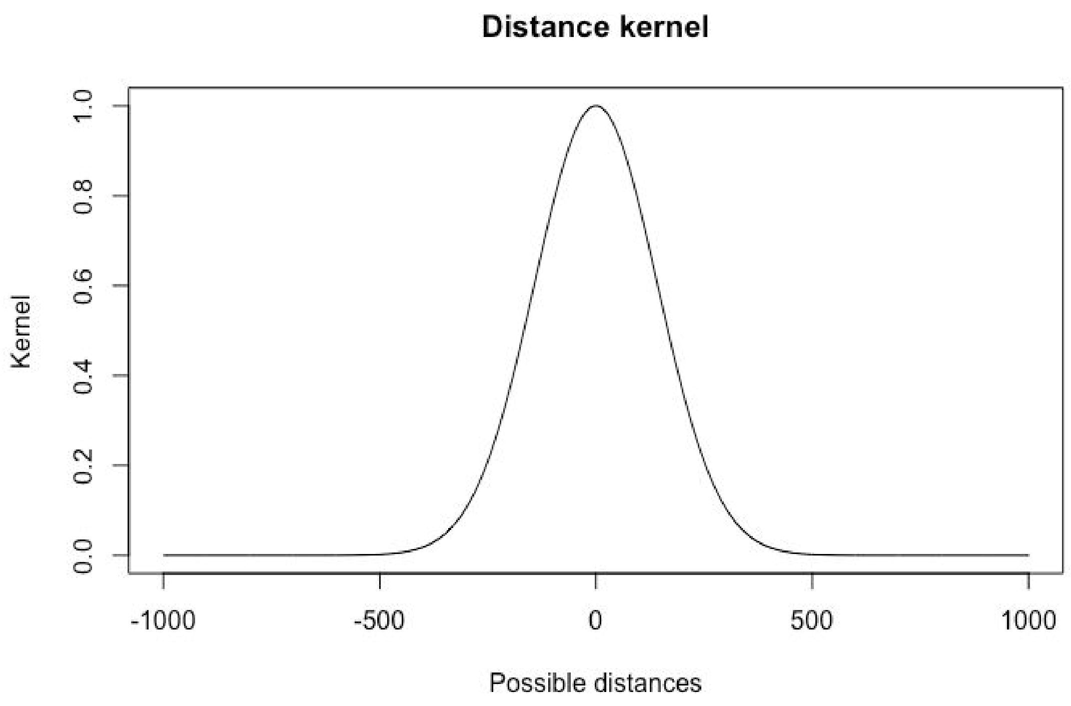
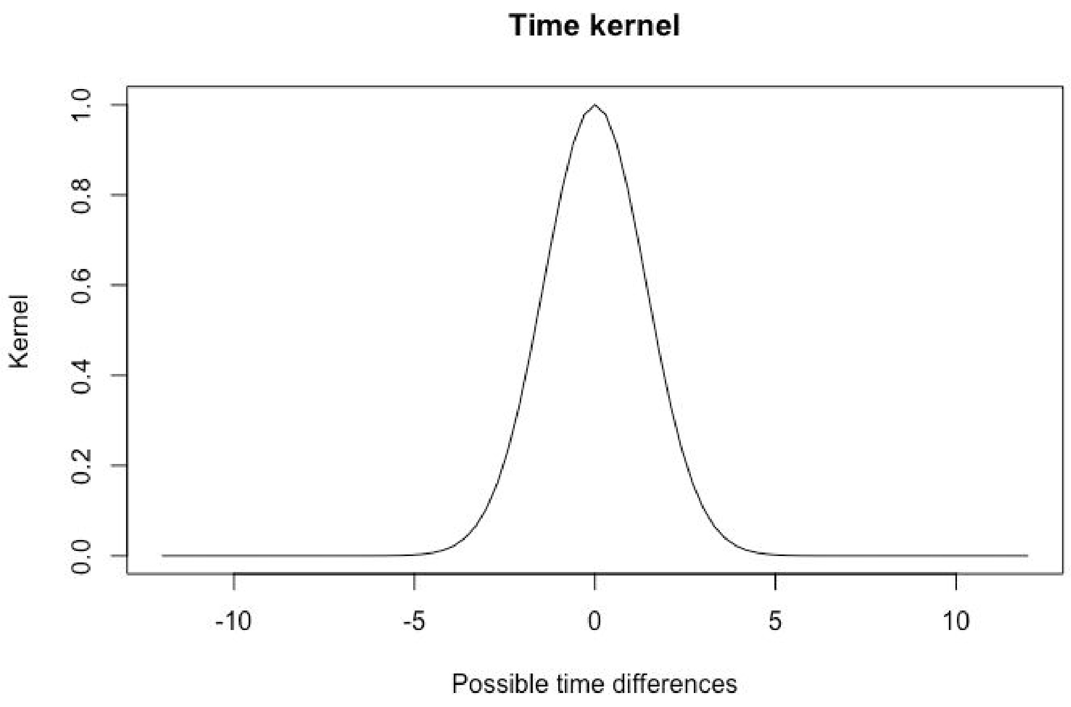
Alice Velander, alive213
David Björelind, davbj395

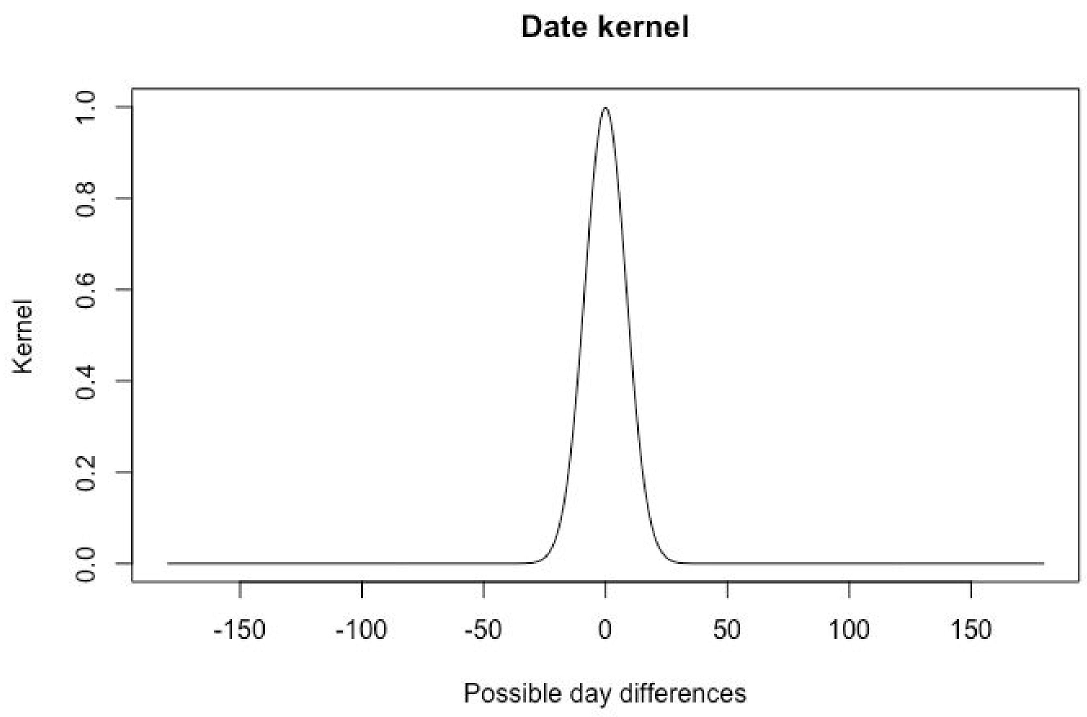
TDDE31: Big Data Analytics, Lab 3

Assignment 1)
Kernels:





Alice Velander, alive213
David Björelind, davbj395



H-parameters were chosen to make sure that values with long distance gave a smaller effect on the evaluated temperature. See graphs for the different kernels above.

a = 55.9903
b = 13.5958
date = “2001-01-01”

h_dist = 200
h_date = 12
h_time = 2

OUTPUT SUM KERNEL:
(Hour, temperature)

```
(24, 3.210637235600159)
(22, 3.4450384552119497)
(20, 3.82931426272712)
(18, 4.232538656763107)
(16, 4.475783216195441)
(14, 4.800393628118278)
(12, 5.019577041525178)
(10, 4.499517804842465)
(8, 3.8114380739884552)
(6, 3.304635094265015)
(4, 3.137402877824947)
```

Alice Velander, alive213
David Björelind, davbj395

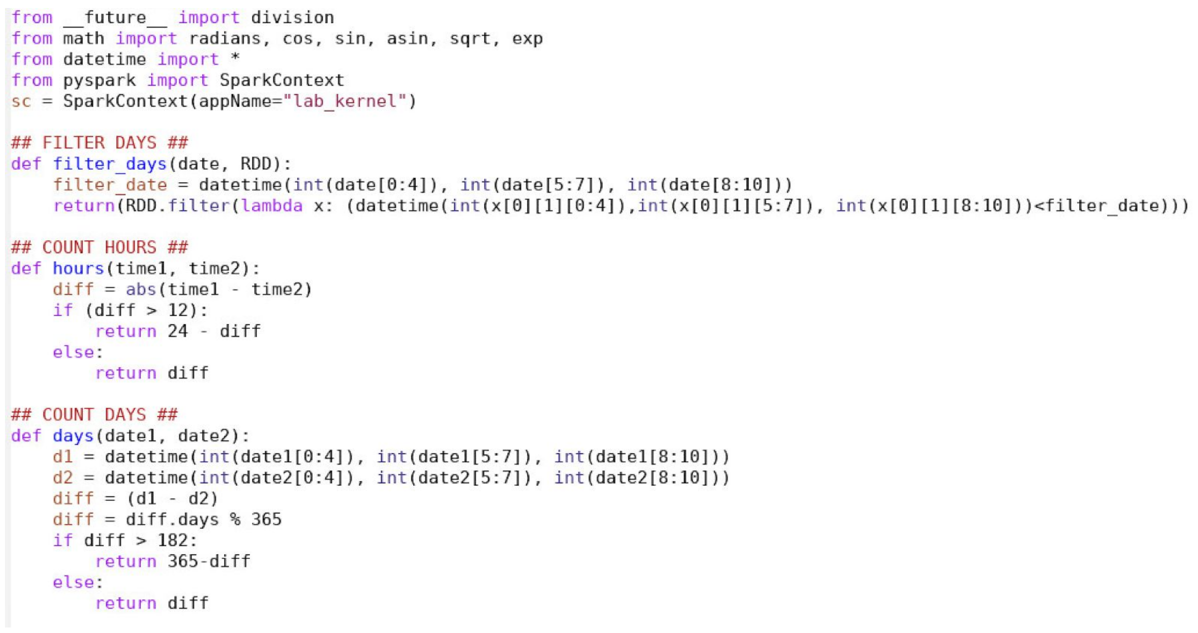
Multiplying the kernels

This will have the effect that if one kernel is close to zero, the whole kernel will also become close to zero, eliminating the value. That is why the preferable kernel to multiply the kernels, rather than using a kernel of sums.

OUTPUT MULT KERNEL:

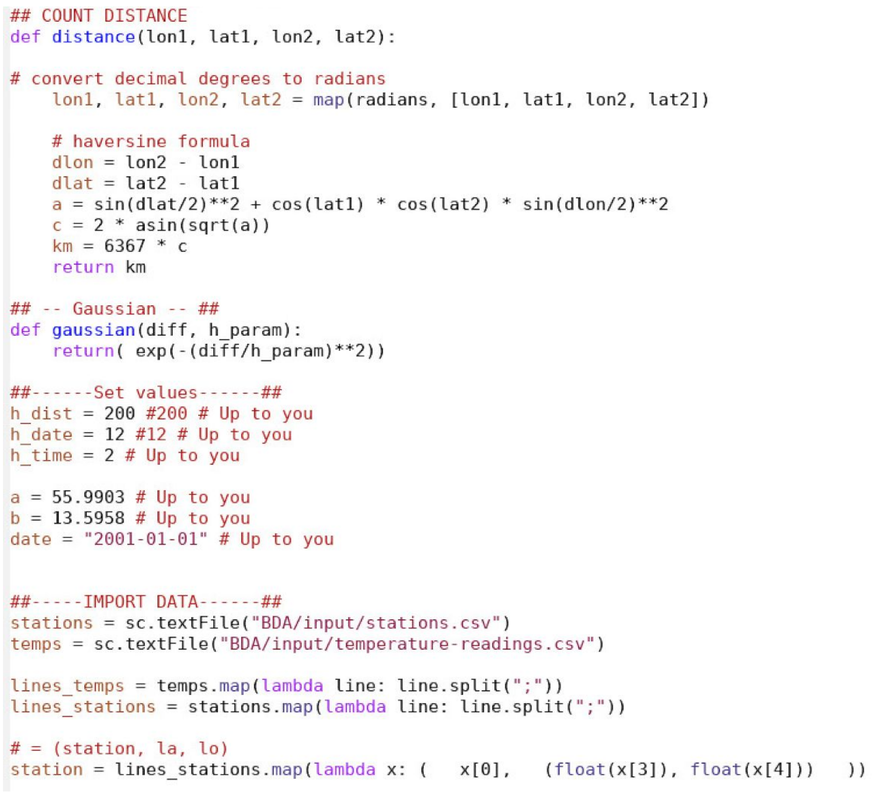
```
(24, -0.9327601538498143)
(22, -0.8199192113614985)
(20, -0.9313088820509134)
(18, -1.0551028053670508)
(16, -0.6349223405468639)
(14, -0.273363214967461)
(12, -0.41145568367220037)
(10, -0.6545670499230174)
(8, -1.2619580133389643)
(6, -1.6027692877609487)
(4, -1.3359730242959709)
```

CODE BELOW.

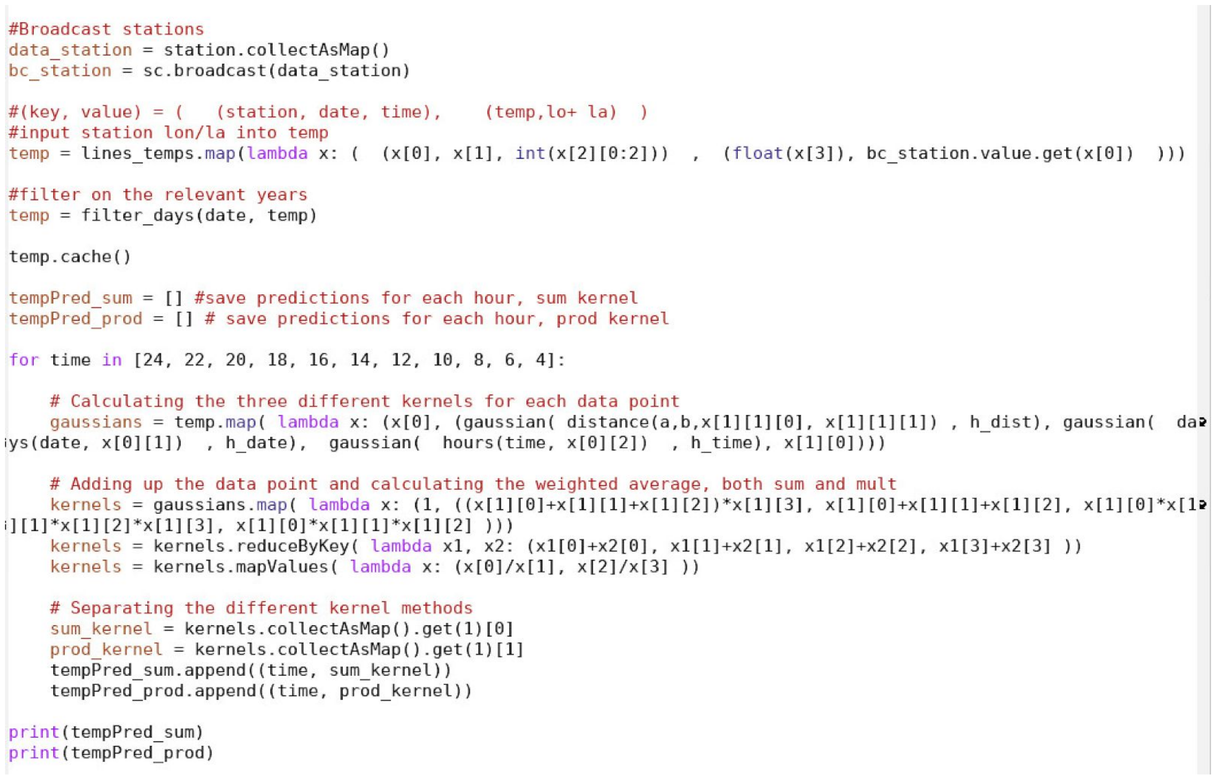


```python
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import *
from pyspark import SparkContext
sc = SparkContext(appName="lab_kernel")

## FILTER DAYS ##
def filter_days(date, RDD):
    filter_date = datetime(int(date[0:4]), int(date[5:7]), int(date[8:10]))
    return(RDD.filter(lambda x: (datetime(int(x[0][1][0:4]),int(x[0][1][5:7]), int(x[0][1][8:10]))<filter_date)))

## COUNT HOURS ##
def hours(time1, time2):
    diff = abs(time1 - time2)
    if (diff > 12):
        return 24 - diff
    else:
        return diff

## COUNT DAYS ##
def days(date1, date2):
    d1 = datetime(int(date1[0:4]), int(date1[5:7]), int(date1[8:10]))
    d2 = datetime(int(date2[0:4]), int(date2[5:7]), int(date2[8:10]))
    diff = (d1 - d2)
    diff = diff.days % 365
    if diff > 182:
        return 365-diff
    else:
        return diff
```

Alice Velander, alive213
David Björelind, davbj395



```
## COUNT DISTANCE
def distance(lon1, lat1, lon2, lat2):

# 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

## -- Gaussian -- ##
def gaussian(diff, h_param):
    return( exp(-(diff/h_param)**2))

##------Set values------##
h_dist = 200 #200 # Up to you
h_date = 12 #12 # Up to you
h_time = 2 # Up to you

a = 55.9903 # Up to you
b = 13.5958 # Up to you
date = "2001-01-01" # Up to you


##-----IMPORT DATA------##
stations = sc.textFile("BDA/input/stations.csv")
temps = sc.textFile("BDA/input/temperature-readings.csv")

lines_temps = temps.map(lambda line: line.split(";"))
lines_stations = stations.map(lambda line: line.split(";"))

# = (station, la, lo)
station = lines_stations.map(lambda x: (   x[0],   (float(x[3]), float(x[4]))   ))
```



```
#Broadcast stations
data_station = station.collectAsMap()
bc_station = sc.broadcast(data_station)

#(key, value) = (   (station, date, time),    (temp,lo+ la)   )
#input station lon/la into temp
temp = lines_temps.map(lambda x: (  (x[0], x[1], int(x[2][0:2]))  ,  (float(x[3]), bc_station.value.get(x[0])  )))

#filter on the relevant years
temp = filter_days(date, temp)

temp.cache()

tempPred_sum = [] #save predictions for each hour, sum kernel
tempPred_prod = [] # save predictions for each hour, prod kernel

for time in [24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4]:

    # Calculating the three different kernels for each data point
    gaussians = temp.map( lambda x: (x[0], (gaussian( distance(a,b,x[1][1][0], x[1][1][1]) , h_dist), gaussian(  da
ys(date, x[0][1])  , h_date),  gaussian(  hours(time, x[0][2])  , h_time), x[1][0])))

    # Adding up the data point and calculating the weighted average, both sum and mult
    kernels = gaussians.map( lambda x: (1, ((x[1][0]+x[1][1]+x[1][2])*x[1][3], x[1][0]+x[1][1]+x[1][2], x[1][0]*x[1
][1]*x[1][2]*x[1][3], x[1][0]*x[1][1]*x[1][2] )))
    kernels = kernels.reduceByKey( lambda x1, x2: (x1[0]+x2[0], x1[1]+x2[1], x1[2]+x2[2], x1[3]+x2[3] ))
    kernels = kernels.mapValues( lambda x: (x[0]/x[1], x[2]/x[3] ))

    # Separating the different kernel methods
    sum_kernel = kernels.collectAsMap().get(1)[0]
    prod_kernel = kernels.collectAsMap().get(1)[1]
    tempPred_sum.append((time, sum_kernel))
    tempPred_prod.append((time, prod_kernel))

print(tempPred_sum)
print(tempPred_prod)
```

Alice Velander, alive213
David Björelind, davbj395