BIG DATA ANALYTICS LAB EXERCISE 3

Martynas Lukosevicius, Alejo Perez Gomez

17/05/2021

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.0.3
```

```
results_prod <- read_csv("results_prod.csv",
    col_names = FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double()
## )
```

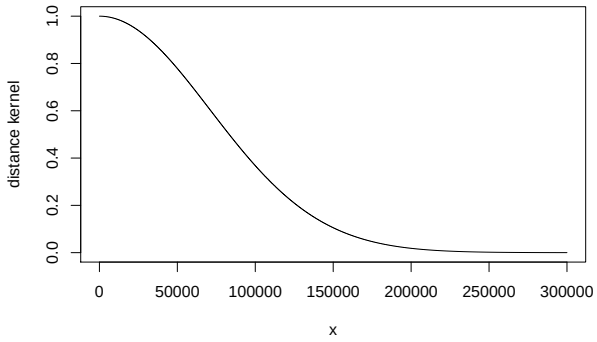
```
results_sum <- read_csv("results_sum.csv",
    col_names = FALSE)
```

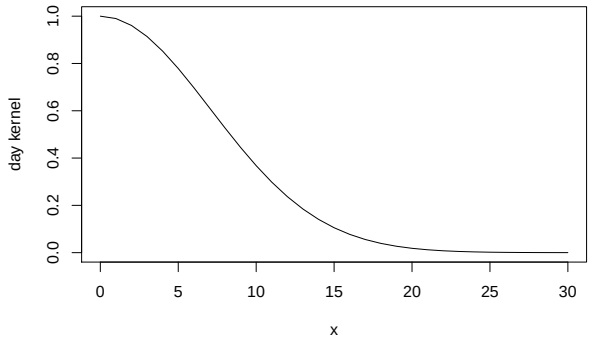
```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double()
## )
```

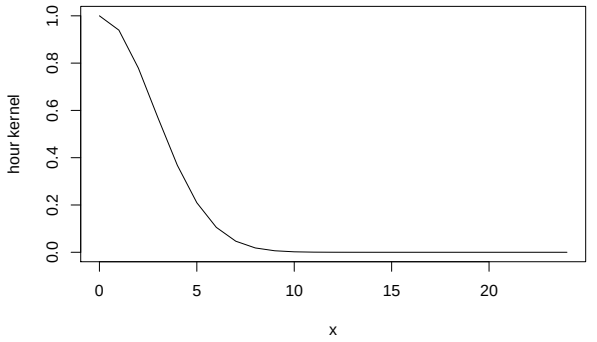
In this assignment we are required to use a Gaussian kernel-based algorithm to predict air temperatures based on Linköping meteorologic station Registers. Three Kernels operations will be computed:

- 1. Based in Haversine Great Circle Distance using coordinates
- 2. Based in time distance in days
- 3. Based in time distance hours

In order to choose the divisor constant h for each kernel we will plot the response of each over a reasonable support. We tried several values h until getting plots which conferred us larger response for smaller distance values and less for bigger ones. Therefore we will choose the following values for h.







Choosing h can be a critical stage for the carryover in the further kernel calculations. Based on the plots we will choose the following h for each kernel.

- 1. $h_{day} = 10$ To include median distances in days in the year
- 2. $h_{hour} = 4$ Will allow us to account for time distances within a day with with a short lapse of time, because during the day hours temperature suffers a strong variability
- 3. $h_{Haversine} = 100$ So as we can account for median distances as weather can be shared in great areas.

However, it is difficult to have control on the behavior of the calculation with respect to h as it doesn't control explicit distances in data.

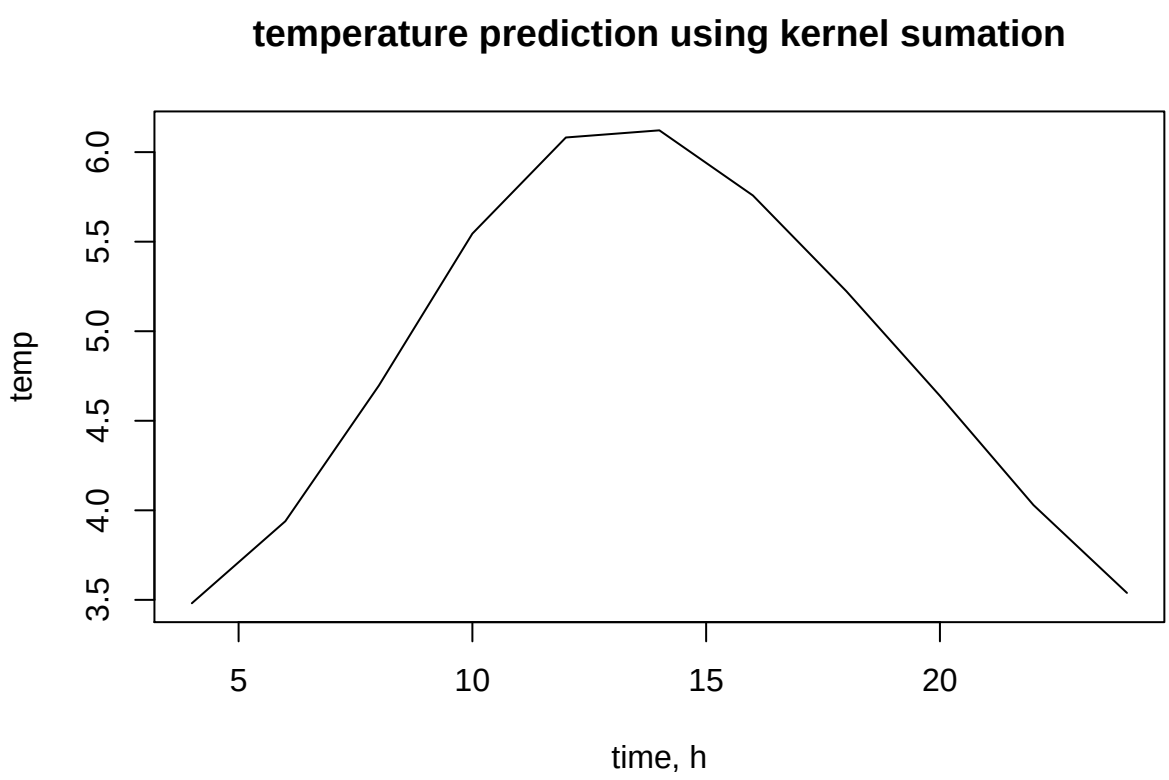
Sum of kernels

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext


## help functions
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

def kernel(norm,h):
    return(exp(-((norm)/h)**2))

def date_diff(x,y):
    date_format = "%Y-%m-%d"
    i = datetime.strptime(x,date_format)
    j = datetime.strptime(y,date_format)
    diff = j-i
    return diff.days

def time_diff(x,y):
    date_format = "%H:%M:%S"
    if x == "24:00:00":
        x = "23:59:59"
    if y == "24:00:00":
        y = "23:59:59"
    i = datetime.strptime(x,date_format)
    j = datetime.strptime(y,date_format)
    diff = min((i - j).seconds/3600, (j - i).seconds/3600) # to take min difference
    return diff

# initial params
h_distance = 100 #  Up to you
h_date = 10 # Up to you
h_time = 4 # Up to you
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-07-04" # Up to you

# spark
sc = SparkContext(appName="lab_kernel")

## load data
stations = sc.textFile("BDA/input/stations.csv").\
map(lambda line: line.split(";"))
temps = sc.textFile("BDA/input/temperature-readings.csv").\
sample(False, 0.1).map(lambda line: line.split(";"))

# filter date
temps = temps.filter(lambda x: x[1] < date)

station_kernel = stations.map(lambda x: (x[0],
```

```
                                        kernel(haversine(float(x[3]), float(x[4]), a, b),
                                        h_distance)))

stat_map = station_kernel.collectAsMap()
stat_broad = sc.broadcast(stat_map)

temps_kernel = temps.map(lambda x: (x[0], (kernel(date_diff(x[1],date) , h_date),
                                           x[2],
                                           stat_broad.value.get(x[0]), x[3])))
temps_kernel.cache()


results = []
for time in ["24:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
"12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]:
    expected_temp = temps_kernel.map(lambda x: (x[1][0] +
                                                x[1][2] +
                                                kernel(time_diff(x[1][1], time),
                                                h_time), x[1][3])).\
    map(lambda x: (x[0], x[0] * float(x[1]))).\
    reduce(lambda x,y: (x[0] + y[0], x[1] + y[1]))
    results.append(expected_temp[1]/expected_temp[0])

results = sc.parallelize(results)
results.saveAsTextFile("BDA/output")
```

```
plot(x = seq(24,4,-2),
     y = results_sum$X1,
     type = "l",
     main = "temperature prediction using kernel sumation",
     xlab = "time, h",
     ylab = "temp")
```



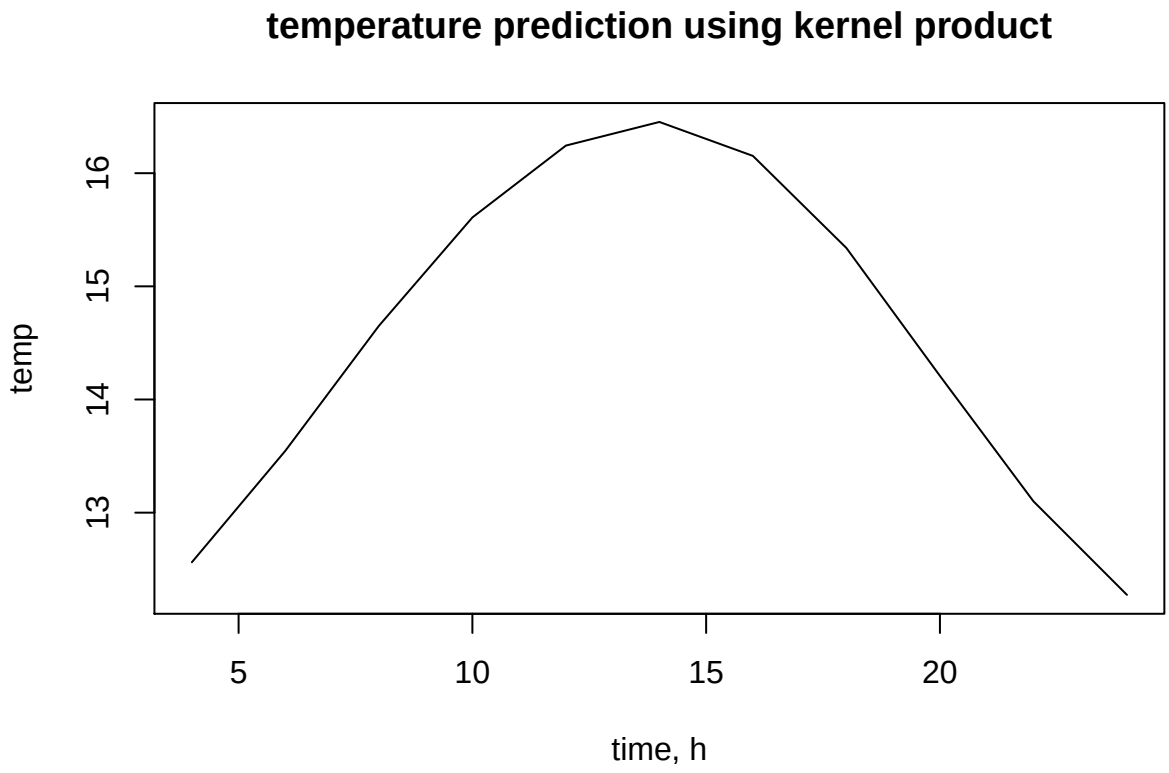
product of kernels

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext


## help functions
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

def kernel(norm,h):
```

```
    return(exp(-((norm)/h)**2))

def date_diff(x,y):
    date_format = "%Y-%m-%d"
    i = datetime.strptime(x,date_format)
    j = datetime.strptime(y,date_format)
    diff = j-i
    return diff.days

def time_diff(x,y):
    date_format = "%H:%M:%S"
    if x == "24:00:00":
        x = "23:59:59"
    if y == "24:00:00":
        y = "23:59:59"
    i = datetime.strptime(x,date_format)
    j = datetime.strptime(y,date_format)
    diff = min((i - j).seconds/3600, (j - i).seconds/3600) # to take min difference
    return diff

# initial params
h_distance = 100 #  Up to you
h_date = 10 # Up to you
h_time = 4 # Up to you
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-07-04" # Up to you

# spark
sc = SparkContext(appName="lab_kernel")

## load data
stations = sc.textFile("BDA/input/stations.csv").map(lambda line: line.split(";"))
temps = sc.textFile("BDA/input/temperature-readings.csv").\
        sample(False, 0.1).map(lambda line: line.split(";"))

# filter date
temps = temps.filter(lambda x: x[1] < date)

station_kernel = stations.map(lambda x: (x[0],
                                         kernel(haversine(float(x[3]), float(x[4]), a, b),
                                         h_distance)))

stat_map = station_kernel.collectAsMap()
stat_broad = sc.broadcast(stat_map)

temps_kernel = temps.map(lambda x: (x[0], (kernel(date_diff(x[1],date) , h_date), x[2], stat_broad.valu
temps_kernel.cache()


results = []
for time in ["24:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
"12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]:
    expected_temp = temps_kernel.map(lambda x: (x[1][0] * x[1][2] * kernel(time_diff(x[1][1], time) , h
```

```
    map(lambda x: (x[0], x[0] * float(x[1]))).\
    reduce(lambda x,y: (x[0] + y[0], x[1] + y[1]))
    results.append(expected_temp[1]/expected_temp[0])

results = sc.parallelize(results.items())
results.saveAsTextFile("BDA/output")
```

```
plot(x = seq(24,4,-2) ,y = results_prod$X1, type = "l", main = "temperature prediction using kernel proc
```



Conclusion

As a conclusion, we consider the outputted temperatures are below the expected for this time of the year (july) for the kernel sum. We blame this bias of the result on the election of the h parameter. We suggest, for future practices, apply a hold-out method. Therefore, we would subset data in training and testing sets for adjusting h hyper-parameter so as we can achieve sensible results of temperature. Unexpectedly based on the behaviour of the kernel product implementation, temperatures are higher and more feasible. It can be produced as product operation allows results to go larger if factors aren't for the most betwen 0 and 1.