Computer Lab 3 Block 1 - Machine Learning | 732A99

Group A-20

Adesijibomi Aderinto | Simon Alsén | Mohamed Ali

12/13/2021

Question 1

KERNEL METHODS 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

To Account
1- Physical distance from a station to the point of interest. For this purpose, use the function *distHaversine* from the R package *geosphere*.
2- Distance between the day a temperature measurement was made and the day of interest.
3- Distance between the hour of the day a temperature measurement was made and the hour of interest.

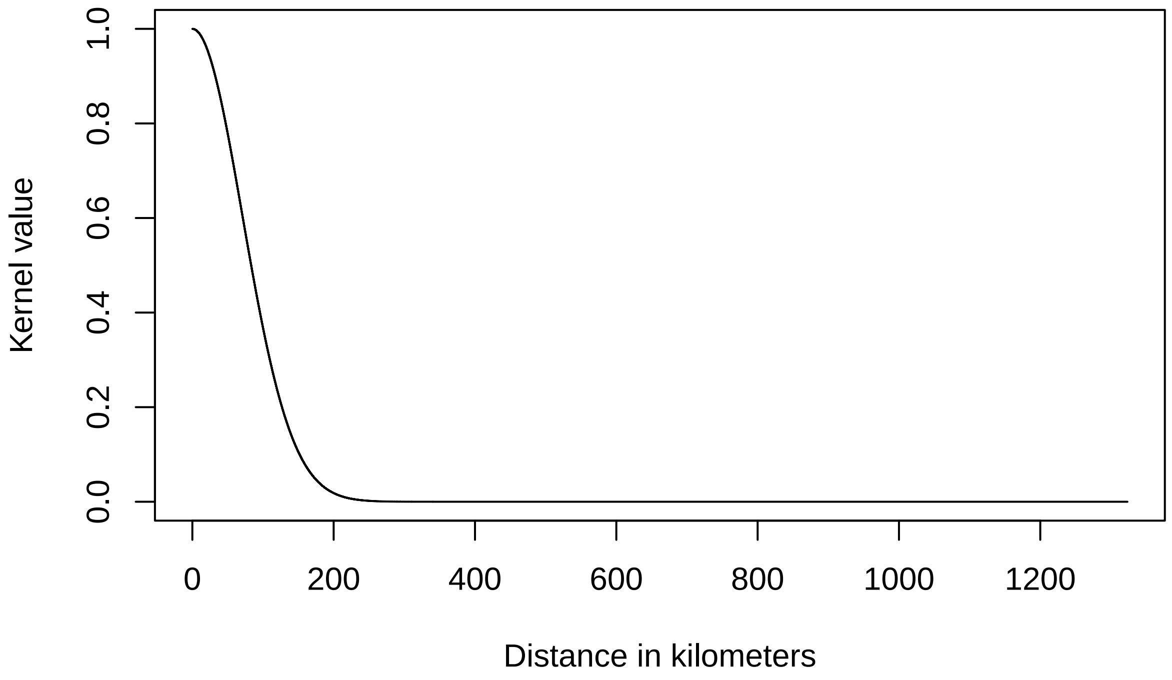
Smoothing coefficient

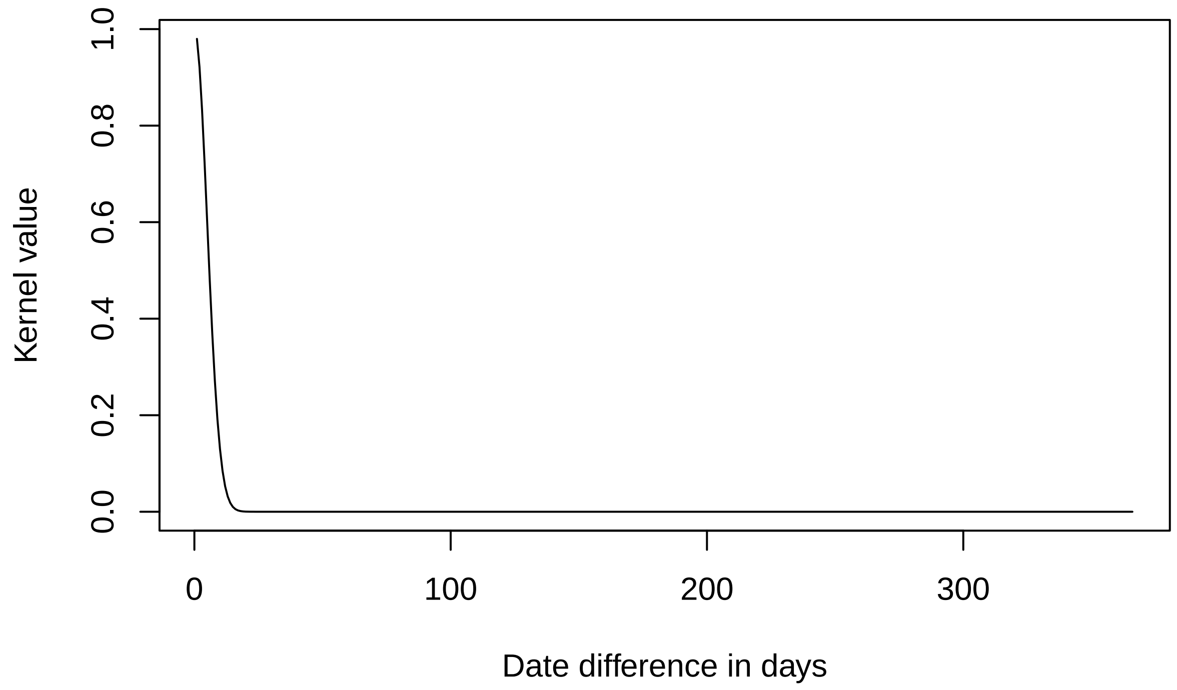
Choose an appropriate smoothing coefficient or width for each of the three kernels above. No cross-validation should be used. Instead, choose manually a width that gives large kernel values to closer points and small values to distant points. Show this with a plot of the kernel value as a function of distance.

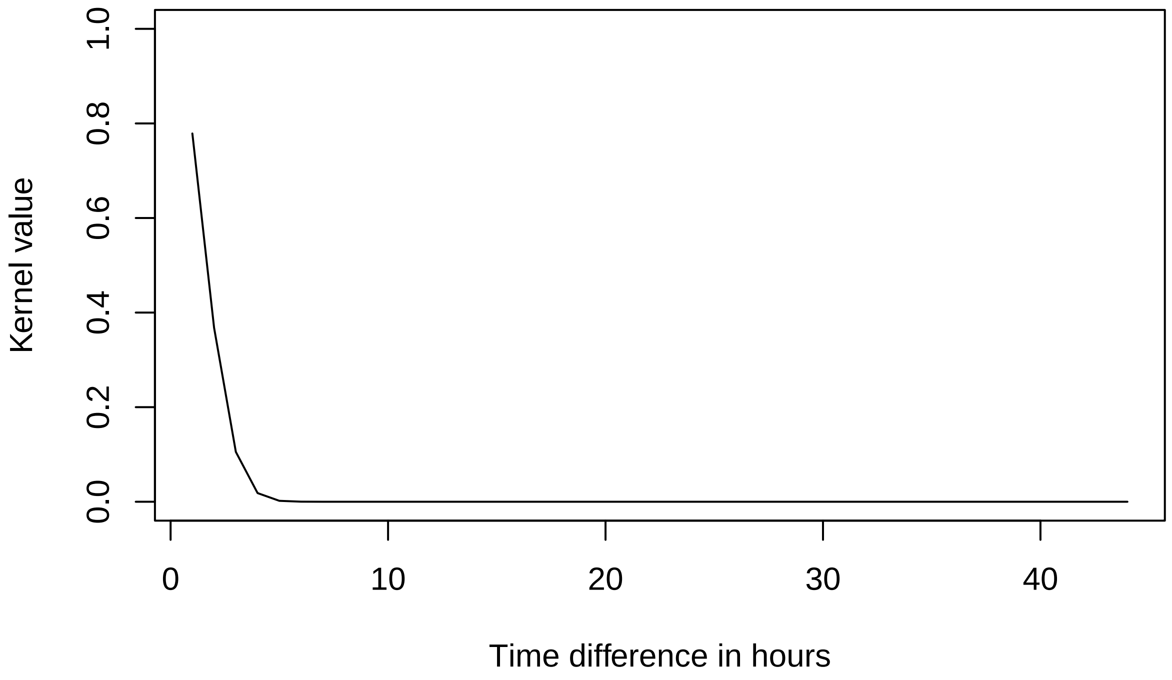
Solution

The coordinates for Linköping, (58.42, 15.63), are chosen as the point of interest, and 2010-04-15 is chosen as the date of interest. All dates posterior to this particular date is filtered out. The smoothing coefficients in the table below were selected for calculating the kernels for the distances. A smoothing coefficient of 100 000 meters, or 100 km, for the physical distances between the stations and the point of interest, 7 days for the distances between the day a temperature measurement was made and the day of interest, and 2 hours for the distances from the time a temperature measurement was made and the times of interest seems reasonable to remove noise and produce a model that generalizes well to new data, but without losing too much information.

Distance	h
Physical distance	100000
Date	7
Time	2

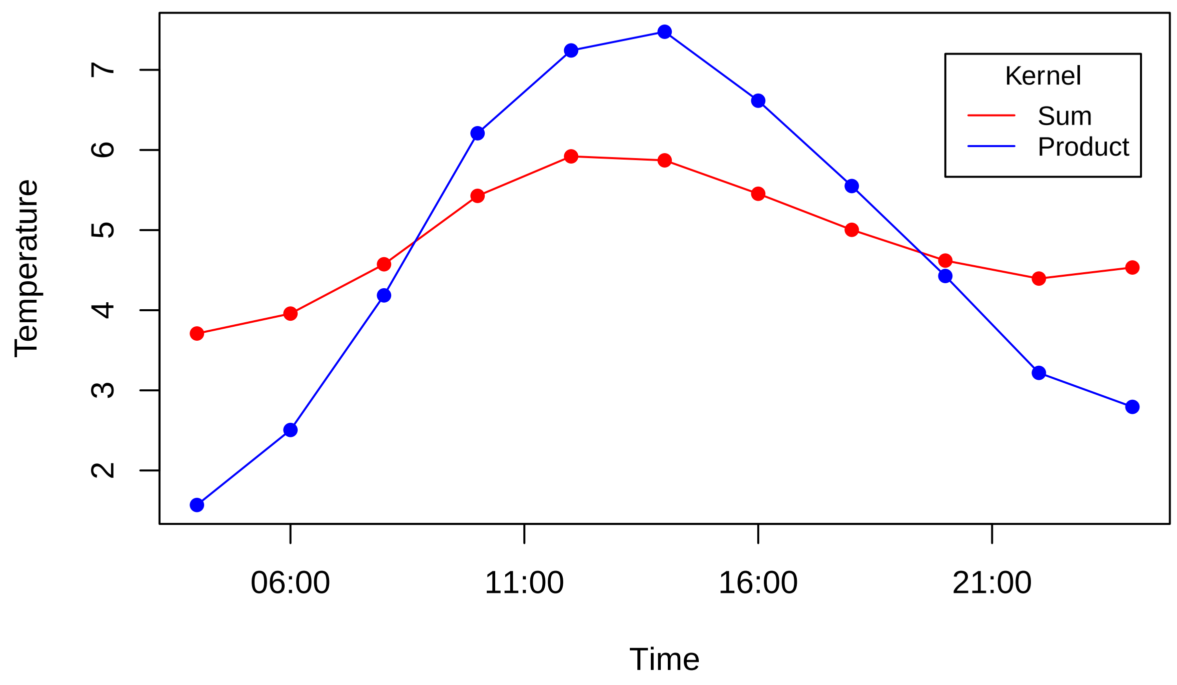

Kernel value
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
Distance in kilometers


Kernel value
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
Date difference in days



Comment
The three plots above shows the kernel values as functions of the corresponding distances. As the distances increases, the kernel values gets smaller. The smaller kernel value an observation is assigned, the less impact the observation will have on the forecasts. From the plots above it can be seen that measurements from stations that lies more than 250 km away from the point of interest, measurements from more than 18 days from the date of interest, and measurement from more than 5 hours from the time of interest have barely any impact on the forecasts.

Forcast



Conclusion The forecasts look quite reasonable. However, it can be seen that the kernels that were combined by multiplication results in more varying forecasts. When combining the three kernels by multiplication, the combined kernel will only take on a large value if all of the three base kernels have large values. When instead combining the three kernels by summation, a single, large kernel value from one of the three base kernels will make its corresponding observation more influential.

Question 2

SUPPORT VECTOR MACHINES

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

Model	Misclassification error	Data used for training	Data used for evaluation
filter0	0.0675000	Training	Validation
filter1	0.0848939	Training	Test
filter2	0.0823970	Training/validation	Test
filter3	0.0212235	Complete dataset	Test

`filter3` has the lowest misclassification error, but this is because the model is evaluated on a subset of the data that was used to train the model. `filter2` uses different datasets for training and testing, however a subset of the training data was also used for tuning the hyperparameter `C`. `filter1` properly uses the validation dataset for hyperparameter tuning, the training dataset for fitting the model, and the test dataset for evaluation. Because of this, `filter1` should be returned to the user.

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

The generalization error gives an estimate of how well the model will perform on future data. `err1` gives the estimate of the generalization error of the filter returned to the user, and is the misclassification error for the only filter that is being properly evaluated using previously unseen data.

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.

Predictions	kernlab predicitions
-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

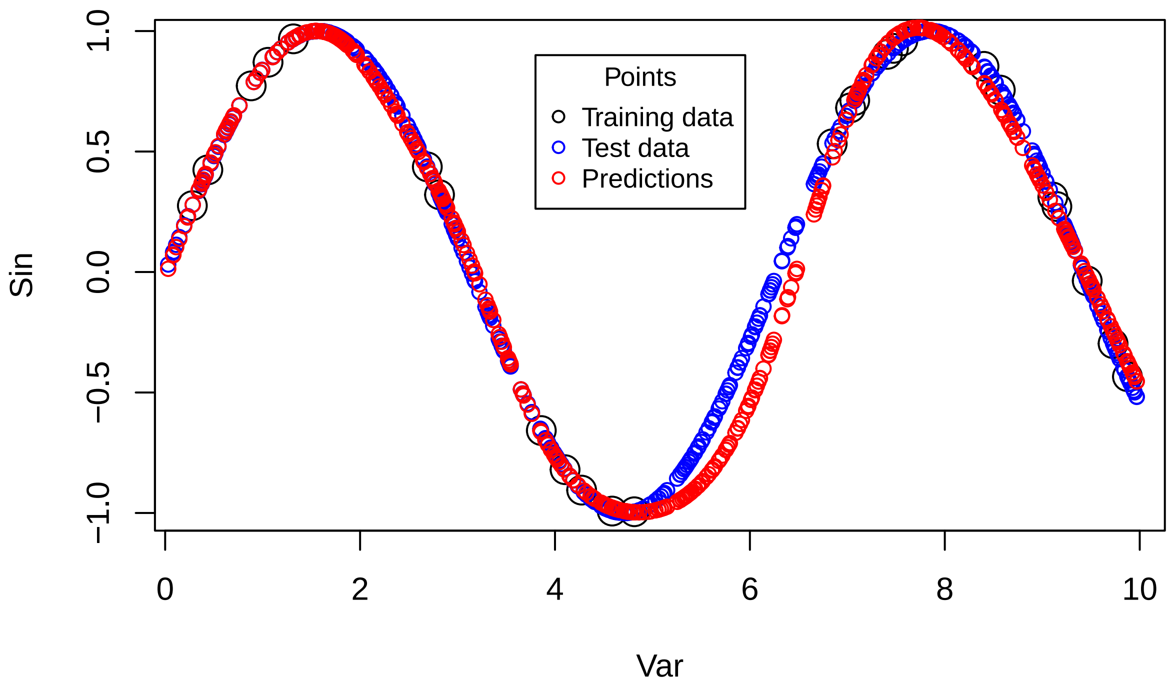
The results from the implemented linear combination matches the prediction from the `predict()` function from the `kernlab` package exactly.

Question 3

NEURAL NETWORKS

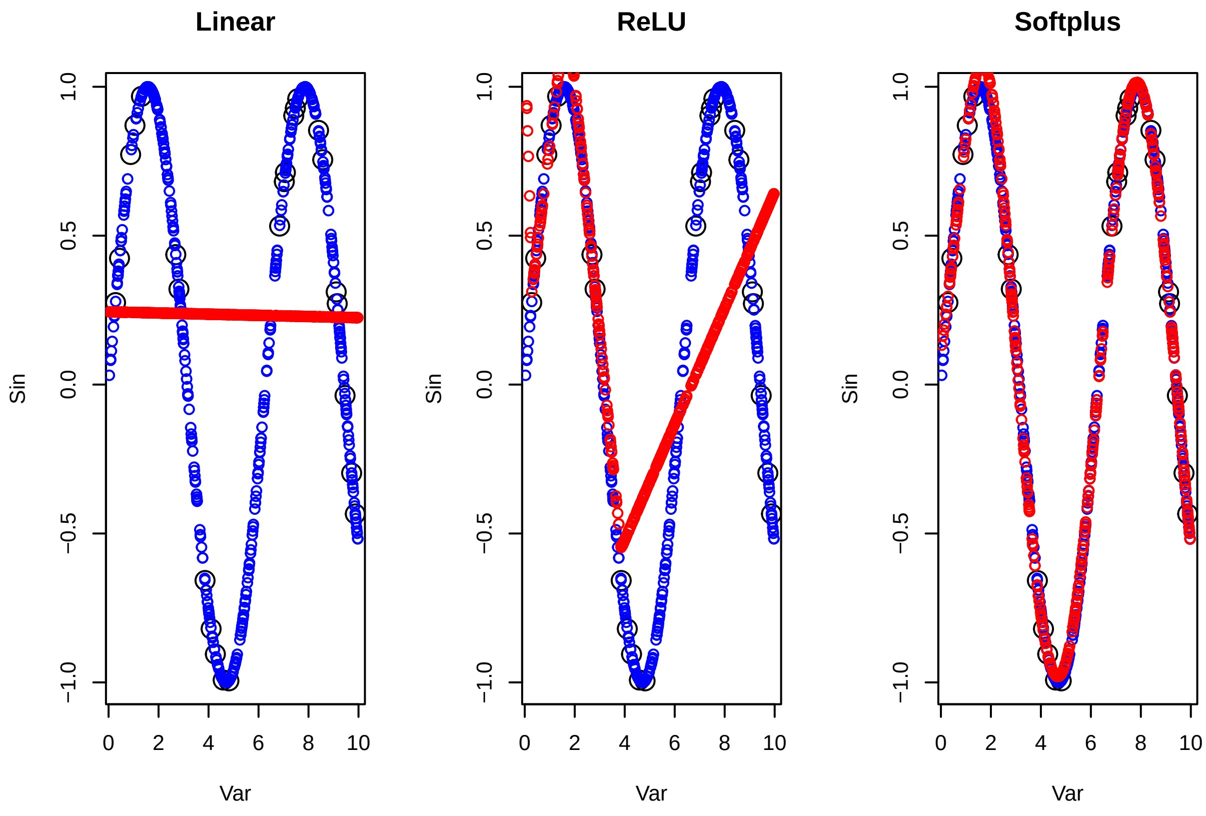
Train a neural network to learn the trigonometric sine function

```
## Warning: package 'neuralnet' was built under R version 4.1.2
```



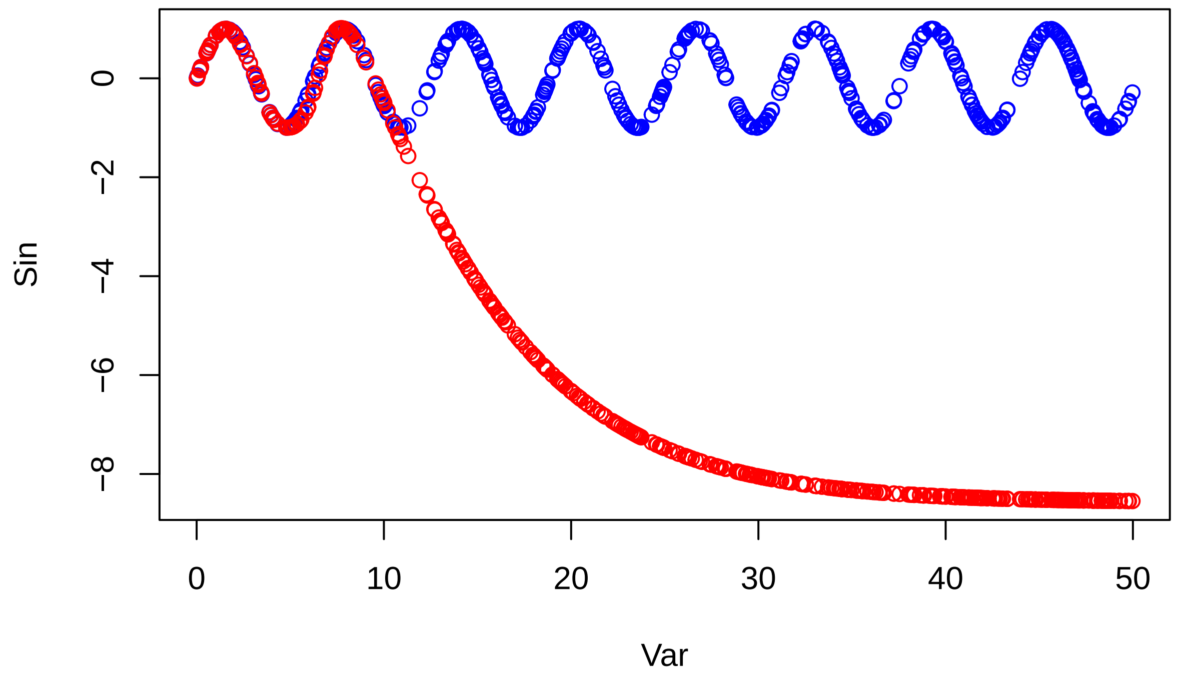
The neural network seems to predict the values of the sine function quite well. However, the model have some problems predicting the sine values between $x = 4.8$ and $x = 6.8$. This is most likely caused by the function changing direction, combined with a lack of training points in this particular region.

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: h1(x) = x, h2(x) = max{0; x} and h3(x) = ln(1 + exp 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.



The neural network that uses the softplus activation function is the only one of the three models that manages to make somewhat accurate predictions of the value of the sine function. The sine function is a non-linear function, and a linear activation function is only able to create linear functions, hence the poor performance in this case. A possible reason why the ReLU activation function performs so poorly could be because of dying neurons, which occurs when too many neurons return 0 and consequently stops the weights from being updated.

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.



After $x = 10$, the neural network fails to accurately predict the value of the sine function. Because values greater than 10 falls outside of the range of the training data, the model will assume that the previous pattern continues for values greater than 10 which is not accurate since the sine function is a non-linear function.

In question (3), the predictions seem to converge to some value. Explain why this happens.

W1	b
-0.8033506	3.3046893
-0.8324709	-0.3271777
-0.1499125	0.4043669
-0.8256431	-0.7669631
-1.8025966	11.9551906
0.7993943	-0.2057986
3.0821365	-6.4721996
-2.3207139	7.9048680
0.1543628	-0.5708964
0.7628867	0.0434247

The predictions seem to converge to the value -8.57. The predictions are calculated as follows:

$$\hat{y} = W_1 x_1 + b$$

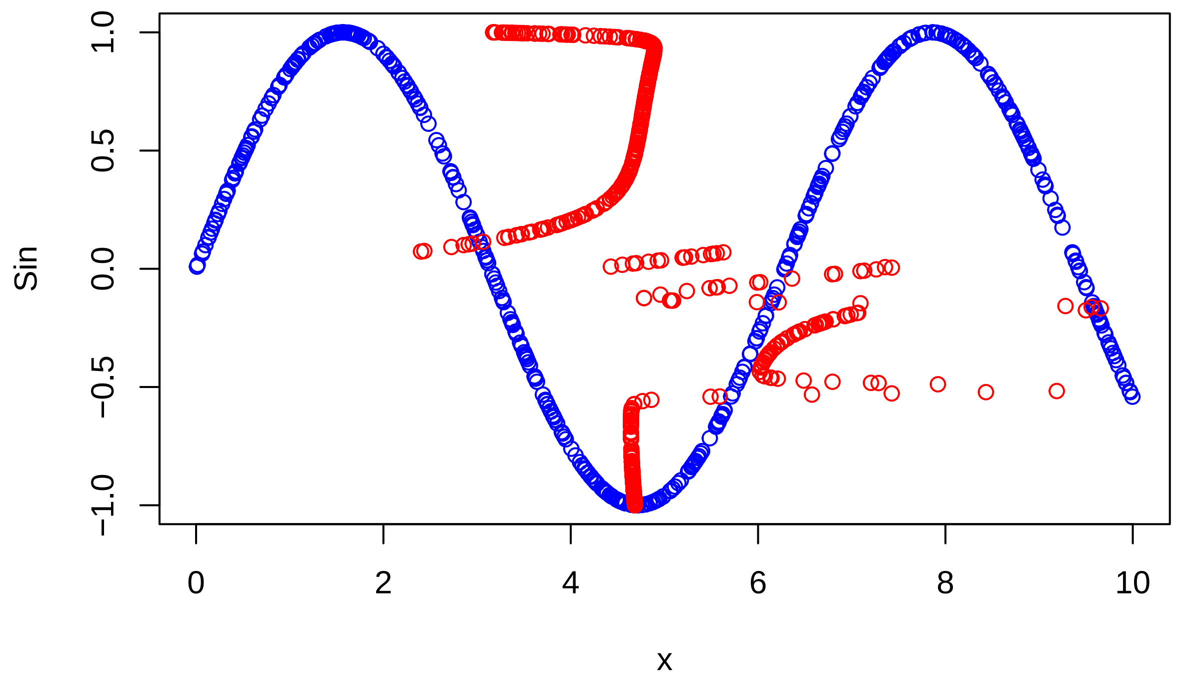
As the value of x increases, the offset term b will become less influential than W_1. The logistic activation function will approach 0 as the input value gets larger. Because of this, weight 6, 7, 9 and 10 will approach 1 while the remaining weights will approach 0 as the value of x increases.

This means that only neuron 6, 7, 9 and 10 in the hidden layer will become activated for large x:s. By summing the corresponding weights for these neurons and the offset term b, the value -8.573145 is obtained.

W2
-4.9649102
-4.8295057
22.1703831
-5.5590648
-4.3747945
0.3489759
-0.7224382
1.8612110
-9.4390597
0.4400295

b
0.7993476

Sample 500 points uniformly at random in the interval [0;10], and apply the sine function to each point.



The model does not manage to predict x from $\sin(x)$. The sine function is not an one-to-one function, which means that more than one element in the domain corresponds to the same element in the functions range. For example, $\sin(\frac{\pi}{2})$, $\sin(\frac{5\pi}{2})$ and $\sin(\frac{9\pi}{2})$ are all equal to 1 which makes it impossible for the neural network to predict a point x from the value of $\sin(x)$.

Statement of Contribution:

Simon, Mohamed and Adesijibomi devised the whole assignment together, the main conceptual ideas and codes outline. Mohamed worked out Assignment 1 (KERNEL METHODS), Simon worked with Assignment 2 (SUPPORT VECTOR MACHINES), Adesijibomi worked with (NEURAL NETWORKS).

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)

set.seed(1234567890)
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
# Points of interest
a <- 58.42
b <- 15.63

# Date of interest
date <- as.POSIXlt(as.Date("2010-04-15"), format = "%d-%m-%Y")

# Filter dates
st_filtered <- st[which(as.Date(st$date) < as.Date(date)),]
st_filtered$date <- as.POSIXlt(as.Date(st_filtered$date), format = "%d-%m-%Y")
st_filtered$time <- as.POSIXct(st_filtered$time, format = "%H:%M:%OS")

# Times of interest
times <- as.POSIXct(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"), format = "%H:%M:%OS")

# Calculating distance and differences between dates and times
diff_distance <- distHaversine(st_filtered[,c(4,5)], c(a,b))
diff_day <- date$yday - st_filtered$date$yday
diff_time <- sapply(times, function(x) difftime(x, st_filtered$time, units = "hours"))
colnames(diff_time) <- times

# Smoothing bandwidth
h_distance <- 100000
h_date <- 7
h_time <- 2
knitr::kable(data.frame(Distance = c("Physical distance", "Date", "Time"),
                        h = format(c(100000, 7, 2), scientific = FALSE)))

# Plots of kernel value as a function of distances
plot((seq(min(diff_distance):max(diff_distance)) / 1000),
     exp(-(seq(min(diff_distance):max(diff_distance)) / h_distance)^2),
     type = "l", xlab = "Distance in kilometers", ylab = "Kernel value")

plot(seq(min(diff_day):max(diff_day)),
     exp(-(seq(min(diff_day):max(diff_day)) / h_date)^2),
     type = "l", xlab = "Date difference in days", ylab = "Kernel value")

plot(seq(min(diff_time):max(diff_time)),
     exp(-(seq(min(diff_time):max(diff_time)) / h_time)^2),
     type = "l", xlab = "Time difference in hours", ylab = "Kernel value", ylim = c(0,1))
# Calculating Gaussian kernels for distances and differences
kernel_dist <- (1/sqrt(2*pi))*exp(-(diff_distance^2/(2*h_distance^2)))
kernel_date <- (1/sqrt(2*pi))*exp(-(diff_day^2/(2*h_date^2)))
kernel_time <- (1/sqrt(2*pi))*exp(-(diff_time^2/(2*h_time^2)))

# Create data frame for storing the predictions
pred <- as.data.frame(matrix(0, nrow = 11, ncol = 3))
colnames(pred) <- c("Time", "Sum", "Prod")
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")

# Loop through the times of interest
for(i in 1:ncol(diff_time)) {
  pred[i,1] <- times[i]
  # Sum the kernels
  kernel_sum <- kernel_dist + kernel_date + kernel_time[,i]
  pred[i,2] <- sum(kernel_sum * st_filtered$air_temperature) / sum(kernel_sum)
  # Multiplying the kernels
  kernel_prod <- kernel_dist * kernel_date * kernel_time[,i]
  pred[i,3] <- sum(kernel_prod * st_filtered$air_temperature) / sum(kernel_prod)
}

# Plot of predictions
plot(as.POSIXct(pred$Time, format = "%H:%M:%OS"), pred$Sum, type = "l",
     xlab = "Time", ylab = "Temperature",
     ylim = c(min(pred[,2:3]), max(pred[,2:3])),
     col = "red")
points(as.POSIXct(pred$Time, format = "%H:%M:%OS"), pred$Sum,
       pch = 16, col = "red")
lines(as.POSIXct(pred$Time, format = "%H:%M:%OS"), pred$Prod,
      col = "blue")
points(as.POSIXct(pred$Time, format = "%H:%M:%OS"), pred$Prod,
       pch = 16, col = "blue")
legend(as.POSIXct("20:00:00", format = "%H:%M:%OS"), 7.2, legend = c("Sum", "Product"),
       col = c("red", "blue"), lty = 1, cex = 0.8,
       title = "Kernel", text.font = 1)
library(kernlab)
set.seed(1234567890)

# Read data
data(spam)
```

```r
# Scale and partition data
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,]

# Testing different values for C and selecting the one with lowest validation error
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))
}

# Fitting SVM with Gaussian kernel to training dataset
filter0 <- ksvm(type ~ ., data = tr,
                kernel = "rbfdot", kpar = list(sigma = 0.05),
                C = which.min(err_va) * by, scaled = FALSE)
# Predicting classes for observations in validation dataset
mailtype <- predict(filter0, va[,-58])
# Calculating misclassification error
t <- table(mailtype, va[,58])
err0 <- (t[1,2] + t[2,1]) / sum(t)

# Fitting SVM with Gaussian kernel to training dataset
filter1 <- ksvm(type ~ ., data = tr,
                kernel = "rbfdot", kpar = list(sigma = 0.05),
                C = which.min(err_va) * by, scaled = FALSE)
# Predicting classes for observations in test dataset
mailtype <- predict(filter1,te[,-58])
# Calculating misclassification error
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)

# Fitting SVM with Gaussian kernel to combined training/validation dataset
filter2 <- ksvm(type ~ ., data = trva,
                kernel = "rbfdot", kpar = list(sigma = 0.05),
                C = which.min(err_va) * by, scaled = FALSE)
# Predicting classes for observations in test dataset
mailtype <- predict(filter2, te[,-58])
# Calculating misclassification error
t <- table(mailtype, te[,58])
err2 <- (t[1,2] + t[2,1]) / sum(t)

# Fitting SVM with Gaussian kernel to the complete dataset
filter3 <- ksvm(type ~ ., data = spam,
                 kernel = "rbfdot", kpar = list(sigma = 0.05),
                 C = which.min(err_va) * by, scaled = FALSE)
# Predicting classes for observations in test dataset
mailtype <- predict(filter3, te[,-58])
# Calculating misclassification error
t <- table(mailtype, te[,58])
err3 <- (t[1,2] + t[2,1]) / sum(t)

err_df <- data.frame(c("filter0", "filter1", "filter2", "filter3"),
                     c(err0, err1, err2, err3),
                     c("Training", "Training", "Training/validation", "Complete dataset"),
                     c("Validation", "Test", "Test", "Test"))
colnames(err_df) <- c("Model", "Misclassification error", "Data used for training",
                      "Data used for evaluation")
knitr::kable(err_df)
# Obtaining indexes of the support vectors
sv <- alphaindex(filter3)[[1]]
# Obtaining the coefficients
co <- as.matrix(coef(filter3)[[1]])
# Obtaining the negative intercept
inte<- - b(filter3)

# Creating matrix with features for the first 10 points
x <- as.matrix(spam[1:10,-58])
# Creating matrix with responses on lines matching the support vector indexes
y <- as.matrix(spam[sv,-58])

# Obtaining sigma value from the model filter3
sigma <- kpar(kernelf(filter3))$sigma

# Creating matrix to store predictions
pred  <- matrix(0, nrow = 10, ncol = 1)

# Predictions for the first 10 points
for(i in 1:10) {
  for(j in 1:length(sv)) {
    pred[i,] <- pred[i,] + exp(sigma * (2*x[i,] %*% y[j,] - sum(y[j,] * y[j,])
                                        - sum(x[i,] * x[i,]))) %*% co[j,]
  }
  pred[i,] <- pred[i,] + inte
}

# Comparing predictions with predictions from kernlab's prediction function
kernlab_pred <- predict(filter3,spam[1:10,-58], type = "decision")
pred_df <- data.frame(pred, kernlab_pred)
colnames(pred_df) <- c("Predictions", "kernlab predicitions")
knitr::kable(pred_df)
library(neuralnet)

# Sampling 500 points uniformly at random in the interval [0,10]
set.seed(1234567890)
Var <- runif(500, 0, 10)
# Applying sine function to each point
```

```r
mydata <- data.frame(Var, Sin = sin(Var))
# Partioning data into training and test datasets
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test

# Random initialization of the weights in the interval [-1, 1]
winit <- runif(10, -1, 1)

# Training a neural network with 10 hidden nodes
nn <- neuralnet(Sin ~ Var,
                data = tr,
                hidden = 10,
                startweights = winit)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
legend(3.8, 0.9, legend = c("Training data", "Test data", "Predictions"),
       col = c("black", "blue", "red"), cex = 0.8, pch = 1,
       title = "Points", text.font = 1)
# Linear activation function
h1 <- function(x) x

# ReLU activation function
h2 <- function(x) ifelse(x >= 0, x, 0)

# softplus activation function
h3 <- function(x) log(1+exp(x))

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

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

# softplus activation function
nn3 <- neuralnet(Sin ~ Var,
                 data = tr,
                 hidden = 10,
                 startweights = winit,
                 act.fct = h3)

# Plotting response from training data (black), test data (blue) and prediction (red)
par(mfrow = c(1,3))
```

```r
plot(tr, cex = 2, main = "Linear")
points(te, col = "blue", cex = 1)
points(te[,1], predict(nn1, te), col = "red", cex = 1)

plot(tr, cex = 2, main = "ReLU")
points(te, col = "blue", cex = 1)
points(te[,1], predict(nn2, te), col = "red", cex = 1)

plot(tr, cex = 2, main = "Softplus")
points(te, col = "blue", cex = 1)
points(te[,1], predict(nn3, te), col = "red", cex = 1)
# Sampling 500 points uniformly at random in the interval [0,50]
Var <- runif(500, 0, 50)
# Apply sine function to each point
newdata <- data.frame(Var, Sin = sin(Var))

# Predicting sine function value using neural network from question 1
pred <- predict(nn, newdata)

# Plot the predictions (red) and the actual value (blue)
plot(newdata, col = "blue", cex=1, ylim = c(min(pred), max(pred)))
points(newdata[,1], predict(nn,newdata), col="red", cex=1)
# Obtaining the weights from neural network in step 1
knitr::kable(data.frame(W1 = nn$weights[[1]][[1]][2,],
                        b = nn$weights[[1]][[1]][1,]))
# Obtaining the weights from neural network in step 1
knitr::kable(data.frame(W2 = nn$weights[[1]][[2]][-1,]))
knitr::kable(data.frame(b = nn$weights[[1]][[2]][1,]))
# Sampling 500 points uniformly at random in the interval [0,10]
x <- runif(500, 0, 10)
# Applying sine function to each point
data_3_5 <- data.frame(x, Sin = sin(x))

# Training neural network with the sampled points as the response
nn4 <- neuralnet(x ~ Sin,
                 data = data_3_5,
                 hidden = 10,
                 threshold = 0.1)

# Using the neural network to predict the values of the sampled points
pred_3_5 <- predict(nn4, data_3_5)

# Plotting the actual values of the sampled points (blue) and the predictions (red)
plot(data_3_5, col = "blue", cex=1)
points(pred_3_5, data_3_5[,2], col = "red", cex = 1)
```