# Machine Learning - Lab03 - Group A2

*Thijs Quast (thiqu264), Anubhav Dikshit(anudi287), Lennart Schilling(lensc874)*

*18-12-2018*

## Contents

## Contributions

For this report Thijs and Anubhav focused on assignment 1. Lennart focused on assignment 2. All code was written individually and independently.

# Assignment 1 - Kernel Methods

```r
options("jtools-digits" = 2, scipen = 999)

# Import data
rm(list=ls())
set.seed(1234567890)
library(geosphere)
library(readr)
stations <- read_csv("stations.csv")
temps <- read_csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")


myfunction <- function(st, latitude, longitude, h_distance, h_date, h_time){
# Determine the coordinates of the location to be predicted.
# Also, determine the gaussian distances. These are further elaborated on in the report.


# Altering the format of dates
date <- as.POSIXct("2013-11-04")
end_date <- date + as.difftime(1, units="days")
minutes <- 60
times <- seq(from = date, by = minutes*120, to = end_date)
times <- as.data.frame(times)
times_nr <- c(1:13)
times <- cbind(times, times_nr)

# Merge datasets, so for each timeframe a "seperate" set of observations are available
data <- merge(st, times, all = TRUE)

data$target_latitude <- latitude
data$target_longitude <- longitude
data$delta_distance <- abs(distHaversine(p1 = data[,c("target_longitude", "target_latitude")],
                                         p2 = data[,c("longitude", "latitude")]))



# Create target dates and target times
data$target_date <- as.Date(data$times)
data$target_time <- format(data$times,"%H:%M:%S")

# Difference in dates from measurement dates to target date
data$delta_date <- (abs(difftime(data$target_date, data$date, units = c("days"))))
data$delta_date <- as.numeric(data$delta_date)


# Difference in dates from measurement times to target times
x <- strptime(paste(data$target_date, data$target_time),format = "%Y-%m-%d %H:%M:%S")
y <- strptime(paste(data$target_date, data$time),format = "%Y-%m-%d %H:%M:%S")
data$delta_time <- difftime(x, y, units ="hours")
data$delta_time <- abs(data$delta_time)
data$delta_time <- as.integer(data$delta_time)
```

```r
# Dropping timeframes which we are not asked to predict
data <- data[!(data$times_nr=="1" | data$times_nr=="2"),]

# Identifying measurements that are post target
data$date_and_time <- paste(data$date, data$time)
data$diff_date_and_time <- as.numeric(difftime(data$target_date, data$date_and_time, units = c("hour")))

# Dropping measurements that are post target
data <- data[!(data$diff_date_and_time < 0 & data$delta_distance < 0), ]

temp <- vector(length=length(times))

# Kernal distances
data$kernal_distance <- exp(-(data$delta_distance/h_distance)^2)
data$kernal_date <- exp(-(data$delta_date/h_date)^2)
data$kernal_time <- exp(-(data$delta_time/h_time)^2)

# Now I compute the kernal estimations, using the formula on slide 8 lecture 3aBlock1

# Kernal summation
data$summation_numerator <- (data$kernal_distance * data$air_temperature) +
  (data$kernal_date * data$air_temperature) +
  (data$kernal_time * data$air_temperature)

data$summation_denominator <- data$kernal_distance + data$kernal_date + data$kernal_time

# Kernal multiplication
data$multiplication_numerator <- (data$kernal_distance * data$air_temperature) *
  (data$kernal_date * data$air_temperature) *
  (data$kernal_time * data$air_temperature)

data$multiplication_denominator <- (data$kernal_distance * data$kernal_date * data$kernal_time)

# Create empty vectors for the predicted data to be stored in.
temp_sum <- c()
temp_mult <- c()

# Loop over each to be predicted time interval.
for (i in times_nr){
  sub_data <- data[data$times_nr == i,]
  temp_sum[i] <- sum(sub_data$summation_numerator) / sum(sub_data$summation_denominator)
  temp_mult[i] <- sum(sub_data$multiplication_numerator) / sum(sub_data$multiplication_denominator)
}

# Create an dataframe for all the results.
temp_sum <- as.character(temp_sum)
temp_mult <- as.character(temp_mult)

outcome <- as.data.frame(cbind(temp_sum, temp_mult))
outcome$temp_sum <- as.numeric(outcome$temp_sum)
outcome$temp_mult <- as.numeric(outcome$temp_mult)
outcome$count <- c(1:13)
```

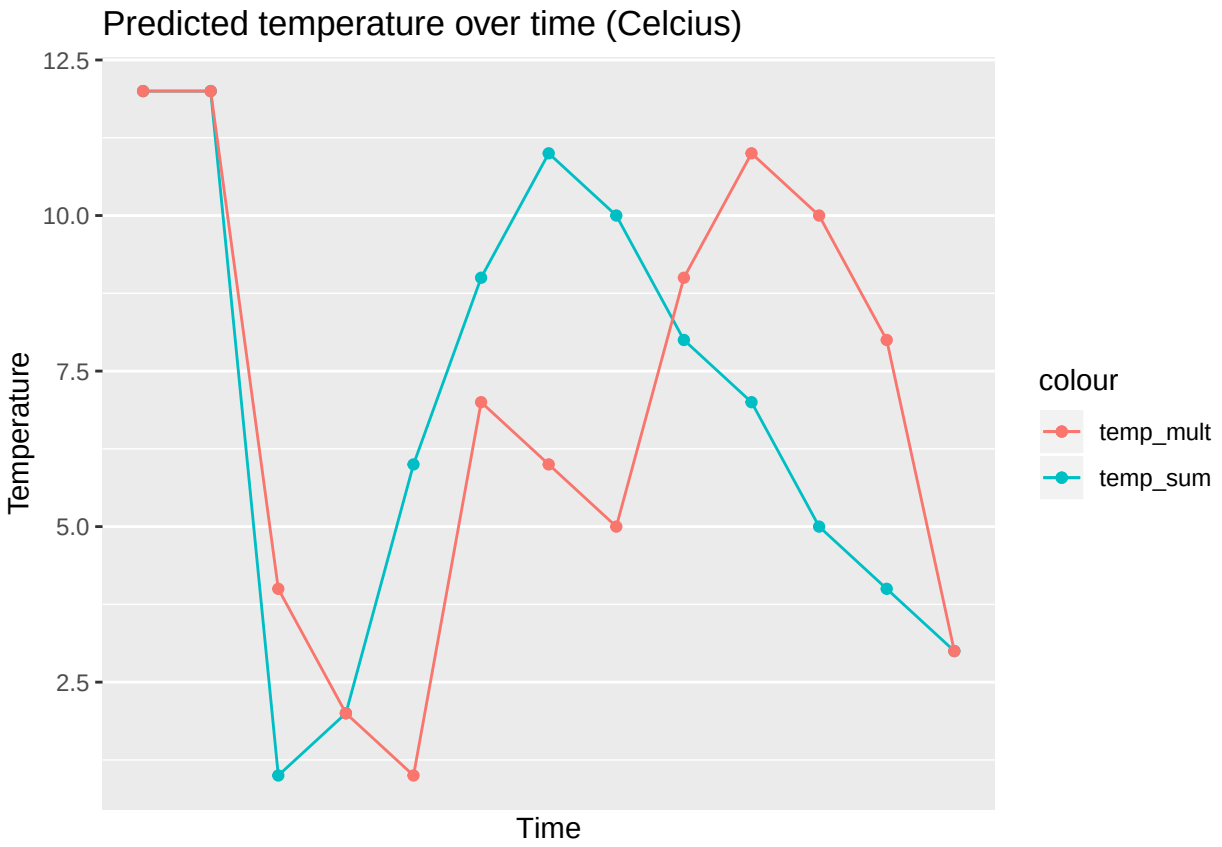
```r
# Plot outcomes, using ggplot2 package
library(ggplot2)

plot <- ggplot(data = outcome, aes(x = count, y = temp_sum, color = "temp_sum", group = 1)) +
  geom_point() + geom_line()

plot <- plot + geom_point(aes(y = temp_mult, color = "temp_mult", group = 1)) +
  geom_line(aes(y = temp_mult, color = "temp_mult", group = 1))

plot <- plot + scale_x_discrete()
plot <- plot + ylab("Temperature") + xlab("Time") + ggtitle("Predicted temperature over time (Celcius)")
return(plot)
}
```

```r
myfunction(st = st,latitude = 58.4274, longitude = 14.826, h_distance = 100000, h_date = 4, h_time = 2)
```




For the kernel widths I have chosen the following values:

Distance: 100 kilometers

Date: 4 days

Time: 2 hours

When checking the weather forecasts on 17-12-2018 at 20:00 on Google, the same temperature is reported for Linköping as it is for Stockholm. The distance between these two cities is 200 kilometers. Most likely temperatures will change a lot more severly if one travels a certain distance in a straight line from south to north than from west to east. To compensate for this, a distance measure of 100 kilometers is chosen, this seems reasonable, but a measure like this is always arbitrary.

A kernel distance of 4 days seems large, however weights decrease as distance in dates increases.

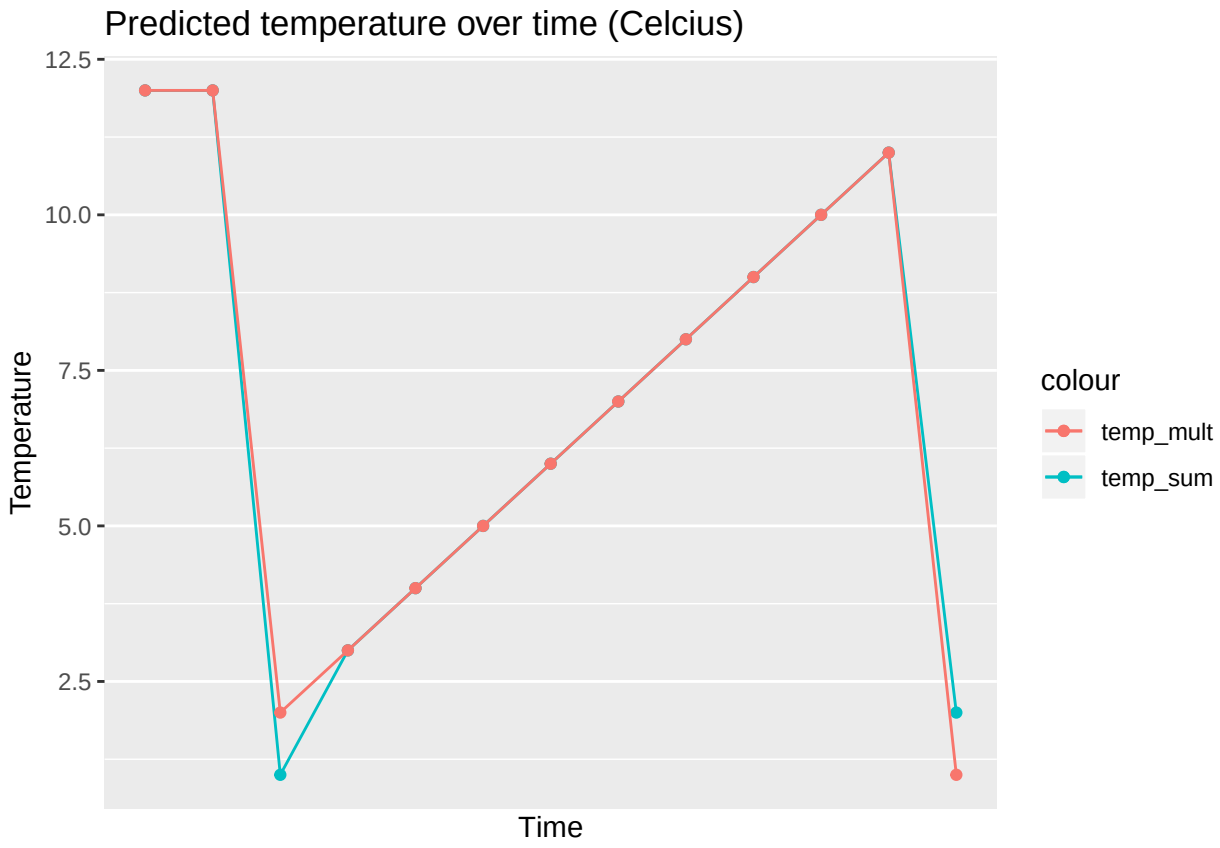
Google weather predictions report per three hours. However, I argue Google has more data to available for weather predictions than the given dataset. Therefore I choose a slightly smaller time interal of 2 hours. This seems reasonable as when one wakes up, temperatues can be colder then halfway the morning (i.e. 2 hours later), around lunchtime temperatures are usually slightly higher and around 14:00 or 15:00 the warmest point of the day is reached.

By definition of the computation applied for gaussian kernel distances, following the formula provided on slide 6, lecture 3a, a Gaussian kernel gives less weights to further observations.

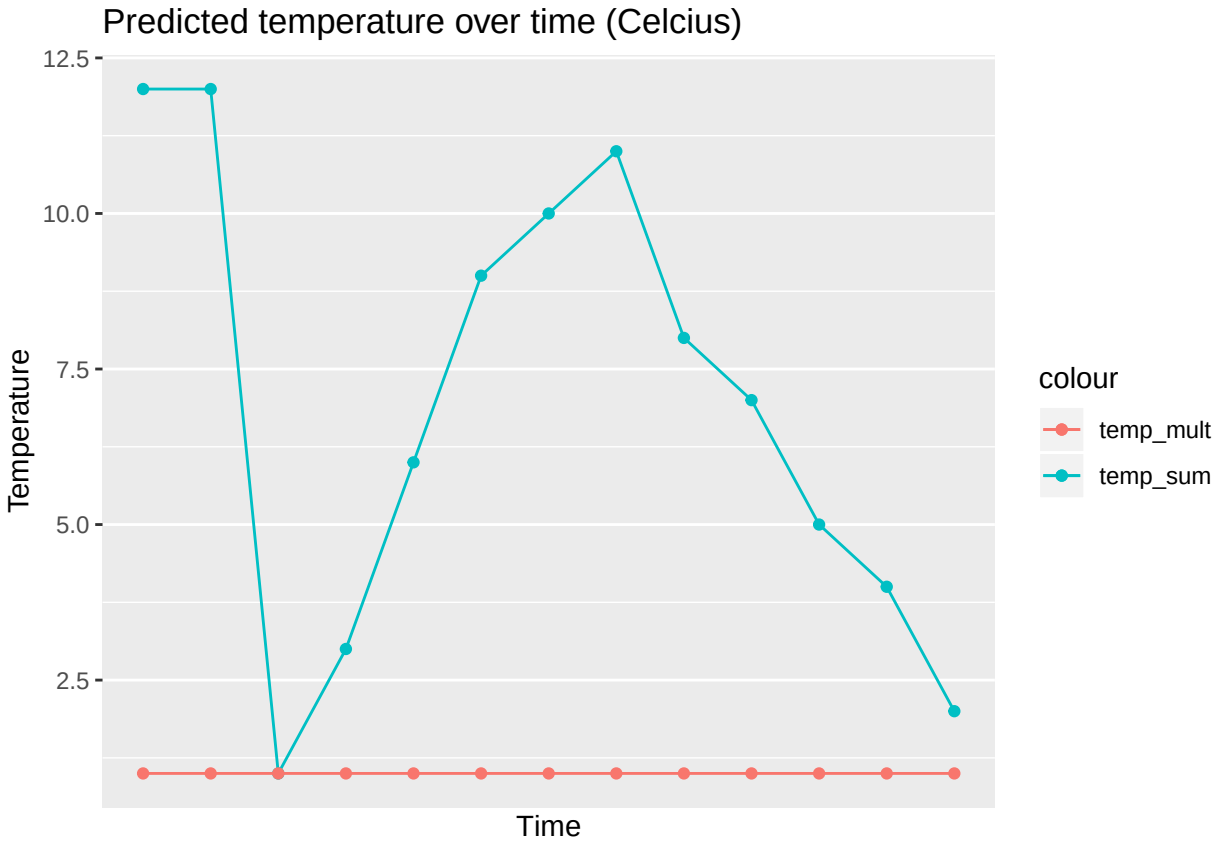
The predicted temperatures for the summation model seem to follow a reasonable pattern. Temperatures fall during the night and in the morning it is cold. During the morning, temperatures gradually rise, and the warmest point is reached in the afternoon. After this point, temperatues fall again during the night. For the multiplication model, the predicted temperatues however seem to follow a rather random pattern.

Results for both models differ, simply because of how the predictions are calculated. When one adds the kernel calculations with each other, this is a completely different thing then when multiplying them with each other. Many weights are small numbers, and if one multiplies these numbers with each other, very quickly, very small numbers arise. Which result in different predictions than when summing.

```
myfunction(st = st,latitude = 58.4274, longitude = 14.826, h_distance = 300000, h_date = 300, h_time = 3
```




```
myfunction(st = st,latitude = 58.4274, longitude = 14.826, h_distance = 10, h_date = 0.1, h_time = 0.1)
```




When one takes very high values for the function parameters, both temperatues for summation and multiplication linearly increase over time and reach their peak in the evening. When one takes very small values for parameters the model with kernel multiplications reaches very low values. This is easy to explain as when multiplying very small numbers with each other the outcome, thus predicted temperatues, will be very small.

## Assignment 2 - Support Vector Machines

In this assignment, a Support Vector Machine (SVM), a supervised learning model, will be used to classify spam dataset that is included within the *kernlab*-package which will be used for this assignment.

In the first step, the package will be loaded, attached and the *spam*-data frame will be read.

```
# loading/attaching kernlab library
library(kernlab)
# importing data
data(spam)
# printing nrow & ncol
dim(spam)
```

```
## [1] 4601   58
```

The *spam* data consists of 4601 observations emails described by in total 58 features. The *type*-feature classifies the mails as either *spam* or *nonspam*.

The model selection will be based on the *holdout method* which uses training data to fit the model and test data to evaluate it. To make sure that enough observations are integrated within both data sets, a relation of 70:30 will be chosen.

```
# dividing data into train and test set
n = dim(spam)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.7))
train = spam[id,]
test = spam[-id,]
```

Using the training data and the *radial basis function (RBF) kernel* with a width of 0.05, three different SVM-models with a different $C$-parameter (0.5, 1 and 5) will be fitted. Within every iteration, the test misclassification error respectively will be calculated. If it will be identified as the lowest error, the model will be saved as *bestModel*.

```
# setting up minimum misclassification rate to 1 for following loop
minMisclassificationRate = 1
# fitting svm-models for different parameter C
for (C in c(0.5, 1, 5)) {
  svmModel = ksvm(type ~ .,
                  data = spam,
                  kernel = "rbfdot",
                  kpar = list(sigma = 0.05),
                  C = C)
  classification = predict(svmModel , test[, -which(colnames(train) == "type")])
  misclassificationRate = mean(test$type != classification)
  print(paste0("Test misclassification error for C = ", C, ": ", round(misclassificationRate, 3)))
  print(table(y = test$type, yFit = predict(svmModel, test[, -which(colnames(test) == "type")])),
        caption = paste0("Confusion matrix for C = ", C, ": "))
  if (misclassificationRate < minMisclassificationRate) {
    minMisclassificationRate = misclassificationRate
    bestModel = svmModel
  }
}
```

```
## [1] "Test misclassification error for C = 0.5: 0.043"
##          yFit
## y         nonspam spam
##   nonspam     813   22
##   spam         38  508
## [1] "Test misclassification error for C = 1: 0.036"
##          yFit
## y         nonspam spam
##   nonspam     816   19
##   spam         31  515
## [1] "Test misclassification error for C = 5: 0.023"
##          yFit
## y         nonspam spam
##   nonspam     825   10
##   spam         22  524
```

Based on the *holdout method*, the model with the lowest test error (third model where $C = 5$) will be identified as the most optimal classifier. In the following, it is summarised:

```
# returning parameter C and erros of best model
knitr::kable(
  x = as.data.frame(
    cbind(C = bestModel@param,
          trainError = round(bestModel@error, 3),
```

```
          testError = round(minMisclassificationRate, 3))),
  caption = "Summary of best identified svm model",
  row.names = FALSE)
```

Table 1: Summary of best identified svm model

| C | trainError | testError |
|---|---|---|
| 5 | 0.022 | 0.023 |

Comparing the train and test error, it can be clearly seen that both values are very similar which indicates that neither too much overfitting nor underfitting seem to occur.

But accuracy is only half the story: Referring also to the confusion matrices, it confirms that model three correctly identifies most of the nonspam mail compared to the others. Thereby, if the aim is to minimize the likelihood of missing out important mails, then model 3 still is the best choice.

$C$ is the cost parameter which penalizes large residuals. So a larger cost will result in a more flexible model with fewer misclassifications. In effect the cost parameter allows you to adjust the bias/variance trade-off. The greater the cost parameter, the more variance in the model and the less bias.The greater the cost, the fewer misclassifications are allowed. Note that here we penalize the residuals resulting in higher variance and lower bias.

# Appendix

```
options("jtools-digits" = 2, scipen = 999)

# Import data
rm(list=ls())
set.seed(1234567890)
library(geosphere)
library(readr)
stations <- read_csv("stations.csv")
temps <- read_csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")


myfunction <- function(st, latitude, longitude, h_distance, h_date, h_time){
# Determine the coordinates of the location to be predicted.
# Also, determine the gaussian distances. These are further elaborated on in the report.


# Altering the format of dates
date <- as.POSIXct("2013-11-04")
end_date <- date + as.difftime(1, units="days")
minutes <- 60
times <- seq(from = date, by = minutes*120, to = end_date)
times <- as.data.frame(times)
times_nr <- c(1:13)
times <- cbind(times, times_nr)

# Merge datasets, so for each timeframe a "seperate" set of observations are available
```

```r
data <- merge(st, times, all = TRUE)

data$target_latitude <- latitude
data$target_longitude <- longitude
data$delta_distance <- abs(distHaversine(p1 = data[,c("target_longitude", "target_latitude")],
                                        p2 = data[,c("longitude", "latitude")]))



# Create target dates and target times
data$target_date <- as.Date(data$times)
data$target_time <- format(data$times,"%H:%M:%S")

# Difference in dates from measurement dates to target date
data$delta_date <- (abs(difftime(data$target_date, data$date, units = c("days"))))
data$delta_date <- as.numeric(data$delta_date)


# Difference in dates from measurement times to target times
x <- strptime(paste(data$target_date, data$target_time),format = "%Y-%m-%d %H:%M:%S")
y <- strptime(paste(data$target_date, data$time),format = "%Y-%m-%d %H:%M:%S")
data$delta_time <- difftime(x, y, units ="hours")
data$delta_time <- abs(data$delta_time)
data$delta_time <- as.integer(data$delta_time)


# Dropping timeframes which we are not asked to predict
data <- data[!(data$times_nr=="1" | data$times_nr=="2"),]

# Identifying measurements that are post target
data$date_and_time <- paste(data$date, data$time)
data$diff_date_and_time <- as.numeric(difftime(data$target_date, data$date_and_time, units = c("hour")))

# Dropping measurements that are post target
data <- data[!(data$diff_date_and_time < 0 & data$delta_distance < 0), ]

temp <- vector(length=length(times))

# Kernal distances
data$kernal_distance <- exp(-(data$delta_distance/h_distance)^2)
data$kernal_date <- exp(-(data$delta_date/h_date)^2)
data$kernal_time <- exp(-(data$delta_time/h_time)^2)

# Now I compute the kernal estimations, using the formula on slide 8 lecture 3aBlock1

# Kernal summation
data$summation_numerator <- (data$kernal_distance * data$air_temperature) +
  (data$kernal_date * data$air_temperature) +
  (data$kernal_time * data$air_temperature)

data$summation_denominator <- data$kernal_distance + data$kernal_date + data$kernal_time

# Kernal multiplication
```

```r
data$multiplication_numerator <- (data$kernal_distance * data$air_temperature) *
  (data$kernal_date * data$air_temperature) *
  (data$kernal_time * data$air_temperature)

data$multiplication_denominator <- (data$kernal_distance * data$kernal_date * data$kernal_time)

# Create empty vectors for the predicted data to be stored in.
temp_sum <- c()
temp_mult <- c()

# Loop over each to be predicted time interval.
for (i in times_nr){
  sub_data <- data[data$times_nr == i,]
  temp_sum[i] <- sum(sub_data$summation_numerator) / sum(sub_data$summation_denominator)
  temp_mult[i] <- sum(sub_data$multiplication_numerator) / sum(sub_data$multiplication_denominator)
}

# Create an dataframe for all the results.
temp_sum <- as.character(temp_sum)
temp_mult <- as.character(temp_mult)

outcome <- as.data.frame(cbind(temp_sum, temp_mult))
outcome$temp_sum <- as.numeric(outcome$temp_sum)
outcome$temp_mult <- as.numeric(outcome$temp_mult)
outcome$count <- c(1:13)

# Plot outcomes, using ggplot2 package
library(ggplot2)

plot <- ggplot(data = outcome, aes(x = count, y = temp_sum, color = "temp_sum", group = 1)) +
  geom_point() + geom_line()

plot <- plot + geom_point(aes(y = temp_mult, color = "temp_mult", group = 1)) +
  geom_line(aes(y = temp_mult, color = "temp_mult", group = 1))

plot <- plot + scale_x_discrete()
plot <- plot + ylab("Temperature") + xlab("Time") + ggtitle("Predicted temperature over time (Celcius)")
return(plot)
}
myfunction(st = st,latitude = 58.4274, longitude = 14.826, h_distance = 100000, h_date = 4, h_time = 2)
myfunction(st = st,latitude = 58.4274, longitude = 14.826, h_distance = 300000, h_date = 300, h_time = 3
myfunction(st = st,latitude = 58.4274, longitude = 14.826, h_distance = 10, h_date = 0.1, h_time = 0.1)
# loading/attaching kernlab library
library(kernlab)
# importing data
data(spam)
# printing nrow & ncol
dim(spam)
# dividing data into train and test set
n = dim(spam)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.7))
train = spam[id,]
```

```
test = spam[-id,]
# setting up minimum misclassification rate to 1 for following loop
minMisclassificationRate = 1
# fitting svm-models for different parameter C
for (C in c(0.5, 1, 5)) {
  svmModel = ksvm(type ~ .,
                  data = spam,
                  kernel = "rbfdot",
                  kpar = list(sigma = 0.05),
                  C = C)
  classification = predict(svmModel , test[, -which(colnames(train) == "type")])
  misclassificationRate = mean(test$type != classification)
  print(paste0("Test misclassification error for C = ", C, ": ", round(misclassificationRate, 3)))
  print(table(y = test$type, yFit = predict(svmModel, test[, -which(colnames(test) == "type")])),
        caption = paste0("Confusion matrix for C = ", C, ": "))
  if (misclassificationRate < minMisclassificationRate) {
    minMisclassificationRate = misclassificationRate
    bestModel = svmModel
  }
}
# returning parameter C and erros of best model
knitr::kable(
  x = as.data.frame(
    cbind(C = bestModel@param,
          trainError = round(bestModel@error, 3),
          testError = round(minMisclassificationRate, 3))),
  caption = "Summary of best identified svm model",
  row.names = FALSE)
```