Machine Learning Lab03

Thijs Quast

17-12-2018

Contents

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")

# Determine the coordinates of the location to be predicted.
# Also, determine the gaussian distances. These are further elaborated on in the report.
latitude <- 59.9953
longitude <- 17.6935
h_distance <- 100000
h_date <- 7
h_time <- 2

# Altering the format of dates
date <- as.POSIXct("2000-05-08")
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")
```

```r
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
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))
```

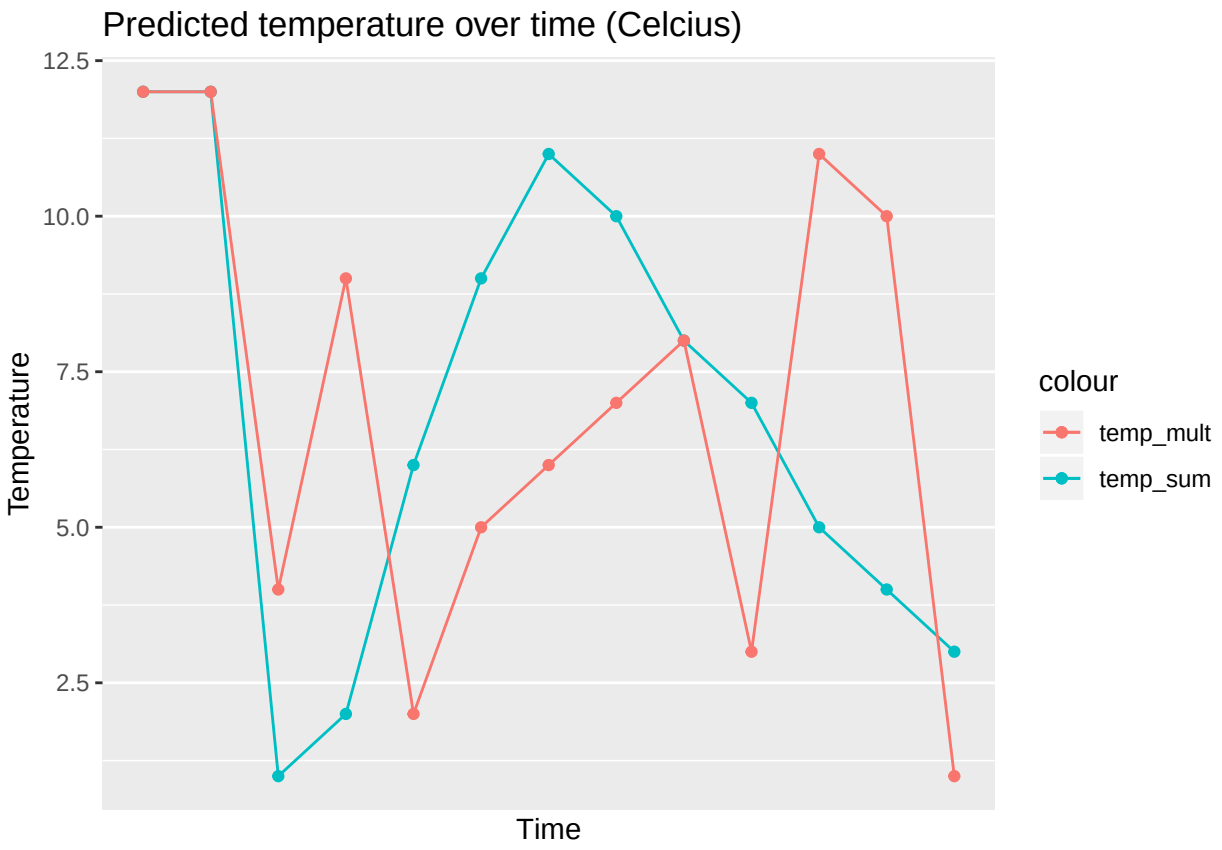
```r
outcome$temp_sum <- as.numeric(outcome$temp_sum)
outcome$temp_mult <- as.numeric(outcome$temp_mult)
outcome$count <- c(1:13)

# 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)")
plot
```



For the kernel widths I have chosen the following values:

Distance: 100 kilometers

Date: 7 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 7 days seems large, however weights decrease as distance in dates increases. Still to account for seasons, structural changes most likely appear in time intervals of weeks rather than days.

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.

Assignment 2 - Support Vector Machines

```
# Import packages and data
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
data(spam)

# Split data into training and test data. 70% of the data is training, 30% is test.
n <- dim(spam)[1]
set.seed(1234567890)
id <- sample(1:n, floor(n*0.7))
train <- spam[id,]
test <- spam[-id,]

# Estimate three different models, based on three different values for C
model1 <- ksvm(type~., data = train,
               kernel = "rbfdot",
               kpar = list(sigma = 0.05),
               C = 0.5)

model2 <- ksvm(type~., data = train,
               kernel = "rbfdot",
               kpar = list(sigma = 0.05),
               C = 1)
```

```r
model3 <- ksvm(type~., data = train,
               kernel = "rbfdot",
               kpar = list(sigma = 0.05),
               C = 5)

# Store all three models in a list
results <- list()
results$model1 <- model1
results$model2 <- model2
results$model3 <- model3

# Return a list of the three different models
results
```

```
## $model1
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1365
##
## Objective Function Value : -396.5758
## Training error : 0.044099
##
## $model2
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1278
##
## Objective Function Value : -591.1496
## Training error : 0.037578
##
## $model3
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1187
##
## Objective Function Value : -1440.594
```

```
## Training error : 0.019565
```

I would go for the second model, model3 seems to overfit, whereas model1 seems to underfit.

```
# Run predictions with each model on the test data
model1_predict <- predict(model1, newdata = test)
model2_predict <- predict(model2, newdata = test)
model3_predict <- predict(model3, newdata = test)

# Compute confusion matrices for each model on the results on the test data
conf1 <- table(test$type, model1_predict)
conf2 <- table(test$type, model2_predict)
conf3 <- table(test$type, model3_predict)

# Determine the error term on the test data for each model, based on previously computed confusion matr
model1_error <- (conf1[2,1] + conf1[1,2])/nrow(test)
model2_error <- (conf2[2,1] + conf2[1,2])/nrow(test)
model3_error <- (conf3[2,1] + conf3[1,2])/nrow(test)

# Report error terms on the test data for all three models
model1_error
```

```
## [1] 0.08182476
```

```
model2_error
```

```
## [1] 0.08037654
```

```
model3_error
```

```
## [1] 0.07748009
```

Given the results from the error term, it now makes most sense to choose model 3. This model reports the lowest error on the test data. Therefore the problem of overfitting, which was mentioned in the previous question is no longer relevant, as model3 performs fine on the unseen test data.

```
# SVM returned to user:
library(kernlab)
data(spam)

n <- dim(spam)[1]
set.seed(1234567890)
id <- sample(1:n, floor(n*0.7))
train <- spam[id,]
test <- spam[-id,]

model3 <- ksvm(type~., data = train,
               kernel = "rbfdot",
               kpar = list(sigma = 0.05),
               C = 5)

model3_predict <- predict(model3, newdata = test)
conf3 <- table(test$type, model3_predict)
model3_error <- (conf3[2,1] + conf3[1,2])/nrow(test)

final_model <- list()
final_model$model <- model3
final_model$confusion_matrix <- conf3
```

```
final_model$error <- model3_error
final_model
```

```
## $model
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1187
##
## Objective Function Value : -1440.594
## Training error : 0.019565
##
## $confusion_matrix
##          model3_predict
##           nonspam spam
##   nonspam     805   39
##   spam         68  469
##
## $error
## [1] 0.07748009
```

Most likely, in a real life scenario I would choose to report the model as shown in the chunk of code above. First I would show how the data is imported and split into training and test data. Then, the model is determined, afterwards I would provide a list with the model, confusion matrix and error on the test data. The user then has a brief but detailed overview of which model is chosen, and how it performs. The C parameter is a penalty factor for violating the contraints set up by the model.