Computer Lab 3 Block 1

Group A7

2022-12-17

Group members:Hamza (hammu144), Dinesh (dinsu875) and Umamaheswarababu (umama339)

Statement of Contribution : Assignment 1 was mostly done by Dinesh, Assignment 2 was mostly done by Umamaheswarababu and Assignment 3 was mostly done by Hamza

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).

The forecast includes predicted temperatures for the interval between 4 am and 24 pm, with a 2-hour interval between each prediction. The predictions are obtained by using a formula that involves computing two different kernels: a sum of three Gaussian kernels and a product of three Gaussian kernels.

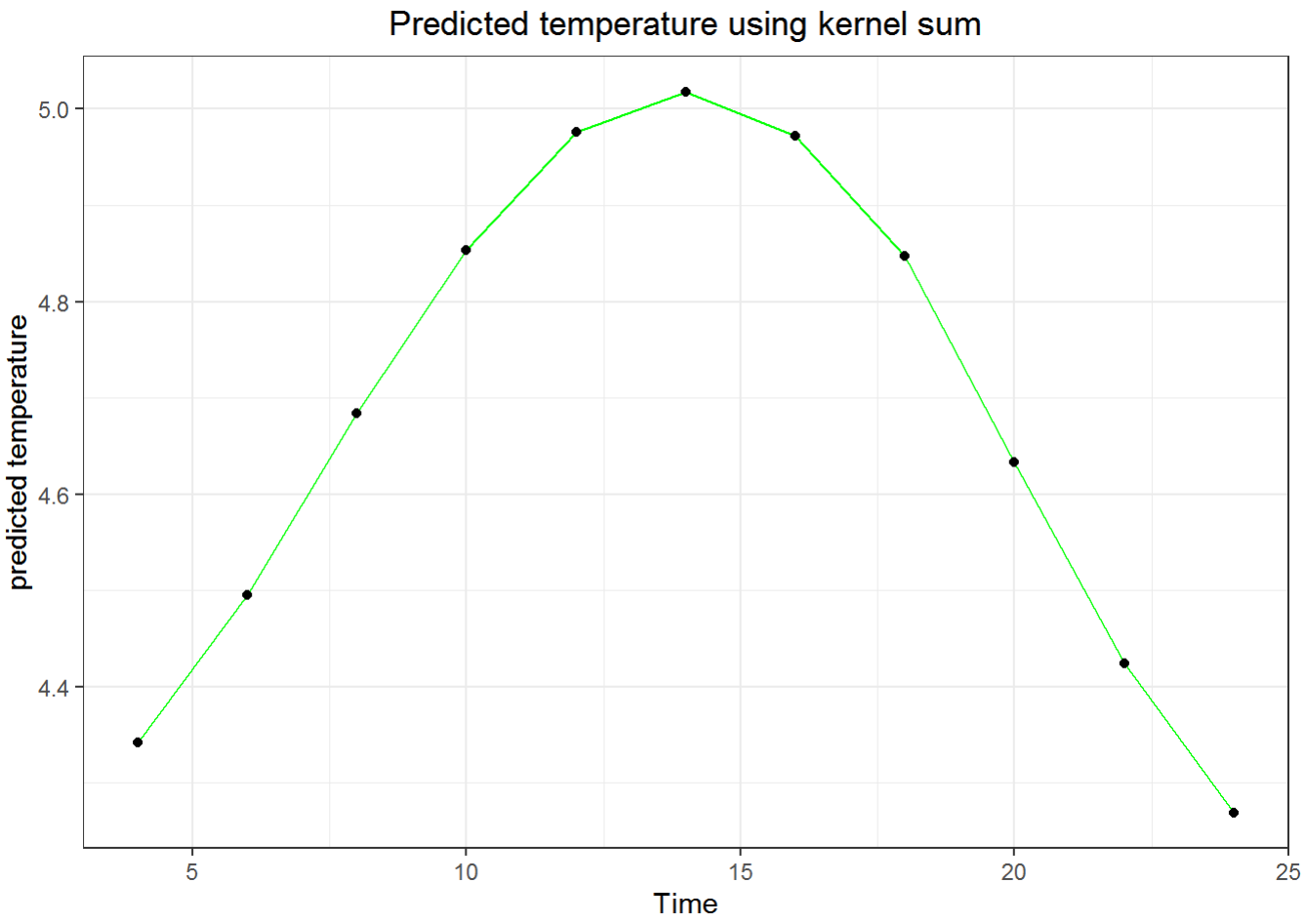
$$y_k(x) = \frac{\sum_n k(\frac{x-x_n}{h})t_n}{\sum_n k(\frac{x-x_n}{h})}$$

```
######################## Assignment 1 ##########################
library(ggplot2)
library(geosphere)
set.seed(12345)
stations <- read.csv("stations.csv", fileEncoding="latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

#smoothing coefficient or width
h_distance <- 30000
h_day <- 7
h_time <- 12

#predicted latitude
a <- 58.4108
#predicted longitude
b <- 15.6214
date <- as.Date("2016-5-30")
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00",
           "16:00:00", "18:00:00", "20:00:00", "22:00:00", "24:00:00")
temp <- length(times)

#Physical distance from a station to the point of interest
lat_long<-cbind(st$latitude,st$longitude)
dist_loc <- distHaversine(lat_long, c(a,b))
# calculating "u" for location
u_dist <- dist_loc / h_distance
#Gaussian Kernels for location
K_dist <- exp(-(u_dist)^2)

#Distance between the day a temperature measurement was made and the day of interest
st$date <- difftime(as.Date(st$date),as.Date(date),units = "days")
st$date <- as.numeric(abs(st$date))
# calculating "u" for days
u_day <- st$date / h_day
#Gaussian Kernels for for days
K_day <- exp(-(u_day)^2)

#Formatting the time for consecutive hours
times1 <- as.POSIXct(times,format="%H:%M:%S")
st$time <- as.POSIXlt(st$time, format="%H:%M:%S")

total_kernel_sum <- c()
total_kernel_mul <- c()
df_weather <- data.frame() # final dataset of predicted values

for(i in 1:temp){
  # Distance between the hour of the day a temperature measurement was made and the hour of inte
rest
  dist_time <- as.numeric(difftime(st$time,times1[i], units = "hours"))
  dist_time <- abs(dist_time)
  dist_time[dist_time > 12] = 24 - dist_time[dist_time > 12]
```

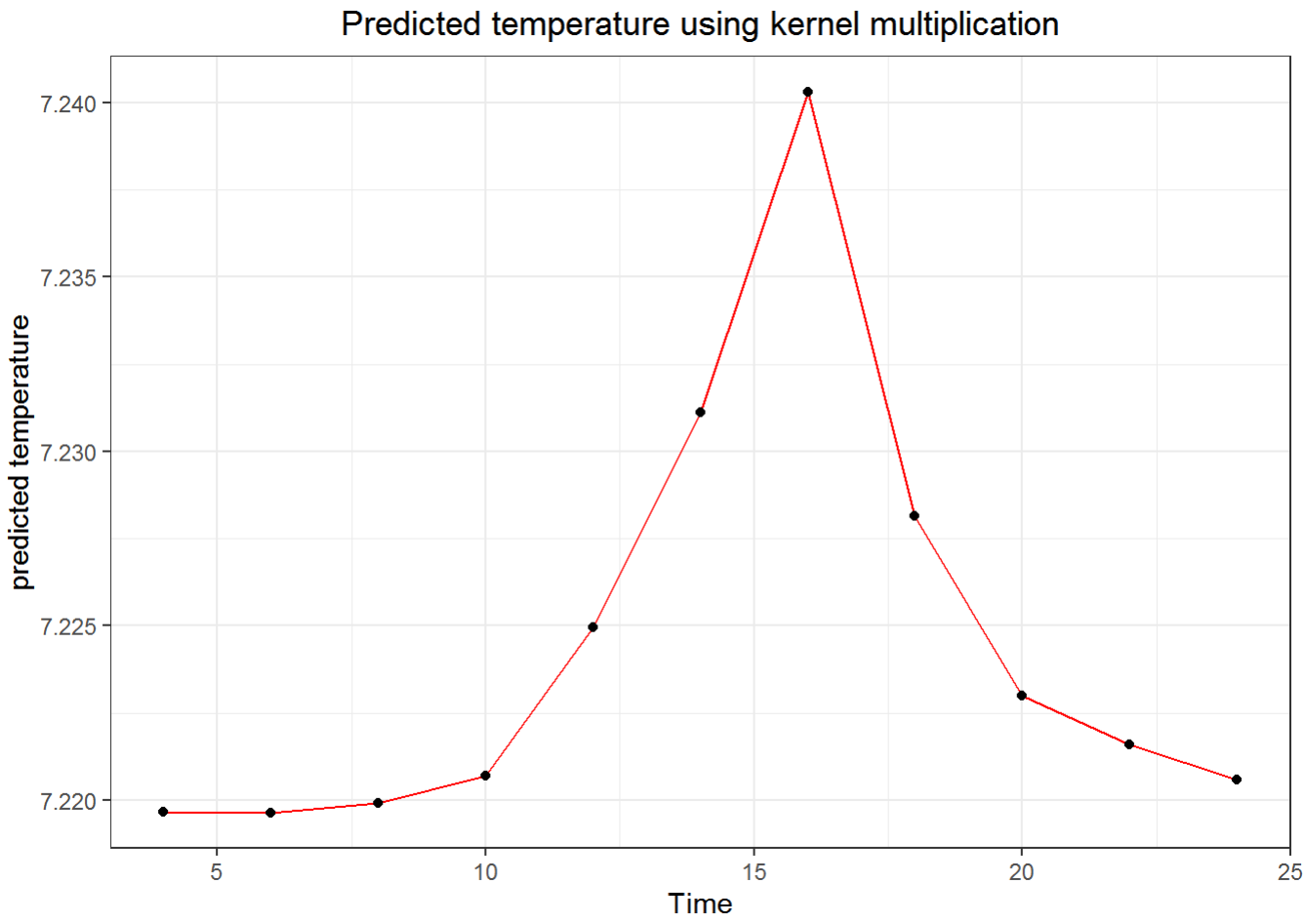
```
  # calculating "u" for time
  u_hour <- dist_time / h_time
  #Gaussian Kernel for time
  K_hour <- exp(-(u_hour)^2)
  # Summing three Gaussian kernels
  kernal_sum <- K_dist + K_day + K_hour
  # Multiplying three Gaussian kernels
  kernal_mul <- K_dist * K_day * K_hour
  total_kernel_sum[i] <- sum(kernal_sum*st$air_temperature) / sum(kernal_sum)
  total_kernel_mul[i] <- sum(kernal_mul*st$air_temperature) / sum(kernal_mul)
  df <- data.frame(TIME = times[i], KERNAL_SUM = total_kernel_sum[i], KERNAL_MUL = total_kernel_
mul[i])
  df_weather <- rbind(df_weather, df)
}
knitr::kable(df_weather)
```

TIME	KERNAL_SUM	KERNAL_MUL
04:00:00	4.341389	7.219652
06:00:00	4.495297	7.219642
08:00:00	4.683386	7.219902
10:00:00	4.853034	7.220707
12:00:00	4.975747	7.224951
14:00:00	5.018038	7.231110
16:00:00	4.972425	7.240312
18:00:00	4.846739	7.228133
20:00:00	4.632890	7.222997
22:00:00	4.424134	7.221585
24:00:00	4.269310	7.220587

```
#Plot kernal_sum
ggplot(df_weather, aes(x=seq(4, 24, 2))) +
  geom_line(aes(y=total_kernel_sum), color="green") +
  geom_point(aes(y=total_kernel_sum), color="black") +
  xlab("Time") +
  ylab("predicted temperature") +
  ggtitle("Predicted temperature using kernel sum") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5), legend.position = "right")
```




```
#plot kernal_mul
ggplot(df_weather, aes(x=seq(4, 24, 2))) +
  geom_line(aes(y=total_kernel_mul), color="red") +
  geom_point(aes(y=total_kernel_mul), color="black") +
   xlab("Time") +
  ylab("predicted temperature") +
  ggtitle("Predicted temperature using kernel multiplication") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5), legend.position = "right")
```



The prediction date is set as May 30th, and the distance from our location to nearby stations in Sweden is approximately 30,000 km. The forecast covers a week, or 7 days, and the time frame is set at 12 hours, or half a day. The smoothing parameter is chosen carefully to avoid overfitting the prediction and to achieve a balance between high variance and low bias. The kernel widths are selected in a sensible manner, taking into account the importance of closer points and practical considerations.

By seeing the two plots, it can be seen that the temperature values obtained from both the summed up and multiplicative kernels appear to be positive. These two kernels differ in their sensitivity to changes in the kernel widths: if one width becomes zero, it will have a greater impact on the multiplicative kernel than the summed up kernel, because the term will become zero in multiplication, affecting the result.When the kernel widths are extremely large, the values become saturated and there is no variation. On the other hand, when the widths are very small, the multiplicative kernel effectively produces a prediction of 0, while the summation kernel produces a highly varying curve. In fact, large variations are observed when random values are used for the h-values, indicating the need for careful selection of appropriate h parameters for accurate prediction.

Assignment 2 : SUPPORT VECTOR MACHINES

```
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,scal
ed=FALSE)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0
```

```
## [1] 0.0675
```

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

```
## [1] 0.08489388
```

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

```
## [1] 0.082397
```

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

```
## [1] 0.02122347
```

Q1: Which filter do you return to the user ? filter0, filter1, filter2 or filter3? Why?

I would recommend *filter2* to the user. *filter2* performs better than the rest of the filters because it uses more data (trva) for training the model and te data as test data which is not at all seen the training data.

The error of *filter0* looks better than *filter1* and *filter2* but the C value used in this model is derived from the best model using tr data for training and va data for testing. The same data is used in *filter0* also so the error obtained is less.

And finally the error of *filter3* looks better than all other filters but the training data used in this is entire data set spam and the train data is te which is already seen in the train data spam. This is the reason for getting low error. The same results may not be obtained for new test data.

Q2: What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why?

error1 is the generalized error because the *filter1* uses tr and te as train and test data respectively but the best value of parameter C was derived from the model that uses va as test data which is not at all seen in the filter *filter1*.

Q3: 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. 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*.

Implementation of SVM model for filter3

```
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,sc
aled=FALSE)
```

The indexes of the support vectors are given by

```
alphaindex(filter3)
```

The linear coefficients for the support vectors are given by

```
coef(filter3)
```

The negative intercept of the linear combination is given by

```
-b(filter3)
```

```
## [1] -0.3130056
```

Comparision of results

```
rbfkernel<-rbfdot(sigma = 0.05)
sv<-alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
k<-NULL

for (i in 1:10){
  k2<-NULL
  for(j in 1:length(sv)){
    k2<-c(k2, rbfkernel(as.vector(unlist(spam[i,-58])),as.vector(unlist(spam[sv[j],-58]))))
  }
  k<-c(k, sum(co*k2)+inte)
}

Predicted<-as.vector(predict(filter3,spam[1:10,-58], type = "decision"))
Calculated<-k
Index<-1:10
df<-data.frame(Index,Predicted,Calculated)
df
```

```
##    Index Predicted Calculated
## 1      1 -1.998999  -1.998999
## 2      2  1.560584   1.560584
## 3      3  1.000278   1.000278
## 4      4 -1.756815  -1.756815
## 5      5 -2.669577  -2.669577
## 6      6  1.291312   1.291312
## 7      7 -1.068444  -1.068444
## 8      8 -1.312493  -1.312493
## 9      9  1.000184   1.000184
## 10    10 -2.208639  -2.208639
```

The results are same.

Assignment 3 : NEURAL NETWORKS

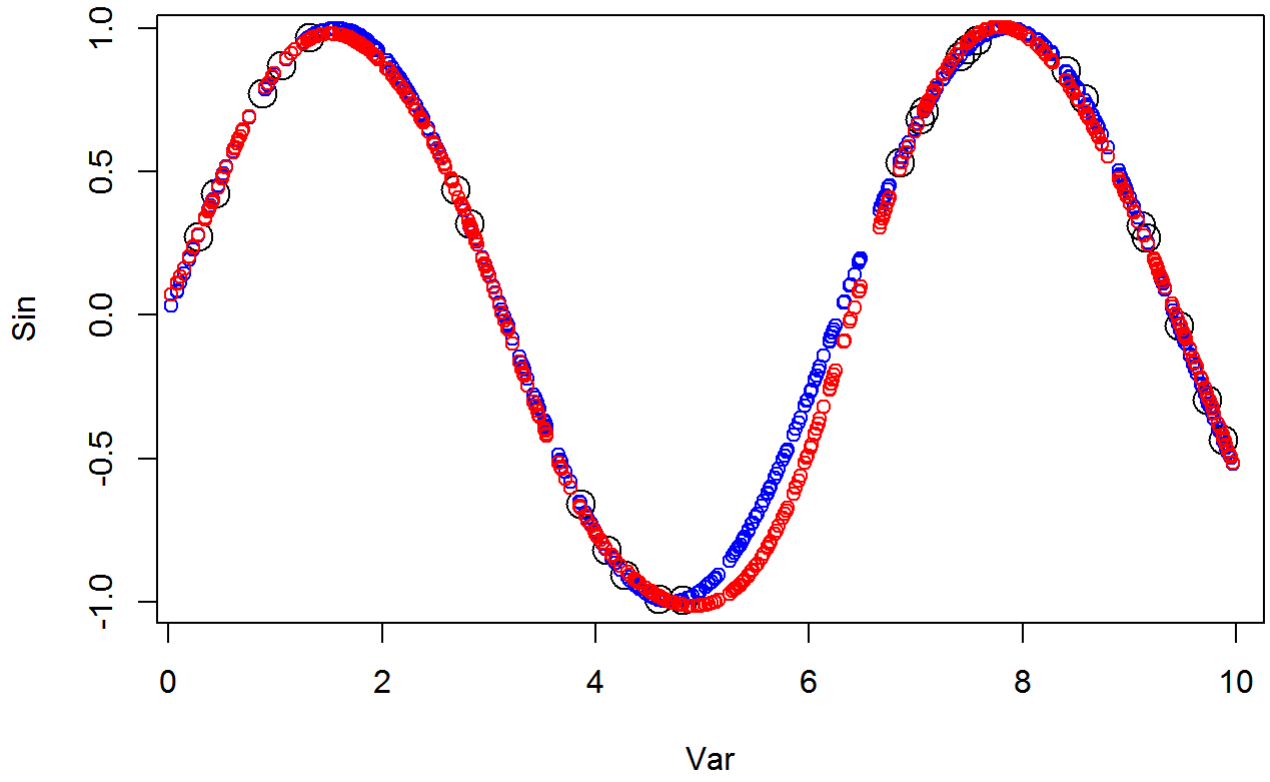
Question.1

With the help of runif function we get random uniform variables with interval (0,10) 1 to 25 data for train and then rest one for test. to learn trigonometry sign function we do by train a neural network. we used 10 hidden layers as we asked in question. after getting predicted values with the help of prediction function we use plot all the three data set on same graph which we did with the help of plot function and then with the help of points function.

```
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 4.2.2
```

```
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

train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10)
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue" , cex= 1)

points(te[,1], p_values, col = "red" , cex= 1)
```



for plot train data points are represented by black test data points are represented by blue predict ones are represented by red.

from above graph we can see that test and train data points are overlapping but for predictive values we only see a slight change from range (4, 6) but this change isn't that big maybe a slight difference between them.

so from above graphical result we can say the predicted values for sign functions are good ones. we get desire learning for sign function.

Question.2

Linear

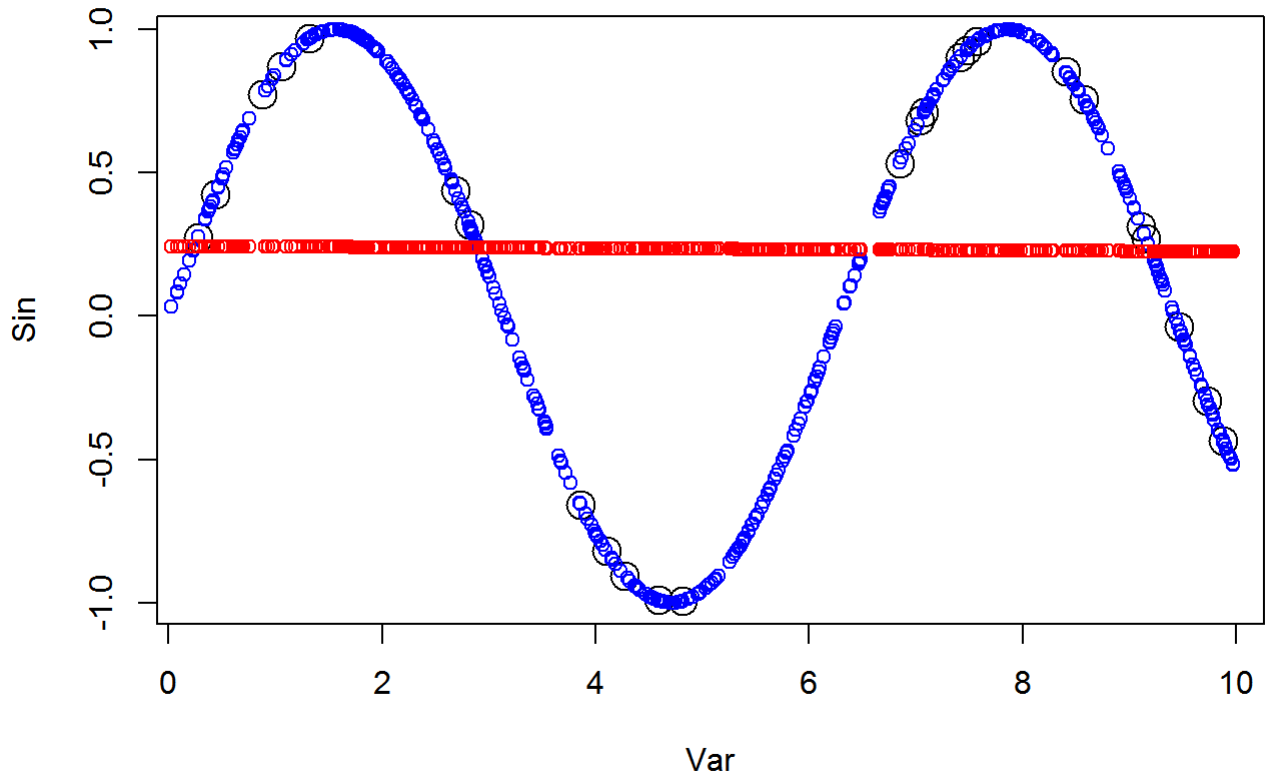
In this part we have asked to replace the default active function with specific functions we are given with first one is linear function. as we know linear equation always represent by.

$$y = mx + c$$

so for each values for x we always get y values. we also know the property of linear function it gives us straight line when we try to plot it on graph.

repeating same thing just changing active function.

```
linear_function <- function(x){
  y = x
}


train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10, act.fct =linear_function )
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue", cex= 1)

points(te[,1], p_values, col = "red" , cex= 1 )
```

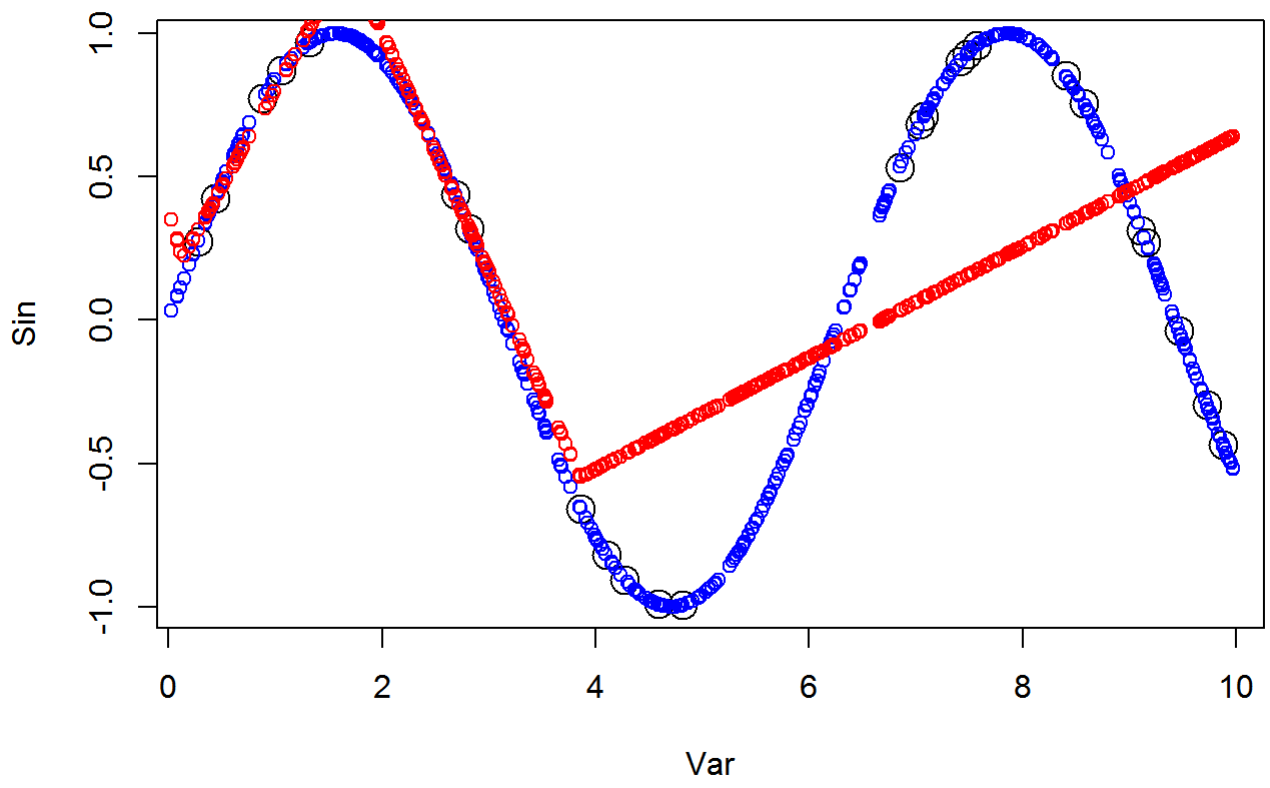


From above plot we can see the changes in the plot this is because we specify the active function. which converted the predicted values into straight line which is actually property of linear function.

RELU

Next step is to try with function RELU stands for rectified linear unit. This function will simply put all the positive values in linear form. It takes only positive values.

```
Relu_function <- function(x){

  ifelse(x>=0,x,0)

}


train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10, act.fct = Relu_function)
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue" , cex= 1)

points(te[,1], p_values, col = "red" , cex= 1)
```



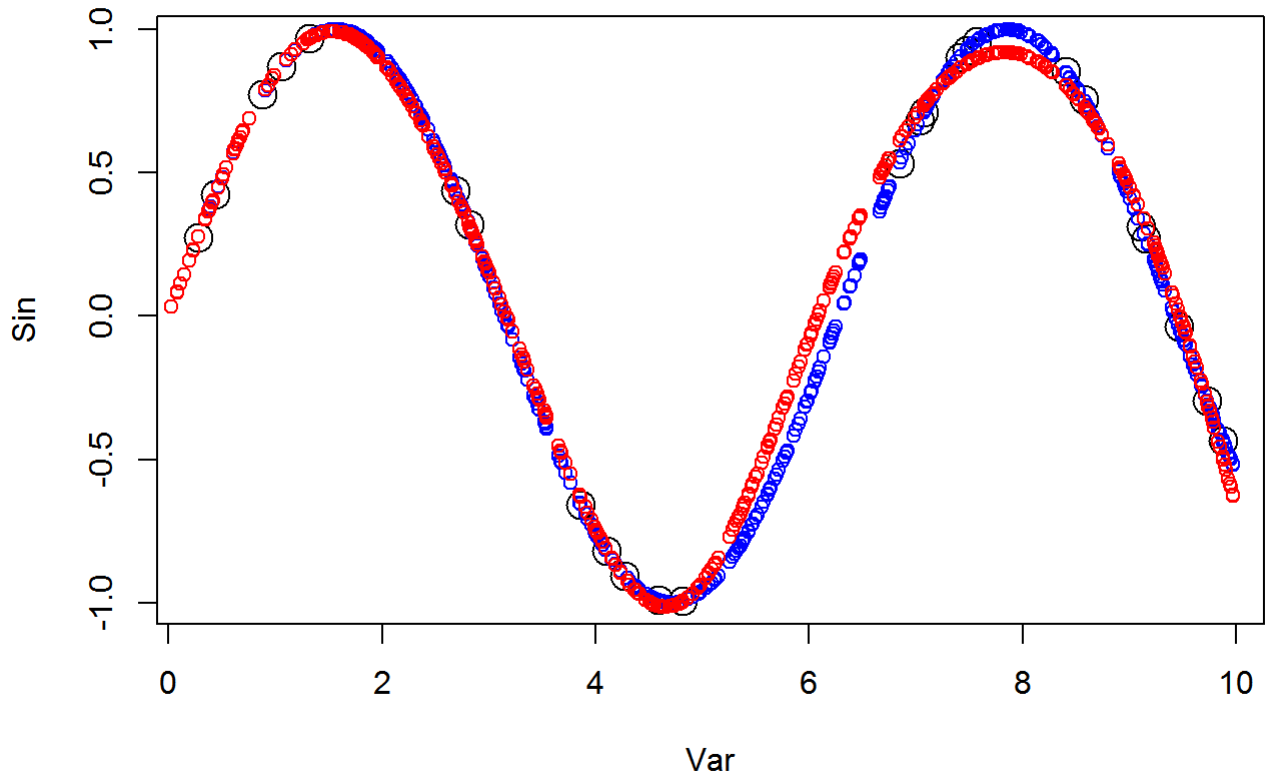
from above results we can see that RElu shows his property and it deviate from negative values and converted them into straight line.

so with relur for all positive values we can see same behaviour of sign functions but it deviate from a point to straight line.

Softplus Activation function

Now we are trying same thing with softplus.

```
soft_plus_function <- function(x){

  y = log(1+exp(x))

}


train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10, act.fct = soft_plus_function)
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue" , cex= 1)

points(te[,1], p_values, col = "red" , cex= 1)
```

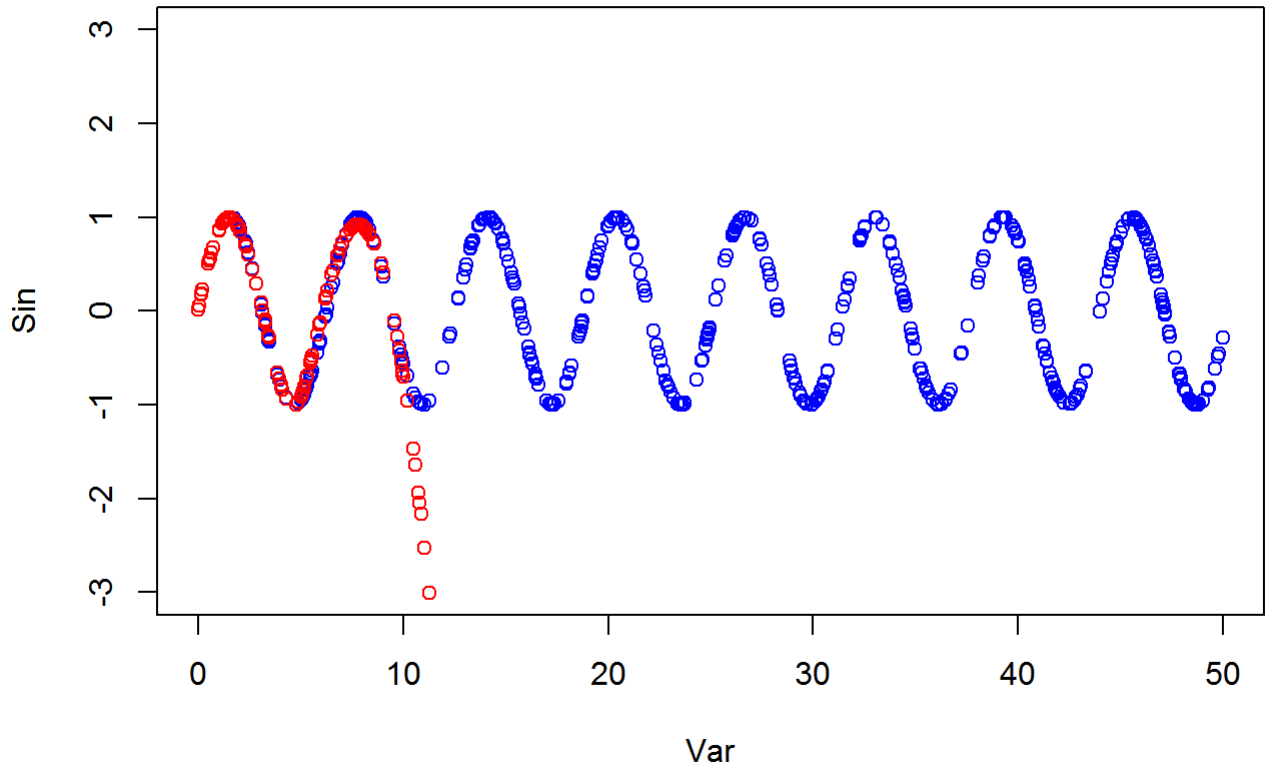


from above we can see that it has same graph like we have one for our prediction values. it because of property of log function. we only see a little changes from point (5,8) which is minor so we can say it same graph.

Question 3

In this question we are going to use another uniform values over (o, 50) interval then we are going to compare it with our previous model.

```
Var <- runif(500, 0, 50)
mydata <- data.frame(Var, Sin=sin(Var))
p_values <- predict(train_nn, mydata  )
plot(mydata, col = "blue", ylim = c(-3,3))
points(mydata[,1],p_values , col="red")
```



from above graph we can see that we change the interval but we didn't train the data for new interval we use the train data for (0,10) and then plot both but we see right after 10th interval we see deviation for our predicted part because it wasn't trained.

Question 4

```
train_nn$weights
```

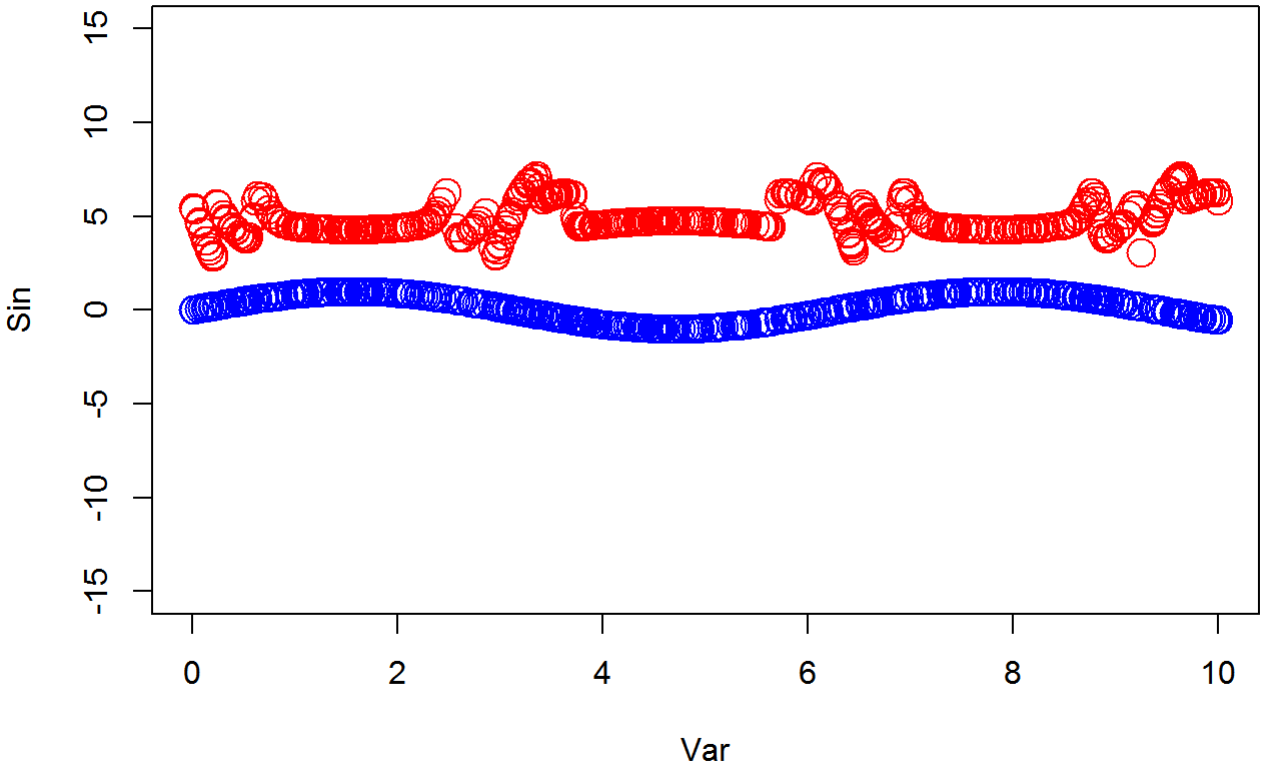
```
## [[1]]
## [[1]][[1]]
##            [,1]      [,2]      [,3]      [,4]       [,5]       [,6]       [,7]
## [1,]  3.8599990 11.112818  2.163115 0.3997523 -1.5990922  1.5033304 -3.8136388
## [2,] -0.5371294 -2.343492 -1.584245 0.8377530  0.9507098 -0.9973926  0.4285792
##           [,8]       [,9]      [,10]
## [1,] 0.8231909  0.9939293 -0.3219623
## [2,] 1.3045852 -0.3508799 -1.1641424
##
## [[1]][[2]]
##              [,1]
##  [1,]  -0.2905792
##  [2,]  -2.3704506
##  [3,]   1.3106358
##  [4,]  -2.6157653
##  [5,]   1.8558796
##  [6,]   1.1405753
##  [7,]   0.7944774
##  [8,] -11.3625655
##  [9,]  -1.0273628
## [10,]  -1.2547845
## [11,]   1.3251712
```

Above results is for trained NN. for trained data we get some neurons weights which we can see in above explanation but if we talk about interval of (0,50) we didn't train the data that's why we get deviation in sign graph from interval (10) to onwards.

Question 5

For this question we have asked for change of prediction before in first part they asked for to predict sin(x) from some values of x but this time we are going to predict x from sin(x) with taking whole data as a trian data.

```
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))

tr <- mydata[1:500,] #training


train_nn <- neuralnet(formula = Var~Sin, data = tr, hidden = 10, threshold = 0.1)
p_values <- predict(train_nn, tr)

plot(mydata, col = "blue", cex=2, ylim = c(-15,15))
points(tr[,1], p_values , col="red", cex=2)
```



We try to do prediction by changing of functions but this doesn't seem goes well it's mean if we try to predict a function for some values then we can get good results but if we try to predict values with the help of function then definitely we will get bad results above graph is example of bad prediction of values from function.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
######################## Assignment 1 ##########################
library(ggplot2)
library(geosphere)
set.seed(12345)
stations <- read.csv("stations.csv", fileEncoding="latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

#smoothing coefficient or width
h_distance <- 30000
h_day <- 7
h_time <- 12

#predicted latitude
a <- 58.4108
#predicted longitude
b <- 15.6214
date <- as.Date("2016-5-30")
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00",
           "16:00:00", "18:00:00", "20:00:00", "22:00:00", "24:00:00")
temp <- length(times)

#Physical distance from a station to the point of interest
lat_long<-cbind(st$latitude,st$longitude)
dist_loc <- distHaversine(lat_long, c(a,b))
# calculating "u" for location
u_dist <- dist_loc / h_distance
#Gaussian Kernels for location
K_dist <- exp(-(u_dist)^2)

#Distance between the day a temperature measurement was made and the day of interest
st$date <- difftime(as.Date(st$date),as.Date(date),units = "days")
st$date <- as.numeric(abs(st$date))
# calculating "u" for days
u_day <- st$date / h_day
#Gaussian Kernels for for days
K_day <- exp(-(u_day)^2)

#Formatting the time for consecutive hours
times1 <- as.POSIXct(times,format="%H:%M:%S")
st$time <- as.POSIXlt(st$time, format="%H:%M:%S")

total_kernel_sum <- c()
total_kernel_mul <- c()
df_weather <- data.frame() # final dataset of predicted values

for(i in 1:temp){
  # Distance between the hour of the day a temperature measurement was made and the hour of inte
rest
  dist_time <- as.numeric(difftime(st$time,times1[i], units = "hours"))
  dist_time <- abs(dist_time)
```

```
  dist_time[dist_time > 12] = 24 - dist_time[dist_time > 12]
  # calculating "u" for time
  u_hour <- dist_time / h_time
  #Gaussian Kernel for time
  K_hour <- exp(-(u_hour)^2)
  # Summing three Gaussian kernels
  kernal_sum <- K_dist + K_day + K_hour
  # Multiplying three Gaussian kernels
  kernal_mul <- K_dist * K_day * K_hour
  total_kernel_sum[i] <- sum(kernal_sum*st$air_temperature) / sum(kernal_sum)
  total_kernel_mul[i] <- sum(kernal_mul*st$air_temperature) / sum(kernal_mul)
  df <- data.frame(TIME = times[i], KERNAL_SUM = total_kernel_sum[i], KERNAL_MUL = total_kernel_
mul[i])
  df_weather <- rbind(df_weather, df)
}
knitr::kable(df_weather)
#Plot kernal_sum
ggplot(df_weather, aes(x=seq(4, 24, 2))) +
  geom_line(aes(y=total_kernel_sum), color="green") +
  geom_point(aes(y=total_kernel_sum), color="black") +
  xlab("Time") +
  ylab("predicted temperature") +
  ggtitle("Predicted temperature using kernel sum") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5), legend.position = "right")
#plot kernal_mul
ggplot(df_weather, aes(x=seq(4, 24, 2))) +
  geom_line(aes(y=total_kernel_mul), color="red") +
  geom_point(aes(y=total_kernel_mul), color="black") +
   xlab("Time") +
  ylab("predicted temperature") +
  ggtitle("Predicted temperature using kernel multiplication") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5), legend.position = "right")
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,scal
ed=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,scal
ed=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,sc
aled=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,sc
aled=FALSE)
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,sc
aled=FALSE)
alphaindex(filter3)
coef(filter3)
-b(filter3)
rbfkernel<-rbfdot(sigma = 0.05)
sv<-alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
k<-NULL

for (i in 1:10){
  k2<-NULL
  for(j in 1:length(sv)){
    k2<-c(k2, rbfkernel(as.vector(unlist(spam[i,-58])),as.vector(unlist(spam[sv[j],-58]))))
  }
  k<-c(k, sum(co*k2)+inte)
}

Predicted<-as.vector(predict(filter3,spam[1:10,-58], type = "decision"))
Calculated<-k
Index<-1:10
```

```
df<-data.frame(Index,Predicted,Calculated)
df
library(neuralnet)
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

train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10)
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue" , cex= 1)

points(te[,1], p_values, col = "red" , cex= 1)

linear_function <- function(x){
  y = x
}


train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10, act.fct =linear_function )
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue", cex= 1)

points(te[,1], p_values, col = "red" , cex= 1 )



Relu_function <- function(x){

  ifelse(x>=0,x,0)

}


train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10, act.fct = Relu_function)
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue" , cex= 1)

points(te[,1], p_values, col = "red" , cex= 1)
```

```
soft_plus_function <- function(x){

  y = log(1+exp(x))

}


train_nn <- neuralnet(formula = Sin ~Var, data = tr, hidden = 10, act.fct = soft_plus_function)
p_values <- predict(train_nn, te)

plot(tr, col = "black", cex= 2)

points(te, col = "blue" , cex= 1)

points(te[,1], p_values, col = "red" , cex= 1)


Var <- runif(500, 0, 50)
mydata <- data.frame(Var, Sin=sin(Var))
p_values <- predict(train_nn, mydata  )
plot(mydata, col = "blue", ylim = c(-3,3))
points(mydata[,1],p_values , col="red")

train_nn$weights

Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))

tr <- mydata[1:500,] #training


train_nn <- neuralnet(formula = Var~Sin, data = tr, hidden = 10, threshold = 0.1)
p_values <- predict(train_nn, tr)

plot(mydata, col = "blue", cex=2, ylim = c(-15,15))
points(tr[,1], p_values , col="red", cex=2)
```