Revision of Computer lab 1 block 3 (732A99 Machine Learning)

Lennart Schilling (lensc874)

04 January 2019

Contents

Assignment 1: Kernel methods

In the first step, the provided data will be imported and merged so that there will be only one data frame which will be used for the forecast. To perform a proper forecast, some data prepraration has to be done as well.

```
library(geosphere)
library(ggplot2)

# importing data
stations = read.csv2("stations.csv", sep = ",")
temps50k = read.csv2("temps50k.csv", sep = ",")

# merging data
data = merge(x = stations,
             y = temps50k,
             by = "station_number")

# preparing data
data$longitude = as.numeric(as.character(data$longitude))
data$latitude = as.numeric(as.character(data$latitude))
data$air_temperature = as.numeric(as.character(data$air_temperature))
```

After these steps, we can write the function which uses different parameters related to our prediction goal and returns a plot with the calculated forecast results:

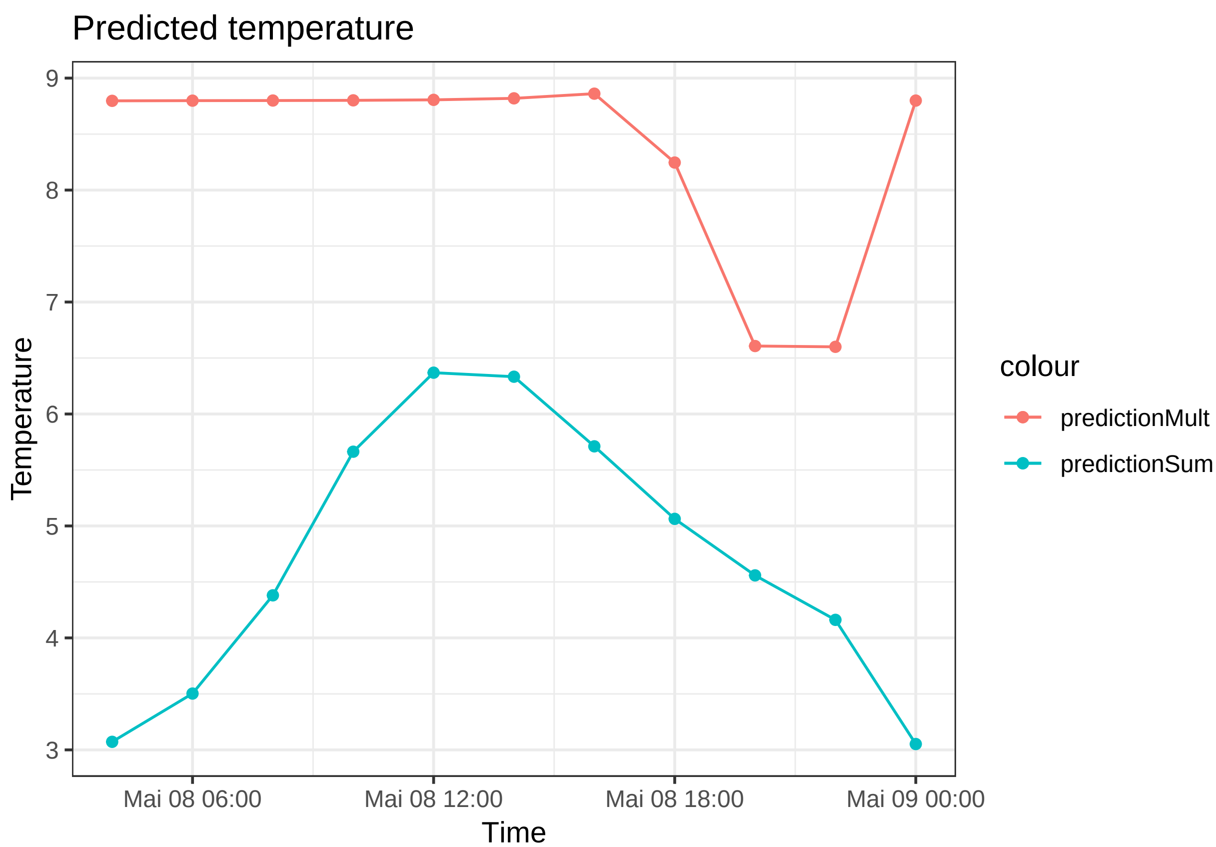
```
temperatureForecast = function(data, date, longitude, latitude, hLocationDist, hDateDist, hHourDist) {

  # identifying all forecast times
  forecastDateTime = seq(from = as.POSIXct(date),
                         to = as.POSIXct(as.Date(date) + 1),
                         by = "hour")[seq(from = 5, to = 25, by = 2)]

  # merging forecast times together with indices to data
  forecastDateTime = data.frame(forecastDateTime, forecastTimeIndex = c(1:length(forecastDateTime)))
  data = merge(data, forecastDateTime, all = TRUE)
```

```r
# adding distances between observations and forecast object to data
# location distance
data$targetLatitude = latitude
data$targetLongitude = longitude
data$locationDist = abs(as.numeric(distHaversine(p1 = data[,c("targetLongitude", "targetLatitude")],
                                      p2 = data[,c("longitude", "latitude")], r=6378137)))
# date distance
data$dateDist = as.numeric(abs(difftime(data$forecastDateTime,
                                        data$date,
                                        units = c("days"))))
# hour distance
time = gsub(":", ".", data$time)
time = as.numeric(gsub(".00", "", time))
data$hourDist = abs(lubridate::hour(data$forecastDateTime) - time)

# dropping measurements that are post target
data = data[
  !(difftime(data$forecastDateTime, paste(data$date, data$time), units = c("hour"))) < 0, ]

# calculating gaussian kernels
data$locationKernel = exp(-(data$locationDist/hLocationDist)^2)
data$dateKernel = exp(-(data$dateDist/hDateDist)^2)
data$hourKernel = exp(-(data$hourDist/hHourDist)^2)

# predicting temperature
# summation of kernels
data$summationNumerator =
  data$locationKernel * (data$air_temperature) +
  data$dateKernel * (data$air_temperature)  +
  data$hourKernel * (data$air_temperature)

data$summationDenominator = data$locationKernel + data$dateKernel + data$hourKernel

# multiplication of kernels
data$multiplicationNumerator =
  data$locationKernel * data$dateKernel * data$hourKernel * (data$air_temperature)
data$multiplicationDenominator = data$locationKernel * data$dateKernel * data$hourKernel

# getting prediction results
temp_sum = c()
temp_mult = c()
for (timeIndex in unique(data$forecastTimeIndex)) {
  subset = data[data$forecastTimeIndex == timeIndex,]
  temp_sum[timeIndex] =
    sum(subset$summationNumerator) /
    sum(subset$summationDenominator)
  temp_mult[timeIndex] =
    sum(subset$multiplicationNumerator) /
    sum(subset$multiplicationDenominator)
}

# plotting results
plotData = data.frame(dateTime = forecastDateTime$forecastDateTime,
```

```
                        predictionSum = temp_sum,
                        predictionMult = temp_mult)
  ggplot(data = plotData) +
    geom_point(aes(x = dateTime, y = predictionSum, color = "predictionSum")) +
    geom_line(aes(x = dateTime, y = predictionSum, color = "predictionSum")) +
    geom_point(aes(x = dateTime, y = predictionMult, color = "predictionMult")) +
    geom_line(aes(x = dateTime, y = predictionMult, color = "predictionMult")) +
    labs(title = "Predicted temperature", x = "Time", y = "Temperature") +
    theme_bw()
}
```

In the first step, we are using the function for a weather forecast for Linköping on the 8th May 2000. 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.

```
temperatureForecast(data = data,
                    date = "2000-05-08",
                    latitude = 58.410807,
                    longitude = 15.621373,
                    hLocationDist = 30000,
                    hDateDist = 2,
                    hHourDist = 3)
```



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

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*.

To select an approrpiate model, training data will be used to fit the model. Using the validation data, the resulting validation errors will decide about the model selection. Finally, using the model with the lowest validation error, a comparison between the classifications for the unseen test data and the actual values will lead to the generalization error. To make sure that enough observations are integrated within all three data sets, a relation of 50:25:25 will be chosen.

```
# dividing data into train, validation and test set
n = dim(spam)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = spam[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = spam[id2,]
id3 = setdiff(id1,id2)
test = spam[id3,]
```

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
minValidationError = 1
# fitting svm-models for different parameter C
for (C in c(0.5, 1, 5)) {
  svmModel = ksvm(type ~ .,
                  data = train,
                  kernel = "rbfdot",
                  kpar = list(sigma = 0.05),
```

```r
                  C = C)
  classification = predict(svmModel, valid[, -which(colnames(valid) == "type")])
  validationError = mean(valid$type != classification)
  print(paste0("Validation error for C = ", C, ": ", round(validationError, 3)))
  print(table(y = valid$type, yFit = predict(svmModel,
                                             valid[, -which(colnames(test) == "type")])),
        caption = paste0("Confusion matrix for C = ", C, ": "))
  if (validationError < minValidationError) {
    minValidationError = validationError
    bestModel = svmModel
  }
}
```

```
## [1] "Validation error for C = 0.5: 0.086"
##          yFit
## y          nonspam spam
##   nonspam     681   22
##   spam         77  370
## [1] "Validation error for C = 1: 0.07"
##          yFit
## y          nonspam spam
##   nonspam     680   23
##   spam         57  390
## [1] "Validation error for C = 5: 0.075"
##          yFit
## y          nonspam spam
##   nonspam     676   27
##   spam         59  388
```

The model with the lowest validation error (second model where $C = 1$) will be identified as the most optimal classifier.

Consindering the unseen test data, the generalization error can be calculated:

```r
classification = predict(bestModel, test[, -which(colnames(test) == "type")])
testError = mean(test$type != classification)
```

In the following, the identified best model with its errors is summarised:

```r
# returning parameter C and erros of best model
knitr::kable(
  x = as.data.frame(
    cbind(C = bestModel@param,
          trainError = round(bestModel@error, 3),
          validationError = round(minValidationError, 3),
          testError = round(testError, 3))),
  caption = "Summary of best identified svm model",
  row.names = FALSE)
```

Table 1: Summary of best identified svm model

C	trainError	validationError	testError
1	0.04	0.07	0.085

Comparing the errors, it can be clearly seen that all values are quite similar which indicates that neither too much overfitting nor underfitting seem to occur.

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.

Finally, the following model will be returned to the user:

```
# training final model with whole dataset and optimal choice of parameter C (C=1)
finalModel = ksvm(type ~ .,
                  data = spam,
                  kernel = "rbfdot",
                  kpar = list(sigma = 0.05),
                  C = 1)
```

Before, the model using $C = 1$ has been identified as the most optimal model. Using this parameter settting and the whole data *spam*, the model will be trained. This model then will be returned to the user.

Appendix

```
library(geosphere)
library(ggplot2)

# importing data
stations = read.csv2("stations.csv", sep = ",")
temps50k = read.csv2("temps50k.csv", sep = ",")

# merging data
data = merge(x = stations,
             y = temps50k,
             by = "station_number")

# preparing data
data$longitude = as.numeric(as.character(data$longitude))
data$latitude = as.numeric(as.character(data$latitude))
data$air_temperature = as.numeric(as.character(data$air_temperature))
temperatureForecast = function(data, date, longitude, latitude, hLocationDist, hDateDist, hHourDist) {

  # identifying all forecast times
  forecastDateTime = seq(from = as.POSIXct(date),
                         to = as.POSIXct(as.Date(date) + 1),
                         by = "hour")[seq(from = 5, to = 25, by = 2)]

  # merging forecast times together with indices to data
  forecastDateTime = data.frame(forecastDateTime, forecastTimeIndex = c(1:length(forecastDateTime)))
  data = merge(data, forecastDateTime, all = TRUE)

  # adding distances between observations and forecast object to data
  # location distance
  data$targetLatitude = latitude
  data$targetLongitude = longitude
  data$locationDist = abs(as.numeric(distHaversine(p1 = data[,c("targetLongitude", "targetLatitude")],
                                       p2 = data[,c("longitude", "latitude")], r=6378137)))
```

```r
# date distance
data$dateDist = as.numeric(abs(difftime(data$forecastDateTime,
                                        data$date,
                                        units = c("days"))))
# hour distance
time = gsub(":", ".", data$time)
time = as.numeric(gsub(".00", "", time))
data$hourDist = abs(lubridate::hour(data$forecastDateTime) - time)

# dropping measurements that are post target
data = data[
  !(difftime(data$forecastDateTime, paste(data$date, data$time), units = c("hour"))) < 0, ]

# calculating gaussian kernels
data$locationKernel = exp(-(data$locationDist/hLocationDist)^2)
data$dateKernel = exp(-(data$dateDist/hDateDist)^2)
data$hourKernel = exp(-(data$hourDist/hHourDist)^2)

# predicting temperature
# summation of kernels
data$summationNumerator =
  data$locationKernel * (data$air_temperature) +
  data$dateKernel * (data$air_temperature)  +
  data$hourKernel * (data$air_temperature)

data$summationDenominator = data$locationKernel + data$dateKernel + data$hourKernel

# multiplication of kernels
data$multiplicationNumerator =
  data$locationKernel * data$dateKernel * data$hourKernel * (data$air_temperature)
data$multiplicationDenominator = data$locationKernel * data$dateKernel * data$hourKernel

# getting prediction results
temp_sum = c()
temp_mult = c()
for (timeIndex in unique(data$forecastTimeIndex)) {
  subset = data[data$forecastTimeIndex == timeIndex,]
  temp_sum[timeIndex] =
    sum(subset$summationNumerator) /
    sum(subset$summationDenominator)
  temp_mult[timeIndex] =
    sum(subset$multiplicationNumerator) /
    sum(subset$multiplicationDenominator)
}

# plotting results
plotData = data.frame(dateTime = forecastDateTime$forecastDateTime,
                      predictionSum = temp_sum,
                      predictionMult = temp_mult)
ggplot(data = plotData) +
  geom_point(aes(x = dateTime, y = predictionSum, color = "predictionSum")) +
  geom_line(aes(x = dateTime, y = predictionSum, color = "predictionSum")) +
  geom_point(aes(x = dateTime, y = predictionMult, color = "predictionMult")) +
```

```r
    geom_line(aes(x = dateTime, y = predictionMult, color = "predictionMult")) +
    labs(title = "Predicted temperature", x = "Time", y = "Temperature") +
    theme_bw()
}
temperatureForecast(data = data,
                    date = "2000-05-08",
                    latitude = 58.410807,
                    longitude = 15.621373,
                    hLocationDist = 30000,
                    hDateDist = 2,
                    hHourDist = 3)
# loading/attaching kernlab library
library(kernlab)
# importing data
data(spam)
# printing nrow & ncol
dim(spam)
# dividing data into train, validation and test set
n = dim(spam)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = spam[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = spam[id2,]
id3 = setdiff(id1,id2)
test = spam[id3,]
# setting up minimum misclassification rate to 1 for following loop
minValidationError = 1
# fitting svm-models for different parameter C
for (C in c(0.5, 1, 5)) {
  svmModel = ksvm(type ~ .,
                  data = train,
                  kernel = "rbfdot",
                  kpar = list(sigma = 0.05),
                  C = C)
  classification = predict(svmModel, valid[, -which(colnames(valid) == "type")])
  validationError = mean(valid$type != classification)
  print(paste0("Validation error for C = ", C, ": ", round(validationError, 3)))
  print(table(y = valid$type, yFit = predict(svmModel,
                                              valid[, -which(colnames(test) == "type")])),
        caption = paste0("Confusion matrix for C = ", C, ": "))
  if (validationError < minValidationError) {
    minValidationError = validationError
    bestModel = svmModel
  }
}
classification = predict(bestModel, test[, -which(colnames(test) == "type")])
testError = mean(test$type != classification)
# returning parameter C and erros of best model
knitr::kable(
  x = as.data.frame(
```

```r
    cbind(C = bestModel@param,
          trainError = round(bestModel@error, 3),
          validationError = round(minValidationError, 3),
          testError = round(testError, 3))),
  caption = "Summary of best identified svm model",
  row.names = FALSE)
# training final model with whole dataset and optimal choice of parameter C (C=1)
finalModel = ksvm(type ~ .,
                data = spam,
                kernel = "rbfdot",
                kpar = list(sigma = 0.05),
                C = 1)
```