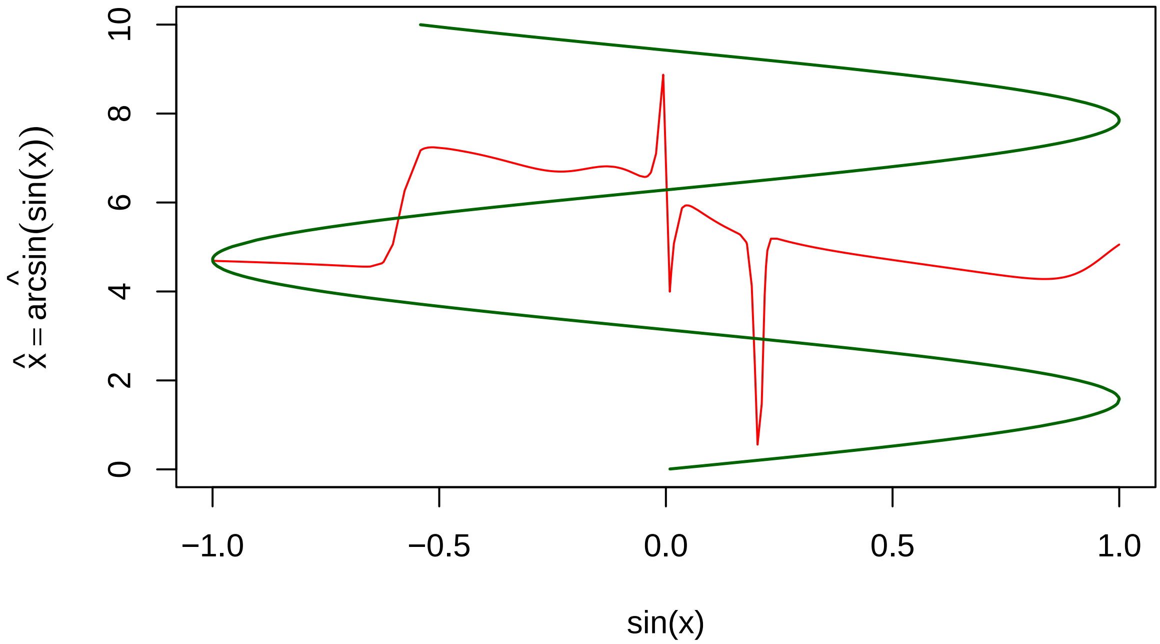
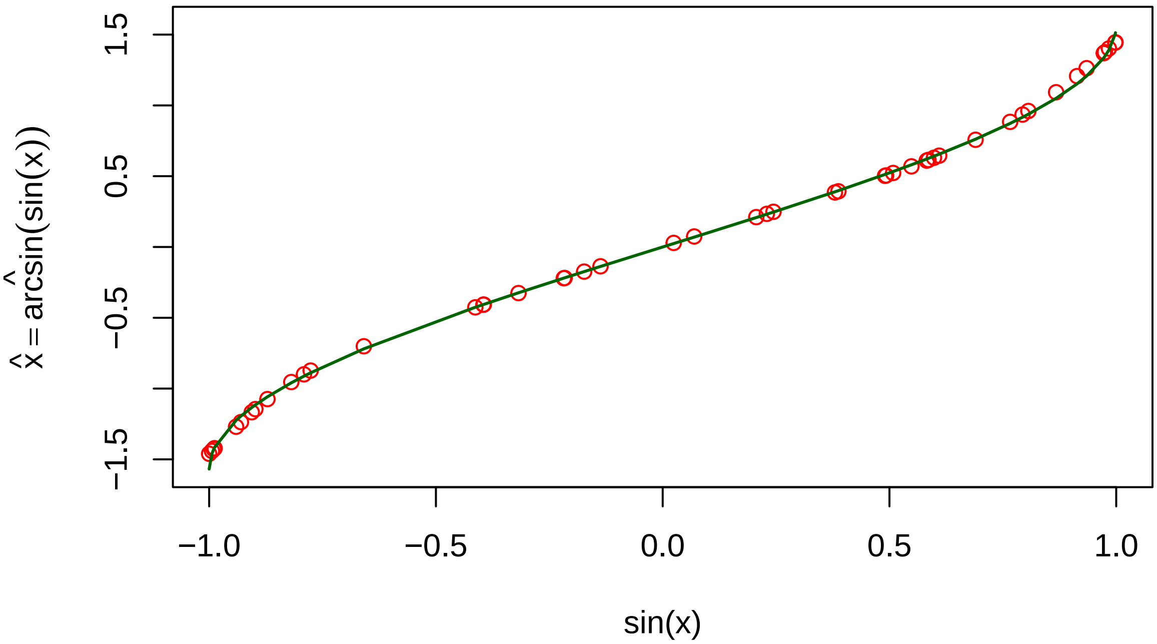


Fig 3.5: NN trained with x−values from $[\frac{-\pi}{2}, \frac{\pi}{2}]$ (appropriate)



Appendix

```
library(geosphere)
library(magrittr)
library(neuralnet)
library(dplyr)
library(latex2exp) # To include "prediction" hats in Figures 3.4 and 3.5

library(knitr) # for caching in Assignment 3
#opts_chunk$set(cache = T)
# Read in data and merge
df_stations <- read.csv("stations.csv", fileEncoding="latin1")
df_temps <- read.csv("temps50k.csv")
df_st <- merge(df_stations, df_temps, by="station_number")

# Define hyper parameters
h_distance <- 4430000
h_date <- 10000
h_time <- 8

# Prediction point
a <- 58.4274
b <- 14.826
# a <- 55.386246
# b <- 13.056955
date <- "2013-06-04"
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")

# Remove rows with observations posterior to wanted time
data_dates <- strptime(df_st[, "date"], "%Y-%m-%d", tz="UTC")
wanted_date <- strptime(date, "%Y-%m-%d", tz="UTC")
df_st_filt <- df_st[which(data_dates < wanted_date),]

# Distance kernel
k_dist <- exp(-(distHaversine(df_st_filt[, c("longitude", "latitude")], c(a, b
    ))^2) / (2 * h_distance^2))

plot(distHaversine(df_st_filt[, c("longitude", "latitude")], c(a, b)),
      k_dist,
      xlab="Distance␣(m)", ylab="Kernel␣Value␣(Gaussian)")
title(paste0("Distance␣Kernel␣(h=", h_distance, ")"))

### FIND BEST h_distance ###
# v <- vector()
# for (i in seq(1000000, 7000000, by=10000)) {
#   k_dist <- exp(-(distHaversine(df_st_filt[, c("longitude", "latitude")], c(
    a, b))^2) / (2 * i^2))
#   v <- append(v, max(k_dist) - min(k_dist))
# }
#
# plot(seq(1000000, 7000000, by=10000), v)

# seq(1000000, 7000000, by=10000)[which(v == max(v))]
############################

# Date kernel
v_date_differences <- as.numeric(as.Date(date, "%Y-%m-%d") - as.Date(df_st_
    filt[, "date"], "%Y-%m-%d"))
k_date <- exp(-(v_date_differences^2) / (2 * h_date^2))

plot(v_date_differences, k_date,
      xlab="Distance␣(days)", ylab="Kernel␣Value␣(Gaussian)")
title(paste0("Date␣Kernel␣(h=", h_date, ")"))

# Hour kernel
# https://stackoverflow.com/questions/53818163/calculate-difference-between-
    hours-not-considering-date
data_hour <- as.numeric(substr(df_st_filt[, "time"], 1, 2))
wanted_hours <- matrix(rep(sapply(times, function(x) {as.numeric(substr(x, 1,
    2))}), nrow(df_st_filt)), ncol=length(times), byrow=TRUE)
v_time_differences <- abs(sweep(wanted_hours, 1, data_hour))
k_time <- exp(-(v_time_differences^2) / (2 * h_time^2))

plot(v_time_differences[, 1], k_time[, 1],
      xlab="Distance␣(hours)", ylab="Kernel␣Value␣(Gaussian)")
title(paste0("Time␣Kernel␣(h=", h_time, ")"))
```

```
# Combined kernel summed
k_comb_sum <- sweep(k_time, 1, k_dist + k_date, FUN="+")

# Non-parametric kernel regression
temp_sum <- colSums(k_comb_sum * df_st_filt$air_temperature) / colSums(k_comb_
   sum)
plot(temp_sum, type="o", xaxt="n", xlab="Time", ylab="Predicted␣Temperature")
axis(1, at=1:length(temp_sum), labels=times)
title(paste0("Summed␣Kernel␣Predictions\n", date))

# Combined kernel product
k_comb_prod <- k_dist * k_date * k_time

# Non-parametric kernel regression
temp_prod <- colSums(k_comb_prod * df_st_filt$air_temperature) / colSums(k_
   comb_prod)
plot(temp_prod, type="o", xaxt="n", xlab="Time", ylab="Predicted␣Temperature")
axis(1, at=1:length(temp_prod), labels=times)
title(paste0("Product␣Kernel␣Predictions\n", date))

# 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, ] # Training set, 3000 samples
va <- spam[3001:3800, ] # Validation set, 800 samples
trva <- spam[1:3800, ] # Training+validation set, 3800 samples
te <- spam[3801:4601, ]  # Test set, 800 samples

nrow(spam) # 4601

by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){ # 0.3 0.6 0.9 1.2 1.5 1.8 2.1 2.4 2.7 3.0 3.3 3.6 3.9
    4.2 4.5 4.8
  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: data=tr, C=which.min(err_va)*by
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]) # Predictions on the validation set
t <- table(mailtype,va[,58])
```

```
err0 <- (t[1,2]+t[2,1])/sum(t)
err0

# The difference between filter0 and filter1 is that filter0 is using the
    validation dataset for the prediction while filter 1 uses the test dataset
     for the prediction.

# Filter1: data=tr,
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]) # Predictions on the test set
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
err1

#The difference between filter1 and filter2 is that filter1 training dataset
    for training while fitler2 is using training and validation for training.

# Filter2: data=trva, not good?, training on the validation set and the
    training set
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]) # Predictions on the test set
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
err2

# Filter3: data=spam, not good because training on the whole dataset, also
    predicting on the test dataset.

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]) # Predictions on the test set
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3

#*** This filter cannot be used because it is using the whole dataset for the
    training, therefore the prediction on the test data, which is included in
    the training data, is not reliable.


# Questions

# 1. Which filter do we return to the user ? filter0, filter1, filter2 or
    filter3? Why?
# -> We return filter0 to the user because it is the only one using the
    validation dataset for prediction. All the other 3 filters are using the
    test dataset for prediction, which is not good because the test dataset is
     supposed to be used only for the final evaluation of 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 estimate of the generalization error of the filter returned to the
```

```
    user is err1 because it is using the same training as filter0 but the
    prediction is made on the test dataset instead of the validation dataset.



# 3. Implementation of SVM predictions.

sv<-alphaindex(filter3)[[1]] # indexes of the support vectors
co<-coef(filter3)[[1]] # linear coefficients for the support vectors
inte<- - b(filter3) # negative intercept of the linear combination
k<-NULL
for(i in 1:10){ # We produce predictions for just the first 10 points in the
    dataset.
  # cat("point:",i,"/10","\n")
  k2<-NULL
  for(j in 1:length(sv)){
    # cat("support vector:",j,"/",length(sv),"\r")
    # a new point is classified according to the sign of a linear combination
        of the kernel function values between the support vectors and the new
        point.
    # "kernel used: "radial basis function kernel (also known as Gaussian)
        with a width of 0.05"
    k2 <- c(k2, co[j]*exp(-0.05*sum((spam[i,-58]-spam[sv[j],-58])^2)))
  }
  k<-c(k, sign(sum(k2)+inte))
}
k
predict(filter3,spam[1:10,-58], type = "decision")
# Setup

library(magrittr)
library(neuralnet)
library(dplyr)
library(latex2exp) # To include "prediction" hats in Figures 3.4 and 3.5

library(knitr) # for caching in Assignment 3
#opts_chunk$set(cache = T)
# Assignment 3.1

# Create sine data
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

# Training a neural network
nn <- neuralnet(data = tr,
                formula = Sin ~ Var,
                hidden = c(10))

# Plot the training, test and predicted data
plot(tr, col="darkgreen",
     main = "Fig␣3.1:␣Simple␣neural␣network␣fitting␣the␣sine␣function",
     xlab = "x",
     ylab = "sin(x)")
points(arrange(te, Var), type="l", col="blue")
points(tr$Var,
       predict(nn, data.frame(tr$Var)),
       col="red", cex=2)

# Assignment 3.2

## NN with linear activation function
nn_linear <- neuralnet(data = tr,
                       formula = Sin ~ Var,
                       hidden = c(10),
                       act.fct = function(x){x})

## NN with soft relu activation function
soft_relu <- function(x){log(1+exp(x))*(exp(x))/(1+exp(x))}
nn_softrelu <- neuralnet(data = tr,
                         formula = Sin ~ Var,
                         hidden = c(10),
                         act.fct = soft_relu)

## NN with softplus activation function
nn_softplus <- neuralnet(data = tr,
                         formula = Sin ~ Var,
                         hidden = c(10),
                         act.fct = function(x){log(1+exp(x))})

# Assignment 3.2

## Plot grid of neural network results
layout(matrix(c(1,2,3), nrow=1))

# Plot result of NN with linear activation
plot(tr, col="darkgreen",
     main = "Linear␣activation",
     xlab = "x",
     ylab = "sin(x)")
points(arrange(te, Var), type="l", col="blue")
points(tr$Var,
       predict(nn_linear, data.frame(tr$Var)),
       col="red", cex=2)

# Plot result of NN with soft relu activation
plot(tr, col="darkgreen",
     main = "Soft␣ReLu␣activation",
     xlab = "x",
     ylab = "sin(x)")
points(arrange(te, Var), type="l", col="blue")
points(tr$Var,
       predict(nn_softrelu, data.frame(tr$Var)),
       col="red", cex=2)

# Plot result of NN with softplus activation
```

```
plot(tr, col="darkgreen",
     main = "Softplus␣activation",
     xlab = "x",
     ylab = "sin(x)")
points(arrange(te, Var), type="l", col="blue")
points(tr$Var,
       predict(nn_softplus, data.frame(tr$Var)),
       col="red", cex=2)

# Set figure title
mtext("Figure␣3.2:␣Three␣activation␣functions", side=3, outer=TRUE, line=-1)
# Assignment 3.3

set.seed(9378364)
Var <- runif(500, 0, 50)
mydata_0_to_50 <- data.frame(Var, Sin=sin(Var))

# Plot NN fitted values to the new data
mydata_0_to_50 <- mydata_0_to_50 %>%
  mutate(nn_prediction = predict(nn, data.frame(mydata_0_to_50$Var))) %>%
  arrange(Var)

plot(x = mydata_0_to_50$Var,
     y = mydata_0_to_50$nn_prediction,
     col="red", type="l", cex=2,
     main = "Fig␣3.3:␣Neural␣network␣predictions␣on␣new␣data␣from␣outside␣the\
        nrange␣of␣training␣data",
     xlab = "x",
     ylab = "sin(x)")
points(tr %>% arrange(Var), col="darkgreen", type = "p", lwd=1.5)
points(arrange(mydata_0_to_50, Var), type="l", col="blue", lwd=2)


# Find the limit of the NN as x increases
# This had to be hardcoded because the neuralnet function has its parameters
# structured in a quite unintuitive way.
sign_of_W1 <- nn$result[c(5,7,9,11,13,15,17,19,21,23)] > 0
limit <- sum(sign_of_W1 * nn$result[25:34]) + nn$result[24]
abline(b=0, a=limit, col="gold")
# Assignment 3.5

# Create data
set.seed(9239462)
x_vals <- runif(500, 0, 10)
df_sindata <- data.frame(x=x_vals, sin_x = sin(x_vals))

# Train a NN for the arcsin function
nn_arcsin <- neuralnet(data = df_sindata,
                       formula = x ~ sin_x,
                       hidden = c(10),
                       threshold = 0.1)

# Arrange the data so observations can be connected with lines in plot below
df_sindata <- df_sindata %>%
  mutate(predicted_x = predict(nn_arcsin, data.frame(df_sindata$sin_x))) %>%
  arrange(sin_x)

# Adjust margins to avoid cropping y label below
par(mar = c(5, 4, 4, 2)+0.5) # Bottom Left Top Right

# Plot the result (using inappropriate data)
plot(x = df_sindata$sin_x,
     y = df_sindata$predicted_x,
     col="red", type="lp", cex=1, xlim = c(-1, 1), ylim = c(0, 10),
     main = paste("Fig␣3.4:␣NN␣trained␣with␣x-values␣from␣[0,␣10]␣(
        inappropriate)"),
     xlab = "sin(x)",
     ylab = TeX(r"($\hat{x}␣=␣\hat{arcsin}(sin(x))$)"))
points(x = df_sindata %>% arrange(x) %>% extract2(2), # sin_x
       y = df_sindata %>% arrange(x) %>% extract2(1), # x
       col="darkgreen", type = "l", lwd=1.5)
# Assignment 3.5

# Create data
set.seed(92834984)
x_vals_2 <- runif(50, -pi/2, pi/2)
df_sindata_2 <- data.frame(x=x_vals_2, sin_x = sin(x_vals_2))

# Train a NN for the arcsin function
nn_arcsin_2 <- neuralnet(data = df_sindata_2,
                         formula = x ~ sin_x,
                         hidden = c(10))

# Arrange the data so observations can be connected with lines in plot below
df_sindata_2 <- df_sindata_2 %>%
  mutate(predicted_x = predict(nn_arcsin_2, data.frame(df_sindata_2$sin_x)))
     %>%
  arrange(predicted_x)

# Adjust margins to avoid cropping y label below
par(mar = c(5, 4, 4, 2)+0.5) # Bottom Left Top Right

# Plot the resulting predictions and true arcsin function (using good data)
plot(x = df_sindata_2$sin_x,
     y = df_sindata_2$predicted_x,
     col="red", type="p", cex=1, xlim = c(-1, 1), ylim = c(-pi/2, pi/2),
     main = expression("Fig␣3.5:␣NN␣trained␣with␣x-values␣from␣[" ~ frac(-pi,
        2) ~ ",␣" ~ frac(pi, 2) ~ "]␣(appropriate)"),
     xlab = "sin(x)",
     ylab = TeX(r"($\hat{x}␣=␣\hat{arcsin}(sin(x))$)"))
points(x = df_sindata_2 %>% arrange(x) %>% extract2(2), # sin_x
       y = df_sindata_2 %>% arrange(x) %>% extract2(1), # x
       col="darkgreen", type = "l", lwd=1.5)
```