Block 1 Lab 3

Mim Kemal Tekin

12/13/2018

Contents

Assignment 1: Kernel Methods

Task 1.1

In this assignment we create a forecast model which uses kernel methods. We have three different gaussian kernels which are based on:
- distances between stations
- day counts between the which that a temperature measurement was made, day of interest
- distances between the hour of the day which a temperature measurement was made, hour of interest

As width of kernels it is reasonable to think about some periods of interests of each. For example for distances, we can pick the logical radius that have similar climates; for day differences, we can pick a width around in same season or closer values; for time, we can pick values which separates the day well by temperature differences. We can test some observations which their temperatures are already known and we can find optimal reasonable values. In this case we set some random reasonable values because of their ranges, mostly about climate, season and daytimes as following:
- $h_{\text{dist}} = (0.1, 0.2, 0.3, 0.5)$
which are x1000 kilometers and examine quite similar climates
- $h_{\text{date}} = (20, 30, 40, 50, 100)$
which are day counts for some terms. It makes sense take between 30-50 in order to capture a season effect
- $h_{\text{date}} = (3, 4, 5, 8)$
which are short terms in a day and it make sense to take it around 4 - 5, in order to able to interpret temperature changes during day

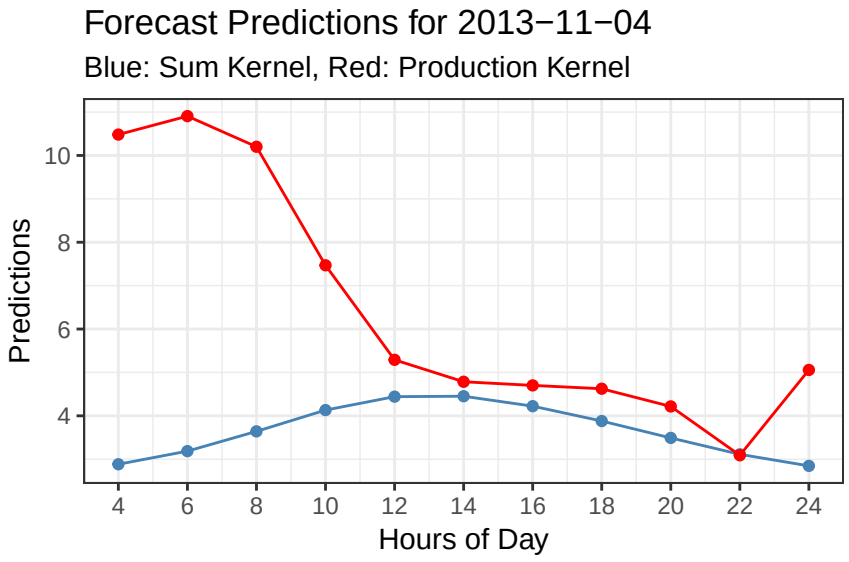
We create a grid for these values in order to test each combination. By this way, we can get more sensible widths and also supported with statistical results. And after creating sum and production kernels for 50 different input test observation, we calculate MSE for these combinations which are listed below ordered by MSE errors with sum kernel. We choose the widths by looking to sum kernel, also it is not changing that much for production kernel.

We selected weights as following:
- $h_{\text{dist}} = 0.5$
- $h_{\text{date}} = 40$
- $h_{\text{date}} = 3$)

After selecting proper widths, we create gaussian kernels by using these widths and predict forecast for "2013-11-04", location (lon:14.826, lat:58.4274) and for daytimes from 04:00 to 24:00 by 2 hours. The results which are predicted with sum of and product of these 3 different gaussian kernel are at following plot. If we examine this plot, sum kernel is more sensible than the product kernel because it has a good distribution in daytime and it is expected to see around 3-4.5 degrees in October in Sweden. But product kernel is following a strange pattern and more higher temperatures than sum kernel. The only similar thing is the decreasement movement between 14 and 22 and the results are same at 22, after that it increases instantly again.

Table 1: MSE ordered by MSE of sum kernel (25 rows)

Var1	Var2	Var3	mse_sumK	mse_prodK
0.5	40	3	54.33083	39.72583
0.3	40	3	54.33135	54.49998
0.2	40	3	54.33211	54.51279
0.1	40	3	54.33363	54.55207
0.5	50	3	54.92026	52.78648
0.3	50	3	54.92075	55.92969
0.2	50	3	54.92148	55.95102
0.1	50	3	54.92297	56.02465
0.5	30	3	55.83737	38.39372
0.3	30	3	55.83795	54.61836
0.2	30	3	55.83884	54.61847
0.1	30	3	55.84069	54.61197
0.5	40	4	58.35383	49.36553
0.3	40	4	58.35434	52.21114
0.2	40	4	58.35514	52.22256
0.1	40	4	58.35695	52.26015
0.5	50	4	58.52393	53.13715
0.3	50	4	58.52441	53.28560
0.2	50	4	58.52516	53.30503
0.1	50	4	58.52687	53.37324
0.5	30	4	60.18373	36.45313
0.3	30	4	60.18432	52.65335
0.2	30	4	60.18524	52.65283
0.1	30	4	60.18744	52.64748
0.5	20	3	60.49365	40.43125



Assignment 2: Support Vector Machines

Task 2.1

```
################################### TASK 2.1 ###################################

library(kernlab)

data(spam)
df_spam = spam
target_index = which(colnames(df_spam)=="type")
n = dim(df_spam)[1]

# split data
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = df_spam[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = df_spam[id2,]
id3 = setdiff(id1,id2)
test = df_spam[id3,]

real_train = train[,target_index]
real_valid = valid[,target_index]
real_test = test[,target_index]


# C parameter
C = c(0.5, 1, 5)

models = list()
rates_train = c()
rates_valid = c()
for(i in C){
  # fit model with C parameter
  curr_model = ksvm(type~., data=train, kernel = "rbfdot", kpar = list(sigma=0.05), C=i)
  models[[paste("svm_",i, sep = "")]] = curr_model

  # predict
  pred_train = predict(curr_model, train)
  pred_valid = predict(curr_model, valid)

  # confusion matrix
  cm_train = conf_matrix(real_train, pred_train, levels=c("nonspam", "spam"))
  cm_valid = conf_matrix(real_valid, pred_valid, levels=c("nonspam", "spam"))

  # misclassification rates
  mis_train = calculate_rate(cm_train)
  mis_valid = calculate_rate(cm_valid)
  rates_train = c(rates_train, mis_train)
  rates_valid = c(rates_valid, mis_valid)
```

```
  models[[paste("svm_",i, sep = "")]] = curr_model
}

df_table = data.frame(C = C,
                      Train = rates_train,
                      Validation = rates_valid)

kableExtra::kable(df_table, "latex", booktabs = T, align = "c",
                  caption = "Misclassification Rates") %>%
                  kable_styling(latex_options = "hold_position")
```

Table 2: Misclassification Rates

C	Train	Validation
0.5	0.0560870	0.0860870
1.0	0.0400000	0.0695652
5.0	0.0213043	0.0747826

If we compare the values in table above, the best model is with C=1. We will use this model for next tasks.

Task 2.2

```
################################### TASK 2.2 ###################################

# we pick the model which has C=1

best_svm = models$svm_1

preds = predict(best_svm, newdata = test)

cm = conf_matrix(preds, real_test, levels=c("nonspam", "spam"))

gen_error = calculate_rate(cm)

print(paste("Generalization Error =", gen_error))
```

```
## [1] "Generalization Error = 0.0851433536055604"
```

Task 2.3

```
################################### TASK 2.3 ###################################

# Summary of best svm model
print(best_svm)
```

```
## 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 : 1007
##
## Objective Function Value : -467.7423
## Training error : 0.04
```

Task 2.4

In this assignment, we have done SVM classification with testing different C parameters by using holdout method. We divide the data to train, test and validation sets. After training data for each C parameters, we calculated misclassification rates with validation data in order to find best parameter. Results in Task 2.1 showed us, the best model according to this holdout process is the model with C=1. We chose this model for the other tasks.

C parameter provides to prepare your model to unknown future data. It sets the margin between observations and the decision boundry. High C values will be classify more accurately than low C values, because high C values will put boundary giving less margins to points. So our model will separate classes more sensitively, but this can be a problem for future observations. Because probably boundry will be closer to a class more than the other one and if we get more observations from the closer class, the probability of being outside of boundary will be higher. But if we keep C parameter too low, then it will lose accuracy, and misclassification rate will increase. So it is important to choose optimal C value in order to keep first misclassification in a good level and also cover unknown future observations.

Appendix

```
knitr::opts_chunk$set(echo = FALSE, fig.width = 4.5, fig.height = 3,
                      fig.align = "center",
                      warning = F, error = F, message = F)

library(ggplot2)
library(kableExtra)

conf_matrix = function(real_data, predicted_data, levels){
  # make the values factor
  real_data = factor(real_data, levels=levels)
  predicted_data = factor(predicted_data, levels = levels(real_data))
  # we can have "not predicted" values in predicted data
  # make equal the levels of predicted values with real data
  levels(predicted_data) = levels(real_data)

  ct = table(real_data, predicted_data)
  df = data.frame(c(as.vector(ct[,1]), sum(ct[,1])),
                  c(as.vector(ct[,2]), sum(ct[,2])),
                  c(sum(ct[1,]), sum(ct[2,]), sum(ct)))
  rownames(df) = c(levels, "Frequencies")
  colnames(df) = c(levels, "Frequencies")
  return(df)
}
```

```r
calculate_rate = function(conf_matrix){
  conf_matrix = as.matrix(conf_matrix)
  return(1 - sum(diag(conf_matrix[1:2,1:2]))/sum(conf_matrix[1:2,1:2]))
}

################################### TASK 1.1 ###################################

gaussian_kernel = function(norm, weight){
  return(exp(-norm^2/(2*weight^2)))
}

sum_kernel = function(...){
  args = list(...)
  for(i in 2:length(args))
    args[[1]] = args[[1]] + args[[i]]
  return(args[[1]])
}

prod_kernel = function(...){
  args = list(...)
  for(i in 2:length(args))
    args[[1]] = args[[1]] * args[[i]]
  return(args[[1]])
}

# coordinate distance function
# a = c(lon_1, lat_1)
# b = c(lon_2, lat_2)
# returns: coordinate distance between 2 points as kilometers
coor_dist = function(a,b){
  if(!is.numeric(a) || !is.numeric(b))
    stop("a and b inputs should be numeric type!")
  if(length(a)!=2 || length(b)!=2)
    stop("a and b inputs should have size 2!")
  dis = geosphere::distHaversine(a,b)
  return(dis/1000)
}

# day counter function between 2 date
# date1 and date2: yyyy-mm-dd
# returns positive day count between 2 date
days_2_date = function(date1, date2){
  y = 2000
  date1 = sub("\\d{4}", y, as.character(date1))
  date2 = sub("\\d{4}", y, as.character(date2))
  day_count = abs(as.numeric(difftime(date1, date2)))
  return(min(day_count, 365-day_count))
}

# hour counter function between 2 time
# time1 and time2: hh:mm:ss
# returns positive hours between 2 time
hours_2_time = function(time1, time2){
```

```
  time1 = as.character(time1)
  time2 = as.character(time2)
  hours = as.difftime(c(time1, time2), units = "hours")
  diff = abs(hours[1]-hours[2])
  return(as.numeric(min(diff, 24-diff)))
}

predict_kernel = function(kernel, real_data){
  pred = sum(kernel*real_data) / sum(kernel)
  return(pred)
}

# predict forecest function
# predicts air tempreture for a day by using sum of 3 kernel
# df_st: train dataset
# input: is a list which is for the day to predict
#     list( loc = c(lat, lon),
#           date = "yyyy-mm-dd",
#           time = "hh:mm:ss")
# weights: is a vector which has weights for 3 kernels. 0.5 is default for each.
#     c(h_distance, h_date, h_time)
# method: is a parameter for final kernel calculation.
#     "S" for sum of all 3 kernels (default)
#     "P" for product of all 3 kernels
forecast = function(df_st, input, weights=c(0.5,0.5,0.5), method="S"){
  # eliminate the days after the input day
  indexes = which(as.Date(input$date) > as.Date(df_st$date))
  df_st = df_st[indexes, ]
  # get real target values
  y_real = df_st$air_temperature
  # set h values ( weights of kernels )
  h_distance = weights[1]
  h_date = weights[2]
  h_time = weights[3]
  # the point to predict
  x_location = input$loc
  # the date to predict
  x_date = input$date
  # the times to predict
  x_times = input$time

  # calculate location distances
  message(paste(x_times,"- location calculations."))
  loc_dist = apply(df_st[,c("longitude", "latitude")], MARGIN = 1,
                   FUN = coor_dist, b = x_location)
  # calculate date distances
  message(paste(x_times,"- date calculations."))
  date_dist = apply(as.data.frame(df_st[,c("date")]), MARGIN = 1,
                    FUN = days_2_date, date2 = x_date)
  # calculate time distances
  message(paste(x_times,"- time calculations."))
  time_dist = apply(as.data.frame(df_st[,c("time")]), MARGIN = 1,
                    FUN = hours_2_time, time2 = x_times)
```

```
  # calculate kernels
  loc_kernel = gaussian_kernel(norm = loc_dist, h_distance)
  date_kernel = gaussian_kernel(norm = date_dist, h_date)
  time_kernel = gaussian_kernel(norm = time_dist, h_time)
  forecast_kernel = NA
  # calculate forecast kernel
  if(method=="S")
    forecast_kernel = loc_kernel + date_kernel + time_kernel
  else if(method=="P")
    forecast_kernel = loc_kernel * date_kernel * time_kernel
  # predict
  prediction = predict_kernel(forecast_kernel, y_real)
  head(forecast_kernel)
  return(prediction)
}

# function that tests different width parameters for kernels
# df_st: dataset
# test_count: count of observations which will test
# widths: list which has the different width values for kernels
#     list( w_dis = c(numeric()),
#           w_date = c(numeric()),
#           w_time = "c(numeric()))
# returns a dataframe which has all combinations of widths and mse for tests
test_widths = function(df_st, test_count=100, widths){
  # create grid to search
  grids = expand.grid(list(widths$w_dis, widths$w_date, widths$w_time))
  # create some test cases from the data
  set.seed(12345)
  id = sample(1:n, test_count)
  test_data = df_st[id,]
  df_st = df_st[-id, ]
  # create another grid to store mse values and set them 0
  # skipped for skipped observations
  g = grids
  g$mse_sumK = 0
  g$mse_prodK = 0
  skipped = 0
  # main iteration to traverse test observations
  for(j in 1:nrow(test_data)){
    message(paste("##############################",j,"test"))
    # get test data and create input object
    test = test_data[j, ]
    input = list(loc = c(test$longitude, test$latitude),
             date = test$date,
             time = test$time)

    # eliminate the days after the input day
    tmp_df = df_st[which(as.Date(input$date) > as.Date(df_st$date)), ]
    y_real = tmp_df$air_temperature

    # calculate location distances
    message(paste(j,"- location calculations."))
```

```
    loc_dist = apply(tmp_df[,c("longitude", "latitude")], MARGIN = 1,
                     FUN = coor_dist, b = input$loc)
    # calculate date distances
    message(paste(j,"- date calculations."))
    date_dist = apply(as.data.frame(tmp_df[,c("date")]), MARGIN = 1,
                      FUN = days_2_date, date2 = input$date)
    # calculate time distances
    message(paste(j,"- time calculations."))
    time_dist = apply(as.data.frame(tmp_df[,c("time")]), MARGIN = 1,
                      FUN = hours_2_time, time2 = input$time)
    # htest prediction temp variables
    h_test_preds_sum = c()
    h_test_preds_prod = c()
    for(i in 1:nrow(grids)){
      # print(paste(i,"grid"))
      # get current widths from grid
      h_vector = as.numeric(grids[i,])
      # calculate kernels
      loc_kernel = gaussian_kernel(norm = loc_dist, h_vector[1])
      date_kernel = gaussian_kernel(norm = date_dist, h_vector[2])
      time_kernel = gaussian_kernel(norm = time_dist, h_vector[3])
      # create sum kernel and product kernel
      sum_kernel = loc_kernel + date_kernel + time_kernel
      prod_kernel = loc_kernel * date_kernel * time_kernel
      # predictions
      pred_sum = predict_kernel(sum_kernel, y_real)
      pred_prod = predict_kernel(prod_kernel, y_real)
      # store predictions
      h_test_preds_sum[i] = pred_sum
      h_test_preds_prod[i] = pred_prod
    }
    # check if predictions has some NaN values, if they have skip this observation
    # because when you try integer(10)+NaN operation, it will be NaN
    if(NaN %in% h_test_preds_prod || NaN %in% h_test_preds_sum){
      message(paste("!!!",j,"skipped !!!"))
      skipped = skipped + 1
      next
    }
    # get real temperature from test
    yyy = rep(test$air_temperature, length(pred_sum))
    # calculate squared residuals and add the vector to our grid
    res_sum = (yyy - h_test_preds_sum)^2
    res_prod = (yyy - h_test_preds_prod)^2
    g$mse_sumK = g$mse_sumK + res_sum
    g$mse_prodK = g$mse_prodK + res_prod
  }
  # get mean of the squared residuals
  g$mse_sumK = g$mse_sumK/(nrow(test_data)-skipped)
  g$mse_prodK = g$mse_prodK/(nrow(test_data)-skipped)

  return(g)
}
```

```r
set.seed(1234567890)
df_stations = read.csv("../dataset/stations.csv", fileEncoding="ISO-8859-1")
df_temps = read.csv("../dataset/temps50k.csv")
df_st = merge(df_stations,df_temps,by="station_number")
n = dim(df_st)[1]
##### sample input
# the point to predict c(long,lat)
x_location = c(14.826, 58.4274)
# the date to predict
x_date = "2013-11-04"
# the times to predict
x_times = paste(02:12*2, "00","00", sep = ":")


############# WIDTH TEST ###########################################################
# implement test width (h) values
hs_distance = c(100/1000, 200/1000, 300/1000, 500/1000)
hs_date =     c(20, 30, 40, 50, 100)
hs_time =     c(3, 4, 5, 8)

ws = list(w_dis = hs_distance,
          w_date = hs_date,
          w_time = hs_time)

res = test_widths(df_st, widths = ws, test_count = 50)

# min sum mse order
o_min_sum = res[order(res$mse_sumK),]
row.names(o_min_sum) = 1:nrow(o_min_sum)
# min prod mse order
o_min_prod = res[order(res$mse_prodK),]
# min mean sum and prod mse order
o_mean_sum_prod = res[order((res$mse_prodK + res$mse_sumK)/2),]
####################################################################################


opt_w = as.numeric(o_min_sum[1, c(1,2,3)])

kableExtra::kable(o_min_sum[1:25,], "latex", booktabs = T,
                  caption = "MSE ordered by MSE of sum kernel (25 rows)") %>%
                    kable_styling(latex_options = "hold_position")
# kableExtra::kable(o_min_sum, "latex", booktabs = T, align = "c",
#                     caption = "MSE ordered by sum MSE") %>%
#                     kable_styling(latex_options = "hold_position")
################################## NORMAL CODE #####################################
# weights of kernals (optimal)
h_distance = opt_w[1]     #300/1000
h_date = opt_w[2]         #40
h_time = opt_w[3]         #3
#
# h_distance = 0.3     #300/1000
# h_date = 40         #40
```

```r
# h_time = 3          #3

w = c(h_distance, h_date, h_time)

preds_sum = c()
preds_prod = c()
for(i in 1:length(x_times)){
  # creating input object
  input = list(loc = x_location,
               date = x_date,
               time = x_times[i])
  message(paste("### sum forecast ", i, "###"))
  preds_sum[i] = forecast(df_st, input, w, method = "S")
  message(paste("### product forecast ", i, "###"))
  preds_prod[i] = forecast(df_st, input, w, method = "P")
}

ggplot() +
  geom_point(aes(x=2:12*2, y=preds_sum), color="steelblue") +
  geom_line(aes(x=2:12*2, y=preds_sum), color="steelblue") +
  geom_point(aes(x=2:12*2, y=preds_prod), color="red") +
  geom_line(aes(x=2:12*2, y=preds_prod), color="red") +
  scale_x_continuous(breaks = seq(2,24,2)) +
  theme_bw() +
  labs(title = "Forecast Predictions for 2013-11-04",
       subtitle = "Blue: Sum Kernel, Red: Production Kernel",
       x = "Hours of Day", y = "Predictions")


################################################################################

################################### TASK 2.1 ###################################

library(kernlab)

data(spam)
df_spam = spam
target_index = which(colnames(df_spam)=="type")
n = dim(df_spam)[1]

# split data
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = df_spam[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = df_spam[id2,]
id3 = setdiff(id1,id2)
test = df_spam[id3,]

real_train = train[,target_index]
real_valid = valid[,target_index]
```

```
real_test = test[,target_index]


# C parameter
C = c(0.5, 1, 5)

models = list()
rates_train = c()
rates_valid = c()
for(i in C){
  # fit model with C parameter
  curr_model = ksvm(type~., data=train, kernel = "rbfdot", kpar = list(sigma=0.05), C=i)
  models[[paste("svm_",i, sep = "")]] = curr_model

  # predict
  pred_train = predict(curr_model, train)
  pred_valid = predict(curr_model, valid)

  # confusion matrix
  cm_train = conf_matrix(real_train, pred_train, levels=c("nonspam", "spam"))
  cm_valid = conf_matrix(real_valid, pred_valid, levels=c("nonspam", "spam"))

  # misclassification rates
  mis_train = calculate_rate(cm_train)
  mis_valid = calculate_rate(cm_valid)
  rates_train = c(rates_train, mis_train)
  rates_valid = c(rates_valid, mis_valid)
  models[[paste("svm_",i, sep = "")]] = curr_model
}

df_table = data.frame(C = C,
                      Train = rates_train,
                      Validation = rates_valid)

kableExtra::kable(df_table, "latex", booktabs = T, align = "c",
                  caption = "Misclassification Rates") %>%
                  kable_styling(latex_options = "hold_position")


################################### TASK 2.2 ###################################

# we pick the model which has C=1

best_svm = models$svm_1

preds = predict(best_svm, newdata = test)

cm = conf_matrix(preds, real_test, levels=c("nonspam", "spam"))

gen_error = calculate_rate(cm)

print(paste("Generalization Error =", gen_error))
```

```
################################### TASK 2.3 ###################################

# Summary of best svm model
print(best_svm)
```