LAB 3 BLOCK 1 - Group A15

*Zuxiang Li (**zuxli371**), Marcos F. Mourao (**marfr825**), Agustín Valencia (**aguva779**)*

17 December 2019

There are some small mistakes but really good report overall and nice explanations and analysis!

Assignment 1: Kernel Methods

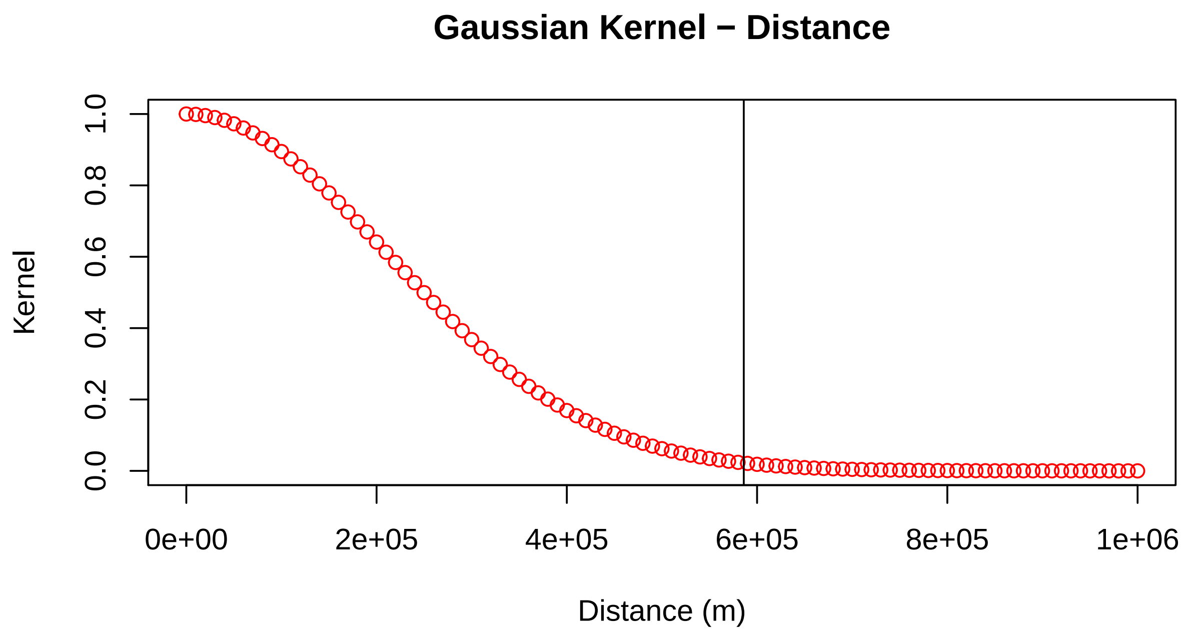
For illustration purposes, a point in Linköping (latitude: 58.41086, longitude: 15.62157) is chosen. We want to predict temperatures in said point at July 7th, 2014.

```r
#please input the coordinates and date of your desired prediction:
date <- "2014-7-11" #The date to predict
lat <- 58.41086 #Latitude of the point to predict
lon <- 15.62157 #Longitude of the point to predict

X <- c(lat,lon)
```

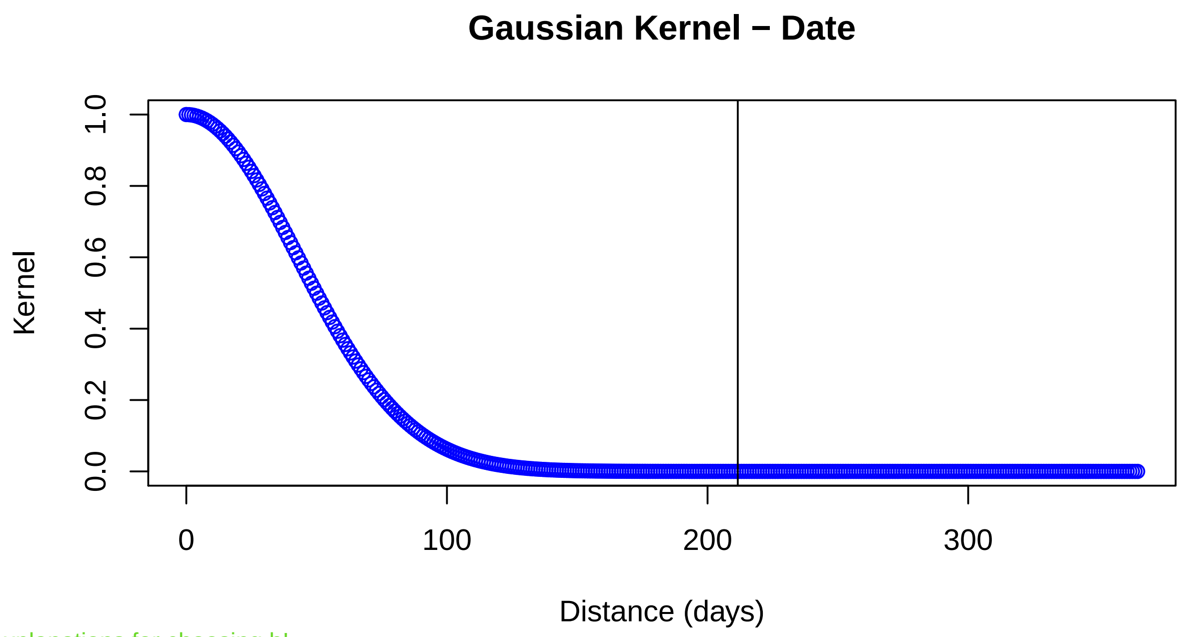
The smoothing coefficients of the kernels were chosen based on the Gaussian kernel behaviour. By using the empirical rule, it is possible to have an idea of the interval on which 95% of our data is (be it distances, days or hours).

The smoothing factor for distance, `h_distance` was set to 300000m (or 300km). It seems reasonable to think that points within an 300km radius should have high impact on the weather prediction. The black line in the plot below shows the band where 95% of distance measures lie. Distances beyond that point (~600km) have almost no impact on the predictions. Again, that seems reasonable.



As for dates (measured in days), a similar thought process was applied: the smoothing coefficient `h_date` was set to 60 days. The plot below shows that it is reasonable to think measurements taken ~100 days apart from the desired prediction have approximately zero weight in said prediction.

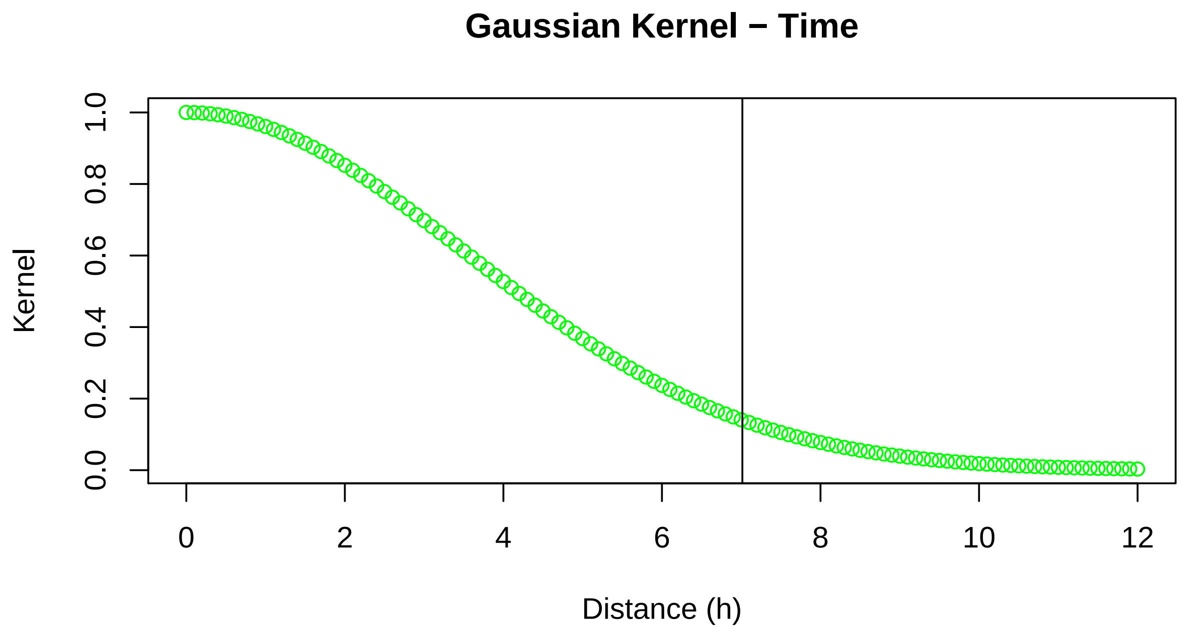
Your code does not handle the case that the distance between Dec 31 and Jan 1 is 1 day.
I think you give low weights only for days that are after the date for which you predict.



Good explanations for choosing h!

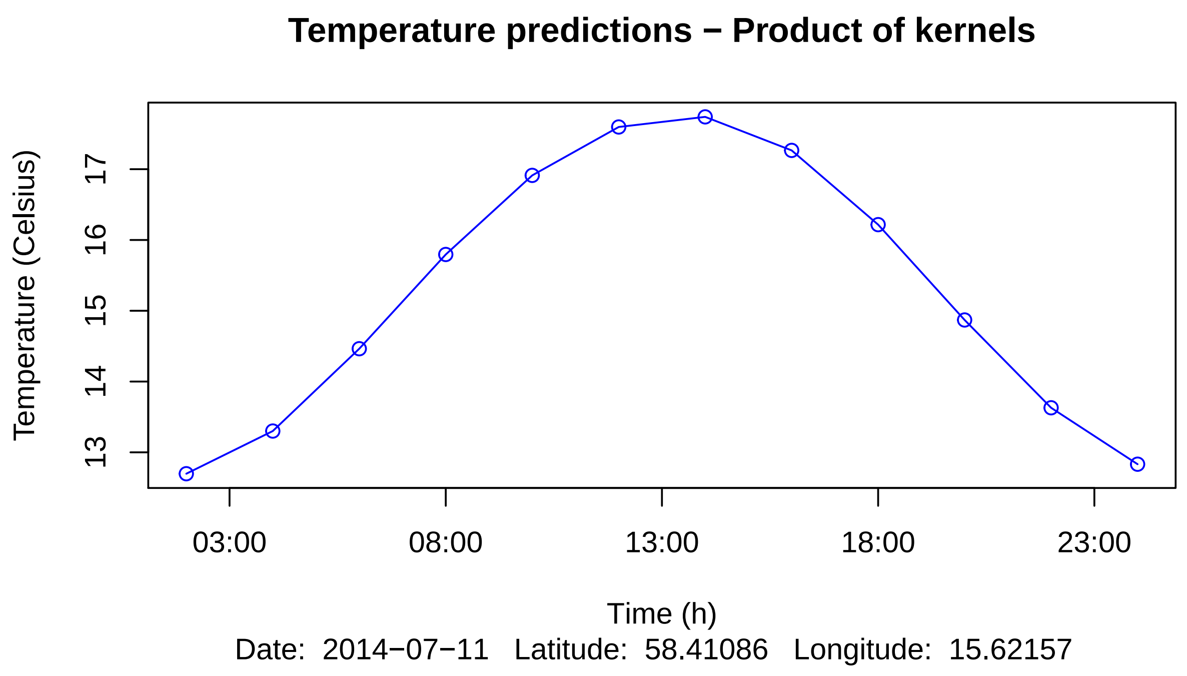
Finally, for time distances (measured in hours), the smoothing coefficient `h_time` was set to 5 hours. Similarly to what has been argued for the above cases, it seems reasonable by looking at the plot that measurements made ~7 hours apart from the desired prediction have lesser influence in said prediction.

Based on below plot, I would maybe reduce h a bit more if you want to give very low weights beyond 7 hrs.

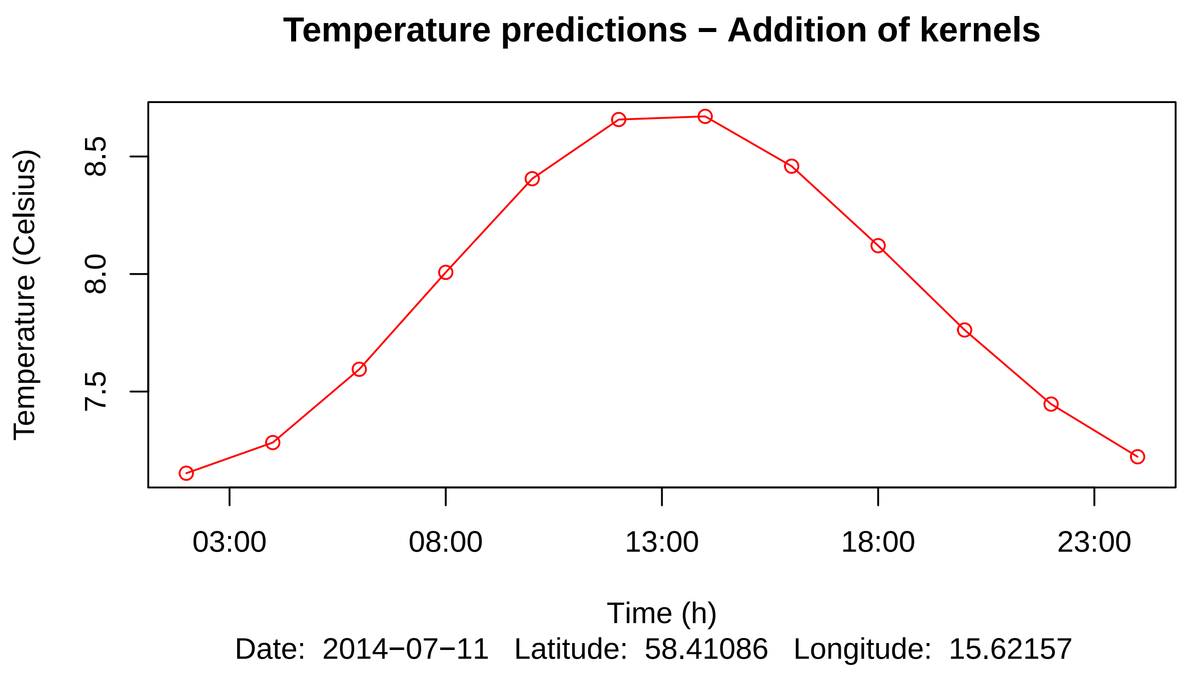


With all distances and kernels (weights) computed, it is possible to make predictions. The predictions were made using two methods: product and summation of all three kernels.

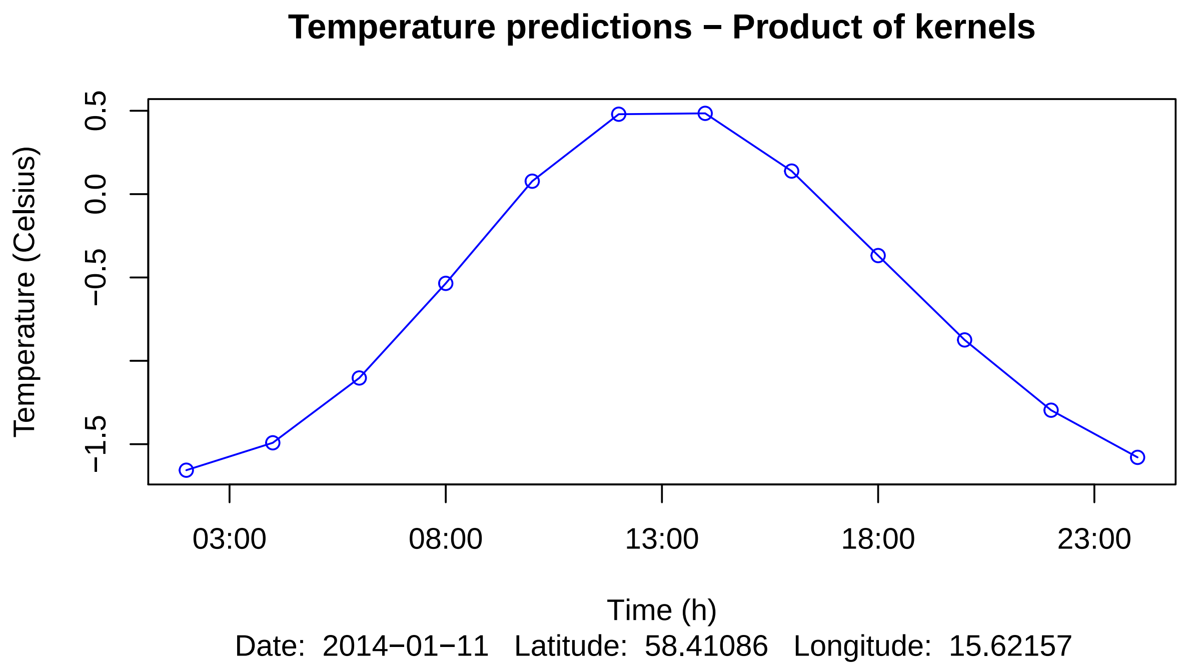
The product of kernels produce reasonable temperature predictions, as it can be seen below. The behaviour of the curve is consisten with an "usual" temperature fluctuation within a day, i.e. warmer temperatures during daytime and coler temperatures at night.

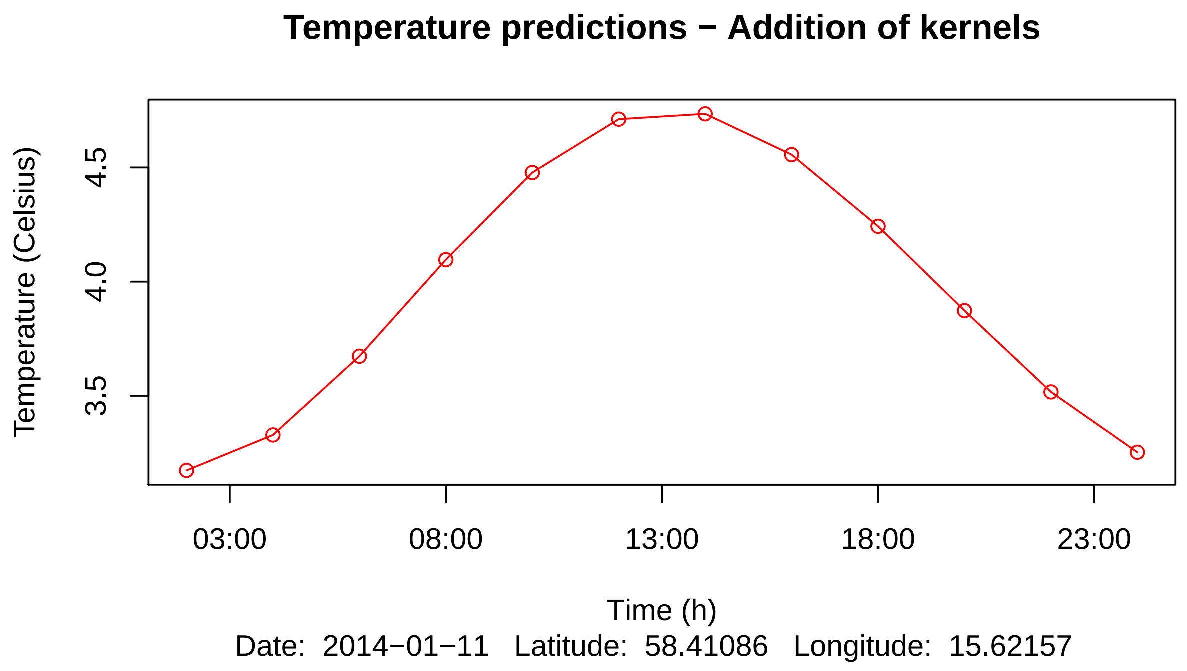


The addition of kernels produced a similar shaped curve. This curve, on the other hand is not reasonable due to the magnitude of these temperatues. Our prediction day is set in summer and intuitevely the predicted temperatures for this time of the year are too low.



The temperatures at the same location are also predicted for January 7th 2014, winter. Again, the behaviour of the predictions is reasonable: warmer temperatures during daytime than night. Also the differences in magnitude between the product and summation of kernels is seen: the product of kernels generate a more reasonable result: i.e. reasonably colder temperatures (negative) during winter.





The conclusion is that while giving some insight in temperature behaviour throught the day, the models proposed are not good: they are too simple. Location, day and hour of measurement is simply not enough to model this type of data. Temperatures throughout the years are impacted by many other factors. For example, global warming.

The different results for product and summation of kernels is due to the independency of kernels: the temperature of a given place is independent from the temperature of a given hour of the day, which is independent from the temperature in a given day of the year. It seems reasonable then to multiply each kernel's contribution to the prediction. This is analogous to the Multiplicative Law of Probability.

Yeah, that can be one way to think about it. But consider when you give reasonable weights in both the kernels

Assignment 2: Support Vector Machines

The spam dataset has been split in a ratio of 70%/30% for training and testing purposes respectively.

In order to compare the predictors performance, it have been ran experiments to get the training error, number of support vectors, true positives , false negative , false positive, false negative rates and overall misclassification from all the predictors

The code of the experiments is detailed in Appendix B and the results are summarized in the following table (rates are given in % format).

C	trainingError	numSV	TPR	TNR	FPR	FNR	Misclassification
0.5	4.41	1365	93.09	91.11	6.91	8.89	8.18
1.0	3.76	1278	91.76	92.08	8.24	7.92	8.04
5.0	1.96	1187	92.32	92.21	7.68	7.79	7.75

Taking into account that our classifier is meant to detect spam emails, it has to be noticed that, from the users perspective, loosing true email in the spambox is way worse than getting spam email in the inbox. That is measured by the False Positive Rate, for which $C = 0.5$ get the best score.

Analyzing the overall misclassification it is seen that $C = 5.0$ gets the better score, though, it also gets a higher False Positive rate than $C = 0.5$, which has been said is the most important score for this application. Thus, we drop it anyway. Nice analysis!

Regarding the amount of support vectors chosen by the model training, one might tend to think that the highest the amount of support vectors the overfitted the model, which is not false at all, though it is not always true. Having a wider soft margin might also produce more support vectors in the sense of the dynamics of *pushing* (support vectors in the right side of the hyperplane) and *pulling* (support vectors in the wrong side of the hyperplane). This can be interpreted as the highest training error for model $C = 0.5$

Therefore the chosen model to be deployed into the customer's system would be the following:

```r
svmToDeploy <- ksvm(
    type ~ .,
    data=spam,
    type="C-svc",
    kernel="rbfdot",
    C = 0.5,
    kpar = list(sigma = 0.05)
)
```

Model should be trained on train+validation data only after model selection.

Parameter C denotes the inverse of the regularization penalization. The bigger the C the smaller the soft margin between the classes to learn, thus, the more strict the learning process which could lead to overfit the data. Fortunately that is not the case for this classifier, since it gets even better scores under testing data than the more relaxed models.

Cross-validation still recommended in order to get the best results possible for the classifier, though for this assignment we are not allowed to perform that type of analisys.

Generalization error is missing!
If you are using the holdout method, then you need to split the data into train-validation-test and the test error is
is a kind of generalization error as it is calculated on unseen data.

Appendix A - Assignment 1 Code

```r
##############################-ASSIGNMENT 1-###############################
#Initial configs.
suppressWarnings(RNGversion('3.5.1'))
set.seed(1234567890)
library(geosphere)
library(readr)
library(ggplot2)
library(kernlab)
library(knitr)
#please input the coordinates and date of your desired prediction:
date <- "2014-7-11" #The date to predict
lat <- 58.41086 #Latitude of the point to predict
lon <- 15.62157 #Longitude of the point to predict

X <- c(lat,lon)
# reading data
stations <- read_csv("Data/stations.csv",
                     locale = locale(encoding = "ISO-8859-1"))
temps <- read.csv("./Data/temps50k.csv")
st <- merge(stations,temps,by="station_number")
#cleaning the data
st <- st[,c(-1,-2,-3,-6,-7,-8, -12)]
st$date <- as.Date(st$date)
st$time <- strptime(st$time, format = "%H:%M:%S")

date <- as.Date(date)
times <- c("02:00:00",
           "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")
times <- strptime(times, format = "%H:%M:%S")


#filtering out posterior dates
ind <- which(st$date > date)
st <- st[ind, ]

#-------------------------------DISTANCES-------------------------

#getting distances (in meters)
get_distance <- function() {
  distance <- distHaversine(st[,1:2], X)
  return(distance)
}
```

```r
#getting date distances (in days)
get_datedist <- function() {
  date.distance <- as.numeric(abs(difftime(time1 = st$date,
                              time2 = date,
                              units = "days")))
  date.distance <- date.distance %% 365
  return(date.distance)
}

#getting time distances (in hours)

get_timedist <- function(idx) {
  time.distance <- as.numeric(abs((difftime(time1 = st$time,
                               time2 = times[idx],
                               units = "hours"))))

  time.distance <- ifelse(time.distance<12, time.distance, 24-time.distance)

  return(time.distance)
}
#-------------SMOOTHING COEFFICIENTS-------------------------
#Distance smoothing
h_distance <- 300000
x <- seq(0,1000000, 10000)
line <- 2 * sd(x)
y <- exp(-(x/h_distance)^2)
plot(x, y, main = "Gaussian Kernel - Distance",
     xlab = "Distance (m)",
     ylab = "Kernel",
     col = "red")
abline(v = line)
#Day smoothing
h_date <- 60
x <- seq(0,365, 1)
line <- 2 * sd(x)
y <- exp(-(x/h_date)^2)
plot(x, y, main = "Gaussian Kernel - Date",
     xlab = "Distance (days)",
     ylab = "Kernel",
     col = "blue")
abline(v = line)
#Hour smoothing
h_time <- 5
x <- seq(0,12, 0.1)
line <- 2 * sd(x)
y <- exp(-(x/h_time)^2)
plot(x, y, main = "Gaussian Kernel - Time",
     xlab = "Distance (h)",
     ylab = "Kernel",
     col = "green")
abline(v = line)
#---------------KERNEL COMPUTATIONS-----------------------------------
#distance kernel
```

```r
distanceKernel <- function(){
  dist <- get_distance()
  kern <- exp(-(dist/h_distance)^2)
  return(kern)
}

#day kernel
dateKernel <- function(){
  dist <- get_datedist()
  kern <- exp(-(dist/h_date)^2)
  return(kern)
}

#hour kernel
timeKernel <- function(index){
  dist <- get_timedist(index)
  kern <- exp(-(dist/h_time)^2)
  return(kern)
}

#-------------------PREDICTIONS------------------------
#addition of kernels
predict_add <- function(index) {
  temp <- sum(st$air_temperature * (distanceKernel() + dateKernel() + timeKernel(index)))/
      (sum(distanceKernel() + dateKernel() + timeKernel(index)))
  return(temp)
}

add.temp <- vector(length = length(times))
for (i in 1:length(times)){
  add.temp[i] <- predict_add(i)
}

#product of kernels
predict_mult <- function(index) {
  temp <- sum(st$air_temperature * (distanceKernel() * dateKernel() * timeKernel(index)))/
      (sum(distanceKernel() * dateKernel() * timeKernel(index)))
  return(temp)
}

mult.temp <- vector(length = length(times))
for (i in 1:length(times)){
  mult.temp[i] <- predict_mult(i)
}
#-------------------------PLOTTING------------------------------
hour.names <- format(times, "%H:%M:%S")
plot.subtitle <- paste("Date: ", date , " ",
                       "Latitude: ", lat, " ",
                       "Longitude: ", lon, " ")

#COMBINED PLOT
# ggplot() +
#   geom_line(aes(x = hour.names, y = mult.temp, colour = "Product of kernels", group = 1)) +
#   geom_line(aes(x = hour.names, y = add.temp, colour = "Addition of kernels", group = 2)) +
#   geom_point(aes(x = hour.names, y = mult.temp, colour = "Product of kernels", group = 1)) +
#   geom_point(aes(x = hour.names, y = add.temp, colour = "Addition of kernels", group = 2)) +
#   ggtitle(label = "Temperature prediction", subtitle = plot.subtitle) +
#   xlab("Time of day (HH:MM:SS)" ) +
#   ylab ("Temperature (Celsius)") +
#   theme(axis.text.x = element_text(angle = 45))
#ORIGINAL PLOT
#mult
plot(x = times, y = mult.temp, type="o", col = "blue",
     main ="Temperature predictions - Product of kernels",
     xlab = "Time (h)", ylab = "Temperature (Celsius)",
     sub = plot.subtitle)
#add
plot(x = times, y = add.temp, type = "o", col = "red",
     main = "Temperature predictions - Addition of kernels",
     xlab = "Time (h)",ylab = "Temperature (Celsius)",
     sub = plot.subtitle)
```

Appendix B - Assignment 2 Code

```r
## Utils
splitData <- function(data, trainRate) {
    n <- dim(data)[1]
    idxs <- sample(1:n, floor(trainRate*n))
    train <- data[idxs,]
    test <- data[-idxs,]
    return (list(train = train, test = test))
}

get_performance <- function(targets, predictions, text) {
    t <- table(targets, predictions)
    tn <- t[1,1]
    tp <- t[2,2]
    fp <- t[1,2]
    fn <- t[2,1]
    total <- sum(t)
    tpr <- tp/(tp+fp) * 100
    tnr <- tn/(tn+fn) * 100
    fpr <- fp/(tp+fp) * 100
    fnr <- fn/(tn+fn) * 100

    return (
        list(
            tpr = tpr,
            tnr = tnr,
            fpr = fpr,
            fnr = fnr,
            misclass = (fp+fn)/total * 100
        )
    )
}

# Data split
data(spam)
split <- splitData(spam, .7)
train <- split$train
test  <- split$test

train.x <- train[,-ncol(train)]
train.y <- train[,ncol(train)]
test.x  <- test[,-ncol(test)]
test.y  <- test[,ncol(test)]

Cs <- c(.5, 1, 5)
kWidth <- 0.05
svmModels <- list()
svmScores <- data.frame (
    C = vector(length = 3),
    trainingError = vector(length = 3),
    numSV = vector(length = 3),
    TPR = vector(length = 3),
    TNR = vector(length = 3),
    FPR = vector(length = 3),
    FNR = vector(length = 3),
    Misclassification = vector(length = 3)
)
for (i in 1:length(Cs)) {
    svmModel <- ksvm(
                    type ~ .,
                    data=train,
                    type="C-svc",
                    kernel="rbfdot",
                    C = Cs[i],
                    kpar = list(sigma = kWidth)
              )
    predictions <- predict(svmModel, test)
    performance <- get_performance(test.y, predictions)
    svmScore <- c(
                    Cs[i],
                    error(svmModel) * 100,
                    nSV(svmModel),
                    performance$tpr,
                    performance$tnr,
                    performance$fpr,
                    performance$fnr,
                    performance$misclass
                  )
    svmScores[i,] <- svmScore
    svmModels[[i]] <- svmModel
}
kable(svmScores, digits = 2)
```