数据挖掘第三次作业说明文档

姓名：洪嘉勇

学号：1452822

附件列表

- hw3_problem1.py
- hw3_problem2.py
- Fptree.py(自己手撸的)

第一问

问题概述

本次作业利用手机与基站连接信号强度的测量报告数据来预测手机所在GPS经纬度的位置。
在本问中将用到随机森林分类器和随机森林回归这两种方法，其中随机森林回归训练MR数据的未知GPS经纬度未知（即GPS经纬度数值坐标）；而随机森林分类器则训练MR数据“信号强度特征”和“对应GPS纬度所在栅格的”分类模型、预测MR测试数据未知GPS经纬度位置所在的栅格。

算法详解

随机森林分类

将决策树作为最基本的评估单元，然后用数据去训练由一堆决策树组成的随机森林，然后对这些决策树进行分类。一般我们用**Expected (classwise) error rate**来评估分类的准确率。

随机森林回归

随机森林回归跟随机森林分类原理相近，也是用数据去训练一堆决策树构建的随机森林，然后进行回归，一般我们用**Expected mean squared error**来预测随机森林回归的准确率。

特征选择

在使用随机森林之前，我们必须要在数据集中选取合适的特征，这样我们在训练的时候才会有意义。在本题的要求中，已经给出了特征即手机的信号强度，但手机的信号强度特征并没有直接给出数据，而是要从现有数据中进行求算。求算并不是很难。

```
# prepare data
for i in range(1, 7):
    self.train['RSSI_' + str(i)] = abs(self.train['RSCP_' + str(i)] - self.train['E
cNo_' + str(i)])
    self.test['RSSI_' + str(i)] = abs(self.test['RSCP_' + str(i)] - self.test['EcNo
_' + str(i)])
```

得到数据之后，我最后选取信号强度和每条数据的唯一标识id作为我们的特征值。

```
self.tests = self.total.ix[:, [u'SRNCID', u'BestCellID', u'RSSI_1', u'RSSI_2', u'RS
SI_3', u'RSSI_4', u'RSSI_5', u'RSSI_6']]
```

结论和分析

在本题中，我随机生成了10次训练集和测试集，比例分别是80%和20%，然后分别用随机森林回归和随机森林分类训练并预测了10次。

```
self.classifier_train_x, self.classifier_test_x, self.classifier_train_y, self.clas
sifier_test_y = train_test_split(self.tests, self.classifier_y, test_size=0.2)
self.regressor_train_x, self.regressor_test_x, self.regressor_train_y, self.regress
or_test_y = train_test_split(self.tests, self.regressor_y, test_size=0.2)

self.regressor.fit(self.regressor_train_x, self.regressor_train_y)
self.classifier.fit(self.classifier_train_x, self.classifier_train_y)

regressor_res = self.regressor.predict(self.regressor_test_x)
classifier_res = self.classifier.predict(self.classifier_test_x)
```

得到了2G和4G数据的回归和分类随机森林的准确率。同时我们将预测的结果逆向计算求得它们所在的经纬度，与真实的坐标进行比对，计算它们之间的欧式距离。这里我所用的欧式距离比较简单，只是简单的经纬度差的平方和再开根。

```
    def distance(self, lo1, la1, lo2, la2):
        '''
        calculate distance
        :param lo1: longitude1
        :param la1: latitude1
```

```
        :param lo2: longitude2
        :param la2: latitude2
        :return:distance
        '''
        dlon = lo2 - lo1
        dlat = la2 - la1
        return math.sqrt(dlon * dlon + dlat * dlat)
```

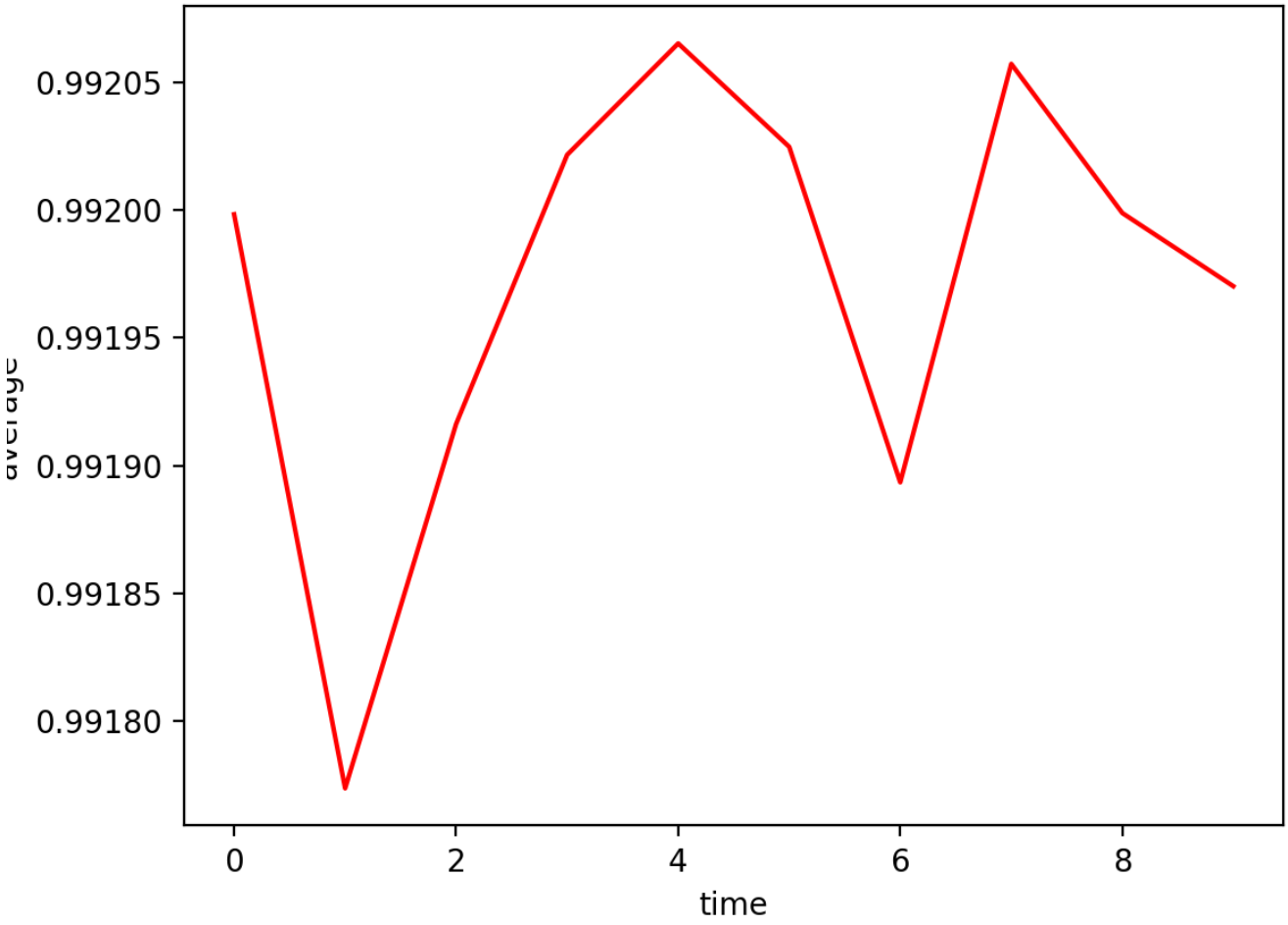
最后得到结论如下：

2G

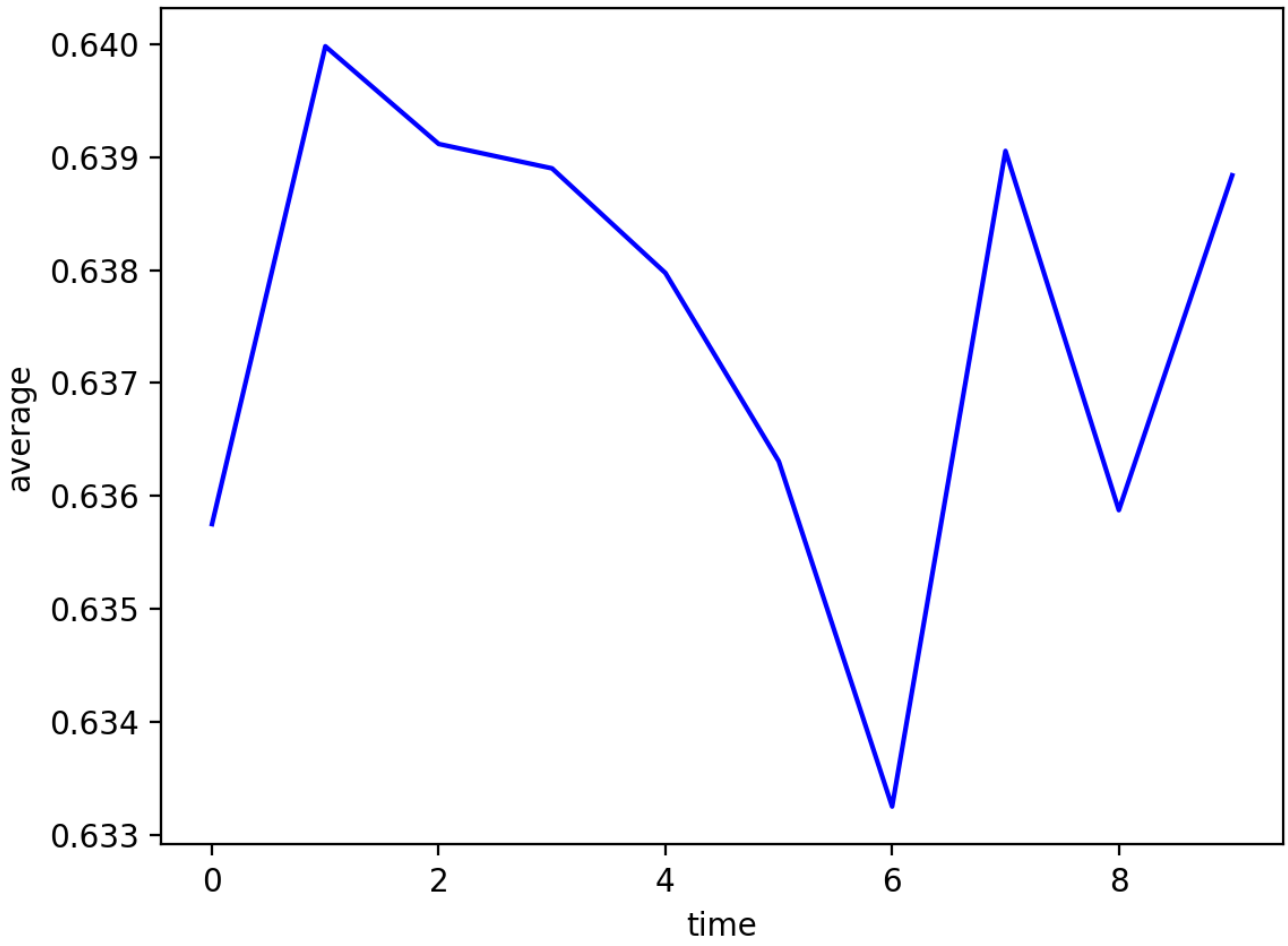
准确率

序号	回归	分类
1	0.991940532359	0.641432540908
2	0.991940532359	0.640876813831
3	0.991940532359	0.642297005249
4	0.991940532359	0.642914479778
5	0.991940532359	0.641803025625
6	0.991940532359	0.642420500154
7	0.991940532359	0.642574868787
8	0.991940532359	0.643408459401
9	0.991940532359	0.641031182464
10	0.991940532359	0.641988267984

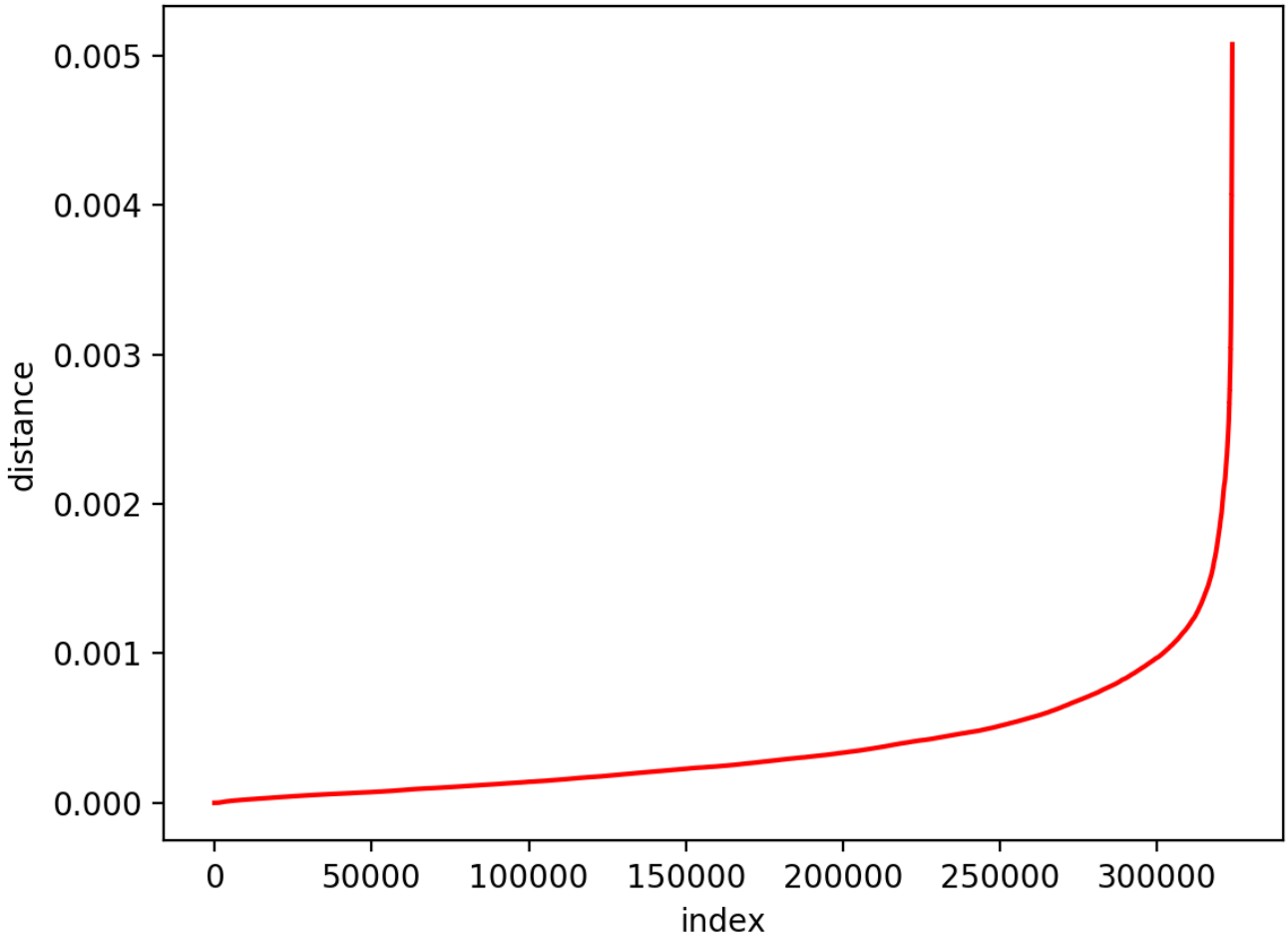
2G回归平均误差图



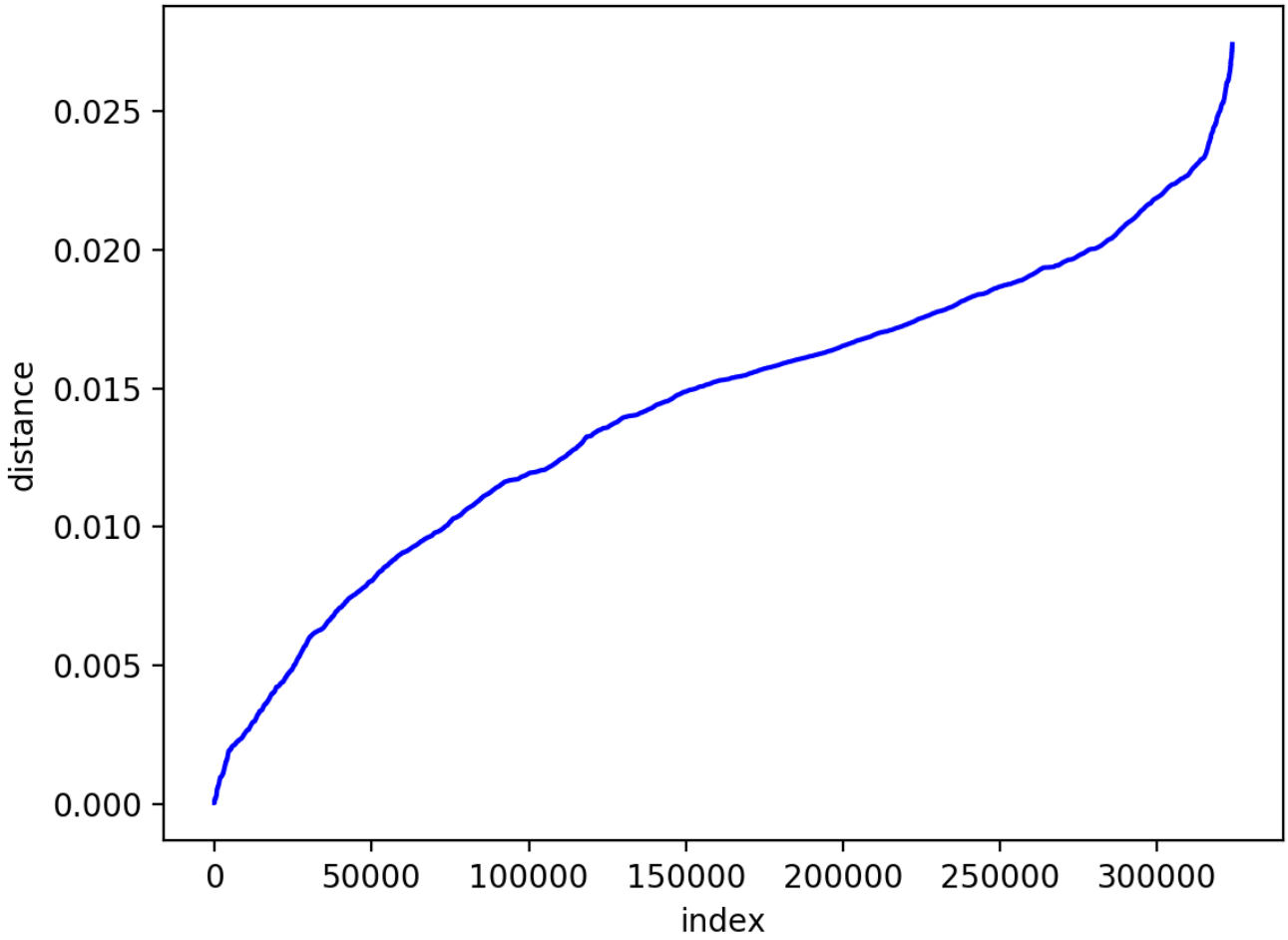
2G分类平均误差图



2G回归误差分布图


0.005
0.004
0.003
0.002
0.001
0.000
distance
0
50000
100000
150000
200000
250000
300000
index

2G分类误差分布图



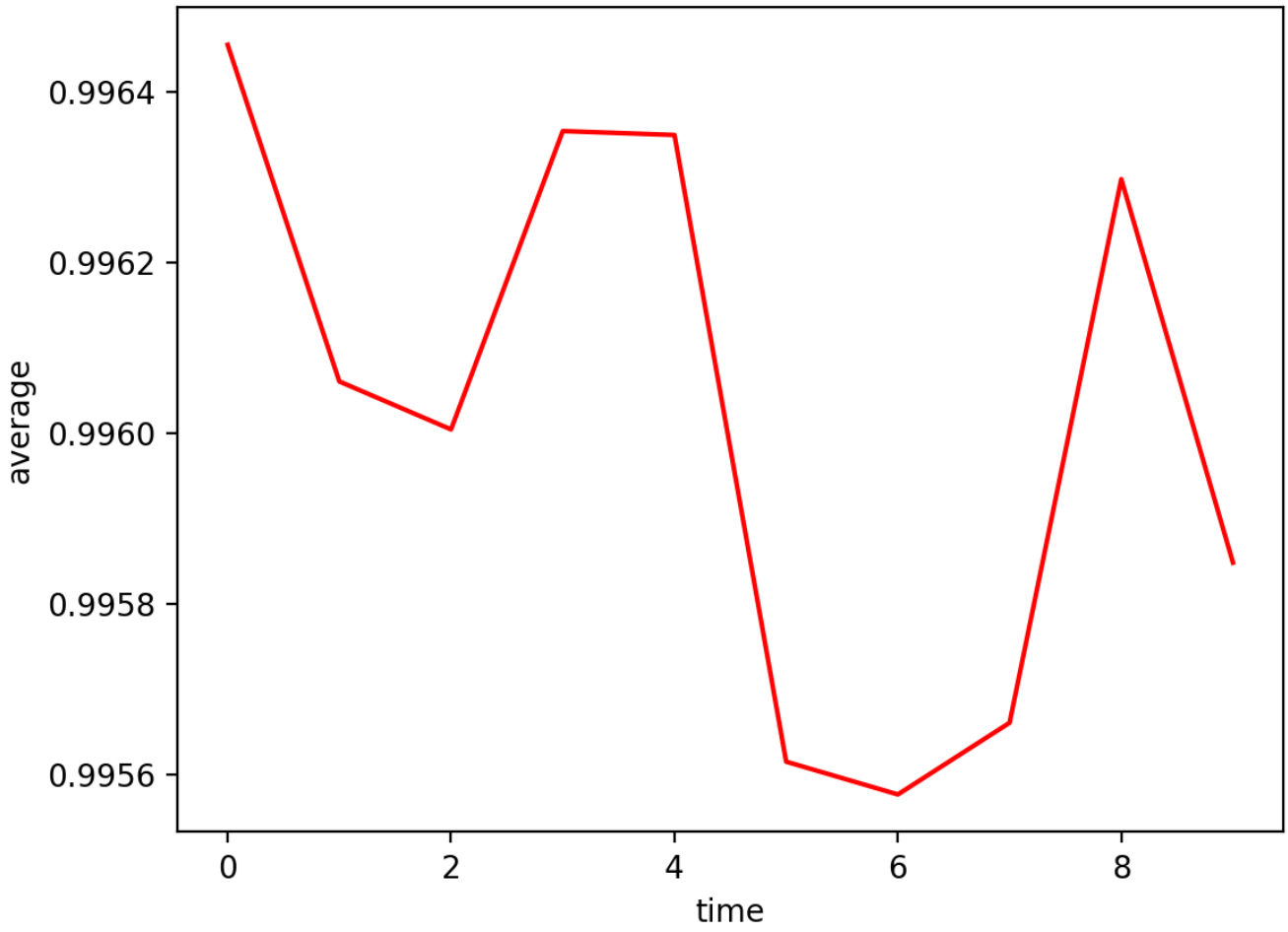
4G

准确率

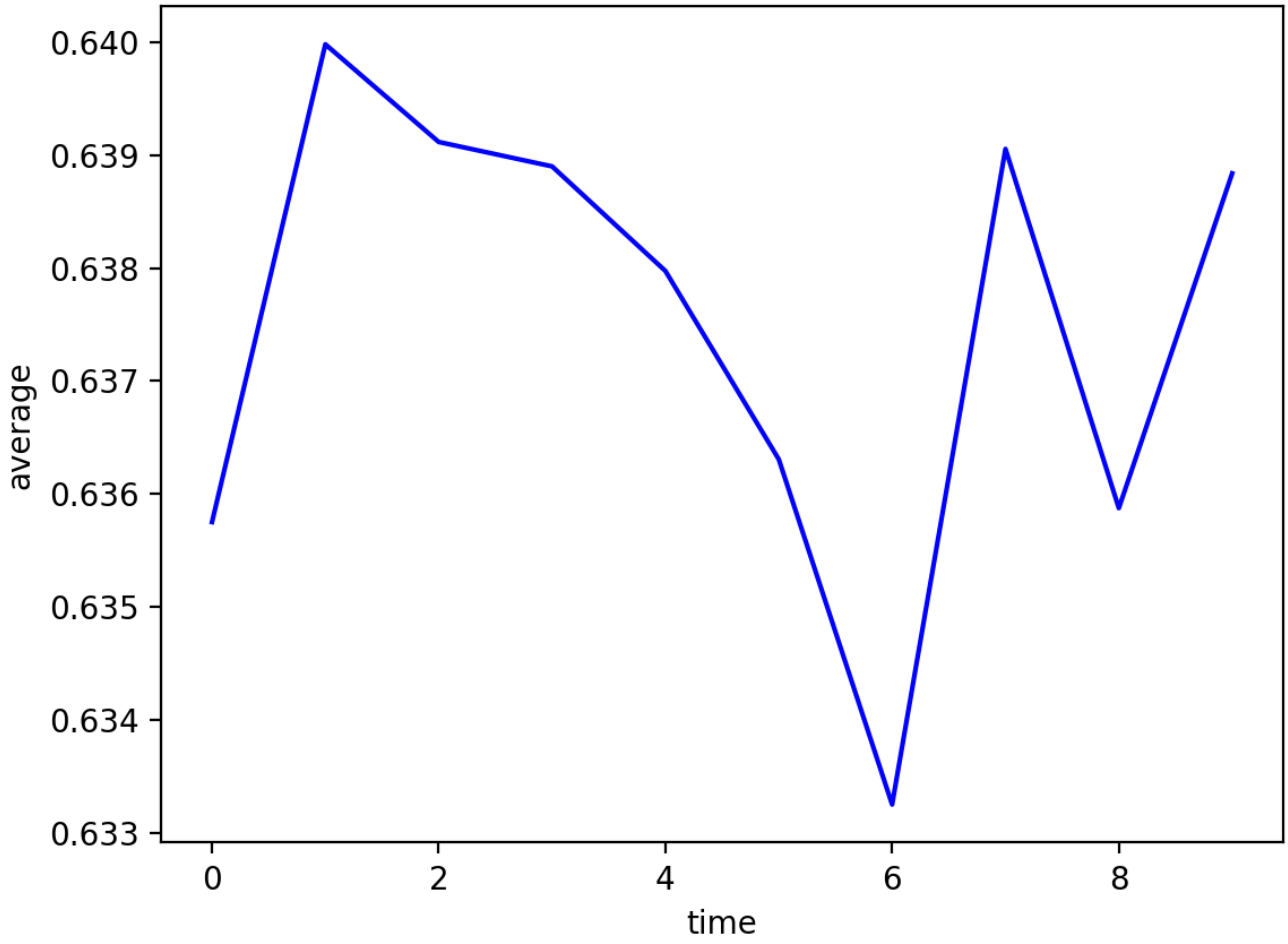
序号	回归	分类
1	0.996480587615	0.872824110158
2	0.996480587615	0.873603533385
3	0.996480587615	0.872824110158
4	0.996480587615	0.871135359834
5	0.996480587615	0.873083917901
6	0.996480587615	0.870485840478
7	0.996480587615	0.870485840478
8	0.996480587615	0.872174590803

9	0.996480587615	0.871914783061
10	0.996480587615	0.868537282411

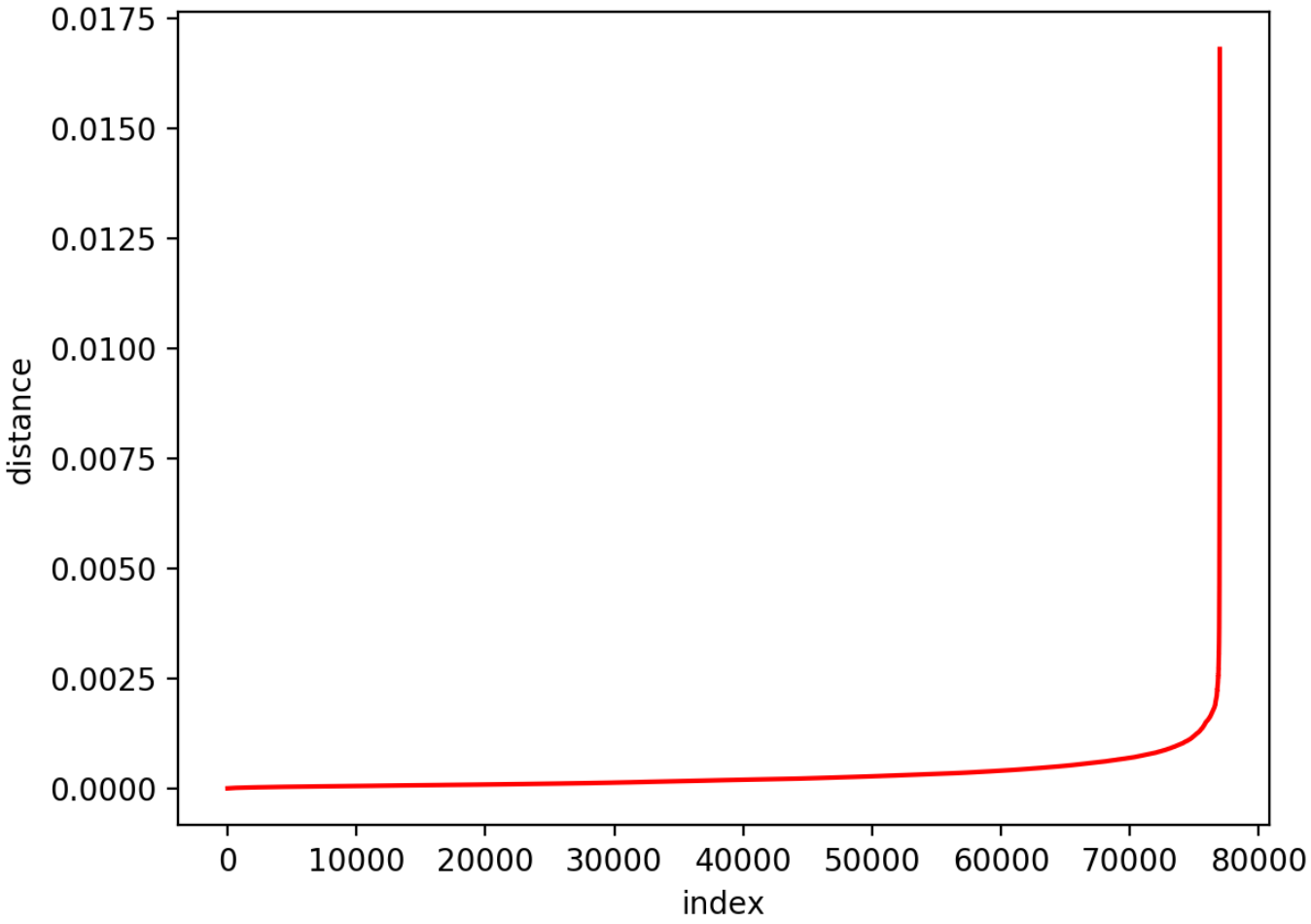
4G回归平均误差图



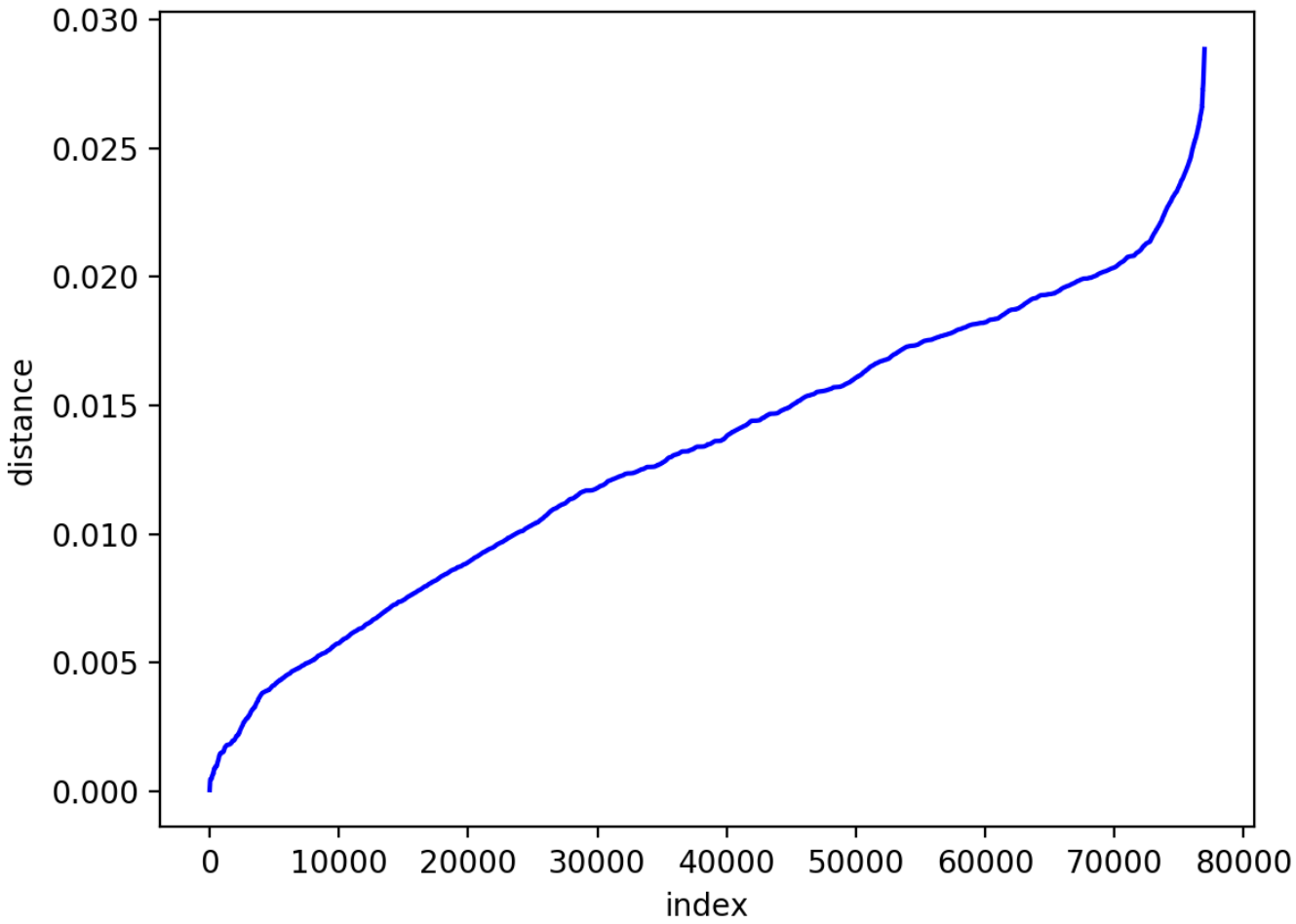
4G分类平均误差图



4G回归误差分布图


0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
distance
0
10000
20000
30000
40000
50000
60000
70000
80000
index

4G分类误差分布图



对于2G和4G准确率的讨论

从预测的准确率来看，2G与4G数据的回归预测数值相近，但是在分类上4G数据更加具有优势，而且4G数据准确率的方差要小于2G数据。所以总体来说，用4G数据进行定位优于2G数据。

第二问

问题概述

利用MR对应的GPS位置进行频繁集挖掘，手机标识符对应于频繁集挖掘中的交易ID，GPS位置所在栅格作为商品ID，将手机的数据看做是一条交易记录进行频繁集的挖掘。

算法详解

Apiori算法

Apriori 算法是一种最有影响力的挖掘布尔关联规则的频繁项集的 算法，它是由Rakesh Agrawal

和RamakrishnanSkrikant 提出的。它使用一种称作逐层搜索的迭代方法，k- 项集用于探索（k+1）- 项集。首先，找出频繁 1- 项集的集合。该集合记作L1。L1 用于找频繁2- 项集的集合L2，而L2 用于找L2，如此下去，直到不能找到 k- 项集。每找一个 Lk 需要一次数据库扫描。为提高频繁项集逐层产生的效率，一种称作Apriori 性质的重 要性质 用于压缩搜索空间。其运行定理在于一是频繁项集的所有非空子集都必须也是频繁的，二是非频繁项集的所有父集都是非频繁的。
Apriori算法过程分为两个步骤：

- 通过迭代，检索出事务数据库中的所有频繁项集，即支持度不低于用户设定的阈值的项集；
- 利用频繁项集构造出满足用户最小信任度的规则。具体做法就是：
  - 首先找出频繁1-项集，记为L1
  - 然后利用L1来产生候选项集C2，对C2中的项进行判定挖掘出L2，即频繁2-项集
  - 不断如此循环下去直到无法发现更多的频繁k-项集为止。每挖掘一层Lk就需要扫描整个数据库一遍。算法利用了一个性质：任一频繁项集的所有非空子集也必须是频繁的。意思就是说，生成一个k-itemset的候选项时，如果这个候选项有子集不在(k-1)-itemset(已经确定是frequent的)中时，那么这个候选项就不用拿去和支持度判断了，直接删除

FpGrowth算法

FPGrowth算法使用了一种类似于前缀树的数据结构。它通过逐个读入事务，并把事务映射到FP树中的一条路径来构造，由于不同的事务可能会有若干个相同的项，因此它们的路径可能部分重叠。路径相互重叠越多，使用FP树结构获得的压缩效果越好，如果FP树足够小，能够存放在内存中，就可以直接从这个内存中的结构提取频繁项集，而不必重复地扫描存放在硬盘上的数据，加快了处理的速度。具体步骤如下：

- 扫描数据项，进行排序
- 针对数据项生成FP树
- 从数量最小的数据项开始，递归生成FP树直到出现单根树位置，其排列组合即为频繁项集

GSP算法

GSP算法是AprioriAll算法的扩展算法，而AprioriAll算法为Apriori类算法，故GSP算法也是一个Apriori类算法。在GSP算法中，引入了时间约束、滑动时间窗和分类层次技术，增加了扫描的约束条件，有效地减少了需要扫描的候选序列的数量，同时还克服了基本序列模型的局限性，更切合实际，减少多余的无用模式的产生。另外，GSP利用哈希树来存储候选序列，减少了需要扫描的序列数量，同时对数据序列的表示方法进行了转换，这样就可以有效地发现一个候选项是否是数据序列的子序列。具体步骤如下：

- 扫描序列数据库，得到长度为1的序列模式L1，作为初始的种子集
- 根据长度为i 的种子集Li ，通过连接操作和修剪操作生成长度为i+1的候选序列模式Ci+1；然后扫描序列数据库，计算每个候选序列模式的支持度，产生长度为i+1的序列模式Li+1，并将

Li+1作为新的种子集

- 重复第二步，直到没有新的序列模式或新的候选序列模式产生为止

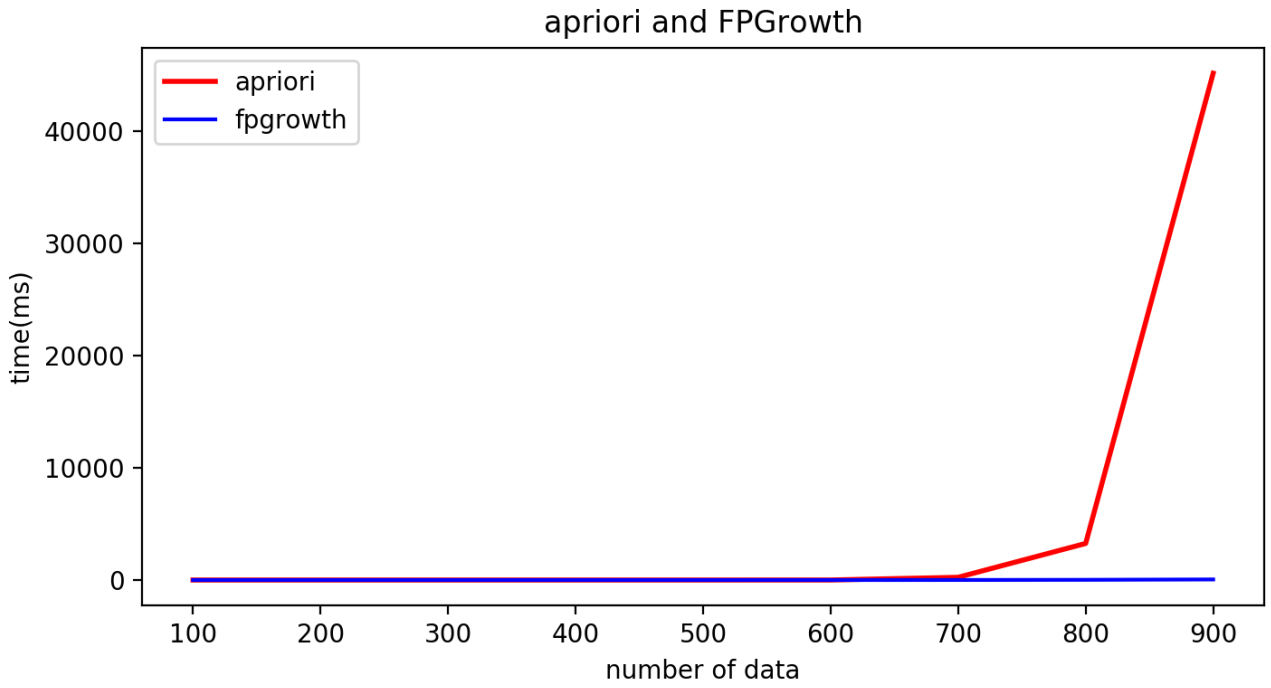
SPADE算法

SPADE算法是为了解决GSP算法计算开销过大而开发的。SPADE算法的原理和GSP基本相近，其中寻找成员1和成员2的方法基本相同，但是在之后的寻找3成员的过程中，使用了一种“作弊”的方法，加速了算法。

结论和分析

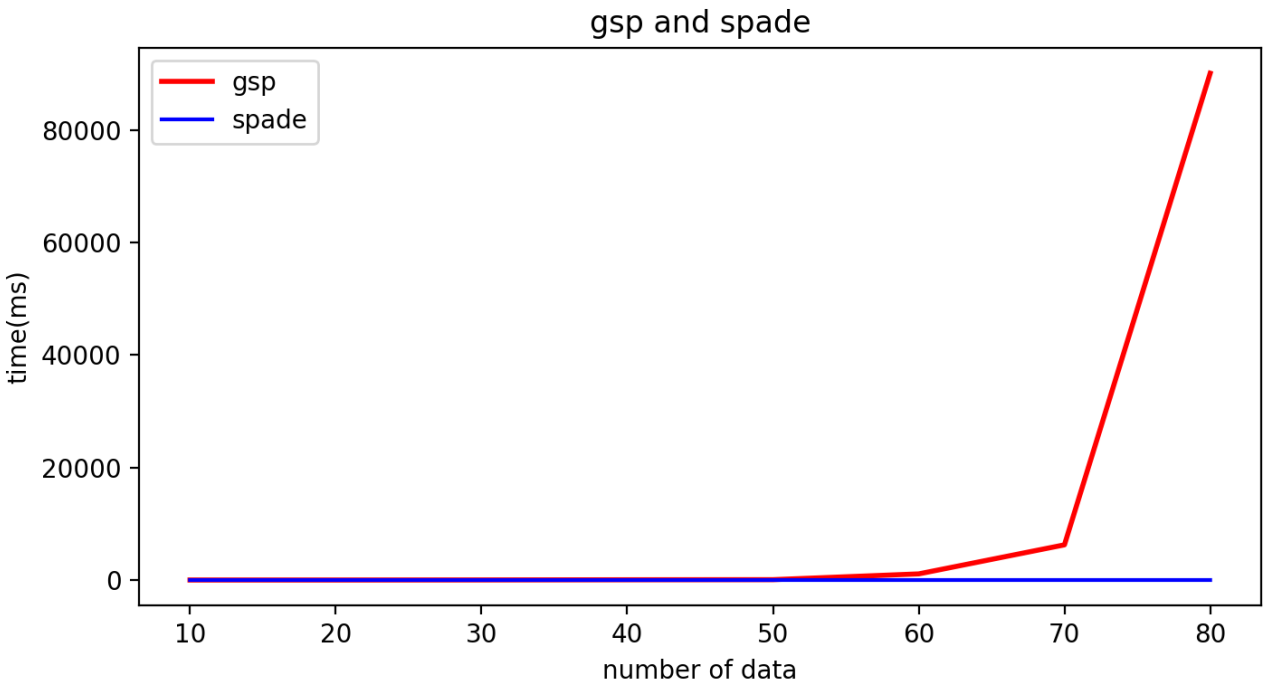
Apiori和FPGrowth的比较

因为所给的数据过大而且只有一个手机号码，根据复杂度的分析根本不可能把它跑出来，所以在这一问中我使用了分割的小的数据集。
我从数据集中选取100-800条数据进行操作用java的包进行运算，最后得到结果如下：



由图上图可知，在数据量较小时，apiori和FPGrowth的差距并不大，但是当数据量逐渐增大时，apiori的时间开销几乎是指数性增长的，但是FPGrowth依旧稳健。
从算法上来看这也是意料之中的事情，apiori计算时需要不断地去扫描数据库，随着数据量的增大，扫描的开销越来越大，但是FPGrowth通过使用巧妙的数据结构使得这部分开销大大减小，在数据量增大的时候就显示出了优越性。

GSP和SPADE的比较



由上图可知，在数据量较小时，GSP算法和SPADE算法的时间开销是相近的，但是当数据量较大时，GSP算法的时间效率就大大上升了。从算法上来看，是因为GSP算法在运算过程中会产生大量的遍历原始数据的过程，这笔开销在数据量增大的过程中非常可怕，最后导致时间开销指数性上升。而SPADE优化的地方正是在扫描这一块，它使用几种方式直接推出频繁项集从而大大减小了扫描的时间开销，从而加快了速度。

心得体会

在本次作业中，最大的体会就是做数据挖掘对机器的要求实在是高，我的机子在跑数据的时候风扇呼呼作响令我十分心疼。但是在写算法的过程中，的确让我学到了不少东西，特别是我自己手撸了一下FPGrowth算法，非常地开心，但是总归有一些遗憾。
本次作业的数据不是很好，使得第二问数据项只有一条，挖掘起来感觉没什么意义，关键是还跑不出来，不是时间爆炸就是内存爆炸，只能拆成小的数据进行计算。
希望下一次助教可以给出更好的数据~
谢谢~