Lab3

Balaji Ramkumar(balra340) and Priya Kurian Pullolickal (priku577)

**My\_code:**

from \_\_future\_\_ import division  
from pyspark import SparkContext  
from math import radians, cos, sin, asin, sqrt, exp, fabs  
from datetime import datetime  
import math, collections  
from datetime import timedelta  
  
  
sc = SparkContext(appName="lab\_kernel")  
  
  
temperature = sc.textFile("/user/x\_priku/data/temperature-readings.csv")  
temperature = temperature.map(lambda l: l.split(";"))  
temperature = temperature.map(lambda l: (int(l[0]), (str(l[1]), str(l[2]), float(l[3]))))  
  
                   
stations = sc.textFile("/user/x\_priku/data/stations.csv")  
stations = stations.map(lambda l: l.split(";"))  
stations = stations.map(lambda l: (int(l[0]), (float(l[3]), float(l[4]))))                   
                         
#Broadcast stations   
rdd = stations.collectAsMap()         
sb = sc.broadcast(rdd)  
                                         
def joining\_func(x):  
    dictvalues = list(sb.value[x[0]])  
    values = list(x[1])  
    values.extend((dictvalues[0], dictvalues[1]))  
    res = (x[0], tuple(values))  
    return res  
  
  
data = temperature.map(lambda row: joining\_func(row))  
  
  
def haversine(lon1, lat1, lon2, lat2):  
     
   #Convert decimal degrees to radians  
   lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])        
         
  #Havershine 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 gauss\_distance(dist, h):  
        if isinstance(dist, collections.Iterable):  
            distance = []  
            for d in dist:  
                      distance.append(exp(float(-(d\*\*2))/float((2\*(h\*\*2)))))  
        else:  
             distance = exp(float(-(dist\*\*2))/float((2\*(h\*\*2))))  
        return distance  
      
           
            
def date\_distance(date1, date2):  
     
   date1 = datetime.strptime(date1, '%Y-%m-%d')  
   date2 = datetime.strptime(date2, '%Y-%m-%d')  
   days\_diff = (date1 - date2).days  
   return days\_diff  
  
  
  
def time\_format(time):            
   if isinstance(time, collections.Iterable):        
               result = []      
               for i in time:  
                      if i <= -12:  
                         result.append(24 + i)  
                      else:  
                         result.append(fabs(i))  
   else:  
          if time <= -12:  
                  result = 24 + time  
          else:  
                  result = fabs(time)  
   return result  
  
def time\_distance(time1, time2):  
    a = timedelta(hours=int(time1[0:2]), minutes= int(time1[3:5]), seconds=int(time1[6:8]))  
    b = timedelta(hours=int(time2[0:2]), minutes= int(time2[3:5]), seconds=int(time2[6:8]))  
    times=(a-b).total\_seconds()/3600  
    time\_diff = time\_format(times)  
    return(time\_diff)  
        
  
def gauss\_kernel(prediction, data):  
  
     results = []  
     output=data.cache()  
  
     for i in prediction:  
  
               output2 = output.filter(lambda x: datetime.strptime(x[1][0], '%Y-%m-%d') < datetime.strptime(i[1], '%Y-%m-%d'))  
  
               output2 = output2.map(lambda x: (x[1][2], (time\_distance(i[0], x[1][1]),date\_distance(i[1], x[1][0]),haversine(lon1=i[3],lat1=i[2],lon2=x[1][4],lat2=x[1][3]))))  
  
               output2 = output2.map(lambda (results, (dtime, ddate, dplace)): (results, (gauss\_distance(dtime, h=h\_time),gauss\_distance(ddate, h=h\_date),gauss\_distance(dplace, h=h\_distance))))  
  
               output2 = output2.map(lambda (results, (k1, k2, k3)): (results, k1+k2+k3))  
  
               output2 = output2.map(lambda (results, ksum): (results, (ksum, ksum\*results)))  
  
               output2 = output2.map(lambda (results, (ksum, tot\_ksum)): (None, (ksum, tot\_ksum)))  
  
               output2 = output2.reduceByKey(lambda (ksum1, tot\_ksum1), (ksum2, tot\_ksum2): (ksum1+ksum2, tot\_ksum1+tot\_ksum2))  
  
               output2 = output2.map(lambda (key, (finalksum, finaltot\_ksum)): (float(finaltot\_ksum)/float(finalksum)))  
  
               results.append((i[0], output2.collect()))  
  
     return results                                   
  
h\_distance = 50  
h\_date = 10  
h\_time = 2  
a = 58.39     
b = 15.62     
date = "2013-01-27"        
            
        
predict = (('04:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('06:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('08:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('10:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('12:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('14:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('16:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('18:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('20:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('22:00:00', '2015-01-15', float(58.41), float(15.62)),  
           ('00:00:00', '2015-01-15', float(58.41), float(15.62)))         
          
my\_pred = gauss\_kernel(prediction = predict, data = data)  
my\_predRdd = sc.parallelize(my\_pred).repartition(1)  
my\_predRdd.saveAsTextFile("Labr1")

**final\_output:**

('10:00:00', [5.835490837622072])  
('20:00:00', [4.7877298982318885])  
('04:00:00', [3.509654977349956])  
('16:00:00', [5.8943810158401755])  
('06:00:00', [3.9161715403095])  
('22:00:00', [4.4744772474366])  
('18:00:00', [5.287906314375653])  
('00:00:00', [3.6164628073530087])  
('12:00:00', [6.396710872677871])  
('08:00:00', [4.770047263531616])  
('14:00:00', [6.394809534096611])

**Answers:**

1. In the above case, we have considered h\_distance as 50, h\_time as 2 and h\_date as 10. The reason for choosing a small value is with larger values is ,when we take large values the region that is considered for calculation is huge. This makes the prediction less accurate. T overcome this, We used a lesser values for h.
2. Yes, Like mentioned the predicted temperature don’t differ much from each other. The independency might have an effect on the predictions because when summing it all three kernels. Even one wrongs assumption of h value can have an impact in the prediction. The alternate method is combining of kernels using multiplication. This can actually provide better prediction when compared to summation of kernels.