LAB REPORT 3

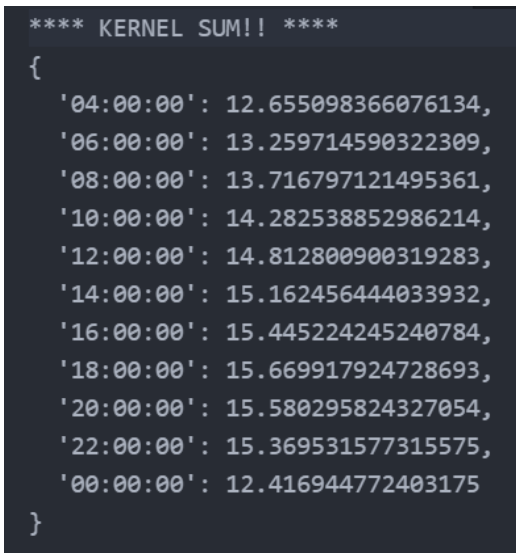
TDDE31

Prepared by: Aswathi Karunakaran Beena(aswka169)
Nandana Krishna (nankr143)

● **Show that your choice for the kernels' width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.**

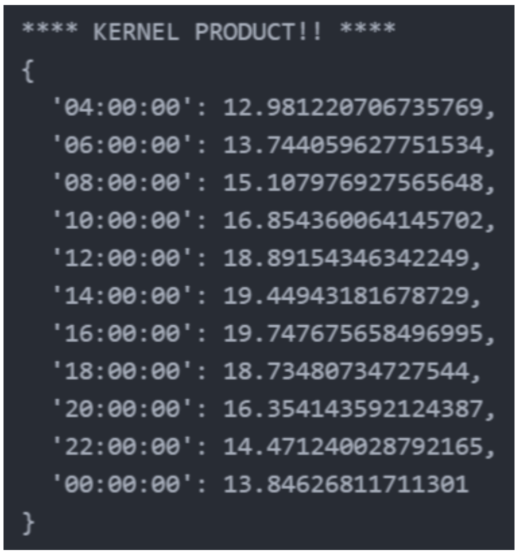
The h parameters for the different kernels were selected based on a reasonable weight to closer points. For the distance kernel, a distance of 100 km was selected, because less than that, say a 50 would be too less to consider all the different temperatures in a region. And too large a distance would give us too many data points and varied temperature differences. The width of the distance between the days was selected as 5 because as in the case above, a lesser distance would give very similar temperatures and we cannot derive the required number of data points from the kernel. This is also the reason behind the selection of time kernel width as 3 hours, which is enough to get the different temperature readings without being too similar to each other.

● **Repeat the exercise using a kernel that is the product of the three Gaussian kernels above. Compare the results with those obtained for the additive kernel. If they differ, explain why.**



```
**** KERNEL SUM!! ****
{
  '04:00:00': 12.655098366076134,
  '06:00:00': 13.259714590322309,
  '08:00:00': 13.716797121495361,
  '10:00:00': 14.282538852986214,
  '12:00:00': 14.812800900319283,
  '14:00:00': 15.162456444033932,
  '16:00:00': 15.445224245240784,
  '18:00:00': 15.669917924728693,
  '20:00:00': 15.580295824327054,
  '22:00:00': 15.369531577315575,
  '00:00:00': 12.416944772403175
}
```

Sum of Kernels



```
**** KERNEL PRODUCT!! ****
{
  '04:00:00': 12.981220706735769,
  '06:00:00': 13.744059627751534,
  '08:00:00': 15.107976927565648,
  '10:00:00': 16.854360064145702,
  '12:00:00': 18.89154346342249,
  '14:00:00': 19.44943181678729,
  '16:00:00': 19.747675658496995,
  '18:00:00': 18.73480734727544,
  '20:00:00': 16.354143592124387,
  '22:00:00': 14.471240028792165,
  '00:00:00': 13.84626811711301
}
```

Product of Kernels

When the results of both the kernels were compared, a few changes in the temperature value contribution from the different kernels were noticed. This could be due to the fact that additive kernels would give a kernel value sum that would not change even if one of the kernel values turned out to be a value closer to zero. But for the multiplication of kernels, a data point with kernel value zero or near, would not contribute to the temperature prediction.

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

sc = SparkContext(appName="lab_kernel")

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

h_distance = 100 # 100 km
h_date = 5 # 5 days
h_time = 3*3600 # 10800 seconds or 3 hours
a = 58.4274
b = 14.826
date = "2013-07-04"

stations_file = sc.textFile("BDA/input/stations.csv")
temps_file = sc.textFile("BDA/input/temperature-readings.csv")
stations = stations_file.map(lambda line: line.split(";"))
stations = stations.map(lambda x: (int(x[0]), float(x[3]), float(x[4])))
temps = temps_file.map(lambda line: line.split(";"))
temps = temps.map(lambda x: (int(x[0]),x[1],x[2],float(x[3])))

#Gaussian kernel for Distance
def kernelfordistance(long1,lat1,long2,lat2,h):
        distance = haversine(long1,lat1,long2,lat2)
        rdist = exp(-((distance/h)**2))
        return rdist
#Gaussian kernel for Date
def kernelfordate(x,date,h):
        diffdate = (datetime(int(x[0:4]),int(x[5:7]),int(x[8:10])) -
datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))).days
        rdate = exp(-((diffdate/h)**2))
        return rdate
```

```
#Gaussian kernel for Time
def kernelfortime(x,time,date,h):
        difftime =
(datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]),int(x[0:2]),int(x[3:5]),int(x[6:8])) -
datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]),int(time[0:2]),int(time[3:5]),int(time[6:8]))).
seconds
        rtime = exp(-((difftime/h)**2))
        return rtime

distkernel = stations.map(lambda x: (x[0], kernelfordistance(x[2],x[1],b,a,h_distance)))
diststations = sc.broadcast(distkernel.collectAsMap())
#Filtering on the dates
datefilter = temps.filter(lambda x: (datetime(int(date[0:4]),int(date[5:7]),int(date[8:10])) >=
datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]))))
#caching the filtered data
datefilter.cache()
#creating dictionary for sum and product of kernels
temperature_sum = {}
temperature_product = {}

for time in ["00:00:00", "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"]:
        #Filtering on the times
        datefilter = datefilter.filter(lambda x:
(datetime(int(date[0:4]),int(date[5:7]),int(date[8:10])) ==
datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]))))
        timefilter = datefilter.filter(lambda x:
(datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]),int(time[0:2]),int(time[3:5]),int(time[6:8]))
>= datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]),int(x[2][0:2]),int(x[2][3:5]),int(x[2][6:8]))))
        kernel = timefilter.map(lambda x: (diststations.value[x[0]],
kernelfordate(x[1],date,h_date), kernelfortime(x[2],time,date,h_time),x[3]))

        #Sum of kernels
        sum = kernel.map(lambda x: (x[0] + x[1] + x[2],x[3]))
        sum = sum.map(lambda x: (x[0]*x[1],x[0]))
        sum = sum.reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
        temperature_sum[time] = sum[0]/sum[1]

        #Product of kernels
        product = kernel.map(lambda x: (x[0] * x[1] * x[2],x[3]))
        product = product.map(lambda x: (x[0]*x[1],x[0]))
        product = product.reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
        temperature_product[time] = product[0]/product[1]

print("**** KERNEL SUM!! ****")
print(temperature_sum)
print("**** KERNEL PRODUCT!! ****")
print(temperature_product)
```