BDA3

Rabnawaz jansher & Saman Zahid

5/26/2018

BDA3 - Machine Learning with Spark - Exercises

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

import math, collections
from datetime import datetime

#spark object
sc = SparkContext(appName="BDA3")

#kernel widths
h_distance = 100
h_date = 10
h_time = 1
a = 58.3987 #latitude
b = 15.5771 #longitude
date = "2013-07-04"
timeList = ['04:00:00','06:00:00','08:00:00','10:00:00','11:00:00','12:00:00','14:00:00','16:00:00','18:00:00','20:00:00','22:00:00','00:00:00' ]
###############

# Reading data from temperature file
temps = sc.textFile("/user/x_samza/data/temperature-readings.csv")
#for testing sample
temps = temps.sample(False, 0.1)
temp_lines = temps.map(lambda line: line.split(";"))

temperatures = temp_lines.map(lambda l: (int(l[0]), (str(l[1]), l[2], float(l[3])  )))

#filter posterior data
temperatures = temperatures.filter(lambda x: ( datetime.strptime(str(x[1][0]), '%Y-%m-%d') < datetime.strptime(date, '%Y-%m-%d')))
stations = sc.textFile("/user/x_samza/data/stations.csv")
station_lines = stations.map(lambda line: line.split(";"))
```

```
#station rdd with key value pair
stations = station_lines.map(lambda l: (int(l[0]), (float(l[3]), float(l[4]))))

#broadcast stations Rdds
rdd = stations.collectAsMap()
stationsBroadcast = sc.broadcast(rdd)

#combinig rdds with station lat lng, with station rdds
def joiningRdds(x):
    station_dict = list(stationsBroadcast.value[x[0]]) #make list
    values = list(x[1])
    values.extend((station_dict[0], station_dict[1]))
    result = (x[0], tuple(values))
    return result

#call function to join join data: (station, (date, time, temp, lat, lon))
data = temperatures.map(lambda row: joiningRdds(row))


#def functions
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
```

```python
    return km

#gussian distance
def gaussDistance(dis, h):
    if isinstance(dis, collections.Iterable):
        distance = []
        for d in dis:
            distance.append(exp(float(-(d**2))/float((2*(h**2)))))
    else:
        distance = exp(float(-(dis**2))/float((2*(h**2))))
    return distance


# date distance
def dateDistance(day1, day2):
    d1 = datetime.strptime(day1, '%Y-%m-%d')
    d2 = datetime.strptime(day2, '%Y-%m-%d')
    daydiff = (d1 - d2).days
    return daydiff

# time formating
def timeFormat(time):
    if isinstance(time, collections.Iterable):
        res = []
        for t in time:
            if t <= -12:
                res.append(24 + t)
            else:
                res.append(fabs(t))
    else:
        if time <= -12:
            res = 24 + time
```

```python
        else:
            res = fabs(time)
    return res



#time distance
def timeDistance(time1, time2):
    t1 = datetime.strptime(time1, '%H:%M:%S')
    t2 = datetime.strptime(time2, '%H:%M:%S')
    times = (t1 - t2).total_seconds()/3600
    timeDiff = timeFormat(times)
    return(timeDiff)

#Main kernel
def gaussKernel(pred, data):
    results = []
    for p in pred:
        #filter posteriro data
        temp = data.filter(lambda x: datetime.strptime(x[1][0], '%Y-%m-%d') < datetime.strptime(p[1], '%Y-%m-%d')).cache()\
                .map(lambda x: (x[1][2], (timeDistance(p[0], x[1][1]),\
                                          dateDistance(p[1], x[1][0]),\
                                          haversine(lon1=p[3],lat1=p[2],lon2=x[1][4],lat2=x[1][3]))))\
                .map(lambda (temp, (dtime, ddate, dplace)): (temp, (gaussDistance(dtime, h=h_time),\
                                                                    gaussDistance(ddate, h=h_date),\
                                                                    gaussDistance(dplace, h=h_distance))))\
                .map(lambda (temp, (k1, k2, k3)): (temp, k1+k2+k3))\
                .map(lambda (temp, kSum): (temp, (kSum, kSum*temp)))\
                .map(lambda (temp, (kSum, tkSum)): (None, (kSum, tkSum)))\
                .reduceByKey(lambda (kSum1, tkSum1), (kSum2, tkSum2): (kSum1+kSum2, tkSum1+tkSum2))\
                .map(lambda (key, (totalkSum, totaltkSum)): (float(totaltkSum)/float(totalkSum)))
        results.append(temp.collect())
```

```python
        results.append(temp.collect())
    return results #return result


#predict tuples
predict = (('04:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('06:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('08:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('10:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('12:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('14:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('16:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('18:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('20:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('22:00:00', '2013-07-04', float(58.3987), float(15.5771)),
           ('00:00:00', '2013-07-05', float(58.3987), float(15.5771)))

#call gussian kernel method
myPrediction = gaussKernel(pred = predict, data = data)

#repartion data and save result to file
myPredictionRdd = sc.parallelize(myPrediction).repartition(1)
myPredictionRdd.saveAsTextFile("weights")
```

```
[6.304296689646911]
[5.736471905295238]
[4.7858819316861165]
[5.5156957925473575]
[6.550546935862137]
[6.559789673556806]
[5.718505440633309]
[5.06338970365915]
[5.003903846962087]
[6.246724337458326]
[5.633666917188279]
```

Q.1 - Show that your choice for the kernelsí width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.

By running the code on different values of h, we have observed the fact that higher the values of h, lesser will be the effect on kernel. It is because of the fact that when we take higher values of h, we get the wider kernel which means that more points are being considered for calculation. Thus, we have chosen a reasonably smaller value of h so that more weight is assigned to the close points which will be more effective in calculation thus giving better predictions.

Q.2 - It is quite likely that the predicted temperatures do not differ much from one another. Do you think that the reason may be that the three Gaussian kernels are independent one of another? If so, propose an improved kernel, e.g. propose an alternative way of combining the three Gaussian kernels described above.

The predicted temperatures do not differ much from one another, it could be because the we are summing over the three independent kernels, which might gives result on the basis of all measures that is if distance is small but time difference is large, then time alone will also effect the prediction (because of kernel independency).

The alternate way could be by combining kernels by taking product of three kernels instead of summing them up and then averaging them. By combining the kernels via multiplication, proportions of the three different kernels is preserved.