BDA3

vinbe289, omkbh878

May 25, 2019

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_rev2")

stations = sc.textFile("/user/x_vinbe/data/stations.csv") \
               .map(lambda line: line.split(";")) \
               .map(lambda obs: (obs[0], (float(obs[3]),float(obs[4])))) \
               .collect()

stationsDict = {}
for s in stations:
    stationsDict[s[0]] = s[1]

stationsBraodcast = sc.broadcast(stationsDict)

temperatures = sc.textFile("/user/x_vinbe/data/temperature-readings.csv") \
                   .map(lambda line: line.split(";")) \
                   .map(lambda l: (l[0], (str(l[1]), str(l[2]), float(l[3]))))

temperatures.cache()

def GaussianKernal(distance, h):
    res = distance/h
    return exp(-res**2)

def haversine(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

def timeKernal(time):
    result = []
    if hasattr(time, '__iter__'):
        for x in time:
```

```
            if x <= -12:
                result.append(24 + x)
            else:
                result.append(x)
    else:
        if time <= -12:
            result = 24 + time
        else:
            result = time
    return result

def HourKernal(time1, time2):
    h_delta = datetime.strptime(time1, '%H:%M:%S')-datetime.strptime(time2, '%H:%M:%S')
    t_diff = h_delta.total_seconds()/3600
    time_result = timeKernal(t_diff)
    return time_result

def DayKernal(day1, day2):
    d_delta = datetime.strptime(day1, '%Y-%m-%d')-datetime.strptime(day2, '%Y-%m-%d')
    return d_delta.days

def CombineFunction(x):
    s_vals = list(stationsBraodcast.value[x[0]])
    vals = list(x[1])
    vals.extend(s_vals)
    return (x[0],tuple(vals))

def kernelFunction(predict, data):
    distance = (6, 7, 100)
    result = list()
    for p in predict:
        temp = data \
            .filter(lambda x: \
                datetime.strptime(x[1][0], '%Y-%m-%d') < \
                datetime.strptime(p[1], '%Y-%m-%d')) \
            .map(lambda x: \
                (x[1][2], (HourKernal(p[0],x[1][1]),DayKernal(p[1], x[1][0]),haversine(lon1 = p[2], \
                                lat1 = p[3], lon2 = x[1][4], lat2 = x[1][3])))) \
            .map(lambda (temp, (distTime, distDays, distKM)): \
                (temp,(GaussianKernal(distTime, h = distance[0]), \
                        GaussianKernal(distDays, h = distance[1]), \
                        GaussianKernal(distKM, h = distance[2])))) \
            .map(lambda (temp, (ker1, ker2, ker3)): (temp,ker1 + ker2 + ker3)) \
            .map(lambda (temp, kerSum): (temp, (kerSum, temp*kerSum))) \
            .map(lambda (temp, (kerSum, tkSum)): \
                (None, (kerSum, tkSum))) \
            .reduceByKey(lambda (kerSum1, tkSum1), (kerSum2, tkSum2): \
                (kerSum1 + kerSum2, tkSum1 + tkSum2)) \
            .map(lambda (key,(sumKerSum, sumTkSum)): \
                (float(sumTkSum)/float(sumKerSum)))
        result.append((p[0], temp.collect()))
    return result
```

```
train = temperatures.map(lambda l: CombineFunction(l))
predict = (('04:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('06:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('08:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('10:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('12:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('14:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('16:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('18:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('20:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('22:00:00', '2014-09-02',float(15.621373), float(58.410807)),
            ('00:00:00', '2014-09-02',float(15.621373), float(58.410807)))

resultKernel = kernelFunction(predict, train)
resultKernel_rdd = sc.parallelize(resultKernel).repartition(1).saveAsTextFile("BDA3_rev2")
```

Output

```
('12:00:00', [5.738327656873799])
('06:00:00', [4.39726221018973])
('20:00:00', [5.254604178670104])
('18:00:00', [5.5032793018222925])
('08:00:00', [4.899234930371564])
('10:00:00', [5.395594531640871])
('14:00:00', [5.845532723118549])
('22:00:00', [5.073864745050692])
('04:00:00', [4.030567181516501])
('00:00:00', [4.014042431799903])
('16:00:00', [5.73582285250762])
```

Questions

1) 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.

Width for the distance kernel is chosen to be 100 kilometers since it is reasonable to give more weightage to the stations which are within the radius of 100 kilometers from the point of interest. Width for the date kernel is chosen to be 7 days since it is reasonable to consider temperature measurements of 7 days i.e. a week prior to the date of interest to contribute more to prediction of temperature on the particular day of interest as the weather remains almost the same over a week. Width for the time kernel is chosen to be 6 hours since it seems to be reasonable to consider prior 6 hours to contribute more towards the prediction of temperature at a particular point of time as i have considered 1/4th part of the day.

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.

It is quite evident from the output that the predicted temperatures do not differ much from one another. The reason for this being that the three Gaussian kernels are independent of one another and the additive kernel is being used. A multiplicative kernel, three Gaussian Kernels can multiplied with each other, can be used instead of additicve kernel to obtain a better result.