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.

Loading The Libraries

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(geosphere, kernlab, geosphere, ggplot2,caret)

set.seed(12345)
options("jtools-digits" = 2, scipen = 999)

# colours (colour blind friendly)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2",
               "#D55E00", "#CC79A7")
```

Assignment 1 - Kernel Methods

```
rm(list=ls())
set.seed(1234567890)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
rm(temps, stations)
```

defining the function

```
kernel_method <- function(df, date, loc_long, loc_lat, h1, h2, h3) {

set.seed(1234567890)
start <- as.POSIXct(date)
interval <- 60
end <- start + as.difftime(1, units="days")
time_seq <- seq(from=start, by=interval*120, to=end)
time_seq <- as.data.frame(time_seq)
colnames(time_seq) <- "new_date_time"
time_seq$time_index <- rownames(time_seq)

df_new <- merge.data.frame(df,time_seq,all=TRUE)
rm(df)

df_new$new_date <- as.Date(df_new$new_date_time)
df_new$new_time <- format(df_new$new_date_time,"%H:%M:%S")
df_new$loc_long <- loc_long
df_new$loc_lat <-  loc_lat


df_new$h_distance <-  abs(distHaversine(p1 = df_new[,c("loc_long", "loc_lat")],
                                        p2 = df_new[,c("longitude", "latitude")]))

df_new$h_date <- as.numeric(abs(difftime(df_new$new_date, df_new$date, units = c("days"))))

df_new$h_time <- as.numeric(abs(difftime(strptime(paste(df_new$new_date,
                                                        df_new$new_time),"%Y-%m-%d%H:%M:%S"),
```

```
                                        strptime(paste(df_new$new_date, df_new$time),
                                    "%Y-%m-%d %H:%M:%S"),
                            units = c("hour"))))


df_new$date_time <- paste(df_new$date, df_new$time)
df_new$hd_dist <- as.numeric(difftime(df_new$new_date_time,
                            df_new$date_time,
                            units = c("hour")))

## removing any negative dates and time
df_new$posterior_flag <- as.factor(ifelse(df_new$h_distance > 0 & df_new$hd_dist > 0, "retain", "drop"))


## calculating kernel distance and choosing guassian kernel
df_new$h_distance_kernel <- exp(-(df_new$h_distance/h1)^2)
df_new$h_date_kernel <- exp(-(df_new$h_date/h2)^2)
df_new$h_time_kernel <- exp(-(df_new$h_time/h3)^2)
df_new$total_additive_dist <- (df_new$h_distance_kernel + df_new$h_date_kernel + df_new$h_time_kernel)
df_new$total_mul_dist <- (df_new$h_distance_kernel * df_new$h_date_kernel * df_new$h_time_kernel)

df_new$additive_num <- ifelse(df_new$posterior_flag == "retain",
                              df_new$h_distance_kernel*df_new$air_temperature +
                                df_new$h_date_kernel*df_new$air_temperature +
                                df_new$h_time_kernel*df_new$air_temperature,0)

df_new$mul_num <- ifelse(df_new$posterior_flag == "retain",
                            (df_new$h_distance_kernel) *
                                   (df_new$h_date_kernel) *
                                   (df_new$h_time_kernel*df_new$air_temperature),0)

df_new$additive_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_additive_dist, 0)
df_new$mul_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_mul_dist, 0)

time = unique(time_seq$time_index)
result <- NULL

for(i in time){
temp <- df_new[df_new$time_index == i,]
additive_temp <- sum(temp$additive_num)/sum(temp$additive_den)
mult_temp <- sum(temp$mul_num)/sum(temp$mul_den)

temp <- cbind(additive_temp, mult_temp, i)
result <- rbind(temp,result)
}

result <- as.data.frame(result)
result <- merge(x =result, y = time_seq, by.x = "i", by.y = "time_index", all.x = TRUE)
result$additive_temp <- as.numeric(as.character(result$additive_temp))
result$mult_temp <- as.numeric(as.character(result$mult_temp))

p1 <- ggplot(data=result, aes(x=new_date_time)) +
  geom_point(aes(y = additive_temp)) +
```

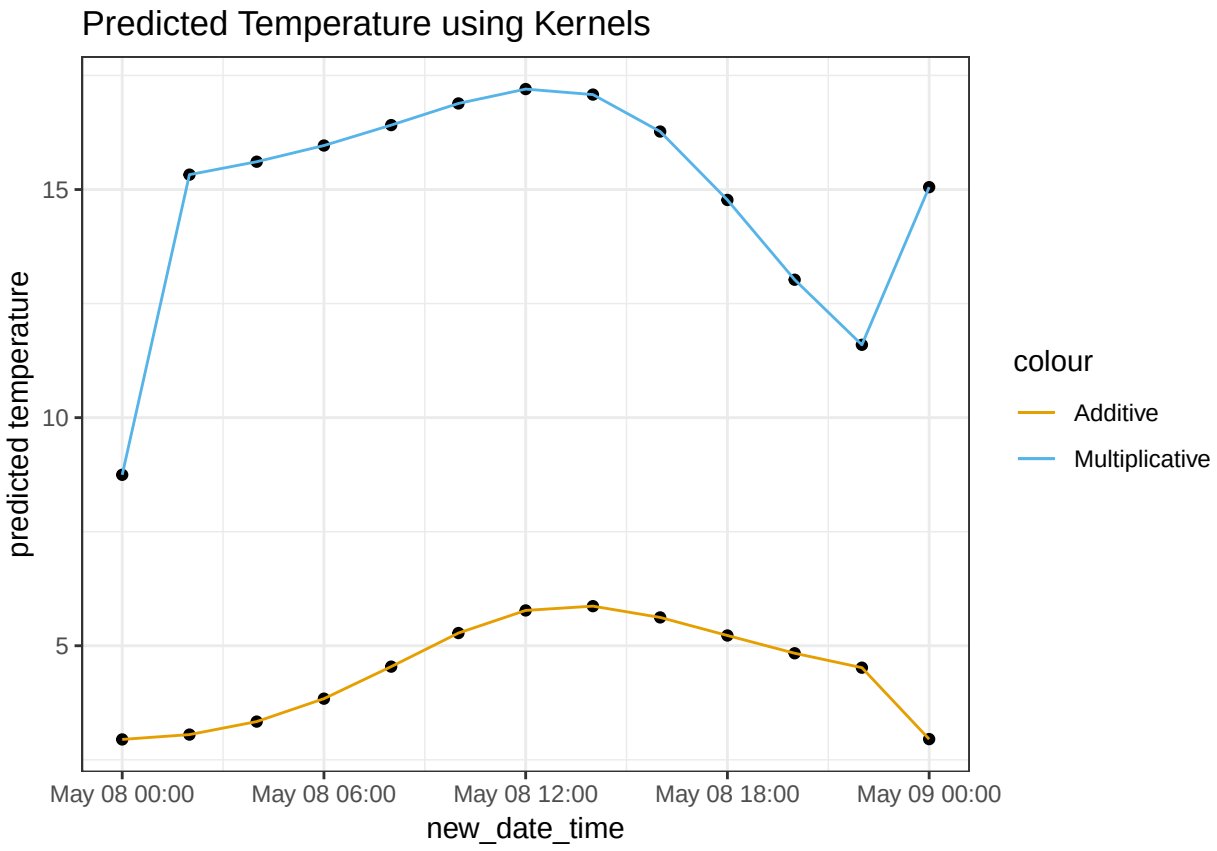
```
  geom_point(aes(y = mult_temp)) +
  geom_line(aes(y = additive_temp, color = "Additive")) +
  geom_line(aes(y = mult_temp, color = "Multiplicative")) +
  scale_color_manual(values=c("#E69F00", "#56B4E9")) +
  ylab("predicted temperature") +
  theme_bw() +
  ggtitle("Predicted Temperature using Kernels")

final <- list(p1)
return(final)
}
```

calling function

```
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 2, h3 = 5)
```

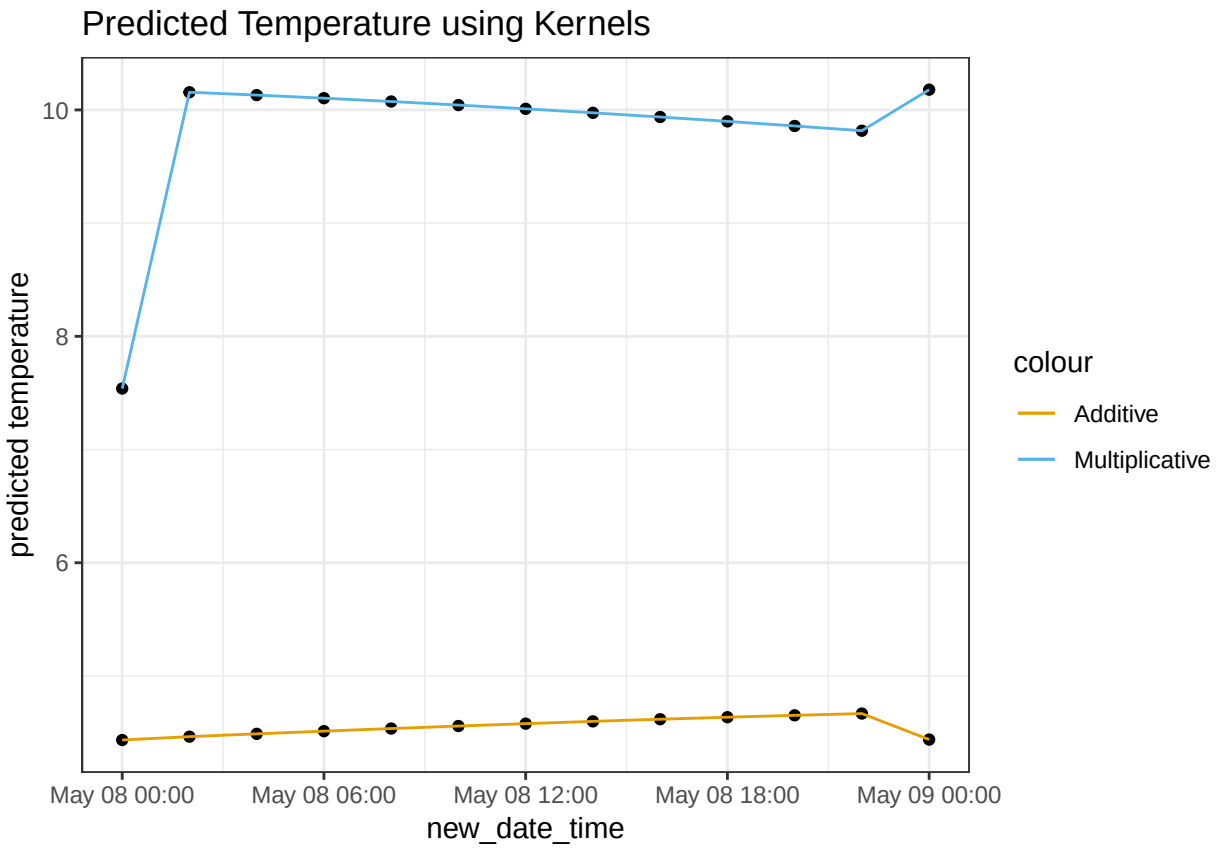
```
## [[1]]
```



high values

```
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 100, h3 = 30)
```

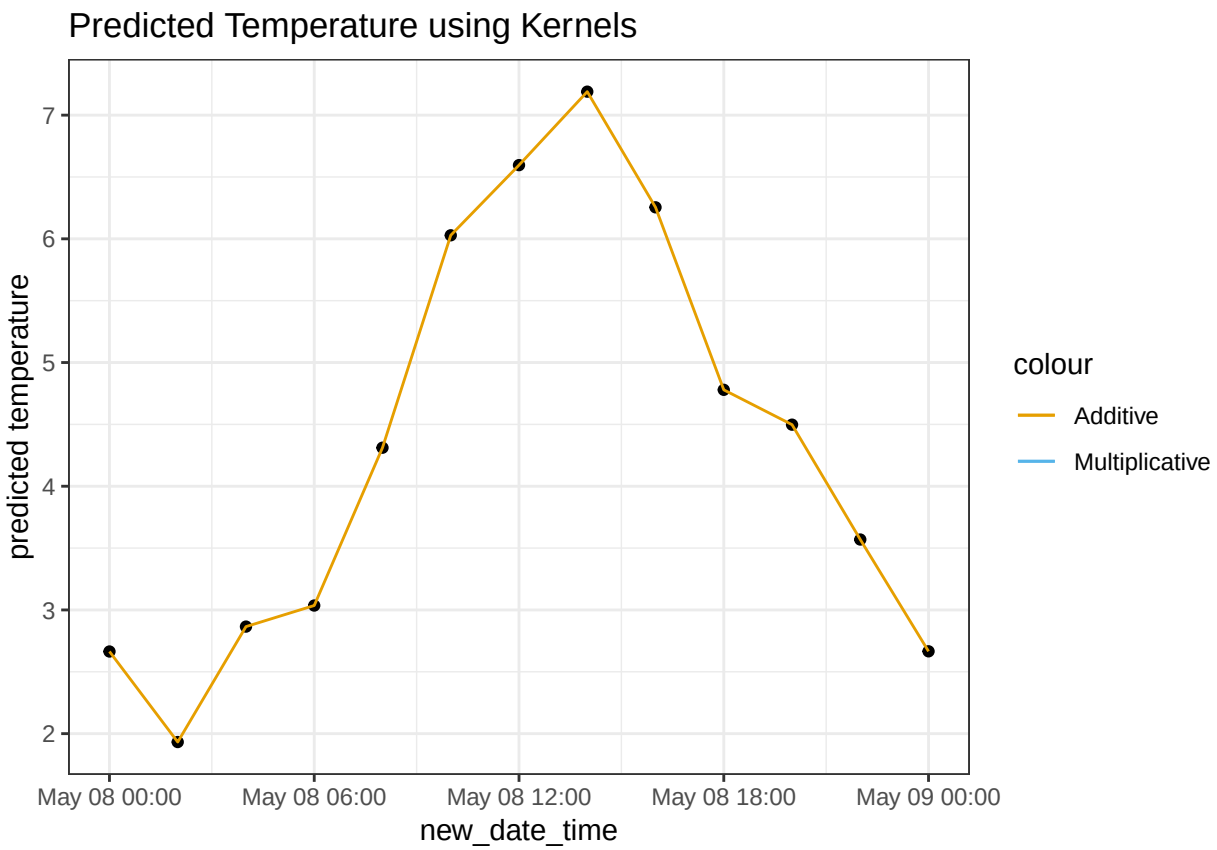
```
## [[1]]
```



low values

```
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 10, h2 = 0.05, h3 = 0.05)
```

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



Analysis:

For the kernel widths I have chosen the following values:

Distance: 30 kilometers

Date: 2 days

Time: 5 hours As evident from the plots using extreme values makes either multicative model or additive model (either terms tend to zero or all terms converge to one).When the widths are extremely large the values are saturated and there is no variations while when the values are really small the prediction by multiplicative effectively yields 0 while summation kernel predicted highly varying curve.

A good width for the distance is 30Kms, the reasoning behind this is that temperature in Linkoping and Norrkoping tend to be similar but they vary by a few degree, given that Sweden is way up north the temperature flucations will be less senstive to distance than compared to equator, thus 30Kms tend to be reasonable.

The width for the distance for day is 2, because I have personally experienced days where one days its freezing and next day I am sweating, thus 2 days is what I have choosen for my width.

For the width of time, considering the shorter winter days I do expect 3 hour of the time to be ideal window for temperature.

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

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(geosphere, kernlab, geosphere, ggplot2,caret)

set.seed(12345)
options("jtools-digits" = 2, scipen = 999)

# colours (colour blind friendly)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2",
               "#D55E00", "#CC79A7")

rm(list=ls())
set.seed(1234567890)
```

```r
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
rm(temps, stations)
kernel_method <- function(df, date, loc_long, loc_lat, h1, h2, h3) {

set.seed(1234567890)
start <- as.POSIXct(date)
interval <- 60
end <- start + as.difftime(1, units="days")
time_seq <- seq(from=start, by=interval*120, to=end)
time_seq <- as.data.frame(time_seq)
colnames(time_seq) <- "new_date_time"
time_seq$time_index <- rownames(time_seq)

df_new <- merge.data.frame(df,time_seq,all=TRUE)
rm(df)

df_new$new_date <- as.Date(df_new$new_date_time)
df_new$new_time <- format(df_new$new_date_time,"%H:%M:%S")
df_new$loc_long <- loc_long
df_new$loc_lat <-  loc_lat


df_new$h_distance <-  abs(distHaversine(p1 = df_new[,c("loc_long", "loc_lat")],
                                        p2 = df_new[,c("longitude", "latitude")]))

df_new$h_date <- as.numeric(abs(difftime(df_new$new_date, df_new$date, units = c("days"))))

df_new$h_time <- as.numeric(abs(difftime(strptime(paste(df_new$new_date,
                                                        df_new$new_time),"%Y-%m-%d%H:%M:%S"),
                                         strptime(paste(df_new$new_date, df_new$time),
                                                  "%Y-%m-%d %H:%M:%S"),
                                units = c("hour"))))


df_new$date_time <- paste(df_new$date, df_new$time)
df_new$hd_dist <- as.numeric(difftime(df_new$new_date_time,
                                      df_new$date_time,
                                      units = c("hour")))

## removing any negative dates and time
df_new$posterior_flag <- as.factor(ifelse(df_new$h_distance > 0 & df_new$hd_dist > 0, "retain", "drop"))


## calculating kernel distance and choosing guassian kernel
df_new$h_distance_kernel <- exp(-(df_new$h_distance/h1)^2)
df_new$h_date_kernel <- exp(-(df_new$h_date/h2)^2)
df_new$h_time_kernel <- exp(-(df_new$h_time/h3)^2)
df_new$total_additive_dist <- (df_new$h_distance_kernel + df_new$h_date_kernel + df_new$h_time_kernel)
df_new$total_mul_dist <- (df_new$h_distance_kernel * df_new$h_date_kernel * df_new$h_time_kernel)

df_new$additive_num <- ifelse(df_new$posterior_flag == "retain",
```

```
                              df_new$h_distance_kernel*df_new$air_temperature +
                                df_new$h_date_kernel*df_new$air_temperature +
                                df_new$h_time_kernel*df_new$air_temperature,0)

df_new$mul_num <- ifelse(df_new$posterior_flag == "retain",
                         (df_new$h_distance_kernel) *
                                (df_new$h_date_kernel) *
                                (df_new$h_time_kernel*df_new$air_temperature),0)

df_new$additive_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_additive_dist, 0)
df_new$mul_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_mul_dist, 0)

time = unique(time_seq$time_index)
result <- NULL

for(i in time){
temp <- df_new[df_new$time_index == i,]
additive_temp <- sum(temp$additive_num)/sum(temp$additive_den)
mult_temp <- sum(temp$mul_num)/sum(temp$mul_den)

temp <- cbind(additive_temp, mult_temp, i)
result <- rbind(temp,result)
}

result <- as.data.frame(result)
result <- merge(x =result, y = time_seq, by.x = "i", by.y = "time_index", all.x = TRUE)
result$additive_temp <- as.numeric(as.character(result$additive_temp))
result$mult_temp <- as.numeric(as.character(result$mult_temp))

p1 <- ggplot(data=result, aes(x=new_date_time)) +
  geom_point(aes(y = additive_temp)) +
  geom_point(aes(y = mult_temp)) +
  geom_line(aes(y = additive_temp, color = "Additive")) +
  geom_line(aes(y = mult_temp, color = "Multiplicative")) +
  scale_color_manual(values=c("#E69F00", "#56B4E9")) +
  ylab("predicted temperature") +
  theme_bw() +
  ggtitle("Predicted Temperature using Kernels")

final <- list(p1)
return(final)
}

kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 2, h3 = 5)
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 100, h3 = 30)
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 10, h2 = 0.05, h3 = 0.05)
# loading/attaching kernlab library
library(kernlab)
# importing data
data(spam)
```

```r
# 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)
```