数据挖掘第三次作业

Documentation of the Data Mining Class, homework 3

1552635　胡嘉鑫

2018.6.8

Contents

作业1：手机信号强度提取

1

1.1 a 划分栅格后信号强度分类

1.1.1 题目说明

本道题利用GPS定位系统，在同济大学嘉定校区主要道路上随机取得GPS数据，包括自身的GPS定位、周边主要的七个基站的信号强度信息，其原始数据的特征有：

- Longitude,Latitude: 测量的经纬度
- AsuLevel,RSSI：基站的信号强度，若不存在则赋予空值
- MRTime：测量的时间
- IMSI：手机卡标识
- RNCID, CellID：标识基站的信息
- gongcan_Latitude, gongcan_Longitude：和基站标识信息一一对应的基站的经纬度信息

其中，利用数据中基站的信号强度作为已知的特征值，手机测量时的位置（为了提升模型的预测精确度将经纬度转化为范围稍大一些的栅格ID）。一共6096条数据，分别对7个不同的分类器进行数量为10的交叉验证以后得到平均的误差数组，构建CDF图像，以评价这些分类器的好坏。

1.1.2 处理步骤

- 处理合并数据
  原始数据有data_2g.csv，含有坐标数据和基站信号强度数据的特征；以及基站的公参数据：2g_gongcan.csv,记录了基站的经纬度信息和基站的标记信息。将两部分数据进行融合，用基站的经纬度替换基站的坐标信息，并将GPS的经纬度坐标信息转换为栅格ID。具体步骤如下：

- 首先统计出经纬度的最大最小值，算出最小经纬度和最大经纬度围成的范围的宽度和高度。
- 根据宽度和高度将范围划分为20 x 20的栅格。
- 从经纬度最小的栅格开始，标记栅格ID为1，按照经度从小到大，纬度从小到大的方式进行ID的递增。

经纬度 — 距离的转换计算公式：

$distance = acos(sin(lat1) \times sin(lat2) + cos(lat1) \times cos(lat2) * cos(lon1 - lon2)) \times 6.371 \times 10^7$

$distance$: 两点间的距离

$lat1$fi$lon1$: 第一个点的经纬度坐标值

$lat2, lon2$: 第二个点的经纬度坐标值

- 将得到的数据分为特征数据集和labels
  特征数据集选取了7个基站的信号强度和经纬度信息，label选取了每个特征数据集对应的栅格ID。

- 10 折交叉验证
  为了避免数据本身的偶然性，需要针对数据进行10折交叉分组，经分类器训练、预测后得到10次训练后的平均误差值，这样可以有效减少数据本身的分布偶然性对分类器性能评价造成的影响。*sklearn.model_selection.KFold*是sklearn给出的K折交叉验证的函数包。通过n_split参数可以设置分组的个数，shaffle=true保证每次进行交叉验证分组。

- 使用7个分类器进行训练和预测
  7个分类器分别为：
  - Decision Tree Classifier
  - Gaussian Natural Bayes
  - K Neighbor Classifier
  - AdaBoost Classifier
  - Random Forest Classifier
  - Bagging Classifier
  - Gradient Boost Classifier

前三个分类器是单一的分类器，决策树通过不断构建子节点进行特征判断，朴素贝叶斯利用条件概率进行模型计算，K临近算法根据算出相同label的数据的相对距离进行分组。后四个分类器是组合的分类器，即将多个单一的分类器组合而成的分类器。四个分类器的默认简单分类器为决策树分类器。由于本题数据量不大，对于Gradient Boost分类器而言耗费时间过长，因此主要采用了前6个分类器进行训练和比较。

- 预测栅格ID转换为经纬度，计算误差
  根据预测出来的每个栅格ID，计算这个栅格的中心点距离经纬度最小点的距离，再将其与label中这个点与经纬度最小的点的实际距离进行欧式距离的计算，算成这个点的预测的误差距离。一次预测出来的约400个点进行误差从小到大的排序，最后对于每个分类器可以得到10组这样的经过排序的误差数组。

- 构建图像进行评价。
  对于10组误差值，取每位上的平均值构建分类器的CDF图；并且根据预测出的结果算出分类器的precision、recall和f1 score值，构建每个分类器的柱状图比较性能。

1.1.3 运行结果截图

下面是代码运行的截图：

```
################################################################################
Decision Tree:  precision score: 0.257  recall score:0.266  f1 score:0.247
Gaussian:       precision score: 0.075  recall score:0.087  f1 score:0.073
k Neighbor:     precision score: 0.230  recall score:0.236  f1 score:0.220
AdaBoost:       precision score: 0.001  recall score:0.008  f1 score:0.002
Bagging:        precision score: 0.296  recall score:0.297  f1 score:0.283
Random Forest:  precision score: 0.317  recall score:0.321  f1 score:0.304
```

Figure 1.1 – 运行第一次截图

```
################################################################################
Decision Tree:  precision score: 0.258  recall score:0.281  f1 score:0.255
Gaussian:       precision score: 0.076  recall score:0.090  f1 score:0.076
k Neighbor:     precision score: 0.200  recall score:0.216  f1 score:0.196
AdaBoost:       precision score: 0.003  recall score:0.006  f1 score:0.002
Bagging:        precision score: 0.304  recall score:0.314  f1 score:0.294
Random Forest:  precision score: 0.287  recall score:0.298  f1 score:0.275
```

Figure 1.2 – 运行第二次截图

```
################################################################################
Decision Tree:  precision score: 0.285  recall score:0.294  f1 score:0.277
Gaussian:       precision score: 0.072  recall score:0.095  f1 score:0.076
k Neighbor:     precision score: 0.257  recall score:0.258  f1 score:0.246
AdaBoost:       precision score: 0.001  recall score:0.007  f1 score:0.001
Bagging:        precision score: 0.320  recall score:0.317  f1 score:0.308
Random Forest:  precision score: 0.311  recall score:0.321  f1 score:0.305
```

Figure 1.3 – 运行第三次截图

```
################################################################################
Decision Tree:  precision score: 0.254  recall score:0.274  f1 score:0.251
Gaussian:       precision score: 0.069  recall score:0.088  f1 score:0.069
k Neighbor:     precision score: 0.217  recall score:0.225  f1 score:0.208
AdaBoost:       precision score: 0.003  recall score:0.006  f1 score:0.003
Bagging:        precision score: 0.303  recall score:0.322  f1 score:0.298
Random Forest:  precision score: 0.284  recall score:0.309  f1 score:0.284
```

Figure 1.4 – 运行第四次截图

```
################################################################################
Decision Tree:  precision score: 0.253  recall score:0.257  f1 score:0.240
Gaussian:       precision score: 0.069  recall score:0.090  f1 score:0.070
k Neighbor:     precision score: 0.200  recall score:0.208  f1 score:0.189
AdaBoost:       precision score: 0.004  recall score:0.011  f1 score:0.005
Bagging:        precision score: 0.262  recall score:0.276  f1 score:0.253
Random Forest:  precision score: 0.264  recall score:0.273  f1 score:0.252
```

Figure 1.5 – 运行第五次截图

```
################################################################################
Decision Tree:  precision score: 0.264  recall score:0.286  f1 score:0.261
Gaussian:       precision score: 0.070  recall score:0.088  f1 score:0.070
k Neighbor:     precision score: 0.199  recall score:0.210  f1 score:0.192
AdaBoost:       precision score: 0.002  recall score:0.007  f1 score:0.002
Bagging:        precision score: 0.298  recall score:0.299  f1 score:0.287
Random Forest:  precision score: 0.303  recall score:0.319  f1 score:0.298
```

Figure 1.6 – 运行第六次截图

```
################################################################################
Decision Tree:  precision score: 0.284  recall score:0.283  f1 score:0.271
Gaussian:       precision score: 0.102  recall score:0.113  f1 score:0.099
k Neighbor:     precision score: 0.210  recall score:0.214  f1 score:0.200
AdaBoost:       precision score: 0.002  recall score:0.011  f1 score:0.003
Bagging:        precision score: 0.289  recall score:0.286  f1 score:0.275
Random Forest:  precision score: 0.298  recall score:0.299  f1 score:0.285
```

Figure 1.7 – 运行第七次截图

```
################################################################################
Decision Tree:  precision score: 0.257  recall score:0.258  f1 score:0.245
Gaussian:       precision score: 0.084  recall score:0.110  f1 score:0.088
k Neighbor:     precision score: 0.219  recall score:0.220  f1 score:0.205
AdaBoost:       precision score: 0.001  recall score:0.008  f1 score:0.002
Bagging:        precision score: 0.277  recall score:0.274  f1 score:0.263
Random Forest:  precision score: 0.294  recall score:0.290  f1 score:0.278
```

Figure 1.8 – 运行第八次截图

```
################################################################################
Decision Tree:  precision score: 0.262  recall score:0.265  f1 score:0.251
Gaussian:       precision score: 0.064  recall score:0.087  f1 score:0.068
k Neighbor:     precision score: 0.198  recall score:0.204  f1 score:0.191
AdaBoost:       precision score: 0.003  recall score:0.008  f1 score:0.002
Bagging:        precision score: 0.273  recall score:0.282  f1 score:0.266
Random Forest:  precision score: 0.277  recall score:0.284  f1 score:0.269
```

Figure 1.9 – 运行第九次截图

```
################################################################################
Decision Tree:  precision score: 0.239  recall score:0.254  f1 score:0.234
Gaussian:       precision score: 0.065  recall score:0.087  f1 score:0.066
k Neighbor:     precision score: 0.190  recall score:0.204  f1 score:0.185
AdaBoost:       precision score: 0.001  recall score:0.007  f1 score:0.001
Bagging:        precision score: 0.265  recall score:0.276  f1 score:0.258
Random Forest:  precision score: 0.252  recall score:0.266  f1 score:0.244
```

Figure 1.10 – 运行第十次截图

1.1.4 性能图表

1. 首先来看分类器的CDF图表： 这张图呈现出不同分类器的误差值从小到大排序

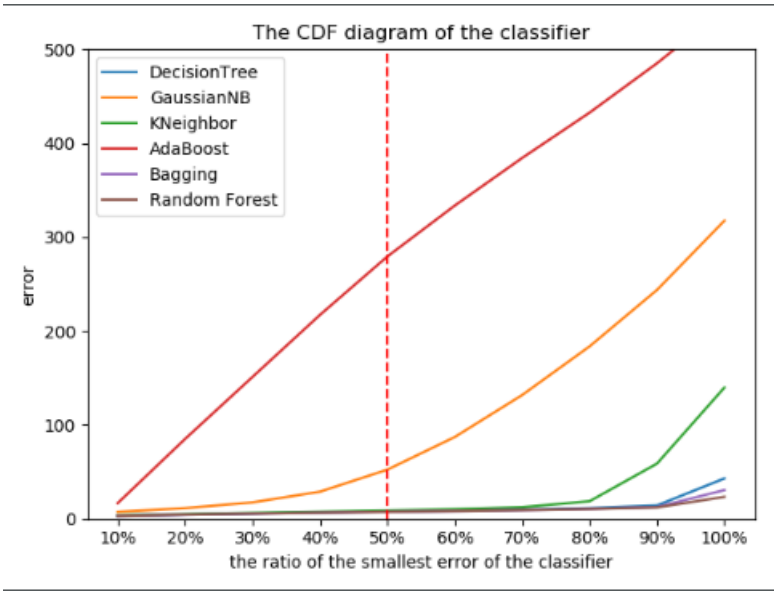

Figure 1.11 – CDF Diagram

后的分布情况。可以看出，除了Gaussian分类器和AdaBoost分类器之外，其余的四个分类器中位误差都能保持在20左右的位置，可以说预测出来的结果还不错。下面就表现最好和最坏的分类器进行探讨：

- Gaussian Naive Bayes:
  假设数据按照正太分布进行分布（默认数据在一个区间内密集分布），按照如下模型进行训练：

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

然而，根据划分栅格后画出的栅格分布图来看，数据点并没有集中聚集的地方，分布比较分散，不适合正太分布模型去拟合，因此用Gaussian Naive Bayes分类的效果并不是很理想。

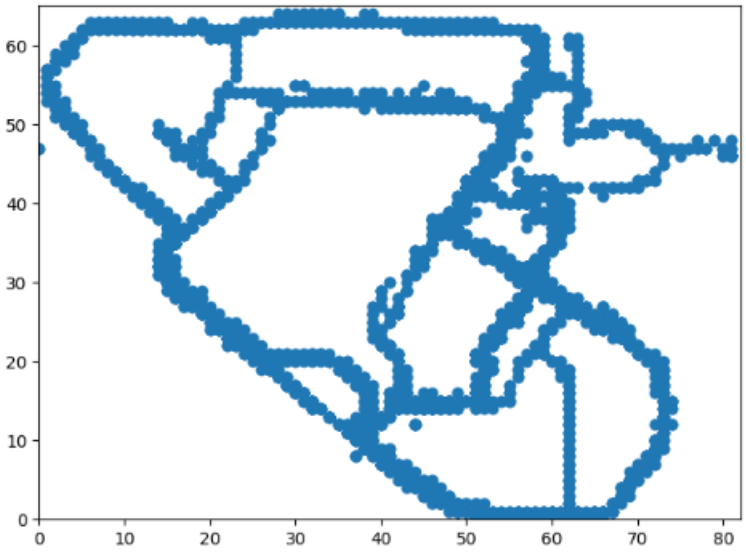

Figure 1.12 – 栅格ID分布图

- AdaBoost Classifier:
  Adaboost每一轮迭代的时候都会训练一个新的弱分类器，直至达到某个预定的足够小的错误率，迭代公式如下：

$$f_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

  因此缺点是对异常点极其敏感，每次异常点都会影响后续的分类器的分类效果。

- Random Forest:
  随机森林分类器选取决策树作为弱分类器，用多个并行的决策树去训练模拟数据，每个决策树的特征数量和树的深度都死随机的，最后选取所有决策树结果的平均值作为训练后的模型。该模型具有一定的随机性，但由于是多个若分类器的组合，其效果要比单一的决策树要好。

2. 接下来是 precision，recall和f1的柱状图比对： 由图可以看出，其柱状图的数

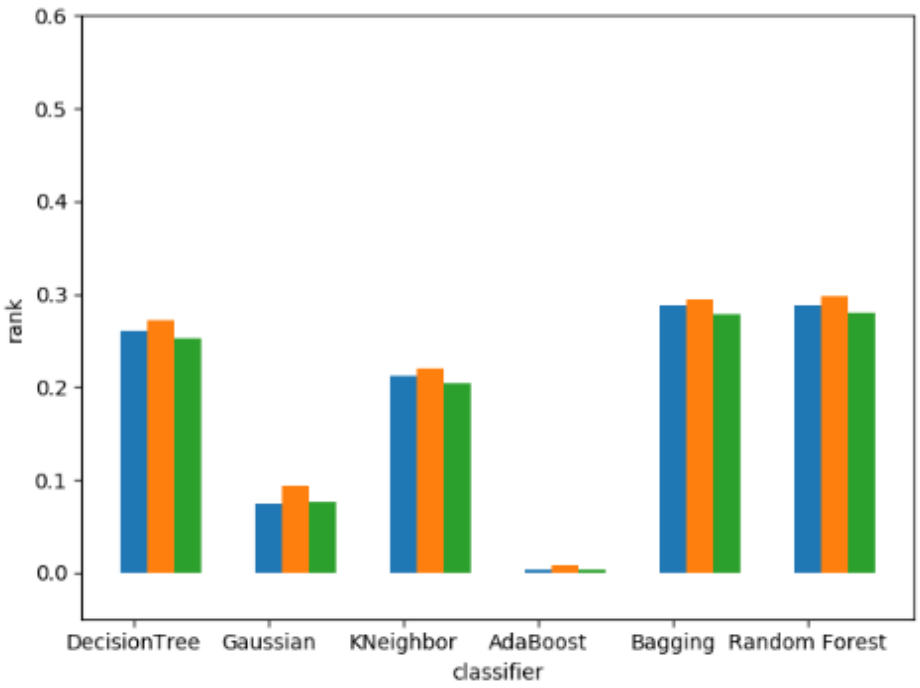

Figure 1.13 – 评测图

据基本对应了CDF图中各个模型的好坏。precision和recall越大，f1的值也相应越大，说明余预测出来的准确性和覆盖率也越高。由图可见，Random Forest值最高，AdaBoost值最低，也符合了CDF图的特点。

1.2 b 更改算法优化定位算法

1.2.1 题目说明

由于使用GPS测量的时候会出现GPS数据定位不准确的现象，现优化定位算法，在训练数据的过程中对不准确数据进行处理，使得效果更好。获取原始数据的经纬度图表如下： 由图表可以看出，数据点总体来说沿着同济大学嘉定校区的主干道均

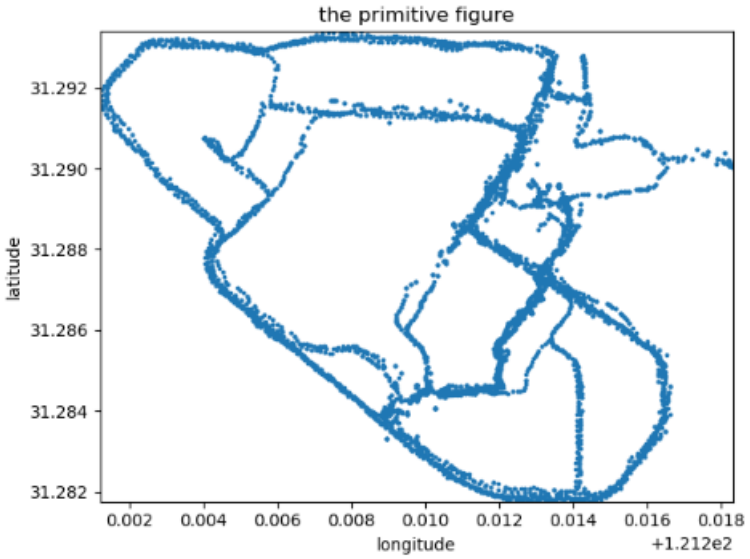

Figure 1.14 – GPS分布图表

匀分布，但不乏一些特殊的点属于偏差的点。我的做法是：根据速度提取这些点的坐标，更改其GPS位置完成优化。

1.2.2 处理步骤

- 对原始数据处理：
  根据IMSI（手机卡）进行分组，然后按照时间从早到晚的顺序依次查看数据坐标点。计算出这个点距离前一个点的速度（根据经纬度差值除以时间差）和距离后一个点的速度。如果这个速度大于5m/s的话，则认为这个点是偏离点，将这个点重新定位为上一个点加上上一个点速度乘以时间的那部分距离：

$$longitude = last\_longitude + speed_longitude * time$$
$$latitude = last\_latitude + speed_latitude * time$$

  之后得到更新以后的GPS点状分布图：

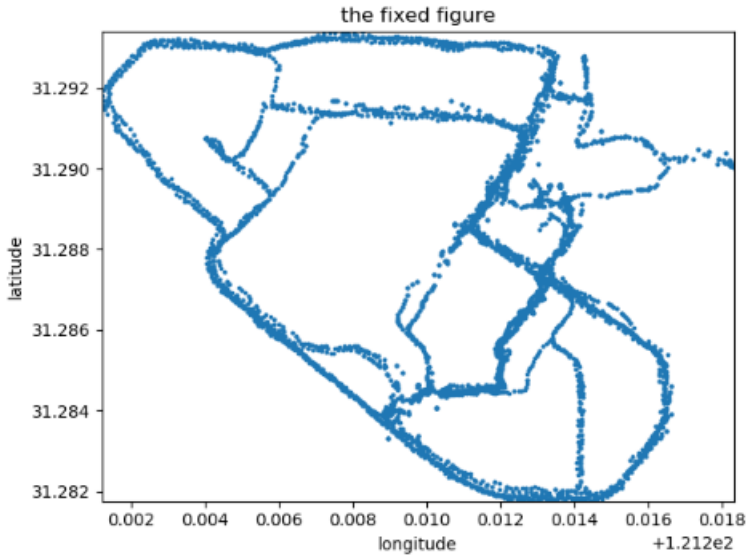

Figure 1.15 – fixed figure

- 使用分类器对原始数据和更改后的数据进行性能比对，得到平均误差数组。
- 十折交叉验证后，对数据进行平均处理，分别算出前10% 20%...的平均误差值。

1.2.3 运行截图

首先是更改坐标的日志截图

```
121.2153138
121.2152442
121.2143175
121.2143178
121.2143
121.2143175
121.2143222
121.2143275
121.2143832
121.2143481
```

Figure 1.16 – log 1

```
121.2013037
121.2014053
121.2012583
121.2013037
121.2012178
121.2012583
121.2012049
121.2012178
121.2012302
```

Figure 1.17 – log 2

```
121.20133909999998
121.20139679999998
121.2013555
121.201624
121.2015622
121.2017446
121.201624
121.2035763
121.2034659
```

Figure 1.18 – log 3

然后是误差的日志截图：

```
[0.0, 0.0, 0.0, 0.0, 13.795379537953796, 45.0, 67.24705882352941, 89.22776261153919, 127.12228483674717, 497.5668606310278]
[0.0, 0.0, 0.0, 0.0, 15.181518151815181, 46.15384615384615, 68.23529411764706, 90.51810578270492, 132.25805216272295, 505.13069787590223]
```

Figure 1.19 – 误差截图

可以看出，对于优化过后的算法，与原有的算法相比平均误差有所下降，结合GPS坐标位置图表来看，这是因为将离群的点重新进行整合，缩小了误差。

1.2.4 优化算法的思考

- 算法还可以继续优化。我的算法考虑到如果一个点是离群点，则将其按照原有的速度进行排序。但由于这个点更改位置后的基站信号强度的GPS位置也可能发生变化，这种算法其实一定程度上产生了一些误差值，所以最终优化效果一般。

- 最好不要轻易改动原始数据。由于本题目要求在偏差值下进行算法优化，我对坐标进行了适度的调整。但是由于样本数据本身就会存在偏差，训练过程中也无法避免误差的存在，因此与更改GPS坐标相比，直接删去利群点也是不错的主意。

1.3 c 根据基站分组，重新构建特征进行训练

1.3.1 题目说明

与a问不同的是，这一问首先将原有数据根据第一个基站标识进行分组，算出每个坐标点相对于主基站的相对位置。将相对位置作为label，分别对数据进行训练得到基站信号强度和坐标相对于主基站位置的信息，最后将相对位置转化为绝对的经纬度坐标值。

1.3.2 处理步骤

- 根据主基站的RNCID和CellID（之前步骤已经转化成主基站的经纬度坐标值）进行分组

- 遍历分组后的每条数据，将GPS信号的绝对位置转化为相对位置。相对位置的计算方式如下：

$$latitude = (Latitude - base_latitude)$$
$$longitude = (Longitude - base_longitude)$$

其中，latitude是相对纬度，longitude是相对经度

- 对分组后的数据分别进行Random Forest Regression 回归算法。这是因为label取经纬度两个值作为标签，且最终预测的经纬度并不想局限于label中的已有数据，因此选择回归算法。遍历所有的基站算出一组误差，进行排序。

- 将预测出的十组结果排序、算出平均误差并按照10%,20%...选出10个误差值的点。

- 按照a问的做法算出RandomForestRegression预测结果，画出图表分析好坏。

1.3.3 运行截图

按照基站分组的运行结果：

```
############################################################
train with base: latitude: 121.191709    lonitude: 31.287846
the data number is: 44
error: 31.442235
############################################################
train with base: latitude: 121.196365    lonitude: 31.295884
the data number is: 7
error: 36.501134
############################################################
train with base: latitude: 121.198241    lonitude: 31.279433
the data number is: 3
error: 14.239869
############################################################
train with base: latitude: 121.203915    lonitude: 31.280386
the data number is: 40
error: 35.607762
############################################################
train with base: latitude: 121.205655    lonitude: 31.277705
the data number is: 2
error: 0.000000
```

Figure 1.20 – base station 1 - 5

```
############################################################
train with base: latitude: 121.206155    lonitude: 31.294312
the data number is: 480
error: 27.335520
############################################################
train with base: latitude: 121.206769    lonitude: 31.286810
the data number is: 63
error: 27.889367
############################################################
train with base: latitude: 121.208105    lonitude: 31.289402
the data number is: 787
error: 42.374653
############################################################
train with base: latitude: 121.208284    lonitude: 31.289647
the data number is: 773
error: 27.341019
############################################################
train with base: latitude: 121.209638    lonitude: 31.285875
the data number is: 9
error: 18.469783
```

Figure 1.21 – base station 6 - 10

```
############################################################
train with base: latitude: 121.209767    lonitude: 31.284987
the data number is: 178
error: 32.700497
############################################################
train with base: latitude: 121.210909    lonitude: 31.278470
the data number is: 38
error: 20.416937
############################################################
train with base: latitude: 121.211928    lonitude: 31.288649
the data number is: 806
error: 33.626551
############################################################
train with base: latitude: 121.211975    lonitude: 31.276588
the data number is: 329
error: 37.465770
############################################################
train with base: latitude: 121.212407    lonitude: 31.282261
the data number is: 162
error: 25.322291
```

Figure 1.22 – base station 11 - 15

```
############################################################
train with base: latitude: 121.213001    lonitude: 31.289531
the data number is: 81
error: 20.898497
############################################################
train with base: latitude: 121.214295    lonitude: 31.280418
the data number is: 267
error: 20.332869
############################################################
train with base: latitude: 121.216448    lonitude: 31.289300
the data number is: 52
error: 13.594074
############################################################
train with base: latitude: 121.216674    lonitude: 31.287692
the data number is: 18
error: 34.457723
############################################################
train with base: latitude: 121.217402    lonitude: 31.281802
the data number is: 301
error: 49.633749
```

Figure 1.23 – base station 16 - 20

```
train with base: latitude: 121.217871    lonitude: 31.280800
the data number is: 99
error: 41.459776
################################################################
train with base: latitude: 121.218285    lonitude: 31.288788
the data number is: 323
error: 20.950109
################################################################
train with base: latitude: 121.218479    lonitude: 31.276852
the data number is: 133
error: 27.762936
################################################################
train with base: latitude: 121.218509    lonitude: 31.286137
the data number is: 70
error: 0.000000
################################################################
train with base: latitude: 121.219323    lonitude: 31.289351
the data number is: 72
error: 58.813257
```

Figure 1.24 – base station 21 - 25

```
################################################################
train with base: latitude: 121.220505    lonitude: 31.290893
the data number is: 62
error: 36.309219
################################################################
train with base: latitude: 121.220639    lonitude: 31.281873
the data number is: 427
error: 23.969562
################################################################
train with base: latitude: 121.220772    lonitude: 31.275301
the data number is: 128
error: 20.051550
################################################################
train with base: latitude: 121.221072    lonitude: 31.257603
the data number is: 183
error: 22.456410
################################################################
train with base: latitude: 121.221816    lonitude: 31.293837
the data number is: 8
error: 7.165443
```

Figure 1.25 – base station 26 - 30

```
################################################################
train with base: latitude: 121.222920    lonitude: 31.287199
the data number is: 85
error: 52.355058
################################################################
train with base: latitude: 121.224621    lonitude: 31.278733
the data number is: 44
error: 36.177926
```

Figure 1.26 – base station 31 - 32

下面是随机森林分类器的运行截图：

```
################################################################################
Decision Tree:  precision score: 0.257  recall score:0.258  f1 score:0.245
################################################################################
Decision Tree:  precision score: 0.270  recall score:0.295  f1 score:0.268
################################################################################
Decision Tree:  precision score: 0.296  recall score:0.298  f1 score:0.285
################################################################################
Decision Tree:  precision score: 0.265  recall score:0.264  f1 score:0.255
################################################################################
Decision Tree:  precision score: 0.296  recall score:0.304  f1 score:0.287
################################################################################
Decision Tree:  precision score: 0.307  recall score:0.311  f1 score:0.295
################################################################################
Decision Tree:  precision score: 0.316  recall score:0.316  f1 score:0.302
################################################################################
Decision Tree:  precision score: 0.282  recall score:0.302  f1 score:0.280
################################################################################
Decision Tree:  precision score: 0.300  recall score:0.291  f1 score:0.284
################################################################################
Decision Tree:  precision score: 0.279  recall score:0.278  f1 score:0.267
```

Figure 1.27 – base station 11 - 15

1.3.4 性能讨论分析

首先观察两种方法对应的误差折线图： 可以得出如下结论：

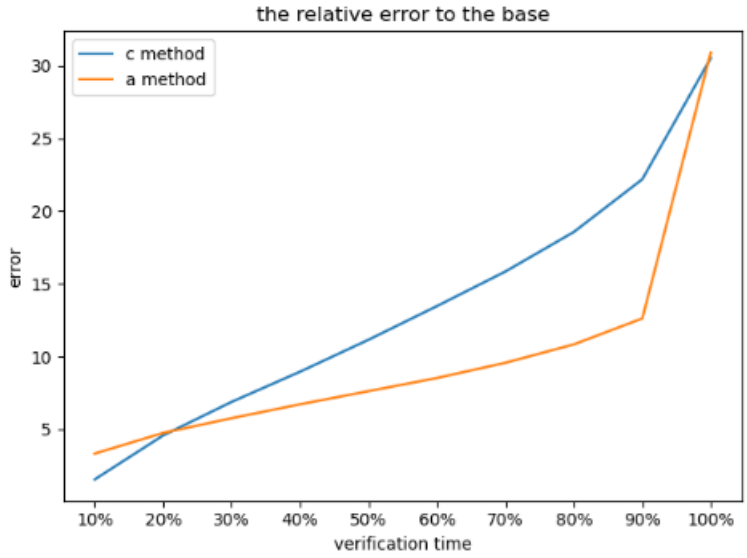

Figure 1.28 – Caption

- 由折线图可以看出来，改进的方法在极小和所有的平均误差中，和原有的方法持平。这是因为两个分类器在总体上都采用随机森林进行分类和回归，其随机取样特征和树的深度可能会影响中间的某些值，但最终整体的效果相当。

- 从截取的运行截图中可以看出来，c在对主基站进行分类时，每个基站的样本数量差异巨大。有的基站样本有400+个，而有的只有3，4个，样本数量越少，训练出来的模型越不可靠，因此和整体做优化的a方法来比还是比较简略粗糙的。

1.4 d 增补基站数据算法

1.4.1 题目说明

根据c)中计算每个分组定位模型的中位误差进行排序，取得中位误差最小的前top-k分组（记为topk+）和中位误差最大的后top-k分组（记为topk-），其中k=K*0.2，利用topk+分组中的MR数据融入到topk-分组中，重新处理c）的定位步骤。
由上一问得出的结论：某些基站数据过少不适合训练，因此要补全数据

1.4.2 处理步骤

- 获取c问中平均误差大的6组基站ID和c问中平均误差小的6组基站ID
- 将误差小的基站ID和误差大的基站ID进行融合
- 重复c问的训练过程

1.4.3 代码截图

```
################################################################
train with base: latitude: 121.191709    lonitude: 31.287846
the data number is: 44
error: 31.126293
################################################################
train with base: latitude: 121.196365    lonitude: 31.295884
the data number is: 7
error: 37.151578
################################################################
train with base: latitude: 121.198241    lonitude: 31.279433
the data number is: 3
error: 14.239869
################################################################
train with base: latitude: 121.203915    lonitude: 31.280386
the data number is: 40
error: 39.722312
################################################################
train with base: latitude: 121.205655    lonitude: 31.277705
the data number is: 2
error: 0.000000
```

Figure 1.29 – log 1 - 5

```
train with base: latitude: 121.206155    lonitude: 31.294312
the data number is: 480
error: 50.555894
################################################################
train with base: latitude: 121.206769    lonitude: 31.286810
the data number is: 63
error: 20.879401
################################################################
train with base: latitude: 121.208105    lonitude: 31.289402
the data number is: 787
error: 41.470067
################################################################
train with base: latitude: 121.208284    lonitude: 31.289647
the data number is: 773
error: 31.465637
################################################################
train with base: latitude: 121.209638    lonitude: 31.285875
the data number is: 9
error: 18.431657
```

Figure 1.30 – log 6 - 10

```
################################################################
train with base: latitude: 121.209767    lonitude: 31.284987
the data number is: 178
error: 35.795758
################################################################
train with base: latitude: 121.210909    lonitude: 31.278470
the data number is: 38
error: 20.138611
################################################################
train with base: latitude: 121.211928    lonitude: 31.288649
the data number is: 806
error: 32.232783
################################################################
train with base: latitude: 121.211975    lonitude: 31.276588
the data number is: 329
error: 39.172288
################################################################
train with base: latitude: 121.212407    lonitude: 31.282261
the data number is: 162
```

Figure 1.31 – log 11 - 15

```
train with base: latitude: 121.213001    lonitude: 31.289531
the data number is: 81
error: 20.160450
################################################################
train with base: latitude: 121.214295    lonitude: 31.280418
the data number is: 267
error: 22.531261
################################################################
train with base: latitude: 121.216448    lonitude: 31.289300
the data number is: 52
error: 14.263798
################################################################
train with base: latitude: 121.216674    lonitude: 31.287692
the data number is: 18
error: 32.850506
################################################################
train with base: latitude: 121.217402    lonitude: 31.281802
the data number is: 301
error: 47.851103
```

Figure 1.32 – log 16 - 20

```
train with base: latitude: 121.217871    lonitude: 31.280800
the data number is: 99
error: 36.189371
################################################################
train with base: latitude: 121.218285    lonitude: 31.288788
the data number is: 323
error: 20.155658
################################################################
train with base: latitude: 121.218479    lonitude: 31.276852
the data number is: 133
error: 26.097140
################################################################
train with base: latitude: 121.218509    lonitude: 31.286137
the data number is: 70
error: 0.000000
################################################################
train with base: latitude: 121.219323    lonitude: 31.289351
the data number is: 72
error: 42.963882
```

Figure 1.33 – log 21 - 25

```
train with base: latitude: 121.220505    lonitude: 31.290893
the data number is: 62
error: 31.770753
################################################################
train with base: latitude: 121.220639    lonitude: 31.281873
the data number is: 427
error: 23.604325
################################################################
train with base: latitude: 121.220772    lonitude: 31.275301
the data number is: 128
error: 22.639962
################################################################
train with base: latitude: 121.221072    lonitude: 31.257603
the data number is: 183
error: 26.041798
################################################################
train with base: latitude: 121.221816    lonitude: 31.293837
the data number is: 8
error: 7.165263
```

Figure 1.34 – log 26 - 30

```
train with base: latitude: 121.222617    lonitude: 31.290824
the data number is: 22
error: 34.831001
################################################################
train with base: latitude: 121.222920    lonitude: 31.287199
the data number is: 85
error: 51.619008
################################################################
train with base: latitude: 121.224621    lonitude: 31.278733
the data number is: 44
error: 35.569365
```

Figure 1.35 – log 31 - 33

1.4.4 性能分析

首先得到两个方法比较的折线图

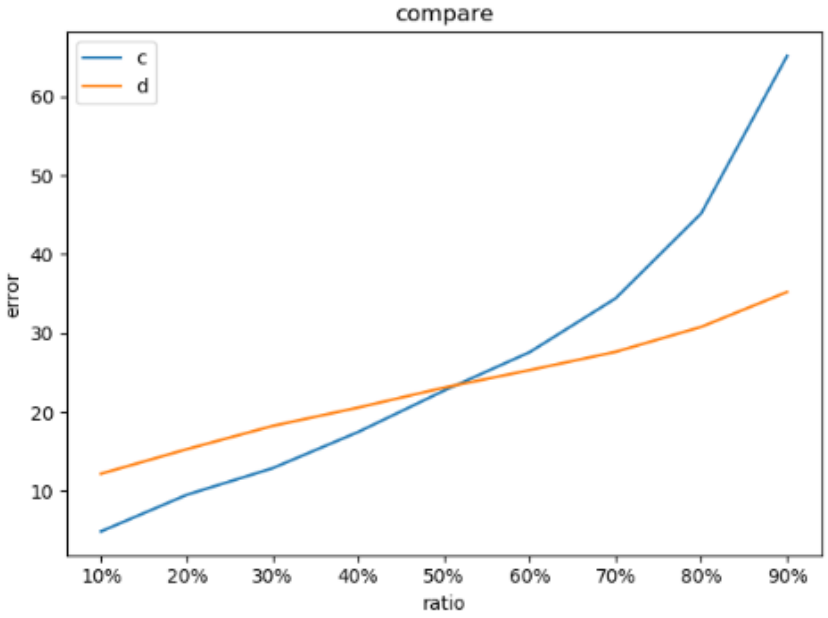

Figure 1.36 – Compare

- 首先可以看到，d问的方法总体而言误差要比c问的方法要小，这是因为误差大的数据集往往是数据集不够，模型训练不够导致的。d问添加了数据集进去，使得训练效果更好。

- ratio比较小的时候d问不如c问好。同样因为c问各个数据集数目迥异，偶然性很大，有些数据集数量虽然少但预测准确率相当高，因此会有少部分数据预测很准确。另外，对于d问来说，直接向预测结果不好的 集合里添加新的数据的话由于围绕的主基站不同，因此可能会影响个别本来预测准确的数据变得不准确，因此比率较小时预测效果不是很好。

1.5 e 构建相似性分组

1.5.1 题目说明

由d问引起的启发：如果我不再取预测最好的几个集合去补充预测不好的集合，而是让基站距离较近的几个集合组成一个大的集合，这样即避免了个别集合数量过少的尴尬，又避免了集合融合不当引起的数据预测偏差。

1.5.2 处理步骤

- 利用KMeans算法进行聚类，得到聚类的label

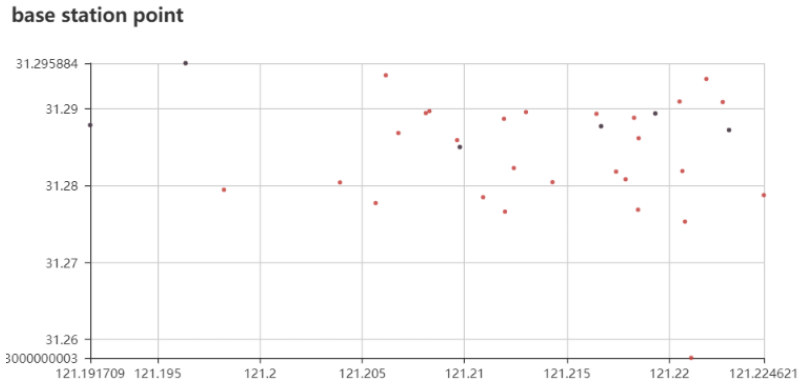

Figure 1.37 – point dragram

- 根据聚类结果融合数据
- 重复c的预测工作

1.5.3 运行截图

```
train with base: latitude: 121.191709    lonitude: 31.287846
the data number is: 44
error: 31.585138
################################################################
train with base: latitude: 121.196365    lonitude: 31.295884
the data number is: 7
error: 36.533383
################################################################
train with base: latitude: 121.198241    lonitude: 31.279433
the data number is: 3
error: 14.239869
################################################################
train with base: latitude: 121.203915    lonitude: 31.280386
the data number is: 40
error: 35.696230
################################################################
train with base: latitude: 121.205655    lonitude: 31.277705
the data number is: 2
error: 0.000000
```

Figure 1.38 – log 1 - 5

```
train with base: latitude: 121.206155    lonitude: 31.294312
the data number is: 480
error: 53.356721
################################################################
train with base: latitude: 121.206769    lonitude: 31.286810
the data number is: 63
error: 31.153438
################################################################
train with base: latitude: 121.208105    lonitude: 31.289402
the data number is: 787
error: 41.572986
################################################################
train with base: latitude: 121.208284    lonitude: 31.289647
the data number is: 773
error: 26.172769
################################################################
train with base: latitude: 121.209638    lonitude: 31.285875
the data number is: 9
error: 18.484426
```

Figure 1.39 – log 6 - 10

```
train with base: latitude: 121.209767    lonitude: 31.284987
the data number is: 178
error: 33.505485
################################################################
train with base: latitude: 121.210909    lonitude: 31.278470
the data number is: 38
error: 20.233926
################################################################
train with base: latitude: 121.211928    lonitude: 31.288649
the data number is: 806
error: 33.436120
################################################################
train with base: latitude: 121.211975    lonitude: 31.276588
the data number is: 329
error: 34.060566
################################################################
train with base: latitude: 121.212407    lonitude: 31.282261
the data number is: 162
error: 11.576084
```

Figure 1.40 – log 11 - 15

```
train with base: latitude: 121.213001    lonitude: 31.289531
the data number is: 81
error: 20.299860
################################################################
train with base: latitude: 121.214295    lonitude: 31.280418
the data number is: 267
error: 20.574549
################################################################
train with base: latitude: 121.216448    lonitude: 31.289300
the data number is: 52
error: 20.317437
################################################################
train with base: latitude: 121.216674    lonitude: 31.287692
the data number is: 18
error: 35.708540
################################################################
train with base: latitude: 121.217402    lonitude: 31.281802
the data number is: 301
error: 46.714451
```

Figure 1.41 – log 16 - 20

```
train with base: latitude: 121.217871    lonitude: 31.280800
the data number is: 99
error: 38.081421
################################################################
train with base: latitude: 121.218285    lonitude: 31.288788
the data number is: 323
error: 20.982396
################################################################
train with base: latitude: 121.218479    lonitude: 31.276852
the data number is: 133
error: 26.976323
################################################################
train with base: latitude: 121.218509    lonitude: 31.286137
the data number is: 70
error: 0.000000
################################################################
train with base: latitude: 121.219323    lonitude: 31.289351
the data number is: 72
error: 40.151164
```

Figure 1.42 – log 21 - 25

```
train with base: latitude: 121.220505    lonitude: 31.290893
the data number is: 62
error: 27.806244
################################################################
train with base: latitude: 121.220639    lonitude: 31.281873
the data number is: 427
error: 21.675593
################################################################
train with base: latitude: 121.220772    lonitude: 31.275301
the data number is: 128
error: 24.146125
################################################################
train with base: latitude: 121.221072    lonitude: 31.257603
the data number is: 183
error: 28.503088
################################################################
train with base: latitude: 121.221816    lonitude: 31.293837
the data number is: 8
error: 7.166940
```

Figure 1.43 – log 26 - 30

```
train with base: latitude: 121.222617    lonitude: 31.290824
the data number is: 22
error: 38.737457
############################################################
train with base: latitude: 121.222920    lonitude: 31.287199
the data number is: 85
error: 54.464016
############################################################
train with base: latitude: 121.224621    lonitude: 31.278733
the data number is: 44
error: 31.908170
```

Figure 1.44 – log 31 - 33

1.5.4 性能评价

首先来看d问和e问的对比折线图： 由图表可以看出，两种方法在误差改进上大致

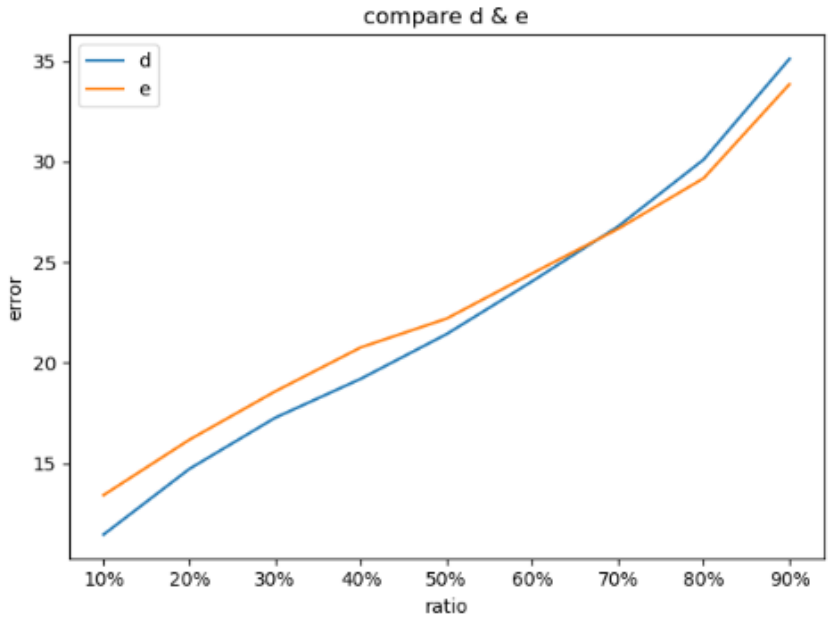

Figure 1.45 – Compare e & d

相同，我的理解是这样：

- 对数据进行聚类后，仍然有部分基站的点是单独游离在外的，这些点的预测误差将使整体误差拉大。
- 由于聚类以后，同一个类别下的基站距离相对较近，对于减小整体的误差有帮助

float

作业2：商品信息特征提取和训练 2

2.1 a 特征提取

2.1.1 题目说明

trade_new.csv文件中包含了每个用户不同时间段下购买商品的信息，如果想要根据用户的购买记录预测将来可能的购买情况的话，需要对特征进行处理，即特征工程。首先抽取出来关键性的特征：

- vipno：用户ID
- pluno：商品ID
- bndno：商品品牌
- dptno：商品种类
- sldtime: 交易完成的时间

根据上述特征，分别生成复合的新的特征，将会提升预测出来的效果。
我的做法是：根据建议的特征提取方式，首先将特征逐步提取成不同的csv文件，最后在生成特征数据的时候将想要使用的特征csv文件进行合并，这样既能使特征工程独立开来，便于管理，也能大大降低合成特征的速度，提升预测的时间。
另外，本题利用2，3，4月份的数据作为特征数据，5月份作为label训练模型，最后选取合适的模型对6，7，8月份的数据进行预测，得到预测结果。生成的特征文件存放在q2.a.data文件目录下,预测文件存放在q2.a.predict文件目录下,文件结构一一对应。

2.1.2 处理步骤

本题的处理步骤就是生成特征的过程。这里关注于2，3，4月份的特征生成过程，6，7，8月份的数据生成过程与之相同。 1. Type1
1.1 count 统计每个用户、商品、品牌、种类及其组合的购买次数和购买金额，分别按照2，3，4月份进行统计并且最后统计三个月的总体值。这里的购买次数是指购买商品的总数量。
文件及特征：

- data_u.csv: 统计每个用户2,3,4月份和总体的用户购买商品的数量、金额总量。
  特征：$u, u\_count\_all, u\_count_2, u\_count\_3, u\_count\_4,$
  $u\_amount\_all, u\_amount\_2, u\_amount\_3, u\_amount\_4$

- data_i.csv: 统计每个商品2,3,4月份和总体的购买的数量、金额总量。
  特征：$i, i\_count\_all, i\_count_2, i\_count\_3, i\_count\_4,$
  $i\_amount\_all, i\_amount\_2, i\_amount\_3, i\_amount\_4$

- data_b.csv: 统计每个品牌2,3,4月份和总体的购买数量、金额总量。
  特征：$b, b\_count\_all, b\_count_2, b\_count\_3, b\_count\_4,$
  $b\_amount\_all, b\_amount\_2, b\_amount\_3, b\_amount\_4$

- data_c.csv: 统计每个商品类别2,3,4月份和总体被购买的数量、金额总量。
  特征：$c, c\_count\_all, c\_count_2, c\_count\_3, c\_count\_4,$
  $c\_amount\_all, c\_amount\_2, c\_amount\_3, c\_amount\_4$

- data_u_i.csv: 统计每个用户购买的特定商品在2,3,4月份和总体被购买的数量、金额总量。
  特征：$u, i, u\_i\_count\_all, u\_i\_count\_2, u\_i\_count\_3, u\_i\_count\_4,$
  $u\_i\_amount\_all, u\_i\_amount\_2, u\_i\_amount\_3, u\_i\_amount\_4$

- data_u_b.csv: 统计每个用户购买的特定品牌在2,3,4月份和总体被购买的数量、金额总量。
  特征：$u, b, u\_b\_count\_all, u\_b\_count\_2, u\_b\_count\_3, u\_b\_count\_4,$
  $u\_b\_amount\_all, u\_b\_amount\_2, u\_b\_amount\_3, u\_b\_amount\_4$

- data_u_c.csv: 统计每个用户购买的各个种类在2,3,4月份和总体被购买的数量、金额总量。
  特征：$u, c, u\_c\_count\_all, u\_c\_count\_2, u\_c\_count\_3, u\_c\_count\_4,$
  $u\_c\_amount\_all, u\_c\_amount\_2, u\_c\_amount\_3, u\_c\_amount\_4$

1.2 diversity 统计每个用户2，3，4月和三个月内购买的商品、品牌类型、品牌种类的数目，以及每个品牌和种类被购买的商品的数目
文件及特征：

- data_u_i: 统计每个用户2，3，4月和三个月内购买的商品的数目。
  特征：$u, u\_i\_count\_unique\_all, u\_i\_count\_unique\_2,$
  $u\_i\_count\_unique\_3, u\_i\_count\_unique\_4$

- data_u_b: 统计每个用户2，3，4月和三个月内购买的品牌的数目。
  特征：$u, u\_b\_count\_unique\_all, u\_b\_count\_unique\_2,$
  $u\_b\_count\_unique\_3, u\_b\_count\_unique\_4$

- data_u_c: 统计每个用户2，3，4月和三个月内购买的商品种类的数目。
  特征：$u, u\_c\_count\_unique\_all, u\_c\_count\_unique\_2,$
  $u\_c\_count\_unique\_3, u\_c\_count\_unique\_4$

- data_b_i: 统计每个品牌2，3，4月和三个月内购买的商品的数目。
  特征：$b, b\_i\_count\_unique\_all, b\_i\_count\_unique\_2,$
  $b\_i\_count\_unique\_3, b\_i\_count\_unique\_4$

- data_c_i: 统计每个种类2，3，4月和三个月内购买的商品的数目。
  特征：$c, c\_i\_count\_unique\_all, c\_i\_count\_unique\_2,$
  $c\_i\_count\_unique\_3, c\_i\_count\_unique\_4$

1.3 penetration 统计购买不同商品，不同商品品牌，不同商品类型的用户的个数。
文件及特征：

- data_b.csv: 每个类别2，3，4月份和三个月份被用户购买的vipno数量
  特征：$b, b\_user\_all, b\_user\_2, b\_user\_3, b\_user\_4$

- data_c,csv: 每个种类2，3，4月份和三个月份被用户购买的vipno数量
  特征：$c, c\_user\_all, c\_user\_2, c\_user\_3, c\_user\_4$

- data_i,csv: 每个商品2，3，4月份和三个月份被用户购买的vipno数量
  特征：$i, i\_user\_all, i\_user\_2, i\_user\_3, i\_user\_4$

2.1 month_agg 根据type1的数据生成对于每个月的平均值，中位值，最大值和标准差
2.1.1 count(diversity 和 penetration处理方式和count相同，就不再进行列举)

- data_b.csv:
  $b, b\_agg\_count\_max, b\_agg\_count\_median, b\_agg\_count\_mean,$
  $b\_agg\_count\_std, b\_agg\_amount\_max, b\_agg\_amount\_median,$
  $b\_agg\_amount\_mean, b\_agg\_amount\_std$

- data_c.csv:
  $c, c\_agg\_count\_max, c\_agg\_count\_median, c\_agg\_count\_mean,$
  $c\_agg\_count\_std, c\_agg\_amount\_max, c\_agg\_amount\_median,$
  $c\_agg\_amount\_mean, c\_agg\_amount\_std$

- data_i.csv:
  $i, i\_agg\_count\_max, i\_agg\_count\_median, i\_agg\_count\_mean,$
  $i\_agg\_count\_std, i\_agg\_amount\_max, i\_agg\_amount\_median,$
  $i\_agg\_amount\_mean, i\_agg\_amount\_std$

- data_u.csv:
  $u, u\_agg\_count\_max, u\_agg\_count\_median, u\_agg\_count\_mean,$
  $u\_agg\_count\_std, u\_agg\_amount\_max, u\_agg\_amount\_median,$
  $u\_agg\_amount\_mean, u\_agg\_amount\_std$

- data_u_i.csv:
  $u, i, u\_i\_agg\_count\_max, u\_i\_agg\_count\_median, u\_i\_agg\_count\_mean,$
  $u\_i\_agg\_count\_std, u\_i\_agg\_amount\_max, u\_i\_agg\_amount\_median,$
  $u\_i\_agg\_amount\_mean, u\_i\_agg\_amount\_std$

- data_u_b.csv:
  $u, b, u\_b\_agg\_count\_max, u\_b\_agg\_count\_median, u\_b\_agg\_count\_mean,$
  $u\_b\_agg\_count\_std, u\_b\_agg\_amount\_max, u\_b\_agg\_amount\_median,$
  $u\_b\_agg\_amount\_mean, u\_b\_agg\_amount\_std$

- data_u_c.csv:
  $u, i, u\_c\_agg\_count\_max, u\_c\_agg\_count\_median, u\_c\_agg\_count\_mean,$

$u\_c\_agg\_count\_std, u\_c\_agg\_amount\_max, u\_c\_agg\_amount\_median,$
$u\_c\_agg\_amount\_mean, u\_c\_agg\_amount\_std$

2.2 user_agg： 先根据用户进行分类，计算总时间内购买的次数和金额，然后进行aggregation操作。

- user_agg_b.csv:统计购买品牌的次数和金额
  特征：$b, b\_user\_agg\_count\_mean, b\_user\_agg\_count\_std, b\_user\_agg\_count\_max,$
  $b\_user\_agg\_count\_median, b\_user\_agg\_amount\_mean, b\_user\_agg\_amount\_std,$
  $b\_user\_agg\_amount\_max, b\_user\_agg\_amount\_median$

- user_agg_c.csv:统计购买品牌的次数和金额
  特征：$c, c\_user\_agg\_count\_mean, c\_user\_agg\_count\_std, c\_user\_agg\_count\_max,$
  $c\_user\_agg\_count\_median, c\_user\_agg\_amount\_mean, c\_user\_agg\_amount\_std,$
  $c\_user\_agg\_amount\_max, c\_user\_agg\_amount\_median$

- user_agg_i.csv:统计购买品牌的次数和金额
  特征：$i, i\_user\_agg\_count\_mean, i\_user\_agg\_count\_std, i\_user\_agg\_count\_max,$
  $i\_user\_agg\_count\_median, i\_user\_agg\_amount\_mean, i\_user\_agg\_amount\_std,$
  $i\_user\_agg\_amount\_max, i\_user\_agg\_amount\_median$

2.3 brand/category/item AGG：按照u分组，针对单个的b，c，i进行购买次数、购买金额的统计然后做aggregation

- brand_agg.csv:用户购买品牌的统计
  特征: $u, u\_b\_user\_agg\_count\_mean, u\_b\_user\_agg\_count\_std,$
  $u\_b\_user\_agg\_count\_max, u\_b\_user\_agg\_count\_median,$
  $u\_b\_user\_agg\_amount\_mean, u\_b\_user\_agg\_amount\_std,$
  $u\_b\_user\_agg\_amount\_max, u\_b\_user\_agg\_amount\_median$

- category_agg.csv:用户购买商品类别的统计
  特征: $u, u\_c\_user\_agg\_count\_mean, u\_c\_user\_agg\_count\_std,$
  $u\_c\_user\_agg\_count\_max, u\_c\_user\_agg\_count\_median,$
  $u\_c\_user\_agg\_amount\_mean, u\_c\_user\_agg\_amount\_std,$
  $u\_c\_user\_agg\_amount\_max, u\_c\_user\_agg\_amount\_median$

- item_agg.csv:用户购买商品的统计
  特征: $u, u\_i\_user\_agg\_count\_mean, u\_i\_user\_agg\_count\_std,$
  $u\_i\_user\_agg\_count\_max, u\_i\_user\_agg\_count\_median,$

$u\_i\_user\_agg\_amount\_mean, u\_i\_user\_agg\_amount\_std,$
$u\_i\_user\_agg\_amount\_max, u\_i\_user\_agg\_amount\_median$

2.1.3 运行截图

因为6，7，8月份的预测数据的生成过程和2，3，4月份的生成过程基本一样，因此只贴出了生成2，3，4月份特征数据的截图：

```
################################################
create count grouped by vipno...
creat count grouped by user successfully

################################################
create count grouped by brand...
creat count grouped by brand successfully

################################################
create count grouped by category...
creat count grouped by category successfully

################################################
create count grouped by item...
creat count grouped by item successfully

################################################
create count grouped by user and brand...
creat count grouped by user and brand successfully

################################################
create count grouped by user and category...
creat count grouped by user and category successfully

################################################
create count grouped by user and item...
```

Figure 2.1 – 生成count

```
################################################
create diversity_pluno grouped by vipno...
create diversity_pluno grouped by vipno successfully

################################################
create diversity_bndno grouped by vipno...
create diversity_bndno grouped by vipno successfully

################################################
create diversity_dptno grouped by vipno...
create diversity_dptno grouped by vipno successfully

################################################
create diversity_pluno grouped by bndno...
create diversity_pluno grouped by bndno successfully

################################################
create diversity_pluno grouped by dptno...
create diversity_pluno grouped by dptno successfully
```

Figure 2.2 – 生成diversity

```
################################################
create penetration grouped by brand ...
create penetration grouped by brand successfully

################################################
create penetration grouped by category ...
create penetration grouped by category successfully

################################################
create penetration grouped by category ...
create penetration grouped by category successfully
```

Figure 2.3 – 生成penetration

```
################################################
create data_agg_count_u grouped by vipno...
create data_agg_count_u grouped by vipno successfully

################################################
create data_agg_count_u grouped by bndno...
create data_agg_count_b grouped by bndno successfully

################################################
create data_agg_count_c grouped by dptno...
create data_agg_count_c grouped by dptno successfully

################################################
create data_agg_count_i grouped by pluno...
create data_agg_count_i grouped by pluno successfully

################################################
create data_agg_count_u_b grouped by vipno and bndno...
create data_agg_count_u_b grouped by vipno and bndno successfully

################################################
create data_agg_count_u_c grouped by vipno and dptno...
create data_agg_count_u_c grouped by vipno and dptno successfully
```

Figure 2.4 – 生成month_agg_count

```
################################################
create data_agg_diversity_pluno grouped by vipno..
create data_agg_diversity_pluno grouped by vipno successfully

################################################
create data_agg_diversity_bndno grouped by vipno..
create data_agg_diversity_bndno grouped by vipno successfully

################################################
create data_agg_diversity_dptno grouped by vipno..
create data_agg_diversity_dptno grouped by vipno successfully

################################################
create data_agg_diversity_pluno grouped by bndno..
create data_agg_diversity_pluno grouped by bndno successfully

################################################
create data_agg_diversity_pluno grouped by dptno..
create data_agg_diversity_pluno grouped by dptno successfully
```

Figure 2.5 – 生成month_agg_diversity

```
################################################
create data_agg_penetration_vipno grouped by bndno..
create data_agg_diversity_vipno grouped by bndno successfully

################################################
create data_agg_penetration_vipno grouped by dptno..
create data_agg_diversity_vipno grouped by dptno successfully

################################################
create data_agg_penetration_vipno grouped by pluno..
create data_agg_diversity_vipno grouped by pluno successfully
```

Figure 2.6 – 生成month_agg_penetration

```
################################################
create data_user_agg_vipno grouped by bndno..
create data_user_agg_vipno grouped by bndno successfully

################################################
create data_user_agg_dptno grouped by vipno..
create data_user_agg_vipno grouped by dptno successfully

################################################
create data_user_agg_vipno grouped by pluno..
create data_user_agg_vipno grouped by pluno successfully
```

Figure 2.7 – 生成user_agg

```
################################################
create data_user_agg_bndno grouped by vipno..
create data_user_agg_bndno grouped by vipno successfully

################################################
create data_user_agg_dptno grouped by vipno..
create data_user_agg_dptno grouped by vipno successfully

################################################
create data_user_agg_pluno grouped by vipno..
create data_user_agg_pluno grouped by vipno successfully
```

Figure 2.8 – 生成multiple_agg

2.2 b 利用特征预测用户购买商品

2.2.1 题目说明

这道题要求提取出2，3，4月份的vipno - pluno的所有键值对并提取其特征。根据已经提取出来的特征，可以融合上述提取出的特征进行训练。
提取出5月份购买记录中的vipno - pluno，若它在2，3，4月份曾经出现过，则在2，3，4月份的数据中标记1，没有则标记0.这样构成了训练模型的标签。
由此可见，这是一个二分类问题。然而经过label的输出，我们可以看到：

> 有405个正样本标签，8397个负样本标签

在正负样本数量差距悬殊的时候，使用accuracy进行模型预测已经不能很好的表示预测结果了，尤其是正样本数过少的情况下。因此，采用ROC曲线进行分类器性能的刻画。
与precision， recall， f1 score 一样，ROC曲线依从confusion matrix进行结果的刻画。该曲线以false positive的比率作为横轴，true positive的比率作为纵轴，在FP较小的时候TP越大，即图像越偏向左上角，说明预测出假结果很低的情况下能预测出来很高的正结果，说明这个分类器性能就更好一些。下面是precision， recall， f1 score 的运算公式：

$$
\begin{aligned}
precision &= \frac{TP}{(TP+FP)} \\
recall &= \frac{TP}{TP+FN} \\
f1 &= \frac{2 \times precision \times recall}{precision+recall}
\end{aligned}
$$

2.2.2 步骤说明

- 首先根据已经提取的特征做合并，得到2，3，4月份的数据，然后再得到5月份的数据做label。两者合并生成训练测试数据。
- 使用KFold进行10折交叉验证生成数据集。
- 对于每一份数据集进行七个分类器的训练。
- 得到每个分类器的precision、recall、f1
- 得到ROC曲线图

2.2.3 运行截图

以下是10次交叉验证7个分类器的precision， recall 和f1 预测值

```
################################################################################
decision tree:          precision: 0.160000 recall: 0.170000    f1 score: 0.160000
k neighbor:             precision: 0.090000 recall: 0.100000    f1 score: 0.090000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.600000 recall: 0.070000    f1 score: 0.130000
Bagging Classifier:     precision: 0.470000 recall: 0.170000    f1 score: 0.250000
Adaboost Classifier:    precision: 0.400000 recall: 0.050000    f1 score: 0.090000
GradientBoost Classifier:   precision: 0.500000 recall: 0.050000    f1 score: 0.090000
```

Figure 2.9 – log 1

```
################################################################################
decision tree:          precision: 0.140000 recall: 0.130000    f1 score: 0.140000
k neighbor:             precision: 0.120000 recall: 0.120000    f1 score: 0.120000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.500000 recall: 0.020000    f1 score: 0.040000
Bagging Classifier:     precision: 0.560000 recall: 0.100000    f1 score: 0.160000
Adaboost Classifier:    precision: 0.800000 recall: 0.080000    f1 score: 0.140000
GradientBoost Classifier:   precision: 0.330000 recall: 0.020000    f1 score: 0.040000
```

Figure 2.10 – log 2

```
################################################################################
decision tree:          precision: 0.110000 recall: 0.070000    f1 score: 0.090000
k neighbor:             precision: 0.150000 recall: 0.110000    f1 score: 0.130000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.400000 recall: 0.040000    f1 score: 0.070000
Bagging Classifier:     precision: 0.450000 recall: 0.090000    f1 score: 0.150000
Adaboost Classifier:    precision: 0.300000 recall: 0.050000    f1 score: 0.090000
GradientBoost Classifier:   precision: 0.430000 recall: 0.050000    f1 score: 0.100000
```

Figure 2.11 – log 3

```
################################################################################
decision tree:          precision: 0.140000 recall: 0.160000    f1 score: 0.150000
k neighbor:             precision: 0.180000 recall: 0.220000    f1 score: 0.200000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.500000 recall: 0.040000    f1 score: 0.080000
Bagging Classifier:     precision: 0.200000 recall: 0.040000    f1 score: 0.070000
Adaboost Classifier:    precision: 0.300000 recall: 0.070000    f1 score: 0.110000
GradientBoost Classifier:   precision: 0.500000 recall: 0.070000    f1 score: 0.120000
```

Figure 2.12 – log 4

```
################################################################################
decision tree:          precision: 0.170000 recall: 0.280000    f1 score: 0.210000
k neighbor:             precision: 0.140000 recall: 0.220000    f1 score: 0.170000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.400000 recall: 0.060000    f1 score: 0.100000
Bagging Classifier:     precision: 0.250000 recall: 0.060000    f1 score: 0.090000
Adaboost Classifier:    precision: 0.290000 recall: 0.060000    f1 score: 0.090000
GradientBoost Classifier:   precision: 0.670000 recall: 0.060000    f1 score: 0.100000
```

Figure 2.13 – log 5

```
################################################################################
decision tree:          precision: 0.150000 recall: 0.140000    f1 score: 0.140000
k neighbor:             precision: 0.190000 recall: 0.180000    f1 score: 0.180000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.400000 recall: 0.040000    f1 score: 0.060000
Bagging Classifier:     precision: 0.330000 recall: 0.040000    f1 score: 0.060000
Adaboost Classifier:    precision: 0.500000 recall: 0.050000    f1 score: 0.100000
GradientBoost Classifier:   precision: 0.750000 recall: 0.050000    f1 score: 0.100000
```

Figure 2.14 – log 6

```
################################################################################
decision tree:          precision: 0.210000 recall: 0.280000    f1 score: 0.240000
k neighbor:             precision: 0.180000 recall: 0.190000    f1 score: 0.190000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.400000 recall: 0.040000    f1 score: 0.080000
Bagging Classifier:     precision: 0.330000 recall: 0.090000    f1 score: 0.140000
Adaboost Classifier:    precision: 0.670000 recall: 0.040000    f1 score: 0.080000
GradientBoost Classifier:   precision: 0.500000 recall: 0.020000    f1 score: 0.040000
```

Figure 2.15 – log 7

```
################################################################################
decision tree:          precision: 0.270000 recall: 0.320000    f1 score: 0.290000
k neighbor:             precision: 0.070000 recall: 0.090000    f1 score: 0.080000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.500000 recall: 0.070000    f1 score: 0.120000
Bagging Classifier:     precision: 0.400000 recall: 0.050000    f1 score: 0.080000
Adaboost Classifier:    precision: 0.450000 recall: 0.110000    f1 score: 0.180000
GradientBoost Classifier:   precision: 0.400000 recall: 0.050000    f1 score: 0.080000
```

Figure 2.16 – log 8

```
################################################################################
decision tree:          precision: 0.200000 recall: 0.190000    f1 score: 0.200000
k neighbor:             precision: 0.250000 recall: 0.170000    f1 score: 0.210000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.710000 recall: 0.080000    f1 score: 0.140000
Bagging Classifier:     precision: 0.500000 recall: 0.060000    f1 score: 0.110000
Adaboost Classifier:    precision: 0.570000 recall: 0.060000    f1 score: 0.110000
GradientBoost Classifier:   precision: 0.500000 recall: 0.050000    f1 score: 0.090000
```

Figure 2.17 – log 9

```
################################################################################
decision tree:          precision: 0.190000 recall: 0.200000    f1 score: 0.190000
k neighbor:             precision: 0.180000 recall: 0.170000    f1 score: 0.170000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.600000 recall: 0.060000    f1 score: 0.100000
Bagging Classifier:     precision: 0.360000 recall: 0.090000    f1 score: 0.150000
Adaboost Classifier:    precision: 0.710000 recall: 0.090000    f1 score: 0.160000
GradientBoost Classifier:   precision: 0.400000 recall: 0.040000    f1 score: 0.070000
```

Figure 2.18 – log 10

2.2.4 性能评价

首先贴出ROC曲线图：

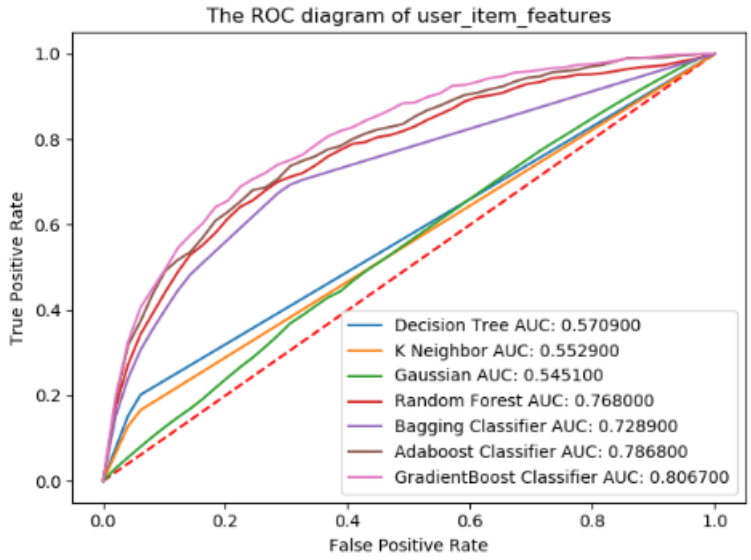

Figure 2.19 – Caption

- 由ROC整体曲线的分布来看，三种弱分类器：Decision Tree, Gaussian Naive Bayes 和 K Neighbor 算法的效果比较差这是因为特征数量很大(80+)，而且正负样本差距很大，弱分类器很难对所有特征进行综合评估。相比之下，ensemble分类器无论是迭代还是并行弱分类器，都能兼顾各种特征所取得的影响，效果会好。

- 总体上来看，ROC与x轴围城的面积：AUC大体在0.7左右，这是分类效果不好的信号（正常在0.7- 0.9），这也是由于正样本数量过少造成的。

2.3 ci 用户特征数据

本次提取用户以及用户相关的特征，共65个特征数量。

2.3.1 运行截图

```
################################################################################
decision tree:          precision: 0.810000 recall: 0.690000    f1 score: 0.750000
k neighbor:             precision: 0.830000 recall: 0.750000    f1 score: 0.790000
Gaussian:               precision: 0.760000 recall: 1.000000    f1 score: 0.860000
Random Forest:          precision: 0.830000 recall: 0.940000    f1 score: 0.880000
Bagging Classifier:     precision: 0.860000 recall: 0.940000    f1 score: 0.900000
Adaboost Classifier:    precision: 0.820000 recall: 0.880000    f1 score: 0.850000
GradientBoost Classifier:   precision: 0.840000 recall: 1.000000    f1 score: 0.910000
```

Figure 2.20 – log 1

```
################################################################################
decision tree:          precision: 0.790000 recall: 0.770000    f1 score: 0.780000
k neighbor:             precision: 0.710000 recall: 0.730000    f1 score: 0.720000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.760000 recall: 0.830000    f1 score: 0.790000
Bagging Classifier:     precision: 0.790000 recall: 0.730000    f1 score: 0.760000
Adaboost Classifier:    precision: 0.790000 recall: 0.870000    f1 score: 0.830000
GradientBoost Classifier:   precision: 0.780000 recall: 0.830000    f1 score: 0.810000
```

Figure 2.21 – log 2

```
################################################################################
decision tree:          precision: 0.630000 recall: 0.710000    f1 score: 0.670000
k neighbor:             precision: 0.540000 recall: 0.620000    f1 score: 0.580000
Gaussian:               precision: 0.570000 recall: 1.000000    f1 score: 0.730000
Random Forest:          precision: 0.720000 recall: 0.960000    f1 score: 0.820000
Bagging Classifier:     precision: 0.760000 recall: 0.920000    f1 score: 0.830000
Adaboost Classifier:    precision: 0.700000 recall: 0.960000    f1 score: 0.810000
GradientBoost Classifier:   precision: 0.680000 recall: 0.880000    f1 score: 0.760000
```

Figure 2.22 – log 3

```
################################################################################
decision tree:          precision: 0.870000 recall: 0.840000    f1 score: 0.850000
k neighbor:             precision: 0.760000 recall: 0.810000    f1 score: 0.780000
Gaussian:               precision: 0.740000 recall: 1.000000    f1 score: 0.850000
Random Forest:          precision: 0.840000 recall: 0.870000    f1 score: 0.860000
Bagging Classifier:     precision: 0.760000 recall: 0.710000    f1 score: 0.730000
Adaboost Classifier:    precision: 0.790000 recall: 0.840000    f1 score: 0.810000
GradientBoost Classifier:   precision: 0.810000 recall: 0.840000    f1 score: 0.830000
```

Figure 2.23 – log 4

```
################################################################################
decision tree:          precision: 0.810000 recall: 0.700000    f1 score: 0.750000
k neighbor:             precision: 0.710000 recall: 0.900000    f1 score: 0.790000
Gaussian:               precision: 0.710000 recall: 1.000000    f1 score: 0.830000
Random Forest:          precision: 0.780000 recall: 0.970000    f1 score: 0.870000
Bagging Classifier:     precision: 0.800000 recall: 0.800000    f1 score: 0.800000
Adaboost Classifier:    precision: 0.760000 recall: 0.870000    f1 score: 0.810000
GradientBoost Classifier:   precision: 0.820000 recall: 0.930000    f1 score: 0.870000
```

Figure 2.24 – log 5

```
################################################################################
decision tree:          precision: 0.870000 recall: 0.900000    f1 score: 0.890000
k neighbor:             precision: 0.810000 recall: 0.730000    f1 score: 0.770000
Gaussian:               precision: 0.710000 recall: 1.000000    f1 score: 0.830000
Random Forest:          precision: 0.770000 recall: 1.000000    f1 score: 0.870000
Bagging Classifier:     precision: 0.750000 recall: 0.900000    f1 score: 0.820000
Adaboost Classifier:    precision: 0.760000 recall: 0.930000    f1 score: 0.840000
GradientBoost Classifier:   precision: 0.760000 recall: 0.970000    f1 score: 0.850000
```

Figure 2.25 – log 6

```
################################################################################
decision tree:          precision: 0.720000 recall: 0.720000    f1 score: 0.720000
k neighbor:             precision: 0.640000 recall: 0.720000    f1 score: 0.680000
Gaussian:               precision: 0.610000 recall: 1.000000    f1 score: 0.760000
Random Forest:          precision: 0.710000 recall: 0.880000    f1 score: 0.790000
Bagging Classifier:     precision: 0.780000 recall: 0.840000    f1 score: 0.810000
Adaboost Classifier:    precision: 0.680000 recall: 0.840000    f1 score: 0.750000
GradientBoost Classifier:   precision: 0.700000 recall: 0.840000    f1 score: 0.760000
```

Figure 2.26 – log 7

```
################################################################################
decision tree:          precision: 0.860000 recall: 0.710000    f1 score: 0.770000
k neighbor:             precision: 0.830000 recall: 0.710000    f1 score: 0.760000
Gaussian:               precision: 0.820000 recall: 0.970000    f1 score: 0.890000
Random Forest:          precision: 0.860000 recall: 0.940000    f1 score: 0.900000
Bagging Classifier:     precision: 0.900000 recall: 0.790000    f1 score: 0.840000
Adaboost Classifier:    precision: 0.900000 recall: 0.820000    f1 score: 0.860000
GradientBoost Classifier:   precision: 0.910000 recall: 0.940000    f1 score: 0.930000
```

Figure 2.27 – log 8

```
################################################################################
decision tree:          precision: 0.930000 recall: 0.780000    f1 score: 0.850000
k neighbor:             precision: 0.880000 recall: 0.640000    f1 score: 0.740000
Gaussian:               precision: 0.880000 recall: 0.970000    f1 score: 0.920000
Random Forest:          precision: 0.890000 recall: 0.940000    f1 score: 0.920000
Bagging Classifier:     precision: 0.890000 recall: 0.890000    f1 score: 0.890000
Adaboost Classifier:    precision: 0.920000 recall: 0.940000    f1 score: 0.930000
GradientBoost Classifier:   precision: 0.900000 recall: 1.000000    f1 score: 0.950000
```

Figure 2.28 – log 9

2.3.2 性能说明

下面是7个分类器的ROC曲线图：

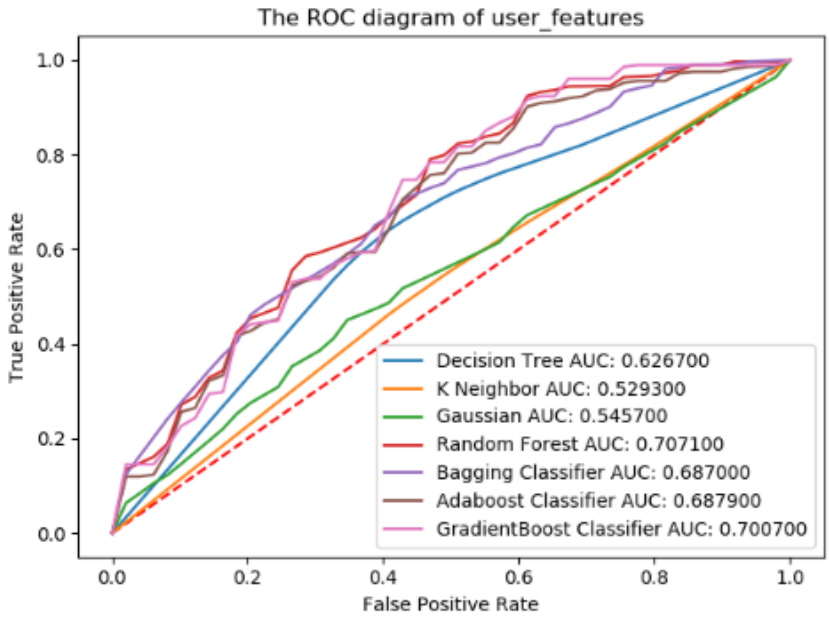

Figure 2.29 – Caption

- 决策树分类器性能有所回升，四种组合分类器性能较好，另外两种弱分类器性能不好。
- 与u - i对分析相比，这次roc曲线围成的面积并不是很大，说明预测效果不好。这跟数据集数量小有关。
- 综合上述两道题，比较适合这个数据的分类器是Gradient Boost Classifier 和Random Forest Classifier

2.4 cii 用户 - 品牌分组特征提取训练

这道题提取用户 - 品牌为对的数据进行特征提取, 共83个特征

2.4.1 步骤说明

- 根据先前生成好的特征进行组合（u - b组合）
- 十折交叉验证区分训练集测试集
- 7个分类器进行训练、预测，算出precision recall f1
- 画出roc曲线图
- 根据6，7，8月份提取相同的特征，进行预测

2.4.2 运行截图

```
################################################################################
decision tree:          precision: 0.240000 recall: 0.210000    f1 score: 0.220000
k neighbor:             precision: 0.320000 recall: 0.280000    f1 score: 0.300000
Gaussian:               precision: 0.100000 recall: 1.000000    f1 score: 0.180000
Random Forest:          precision: 0.600000 recall: 0.150000    f1 score: 0.240000
Bagging Classifier:     precision: 0.710000 recall: 0.130000    f1 score: 0.220000
Adaboost Classifier:    precision: 0.330000 recall: 0.100000    f1 score: 0.160000
GradientBoost Classifier:   precision: 0.800000 recall: 0.100000    f1 score: 0.180000
```

Figure 2.30 – log 1

```
################################################################################
decision tree:          precision: 0.180000 recall: 0.180000    f1 score: 0.180000
k neighbor:             precision: 0.170000 recall: 0.130000    f1 score: 0.150000
Gaussian:               precision: 0.100000 recall: 1.000000    f1 score: 0.180000
Random Forest:          precision: 0.670000 recall: 0.110000    f1 score: 0.180000
Bagging Classifier:     precision: 0.450000 recall: 0.130000    f1 score: 0.200000
Adaboost Classifier:    precision: 0.330000 recall: 0.110000    f1 score: 0.160000
GradientBoost Classifier:   precision: 0.600000 recall: 0.080000    f1 score: 0.140000
```

Figure 2.31 – log 2

```
################################################################################
decision tree:          precision: 0.210000 recall: 0.280000    f1 score: 0.240000
k neighbor:             precision: 0.240000 recall: 0.280000    f1 score: 0.260000
Gaussian:               precision: 0.090000 recall: 1.000000    f1 score: 0.170000
Random Forest:          precision: 0.670000 recall: 0.060000    f1 score: 0.100000
Bagging Classifier:     precision: 0.500000 recall: 0.110000    f1 score: 0.180000
Adaboost Classifier:    precision: 0.770000 recall: 0.280000    f1 score: 0.410000
GradientBoost Classifier:   precision: 1.000000 recall: 0.060000    f1 score: 0.110000
```

Figure 2.32 – log 3

```
################################################################################
decision tree:          precision: 0.260000 recall: 0.370000    f1 score: 0.310000
k neighbor:             precision: 0.140000 recall: 0.130000    f1 score: 0.140000
Gaussian:               precision: 0.080000 recall: 1.000000    f1 score: 0.140000
Random Forest:          precision: 0.140000 recall: 0.030000    f1 score: 0.050000
Bagging Classifier:     precision: 0.200000 recall: 0.070000    f1 score: 0.100000
Adaboost Classifier:    precision: 0.500000 recall: 0.130000    f1 score: 0.210000
GradientBoost Classifier:   precision: 0.120000 recall: 0.030000    f1 score: 0.050000
```

Figure 2.33 – log 4

```
################################################################################
decision tree:          precision: 0.260000 recall: 0.220000    f1 score: 0.240000
k neighbor:             precision: 0.150000 recall: 0.090000    f1 score: 0.110000
Gaussian:               precision: 0.120000 recall: 1.000000    f1 score: 0.210000
Random Forest:          precision: 0.290000 recall: 0.040000    f1 score: 0.080000
Bagging Classifier:     precision: 0.670000 recall: 0.130000    f1 score: 0.220000
Adaboost Classifier:    precision: 0.560000 recall: 0.200000    f1 score: 0.290000
GradientBoost Classifier:   precision: 0.500000 recall: 0.070000    f1 score: 0.120000
```

Figure 2.34 – log 5

```
################################################################################
decision tree:          precision: 0.190000 recall: 0.200000    f1 score: 0.200000
k neighbor:             precision: 0.220000 recall: 0.270000    f1 score: 0.240000
Gaussian:               precision: 0.080000 recall: 1.000000    f1 score: 0.140000
Random Forest:          precision: 0.330000 recall: 0.070000    f1 score: 0.110000
Bagging Classifier:     precision: 0.170000 recall: 0.030000    f1 score: 0.060000
Adaboost Classifier:    precision: 0.200000 recall: 0.070000    f1 score: 0.100000
GradientBoost Classifier:   precision: 0.000000 recall: 0.000000    f1 score: 0.000000
```

Figure 2.35 – log 6

```
################################################################################
decision tree:          precision: 0.300000 recall: 0.270000    f1 score: 0.290000
k neighbor:             precision: 0.170000 recall: 0.140000    f1 score: 0.150000
Gaussian:               precision: 0.110000 recall: 1.000000    f1 score: 0.200000
Random Forest:          precision: 1.000000 recall: 0.090000    f1 score: 0.170000
Bagging Classifier:     precision: 0.300000 recall: 0.070000    f1 score: 0.110000
Adaboost Classifier:    precision: 0.330000 recall: 0.090000    f1 score: 0.140000
GradientBoost Classifier:   precision: 1.000000 recall: 0.050000    f1 score: 0.090000
```

Figure 2.36 – log 7

```
################################################################################
decision tree:          precision: 0.190000 recall: 0.120000    f1 score: 0.140000
k neighbor:             precision: 0.250000 recall: 0.140000    f1 score: 0.180000
Gaussian:               precision: 0.110000 recall: 1.000000    f1 score: 0.190000
Random Forest:          precision: 1.000000 recall: 0.140000    f1 score: 0.250000
Bagging Classifier:     precision: 0.640000 recall: 0.170000    f1 score: 0.260000
Adaboost Classifier:    precision: 0.570000 recall: 0.100000    f1 score: 0.160000
GradientBoost Classifier:   precision: 1.000000 recall: 0.190000    f1 score: 0.320000
```

Figure 2.37 – log 8

```
################################################################################
decision tree:          precision: 0.190000 recall: 0.240000    f1 score: 0.210000
k neighbor:             precision: 0.150000 recall: 0.180000    f1 score: 0.160000
Gaussian:               precision: 0.090000 recall: 1.000000    f1 score: 0.160000
Random Forest:          precision: 0.750000 recall: 0.090000    f1 score: 0.160000
Bagging Classifier:     precision: 0.670000 recall: 0.120000    f1 score: 0.200000
Adaboost Classifier:    precision: 0.670000 recall: 0.120000    f1 score: 0.200000
GradientBoost Classifier:   precision: 1.000000 recall: 0.150000    f1 score: 0.260000
```

Figure 2.38 – log 9

```
################################################################################
decision tree:          precision: 0.320000 recall: 0.340000    f1 score: 0.330000
k neighbor:             precision: 0.160000 recall: 0.130000    f1 score: 0.140000
Gaussian:               precision: 0.100000 recall: 1.000000    f1 score: 0.180000
Random Forest:          precision: 0.360000 recall: 0.130000    f1 score: 0.190000
Bagging Classifier:     precision: 0.180000 recall: 0.050000    f1 score: 0.080000
Adaboost Classifier:    precision: 0.290000 recall: 0.110000    f1 score: 0.150000
GradientBoost Classifier:   precision: 0.210000 recall: 0.080000    f1 score: 0.120000
```

Figure 2.39 – log 10

2.4.3 性能分析

下面是7个分类器的roc曲线图：

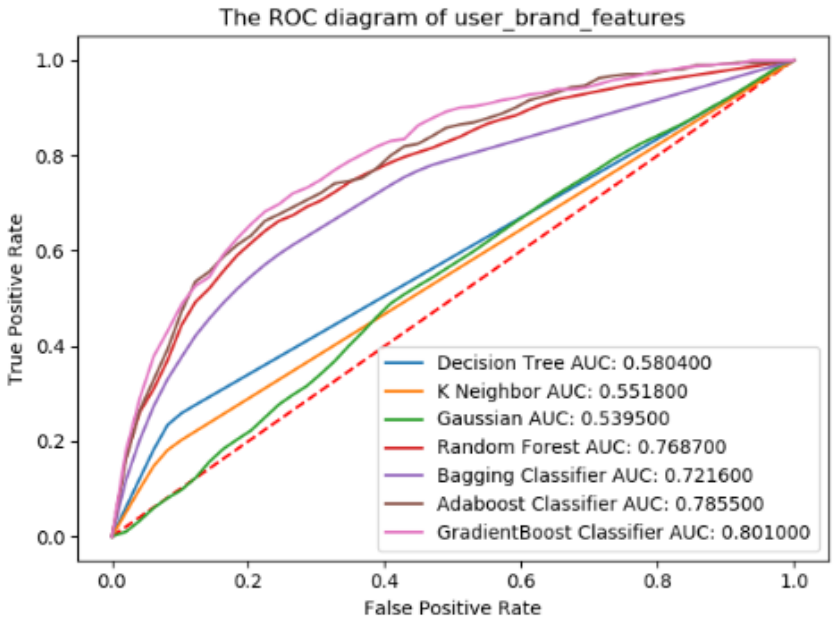

Figure 2.40 – user brand ROC diagram

- 效果最好的分类器是GradientBoost分类器。
- 效果最差的是Gaussian Naive Classifier。
- 和之前相同，ensemble classifier的效果是要好于弱分类器的。

2.5 ciii 用户 - 类别特征分类

这道题提取用户 - 品牌为对的数据进行特征提取, 共83个特征

2.5.1 步骤说明

- 根据先前生成好的特征进行组合（u - c组合）
- 十折交叉验证区分训练集测试集
- 7个分类器进行训练、预测，算出precision recall f1
- 画出roc曲线图
- 根据6，7，8月份提取相同的特征，进行预测

2.5.2 运行截图

```
################################################################################
decision tree:          precision: 0.260000 recall: 0.290000    f1 score: 0.280000
k neighbor:             precision: 0.230000 recall: 0.240000    f1 score: 0.240000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.470000 recall: 0.120000    f1 score: 0.190000
Bagging Classifier:     precision: 0.500000 recall: 0.140000    f1 score: 0.220000
Adaboost Classifier:    precision: 0.460000 recall: 0.080000    f1 score: 0.130000
GradientBoost Classifier:   precision: 0.580000 recall: 0.090000    f1 score: 0.160000
```

Figure 2.41 – log 1

```
################################################################################
decision tree:          precision: 0.150000 recall: 0.160000    f1 score: 0.160000
k neighbor:             precision: 0.180000 recall: 0.190000    f1 score: 0.190000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.500000 recall: 0.090000    f1 score: 0.160000
Bagging Classifier:     precision: 0.240000 recall: 0.050000    f1 score: 0.090000
Adaboost Classifier:    precision: 0.430000 recall: 0.080000    f1 score: 0.140000
GradientBoost Classifier:   precision: 0.380000 recall: 0.040000    f1 score: 0.070000
```

Figure 2.42 – log 2

```
################################################################################
decision tree:          precision: 0.270000 recall: 0.270000    f1 score: 0.270000
k neighbor:             precision: 0.300000 recall: 0.310000    f1 score: 0.300000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.710000 recall: 0.120000    f1 score: 0.210000
Bagging Classifier:     precision: 0.480000 recall: 0.120000    f1 score: 0.200000
Adaboost Classifier:    precision: 0.310000 recall: 0.060000    f1 score: 0.100000
GradientBoost Classifier:   precision: 0.460000 recall: 0.070000    f1 score: 0.130000
```

Figure 2.43 – log 3

```
################################################################################
decision tree:          precision: 0.210000 recall: 0.240000    f1 score: 0.220000
k neighbor:             precision: 0.240000 recall: 0.210000    f1 score: 0.230000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.430000 recall: 0.080000    f1 score: 0.130000
Bagging Classifier:     precision: 0.440000 recall: 0.110000    f1 score: 0.170000
Adaboost Classifier:    precision: 0.310000 recall: 0.050000    f1 score: 0.090000
GradientBoost Classifier:   precision: 0.600000 recall: 0.040000    f1 score: 0.080000
```

Figure 2.44 – log 4

```
################################################################################
decision tree:          precision: 0.230000 recall: 0.250000    f1 score: 0.240000
k neighbor:             precision: 0.180000 recall: 0.200000    f1 score: 0.190000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.600000 recall: 0.090000    f1 score: 0.160000
Bagging Classifier:     precision: 0.460000 recall: 0.090000    f1 score: 0.150000
Adaboost Classifier:    precision: 0.500000 recall: 0.120000    f1 score: 0.200000
GradientBoost Classifier:   precision: 0.800000 recall: 0.060000    f1 score: 0.110000
```

Figure 2.45 – log 5

```
################################################################################
decision tree:          precision: 0.280000 recall: 0.320000    f1 score: 0.300000
k neighbor:             precision: 0.330000 recall: 0.240000    f1 score: 0.280000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.460000 recall: 0.070000    f1 score: 0.130000
Bagging Classifier:     precision: 0.380000 recall: 0.110000    f1 score: 0.170000
Adaboost Classifier:    precision: 0.550000 recall: 0.070000    f1 score: 0.130000
GradientBoost Classifier:   precision: 0.780000 recall: 0.090000    f1 score: 0.150000
```

Figure 2.46 – log 6

```
################################################################################
decision tree:          precision: 0.240000 recall: 0.280000    f1 score: 0.250000
k neighbor:             precision: 0.190000 recall: 0.190000    f1 score: 0.190000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.300000 recall: 0.080000    f1 score: 0.130000
Bagging Classifier:     precision: 0.320000 recall: 0.120000    f1 score: 0.180000
Adaboost Classifier:    precision: 0.580000 recall: 0.100000    f1 score: 0.170000
GradientBoost Classifier:   precision: 0.220000 recall: 0.030000    f1 score: 0.050000
```

Figure 2.47 – log 7

```
################################################################################
decision tree:          precision: 0.220000 recall: 0.210000    f1 score: 0.210000
k neighbor:             precision: 0.290000 recall: 0.210000    f1 score: 0.240000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.500000 recall: 0.070000    f1 score: 0.130000
Bagging Classifier:     precision: 0.430000 recall: 0.070000    f1 score: 0.120000
Adaboost Classifier:    precision: 0.530000 recall: 0.110000    f1 score: 0.180000
GradientBoost Classifier:   precision: 0.460000 recall: 0.070000    f1 score: 0.130000
```

Figure 2.48 – log 8

```
################################################################################
decision tree:          precision: 0.290000 recall: 0.280000    f1 score: 0.290000
k neighbor:             precision: 0.270000 recall: 0.230000    f1 score: 0.250000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.440000 recall: 0.100000    f1 score: 0.160000
Bagging Classifier:     precision: 0.500000 recall: 0.160000    f1 score: 0.240000
Adaboost Classifier:    precision: 0.500000 recall: 0.100000    f1 score: 0.160000
GradientBoost Classifier:   precision: 0.530000 recall: 0.110000    f1 score: 0.180000
```

Figure 2.49 – log 9

```
################################################################################
decision tree:          precision: 0.250000 recall: 0.260000    f1 score: 0.260000
k neighbor:             precision: 0.230000 recall: 0.210000    f1 score: 0.220000
Gaussian:               precision: 0.000000 recall: 0.000000    f1 score: 0.000000
Random Forest:          precision: 0.530000 recall: 0.120000    f1 score: 0.200000
Bagging Classifier:     precision: 0.480000 recall: 0.140000    f1 score: 0.220000
Adaboost Classifier:    precision: 0.600000 recall: 0.120000    f1 score: 0.210000
GradientBoost Classifier:   precision: 0.780000 recall: 0.100000    f1 score: 0.170000
```

Figure 2.50 – log 10

2.5.3 性能分析

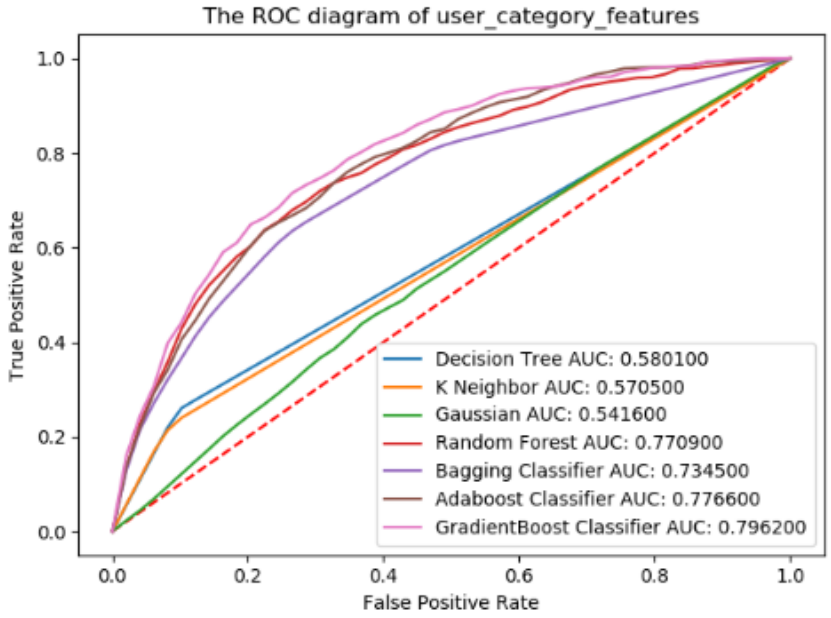

Figure 2.51 – Caption

- 决策树分类器性能有所回升，四种组合分类器性能较好，另外两种弱分类器性能不好。
- 与u - i对分析相比，这次roc曲线围成的面积并不是很大，说明预测效果不好。这跟数据集数量小有关。
- 综合上述两道题，比较适合这个数据的分类器是Gradient Boost Classifier 和Random Forest Classifier

2.6 civ 聚类与用户购买金额

2.6.1 题目说明

这道题探讨user和购买金额amt的关系。由于金额是连续性的变量，这里采用回归模型进行训练和预测：

- DecisionRegressor
- RandomForestRegressor
- BaggingRegressor
- KNeighborRegressor
- AdaboostRegressor
- GradientBoostingRegressor

2.6.2 步骤说明

- 特征提取。
- 十折交叉验证分出训练集数据集。
- 以amount作为label进行训练和检测，计算误差值。
- 构建6个回归模型的CDF图。

2.6.3 运行截图

```
################################################################################
decision tree average error: 101.217619
k neighbor regression average error: 84.177000
random forest regression average error: 91.508810
bagging regression average error: 86.435595
adaboost regression average error: 77.449169
gradient boost regression average error: 86.577656
```

Figure 2.52 – log 1

```
################################################################################
decision tree average error: 131.199762
k neighbor regression average error: 90.526810
random forest regression average error: 99.455000
bagging regression average error: 101.651381
adaboost regression average error: 104.561633
gradient boost regression average error: 98.274979
```

Figure 2.53 – log 2

```
################################################################################
decision tree average error: 180.492619
k neighbor regression average error: 130.458524
random forest regression average error: 126.527310
bagging regression average error: 124.304000
adaboost regression average error: 128.516639
gradient boost regression average error: 129.124462
```

Figure 2.54 – log 3

```
################################################################################
decision tree average error: 138.141429
k neighbor regression average error: 71.673619
random forest regression average error: 77.816619
bagging regression average error: 89.219714
adaboost regression average error: 100.733015
gradient boost regression average error: 81.521527
```

Figure 2.55 – log 4

```
################################################################################
decision tree average error: 141.450952
k neighbor regression average error: 100.099667
random forest regression average error: 115.358667
bagging regression average error: 111.567143
adaboost regression average error: 124.999767
gradient boost regression average error: 115.916608
```

Figure 2.56 – log 5

```
################################################################################
decision tree average error: 102.267619
k neighbor regression average error: 105.821000
random forest regression average error: 79.181952
bagging regression average error: 80.287976
adaboost regression average error: 84.647720
gradient boost regression average error: 75.517753
```

Figure 2.57 – log 6

```
################################################################################
decision tree average error: 99.456341
k neighbor regression average error: 100.715317
random forest regression average error: 97.652024
bagging regression average error: 84.066878
adaboost regression average error: 97.020153
gradient boost regression average error: 91.161504
```

Figure 2.58 – log 7

```
################################################################################
decision tree average error: 75.917317
k neighbor regression average error: 83.762146
random forest regression average error: 74.037463
bagging regression average error: 82.384878
adaboost regression average error: 98.039315
gradient boost regression average error: 70.501235
```

Figure 2.59 – log 8

```
################################################################################
decision tree average error: 107.433902
k neighbor regression average error: 105.116537
random forest regression average error: 90.569098
bagging regression average error: 93.523659
adaboost regression average error: 92.436507
gradient boost regression average error: 104.687246
```

Figure 2.60 – log 9

```
################################################################################
decision tree average error: 187.717561
k neighbor regression average error: 79.677854
random forest regression average error: 79.577659
bagging regression average error: 86.916805
adaboost regression average error: 86.637196
gradient boost regression average error: 102.522919
```

Figure 2.61 – log 10

2.6.4 比较讨论

下面是六个回归模型的CDF图：

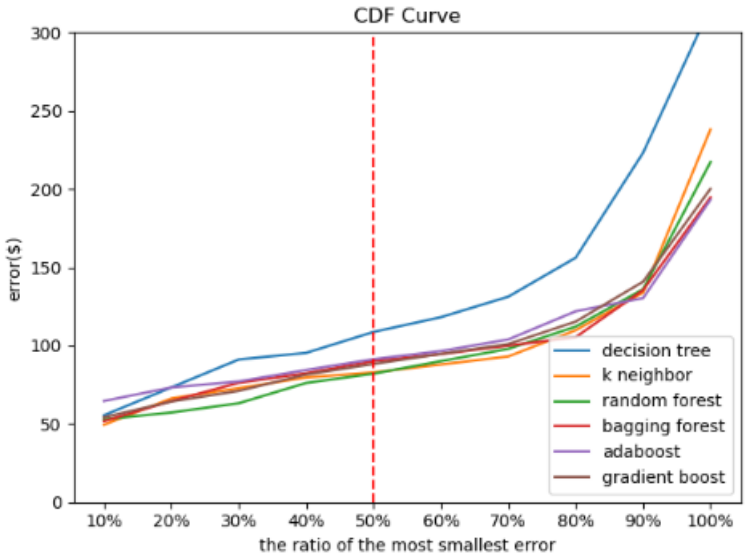

Figure 2.62 – cdf diagram

- 由图可见，decision tree效果最差，误差最高，其余五个回归模型表现均等。
- 效果最好的分类器是GradientBoosting回归器
- 回归模型整体表现偏差比较大，与数据量不大，标签

2.7 cv 预测购买商品

2.7.1 题目说明

这道题使用前四问中生成的预测数据，来预测用户可能会购买的商品。
我的思路如下：

- 合并ci到civ中预测的数据：即将用户,用户和商品品牌， 用户和商品类别中均为yes的部分求交集，然后和用户购买金额做做连接，得到的数据表示：预测用户这个月可能会花这么多钱去买这个品牌、种类的商品。
- 从总体的数据集中获取所有的品牌 - 类别 - 商品 - 价格的元组对。
- 将上述元组对和步骤一得到的数据做交集，可以得出某个用户可能会购买的商品、类别、品牌、价格和他愿意画出的价格。这时，利用价格约束，即商品价格和用户购买价格之差不大于20作为衡量标准，进一步选出用户可能购买的商品。
- 统计每个用户可能购买的商品

2.7.2 处理步骤

处理步骤大致如上述思路。其中特别提及操作中的几点：

- 由于ci - civ是利用6种分类器分别预测结果，可以看出预测的结果并不尽如人意。而实际的结果是正样本过于少以至于很难提取出有效的用户 - 品牌 - 类别的元素对。因此在第五问中我综合了6种分类器预测出来的所有情况（取并集）作为预测可能的结果。
- 由于预测出来的价格普遍偏高，因此我在取价格的时候区间范围取得比较大。
- 最终统计出来的是用户可能会购买的商品，并不是全部

2.7.3 运行截图

以下是运行出的结果截图。

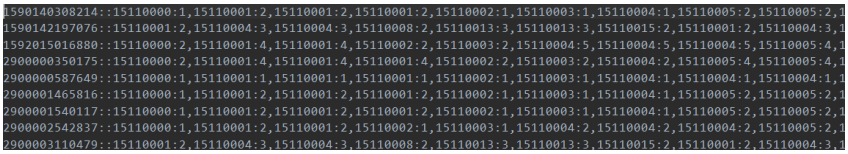

Figure 2.63 – 预测结果

2.7.4 分析讨论

- 和b问直接求得购买商品的过程相比，c问的过程会分段进行求解：即求出用户 - 品牌， 用户 - 种类， 用户 - 购买金额 然后进行综合评断，在数据量大且正负样本较为均匀的情况下是比b问好的选择。

- 实际效果来看，c问和b问预测出来的结果都十分的稀少，这说明：

  1. 负样本过大，干扰了模型的训练过程，这从AUC较低就能够看得出来。
  2.数据处理方式还是比较简单，可以对模型进行进一步的优化。