频繁轨迹挖掘实验报告

文件介绍

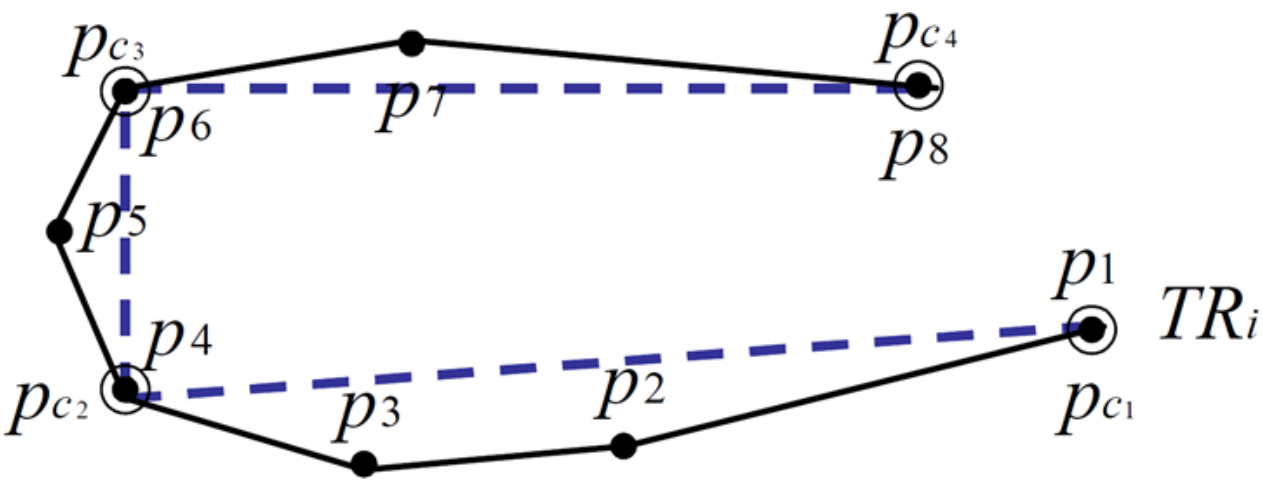
- 频繁轨迹挖掘实验报告.pdf：本文件
- 频繁轨迹挖掘.pptx：汇报所用ppt
- point.py：定义点对象及相应方法
- segmen.py：定义线段对象及相应方法
- partition.py：实现轨迹切割
- cluster.py：实现线段聚类和代表轨迹生成
- __init__.py：初始化文件
- trajCluster_test.py：测试文件
- logs.txt：日志文件

运行环境

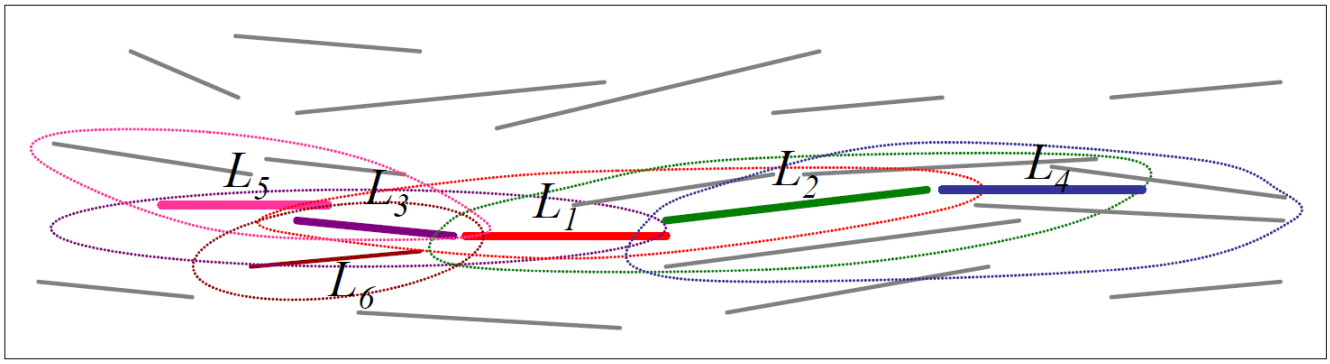
基于python2和spark

算法原理

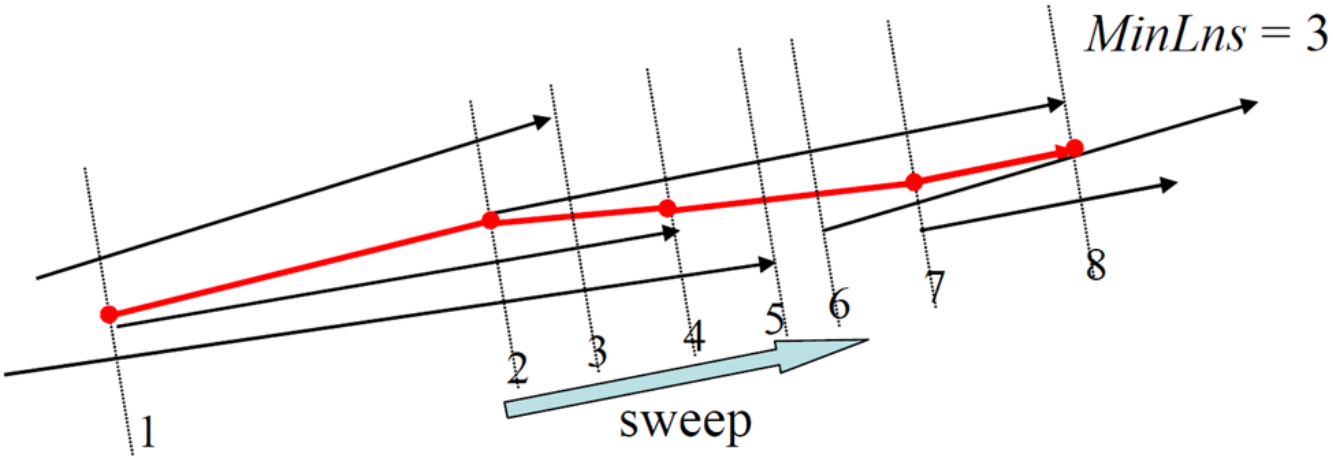
首先利用论文自己定义的线段间的距离对轨迹进行分段。



分段结束后，采用密度聚类的方法对线段进行聚类，得到线段簇。



然后对线段簇生成其代表轨迹。



分布式实现

分布式划分轨迹

我们需要对多条轨迹进行划分，所以采用spark将所有轨迹分片，每一个worker对一部分进行划分，最后将每个worker划分得到的线段集合加起来，就能得到所有轨迹的划分结果。

分布式计算邻域

计算一个线段的邻域也就是遍历所有线段，将满足条件的线段放到一个集合中。这个过程可以将所有线段分片，每一个worker遍历查找一部分满足条件的线段，然后将所有线段整合到一起就得到了邻域。

分布式计算平均方向向量

计算方向向量需要将所有向量相加，然后除以向量个数。可以将所有向量分片，一个worker计算一部分向量的和，然后将每个部分的和相加就能得到所有向量的和。

分布式变换坐标

对点进行坐标变换时可以分布式进行，即每个worker变换一部分点的坐标。

分布式坐标排序

对点进行排序可以分布式进行。

分布式对点计数

对交点的计数可以分布式进行。即一个worker对一部分计数，最后加起来即可。

分布式计算坐标和

计算交点的坐标和可以分布式进行。即一个worker计算一部分交点的坐标和，然后将各部分的坐标和加起来就得到总的坐标和。

采用数据集

1. 俄勒冈州东北部鹿群移动

我们小组其中选取了俄勒冈州东北部鹿群在一年内的移动数据，其中一条轨迹代表一头鹿的移动路线。利用频繁路径挖掘就可以得到该鹿群整体这一年经常去的地方。

https://www.fs.fed.us/pnw/starkey/data/tables/index.shtml

2. 葡萄牙城市Porto出租车轨迹

我们小组其中选取了其中一部分数据，原数据大约有一百万条轨迹，考虑时间限制和最终的图像质量，我们选取了前四千条轨迹分两部分进行处理。

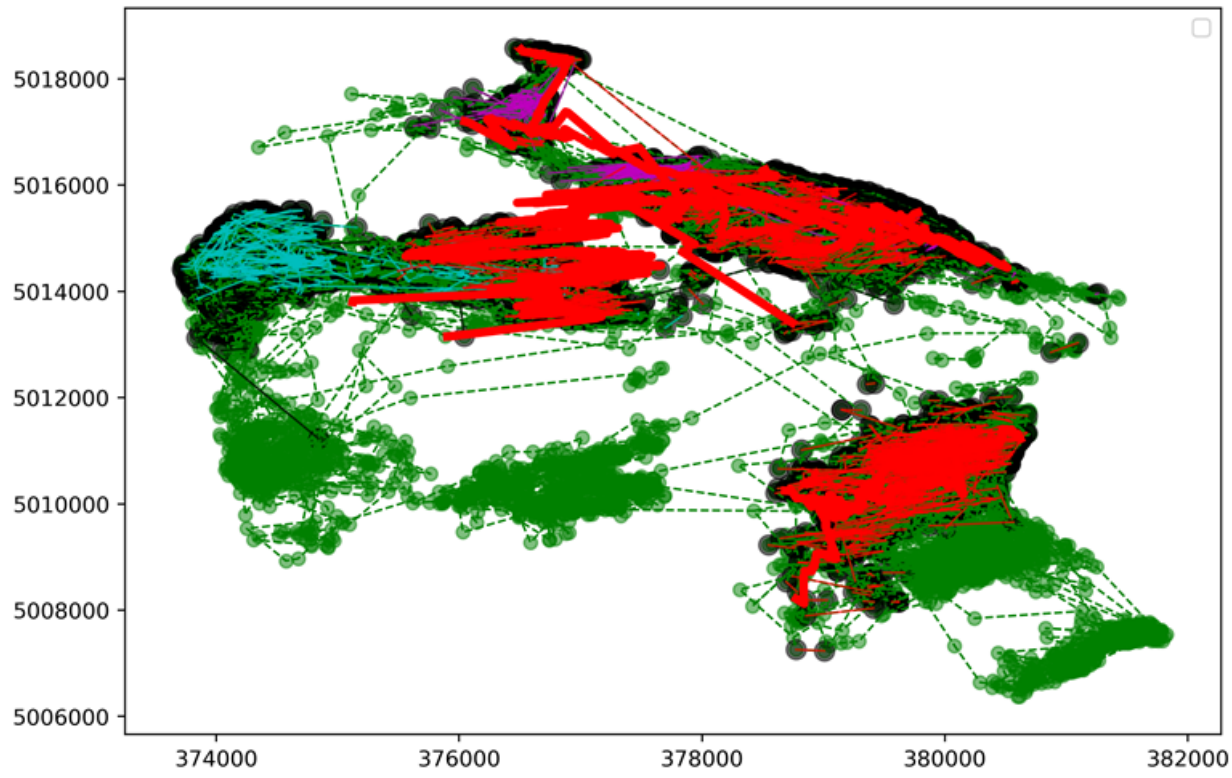
https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data

3. Demo测试数据

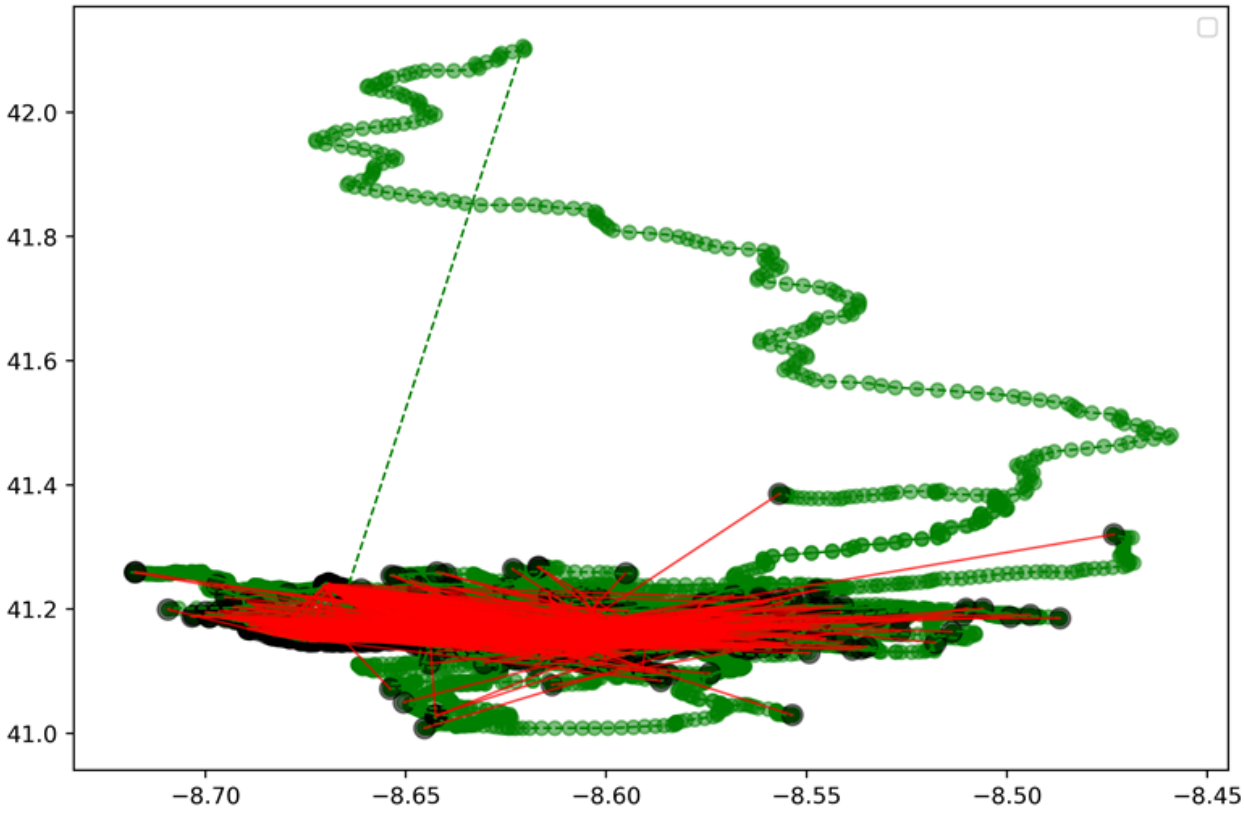
我们随机生成了16条轨迹，一共有680个点。然后对这16条轨迹进行处理。

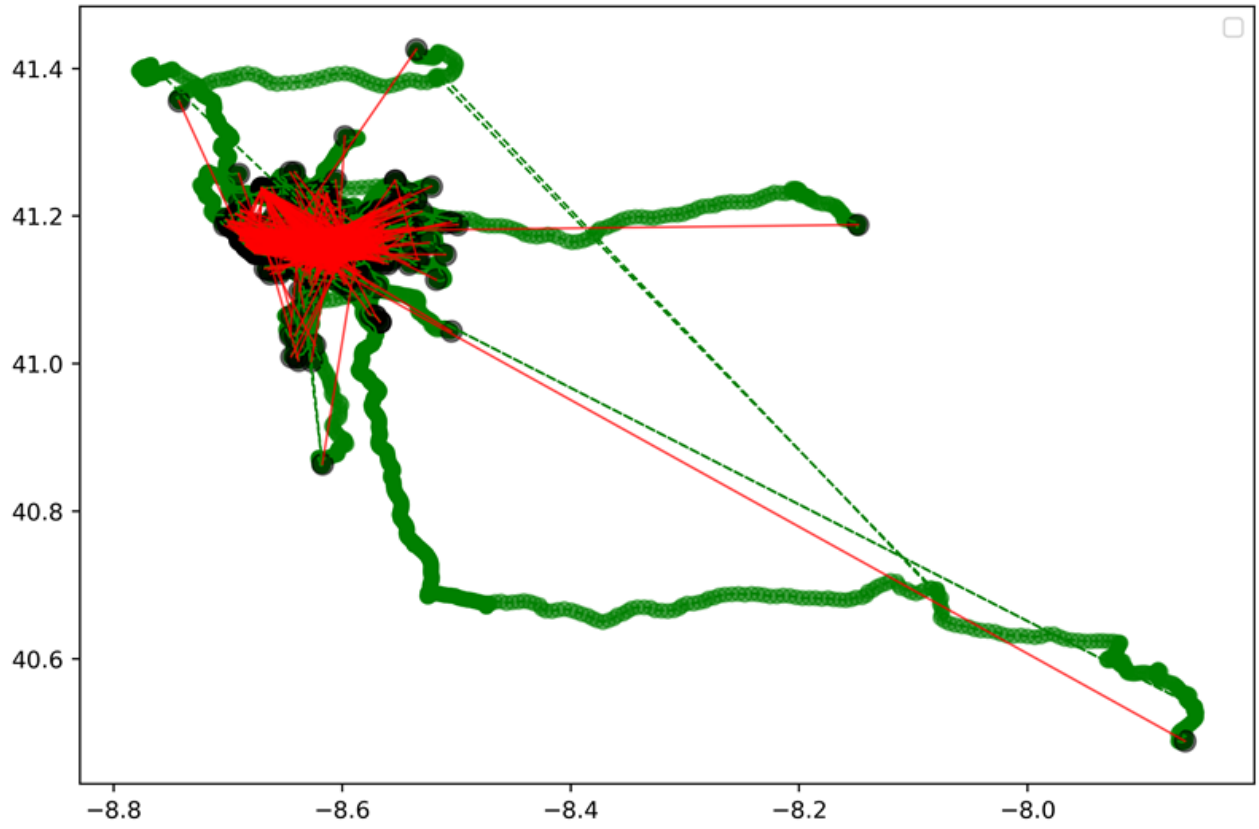
测试结果

1. 俄勒冈州东北部鹿群移动

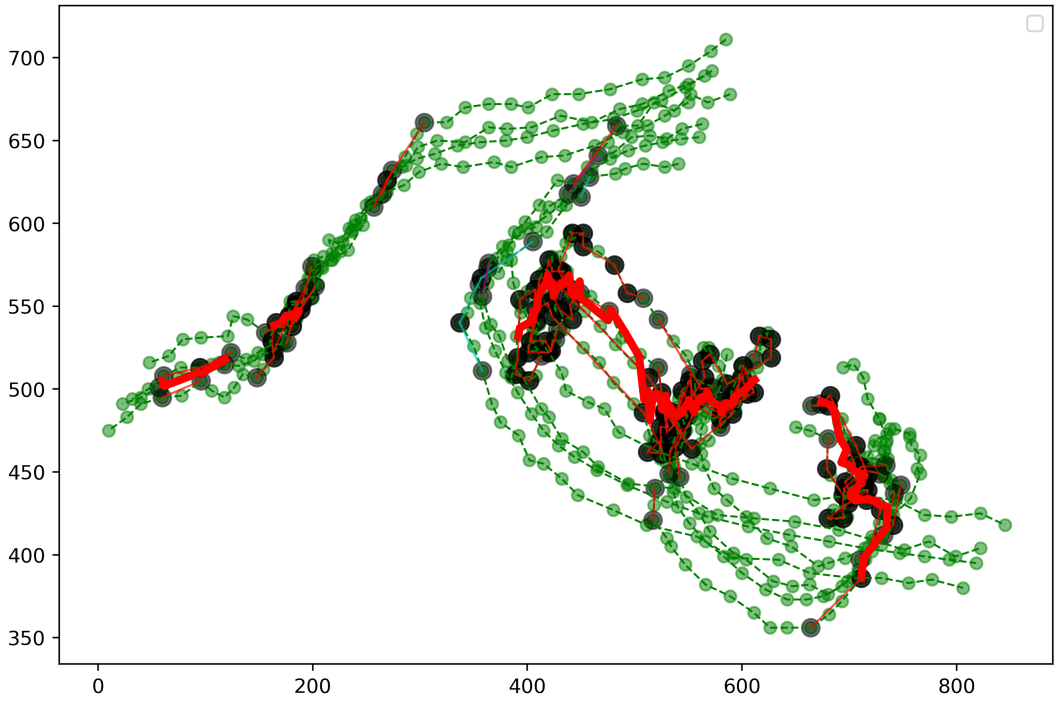


2. 葡萄牙城市Porto出租车轨迹





3. Demo测试数据



经验教训

配置过程

1. 在配置ssh免密码登录时，发现即使第一次登录之后依然需要输入密码，上网查询得知需要更改authorized_keys文件的权限为 600。
2. 重复初始化namenode。重复初始化会带来namenode和datanode节点的clusterID不同。可以将datanode的clusterID改成和namenode一样来解决这一问题。直接将hadoop打包分发，会因为一些参数完全一致带来的问题，可以将所有节点的date和name数据全部清除，然后重新初始化namenode。
3. 时钟同步问题：使用ntpdate和ntp来进行时钟同步，ntpdate和ntp在安装前需要有对应版本的libcrypto（使用的是 https://pkgs.org/ 得到的rpm的分发版本）。在安装过程中发现需要添加--force强制安装才能完成更新，之后就可以完成ntp和ntpdate的安装。时钟同步设置的同步服务器是master节点。在完成配置之后，首先手动进行时间同步，然后启动ntp服务，这样时钟同步的问题得到了解决，hadoop自带的demo可以正常使用。
4. 安装的zookee、hadoop、hbase、scala、spark都在/usr/local/cluster文件夹下，需要将cluster文件夹的所有者设置成hadoop，而且cluster文件夹内存在很多软连接，直接将文件夹通过scp分发没有办法保留软链接关系，所以将cluster文件夹通过tar打包之后分发，这样可以保留软链接关系和拥有者信息。

debug和优化

1. Spark rdd计算得到的结果是原数据的深拷贝。意味着对结果进行修改不会对原数据造成影响，但是当需要对原数据修改时就会有问题，我们在这遭遇了死循环。解决办法是赋予每个对象ID，rdd 计算结果不是数据对象，而是数据对象的ID，然后根据ID对原数据进行修改。
2. 算法一些地方需要除以某条线段的长度，而线段的长度是有可能为零的。解决办法：
    1. 对原数据进行过滤，删除长度为零的线段。
    2. 算法中进行除数检查，为零需进行特殊处理。
3. collect的时间比reduce要长，所以如果出现多次collect的情况，修改为用reduce实现算法。
4. 分片数量过大，运行时间会过长；分片数量过小，有可能会出现超过内存限制的情况。所以要多次尝试，选取最佳分片数量。
5. reduce和collect的运行次数会严重影响程序的运行时间，所以应该尽量减少其运行次数。
6. 我们出现了断开ssh连接后程序突然中途退出的情况，解决方法是利用nohup命令防止进程在账号退出时关闭。

参考资料

- https://github.com/MillerWu2014/trajectory-cluster
- https://www.cnblogs.com/qingyunzong/p/8899715.html
- http://dblab.xmu.edu.cn/blog/spark
- https://www.jianshu.com/p/dc2913a07770