```
## ----setup,
include=FALSE------------------------------------------------------
knitr::opts_chunk$set(echo = TRUE)


## ---- warning=FALSE,
echo=FALSE---------------------------------------------------
set.seed(1234567890)
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")

date <- as.Date("2013-11-04") # The date to predict (up to the students)
filtered_temps <- temps[temps$date < as.Date(date),] #Filter the tempratures
to discard irrelevant dates.

st <- merge(stations,filtered_temps,by="station_number")
# These three values are up to the students
h_distance <- 40000 #Unsure how much we want to consider stations further
away.
h_date <- 9
h_time <- 3
a <- 58.4274 # The point to predict (up to the students) #latitud
b <- 14.826 #longitud
pos_vec <- cbind(b,a)

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")
times_numbers <-c(4,6,8,10,12,14,16,18,20,22,24) #for the plot
temp <- vector(length=length(times))





## ---- warning=FALSE,
echo=FALSE---------------------------------------------------

# Studentsâ\ code here
#physical distance with distHaversine
st_loc<-cbind(st$longitude,st$latitude)
dist_hav<-distHaversine(p1=pos_vec,p2=st_loc)
m<-cbind(dist_hav)

#Gaussian Kernel
#(x_* - x_i)/h from lectures
#diff represents (x_* - x_i)
gaussian_kernel<-function(diff, h_val) {

  u <- diff/h_val
  return(exp(-u*u))
```

```
}


relative_day_dist <- function(d1,d2){
  diff<- as.Date(d1) - as.Date(d2) #Difference between our date and the date
we're comparing (in days).
  return (as.numeric(diff))
}

relative_hour_dist <- function(time1, time2) {
  time_obj1 <-strptime(time1,format="%H:%M:%S") #Create time object so that we
can extract hour
  time_obj2 <-strptime(time2,format="%H:%M:%S")
  h1<-as.integer(format(time_obj1,"%H")) #take the hour value as an integer
  h2<-as.integer(format(time_obj2,"%H"))
  # Convert hours to minutes
  minute1 <- h1 * 60
  minute2 <- h2 * 60

  # Compute the absolute difference in minutes
  minute_diff <- abs(minute1 - minute2)

  # Compute the relative distance in hours
  hour_diff <- minute_diff / 60

  return(hour_diff)
}



## ---- warning=FALSE,
echo=FALSE-------------------------------------------------

#Calculations
predictions = rep(0,11)
k_distance <- gaussian_kernel(dist_hav,h_distance)
k_days <- gaussian_kernel(relative_day_dist(date,filtered_temps$date),h_date)


for (i in 1:length(temp)) {
  rel_h<-relative_hour_dist(times[i],filtered_temps$time)
  k_time <-
gaussian_kernel(relative_hour_dist(times[i],filtered_temps$time),h_time)
  k_sum <- cbind(k_distance + k_days + k_time)
  k_sum <- (k_sum/sum(k_sum)) # Normalize the values in the k_sum matrix by
dividing each element by the sum of the elements in each row
  weighted_temps <- k_sum * filtered_temps$air_temperature #get weighted
temperatures

  predictions[i] <- sum(weighted_temps)


}
```

```
plot(times_numbers,predictions,type="o",xlab = "Time of day (hours)",ylab =
"Predicted temp",main = "Prediction of temperature using sum")


#Used to look at h values
#plot(rel_h,k_time)
#plot(dist_hav,k_distance,xlim=c(0,100000))
#plot(relative_day_dist(date,filtered_temps$date),k_days,xlim=c(0,100))





## ---- warning=FALSE,
echo=FALSE--------------------------------------------------
predictions2 = rep(0,11)
k_distance <- gaussian_kernel(dist_hav,h_distance)
k_days <- gaussian_kernel(relative_day_dist(date,filtered_temps$date),h_date)



for (i in 1:length(temp)) {
  rel_h<-relative_hour_dist(times[i],filtered_temps$time)
  k_time <-
gaussian_kernel(relative_hour_dist(times[i],filtered_temps$time),h_time)
  k_sum <- cbind(k_distance * k_days * k_time)
  k_sum <- (k_sum/sum(k_sum)) # Normalize the values in the k_sum matrix by
dividing each element by the sum of the elements in each row
  weighted_temps <- k_sum * filtered_temps$air_temperature #get weighted
temperatures

  predictions2[i] <- sum(weighted_temps)


}
plot(times_numbers,predictions2,type="o",xlab = "Time of day (hours)",ylab =
"Predicted temp",main = "Prediction of temperature using mult")





## ---- warning=FALSE,
echo=FALSE--------------------------------------------------


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=FA
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
print("Error 0")
err0

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

err1

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

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

print("Error 3")
err3



## ----
echo=FALSE---------------------------------------------------------------
# Get the indices of the support vectors in the SVM classifier
sv <- alphaindex(filter3)[[1]]
```

```
# Get the coefficients of the support vectors in the SVM classifier
co <- coef(filter3)[[1]]

# Get the bias term of the SVM classifier and negate it
inte <- -b(filter3)

# Create an RBF kernel with a standard deviation of 0.05
rbfkernel <- rbfdot(sigma = 0.05)

# Initialize an empty vector to store the predicted values
k <- c()

# Loop over the first 10 rows of the spam dataset
for(i in 1:10) {

  # Initialize a variable to store the predicted value for the current row
  k2 <- 0

  # Loop over each support vector
  for(j in 1:length(sv)) {

    # Apply the RBF kernel to the jth support vector and the ith row of the
spam dataset. We unlist the matrixes to perform the calculations.
    f <- rbfkernel(unlist(spam[sv[j], -58]), unlist(spam[i, -58]))

    # Update the predicted value for the current row by adding the product of
the jth coefficient and the value of the kernel. F is a 1x1 matrix.
    k2 <- k2 + co[j] * f[1]
  }

  # Append the predicted value for the current row to the k vector
  k <- c(k, k2 + inte)
}

# Print the k vector
k

# Use the predict function to make predictions on the first 10 rows of the
spam dataset using the trained SVM classifier
prediction = predict(filter3, spam[1:10, -58], type = "decision")

plot(k, col = "red")
lines(prediction)



## ----
echo=FALSE-----------------------------------------------------------------
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
winit = runif(31, -1,1) #weights: one for each of the 10 hidden nodes, plus
one for the bias term for each of the 10 hidden nodes,
                        #plus one for the output node, plus one for the bias
term for the output node

nn <- neuralnet(Sin ~ Var, data = tr,
                hidden = 10, startweights = winit)
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)



## ----
echo=FALSE-------------------------------------------------------------------
library(neuralnet)

# Set seed for reproducibility
set.seed(1234567890)

# Generate data
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test
winit = runif(31, -1,1)

# Define custom activation functions
h1 <- function(x) x
h2 <- function(x) ifelse(x>0,x,0)
h3 <- function(x) log(1 + exp(x))

# Train neural network with custom activation functions
nn1 <- neuralnet(Sin ~ Var, data = tr,
                hidden = 10, act.fct = h1, startweights = winit)

# Train neural network with custom activation functions
nn2 <- neuralnet(Sin ~ Var, data = tr,
                hidden = 10, act.fct = h2, startweights = winit)

# Train neural network with custom activation functions
nn3 <- neuralnet(Sin ~ Var, data = tr,
                hidden = 10, act.fct = h3, startweights = winit)

# Plot results
plot(tr, cex=3, ylim=c(-1.5, 1.5))
points(te, col = "blue", cex=2)
points(te[,1],predict(nn1,te), col="red", cex=1)
points(te[,1],predict(nn2,te), col="green", cex=1)
points(te[,1],predict(nn3,te), col="pink", cex=1)
legend(1, -0.5, legend=c("train", "test","linear", "ReLu", "Softplus"),
       col=c("black","blue","red", "green", "pink"), lty=1:2, cex=0.8)
```

```
## ----
echo=FALSE---------------------------------------------------------------
library(neuralnet)
set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata <- data.frame(Var, Sin=sin(Var))


# Plot of the training data (black), test data (blue), and predictions (red)

prediction = predict(nn,mydata)

plot(mydata, col = "blue", cex=1, ylim=c(-15, 2), xlim=c(0, 55))
points(mydata[,1],prediction, col="red", cex=1, )


smallestIndex = which.min(prediction)

smallestvalue = prediction[smallestIndex]

weights = nn$weights

weights

smallestvalue




## ----
echo=FALSE---------------------------------------------------------------
# Sample 500 points uniformly at random in the interval [0,10]
Var <- runif(500, 0, 10)

# Apply the sine function to each point
mydata <- data.frame(Var, Sin=sin(Var))
otherWayData <- data.frame(Sin=sin(Var), Var)

winit = runif(31, 0,10)

# Use all these points as training points

# Set the target variable to be Var instead of Sin
nn <- neuralnet(Var ~ Sin, data = mydata, hidden = 10, threshold = 0.1,
startweights = winit )

# Plot the predictions of the neural network
plot(te, col = "blue", cex=1,ylim=c(-10, 10), xlim=c(0, 10))
points(tr[,1],predict(nn,tr), col="red", cex=1)
```

```
plot(otherWayData,col = "blue", cex=1) #kanske borde plotta enligt gamla
training data istÃ¤llet?
points(otherWayData[,1],predict(nn,otherWayData), col="red", cex=1)
```