# Machine Learning Computer Lab 1 Block2 (Group A7)

Qinyuan Qi(qinqi464) Satya Sai Naga Jaya Koushik Pilla (satpi345)
Daniele Bozzoli(danbo826)

2023-12-17

## Assignment 1: KERNEL METHODS(Solved by Qinyuan Qi)

**Answer:**

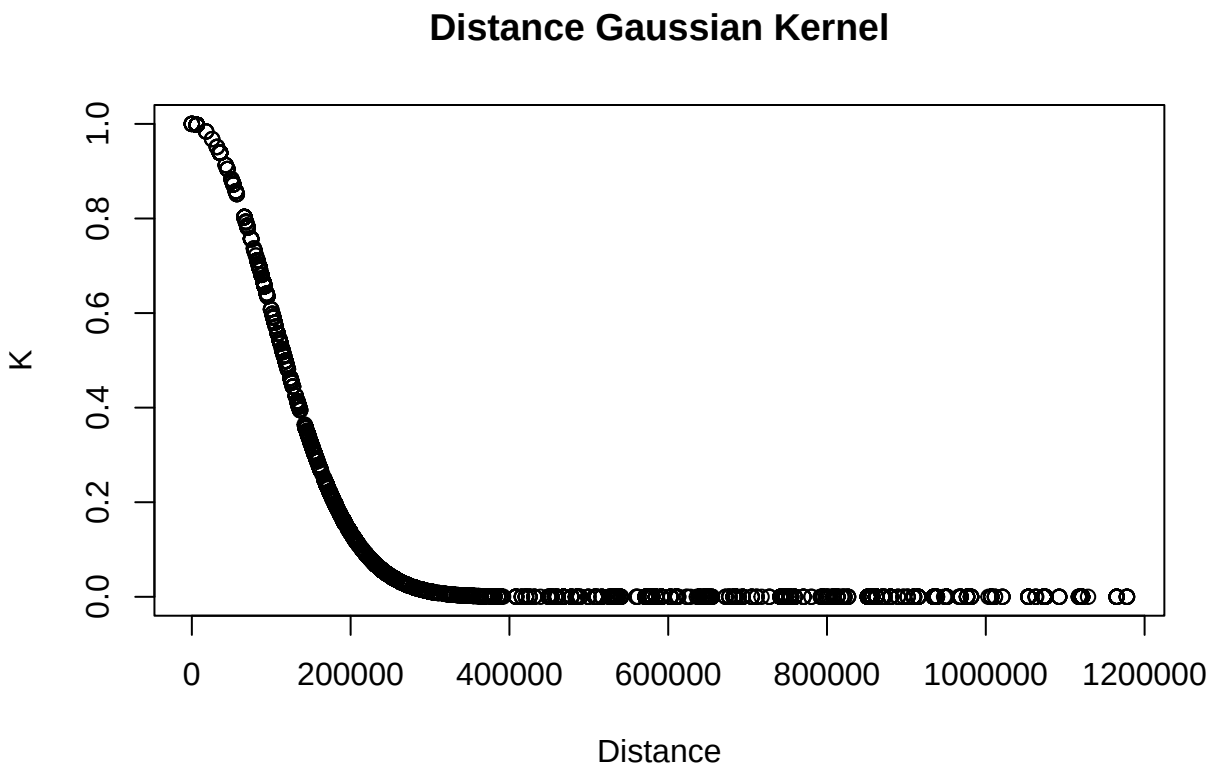
In this assignment, we will try to predict the temp of Linköping station on date 1970-11-05.

The 3 basic kernel functions are implemented as follows.

```
######################### Kernel Code  ###################################
# Gaussian kernel of geo distance
geo_distance <- function(data, location_interest, h_dist) {
  location <- data.frame(longitude = data$longitude,
                         latitude  = data$latitude)
  distances <- distHaversine(location_interest,location)
  kernel_result <- exp(-(distances)^2 / (2 * h_dist^2))
  return(c(distances, kernel_result))
}

# Gaussian kernel of day distance
day_distance <- function(data, day_interest, h_day) {
  distances <- as.numeric(difftime(day_interest,data,
                           units = "days"))
  kernel_result <- exp(-(distances)^2 / (2 * h_day^2))
  return(c(distances, kernel_result))
}

# Gaussian kernel of hour distance
hour_distance <- function(data, hour_interest, h_hour) {

  diff <- sapply(1:length(hour_interest), function(x){
              abs(as.numeric(difftime(strptime(hour_interest[x], "%H:%M:%S"),
                                      strptime(data$time, "%H:%M:%S")), units="hours"))
  })

  distances <- ifelse(diff <= 12, diff, 24-diff)
  kernel_result <- exp(-(distances)^2 / (2 * h_day^2))
  return(kernel_result)
}
```

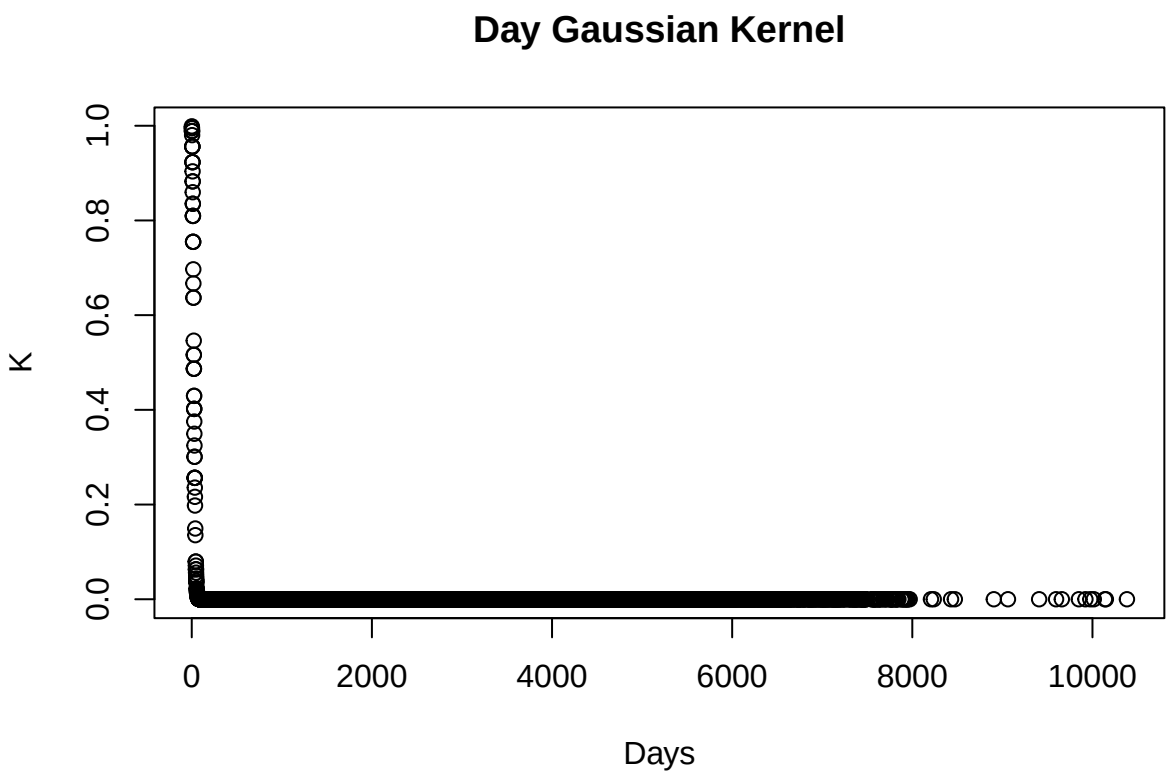
We set h_distance=100000, and set station to Linköping, then we plot the kernel function for distance as follows.

```r
################################ 1.1 Distance Kernel ############################

# These three values are up to the students
h_distance <- 100000

# set coordinates to predict(Linköping)
a <- stations$longitude[station_row]
b <- stations$latitude[station_row]

N <- nrow(train)
i <- 1:N
k_distance <- sapply(i, function(i){geo_distance(train[i, ], c(a, b), h_distance)})
plot(k_distance[1,], k_distance[2,], main = "Distance Gaussian Kernel", xlab = "Distance", ylab = "K")
```




We set h_day=20, then we plot the kernel function as follows.

```r
################################ 1.2 Day Kernel ################################
h_day <- 20
k_day <- sapply(i, function(i) day_distance(as.Date(train$date[i]), date_interest, h_day))
plot(k_day[1,], k_day[2,], main = "Day Gaussian Kernel", xlab = "Days", ylab = "K")
```

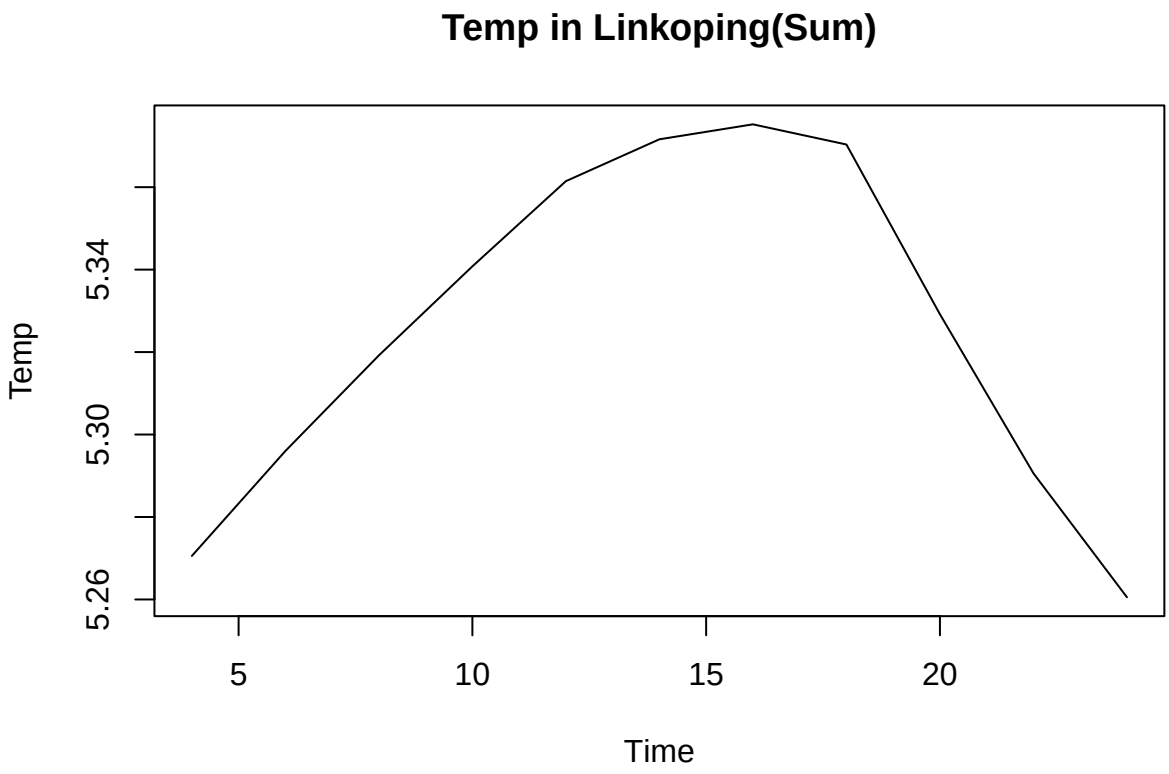



We set h_hour = 7.

```
################################ 1.3 Hour Kernel ################################

h_hour <- 7

# Get the time format we want
times_interest <- c(paste0("0",seq(4,8,by=2),":00:00"), paste0(seq(10,24,by=2),":00:00"))

k_hour <- sapply(i, function(i) hour_distance(train[i,], times_interest, h_hour))
```

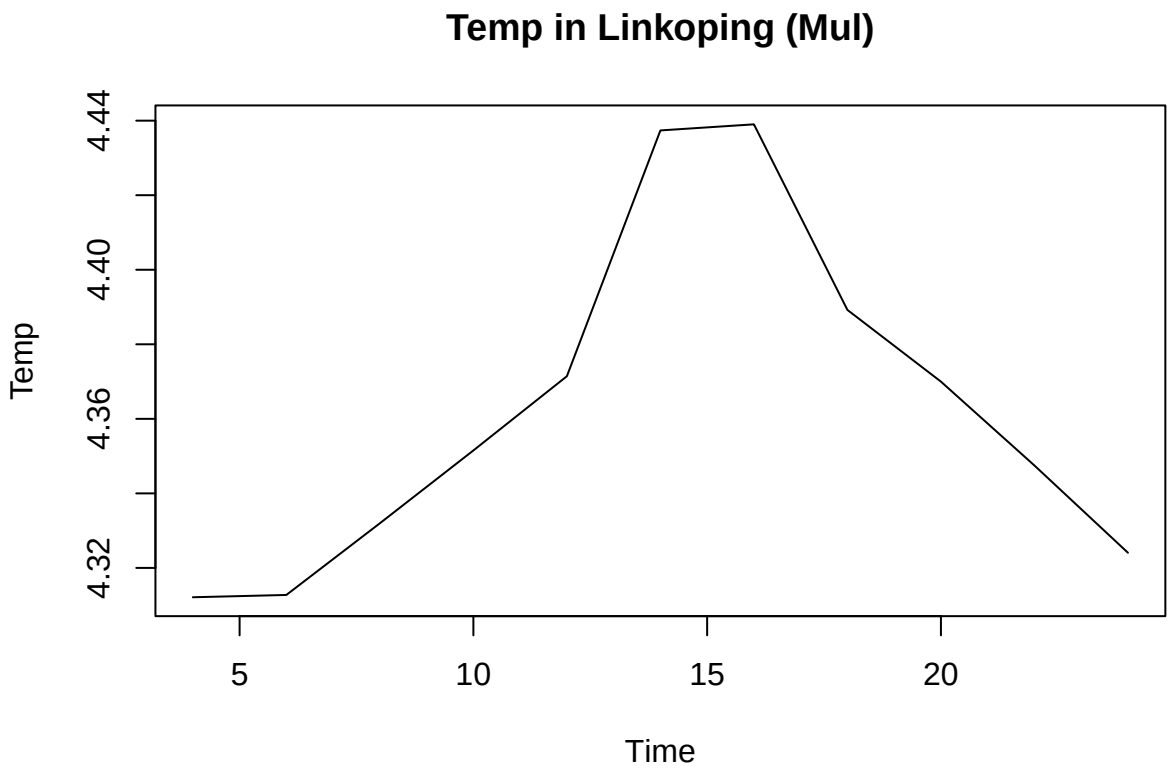
Now we setting up another new kernel(sum of 3 kernels), and plot the prediction value as follows.

```
################################ 1.4 Sum Kernel ################################
predicted_temp_sum <- sapply(1:length(times_interest), function(x)
        sum((k_distance[2,] + k_day[2,] + k_hour[x,]) * train$air_temperature)
        / sum(k_distance[2,], k_day[2,], k_hour[x,]))

plot(seq(4,24,by=2),predicted_temp_sum, type="l",
main = "Temp in Linkoping(Sum)", xlab = "Time", ylab = "Temp")
```




Now we setting up another new kernel(mul of 3 kernels), and plot the prediction value as follows.

```
################################ 1.5 Multi Kernel ################################
predicted_temp_mul <- sapply(1:length(times_interest), function(x)
        sum((k_distance[2,] * k_day[2,] * k_hour[x,]) * train$air_temperature)
        / sum(k_distance[2,] * k_day[2,] * k_hour[x,]))

plot(seq(4,24,by=2),predicted_temp_mul, type="l",
main = "Temp in Linkoping (Mul)", xlab = "Time", ylab = "Temp")
```




Compare the sum and mul version of the kernel, we can see that there have around 1 degree difference. According to the real results, we found the following near Nov in Linköping.

1962-10-09,18:00:00,9.6 / 1971-11-29,06:00:00,0 / 1975-11-06,12:00:00,10 / 1961-11-02,18:00:00,11

It seems that the predictions still have some gaps near the real results.

## Assignment 2: SUPPORT VECTOR MACHINES(Solved by Satya Sai Naga Jaya Koushik Pilla)

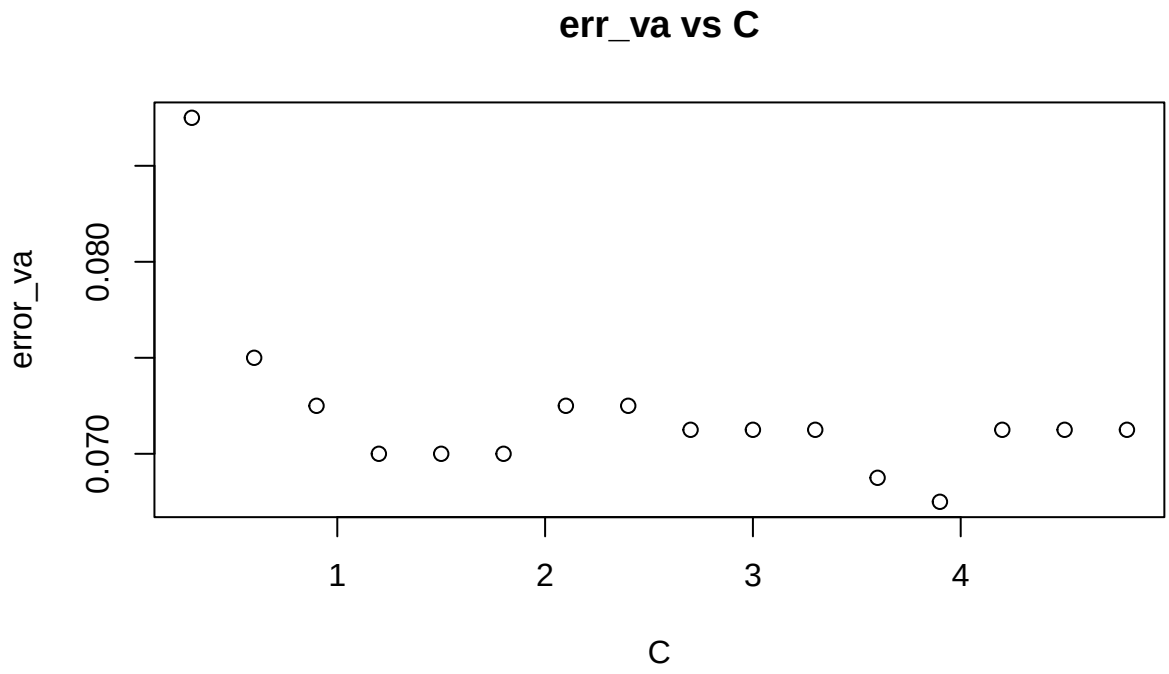
**Answer:**

According to the code, we know that the spam data is splitted into training,validation,training+validation, and test data respectively.

According to the code output, we know that the minimal error is 0.0675 when C = 0.3.

```
## minimal error = 0.0675  when C = 3.9
```

Also we draw the plot of error_va versus C as follows.




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

Then we calc the error rate of 4 filters, accoring to this, we know the accuracy rates are as follows.

| Filter | Accuracy | Accept / Reject |
|---|---|---|
| filter0 | 93.25 % | Accept |

Filter0 follow the best practice and get the highest accuracy rate. It train on training data,it predict on validation data

| Filter | Accuracy | Accept / Reject |
|---|---|---|
| filter1 | 91.51061 % | Reject |

Filter1 follow the best practice but its accuracy rate is lower than filter 0. It train on training data,it predict on test data

| Filter | Accuracy | Accept / Reject |
|---|---|---|
| filter2 | 91.7603 % | Reject |

Filter2 does not follow the best practice but still practiciable, It train on training+validation data, predict on test data, also its accuracy rate is lower than filter 0.

| Filter | Accuracy | Accept / Reject |
|---|---|---|
| filter3 | 97.87765 % | Reject |

Filter3 training on all data,and using the seen data to predict which is not accepetable.

Based on the analysis above, we should return filter0 to the user.

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

Generalization error is the expectation of the the predicted values of an independent test set on our model.

Based on the test data, we got the following result.

The reason why filter0 and filter1 are same is because these 2 filters share the same parameters and are same.

```
## generalization error of filter0 is 8.489388 %
```

```
## generalization error of filter1 is 8.489388 %
```

```
## generalization error of filter2 is 8.2397 %
```

```
## generalization error of filter3 is 2.122347 %
```

### 2.3 Implementation of SVM predictions.

The missing code implemented as follows.

```
sv <- alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
# RBF kernel with sigma = 0.05
rbf <- rbfdot(sigma = 0.05)
k<-NULL
# We produce predictions for just the first 10 points in the dataset.
for(i in 1:10){
  k2<-NULL
  for(j in 1:length(sv)){
    k2 <- k2 + co[j] * rbf(as.numeric(spam[sv[j], -58]),
                           as.numeric(spam[i, -58]))
  }
  k<-c(k,  k2 + inte)
}
k
```

```
## numeric(0)
```

```
pred_filter3 <- predict(filter3,spam[1:10,-58], type = "decision")
pred_filter3
```

```
##            [,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
```

## Assignment 3: NEURAL NETWORKS(Solved by Daniele Bozzoli)

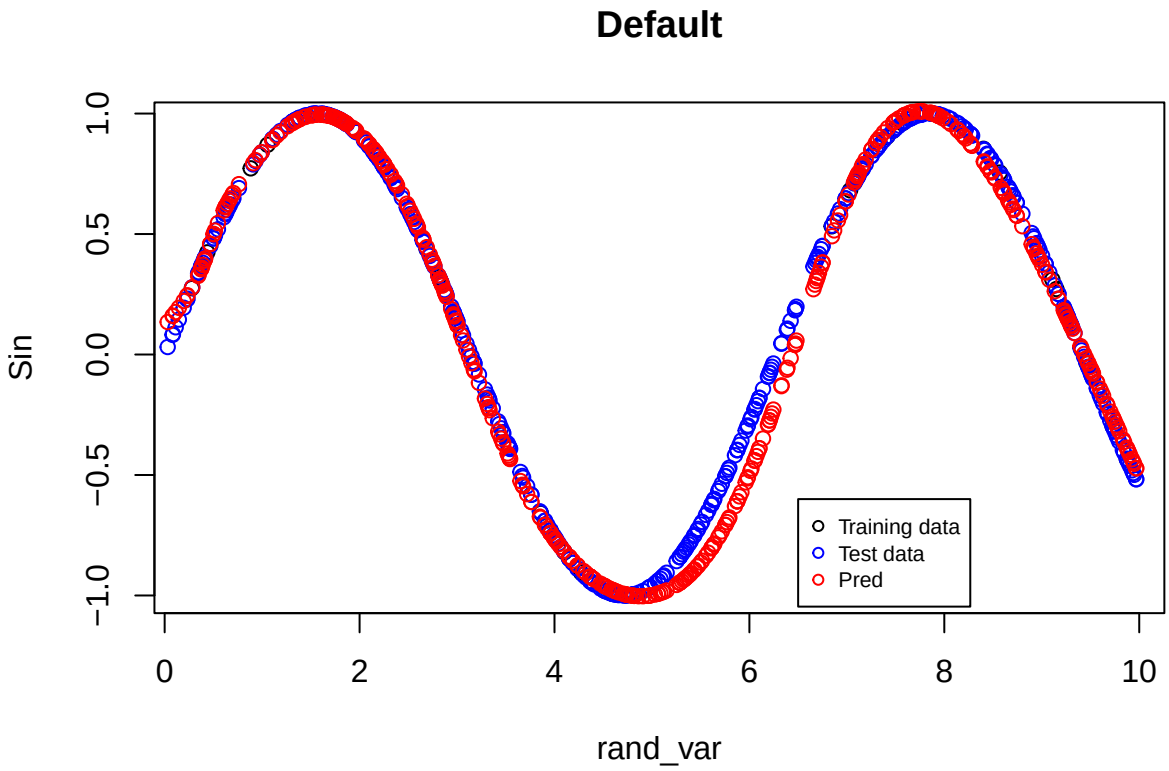
**Answer:**

**3.1**

We observe in the plot that the predictions using neural network seem to be good predictions for the sine function.

Since this function is differentiable, it will also provide a smooth gradient.

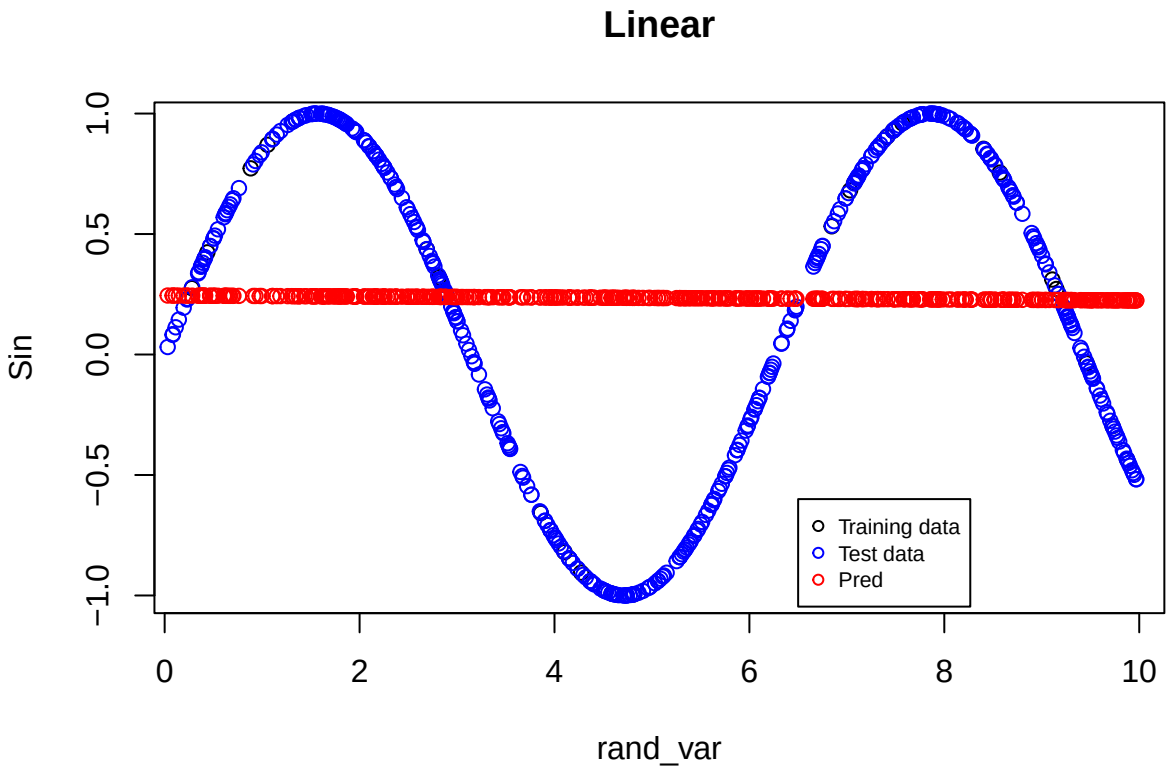
```
#########################  3.1 ###################################################
rand_var <- runif(500, 0, 10)
df <- data.frame(rand_var, Sin=sin(rand_var))
train <- df[1:25,] # Training
test <- df[26:500,] # Test

# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
nn <- neuralnet(data = train, formula = Sin ~ rand_var, hidden = c(10))

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1, main="Default")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)
```




**3.2**

First of all, we define linear activation function as follows. According to the plot, we can see that the prediction is a straight line compared to the train and test data which shows a sine wave.

```
############################  3.2.1 Linear ########################################

#Linear
h1 <- function(x) {
  x
}

nn1 <- neuralnet(data = train, formula = Sin ~ rand_var, hidden = c(10), act.fct = h1)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1, main="Linear")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn1,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)
```

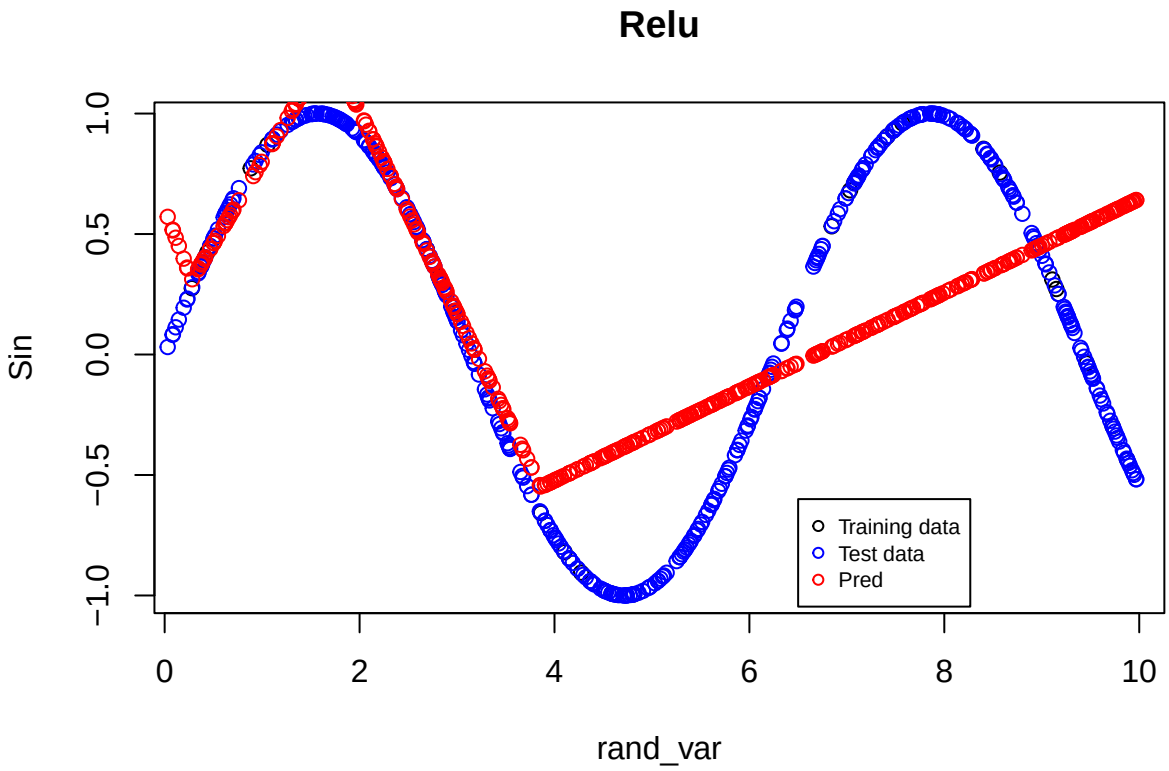



Then we define ReLU activation function as follows, we use ifelse to define the function since max(0,x) not working here.

As what can be seen in the plot, ReLU function as an activation function do not provide good results on test data.

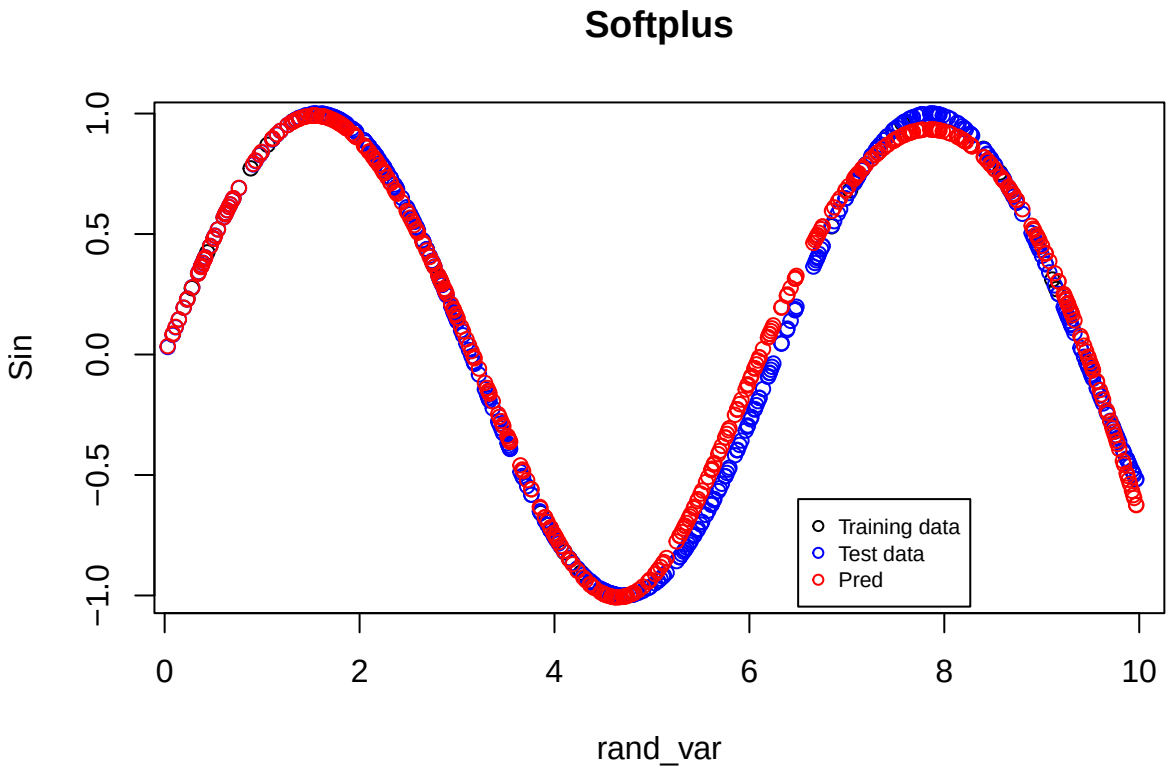
The right prediction range only in around(0.5,4) . when random variable in range (4,10), it shows a straight line as what we see in linear activation function.

```
############################  3.2.2 Relu ########################################

#ReLU
h2 <- function(x) {
  # max(0,x) not working
  ifelse(x>=0,x,0)
}


nn2 <- neuralnet(formula = Sin ~ rand_var, data = train, hidden = 10,
startweights = winit, act.fct = h2)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1,main="Relu")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn2,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)
```




Last we define Softplus activation function as follows, as what can be seen in the plot, Softplus function as an activation function provide good results on test data.

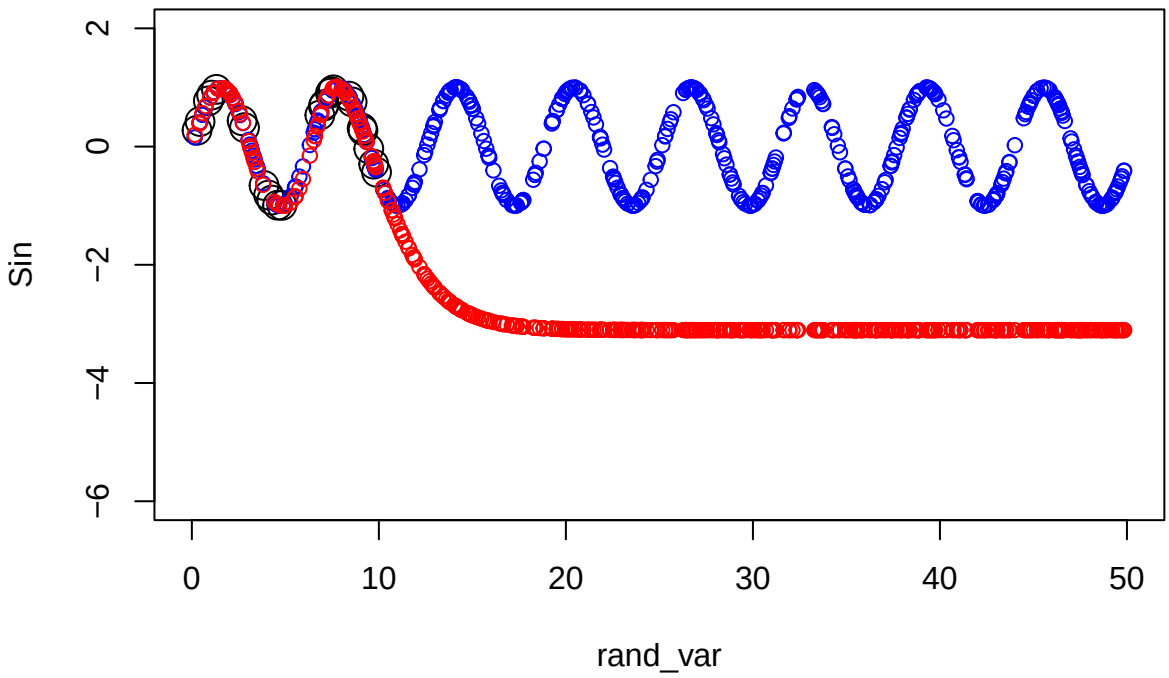
```
############################  3.2.3 Softplus ########################################

#Softplus
h3 <- function(x) {
  log(1+exp(x))
}

nn3 <- neuralnet(formula = Sin ~ rand_var, data = train, hidden = 10,
startweights = winit, act.fct = h3)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1,main="Softplus")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn3,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)
```




**3.3**

The code is as follows. When random in range (0,10), the prediction is match the test data, when it is greater than 10, the prediction begin to get smaller and eventually convergeing to a value which is around -3.

```
############################  3.3  ########################################
# Sample 500 points
set.seed(1234567890)
rand_var <- runif(500, min = 0, max = 50)
df <- data.frame(rand_var, Sin = sin(rand_var))
plot(train, cex = 2, xlim = c(0, 50), ylim = c(-6, 2))
points(df, col = "blue", cex = 1)
points(df[, 1], predict(nn, df), col = "red", cex = 1)
```




```
plot(nn)
```

### 3.4

We define the following code to get the value of the convergence. by setting a relativ big value 100, we can calculate the approximation value of the convergence around -3.105391.

```
############################  3.4  ########################################

sigmoid <- function(x){
  return(1 / (1 + exp(-x)))
}
#
weight_1 <- nn$weights[[1]][[1]][2,]
bias_1 <- nn$weights[[1]][[1]][1,]

sigmoid_val <- sigmoid(weight_1 * 100 + bias_1)

bias_2 <- nn$weights[[1]][[2]][1,]
weight_2 <- nn$weights[[1]][[2]][2:11,]
cat("The value will converge to:", weight_2 %*% sigmoid_val + bias_2, "\n")
```

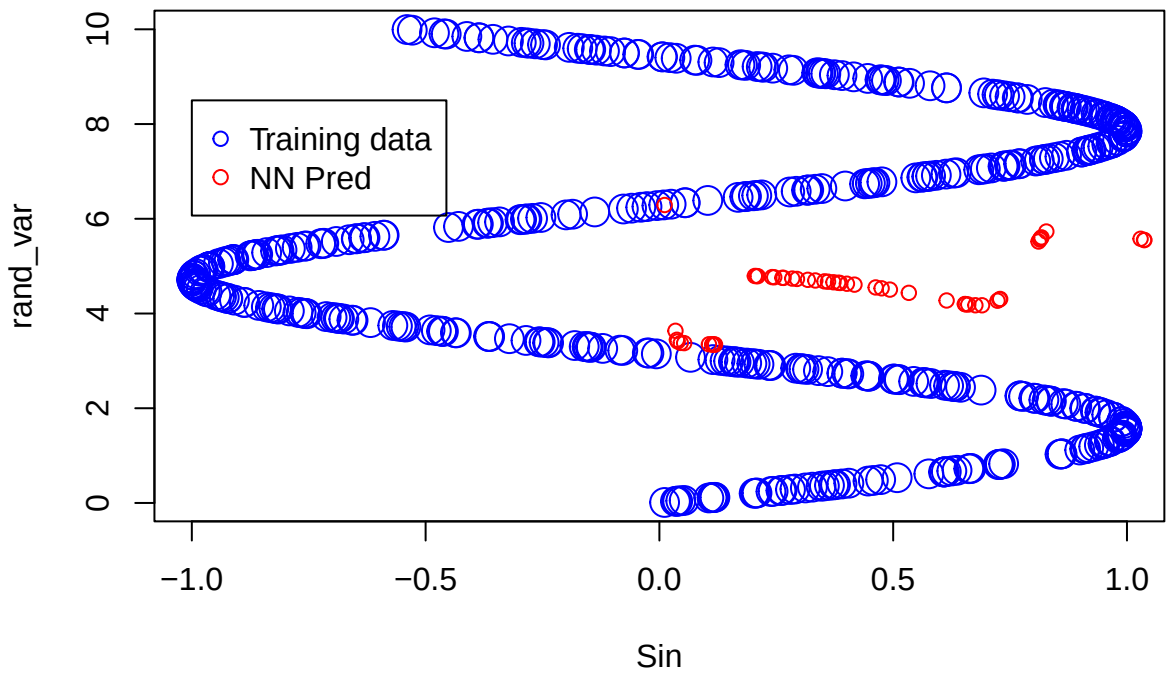
```
## The value will converge to: -3.105391
```

### 3.5

The code as follows.

We can see that the pridcton is not good, the reason is many input values can be mapping to a same value if we using sine function, but if we inverse this function, the original input value can not be calculated through the inverse of sine function.so the NN can not learn the mapping relationship between input and output and generate a bad prediction.

```
##########################  3.5  ######################################

# Sample 500 points
rand_var <- runif(500, min = 0, max = 10)
df <- data.frame(Sin = sin(rand_var), rand_var)
nn4 <- neuralnet(formula = rand_var ~ Sin, data = df, hidden = 10, startweights = winit, threshold = 0.
pred_4 <- predict(nn4, df)

plot(df, cex = 2, col = "blue")
points(df[, 2], pred_4, col = "red", cex = 1)
legend(x = -1, y = 8.5, legend = c("Training data", "NN Pred"),
col = c("blue", "red"), cex = 1, pch = 1)
```



## Appendix: All code for this report

```
############################  Init code For Assignment 1 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(geosphere)
set.seed(1234567890)
############################ init data ######################################
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations, temps, by = "station_number")

# define date of interest
date_interest <- as.Date("1970-11-05")

# filter out data which is later than data of interest
st$date <- as.Date(st$date)
old_sf <- st #for later use
st <- st[-which(difftime(date_interest, st$date) <= 0),]

# define station of interest to Linköping
station_row <- which(stations$station_name == "Linköping")
station <- stations$station_name[station_row]

# split the data to train and test (70/30)
n <- dim(st)[1]
id <- sample(1:n, floor(n * 0.7))
train <- st[id, ]
test <- st[-id, ]
########################## Kernel Code  ######################################
# Gaussian kernel of geo distance
geo_distance <- function(data, location_interest, h_dist) {
  location <- data.frame(longitude = data$longitude,
                         latitude  = data$latitude)
  distances <- distHaversine(location_interest,location)
  kernel_result <- exp(-(distances)^2 / (2 * h_dist^2))
  return(c(distances, kernel_result))
}

# Gaussian kernel of day distance
day_distance <- function(data, day_interest, h_day) {
  distances <- as.numeric(difftime(day_interest,data,
                          units = "days"))
  kernel_result <- exp(-(distances)^2 / (2 * h_day^2))
  return(c(distances, kernel_result))
}

# Gaussian kernel of hour distance
hour_distance <- function(data, hour_interest, h_hour) {

  diff <- sapply(1:length(hour_interest), function(x){
              abs(as.numeric(difftime(strptime(hour_interest[x], "%H:%M:%S"),
                                      strptime(data$time, "%H:%M:%S")), units="hours"))
  })
```

```
  distances <- ifelse(diff <= 12, diff, 24-diff)
  kernel_result <- exp(-(distances)^2 / (2 * h_day^2))
  return(kernel_result)
}
################################ 1.1 Distance Kernel ############################

# These three values are up to the students
h_distance <- 100000

# set coordinates to predict(Linköping)
a <- stations$longitude[station_row]
b <- stations$latitude[station_row]

N <- nrow(train)
i <- 1:N
k_distance <- sapply(i, function(i){geo_distance(train[i, ], c(a, b), h_distance)})
plot(k_distance[1,], k_distance[2,], main = "Distance Gaussian Kernel", xlab = "Distance", ylab = "K")
################################ 1.2 Day Kernel #################################
h_day <- 20
k_day <- sapply(i, function(i) day_distance(as.Date(train$date[i]), date_interest, h_day))
plot(k_day[1,], k_day[2,], main = "Day Gaussian Kernel", xlab = "Days", ylab = "K")
################################ 1.3 Hour Kernel ################################

h_hour <- 7

# Get the time format we want
times_interest <- c(paste0("0",seq(4,8,by=2),":00:00"), paste0(seq(10,24,by=2),":00:00"))

k_hour <- sapply(i, function(i) hour_distance(train[i,], times_interest, h_hour))
################################ 1.4 Sum Kernel #################################
predicted_temp_sum <- sapply(1:length(times_interest), function(x)
        sum((k_distance[2,] + k_day[2,] + k_hour[x,]) * train$air_temperature)
        / sum(k_distance[2,], k_day[2,], k_hour[x,]))

plot(seq(4,24,by=2),predicted_temp_sum, type="l",
main = "Temp in Linkoping(Sum)", xlab = "Time", ylab = "Temp")

################################ 1.5 Multi Kernel ###############################
predicted_temp_mul <- sapply(1:length(times_interest), function(x)
        sum((k_distance[2,] * k_day[2,] * k_hour[x,]) * train$air_temperature)
        / sum(k_distance[2,] * k_day[2,] * k_hour[x,]))

plot(seq(4,24,by=2),predicted_temp_mul, type="l",
main = "Temp in Linkoping (Mul)", xlab = "Time", ylab = "Temp")



############################  Init code For Assignment 2 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(kernlab)
set.seed(1234567890)
############################  Load Data and split data sets #####################
```

```
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, ]
############################  Get error va and corresponding C ####################
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))
}

cat("minimal error =", min(err_va), " when C =", which.min(err_va) * by,"\n")
############################  Plot err_va vs C ####################
c_plot <- seq(by, 5, by)
plot(x = c_plot, y = err_va, type = "p",xlab = "C", ylab = "error_va", main = "err_va vs C")

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)

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)

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


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

#cat("accuracy of filter0 is", (1-err0)*100,"%\n")
#cat("accuracy of filter1 is", (1-err1)*100,"%\n")
#cat("accuracy of filter2 is", (1-err2)*100,"%\n")
#cat("accuracy of filter3 is", (1-err3)*100,"%\n")
filter0_pred <- predict(filter0, te, type = "response")
filter0_gerr <- mean(filter0_pred != te[, 58])

filter1_pred <- predict(filter1, te, type = "response")
filter1_gerr <- mean(filter1_pred != te[, 58])
```

```r
filter2_pred <- predict(filter2, te, type = "response")
filter2_gerr <- mean(filter2_pred != te[, 58])

filter3_pred <- predict(filter3, te, type = "response")
filter3_gerr <- mean(filter3_pred != te[, 58])

cat("generalization error of filter0 is", filter0_gerr*100,"%\n")
cat("generalization error of filter1 is", filter1_gerr*100,"%\n")
cat("generalization error of filter2 is", filter2_gerr*100,"%\n")
cat("generalization error of filter3 is", filter3_gerr*100,"%\n")
sv <- alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
# RBF kernel with sigma = 0.05
rbf <- rbfdot(sigma = 0.05)
k<-NULL
# We produce predictions for just the first 10 points in the dataset.
for(i in 1:10){
  k2<-NULL
  for(j in 1:length(sv)){
    k2 <- k2 + co[j] * rbf(as.numeric(spam[sv[j], -58]),
                           as.numeric(spam[i, -58]))
  }
  k<-c(k,  k2 + inte)
}
k
pred_filter3 <- predict(filter3,spam[1:10,-58], type = "decision")
pred_filter3
############################  Init code For Assignment 3 ########################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(neuralnet)
set.seed(1234567890)
############################  3.1 ################################################
rand_var <- runif(500, 0, 10)
df <- data.frame(rand_var, Sin=sin(rand_var))
train <- df[1:25,] # Training
test <- df[26:500,] # Test

# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
nn <- neuralnet(data = train, formula = Sin ~ rand_var, hidden = c(10))

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1, main="Default")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)
############################  3.2.1 Linear ########################################

#Linear
h1 <- function(x) {
```

```
  x
}

nn1 <- neuralnet(data = train, formula = Sin ~ rand_var, hidden = c(10), act.fct = h1)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1, main="Linear")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn1,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)

############################  3.2.2 Relu ########################################

#ReLU
h2 <- function(x) {
  # max(0,x) not working
  ifelse(x>=0,x,0)
}


nn2 <- neuralnet(formula = Sin ~ rand_var, data = train, hidden = 10,
startweights = winit, act.fct = h2)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1,main="Relu")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn2,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)

############################  3.2.3 Softplus ########################################

#Softplus
h3 <- function(x) {
  log(1+exp(x))
}

nn3 <- neuralnet(formula = Sin ~ rand_var, data = train, hidden = 10,
startweights = winit, act.fct = h3)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train,  col = "black", cex=1,main="Softplus")
points(test, col = "blue", cex=1)
points(test[,1], predict(nn3,test), col="red", cex=1)
legend(x = 6.5, y = -0.6, legend = c("Training data", "Test data",
"Pred"), col = c("black", "blue", "red"), cex = 0.7,pch = 1)

############################  3.3  ########################################
# Sample 500 points
set.seed(1234567890)
rand_var <- runif(500, min = 0, max = 50)
df <- data.frame(rand_var, Sin = sin(rand_var))
plot(train, cex = 2, xlim = c(0, 50), ylim = c(-6, 2))
```

```
points(df, col = "blue", cex = 1)
points(df[, 1], predict(nn, df), col = "red", cex = 1)
plot(nn)
############################  3.4  ########################################

sigmoid <- function(x){
  return(1 / (1 + exp(-x)))
}
#
weight_1 <- nn$weights[[1]][[1]][2,]
bias_1 <- nn$weights[[1]][[1]][1,]

sigmoid_val <- sigmoid(weight_1 * 100 + bias_1)

bias_2 <- nn$weights[[1]][[2]][1,]
weight_2 <- nn$weights[[1]][[2]][2:11,]
cat("The value will converge to:", weight_2 %*% sigmoid_val + bias_2, "\n")

############################  3.5  ########################################

# Sample 500 points
rand_var <- runif(500, min = 0, max = 10)
df <- data.frame(Sin = sin(rand_var), rand_var)
nn4 <- neuralnet(formula = rand_var ~ Sin, data = df, hidden = 10, startweights = winit, threshold = 0.
pred_4 <- predict(nn4, df)

plot(df, cex = 2, col = "blue")
points(df[, 2], pred_4, col = "red", cex = 1)
legend(x = -1, y = 8.5, legend = c("Training data", "NN Pred"),
col = c("blue", "red"), cex = 1, pch = 1)
```