BDA3 - Machine Learning with Spark - Exercises

Stefano Toffol (steto820) and Nahid Farazmand (nahfa911)

11 novembre, 2020

In this lab, we are suppose to Use a kernel that is the sum of three Gaussian kernels:

- The first to account for the distance from a station to the point of interest.
- The second to account for the distance between the day a temperature measurement was made and the day of interest.
- The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

to forecast the temperatures from 4 am (04:00) to 12 am (00:00) in an interval of 2 hours for a date and place in Sweden. To do so, we need to choose an appropriate smoothing coefficient or width for each of the three kernels above.

```
library(ggplot2)

# Function to compute the kernel distance (gaussian)
kernel_fun = function(norm, w){
  return( exp(-norm^2/(2*w^2)) )
}

#-------------------------------------------------------------------------------
# DISTANCE
#-------------------------------------------------------------------------------

# Set the grid for the distance
x_dist = seq(10,500,10)
# Set some plausible values
h_dist = c(50,100,200,400)

# Data frame for plots and kernel values
df_dist = data.frame(x = x_dist)
for(i in 1:length(h_dist)){
  df_dist[,(i+1)] = kernel_fun(x_dist, h_dist[i])
}

names <- c("Distance", "h = 50","h = 100", "h = 200","h = 400")
colnames(df_dist) = names

# Create a function to make the plots
plot_func <- function(data, names) {
  ggplot(data, aes_string(x = names[1])) +
    geom_line(aes(y = data[,2], col = names[2]), size = 0.5) +
    geom_line(aes(y = data[,3], col = names[3]), size = 0.5) +
    geom_line(aes(y = data[,4], col = names[4]), size = 0.5) +
    geom_line(aes(y = data[,5], col = names[5]), size = 0.5) +
    scale_colour_manual(values = c("steelblue", "forestgreen", "red", "maroon"),
                        name = "H values:") +
```

```
    labs(y = "Kernel density") +
    theme_light() +
    guides(colour=guide_legend(ncol=2,nrow=2,byrow=TRUE)) +
    theme(legend.position = "bottom")
}

p1 <- plot_func(df_dist, names) +
  geom_vline(aes(xintercept = 500), lty = 2, col = "black")

#------------------------------------------------------------------------------
# DAYS
#------------------------------------------------------------------------------

# Set the grid for the days
x_days = seq(1, 30, 1)
# Set some plausible values for the h
h_days = c(2,5,10,20)

# Data frame for plots and kernel values
df_days = data.frame(x = x_days)
for(i in 1:length(h_days)){
  df_days[,(i+1)] = kernel_fun(x_days, h_days[i])
}

names <- c("Days", "h = 2","h = 5", "h = 10","h = 20")
colnames(df_days) = names

p2 <- plot_func(df_days, names) +
  geom_vline(aes(xintercept = 14), lty = 2, col = "black")

#------------------------------------------------------------------------------
# HOURS
#------------------------------------------------------------------------------

# Set the grid for the hours
x_hours = seq(1,12,0.5)
# Set some plausible values for the h
h_hours = c(1,2,3,4)

# Data frame for plots and kernel values
df_hours = data.frame(x = x_hours)
for(i in 1:length(h_hours)){
  df_hours[,(i+1)] = kernel_fun(x_hours, h_hours[i])
}
```

```
names <- c("Hours", "h = 1","h = 2", "h = 3","h = 4")
colnames(df_hours) = names

p3 <- plot_func(df_hours, names) +
  geom_vline(aes(xintercept = 6), lty = 2, col = "black")

gridExtra::grid.arrange(p1, p2, p3, ncol = 3)
```

The code for doing this task is as following:

Code:

```
#-------------------------------------------------------------------------------
#-------------------------Sum of three Gaussian kernels-------------------------
#-------------------------------------------------------------------------------

# Importing modules
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext

# Crreating Context
sc = SparkContext(appName="lab_kernel")

# Getting data from external sources and splited each line to words
tempreture_readings = sc.textFile("data/temperature-readings.csv")
temp = tempreture_readings.map(lambda line: line.split(';'))

#==============================#
# Smoothing factors
h_distance = 200
h_date = 28
h_time = 4
#==============================#


#==============================#
# Input
date = "2013-07-04"
a = 58.4274
b = 14.826
times = ['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','00:00:00']
#==============================#
```

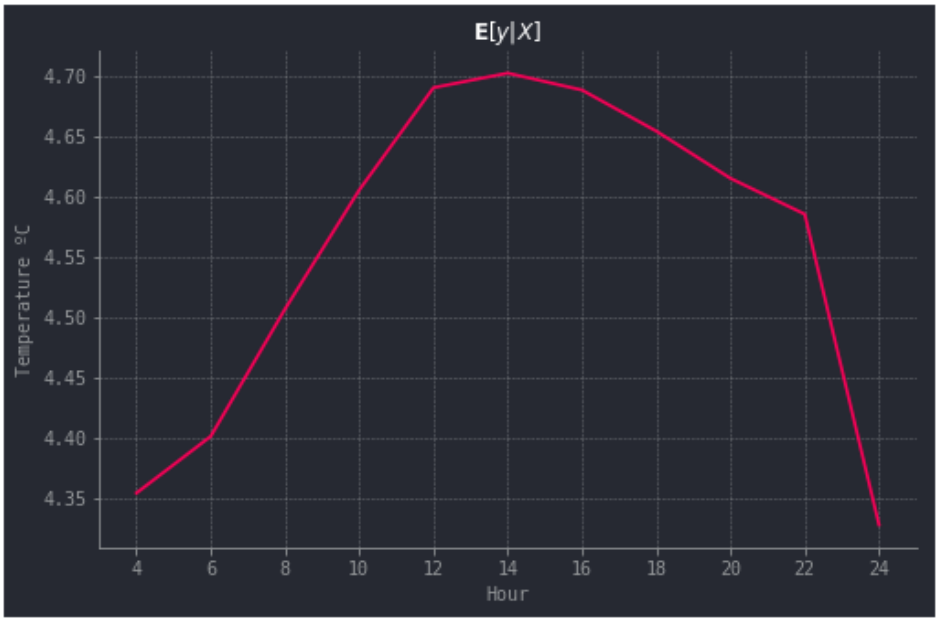
```
# Removing older dates
temp = temp.filter(lambda x: x[1] < date)

stations = sc.textFile('data/stations.csv')
stat = stations.map(lambda x: x.split(';'))

# Distance function (In km)
def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    km = 6367 * c
    return km

# datediff function (In days)
def datediff(date1,date2):
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date1, date_format)
    b = datetime.strptime(date2, date_format)
    delta = b - a
    return delta.days

# timediff function (In hours)
def timediff(time1,time2):
    time_format = "%H:%M:%S"
    a = datetime.strptime(time1, time_format)
    b = datetime.strptime(time2, time_format)
    delta = b - a
    return delta.seconds/3600


# Extracting Key-Value pairs
station_KeyValue = stat.map(lambda x: (x[0],(x[3],x[4])))

temp_KeyValue = temp.map(lambda x: (x[0], (x[1], x[2], float(x[3]))))

# Addind Kernel_dist to the station RDD
```

```
# Distance Kernel
station_distance = station_KeyValue.\
map(lambda x: (x[0],exp(-1*(haversine(float(x[1][0]), float(x[1][1]), a, b)/h_distance)

# Adding Kernel_date to the temprature RDD
# Date kernel
temp_dateKernel = temp_KeyValue.\
map(lambda x: (x[0],(x[1][1],x[1][2],exp(-1*(datediff(x[1][0],date)/h_date)**2))))

# Broadcasting the station RDD (To create copies on all nodes)
stat_map = station_distance.collectAsMap()
stat_broad = sc.broadcast(stat_map)

# Map side join
join_broad = temp_dateKernel.\
map(lambda t: (stat_broad.value.get(t[0], '-'), t[1][0],float(t[1][1]),t[1][2]))

# Time Kernel
# -- Function
def kernel_time(time1,time2):
    return exp(-1*(timediff(time1,time2)/h_time)**2)

# Kernel_sum Function
def kernel_sum(k1,k2,k3):
    return k1 + k2 + k3

join_broad.cache()

output = {}

for i in times:

    kernel_target = join_broad.\
    map(lambda x: (kernel_sum(x[0],x[3],kernel_time(x[1],i)),x[2]))
    num_denum = kernel_target.\
    map(lambda x: (x[0]*x[1], x[0])).reduce(lambda x,y: (x[0]+y[0], x[1]+y[1]))
    output[i] = num_denum[0]/num_denum[1]

output = sc.parallelize(output.items())
output.saveAsTextFile('./results_4/kernel_sum')
```

Result:

For smoothing coefficient:

- h_distance = 200
- h_date = 28
- h_time = 4



An alternative way of combining the three Gaussian kernels described above which is the product of three Gaussian kernels. In this case, the code is as bellow:

Code:

```
#-----------------------------------------------------------------------------------------#
#-------------------------Product of three Gaussian kernels-------------------------------#
#-----------------------------------------------------------------------------------------#

# Importing modules
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext

# Crreating Context
sc = SparkContext(appName="lab_kernel")

# Getting data from external sources and splited each line to words
```

```
tempreture_readings = sc.textFile("data/temperature-readings.csv")
temp = tempreture_readings.map(lambda line: line.split(';'))

#================================#
# Smoothing factors
h_distance = 200
h_date = 28
h_time = 4
#================================#


#================================#
# Input
date = "2013-07-04"
a = 58.4274
b = 14.826
times = ['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','00:00:00']
#================================#


# Removing older dates
temp = temp.filter(lambda x: x[1] < date)

stations = sc.textFile('data/stations.csv')
stat = stations.map(lambda x: x.split(';'))

# Distance function (In km)
def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    km = 6367 * c
    return km

# datediff function (In days)
def datediff(date1,date2):
```

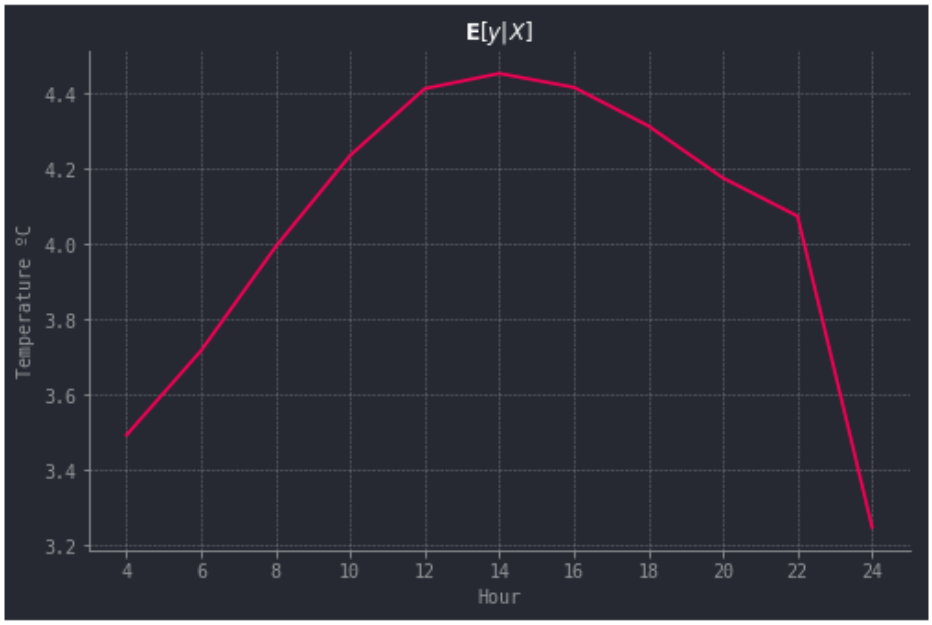
```
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date1, date_format)
    b = datetime.strptime(date2, date_format)
    delta = b - a
    return delta.days

# timediff function (In hours)
def timediff(time1,time2):
    time_format = "%H:%M:%S"
    a = datetime.strptime(time1, time_format)
    b = datetime.strptime(time2, time_format)
    delta = b - a
    return delta.seconds/3600


# Extracting Key-Value pairs
station_KeyValue = stat.map(lambda x: (x[0],(x[3],x[4])))

temp_KeyValue = temp.map(lambda x: (x[0], (x[1], x[2], float(x[3]))))

# Addind Kernel_dist to the station RDD
# Distance Kernel
station_distance = station_KeyValue.\
map(lambda x: (x[0],exp(-1*(haversine(float(x[1][0]), float(x[1][1]), a, b)/h_distance)**2)))


# Adding Kernel_date to the temprature RDD
# Date kernel
temp_dateKernel = temp_KeyValue.\
map(lambda x: (x[0],(x[1][1],x[1][2],exp(-1*(datediff(x[1][0],date)/h_date)**2))))

# Broadcasting the station RDD (To create copies on all nodes)
stat_map = station_distance.collectAsMap()
stat_broad = sc.broadcast(stat_map)

# Map side join
join_broad = temp_dateKernel.\
map(lambda t: (stat_broad.value.get(t[0], '-'), t[1][0],float(t[1][1]),t[1][2]))

# Time Kernel
# -- Function
def kernel_time(time1,time2):
    return exp(-1*(timediff(time1,time2)/h_time)**2)

# Kernel_sum Function
```

```
def kernel_prod(k1,k2,k3):
    return k1 * k2 * k3

join_broad.cache()

output = {}

for i in times:

    kernel_target = join_broad.\
    map(lambda x: (kernel_prod(x[0],x[3],kernel_time(x[1],i)),x[2]))
    num_denum = kernel_target.\
    map(lambda x: (x[0]*x[1], x[0])).reduce(lambda x,y: (x[0]+y[0], x[1]+y[1]))
    output[i] = num_denum[0]/num_denum[1]

output = sc.parallelize(output.items())
output.saveAsTextFile('./results_4/kernel_prod')
```

Result:

For smoothing coefficient:

- h_distance = 200
- h_date = 28
- h_time = 4


E[y|X]
Temperature °C
4.4
4.2
4.0
3.8
3.6
3.4
3.2
4
6
8
10
12
14
16
18
20
22
24
Hour