《时空数据处理与组织课程实习》
实习报告

学　　院:　　遥感信息工程学院

班　　级:　　2006

学　　号:　　2020302131249

姓　　名:　　马文卓

实习地点:　　101 机房

指导教师:　　李晓雷

2022 年　　5 月　　30 日

一. 实验目的

通过实际问题的运用熟悉 spark 大数据框架下的 rdd（rdd 的创建、相关操作函数、格式转换等）、dataframe 对象（创建、运行机制、操作函数等）的操作，以及 sparksql、window 等工具的使用。真正意义上的使用大数据平台去处理实际问题，做到学有所用，将知识融会贯通。

二. 实验环境

Python 版本：3.6.13（conda 虚拟环境 BG）

第三方库：pyspark、findspark、psutil、math 等

系统环境：Window10

综上，即在 Win10 系统下搭建 Spark 环境，使用 Python 语言进行编程解决问题。

三. 实验内容和步骤

1. 实验内容

本次实习的内容概括为分析车辆的轨迹。

1.1 实验数据

这里我们采用的 GPS 轨道数据集是来自微软亚洲研究院的 GEOlife 项目，该数据收集了 2007 年 4 月到 2012 年 8 月的一些交通轨道数据，由一个一个点组成。我们实验采用的是第 170 号数据的第一个点集，编号为 20080428112704，具体数据格式如下：

Line 1…6 are useless in this dataset, and can be ignored. Points are described in following lines, one for each line.

Field 1: Latitude in decimal degrees.

Field 2: Longitude in decimal degrees.

Field 3: All set to 0 for this dataset.

Field 4: Altitude in feet (-777 if not valid).

Field 5: Date - number of days (with fractional part) that have passed since 12/30/1899.

Field 6: Date as a string.

Field 7: Time as a string.

Note that field 5 and field 6&7 represent the same date/time in this dataset. You may use either of them.

Figure 1:Data Fromat

1.2 具体内容

（1）车辆速率计算

从原始轨迹数据中获取每个轨迹点的经纬度坐标，通过该轨迹点与前一个点的距离和两个轨迹点的时间差，计算该点的即时速率。根据实际意义，规定起点和终点的即时速率为零。

（2）车辆停留点分析

停留点识别的算法描述如下：

1） 遍历轨迹点，检查速率值，若小于阈值，则将上一个点作为停留开始点，记录其序号。

2） 继续遍历轨迹点，只要轨迹点的速率值仍小于阈值，则将该点作为停留点。

（3）车辆加减速分析

加速和减速都是持续性的过程，而突发的速率变化则可能是数据采集或预处理阶段的误差引起的正常波动，因此加/减速检测方法可以采用寻找至少连续 N（$N \geqslant 2$）个点速率单调变化的片段。

2. 实验步骤

本次实验的主要步骤为：数据读取与预处理->车辆瞬时速度计算->停留点判断->运动状态分析->结果保存。具体实现过程如下：

2.1 数据读取与预处理

在进行实验之前，首先指定编码格式为 utf-8，初始化 spark 环境，并且导入相关第三方库。具体如下：

```
#coding=utf-8
import findspark
findspark.init()

from pyspark import SparkConf
from pyspark.sql import SparkSession
from pyspark.sql.types import IntegerType, BooleanType, FloatType, StringType
from pyspark.sql import Row
from pyspark.sql.functions import udf
from pyspark.sql import functions as F
from pyspark.sql.window import Window
import math
```

本次实验的数据读取和数据预处理封装在函数 read_data 中，调用该函数可以返回经过预处理的 DataFrame 格式数据。首先创建 sparkSession 对象，使用其 textFile 函数读取相关数据，得到 RDD 格式的数据 data_rdd。然后进行预处理：首先使用 filter 函数过滤表头和异常数据（即限定按逗号分隔后的长度为 7），然后以逗号为分隔符进行数据分割，再使用 zipWithIndex 函数加上索引 id，最后将每行数据加上表头生成 Row 对象（这里我将前四列数据格式转为 float 类型，方便后面计算），转为 DataFrame 格式的 data。

```
def read_data():#数据读取
    spark=SparkSession.builder.config(conf=SparkConf()).getOrCreate()#创建SparkSession对象
    #读入数据，存储到Rdd中
    data_rdd=spark.sparkContext\
        .textFile('.\\data\\Geolife Trajectories 1.3\\Data\\170\\Trajectory\\20080428112704.plt')
    #数据处理（除去表头和异常数据，按逗号为分隔符处理数据,数值化,转为DataFrame）
    data=data_rdd.filter(lambda line:len(line.split(','))==7)\
            .map(lambda line:line.split(','))\
            .zipWithIndex()\
            .map(lambda line : Row(id = int(line[1]), latitude = float(line[0][0]),
            longitude = float(line[0][1]), time= float(line[0][4]),time_str=line[0][6]))\
            .toDF()
    return data
```

编写好函数后，在主体框架中调用即可。

```
#读取数据(此时数据处理已经完成),格式是DataFrame
data=read_data()
```

2.2 瞬时速度计算

瞬时速度的计算公式即：速度=距离/时间间隔。因此这个部分我们需要经历三个步骤：距离计算、时

间间隔计算、速度计算、对 DataFrame 进行计算生成 speed 列。

（1）距离计算：由于我们的初始数据给的是经纬度坐标，因此我们需要从经纬度坐标中计算实际距离。这里我使用的是 Haversine 公式，公式具体如下：

$$d = 2r\,\mathrm{argsin}(\sqrt{\sin^2(\frac{lat2-lat1}{2}) + \cos(lat2)\cos(lat1)\sin^2(\frac{lon2-lon1}{2})})$$

Figure 2:Haversine

按照如上公式，编写获得实际距离的函数 get_distance。这里地球半径取值为 6371.393km（越精确越好），最后结果为 km 单位，转为 m 返回。

```
def get_distance(lat1,lon1,lat2,lon2):#计算两经纬坐标距离
    R = 6371.393#地球平均半径
    Pi = math.pi#圆周率
    #Haversine公式
    a = (math.sin(math.radians(lat1/2-lat2/2)))**2
    b = math.cos(lat1*Pi/180) * math.cos(lat2*Pi/180) * (math.sin((lon1/2-lon2/2)*Pi/180))**2
    L = 2 * R * math.asin((a+b)**0.5)#单位是千米
    return 1000*L#单位为m
```

（2）时间间隔计算：由于数据中既以时间戳的形式给出了时间，由以距离 1899.12.30 日的天数的形式给出时间，因此有了两种计算时间间隔的方法。这里我选择后者计算时间间隔，虽然最后的结果与前者计算有小的差异（小数点后四位开始有差异），但并不影响分析。

编写 get_time 函数返回时间间隔，单位为 s。

```
def get_time(time1,time2):#获取时间间隔
    return 24.0*60.0*60.0*(time2-time1)#单位是s
```

（3）瞬时速度计算：利用前面两个函数获取相邻点的距离和时间间隔，将两者相除，即可得到瞬时速度，单位为 m/s。值得注意的是，根据实际意义，我们将第一点和最后一点的速度设为 0m/s。

```
def get_speed(id,lat1,lon1,lat2,lon2,time1,time2):#计算一点的即时速度
    if id==int(0) or id==int(num-1):#如果是第一行或者最后一行
        return 0.0#速度为0
    time=get_time(time1,time2)#获取时间间隔，单位为s
    distance=get_distance(lat1,lon1,lat2,lon2)#获取距离，单位为千米
    speed=distance/time#计算速度=距离/时间间隔(m/s)
    return speed
```

（4）对 DataFrame 进行计算生成 speed 列：现在已经可以计算每个点的瞬时速度，可以对 data 进行计算。使用 sparksql 中的 window 函数，可以获取 DataFrame 格式中每行前（lag 函数）后（lead 函数）的数据。首先使用 Window 创建窗口（不分类，按照 id 排序），具体如下：

```
#创建窗口
my_window = Window.partitionBy().orderBy("id")#不分组，按id排序
```

然后使用 udf（用户定义函数），将 get_speed 函数的返回类型限定为浮点型构成自定义函数 GET_SPEED。最后使用 withColumn 函数增加 speed 列，其值为 GET_SPEED 函数的运算结果。

```
#计算每一个点的即时速度
num=data.count()#数据行数
GET_SPEED=udf(get_speed,FloatType())
data=data.withColumn('speed',GET_SPEED(
        F.col("id"),
        F.lag("latitude",1).over(my_window),F.lag("longitude",1).over(my_window),
        F.col("latitude"),F.col("longitude"),
        F.lag("time",1).over(my_window),F.col("time")
        ))
```

2.3 停留点判断

停留点判断方法如下：

①停留开始点：当前点的速度大于（等于）速度阈值，下一点的速度小于阈值

②停留点：当前点速度小于速度阈值

根据以上停留点判断方法，编写 is_stoppoint 函数判断停留点，取速度阈值为 1m/s，若是停留点则表上序号，若不是停留点则返回 0。

```
stop_id=0#停留点序号
threshold=1.0#速度阈值设为1m/s
def is_stoppoint(id,speed1,speed2):#标记是否为停留点
    #speed1为当前时刻速度,speed2下一时刻速度,threshold为停留阈值
    global stop_id
    if id==int(num-1):#如果是最后一行
        if(speed1<threshold):
            return int(stop_id)
        else:
            return int(0)
    if(speed1<threshold):#当前时刻速度小于阈值一定是停留点
        return int(stop_id)
    elif(speed2<threshold):#下一时刻速度小于阈值也是停留开始点
        stop_id=stop_id+1#停留点+1
        return int(stop_id)
    else:#其他情况
        return int(0)
```

在主程序中和速度求解类似，首先将 is_stoppoint 返回类型限定为 Integer，包装为 udf 函数 GET_STOP。使用 withColumn 函数增加 stop_point 列，使用 GET_STOP 函数获取其值。

```
#停留点判断
GET_STOP=udf(is_stoppoint,IntegerType())
data=data.withColumn('stop_point',GET_STOP(
        F.col('id'),F.col("speed"),F.lead("speed",1).over(my_window)
        ))
```

2.4 运动状态分析

由题目意思可以得知，运动状态的判断需要连续 N 个区间的速度单调变化，才可认为是加速或者减速。为了判断方便，我们首先求解每个点的加速度（实际上是前一个点到当前点的平均加速度）。其公式为 a=（v2-v1）/（t2-t1），根据公式编写 get_acceleration 函数，其中设定第一个点的加速度为 0，函数返回值单位为 m/s^2 如下：

```
def get_acceleration(id,speed1,speed2,time1,time2):#获取相邻两点间加速度
    if id==int(0):#如果是第一行
        return 0.0#加速度为0
    time=get_time(time1,time2)
    d_speed=speed2-speed1#速度之差
    return d_speed/time#速度增量除时间=加速度(m/s2)
```

在主程序中，使用上面类似的方法添加 acceleration 列。

```
#加速度计算
GET_ACC=udf(get_acceleration,FloatType())
data=data.withColumn('acceleration',GET_ACC(
        F.col("id"),
        F.lag("speed",1).over(my_window),F.col("speed"),
        F.lag("time",1).over(my_window),F.col("time")
        ))
```

然后编写状态判断函数 stop_state。这里取 N=3，即从当前点到下下个点三个点的速度连续变化才可认为当前点为加速或者减速。由于之前计算的加速度为上一点到当前点的平均加速度，因此这里判断当前点的运动状态需要的是下一点的加速度（a1）和下下点的加速度（a2），只要 a1、a2>0 则加速，a1、a2<0 则减速，如果是停留点则标志 stop，如果是最后一个点则标志 destination，如果连续三点速度变化不单调，则标志位 tansition。对于倒数第二个点，由于其后面仅一个点，因此判断两个点的速度是否连续变化即可。具体代码如下：

```
def spot_state(id,stop_point,a1,a2):#判断运动状态
    #这里取N=3，即要在至少后面N-1个点速度单调变化才可
    #id为点号，stop_point为停留点序号
    #a1为当前点到下一点的加速度，a2为下一点到下下点的加速度
    if id==num-1:#最后一个点时
        return 'destination'
    if stop_point>0:#如果是停留点
        return 'stop'
    else:#如果不是停留点
        if id==num-2:#倒数第二点时
            if a1>0:
                return 'accelerate'
            if a1<0:
                return 'decelerate'
        if a1>0 and a2>0:
            return 'accelerate'#加速
        if a1<0 and a2<0:
            return 'decelerate'#减速
        else:
            return 'tansition'#过渡
```

类似的，使用 udf 包装为 GET_STATE 函数。然后使用 withColumn 函数添加 spot_state 列。

```
#运动状态判断
GET_STATE=udf(spot_state,StringType())
data=data.withColumn("spot_state",GET_STATE(
        F.col('id'),F.col('stop_point'),
        F.lead('acceleration',1).over(my_window),
        F.lead('acceleration',2).over(my_window)
        ))
```

2.5 结果保存

由于这里的数据为 spark dataframe 结构，并不是 pandas dataframe，所以并不能使用 to_csv 函数保存。鉴于 spark 分布式框架体系，我们使用如下方式进行保存：

```
#保存到csv中
data.coalesce(1).write.option("header", "true").csv(".\\code\\result.csv")
```

四．实验成果

本次实验成果为一个 result.csv 文件（文件可见附录）。其格式为 id（点号）、latitude（纬度）、longitude（经度）、time（时间）、time_str（时间戳）、speed（速度）、stop_point（停留点标志）、acceleration（加速度）、spot_state（运动状态）

id	latitude	longitude	time	time_str	speed	stop_point	acceleration	spot_state
0	39.975496	116.3337	39566.477	11:27:04	0	0	0	accelerate
1	39.975457	116.33375	39566.477	11:27:06	3.1927714	0	1.5963824	tansition
2	39.975394	116.33374	39566.477	11:27:07	7.0476108	0	3.8548627	decelerate
3	39.975334	116.33375	39566.477	11:27:08	6.6807942	0	-0.36681575	tansition
4	39.975359	116.33365	39566.477	11:27:10	4.2395077	0	-1.2206407	tansition
5	39.975317	116.33375	39566.477	11:27:12	4.6362443	0	0.1983687	decelerate
6	39.975368	116.33371	39566.477	11:27:14	3.1281774	0	-0.75403184	tansition
7	39.975394	116.33372	39566.477	11:27:16	1.455638	0	-0.83627135	tansition
8	39.975432	116.33374	39566.477	11:27:18	2.2335422	1	0.38895124	stop
9	39.975422	116.33375	39566.477	11:27:20	0.8154609	1	-0.7090421	stop
10	39.975414	116.33378	39566.477	11:27:22	1.313255	0	0.24889652	tansition
11	39.975396	116.33381	39566.477	11:27:24	1.6571937	0	0.1719697	decelerate
12	39.975382	116.33383	39566.477	11:27:26	1.285153	2	-0.18601993	stop
13	39.97537	116.33385	39566.477	11:27:28	0.9848042	2	-0.1501747	stop
14	39.975369	116.33388	39566.477	11:27:30	1.1092031	0	0.062199313	tansition
15	39.97538	116.3339	39566.477	11:27:32	1.2283036	3	0.05955034	stop
16	39.975379	116.33392	39566.477	11:27:34	0.8114631	3	-0.20842072	stop
17	39.975384	116.33394	39566.478	11:27:36	0.8157753	3	0.002156103	stop
18	39.975408	116.33394	39566.478	11:27:38	1.3371422	4	0.26068398	stop
19	39.975421	116.33394	39566.478	11:27:40	0.7278161	4	-0.30466244	stop
20	39.97543	116.33395	39566.478	11:27:42	0.5438709	4	-0.09197274	stop
21	39.975438	116.33395	39566.478	11:27:44	0.4763398	4	-0.0337655	stop
22	39.975447	116.33396	39566.478	11:27:46	0.5286364	4	0.026148356	stop

五．问题及解决

本次实验遇到了一些问题，具体如下：

1. **格式问题**：由于使用了窗口函数，导致操作对象全部为 column 对象，导致出现了很多问题。

（1）例如在调用计算速度函数时，由于第一行数据没有前一行，如果在函数中仍然调用前一行数据则会报错。解决方法就是在加入判断，对第一行进行特殊处理。

（2）在计算距离函数中使用 sin、cos 等一系列的数学公式时，我开始调用的是 pyspark.sql 的 functions 中的函数，但由于其操作对象是 column 而报错（当时就是说操作对象不为 column）。解决方法是调用 math 中的相关函数。

（3）最开始使用 withColumn 增加列时，会出现“返回类型不匹配”的错误，解决方法是利用 udf 将运算函数限定类型，再在 withColumn 中调用。如下：

```
GET_SPEED=udf(get_speed,FloatType())
```

2.**特殊数值处理**：比如在计算速度的时候要设置第一行和最后一行的速度为 0m/s。当时因为 dataframe 格式没有办法单独修改某个值而花费了大量的时间。最后使用 zipWithIndex 函数增加了序号列，根据序号进行判断即可特殊设置。（但是这里的问题和上面类型的问题一起遇到，导致花费了大量时间解决，但也收获颇丰）。

六．实习体会

通过本次实验，主要是复习巩固了 spark 的基础知识，将所学内容运用到实际当中。

对于本次实习的学习方式我是比较适应的，由浅入深，循序渐进，在一个一个的小练习中掌握基础知识，在最后的大实验中融会贯通。对于 spark 的 RDD 编程，我认为其更加离散化，虽然也主要是逐行操作，但是通过一些自定义的函数也可以实现数据的自由。而 spark 的 DatFrame 则更加结构化严谨化，其主要是行操作甚至窗口操作。至于 sparksql 则是一个实用额的数据增删改查的工具，将 spark 和 sqsl 语句结合从而提供更加多样化的功能。sparkML 主要是关于机器学习领域，其流水线思的操作步骤，大大简化了机器学习编程的难度。

本次实验主要是分了车辆轨迹的速度、停留点、运动状态等。事实上在事件问题中我们会碰到各种细节，为了使编程模型更加贴近现实，我们必须处理好这些细节，这也有助于我们提高思维能力和编程习惯。尽管在实验中遇到很多困难，但是在老师的耐心帮助下和自己的细心探索之下，完美的完成了实验任务。

本次实验让我初步认识了大数据处理框架，初步熟悉了大数据处理流程，为以后的学习工作奠定了良好的基础。最后感谢老师的教导，致此。