LAB3 kernal lab

Wuhao Wang(wuhwa469)

Mucahit Sahin(mucsa806)

Predictive temperature from **sum kernel**

24	22	20	18	16	14	12	10	8	6	4
6.841	5.329	5.446	5.811	6.663	7.250	7.137	6.673	5.457	4.310	4.121

Our interested target and window

```
h_distance = 40# Up to you
h_date = 10# Up to you
h_time = 2# Up to you
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-07-04" # Up to you
```

Q1: 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.

Kernel value based on physical distance.

The data from closer place will have heavier impact on results. However, we think there is something not so reasonable. When we think about distance, longitude and latitude should have different impact on the temperature. And also, the geographical location(next to the sea or mountain?).

```
(u'85450', 45.28844486923342, 0.5656757566925599)
(u'84060', 44.904407344419326, 0.5711446796886469)
(u'84620', 44.6786665952493, 0.5743620416734648)
(u'84050', 44.259864953684485, 0.5803355603114833)
(u'84020', 44.16653152975839, 0.581667528096289)
(u'85390', 41.86683237941111, 0.6145299074916714)
(u'85240', 41.242584767952465, 0.6234502348329826)
(u'85460', 34.23184113226634, 0.7221623794142671)
(u'85270', 32.11993166036042, 0.7508265907838528)
(u'85180', 28.495481666919968, 0.7980743899429829)
(u'85630', 27.940551269536694, 0.8050474964868356)
(u'85210', 23.15041451136365, 0.8616788063190374)
(u'85220', 22.318605064363176, 0.8707793155090162)
(u'84340', 21.794675890810677, 0.8763877595414241)
(u'84310', 20.835541965616056, 0.8863983399122084)
```

```
u'75040', 165.6882136210334, 0.0004877450072987605)
u'96370', 164.87364282242137, 0.000525625438096654)
u'97210', 164.29498801854527, 0.0005541850127740023)
u'82000', 163.69944094885474, 0.000585085869947879)
u'95500', 163.6954712593147, 0.0005852971332063903)
u'72290', 163.44778326860663, 0.0005986204873928594)
u'95540', 162.26293886740288, 0.000666357861105446)
u'95530', 162.23325718307373, 0.0006681430503898399)
u'82030', 161.707962400135, 0.0007004831080879703)
u'73090', 160.89578065716594, 0.0007533663305663478)
u'82490', 159.99427470689653, 0.0008164032017268408)
u'76160', 159.50822976088688, 0.0008523908619448471)
u'97150', 159.02813537865703, 0.0008893804051485432)
```

kernel val based on date

The kernel value here are influenced by 2 factors. The first one is the month-day distance which we think make sense in season difference. Second factor is year distance. We all know that the climate will change every year and the temperature become warmer and warmer, so the closer year should have heavier impact on the target date.

In our code, we set day difference as

day_dis = abs((d1-d2).days)%365 + 0.2*abs((d1-d2).days)/365

The pictures below can illustrate the formula. Our vi target date is '2013-07-04'. '2010-07-05' and '2011-07-05' ,'1996-07-04' are almost equally distant from target date based on season level but 2011 is closer to 2013 so that it will have higher weight value.

```
(u'2010-07-05', 0.9964064722309933)
(u'2011-07-05', 0.9984012793176064)
(u'2012-07-04', 0.9996000799893344)
```

```
(u'1996-07-04', 0.5781485527804839)
```

```
(u'2012-04-20', 2.5919285614752597e-25)
```

kernel value based on hour

In this section we have the same strategy, closer time will be more important. Here is the example of target time 12:00:00.

```
(u'12:19:00', 0.9045861092143463)
(u'12:18:00', 0.9139311852712282)
(u'11:42:00', 0.9139311852712282)
(u'11:43:00', 0.922859960790065 4)
(u'12:17:00', 0.922859960790065 4)
(u'12:16:00', 0.931358402111352)
(u'12:15:00', 0.9394130628134758)
(u'11:45:00', 0.9394130628134758)
(u'11:51:00', 0.9777512371933363)
(u'12:03:00', 0.9975031223974601)
(u'12:01:00', 0.9997222607988971)
```

```
(u'19:17:00', 9.162305025905954e-24)
(u'04:44:00', 1.1676727839381582e-23)
(u'19:16:00', 1.1676727839381582e-23)
(u'19:15:00', 1.4872921816512705e-23)
(u'04:45:00', 1.4872921816512705e-23)
(u'04:50:00', 4.9448470173055056e-23)
```

Q2

Predictive temperature from multiply kernel

24	22	20	18	16	14	12	10	8	6	4
14.04	14.67	16.32	17.93	18.88	19.30	19.07	18.14	16.57	15.06	13.59

The result below is the closest samples, the fourth column shows the temperature, the last column is the distance from our target input.

```
(u'84310', (u'2013-07-04', u'00:00:00', u'14.0', 20.835541965616056))
(u'84310', (u'2013-07-04', u'01:00:00', u'13.3', 20.835541965616056))
(u'84310', (u'2013-07-04', u'02:00:00', u'13.0', 20.835541965616056))
(u'84310', (u'2013-07-04', u'03:00:00', u'12.8', 20.835541965616056))
(u'84310', (u'2013-07-04', u'04:00:00', u'12.7', 20.835541965616056))
(u'84310', (u'2013-07-04', u'05:00:00', u'12.9', 20.835541965616056))
(u'84310', (u'2013-07-04', u'06:00:00', u'13.2', 20.835541965616056))
(u'84310', (u'2013-07-04', u'07:00:00', u'13.8', 20.835541965616056))
(u'84310', (u'2013-07-04', u'08:00:00', u'14.6', 20.835541965616056))
(u'84310', (u'2013-07-04', u'09:00:00', u'15.5', 20.835541965616056))
(u'84310', (u'2013-07-04', u'10:00:00', u'16.9', 20.835541965616056))
(u'84310', (u'2013-07-04', u'11:00:00', u'18.5', 20.835541965616056))
(u'84310', (u'2013-07-04', u'12:00:00', u'19.1', 20.835541965616056))
(u'84310', (u'2013-07-04', u'13:00:00', u'19.1', 20.835541965616056))
(u'84310', (u'2013-07-04', u'14:00:00', u'18.3', 20.835541965616056))
(u'84310', (u'2013-07-04', u'15:00:00', u'18.1', 20.835541965616056))
(u'84310', (u'2013-07-04', u'16:00:00', u'19.5', 20.835541965616056))
(u'84310', (u'2013-07-04', u'17:00:00', u'19.1', 20.835541965616056))
(u'84310', (u'2013-07-04', u'18:00:00', u'19.2', 20.835541965616056))
(u'84310', (u'2013-07-04', u'19:00:00', u'18.2', 20.835541965616056))
(u'84310', (u'2013-07-04', u'20:00:00', u'17.8', 20.835541965616056))
(u'84310', (u'2013-07-04', u'21:00:00', u'16.7', 20.835541965616056))
(u'84310', (u'2013-07-04', u'22:00:00', u'15.6', 20.835541965616056))
(u'84310', (u'2013-07-04', u'23:00:00', u'15.2', 20.835541965616056))
```

The results from multiply kernel are dramatically different from previous results. We have three kernels here and the ideal situation is: only the data are closed(in three aspects: hour distance date) to target input should take important sits. However, in the sum kernel, if one sample is only closed to target input on the hour level but both the location and date are very far away, this sample would still make certain sense(e.g. 0.8+0.001+0.001 = 0.802),where as in the multiply kernel, the same sample can only have the weight 0.8*0.001*0.001 = 0.00008. So, the multiply kernel can help us select the 'real' closed samples.

APPENDIX

```
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

def ker_loca(dis,h_dis):
    return exp(-dis**2/h_dis**2)

def ker_day(day1,day2,h_day):
    d1 = datetime.strptime(day1,"%Y-%m-%d")
    d2 = datetime.strptime(day2,'%Y-%m-%d')
    day_dis = abs((d1-d2).days)%365 + 0.2*abs((d1-d2).days)/365
    return exp(-day_dis**2/h_day**2)

def ker_hour(t1,t2,h_hour):
    time1_h = int(t1[0:2])
    time1_m = int(t1[3:5])
    time2_h = int(t2[0:2])
    time2_m = int(t2[3:5])
    time1 = time1_h*60+time1_m
    time2 = time2_h*60+time2_m
    t_dis = abs(time1-time2)/60
    return exp(-t_dis**2/h_hour**2)


def predict(time,data,mode = 'sum'):
    kernel_value = data.map(lambda x:(x[0],x[1],ker_hour(time,x[2],h_time),float(x[3])))
    # sum
    #kernel_value = kernel_value.map(lambda x:((x[0]+x[1]+x[2])*x[3],(x[0]+x[1]+x[2])))
    # multiple
    kernel_value = kernel_value.map(lambda x:((x[0]*x[1]*x[2])*x[3],(x[0]*x[1]*x[2])))

    kernel_value = kernel_value.reduce(lambda a,b:(a[0]+b[0],a[1]+b[1]))
```

```
    res = kernel_value[0]/kernel_value[1]
    return res

h_distance = 40# Up to you
h_date = 10# Up to you
h_time = 2# Up to you
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-07-04" # Up to you
int_date = int(date[0:4]+date[5:7]+date[8:10])

stations = sc.textFile("BDA/input/stations.csv")
lines_stations = stations.map(lambda line:line.split(";"))
stations = lines_stations.map(lambda x:(x[0],(haversine(b,a,float(x[4]),float(x[3])))))
m=sc.parallelize(stations.collect()).collectAsMap()
stations_data = sc.broadcast(m)

# now we have the station number and their distance to target location
# (u'102170', 6234.382614181623)

temps = sc.textFile('BDA/input/temperature-readings.csv')
lines_temps = temps.map(lambda line:line.split(";"))
# [u'102170', u'2013-11-01', u'06:00:00', u'6.8', u'G']
# check the distance kernel
#temp = lines_temps.map(lambda x:(x[0],stations_data.value[x[0]],ker_loca(stations_data.value[x[0]],h_distance))).distinct().sortBy(ascending = True, keyfunc=lambda k: k[2])
temp = lines_temps.map(lambda x:(x[0],(x[1],x[2],x[3],stations_data.value[x[0]])))
# (u'102170', (u'2013-11-01', u'06:00:00', u'6.8', 234.382614181623))
temp = temp.filter(lambda x: int(x[1][0][0:4]+x[1][0][5:7]+x[1][0][8:10])<int_date)
# (u'102190', (u'1955-09-08', u'12:00:00', u'17.5', 243.523599180525))

# get the data which are very close to target input to evaluate the result.
# temp = temp.filter(lambda x: int(x[1][0][0:4]+x[1][0][5:7]+x[1][0][8:10])==int_date)
# temp = temp.filter(lambda x: x[1][3]<100).sortBy(ascending = True, keyfunc=lambda k: k[1][3])

data = temp.map(lambda x:(ker_loca(x[1][3],h_distance),ker_day(x[1][0],date,h_date),x[1][1],x[1][2])).cache()

# check day kernel
#data = temp.map(lambda x:(x[1][0],ker_day(x[1][0],date,h_date))).distinct().sortBy(ascending = True, keyfunc=lambda k: k[1])
#(0.013150606009360008, 0.027051846866350416, u'18:00:00', u'14.2')
# check hour kernel
#data = data.map(lambda x:(x[2],ker_hour('12:00:00',x[2],h_time))).distinct().sortBy(ascending = True, keyfunc=lambda k: k[1])

#temp.saveAsTextFile("BDA/output")
# Your code here
time_list = ["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"]

res = [predict(x,data) for x in time_list]
res = sc.parallelize([res])
res.saveAsTextFile("BDA/output")
```