Machine Learning - Lab 3 Block 1 - Group A9

Jaskirat S Marar (jasma356), Daniel Persson (danpe586),
Hussnain Khalid (huskh803)

12/14/2021

Statement of Contribution

For solving this lab, the group decided to split the responsibility equally by assigning 1 question to each member. The split by mutual consensus was as follows:

1. Assignment1: Solution and report by Daniel Persson
2. Assignment2: Solution and report by Jaskirat Marar
3. Assignment3: Solution and report by Hussnain Khalid

We were able to communicate with each other effectively and responsibly. All the group members were forthcoming in discussing issues being faced while solving the problems. We were able to each present our solution to the others well before the deadline and were able to conclude on the structure and content of the final report.

"*We acknowledge that each member has contributed fairly and equally in solving this lab.*"

By undersigned:

Daniel Persson

Jaskirat Marar

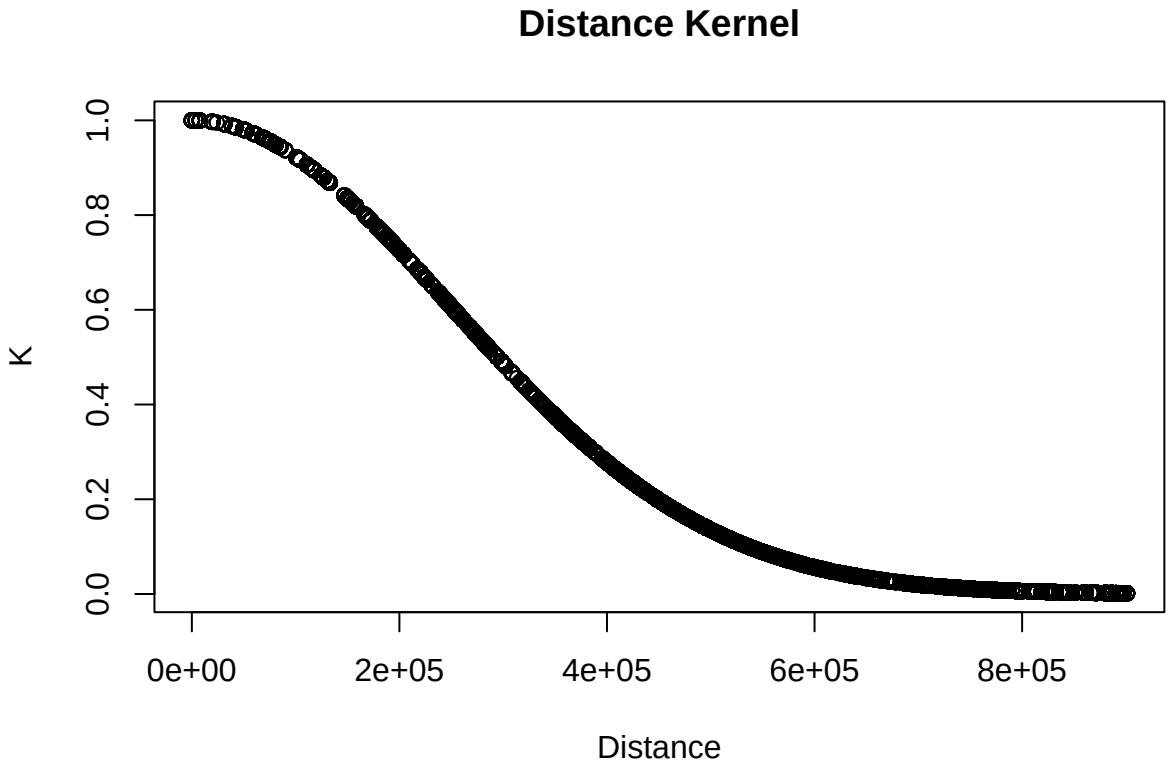
Hussnain Khalid

Assignment 1 Kernel Methods

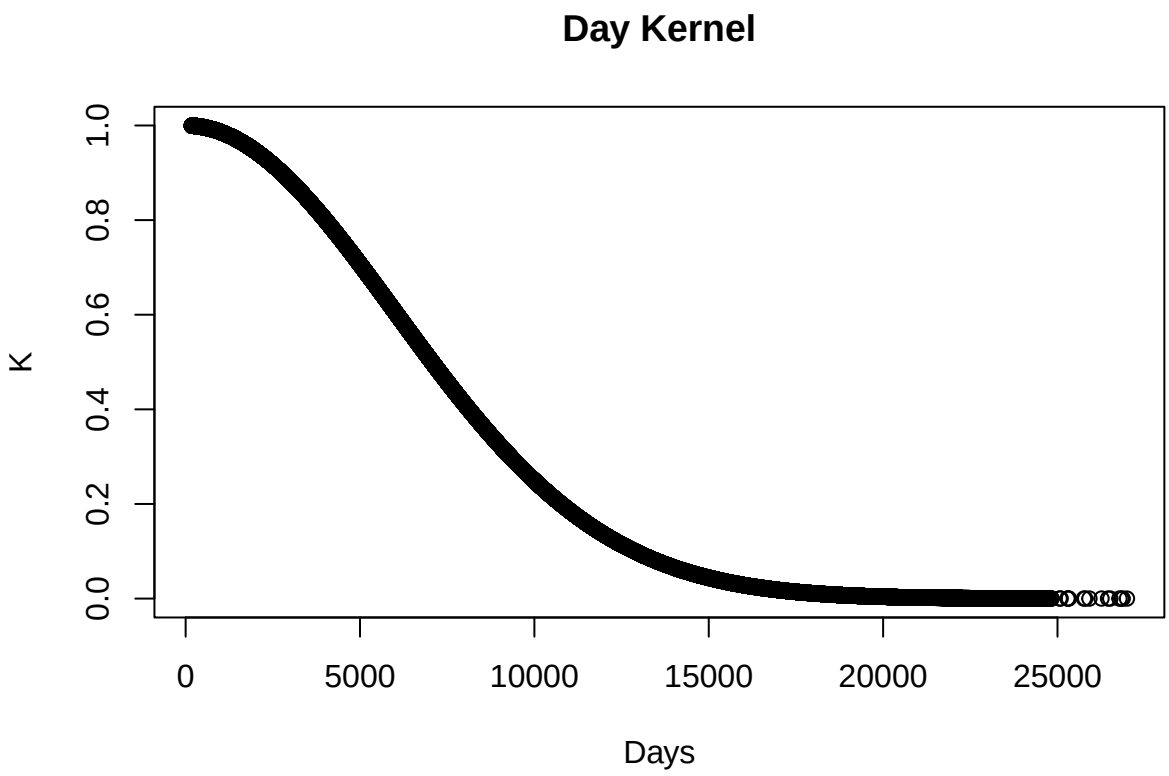
The Kernels

For the kernel methods three different kernels have been used, then the sum and the product of those has been used as final kernels. The underlying kernels are the distance, day and hour, which are plotted bellow. The smoother coefficient has been manually decided after interpreting the graph for each kernel. Finally the combined kernels "Addition" and "Multiplied" are used and plotted for the train and test data.

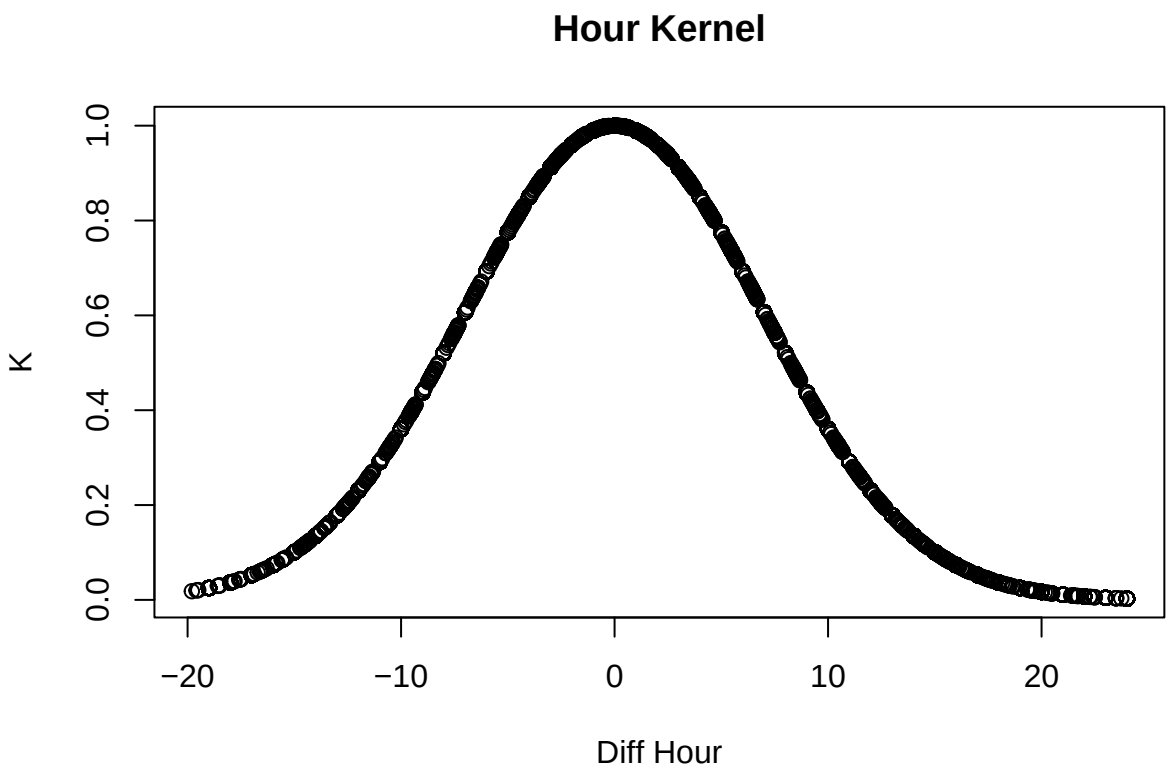
Smoother coefficient h_distance = 250,000 gives the following graph:



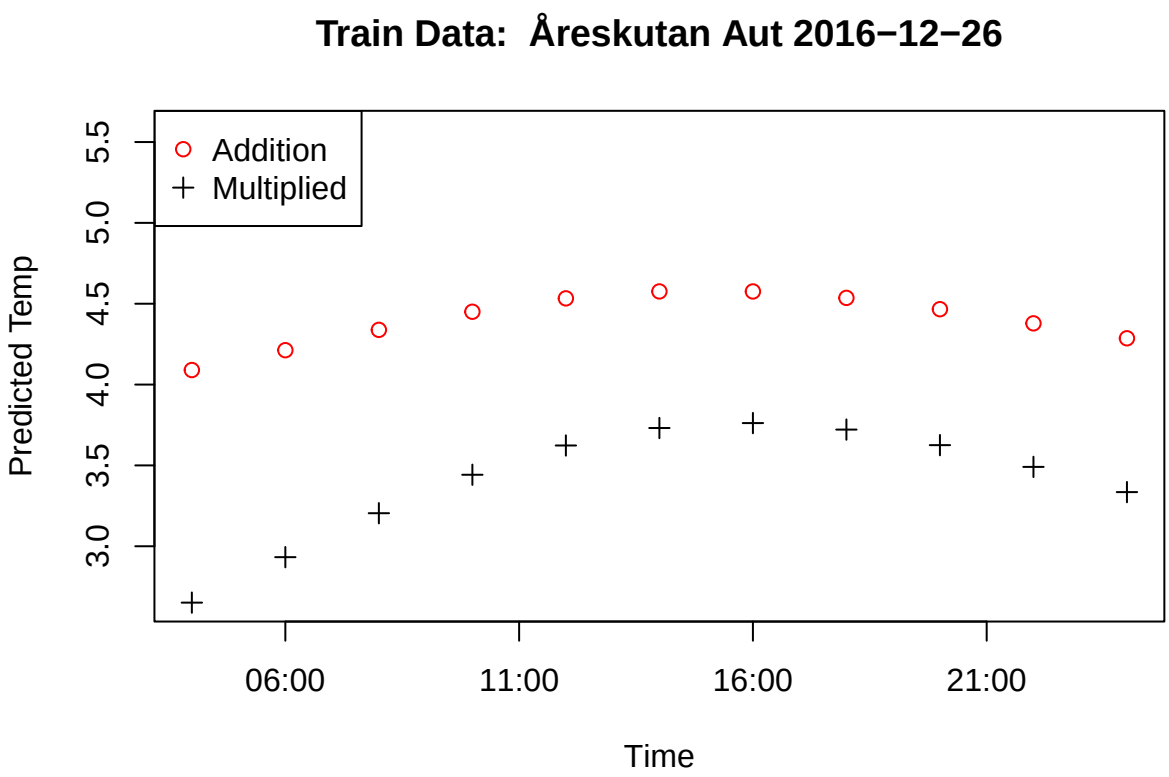
Smoother coefficient h_day = 6000 gives the following graph:

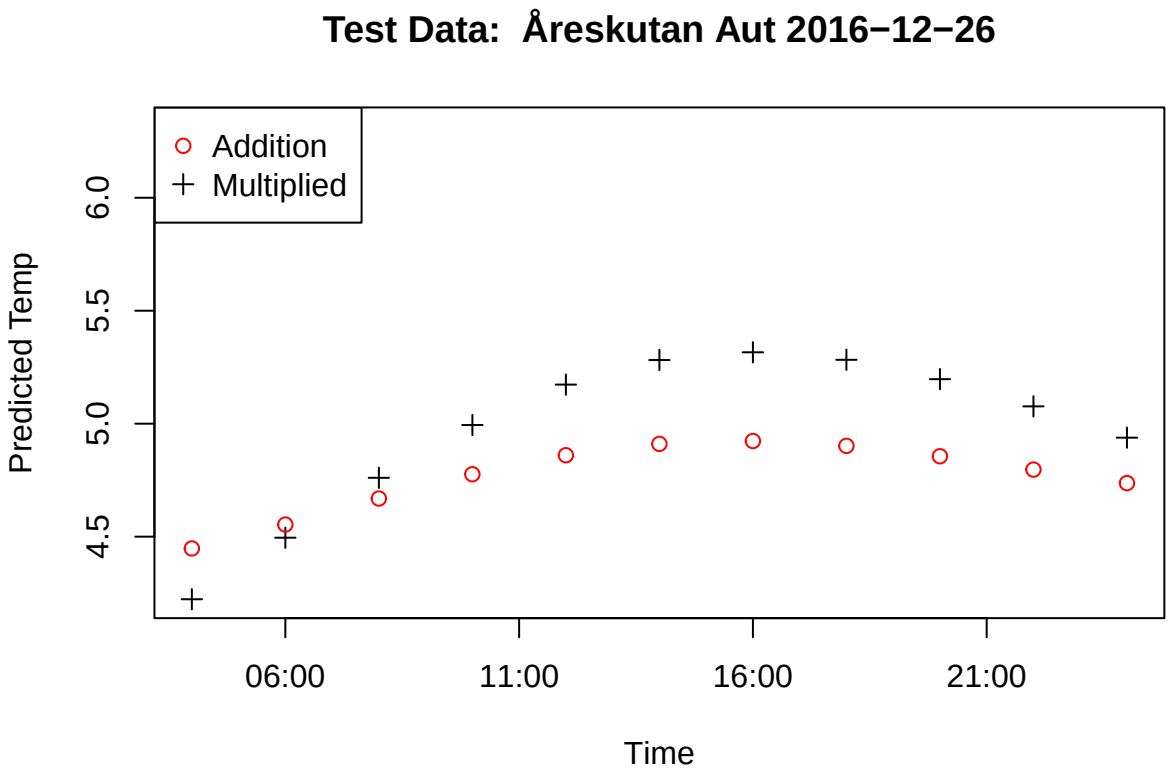


Smoother coefficient h_hour = 7 result in the following graph:



Results


Train Data: Åreskutan Aut 2016−12−26
Addition
Multiplied
Predicted Temp
3.0
3.5
4.0
4.5
5.0
5.5
06:00
11:00
16:00
21:00
Time



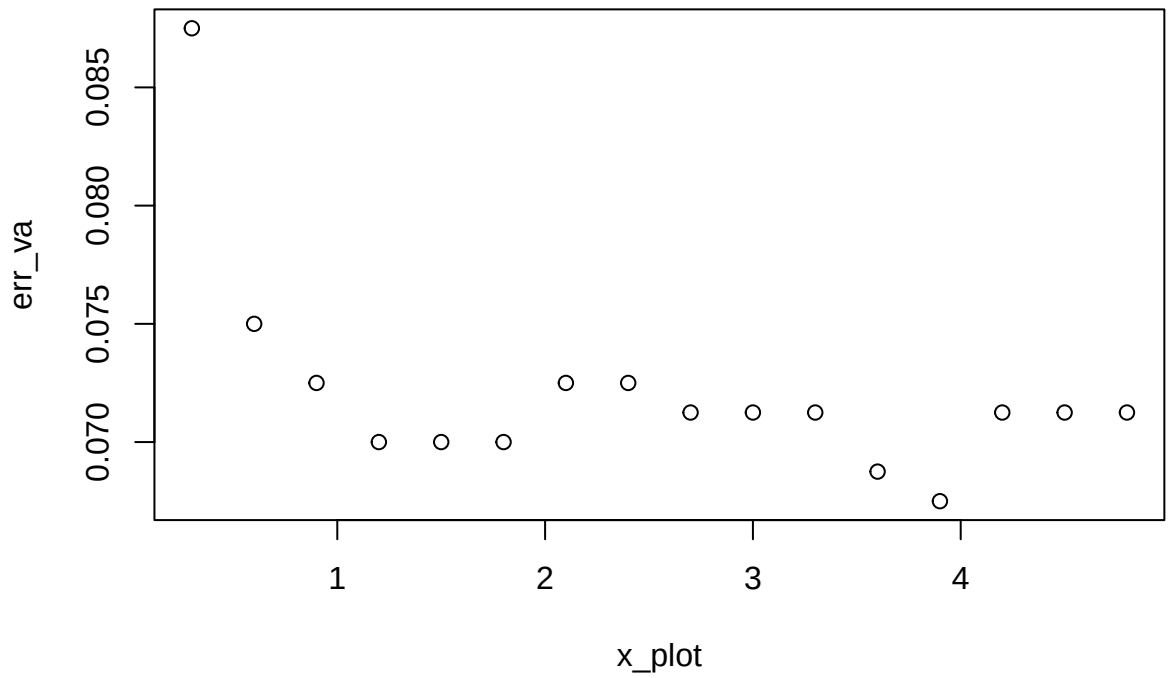
Comparison of Kernels

By comparing the "Addition" (red) and "Multiplied" (black) kernels in the plot above, it can be seen that they have different values. Where the latter has a higher difference between the min and max value since the values are small (< 1) the sum will get larger than the product and the denominator will work as a smoother. Also the prediction doesn't seem very accurate as it probably should be minus degrees since it is in the north of Sweden around Christmas. An issue is that it is not taken into account what day of the year it is, if it is summer or winter, only the number of days from the predicted date is taken into account. Another issue is that the actual prediction place (Areskutan) is 1280 meters above the sea level, which is not taken into account, since there are no kernel for the elevation implemented.

Assignment 2 SVM

The data is first prepped and split into training, test and validation

The following bit of code builds various SVMs with a $\sigma = 0.05$ and multiple values of C. A lower value of σ will result in a narrow decision and higher variance, whereas a larger value would have a larger decision surface and lower variance. The data is not scaled here.



As we can see from the plot that the lowest value of the error from the validation data lies at $C = 3.9$. We will use this now in the making a choice from the following SVMs

filter0 & filter1

Both of these SVMs are learned on the training data, but filter0 predicts validation data whereas, filter1 jumps straight to the test data. Between these two implementations, filter0 is correct as scoring on validation data gives the opportunity to fine tune parameters if needed and also returns a better accuracy of 93.25% as compared to filter1.

```
## [1] 93.25
```

```
## [1] 91.51061
```

filter2

This SVM is trained on both the training and validation data together. While it is better to use validation data separately for fine tuning, this approach is essentially not wrong as we are still testing unseen data. But we can see that accuracy is lower in addition to the previous point. Hence this filter is not desirable.

```
## [1] 91.7603
```

filter3

This SVM trains on the complete spam data which is the wrong practice. Hence any further error scores become irrelevant as the predicted values are on seen data and not unseen data.

```
## [1] 97.87765
```

Questions

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

We know our choice is essentially between filter0 or 1 since, they are the same, but in terms of the over correctness, we would choose filter0 to return to the user due to the reasons mentioned earlier. For adding an additional layer we also calculate the sensitivity and specificity of each SVM and show them in the following table:

```
##   Filter  Accuracy Sensitivity Specificity
## 1      0 0.9325000   0.9540000   0.8966667
## 2      1 0.9151061   0.9525862   0.8635015
## 3      2 0.9176030   0.9547414   0.8664688
## 4      3 0.9787765   0.9935345   0.9584570
```

As we can see from the above info that filter0 also results in better TP rate as well as TN rate vs filter1. Hence we choose to return filter0.

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

The estimate of the generalization error of filter0 is equal to err1. This is because computationally, err1 is 1-accuracy i.e. total incorrect predictions vs all predictions, on the TEST data of filter0 or filter1 as they are built with the same parameters. Generalization error is the expectation of the the predicted values of an independent test set on our model which in this case of filter0 is the mean of all the predicted values not matching the actual test data classificiation. We show this as follows:

```
filter0_predict <- predict(filter0, te, type = "response")
filter0_generr <- mean(filter0_predict != te[, 58])
all.equal(filter0_generr, err1)
```

```
## [1] TRUE
```

3. Implementation of SVM predictions.

filter3 predictions

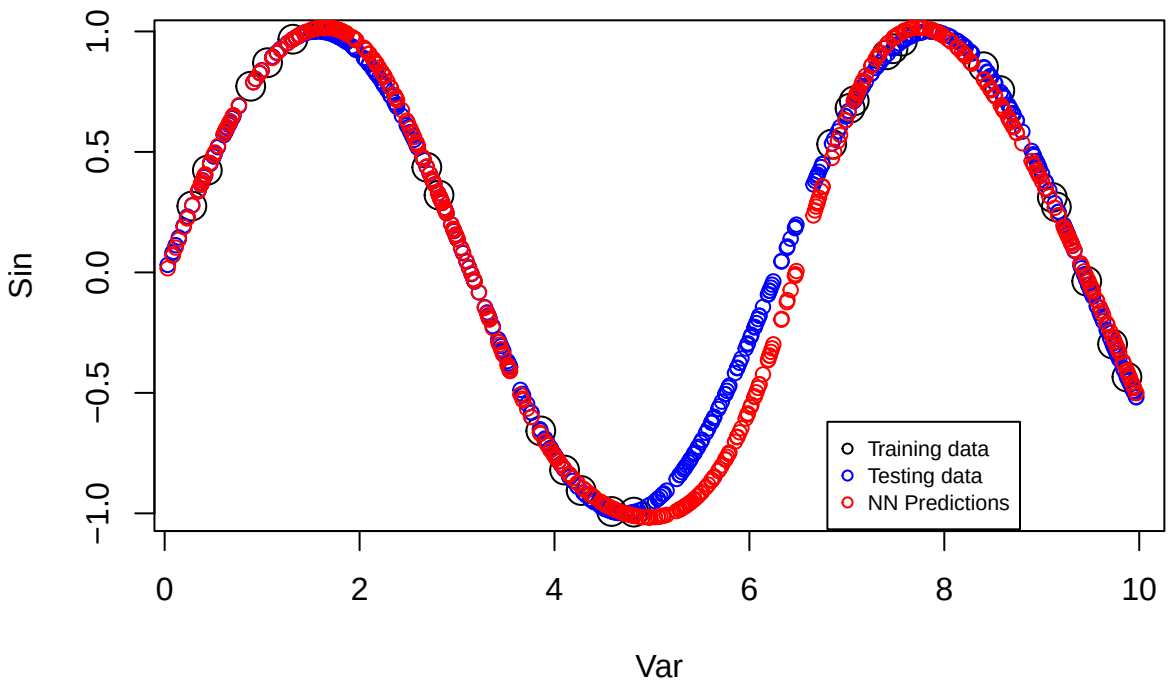
We use the given code to compute the index of the Support vectors, coefficients and the intercept of the SVM using filter3. A kernel generating function is setup using the $\sigma = 0.05$ which has been used throughout this problem. Then we initialize a vector to stoke the predictions and start a nested loop to calculate the dot product of each x_i_ and corresponding coefficient for each predictor variable. each prediction is appended to the resulting vector. Final computations are verified by adding the result from predict() in a data frame.

```
##            k   predict
## 1  -1.998999 -1.998999
## 2   1.560584  1.560584
## 3   1.000278  1.000278
## 4  -1.756815 -1.756815
## 5  -2.669577 -2.669577
## 6   1.291312  1.291312
## 7  -1.068444 -1.068444
## 8  -1.312493 -1.312493
```

```
## 9   1.000184  1.000184
## 10 -2.208639 -2.208639
```

Assignment 3: Neural Networks

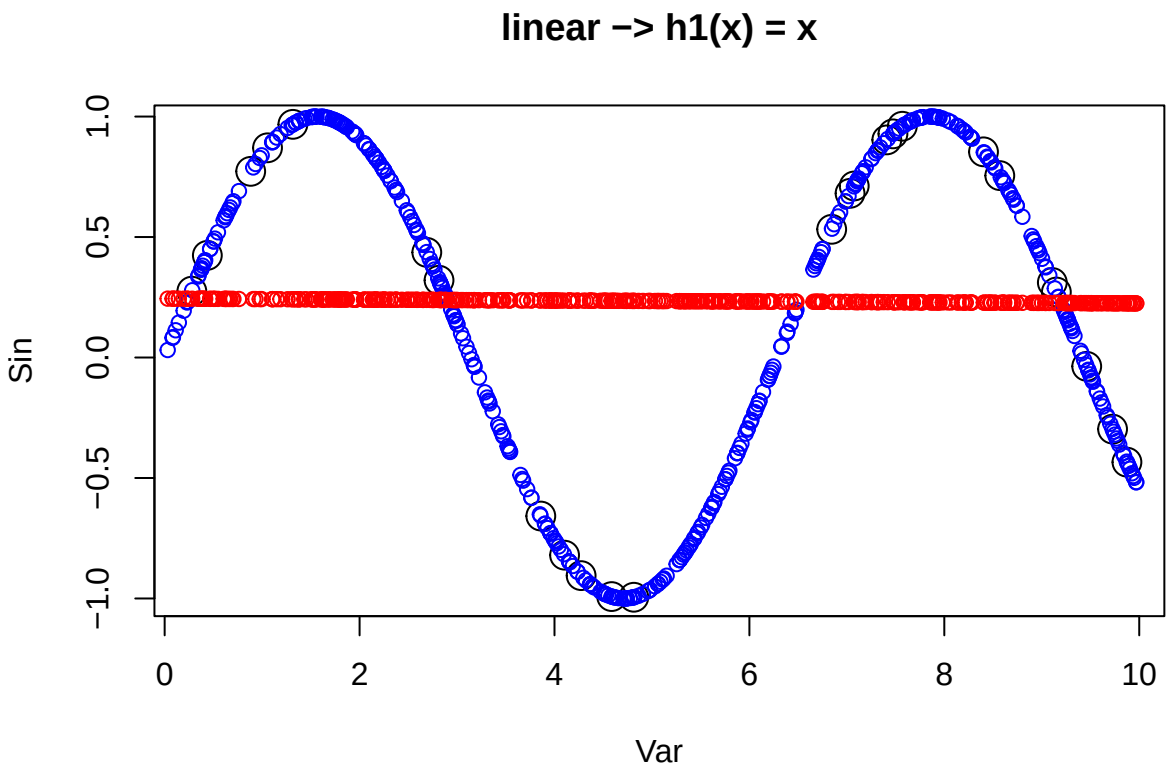
3.1:



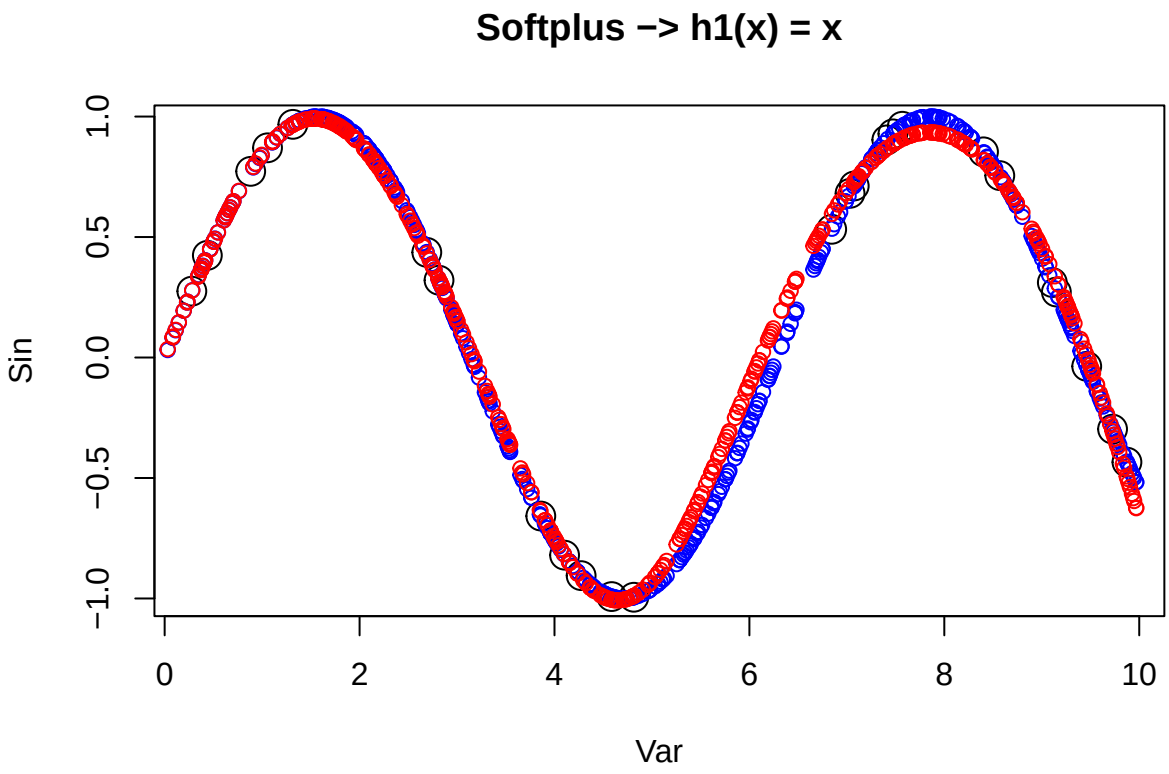
By looking at the plot we can say that our model predict sin(Var) very well. Most of the points are directly on the desired value and remaining are very close. Therefore, we can say that our neural network is working fine to compute the sine function between 0 and 10.

3.2:

```
# Custom activation functions:
h1 <- function(x) {
    x
}  #Linear
h2 <- function(x) max(0, x)  #ReLU
h3 <- function(x) {
    log(1 + exp(x))
}  #Softplus
```

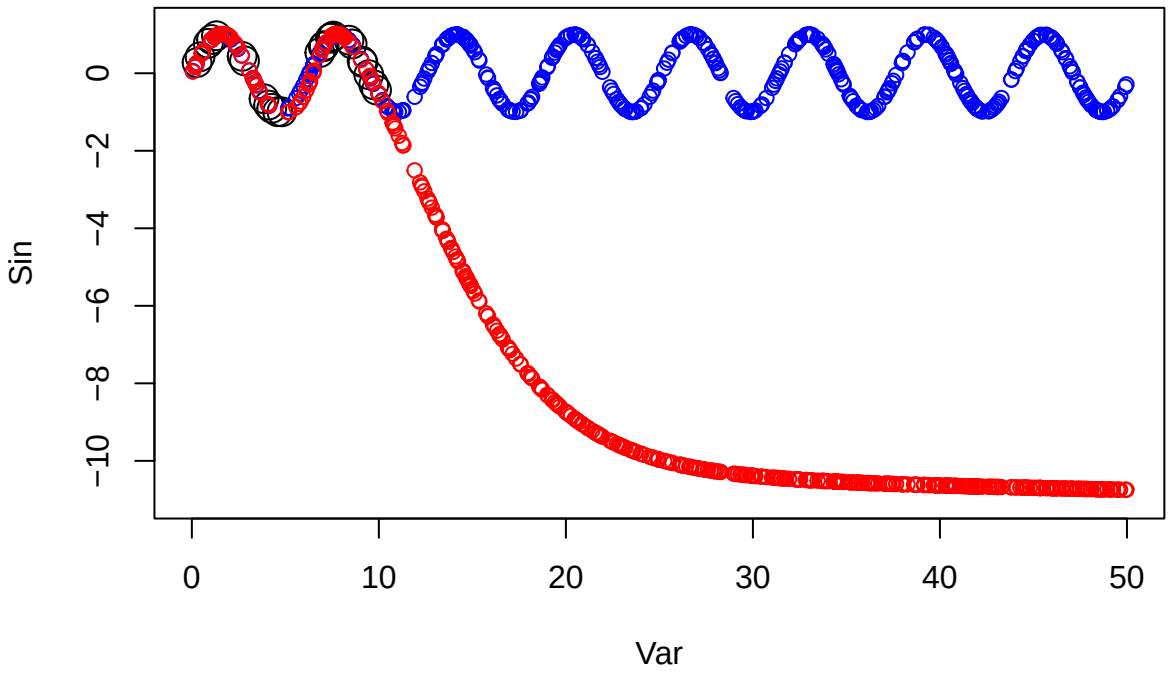



```
## Error in Deriv_(fd, x, env, use.D, dsym, scache, combine, drule. = drule): Could not retrieve body o
```



By changing the activation function we can see how our neural network is working. When activation function h1(x) "linear" is set we can see our model network is not predicting sin(Var) at all as it is trying to plot linear relation and sin function is also nonlinear function. And with activation function h2(x) "ReLU" model is giving us error because it is not differentiable at constant values (0 is case here). Though the best result is achieved with activation function h3(x) "softplus" model which is also smooth approximation of the "ReLu" function.

3.3



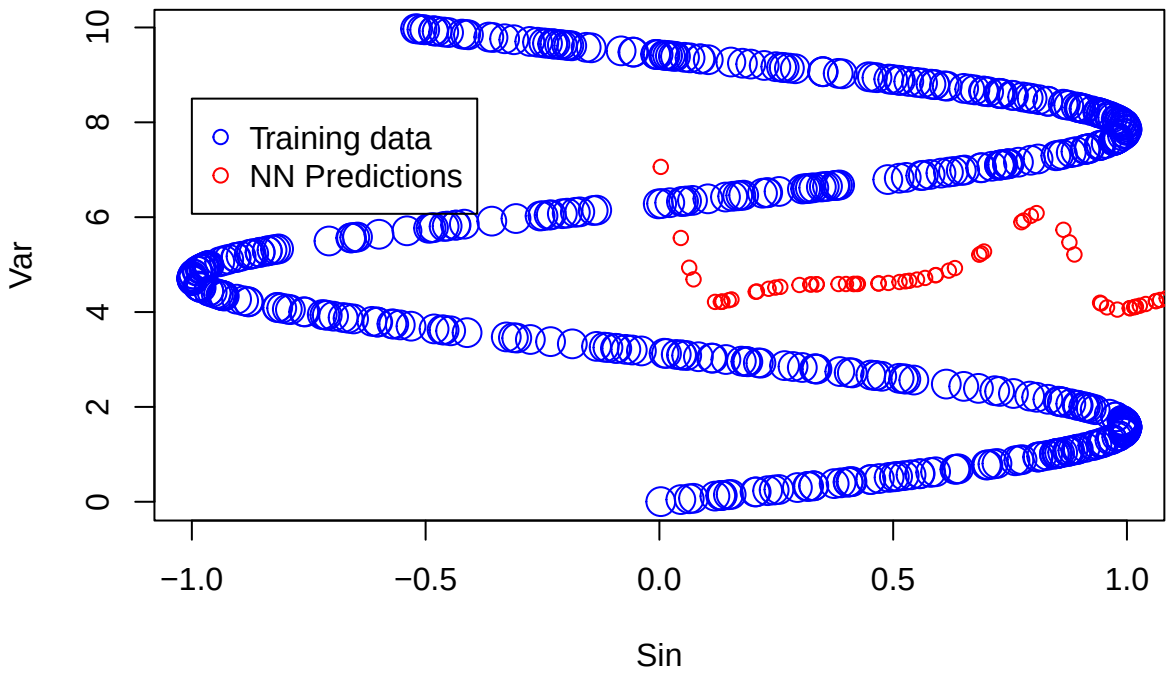
We can see model is generating good results in the range [0,10] because it is trained only within this range. So when x > 10 we are getting bad results from it, as we can see exponentially decreasing line after this range because our model is trying to follow the default sigmoid activation function.

3.4

```
##        Weights        Bias
## 1   4.04249377 -11.8708307
## 2  -0.53686766  -0.9153178
## 3   0.26033546  -3.0316259
## 4  -0.54876561   1.5313105
## 5  -2.23383933   6.5554366
## 6   1.78786162 -11.7685251
## 7  -1.25983363   1.3973249
## 8  -2.19740359   0.2032402
## 9  -0.58860049   0.2948804
## 10 -0.03177278  -0.2132649
```

We can see the with increasing input value the activation function is approaching to 0 by using the default activation sigmoid function {h = sigma(weight * inputs + bias)} but after the weight of 6th neuron it is approaching to 1. Which means that after 6th neuron the hidden layer got activated and our predictions are converging.

3.5



The results we are getting are bad as we are using Sin(x) as independent variable and x is dependent(target variable) on sin function. The sin function is not like having a unique value for every unique value, for example sin(0), sin(180) and sin(360) have same value 0 so model don't know 0 belong to which sin(x).

Appendix

```
knitr::opts_chunk$set(echo = TRUE, tidy.opts = list(width.cutoff = 60),
    tidy = TRUE)
# loading packages
library(geosphere)
library(neuralnet)
library(kernlab)
# Assignment1: Kernel Methods

# load the data
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")

# merge stations and temps data in regards to station
# number
st <- merge(stations, temps, by = "station_number")
# initial values / what to predict

# take out station row and name, e.g. 'Åreskutan Aut',
# 'Stockholm-Arlanda', 'Kiruna'
NA
station <- stations$station_name[station_row]  # used for label in plot

# set coordinates to predict
a <- stations$longitude[station_row]
b <- stations$latitude[station_row]
# add time sequence from 4 am to 24 pm
time <- seq(4, 24, by = 2)
times <- c(paste0("0", time[1:3], ":", "00", ":", "00"), paste0(time[4:length(time)],
    ":", "00", ":", "00"))
# add date to predict
date <- as.POSIXlt("2016-12-26")

# filter out date to predict and dates after that
if (any(as.POSIXlt(st$date) >= date)) {
    st <- st[-which(as.POSIXlt(st$date) >= date), ]
}

# split and partition the data train/test (70/30)%
n <- dim(st)[1]
set.seed(1234567890)
id <- sample(1:n, floor(n * 0.7))
train <- st[id, ]
test <- st[-id, ]
# setting up the Gaussian kernels which is a sum of the 3
# features

# Gaussian kernel distance
k_distance_f <- function(data, a, b, h_dist) {
    # take out the coordinates
    coords <- data.frame(long = (data$longitude), lat = (data$latitude))
    # distance in meters
    distance <- distHaversine(c(a, b), coords)
```

```
    # the kernel
    result <- exp(-((distance)^2)/(2 * h_dist^2))
    return(c(distance, result))
}

N <- nrow(train)
i <- 1:N
h_dist <- 250000  # chosen manually by interpreting the plot
k_distance <- sapply(i, function(i) k_distance_f(train[i, ],
    a, b, h_dist))
plot(k_distance[1, ], k_distance[2, ], main = "Distance Kernel",
    xlab = "Distance", ylab = "K")
# Gaussian kernel day
k_day_f <- function(data, date, h_day) {
    # take out the difference in days
    day <- as.numeric(difftime(date, data, units = "days"))
    # the kernel
    result <- exp(-((day)^2)/(2 * h_day^2))
    return(c(day, result))
}
h_day <- 6000  # chosen manually by interpreting the plot
k_day <- sapply(i, function(i) k_day_f(as.POSIXlt(train$date[i]),
    date, h_day))
plot(k_day[1, ], k_day[2, ], main = "Day Kernel", xlab = "Days",
    ylab = "K")
# Gaussian kernel hour
h_hour <- 7  # chosen manually by interpreting the plot
k_hour_f <- function(data, times, h_hour) {
    # calculate the difftime
    hour <- as.numeric(difftime(strptime(times, "%H:%M:%S"),
        strptime(data$time, "%H:%M:%S"), units = "hours"))
    # the kernel
    result <- exp(-((hour)^2)/(2 * h_hour^2))
    return(c(hour, result))
}
k_hour <- sapply(i, function(i) k_hour_f(train[i, ], times, h_hour))
plot(k_hour[1:11, ], k_hour[12:22, ], main = "Hour Kernel", xlab = "Diff Hour",
    ylab = "K")
# setting up the addition kernel which is the *sum* of the
# 3 kernels distance, day & hour
k_sum <- matrix(NA, nrow = 11, ncol = N)
temp_pred_sum <- matrix(NA, nrow = 11, ncol = 1)
k <- 1
for (x in 12:22) {
    # sum the kernels
    k_sum[k, ] <- (k_distance[2, ] + k_day[2, ] + k_hour[x, ])
    temp_pred_sum[k] <- (k_sum[k, ] %*% train$air_temperature)/sum(k_sum[k,
        ])
    k <- k + 1
}
```

```
# setting up the Gaussian kernels which is the
# *multiplying* of the 3
k_mult <- matrix(NA, nrow = 11, ncol = N)
temp_pred_mult <- matrix(NA, nrow = 11, ncol = 1)
k <- 1
for (x in 12:22) {
    k_mult[k, ] <- k_distance[2, ] * k_day[2, ] * k_hour[x, ]
    temp_pred_mult[k] <- (k_mult[k, ] %*% train$air_temperature)/sum(k_mult[k,
        ])
    k <- k + 1
}
# combined plot for addition and multiplied kernel
plot(strptime(times, format = "%H:%M:%S"), temp_pred_mult, ylab = "Predicted Temp",
    xlab = "Time", main = paste("Train Data: ", station, date),
    ylim = c(min(temp_pred_mult, temp_pred_sum), 1 + max(temp_pred_mult,
        temp_pred_sum)), pch = 3)
points(strptime(times, format = "%H:%M:%S"), temp_pred_sum, col = "red")
legend(x = "topleft", legend = c("Addition", "Multiplied"), col = c("red",
    "black"), pch = c(1, 3))
N <- nrow(test)
i <- 1:N
k_distance <- sapply(i, function(i) k_distance_f(train[i, ],
    a, b, h_dist))
k_day <- sapply(i, function(i) k_day_f(as.POSIXlt(test$date[i]),
    date, h_day))
k_hour <- sapply(i, function(i) k_hour_f(test[i, ], times, h_hour))

k_sum <- matrix(NA, nrow = 11, ncol = N)
temp_pred_sum <- matrix(NA, nrow = 11, ncol = 1)
k <- 1
for (x in 12:22) {
    # sum the kernels
    k_sum[k, ] <- (k_distance[2, ] + k_day[2, ] + k_hour[x, ])
    temp_pred_sum[k] <- (k_sum[k, ] %*% test$air_temperature)/sum(k_sum[k,
        ])
    k <- k + 1
}
k_mult <- matrix(NA, nrow = 11, ncol = N)
temp_pred_mult <- matrix(NA, nrow = 11, ncol = 1)
k <- 1
for (x in 12:22) {
    k_mult[k, ] <- k_distance[2, ] * k_day[2, ] * k_hour[x, ]
    temp_pred_mult[k] <- (k_mult[k, ] %*% test$air_temperature)/sum(k_mult[k,
        ])
    k <- k + 1
}
# combined plot for addition and multiplied kernel
plot(strptime(times, format = "%H:%M:%S"), temp_pred_mult, ylab = "Predicted Temp",
    xlab = "Time", main = paste("Test Data: ", station, date),
    ylim = c(min(temp_pred_mult, temp_pred_sum), 1 + max(temp_pred_mult,
        temp_pred_sum)), pch = 3)
points(strptime(times, format = "%H:%M:%S"), temp_pred_sum, col = "red")
legend(x = "topleft", legend = c("Addition", "Multiplied"), col = c("red",
```

```
    "black"), pch = c(1, 3))


# Assignment2: SVMs

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, ]
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))
}

x_plot <- seq(by, 5, by)
plot(x = x_plot, y = err_va, type = "p")

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)
(1 - err0) * 100
sen0 <- t[1, 1]/sum(t[, 1])  #TP/(TP+FN)
spec0 <- t[2, 2]/sum(t[, 2])  #TN/(TN+FP)


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)
(1 - err1) * 100
sen1 <- t[1, 1]/sum(t[, 1])  #TP/(TP+FN)
spec1 <- t[2, 2]/sum(t[, 2])  #TN/(TN+FP)


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

```
(1 - err2) * 100
sen2 <- t[1, 1]/sum(t[, 1])  #TP/(TP+FN)
spec2 <- t[2, 2]/sum(t[, 2])  #TN/(TN+FP)

filter3 <- ksvm(type ~ ., data = spam, kernel = "rbfdot", kpar = list(sigma = 0.05),
    C = which.min(err_va) * by, scaled = FALSE)
mailtype <- predict(filter3, te[, -58])
t <- table(mailtype, te[, 58])
err3 <- (t[1, 2] + t[2, 1])/sum(t)
(1 - err3) * 100
sen3 <- t[1, 1]/sum(t[, 1])  #TP/(TP+FN)
spec3 <- t[2, 2]/sum(t[, 2])  #TN/(TN+FP)


filter_select <- data.frame(Filter = c(0:3), Accuracy = c(1 -
    err0, 1 - err1, 1 - err2, 1 - err3), Sensitivity = c(sen0,
    sen1, sen2, sen3), Specificity = c(spec0, spec1, spec2, spec3))

filter_select
filter0_predict <- predict(filter0, te, type = "response")
filter0_generr <- mean(filter0_predict != te[, 58])
all.equal(filter0_generr, err1)


sv <- alphaindex(filter3)[[1]]  # index of Support Vectors
# sv
co <- coef(filter3)[[1]]  # linear Coefficients of SVs
# co
inte <- -b(filter3)  # intercept
rbf <- rbfdot(sigma = 0.05)  # kernel generating function
k <- c()
for (i in 1:10) {
    # We produce predictions for just the first 10 points
    # in the dataset.
    k2 <- 0
    for (j in 1:length(sv)) {
        k2 <- k2 + co[j] * rbf(as.numeric(spam[sv[j], -58]),
            as.numeric(spam[i, -58]))
        # prediction without intercept = coefficient * dot
        # prod<SV, x_i>
    }
    k <- c(k, k2 + inte)  #add intercept to prediction
}

result <- data.frame(k = k, predict = predict(filter3, spam[1:10,
    -58], type = "decision"))

result


# Assignment3: NN
```

```
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

# Random initialization of the weights in the interval [-1,
# 1]
winit <- runif(31, -1, 1)
nn <- neuralnet(formula = 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)
legend(x = 6.8, y = -0.62, legend = c("Training data", "Testing data",
    "NN Predictions"), col = c("black", "blue", "red"), cex = 0.7,
    pch = 1)
# Custom activation functions:
h1 <- function(x) {
    x
}  #Linear
h2 <- function(x) max(0, x)  #ReLU
h3 <- function(x) {
    log(1 + exp(x))
}  #Softplus
# NN for h1(x)
nn2_a <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10,
    startweights = winit, act.fct = h1)
plot(tr, cex = 2, main = "linear -> h1(x) = x")
points(te, col = "blue", cex = 1)
points(te[, 1], predict(nn2_a, te), col = "red", cex = 1)
# NN for h2(x)
nn2_b <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10,
    startweights = winit, act.fct = h2)
# NN for h3(x)
nn2_c <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10,
    startweights = winit, act.fct = h3)
plot(tr, cex = 2, main = "Softplus -> h1(x) = x")
points(te, col = "blue", cex = 1)
points(te[, 1], predict(nn2_c, te), col = "red", cex = 1)
Var <- runif(500, min = 0, max = 50)  # Sample 500 points
mydata_2 <- data.frame(Var, Sin = sin(Var))

plot(tr, cex = 2, xlim = c(0, 50), ylim = c(-11, 1.2))
points(mydata_2, col = "blue", cex = 1)
points(mydata_2[, 1], predict(nn, mydata_2), col = "red", cex = 1)
# Obtaining the weights and bias from neural network input
# to layer
WB1 <- data.frame(Weights = nn$weights[[1]][[1]][2, ], Bias = nn$weights[[1]][[1]][1,
    ])
```

```
WB1
Var <- runif(500, min = 0, max = 10)  # Sample 500 points
mydata_3 <- data.frame(Sin = sin(Var), Var)

nn3 <- neuralnet(formula = Var ~ Sin, data = mydata_3, hidden = 10,
    startweights = winit, threshold = 0.1)
pred_3 <- predict(nn3, mydata_3)

plot(mydata_3, cex = 2, col = "blue")
points(mydata_3[, 2], pred_3, col = "red", cex = 1)
legend(x = -1, y = 8.5, legend = c("Training data", "NN Predictions"),
    col = c("blue", "red"), cex = 1, pch = 1)
```