Machine Learning Lab3

Lepeng Zhang, Xuan Wang & Priyarani Patil

2023-12-07

The group report was made based on the discussion after all of us had finished all three assignments.

Assignment 1 was mainly contributed by Lepeng Zhang.

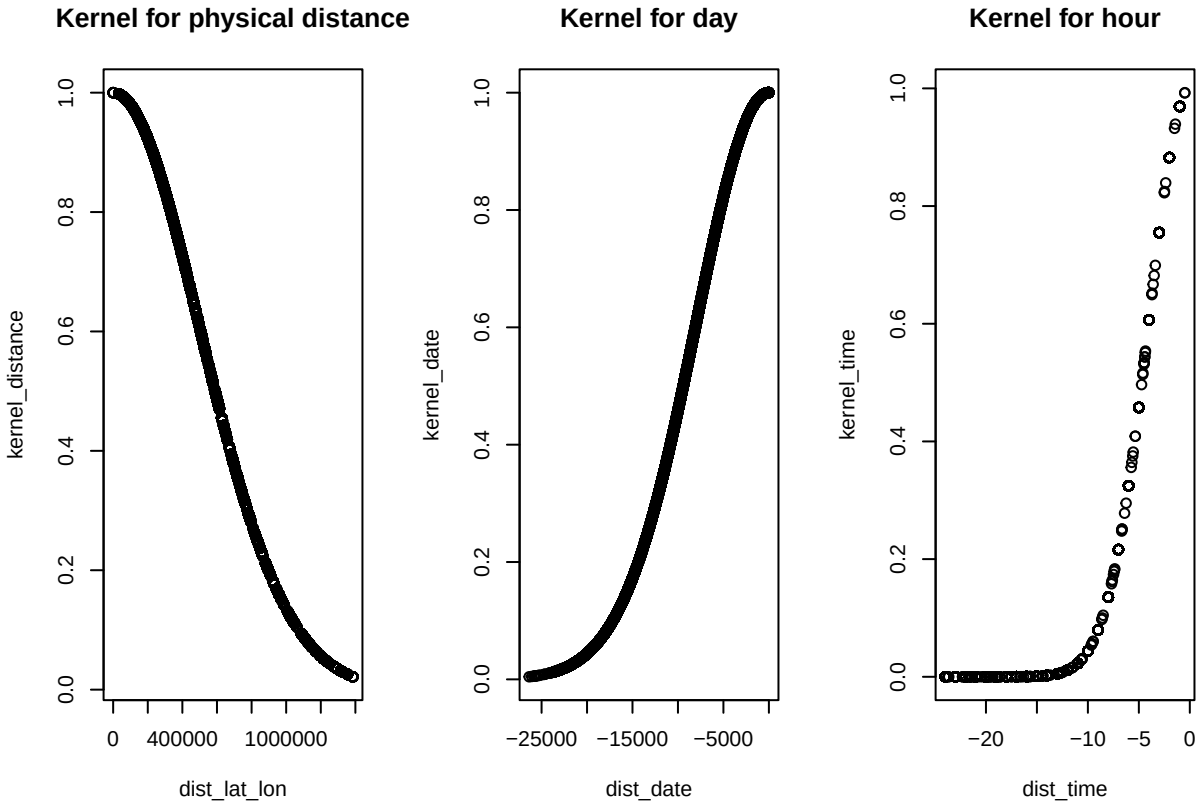
Assignment 2 was mainly contributed by Xuan Wang.

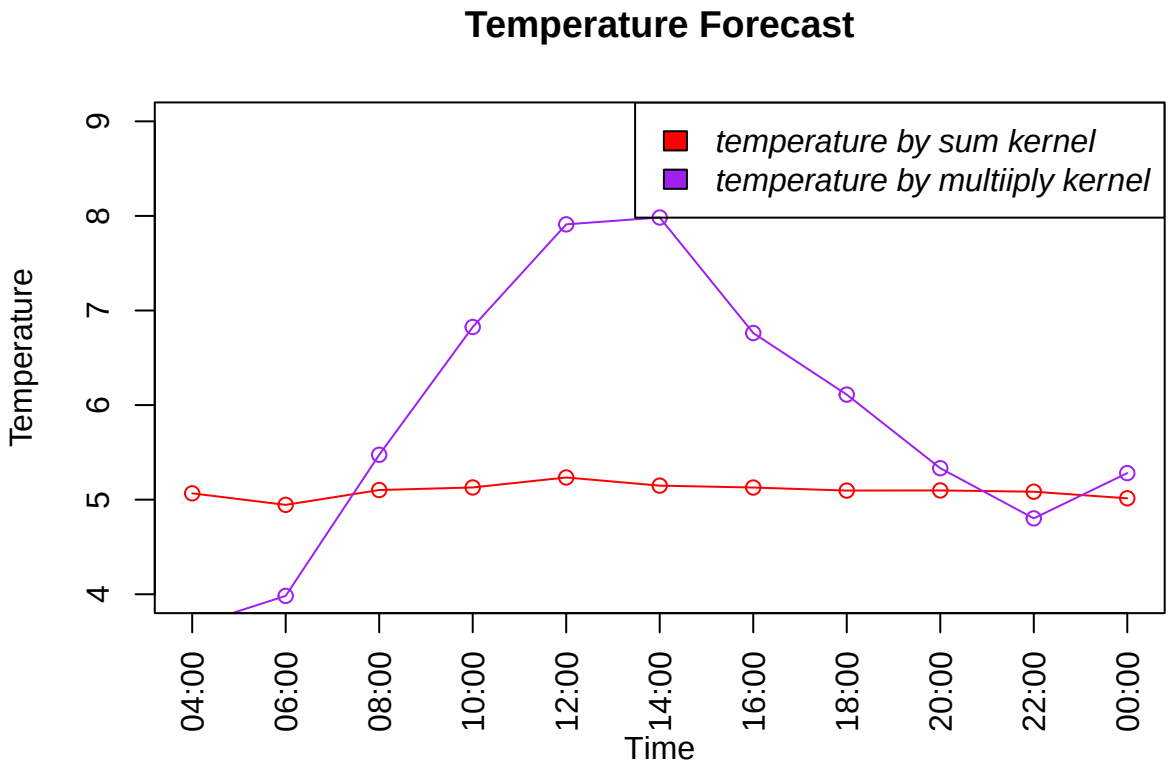
Assignment 2 was mainly contributed by Priyarani Patil.

1. KERNEL METHODS

Q

```
## The legacy packages maptools, rgdal, and rgeos, underpinning the sp package,
## which was just loaded, were retired in October 2023.
## Please refer to R-spatial evolution reports for details, especially
## https://r-spatial.org/r/2023/05/15/evolution4.html.
## It may be desirable to make the sf package available;
## package maintainers should consider adding sf to Suggests:.
```





2. SUPPORT VECTOR MACHINES

Q1

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

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

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

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

Below are details of each filter:

filter0: Trained on the training dataset and validated on the validation dataset.

filter1: Trained on the training dataset and tested on the testing dataset.

filter2: Trained on both the training and validation datasets and tested on the testing dataset.

filter3: Trained on the entire dataset and tested on the testing dataset.

According to R results, err3 $<$ err0 $<$ err2 $<$ err1, although filter3 has the lowest error, it is trained on the entire dataset, including the testing dataset. The model is likely overfitting and the error rate is likely optimistically biased, as the testing data has already been seen during the training process.

On the other hand, filter0, filter1 and filter2 have higher error rates, but they might provide more realistic estimate of the model's performance on the unseen data. Among these, filter0 and filter1 are trained only on the training set, but filter0 is validated on the validation set while filter1 is tested on the test set. filter2 is trained on both the training and validation sets and tested on the test set.

Given these considerations, filter2 might be a good choice to return to the user, as it is trained on a larger portion of the dataset compared to filter0 and filter1 (which could potentially lead to a better model), and unlike filter3, it has not seen the test data during training.

Q2

The estimate of the generalization error returned to the user would be err2, the error of filter2 on the test set. This is because the test set error is often used as an estimate of the model's generalization error.

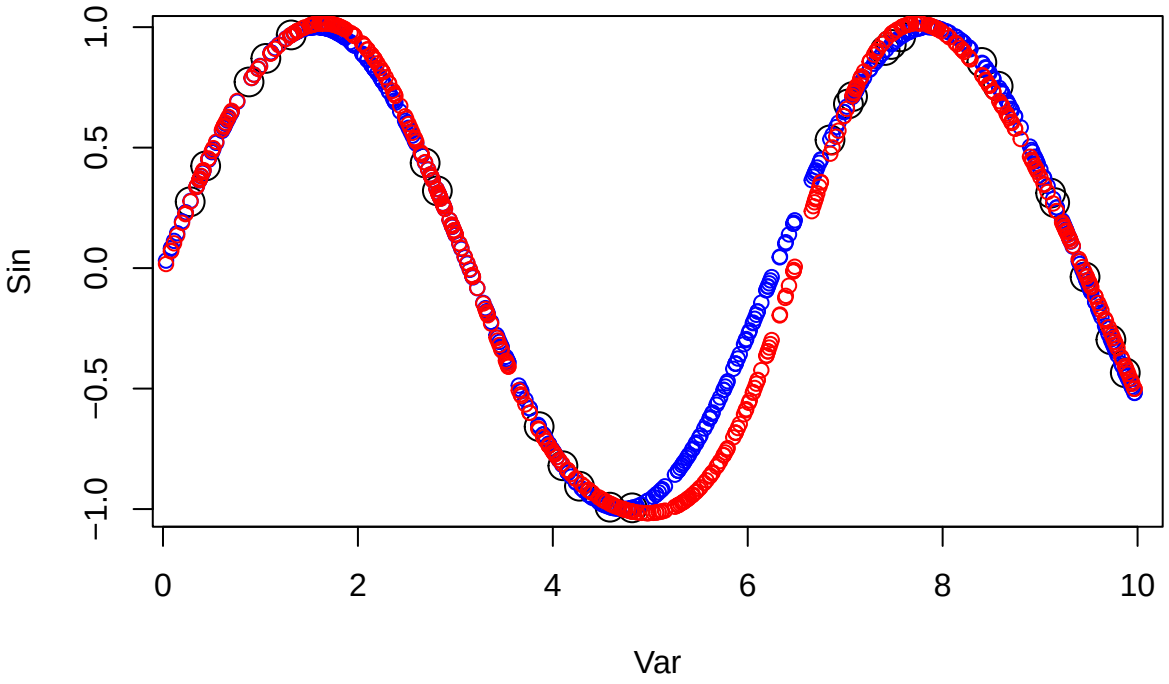
Q3

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

```
##            [,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
```

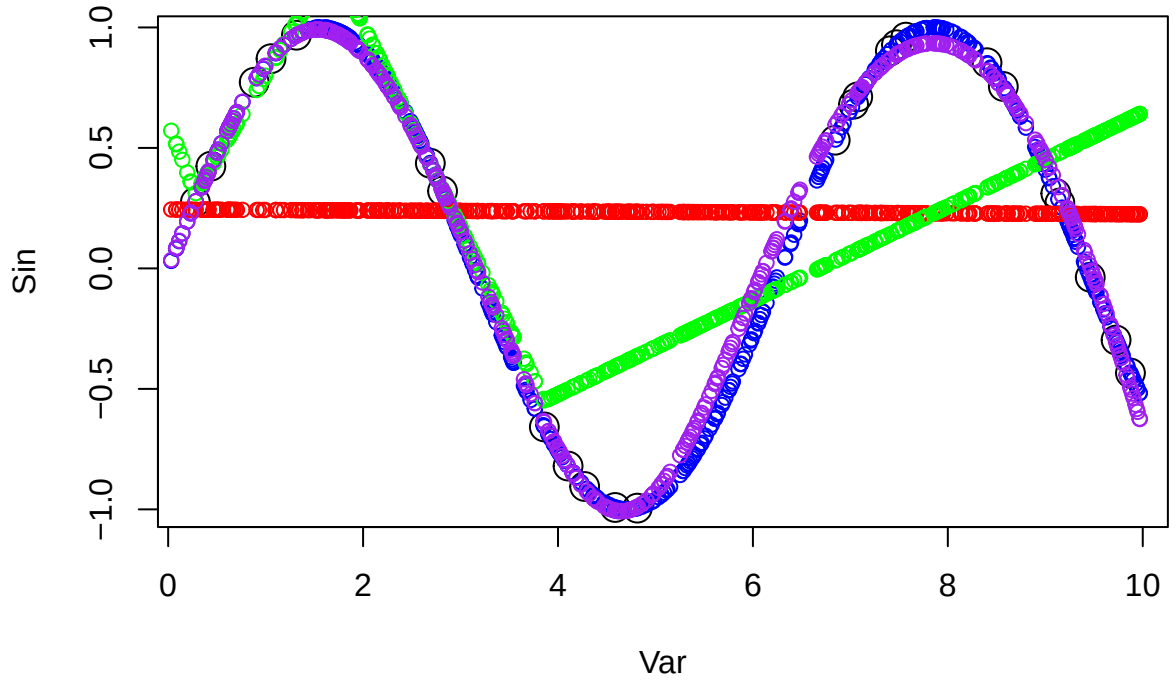
3. NEURAL NETWORKS

Q1



According to the plot, it can be seen that prediction data are nearly matched with the original testing dataset. Beside, the mean squared error is quite small, it indicates that neural network model can predict sin(x) from x very well with 10 hiden units.

Q2



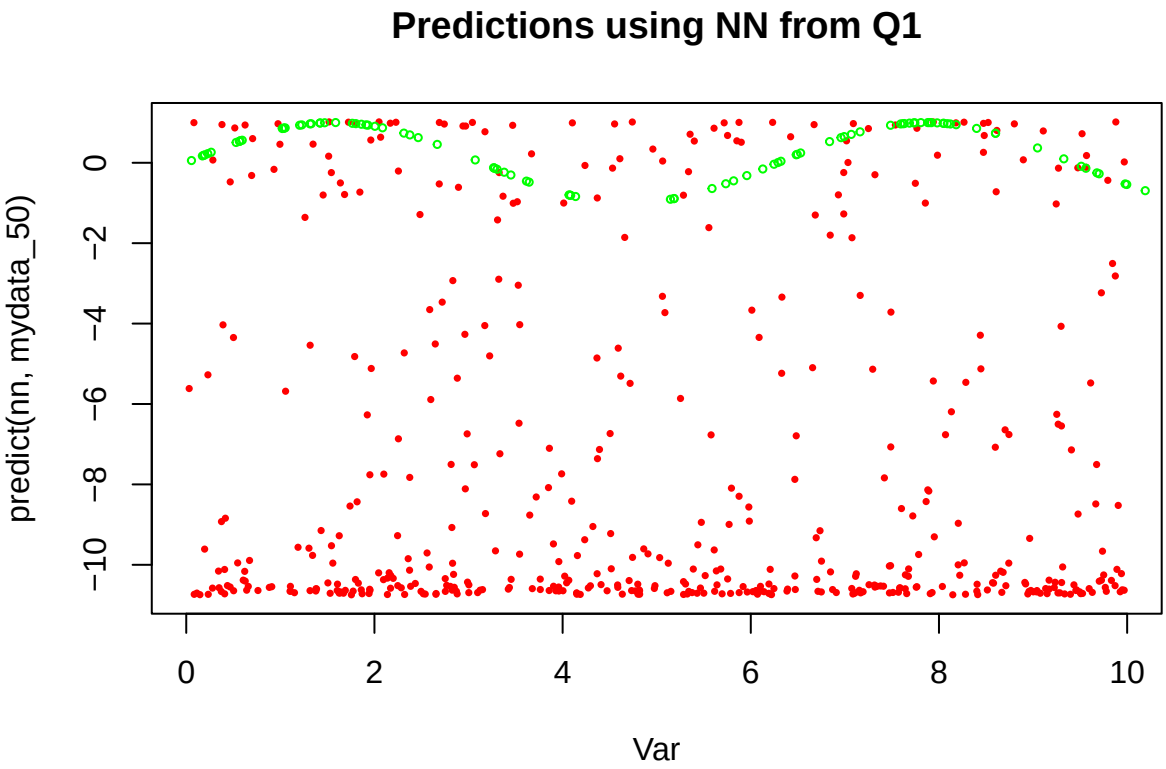
.

According to the plots: The flat line for h1 indicates that the neural network is not learning complex patterns. Linear activation functions are not suitable for capturing non-linear relationships, and the model essentially behaves like a linear regression.

The predicted values from h2 are closer to the real testing values, suggesting that ReLU is able to capture some non-linear features. However, the straighter line indicates that the model might not be capturing more intricate patterns in the data.

The predicted values from h3 are nearly matched with the real testing values. Softplus provides a smooth, differentiable activation function, allowing the model to learn more complex relationships in the data.

Q3



It is observed that predicted values are concentrated around a particular range[-12,0], especially most are -12, is likely because the first neural network model is using the default logistic (sigmoid) activation function, which saturates for certain input ranges, thus the mode might produce predictions that cluster around specific values.

Besides, neural networks model, with limited training dataset, may struggle with accurate predictions for input values significantly outside the training range. They may extrapolate poorly to unseen data.

Q4

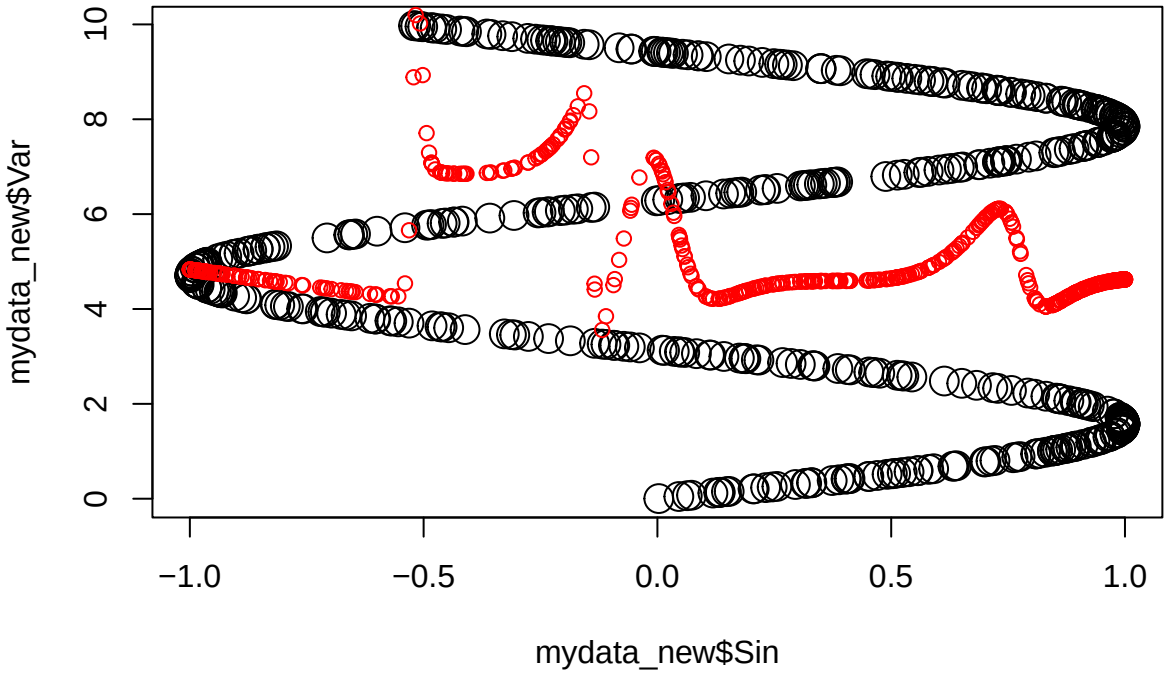
```
## [[1]]
## [[1]][[1]]
##            [,1]       [,2]       [,3]       [,4]      [,5]       [,6]      [,7]
## [1,] -11.870831 -0.9153178 -3.0316259  1.5313105  6.555437 -11.768525  1.397325
## [2,]   4.042494 -0.5368677  0.2603355 -0.5487656 -2.233839   1.787862 -1.259834
##            [,8]       [,9]       [,10]
## [1,]  0.2032402  0.2948804 -0.21326487
## [2,] -2.1974036 -0.5886005 -0.03177278
##
## [[1]][[2]]
##              [,1]
##  [1,]  -0.1088032
##  [2,]   0.7716196
##  [3,]  -1.0436754
##  [4,] -16.0529801
##  [5,]  -1.2718986
##  [6,]   3.2805756
##  [7,]   4.3076898
##  [8,]   0.1220703
```

```
##  [9,]  -1.2234185
## [10,]  -2.7618205
## [11,]   2.3696931
```

When train the network on data in the range [0, 10] and then test it on data in the range [0, 50], the inputs to the neurons in the hidden layer for the test data are likely to be much larger than those for the training data. This is because the weights and biases of the network are adjusted during training to produce the correct output for the training data, and these adjustments depend on the range of the training data.

When these same weights and biases are applied to the test data, the larger inputs result in the neurons in the hidden layer saturating, i.e., they output values very close to 1 or 0. This results in the output of the network converging to a certain value(-12), as the contribution of the saturated neurons to the output becomes constant.

Q5



Appendix

Code for Assignment 1

Q

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

# Define the point and date to predict
a <- 58.4274      # Latitude of the point to predict
b <- 14.826       # Longitude of the point to predict
date <- as.Date("2013-11-04")  # Date to predict
```

```r
# Define the times to predict
times <- seq(from = as.POSIXct("04:00:00", format = "%H:%M:%S"),
             to = as.POSIXct("24:00:00", format = "%H:%M:%S"),
             by = "2 hours")

# Define parameters
h_distance <- 500000  # Smoothing coefficient for physical distance
h_date <- 8000      # Smoothing coefficient for date difference
h_time <- 4       # Smoothing coefficient for time difference

#h_distance <- 300000
#h_date <- 6000
#h_time <- 4

# Initialize the temperature vector
temp_1 <- vector(length = length(times))
temp_2 <- vector(length = length(times))

# Filter out measurements that are posterior to the day and hour of the forecast
st <- st[as.Date(st$date) <= date, ]

# Kernel function
kernel <- function(dist_h, h) {
  exp(-(dist_h^2) / (2 * h^2))
}

# Calculate temperature predictions using sum of three Gaussian kernels and multiply the kernels
for (i in seq_along(times)) {
  # Calculate distances
  dist_lat_lon <- distHaversine(matrix(c(a, b), nrow = 1), matrix(c(st$latitude, st$longitude), nrow = r
  dist_date <- as.numeric(as.Date(st$date) - date)
  dist_time <- as.numeric((as.POSIXct(st$time, format = "%H:%M:%S") - times[i]) / 60)

  # Calculate kernel values
  kernel_distance <- kernel(dist_lat_lon, h_distance)
  kernel_date <- kernel(dist_date, h_date)
  kernel_time <- kernel(dist_time, h_time)

  # Calculate the weights using sum of three Gaussian kernels
  weights_1 <- kernel_distance + kernel_date + kernel_time
  # Calculate the weights using multiply three Gaussian kernels
  weights_2 <- kernel_distance * kernel_date * kernel_time

  # Calculate the predicted temperature
  temp_1[i] <- sum(weights_1 * st$air_temperature) / sum(weights_1)
  temp_2[i] <- sum(weights_2 * st$air_temperature) / sum(weights_2)
}

par(mfrow=c(1,3))
plot(dist_lat_lon, kernel_distance, main = "Kernel for physical distance")
plot(dist_date, kernel_date, main = "Kernel for day")
plot(dist_time, kernel_time, main = "Kernel for hour")
```

```r
# Plot the predicted temperatures
par(mfrow=c(1,1))
plot(times, temp_1, type = "o", col="red",
     ylim = c(4,9),
     xlab = "Time", ylab = "Temperature", main = "Temperature Forecast", xaxt = "n")
lines(times,temp_2, type = "o", col="purple")
axis.POSIXct(1, at=seq(as.POSIXct("04:00:00", format = "%H:%M:%S"), as.POSIXct("24:00:00", format = "%H
legend(x = "topright",  text.font = 3,
       fill= c("red","purple"),
       legend=c("temperature by sum kernel", "temperature by multiiply kernel"))
```

Code for Assignment 2

Q1

```r
library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58])
tr <- spam[1:3000, ]
va <- spam[3001:3800, ]
trva <- spam[1:3800, ]
te <- spam[3801:4601, ]

by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=i,scaled=FALSE)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}

filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0

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

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

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

Q3

```
sv<-alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)
k<-NULL
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  k2<-NULL
  for(j in 1:length(sv)){
    k2<- c(k2, co[j] * exp(-0.05 * sum((spam[sv[j],-58] - spam[i,-58])^2)))
  }
  k<-c(k, sum(k2) + inte)
}
k
predict(filter3,spam[1:10,-58], type = "decision")
```

Code for Assignment 3

Q1

```
# Load the required library
library(neuralnet)

# Set the seed for reproducibility
set.seed(1234567890)

# Generate 500 random points in the interval [0, 10]
Var <- runif(500, 0, 10)

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

# Split the data into training and test sets
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test

# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, min = -1, max = 1)

# Train the neural network
nn <- neuralnet(Sin ~ Var, 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)
```

Q2

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


# Train the neural network with the custom activation functions
nn_h1 <- neuralnet(Sin ~ Var, tr, hidden = 10, startweights = winit, act.fct = h1)
nn_h2 <- neuralnet(Sin ~ Var, tr, hidden = 10, startweights = winit, act.fct = h2)
nn_h3 <- neuralnet(Sin ~ Var, tr, hidden = 10, startweights = winit, act.fct = h3)

# Plot the predictions for each activation function
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1], predict(nn_h1, te), col="red", cex=1)
points(te[,1], predict(nn_h2, te), col="green", cex=1)
points(te[,1], predict(nn_h3, te), col="purple", cex=1)
```

Q3

```
# Generate 500 random points in the interval [0, 50]
Var_50 <- runif(500, 0, 50)

# Apply the sine function to each point
mydata_50 <- data.frame(Var=Var_50, Sin=sin(Var_50))


# Plot new 500 random data points (green), and predictions (red)
plot(Var, predict(nn, mydata_50), col = "red", pch = 16, cex = 0.5, main = "Predictions using NN from Q
points(mydata_50, col = "green", cex=0.5)
```

Q4

```
nn$weights
```

Q5

```
# Generate 500 random points in the interval [0, 10]
Var_new <- runif(500, 0, 10)

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

# Train the neural network that tries to predict x from sin(x)
nn_reverse <- neuralnet(Var ~ Sin, mydata_new, hidden = 10, startweights = winit, threshold = 0.1)


# Plot of the training data (black), predictions (red)
plot(mydata_new$Sin, mydata_new$Var, cex=2)
points(mydata_new$Sin, predict(nn_reverse, newdata = mydata_new), col = "red", cex=1)
```