Laboration report in Machine Learning

Computer lab 3 block 1

732A99

Simge Cinar
Duc Tran
William Wiik

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

17 december 2023

Question 1: KERNEL METHODS

Question: 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).
You are asked to provide a temperature forecast for a date and place in Sweden. The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels:

- The first to account for the physical distance from a station to the point of interest. For this purpose, use the function distHaversine from the R package geosphere.
- The second to account for the distance between the day a temperature measurement was made and the day of interest.
- The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Choose an appropriate smoothing coefficient or width for each of the three kernels above. No cross-validation should be used. Instead, choose manually a width that gives large kernel values to closer points and small values to distant points. Show this with a plot of the kernel value as a function of distance. Help: Note that the file temps50k.csv may contain temper- ature measurements that are posterior to the day and hour of your forecast. You must filter such measurements out, i.e. they cannot be used to compute the forecast.

Finally, repeat the exercise above by combining the three kernels into one by multiplying them, instead of summing them up. Compare the results obtained in both cases and elaborate on why they may differ.

Answer: First we created a new column datetime to merge the date and time, then we eliminated the posterior values from the target datetime. Since temperature in a year is seasonal, we calculated only day differences and ignore the year by ignoring global warming since it has a minor effect. Moreover, we convert hour to 12 values. For example we assumed that 23.00 and 01.00 has 2 hour difference, not 22.

The code is as follows:

```
set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number") # merged dataframe

# Create a new column with date and time
st$datetime <- as.POSIXct(paste(as.character(st$date), as.character(st$time)),
                          format = "%Y-%m-%d %H:%M:%S", tz = "GMT")

# choose manually a width that gives large kernel values to closer points
# and small values to distant points
h_distance <- 50
h_date <- 1
h_time <- 0.4
```

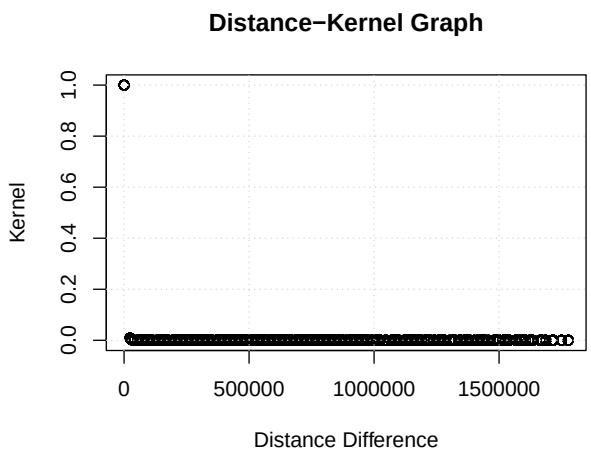
```r
# Predict for each of these times
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00",
           "14:00:00", "16:00:00", "18:00:00", "20:00:00", "22:00:00",
           "24:00:00")

gaussian_kernel <- function(diff_vector, l){
  kernel <- exp(-(diff_vector / (2*(l^2))))
  return(kernel)
}
```

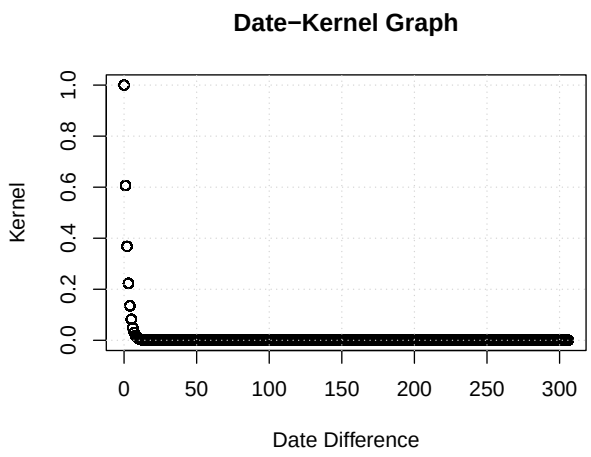
```r
# Point to predict
a <-    55.3836 #latitude
b <- 12.8203     #longitude
date <- "1995-11-03"

i <- 1
target_datetime <- as.POSIXct(paste(date, times[i]), format = "%Y-%m-%d %H:%M:%S", tz = "GMT")
datetime_diff <- target_datetime - st$datetime
st_new <- st[which(datetime_diff > 0), ] # eliminate the posterior

# distance calculation
distance_new <- distHaversine(cbind(st_new$latitude, st_new$longitude), c(a, b))

# date calculation
date1 <- as.Date(date, format = "%Y-%m-%d")
date2 <- as.Date(st_new$date, format = "%Y-%m-%d")

day_of_year1 <- as.numeric(format(date1, "%j"))
day_of_year2 <- as.numeric(format(date2, "%j"))

date_new <- abs(day_of_year2 - day_of_year1)

# hour calculation
st_new$hour <- as.numeric(format(as.POSIXct(st_new$datetime), "%I"))
target_hour <- as.numeric(substr(times[i], 1, 2))
time_new_init <- abs(st_new$hour - target_hour)
time_new <- ifelse(time_new_init < 12, time_new_init, 24-time_new_init) # convert to 12

# kernel calculation
distance_kernel <- gaussian_kernel(distance_new, h_distance)
date_kernel <- gaussian_kernel(date_new, h_date)
time_kernel <- gaussian_kernel(time_new, h_time)

kernel_diff <- c(gaussian_kernel(max(distance_new), h_distance),
                 gaussian_kernel(min(distance_new), h_distance),
                 gaussian_kernel(max(date_new), h_date),
                 gaussian_kernel(min(date_new), h_date),
                 gaussian_kernel(max(distance_new), h_time),
```

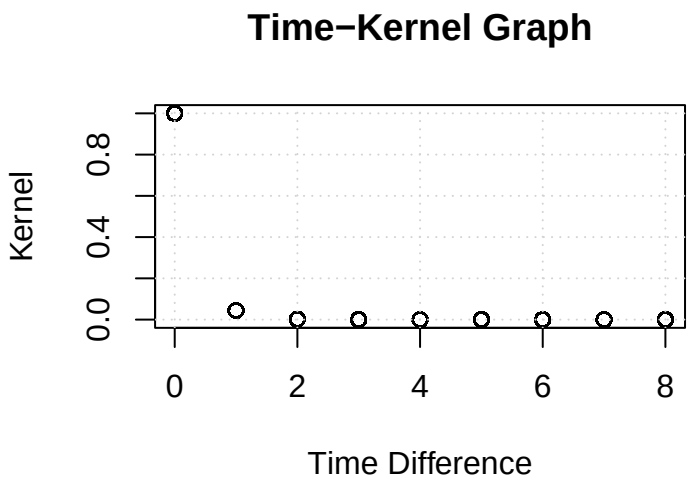
```
                  gaussian_kernel(min(distance_new), h_time))
diff_name <- c("max distance", "min distance",
           "max date", "min date",
           "max time", "min time")

kernel_df <- data.frame("Difference" = kernel_diff)
rownames(kernel_df) <- diff_name
kable(kernel_df, caption = "Kernel Differences")
```

Table 1: Kernel Differences

	Difference
max distance	0
min distance	1
max date	0
min date	1
max time	0
min time	1







The bandwith values are selected 50, 1 and 0.4 for distance, date and time respectively. From the graphs and table above it can be observed that kernel is low when distance is high and vice versa. Also kernel values vary between 0 and 1 which is important because otherwise predictions will be around average temperature value.

The functions to predict temperature is as follows for summation and multiplication:

```
summation_kernel_fnc <- function(target_date, a, b){
  temp <- numeric(length=length(times))
  for (i in 1:length(times)){
    target_datetime <- as.POSIXct(paste(target_date, times[i]),
                                  format = "%Y-%m-%d %H:%M:%S", tz = "GMT")
    datetime_diff <- target_datetime - st$datetime
    st_new <- st[which(datetime_diff > 0), ] # eliminate the posterior

    # distance calculation
    distance_new <- distHaversine(cbind(st_new$latitude, st_new$longitude), c(a, b))

    # date calculation
    date1 <- as.Date(target_date, format = "%Y-%m-%d")
    date2 <- as.Date(st_new$date, format = "%Y-%m-%d")

    day_of_year1 <- as.numeric(format(date1, "%j"))
    day_of_year2 <- as.numeric(format(date2, "%j"))

    date_new <- abs(day_of_year2 - day_of_year1)

    # hour calculation
    st_new$hour <- as.numeric(format(as.POSIXct(st_new$datetime), "%I"))
    target_hour <- as.numeric(substr(times[i], 1, 2))
```

```
    time_new_init <- abs(st_new$hour - target_hour)
    time_new <- ifelse(time_new_init < 12, time_new_init, 24-time_new_init) # convert to 12

    weight_distance <- gaussian_kernel(distance_new, h_distance)
    weight_date <- gaussian_kernel(date_new, h_date)
    weight_time <- gaussian_kernel(time_new, h_time)

    w <- weight_distance + weight_date + weight_time
    total_weight <- sum(w)
    weighted_sum <- sum(w * st_new$air_temperature)

    temp[i] <- weighted_sum/total_weight
  }
  return(temp)
}
```

```
multiplication_kernel_fnc <- function(target_date, a, b){
  temp <- numeric(length=length(times))
  for (i in 1:length(times)){
    target_datetime <- as.POSIXct(paste(target_date, times[i]),
                                  format = "%Y-%m-%d %H:%M:%S", tz = "GMT")
    datetime_diff <- target_datetime - st$datetime
    st_new <- st[which(datetime_diff > 0), ] # eliminate the posterior

    # distance calculation
    distance_new <- distHaversine(cbind(st_new$latitude, st_new$longitude), c(a, b))

    # date calculation
    date1 <- as.Date(target_date, format = "%Y-%m-%d")
    date2 <- as.Date(st_new$date, format = "%Y-%m-%d")

    day_of_year1 <- as.numeric(format(date1, "%j"))
    day_of_year2 <- as.numeric(format(date2, "%j"))

    date_new <- abs(day_of_year2 - day_of_year1)

    # hour calculation
    st_new$hour <- as.numeric(format(as.POSIXct(st_new$datetime), "%I"))
    target_hour <- as.numeric(substr(times[i], 1, 2))
    time_new_init <- abs(st_new$hour - target_hour)
    time_new <- ifelse(time_new_init < 12, time_new_init, 24-time_new_init) # convert to 12

    weight_distance <- gaussian_kernel(distance_new, h_distance)
    weight_date <- gaussian_kernel(date_new, h_date)
    weight_time <- gaussian_kernel(time_new, h_time)

    w <- weight_distance * weight_date * weight_time
    total_weight <- sum(w)
    weighted_sum <- sum(w * st_new$air_temperature)
```

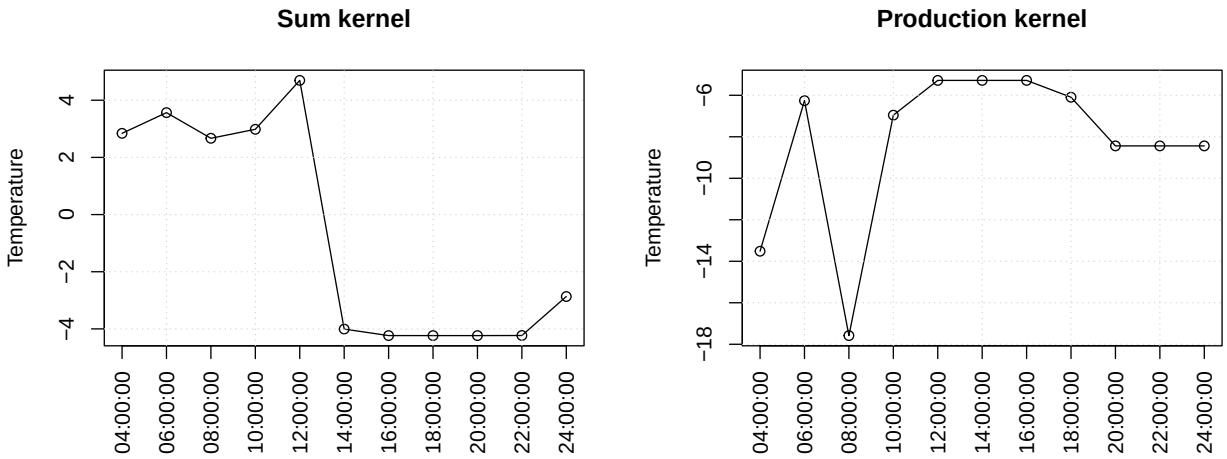
```
    temp[i] <- weighted_sum/total_weight
  }
  return(temp)
}
```

Prediction 1:

```
## Actual temperature is: -39.7 at 06:00:00 in location ( 65.328 , 15.0686 )
```

Table 2: Forecast Low Temperature

Time	sum_kernel_forecast	prod_kernel_forecast
04:00:00	2.842084	-13.515902
06:00:00	3.566205	-6.259956
08:00:00	2.668622	-17.582387
10:00:00	2.983245	-6.958630
12:00:00	4.693804	-5.282525
14:00:00	-4.005696	-5.282525
16:00:00	-4.233613	-5.282528
18:00:00	-4.234063	-6.094086
20:00:00	-4.234058	-8.439311
22:00:00	-4.230975	-8.439345
24:00:00	-2.867028	-8.439345

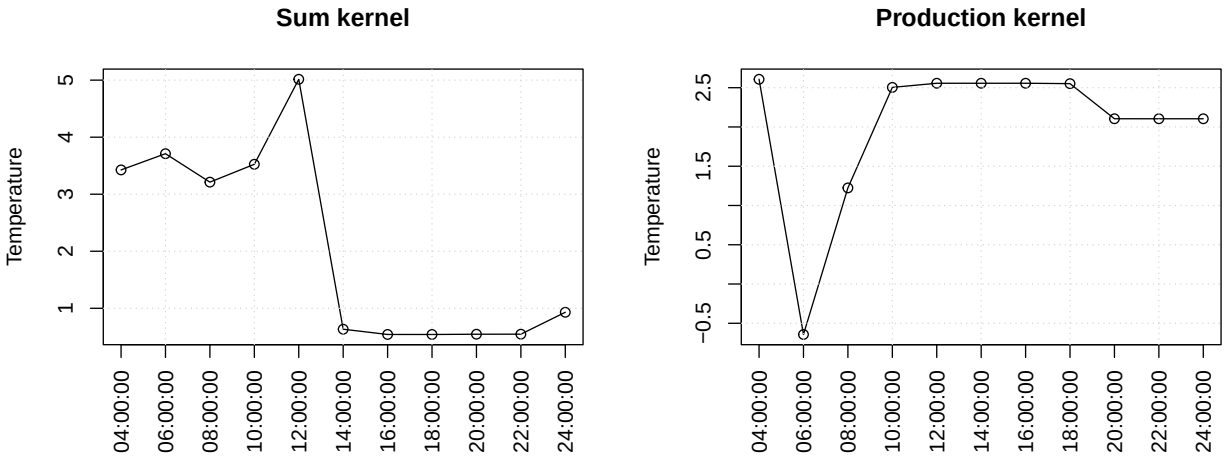


Prediction 2:

```
## Actual temperature is: 2.3 at 03:00:00 in location ( 57.6614 , 18.3428 )
```

Table 3: Forecast 1

Time	sum_kernel_forecast	prod_kernel_forecast
04:00:00	3.4259981	2.6071782
06:00:00	3.7101256	-0.6427259
08:00:00	3.2115129	1.2238477
10:00:00	3.5248091	2.5039748
12:00:00	5.0149447	2.5567314
14:00:00	0.6330000	2.5567314
16:00:00	0.5410482	2.5567313
18:00:00	0.5407263	2.5516399
20:00:00	0.5453067	2.1045378
22:00:00	0.5461107	2.1043899
24:00:00	0.9301687	2.1043899



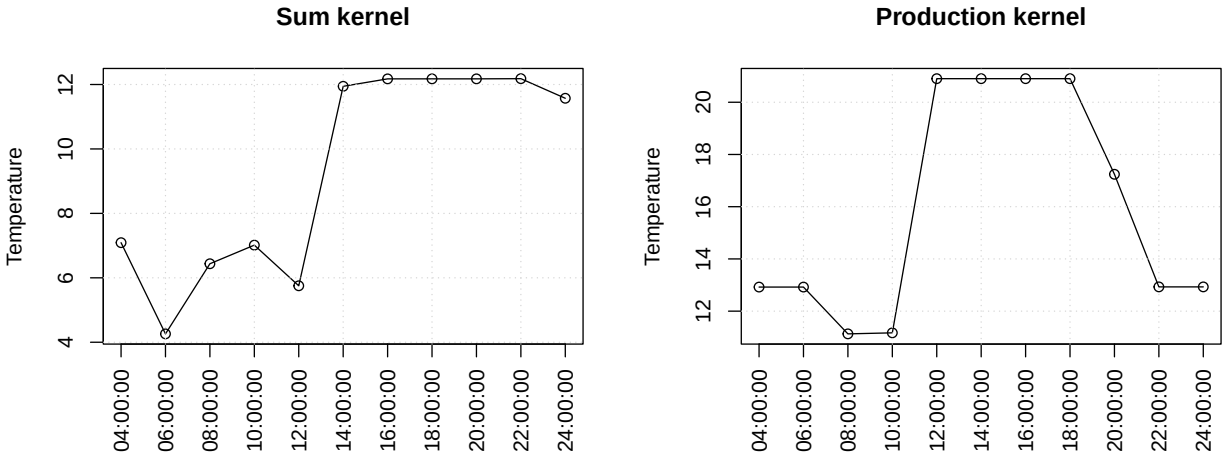
Prediction 3:

```
## Actual temperature is: 13 at 20:00:00 in location ( 60.4889 , 15.4324 )
```

Table 4: Forecast

Time	sum_kernel_forecast	prod_kernel_forecast
04:00:00	7.090685	12.92200
06:00:00	4.259898	12.92200
08:00:00	6.436571	11.12987
10:00:00	7.013998	11.16905
12:00:00	5.752163	20.90262
14:00:00	11.944127	20.90262
16:00:00	12.175107	20.90262

Time	sum_kernel_forecast	prod_kernel_forecast
18:00:00	12.175569	20.90262
20:00:00	12.175567	17.24092
22:00:00	12.181189	12.92910
24:00:00	11.572994	12.92907

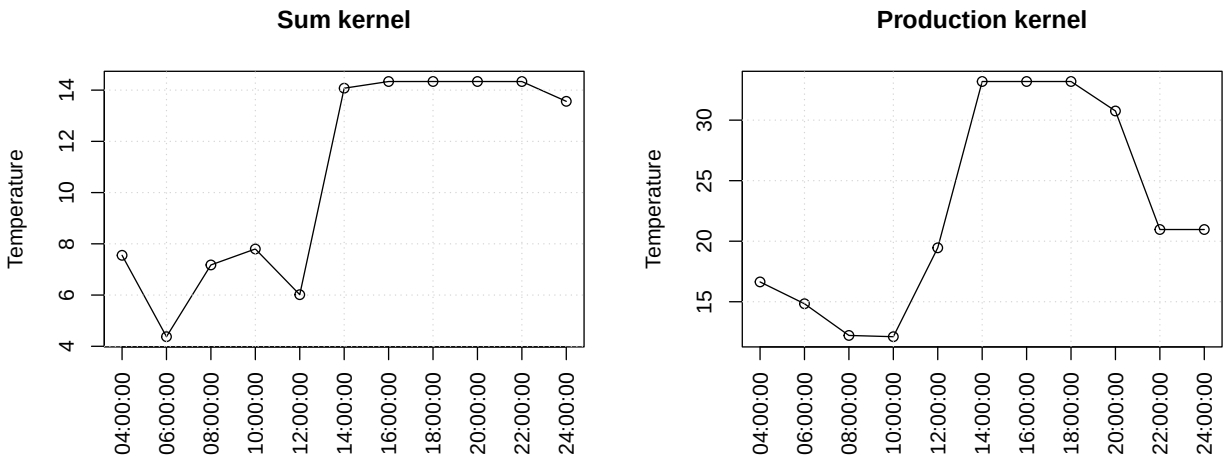


Prediction 4:

```
## Actual temperature is: 33.5 at 12:00:00 in location ( 57.7213 , 16.4683 )
```

Table 5: Forecast High Temperature

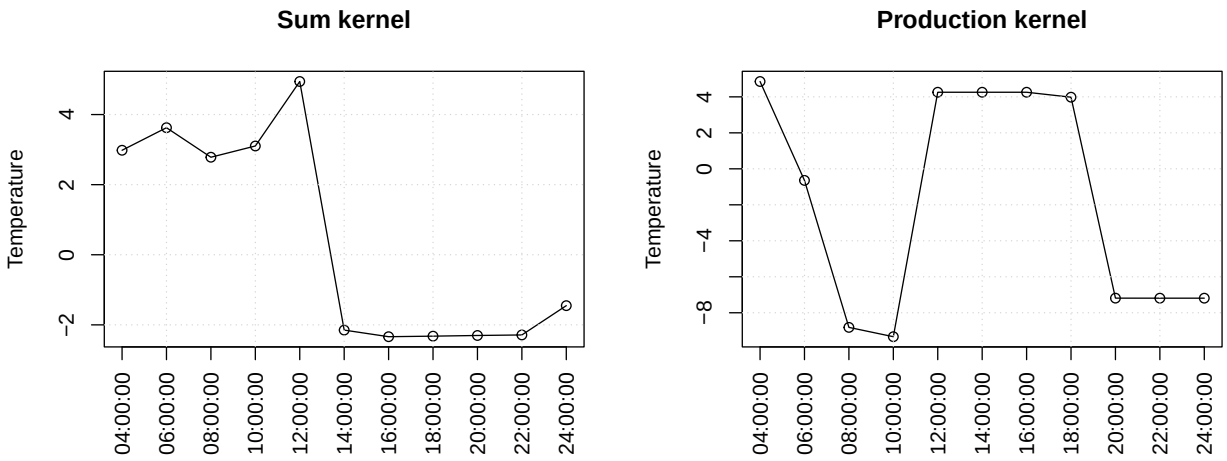
Time	sum_kernel_forecast	prod_kernel_forecast
04:00:00	7.551897	16.63694
06:00:00	4.374456	14.83459
08:00:00	7.175371	12.21617
10:00:00	7.799333	12.11232
12:00:00	6.014420	19.45366
14:00:00	14.074602	33.19088
16:00:00	14.335596	33.19088
18:00:00	14.336116	33.19087
20:00:00	14.336113	30.75624
22:00:00	14.334473	20.96520
24:00:00	13.562189	20.96501



Prediction Winter: Let's forecast the date "2000-02-06" in the coordinates (57.7213, 16.4683)

Table 6: Forecast Winter

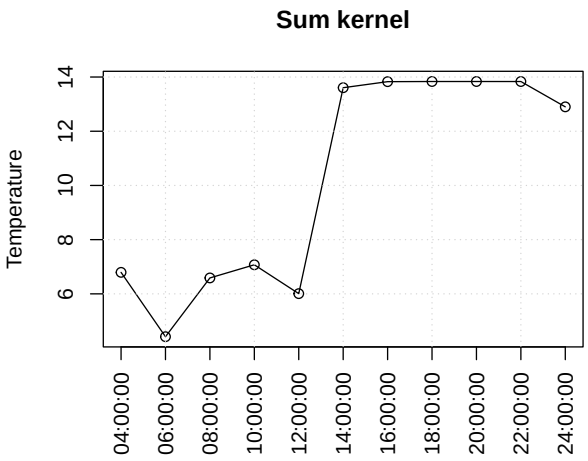
Time	sum_kernel_forecast	prod_kernel_forecast
04:00:00	2.981215	4.8531421
06:00:00	3.624849	-0.6435843
08:00:00	2.782648	-8.8128414
10:00:00	3.102866	-9.3304479
12:00:00	4.942731	4.2560008
14:00:00	-2.149104	4.2560008
16:00:00	-2.337570	4.2559997
18:00:00	-2.322048	3.9815031
20:00:00	-2.302735	-7.1869048
22:00:00	-2.287620	-7.1886401
24:00:00	-1.451600	-7.1886401

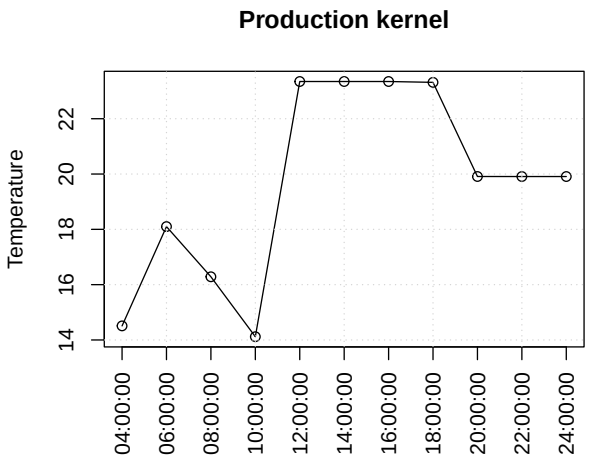


Prediction Summer: Now, let's forecast the date "2000-07-06" in the coordinates (57.7213, 16.4683)

Table 7: Forecast Summer

Time	sum_kernel_forecast	prod_kernel_forecast
04:00:00	6.792954	14.50921
06:00:00	4.419040	18.09955
08:00:00	6.587117	16.28440
10:00:00	7.072164	14.11802
12:00:00	6.007956	23.34599
14:00:00	13.605984	23.34599
16:00:00	13.828866	23.34599
18:00:00	13.834629	23.31511
20:00:00	13.834626	19.91099
22:00:00	13.833734	19.90958
24:00:00	12.899440	19.90958





Conclusion

We cannot say for sure which kernel -summation or multiplication- is better since we don't have a test data but we can make a conclusion using our intuition. It can be observed that temperature predictions is lower in nights and early mornings. Also temperature is lower in winter and higher in summer. By looking at the prediction 1 and 4, it can be concluded that multiplication kernel is better at predicting extreme values since its more sensitive. Multiplication operation emphasizes differences between extreme values more.

Assignment 2. Support Vector Machines

In this assignment, the data set "spam" from the R-package kernlab is used. The dataset consists of 4601 observations, 1 response variable labelling the observation spam or not spam, and 57 numerical variables.

Question 2.1

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

Answer: In this task data was divided into four different data sets:

- Train (3000 observations)
- Validation (800 observations)
- Test (801 observations)
- Train+Validation (3800 observations)

where the last dataset is training and validation combined. The code used to train each filter is as follows:

```
# All the following code is supplied by the task.
#-----------------------------------------------------------------#
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, ]


# Finds the best value of C by using validation data
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)
```

```
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=FAL
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=FAL
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
#-----------------------------------------------------------------#
```

From the code, the value of C for each filter was found by using the validation data set. In table X, the data the filters were trained on, the data used to estimate of the generalization error, and the error are presented.

Table 8: Summarized information about filters

	Training data	Testing data	Error on testing data
Filter 0	training	validation	0.0675
Filter 1	training	test	0.0849
Filter 2	train+val	test	0.0824
Filter 3	all data	test	0.0212

From table X, filter 0 used training data to train, and tested the filter on validation data. Filter 0 can be returned but we will not be able to give an unbiased estimate of the generalization error.
Filter 1 used training data to train, and tested the filter on test data, filter 1 only used each observation of data once and the estimate of the generalization error will be unbiased.
Filter 2 used training and validation data to train. Since validation data was used to find the parameter c, filter 2 will be a bit more overfitted on the training data compared to filter 0 and 1.
Filter 3 used all data to train, this means the model is a overfitted on training data and we will not be able to give an unbiased estimate of the generalization error.

In conclusion, the best filter to return is filter 1 since this is the only filter that only used each observation once.

Question 2.2

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

Answer: In question 2.1, we concluded that filter 0, filter 2, and filter 3 used each observation more than once, and therefore the filter can not give an unbiased estimate of the generalization error. The estimate of the generalization error of the filter should always be the error estimate of the filter we return. In this case filter 1 was chosen to be returned in question 2.1. The generalization error we returned is therefore 0.0849, which means that around 8.49% of future emails will be wrongly classified by this filter.

Question 2.3

Question: 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*.

Answer: In SVM a new observation ($x_\star$) is predicted as:

$$\hat{y}(\mathbf{x}_\star) = \hat{\alpha}^T \mathbf{K}(\mathbf{X}, \mathbf{x}_\star) \tag{1}$$

where $\hat{\alpha}$ are the coefficients for the support vectors and $\mathbf{K}(\mathbf{X}, \mathbf{x}_\star)$ is the kernel values between the support vectors and the observation $x_\star$. The kernel function used for *filter 3* is the Gaussian RBF kernel which is:

$$K(x, x_\star) = exp(-\sigma||x - x_\star||^2) \tag{2}$$

where $\sigma = 0.05$ was used.

```
# Support vectors
sv<-alphaindex(filter3)[[1]]
# Coefficients for the support vectors
co<-coef(filter3)[[1]]
# negative intercept
intercept <- - b(filter3)

# Support vector observations with response variable removed
x <- spam[sv, -58] # Remove y value.

# The 10 observations we need to predict
x_stars <- spam[1:10, -58]

# The Gaussian RBF kernel function
rbfkernel <- rbfdot(sigma = 0.05)
```

```r
k<-c()
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[j] <- co[j]*rbfkernel(unlist(x[j,]), unlist(x_stars[i,]))
  }
  prediction <- intercept + sum(k2)
  prediction
  k[i]<- prediction
}
```

```r
manual_pred <- k
function_pred <- predict(filter3,spam[1:10,-58], type = "decision")

table_data <- data.frame(manual_pred, function_pred)
colnames(table_data) <- c("Manual prediction", "Predict function")
kable(table_data, caption = "Comparision between manual prediction and the predict function for the firs
```

Table 9: Comparision between manual prediction and the predict function for the first 10 observations in the spam data set.

Manual prediction	Predict function
-1.998999	-1.998999
1.560584	1.560584
1.000278	1.000278
-1.756815	-1.756815
-2.669577	-2.669577
1.291312	1.291312
-1.068444	-1.068444
-1.312493	-1.312493
1.000183	1.000183
-2.208639	-2.208639

Comparing the values from manual prediction and the predict function we get the same value for the prediction of the first 10 observations.

Assignment 3. Neural Networks

3.1

Train a neural network to learn the trigonometric sine function. To do so, sample 500 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting value pairs are the data points available to you. Use 25 of the 500 points for training and the rest for test. Use one hidden layer with 10 hidden units. You do not need to apply early stopping. Plot the training and test data, and the predictions of the learned NN on the test data. You should get good results. Comment your results.

```
# The data
set.seed(1234567890)
data_runif <- runif(500, min = 0, max = 10)
data_df <- data.frame(runif = data_runif, sin = sin(data_runif))

tr_data <- data_df[1:25,]
te_data <-  data_df[26:500,]

# Random initialization of the weights in the interval [-1, 1], 20 weights + 11 bias = 31
winit <- runif(31, min = -1, max = 1)
```

```
# Neural Network
nn_model_assignment_1 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10), startweights = winit)

predictions <- predict(nn_model_assignment_1, newdata = te_data) %>% as.vector()
```

```
plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex = 1)
points(te_data[,1], predictions, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)
```

In figure 1, the neural network have successfully learned the sin(x) pattern.

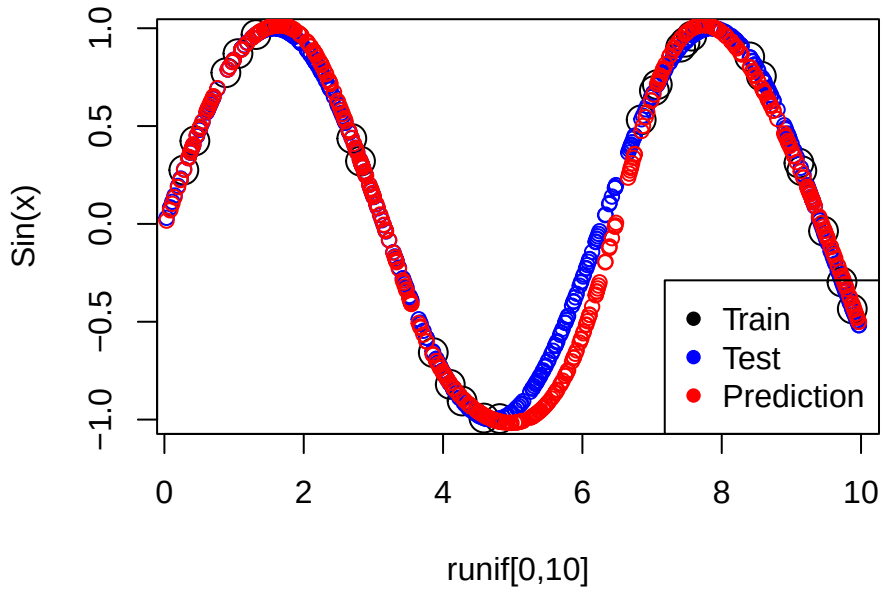


Figure 1: Training data, test data and predictions

3.2

In question (1), you used the default logistic (a.k.a. sigmoid) activation function, i.e. act.fct = "logistic". Repeat question (1) with the following custom activation functions: $h_1(x) = x, h_2(x) = max\{0, x\}$ and $h_3(x) = \ln(1 + e^x)$ (a.k.a. linear, ReLU and softplus). See the help file of the neuralnet package to learn how to use custom activation functions. Plot and comment your results.

```
# Three custom activation function
h1 <- function(x) x
h2 <- function(x) ifelse(x > 0, x, 0) # same as max(0,x)
h3 <- function(x) log(1 + exp(x))


# Neural Network
nn_model_h1 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10),
                         act.fct = h1, startweights = winit)
nn_model_h2 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10),
                         act.fct = h2, startweights = winit)
nn_model_h3 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10),
                         act.fct = h3, startweights = winit)

predictions_h1 <- predict(nn_model_h1, newdata = te_data) %>% as.vector()
predictions_h2 <- predict(nn_model_h2, newdata = te_data) %>% as.vector()
predictions_h3 <- predict(nn_model_h3, newdata = te_data) %>% as.vector()
```

```
plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex=1)
points(te_data[,1], predictions_h1, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)
```

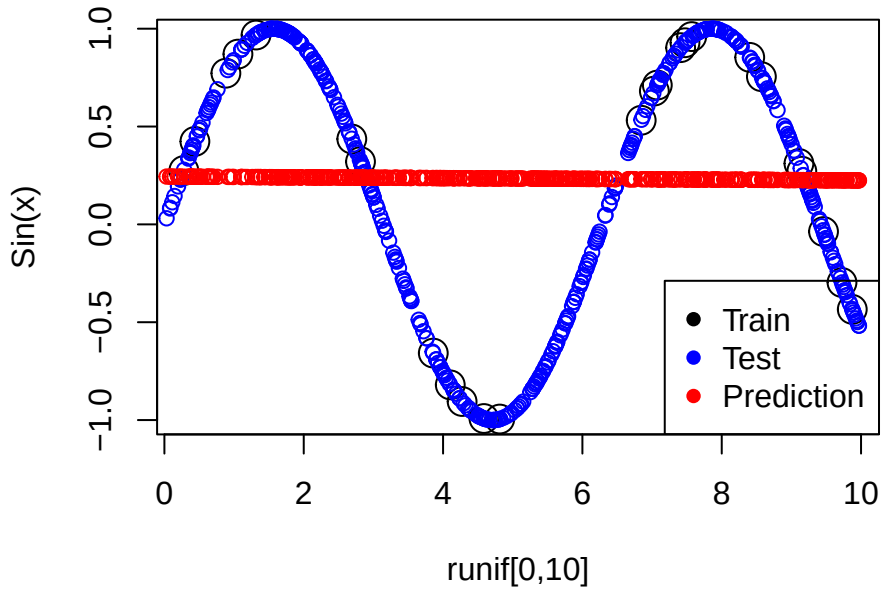


Figure 2: Training data, test data and predictions for first activation function (linear)

In figure 2, we can see that the prediction is just a straight line and does not not fit the test data well. This is because the linear activation function can only represent linear mappings between input and output. Therefore, the prediction does not fit the test data well because the data is non-linear.

```
plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex = 1)
points(te_data[,1], predictions_h2, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)
```

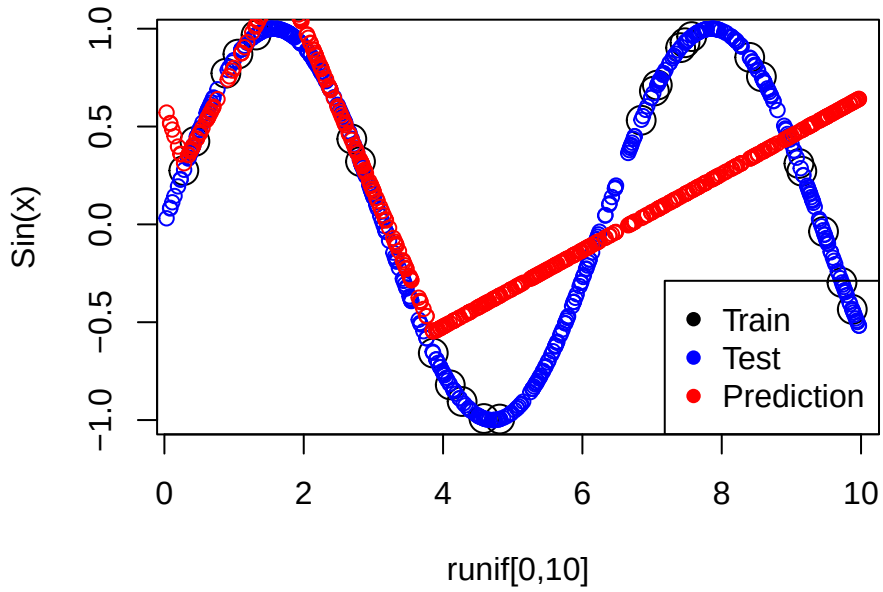


Figure 3: Training data, test data and predictions for second activation function (ReLu)

In figure 3, when we use ReLu as a activation function we get better result than the linear activation function, but still this is very bad fit to the test data.

```
plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex=1)
points(te_data[,1], predictions_h3, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)
```

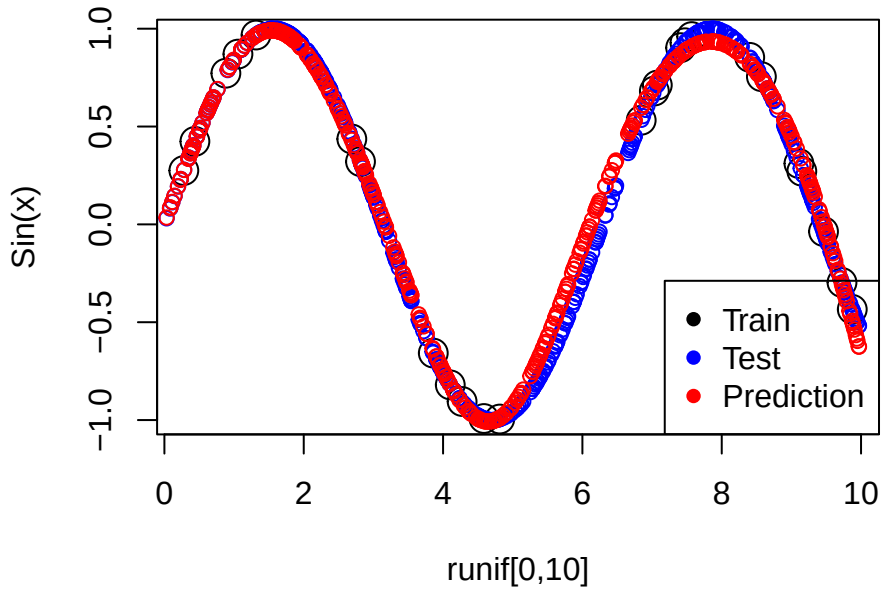


Figure 4: Training data, test data and predictions for third activation function

In figure 4, we use activation function softmax $\ln(1 + e^x)$ and the prediction seems to follow the the test data well. This activation function has the best prediction between the three activation function we have used.

3.3

Sample 500 points uniformly at random in the interval [0, 50], and apply the sine function to each point. Use the NN learned in question (1) to predict the sine function value for these new 500 points. You should get mixed results. Plot and comment your results.

```
# The data
set.seed(1234567890)
data_runif <- runif(500, min = 0, max = 50)
data_df <- data.frame(runif = data_runif, sin =  sin(data_runif))




predictions <- predict(nn_model_assignment_1, newdata = data_df) %>% as.vector()
```

```
plot(tr_data, cex = 2, xlab = "runif[0,50]", ylab = "Sin(x)", ylim = c(-11, 2), xlim = c(0,50))
points(data_df, col = "blue", cex=1)
points(data_df[,1], predictions, col = "red", cex=1)
legend("bottomleft", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)
```

In figure 5, the predictions follows the test data well between values from 0 to 10. This makes sense, because the NN learned in question (1) trained values between 0 to 10.

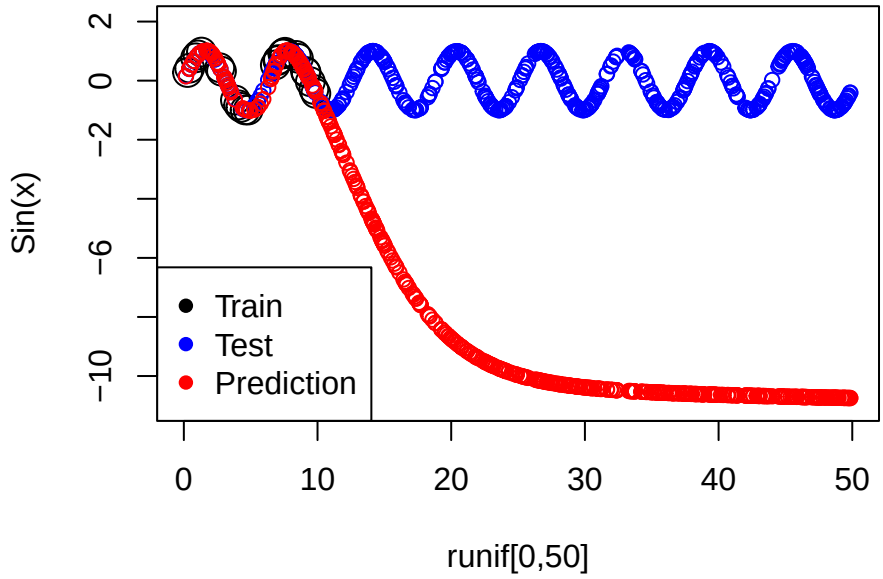


Figure 5: Training data, test data and predictions with runif[0,50]

3.4

In question (3), the predictions seem to converge to some value. Explain why this happens. To answer this question, you may need to get access to the weights of the NN learned. You can do it by running *nn* or *nn*$*weights* where *nn* is the NN learned.

Answer: To understand why the predictions seem to converge to some value we need to look at the formula for the full NN model (Lindholm et al., 2022).

$$\mathbf{q} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^1)$$
$$\hat{y} = \mathbf{W}^{(2)}\mathbf{q} + \mathbf{b}^2$$

- h is the activation function $h(z) = \frac{1}{1+e^{-z}}$
- $\mathbf{W}$ is the weight matrix.
- $\mathbf{x}$ is the input.
- $\mathbf{b}$ is the bias.
- $\hat{y}$ is the prediction.

In the table below we print out the weights and bias from the NN.

```
W1 <- nn_model_assignment_1$weights[[1]][[1]][2,]
b1 <- nn_model_assignment_1$weights[[1]][[1]][1,]

W2 <- nn_model_assignment_1$weights[[1]][[2]][2:11,]
b2 <- nn_model_assignment_1$weights[[1]][[2]][1,]

mat <- matrix("", 10, 4)
colnames(mat) <- c("W1", "W2", "b1", "b2")

mat[1:10,1] <- round(W1,2)
mat[1:10,2] <- round(W2,2)
mat[1:10,3] <- round(b1,2)
mat[1,4] <-round(b2,2)

kable(mat, caption = "The weights and the bias")
```

Table 10: The weights and the bias

W1	W2	b1	b2
4.04	0.77	-11.87	-0.11
-0.54	-1.04	-0.92	
0.26	-16.05	-3.03	
-0.55	-1.27	1.53	
-2.23	3.28	6.56	

W1	W2	b1	b2
1.79	4.31	-11.77	
-1.26	0.12	1.4	
-2.2	-1.22	0.2	
-0.59	-2.76	0.29	
-0.03	2.37	-0.21	

Now we calculate $\mathbf{q}$ by using $\mathbf{W}^{(1)}$, $\mathbf{x}$ (some high number, we use 50) and $\mathbf{b}^1$.

```
# Activation function
logistic <- function(x){
  out <- 1 / (1 + exp(-x))
 return(out)
}

# Input (some high value)
x <- 50
q <- logistic(W1 * x + b1)

kable(q, digits = 0, col.names = "q")
```

q
1
0
1
0
0
1
0
0
0
0

```
# W2 %*% q + b2
```

We know that the logistic function are going to return 1 if the input is large and 0 if the input is small and that is why the $\mathbf{q}$ are going to have the same output if we have a large value for x. This is because all positive weights from $\mathbf{W}^{(1)}$ become large, and negative weights become small as x increases.

Now we can calculate $\hat{y}$, and because there are seven 0:s and three 1:s on index 1,3,5 , we can write:

$$\hat{y} = W_1^2 + W_3^2 + W_5^2 + \mathbf{b}^2 = 0.77 + (-16.05) + 4.31 + (-0.11) = -10.75$$

This tells us that the NN should converge to value -10.75 when input get large which we can see in figure 5.

3.5

Sample 500 points uniformly at random in the interval [0, 10], and apply the sine func-tion to each point. Use all these points as training points for learning a NN that tries to predict x from $sin(x)$, i.e. unlike before when the goal was to predict $sin(x)$ from x. Use the learned NN to predict the training data. You should get bad results. Plot and comment your results. Help: Some people get a convergence error in this question. It can be solved by stopping the training before reaching convergence by setting threshold = 0.1.

```
set.seed(1234567890)
data_runif <- runif(500, min = 0, max = 10)
data_df <- data.frame(runif = data_runif, sin = sin(data_runif))

tr_data <- data_df[1:25,]
te_data <-  data_df[26:500,]

# Random initialization of the weights in the interval [-1, 1], 20 weights + 11 bias
winit <- runif(31, min = -1, max = 1)
```

```
# Neural Network
nn_model <- neuralnet(runif ~ sin, data = tr_data, hidden = c(10), startweights = winit,
                      threshold = 0.1)

predictions <- predict(nn_model, newdata = te_data) %>% as.vector()
```

```
plot(tr_data, cex = 2, ylab = "runif[0,10]", xlab = "Sin(x)", ylim = c(-2, 11))
points(te_data, col = "blue", cex = 1)
points(predictions, te_data[,1], col = "red", cex = 1)
legend("topleft", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)
```

In figure 6, we can see that the predicted values does not fit the a test data at all. This is because the $sin(x)$ function can return same value for different x values and therefore the NN gets confused.

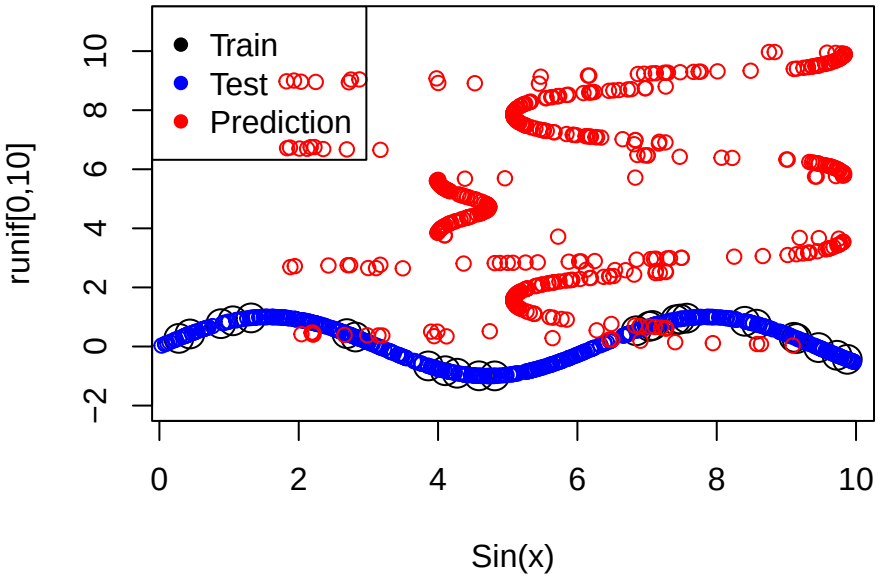


Figure 6: Training data, test data and predictions, x ~ sin(x)

Statement of Contribution

We worked on the assignment individually for the computer labs (to be more efficient when asking questions), William on task 3, Duc on task 2, and Simge on task 1. We later solved all assignment individually and compared and discussed our solutions before dividing the task of writing the laboration report.

Question 1

Text written by Simge.

Question 2

Text written by Duc.

Question 3

Text written by William.

Appendix

The code used in this laboration report are summarised in the code as follows:

```
library(ggplot2)
library(knitr)
library(dplyr)
library(neuralnet)
library(cowplot)
library(geosphere)
#library(kableExtra)
knitr::opts_chunk$set(
  echo = TRUE,
  fig.width = 5,
  fig.height = 4)
set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number") # merged dataframe

# Create a new column with date and time
st$datetime <- as.POSIXct(paste(as.character(st$date), as.character(st$time)),
                          format = "%Y-%m-%d %H:%M:%S", tz = "GMT")

# choose manually a width that gives large kernel values to closer points
# and small values to distant points
h_distance <- 50
```

```r
h_date <- 1
h_time <- 0.4

# Predict for each of these times
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")

gaussian_kernel <- function(diff_vector, l){
  kernel <- exp(-(diff_vector / (2*(l^2))))
  return(kernel)
}
# Point to predict
a <-    55.3836 #latitude
b <- 12.8203     #longitude
date <- "1995-11-03"

i <- 1
target_datetime <- as.POSIXct(paste(date, times[i]), format = "%Y-%m-%d %H:%M:%S", tz = "GMT")
datetime_diff <- target_datetime - st$datetime
st_new <- st[which(datetime_diff > 0), ] # eliminate the posterior

# distance calculation
distance_new <- distHaversine(cbind(st_new$latitude, st_new$longitude), c(a, b))

# date calculation
date1 <- as.Date(date, format = "%Y-%m-%d")
date2 <- as.Date(st_new$date, format = "%Y-%m-%d")

day_of_year1 <- as.numeric(format(date1, "%j"))
day_of_year2 <- as.numeric(format(date2, "%j"))

date_new <- abs(day_of_year2 - day_of_year1)

# hour calculation
st_new$hour <- as.numeric(format(as.POSIXct(st_new$datetime), "%I"))
target_hour <- as.numeric(substr(times[i], 1, 2))
time_new_init <- abs(st_new$hour - target_hour)
time_new <- ifelse(time_new_init < 12, time_new_init, 24-time_new_init) # convert to 12

# kernel calculation
distance_kernel <- gaussian_kernel(distance_new, h_distance)
date_kernel <- gaussian_kernel(date_new, h_date)
time_kernel <- gaussian_kernel(time_new, h_time)

kernel_diff <- c(gaussian_kernel(max(distance_new), h_distance),
                 gaussian_kernel(min(distance_new), h_distance),
                 gaussian_kernel(max(date_new), h_date),
                 gaussian_kernel(min(date_new), h_date),
```

```
                gaussian_kernel(max(distance_new), h_time),
                gaussian_kernel(min(distance_new), h_time))
diff_name <- c("max distance", "min distance",
          "max date", "min date",
          "max time", "min time")

kernel_df <- data.frame("Difference" = kernel_diff)
rownames(kernel_df) <- diff_name
kable(kernel_df, caption = "Kernel Differences")
par(mfrow = c(1, 2))
plot(distance_new, distance_kernel, main = "Distance-Kernel Graph",
     xlab = "Distance Difference", ylab = "Kernel")
grid()

plot(date_new, date_kernel, main = "Date-Kernel Graph",
     xlab = "Date Difference", ylab = "Kernel")
grid()
plot(time_new, time_kernel, main = "Time-Kernel Graph",
     xlab = "Time Difference", ylab = "Kernel")
grid()
summation_kernel_fnc <- function(target_date, a, b){
  temp <- numeric(length=length(times))
  for (i in 1:length(times)){
    target_datetime <- as.POSIXct(paste(target_date, times[i]),
                                  format = "%Y-%m-%d %H:%M:%S", tz = "GMT")
    datetime_diff <- target_datetime - st$datetime
    st_new <- st[which(datetime_diff > 0), ] # eliminate the posterior

    # distance calculation
    distance_new <- distHaversine(cbind(st_new$latitude, st_new$longitude), c(a, b))

    # date calculation
    date1 <- as.Date(target_date, format = "%Y-%m-%d")
    date2 <- as.Date(st_new$date, format = "%Y-%m-%d")

    day_of_year1 <- as.numeric(format(date1, "%j"))
    day_of_year2 <- as.numeric(format(date2, "%j"))

    date_new <- abs(day_of_year2 - day_of_year1)

    # hour calculation
    st_new$hour <- as.numeric(format(as.POSIXct(st_new$datetime), "%I"))
    target_hour <- as.numeric(substr(times[i], 1, 2))
    time_new_init <- abs(st_new$hour - target_hour)
    time_new <- ifelse(time_new_init < 12, time_new_init, 24-time_new_init) # convert to 12

    weight_distance <- gaussian_kernel(distance_new, h_distance)
    weight_date <- gaussian_kernel(date_new, h_date)
    weight_time <- gaussian_kernel(time_new, h_time)
```

```
    w <- weight_distance + weight_date + weight_time
    total_weight <- sum(w)
    weighted_sum <- sum(w * st_new$air_temperature)

    temp[i] <- weighted_sum/total_weight
  }
  return(temp)
}
multiplication_kernel_fnc <- function(target_date, a, b){
  temp <- numeric(length=length(times))
  for (i in 1:length(times)){
    target_datetime <- as.POSIXct(paste(target_date, times[i]),
                                  format = "%Y-%m-%d %H:%M:%S", tz = "GMT")
    datetime_diff <- target_datetime - st$datetime
    st_new <- st[which(datetime_diff > 0), ] # eliminate the posterior

    # distance calculation
    distance_new <- distHaversine(cbind(st_new$latitude, st_new$longitude), c(a, b))

    # date calculation
    date1 <- as.Date(target_date, format = "%Y-%m-%d")
    date2 <- as.Date(st_new$date, format = "%Y-%m-%d")

    day_of_year1 <- as.numeric(format(date1, "%j"))
    day_of_year2 <- as.numeric(format(date2, "%j"))

    date_new <- abs(day_of_year2 - day_of_year1)

    # hour calculation
    st_new$hour <- as.numeric(format(as.POSIXct(st_new$datetime), "%I"))
    target_hour <- as.numeric(substr(times[i], 1, 2))
    time_new_init <- abs(st_new$hour - target_hour)
    time_new <- ifelse(time_new_init < 12, time_new_init, 24-time_new_init) # convert to 12

    weight_distance <- gaussian_kernel(distance_new, h_distance)
    weight_date <- gaussian_kernel(date_new, h_date)
    weight_time <- gaussian_kernel(time_new, h_time)

    w <- weight_distance * weight_date * weight_time
    total_weight <- sum(w)
    weighted_sum <- sum(w * st_new$air_temperature)

    temp[i] <- weighted_sum/total_weight
  }
  return(temp)
}
predict_data <- st[which(st$air_temperature==min(st$air_temperature)),]
target_date <- predict_data$date
```

```
a <- predict_data$latitude
b <- predict_data$longitude
actual_temp <- predict_data$air_temperature
df_forecast <- data.frame("Time" = times,
                          "sum_kernel_forecast" = summation_kernel_fnc(target_date, a, b),
                          "prod_kernel_forecast" = multiplication_kernel_fnc(target_date, a, b))

cat("Actual temperature is:", actual_temp, "at", predict_data$time,
    "in location (", predict_data$latitude, ",", predict_data$longitude, ")\n")
kable(df_forecast, caption = "Forecast Low Temperature")
par(mfrow = c(1, 2))
plot(df_forecast$sum_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Sum kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()

plot(df_forecast$prod_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Production kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()
predict_data <- st[11769,]
target_date <- predict_data$date
a <- predict_data$latitude
b <- predict_data$longitude
actual_temp <- predict_data$air_temperature
df_forecast <- data.frame("Time" = times,
                          "sum_kernel_forecast" = summation_kernel_fnc(target_date, a, b),
                          "prod_kernel_forecast" = multiplication_kernel_fnc(target_date, a, b))

cat("Actual temperature is:", actual_temp, "at", predict_data$time,
    "in location (", predict_data$latitude, ",", predict_data$longitude, ")\n")
kable(df_forecast, caption = "Forecast 1")
par(mfrow = c(1, 2))
plot(df_forecast$sum_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Sum kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()

plot(df_forecast$prod_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Production kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()
predict_data <- st[26872,]
target_date <- predict_data$date
a <- predict_data$latitude
b <- predict_data$longitude
actual_temp <- predict_data$air_temperature
df_forecast <- data.frame("Time" = times,
                          "sum_kernel_forecast" = summation_kernel_fnc(target_date, a, b),
```

```
                          "prod_kernel_forecast" = multiplication_kernel_fnc(target_date, a, b))

cat("Actual temperature is:", actual_temp, "at", predict_data$time,
    "in location (", predict_data$latitude, ",", predict_data$longitude, ")\n")
kable(df_forecast, caption = "Forecast")

par(mfrow = c(1, 2))
plot(df_forecast$sum_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Sum kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()

plot(df_forecast$prod_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Production kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()
predict_data <- st[which(st$air_temperature==max(st$air_temperature)),]
target_date <- predict_data$date
a <- predict_data$latitude
b <- predict_data$longitude
actual_temp <- predict_data$air_temperature
df_forecast <- data.frame("Time" = times,
                          "sum_kernel_forecast" = summation_kernel_fnc(target_date, a, b),
                          "prod_kernel_forecast" = multiplication_kernel_fnc(target_date, a, b))

cat("Actual temperature is:", actual_temp, "at", predict_data$time,
    "in location (", predict_data$latitude, ",", predict_data$longitude, ")\n")
kable(df_forecast, caption = "Forecast High Temperature")

par(mfrow = c(1, 2))
plot(df_forecast$sum_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Sum kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()

plot(df_forecast$prod_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Production kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()
target_date <- "2000-02-06"
a <- 57.7213
b <- 16.4683
actual_temp <- predict_data$air_temperature
df_forecast <- data.frame("Time" = times,
                          "sum_kernel_forecast" = summation_kernel_fnc(target_date, a, b),
                          "prod_kernel_forecast" = multiplication_kernel_fnc(target_date, a, b))

kable(df_forecast, caption = "Forecast Winter")
```

```r
par(mfrow = c(1, 2))
plot(df_forecast$sum_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Sum kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()

plot(df_forecast$prod_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Production kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()
target_date <- "2000-07-06"
a <- 57.7213
b <- 16.4683
actual_temp <- predict_data$air_temperature
df_forecast <- data.frame("Time" = times,
                          "sum_kernel_forecast" = summation_kernel_fnc(target_date, a, b),
                          "prod_kernel_forecast" = multiplication_kernel_fnc(target_date, a, b))

kable(df_forecast, caption = "Forecast Summer")

par(mfrow = c(1, 2))
plot(df_forecast$sum_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Sum kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()

plot(df_forecast$prod_kernel_forecast, type="o", xaxt="n", ylab = "Temperature",
     xlab = " ", main = "Production kernel")
axis(side = 1, at = 1:length(times), labels = times, las = 2)
grid()
# All the following code is supplied by the task.
#-----------------------------------------------------------------#
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, ]


# Finds the best value of C by using validation data
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)

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=FAL
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=FAL
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
#-------------------------------------------------------------#
filter_0 <- c("training", "validation", err0)
filter_1 <- c("training", "test", err1)
filter_2 <- c("train+val","test", err2)
filter_3 <- c("all data", "test", err3)
table_data <- data.frame(rbind(filter_0, filter_1, filter_2, filter_3))
colnames(table_data) <- c("Training data", "Testing data", "Error on testing data")
rownames(table_data) <- c("Filter 0", "Filter 1", "Filter 2", "Filter 3")
table_data$`Error on testing data` <- round(as.numeric(table_data$`Error on testing data`), 4)
kable(table_data, caption = "Summarized information about filters")
# Support vectors
sv<-alphaindex(filter3)[[1]]
# Coefficients for the support vectors
co<-coef(filter3)[[1]]
# negative intercept
intercept <- - b(filter3)

# Support vector observations with response variable removed
x <- spam[sv, -58] # Remove y value.

# The 10 observations we need to predict
x_stars <- spam[1:10, -58]

# The Gaussian RBF kernel function
```

```r
rbfkernel <- rbfdot(sigma = 0.05)

k<-c()
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[j] <- co[j]*rbfkernel(unlist(x[j,]), unlist(x_stars[i,]))
  }
  prediction <- intercept + sum(k2)
  prediction
  k[i]<- prediction
}
manual_pred <- k
function_pred <- predict(filter3,spam[1:10,-58], type = "decision")

table_data <- data.frame(manual_pred, function_pred)
colnames(table_data) <- c("Manual prediction", "Predict function")
kable(table_data, caption = "Comparision between manual prediction and the predict function for the firs

# The data
set.seed(1234567890)
data_runif <- runif(500, min = 0, max = 10)
data_df <- data.frame(runif = data_runif, sin = sin(data_runif))

tr_data <- data_df[1:25,]
te_data <-  data_df[26:500,]

# Random initialization of the weights in the interval [-1, 1], 20 weights + 11 bias = 31
winit <- runif(31, min = -1, max = 1)




# Neural Network
nn_model_assignment_1 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10), startweights = winit)

predictions <- predict(nn_model_assignment_1, newdata = te_data) %>% as.vector()


plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex = 1)
points(te_data[,1], predictions, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)



# Three custom activation function
```

```r
h1 <- function(x) x
h2 <- function(x) ifelse(x > 0, x, 0) # same as max(0,x)
h3 <- function(x) log(1 + exp(x))


# Neural Network
nn_model_h1 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10),
                         act.fct = h1, startweights = winit)
nn_model_h2 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10),
                         act.fct = h2, startweights = winit)
nn_model_h3 <- neuralnet(sin ~ runif, data = tr_data, hidden = c(10),
                         act.fct = h3, startweights = winit)

predictions_h1 <- predict(nn_model_h1, newdata = te_data) %>% as.vector()
predictions_h2 <- predict(nn_model_h2, newdata = te_data) %>% as.vector()
predictions_h3 <- predict(nn_model_h3, newdata = te_data) %>% as.vector()




plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex=1)
points(te_data[,1], predictions_h1, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)



plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex = 1)
points(te_data[,1], predictions_h2, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)




plot(tr_data, cex = 2, xlab = "runif[0,10]", ylab = "Sin(x)")
points(te_data, col = "blue", cex=1)
points(te_data[,1], predictions_h3, col = "red", cex = 1)
legend("bottomright", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)




# The data
set.seed(1234567890)
data_runif <- runif(500, min = 0, max = 50)
```

```r
data_df <- data.frame(runif = data_runif, sin =  sin(data_runif))




predictions <- predict(nn_model_assignment_1, newdata = data_df) %>% as.vector()



plot(tr_data, cex = 2, xlab = "runif[0,50]", ylab = "Sin(x)", ylim = c(-11, 2), xlim = c(0,50))
points(data_df, col = "blue", cex=1)
points(data_df[,1], predictions, col = "red", cex=1)
legend("bottomleft", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)


W1 <- nn_model_assignment_1$weights[[1]][[1]][2,]
b1 <- nn_model_assignment_1$weights[[1]][[1]][1,]

W2 <- nn_model_assignment_1$weights[[1]][[2]][2:11,]
b2 <- nn_model_assignment_1$weights[[1]][[2]][1,]

mat <- matrix("", 10, 4)
colnames(mat) <- c("W1", "W2", "b1", "b2")

mat[1:10,1] <- round(W1,2)
mat[1:10,2] <- round(W2,2)
mat[1:10,3] <- round(b1,2)
mat[1,4] <-round(b2,2)

kable(mat, caption = "The weights and the bias")


# Activation function
logistic <- function(x){
  out <- 1 / (1 + exp(-x))
 return(out)
}

# Input (some high value)
x <- 50
q <- logistic(W1 * x + b1)

kable(q, digits = 0, col.names = "q")

# W2 %*% q + b2
```

```r
set.seed(1234567890)
data_runif <- runif(500, min = 0, max = 10)
data_df <- data.frame(runif = data_runif, sin = sin(data_runif))

tr_data <- data_df[1:25,]
te_data <-  data_df[26:500,]

# Random initialization of the weights in the interval [-1, 1], 20 weights + 11 bias
winit <- runif(31, min = -1, max = 1)



# Neural Network
nn_model <- neuralnet(runif ~ sin, data = tr_data, hidden = c(10), startweights = winit,
                      threshold = 0.1)

predictions <- predict(nn_model, newdata = te_data) %>% as.vector()


plot(tr_data, cex = 2, ylab = "runif[0,10]", xlab = "Sin(x)", ylim = c(-2, 11))
points(te_data, col = "blue", cex = 1)
points(predictions, te_data[,1], col = "red", cex = 1)
legend("topleft", legend = c("Train", "Test", "Prediction"),
col = c("black", "blue", "red"), pch = 16)
```