【半程彩蛋】选手分享，思路解答！

官方业务数据分析师   |   C/C++

**开源是一种文化，一种精神，分享对于时代创新与进步有着重要意义，一直以来，DC都在鼓励参赛者们发扬开源精神，在分享中相互学习。**

**本次半程彩蛋来自---阮耀灿**，这是史上最详尽的一次分享，从数据处理到基本思路到解决策略都一一展示，希望大家能从中得到些启发，我们也期待更多参赛者来分享开源！

***祝您玩儿的愉快～***

--------------------------------------------

备注：

分析工具：anaconda上写python，画图用到pyecharts，所以后续的内容都是先一块代码，再一块图或者文字描述。

## 一、熟悉数据

不急着上模型，调参数，先熟悉下数据:

**1.** 这是一个手机游戏的客户数据，注册个账号玩了一下。

**2.**有什么类型的feature：user\_id、注册时间、游戏资源（木头、石头、象牙、肉、魔法等）、兵种（勇士、驯兽师、萨满）、加速券（通用、建筑、科研、训练、治疗）、建筑、科研、PVP（玩家对战）、PVE（人机对战）、在线时长、付费金额、付费次数。这些都 是注册前7天的信息

**3.** label：前45天的付费金额.

再问几个问题：

**1.**user\_id在train和test中有没重复？

**2.**数据在时间上的分布？

**3.**有多少付费玩家，会给多少钱，分布如何？比例随时间的变化？

**4.**在线时长如何？

**5.**单次付费有什么额度的？

### 1、user\_id在train和test中有没重复？

*# 读取数据*

data = pd.read\_csv("tap\_fun\_train.csv", parse\_dates=True)

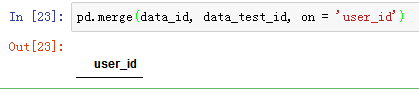
data\_test = pd.read\_csv("tap\_fun\_test.csv", parse\_dates=True)

*# 提取user\_id列，并做合并处理*

data\_id = pd.DataFrame(data['user\_id'],columns=['user\_id'])

data\_test\_id = pd.DataFrame(data\_test['user\_id'],columns=['user\_id'])

pd.merge(data\_id, data\_test\_id, on = 'user\_id')



没有重复的user\_id，就是检查下，不要多想，哈哈！

### 2、 玩家注册时间分布

*# 增加两列*

data['register\_time\_month'] = data.register\_time.str[:7]

data['register\_time\_day'] = data.register\_time.str[6:10]

*# 统计并保存为dataframe*

data\_month\_df = pd.DataFrame(data['register\_time\_month'].value\_counts()).sort\_index()

*# print(data\_month\_df)*

data\_day\_df = pd.DataFrame(data['register\_time\_day'].value\_counts()).sort\_index()

*# print(data\_day\_df)*

*# 画图 用了echarts的python版，挺好用的，也美观*

from pyecharts import Line, Grid

line1 = Line("玩家数量统计-月")

line1.add("玩家数量", data\_month\_df.index, data\_month\_df['register\_time\_month'], mark\_line=["average"], mark\_point=["max", "min"])

line2 = Line("玩家数量统计-日",title\_top="50%")

line2.add("玩家数量", data\_day\_df.index, data\_day\_df['register\_time\_day'], mark\_line=["average"], mark\_point=["max", "min"])

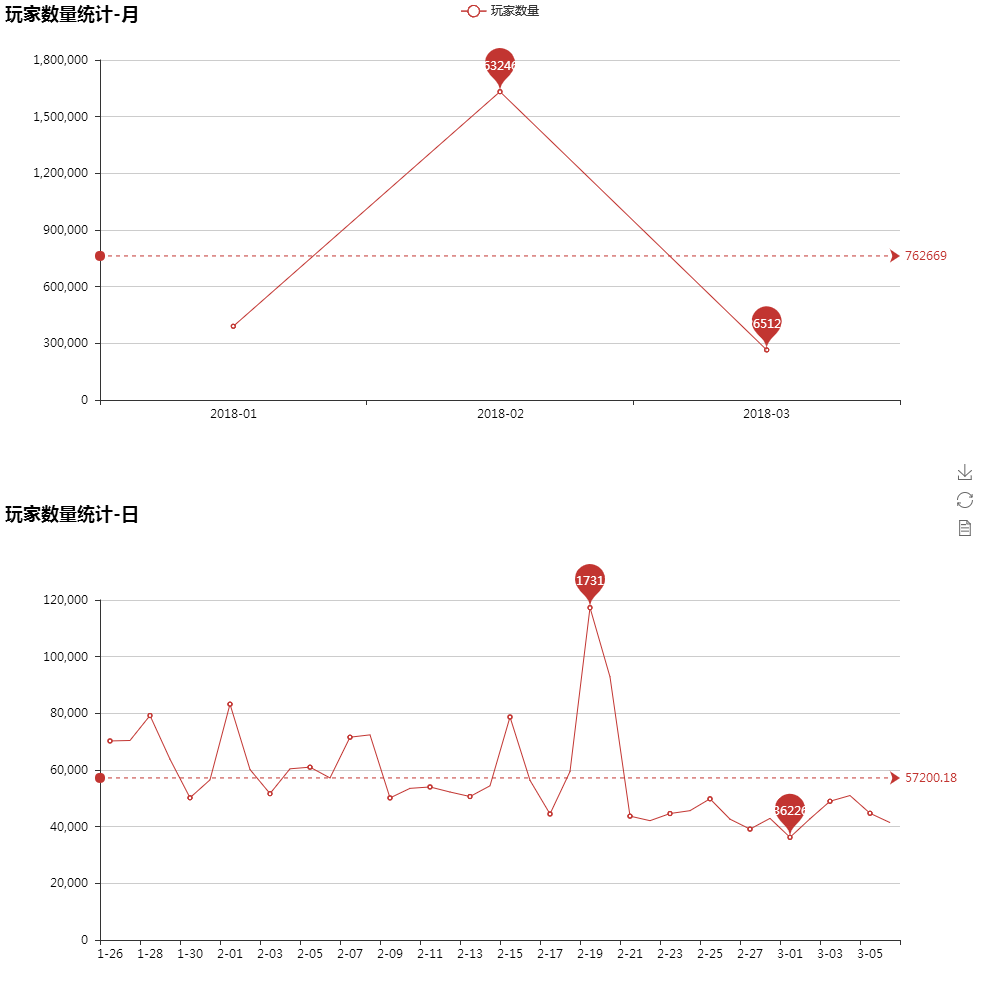
grid = Grid(width = 1000, height = 1000)

grid.add(line1, grid\_bottom="60%")

grid.add(line2, grid\_top="60%")

grid.render()

grid



月数据：不完整的，1月是从26号开始的数据，3月只有到6号的数据。

日数据：平均每天注册人数有5.7万人，挺多的啊。其中2月19号有11.7万人注册，是平时的两倍，20号还有9.3万人注册，有推广活动吗？但是整体趋势上，注册用户是下降的。

### 3、 有多少付费玩家，比例随时间的变化，会给多少钱，分布如何？

有多少付费玩家？

data\_pay\_7 = copy.copy(data[data['pay\_price']>0])

print(data\_pay\_7.shape) # (41439, 111)

print(data\_pay\_7.shape[0]/data.shape[0]) # 0.018111395638212645

七天内付费的玩家有41439个，占比大概是1.811%。

   》付费玩家比例随时间的变化

*# ----------------------------- 统计，改列明（避免冲突），合并，计算比例*

data\_pay\_7\_day\_df = pd.DataFrame(data\_pay\_7['register\_time\_day'].value\_counts()).sort\_index()

*# print(data\_pay\_7\_day\_df)*

data\_pay\_7\_day\_df.rename(columns={'register\_time\_day':'pay\_register\_time\_day'}, inplace = True)

data\_day\_count = pd.concat([data\_pay\_7\_day\_df, data\_day\_df], axis=1)

*# print(data\_day\_count)*

data\_day\_count['pay\_percent'] = data\_day\_count['pay\_register\_time\_day']/data\_day\_count['register\_time\_day']

*# print(data\_day\_count)*

*# ----------------------------- 画图*

from pyecharts import Overlap

line3 = Line()

line3.add("注册玩家数量", data\_day\_count.index, data\_day\_count['register\_time\_day'], mark\_line=["average"], mark\_point=["max", "min"])

line4 = Line()

line4.add("7天内付费玩家数量", data\_day\_count.index, data\_day\_count['pay\_register\_time\_day'], mark\_line=["average"],

mark\_point=["max", "min"], yaxis\_max=3000)

overlap = Overlap()

*# 默认不新增 x y 轴，并且 x y 轴的索引都为 0*

overlap.add(line3)

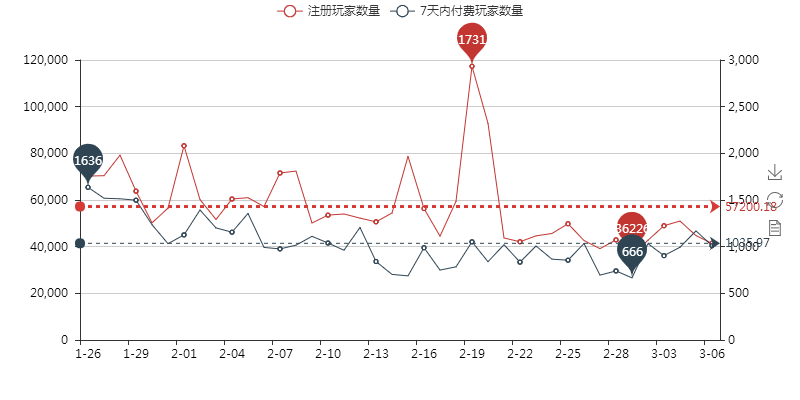
*# 新增一个 y 轴，此时 y 轴的数量为 2，第二个 y 轴的索引为 1（索引从 0 开始），所以设置 yaxis\_index = 1*

*# 由于使用的是同一个 x 轴，所以 x 轴部分不用做出改变*

overlap.add(line4, yaxis\_index=1, is\_add\_yaxis=True)

overlap.render()

overlap



from pyecharts import Bar, Overlap

line3 = Line()

line3.add("注册玩家数量", data\_day\_count.index, data\_day\_count['register\_time\_day'], mark\_line=["average"], mark\_point=["max", "min"])

bar = Bar()

bar.add("7天内付费玩家比例", data\_day\_count.index, data\_day\_count['pay\_percent'], yaxis\_max=0.1)

overlap = Overlap()

*# 默认不新增 x y 轴，并且 x y 轴的索引都为 0*

overlap.add(line3)

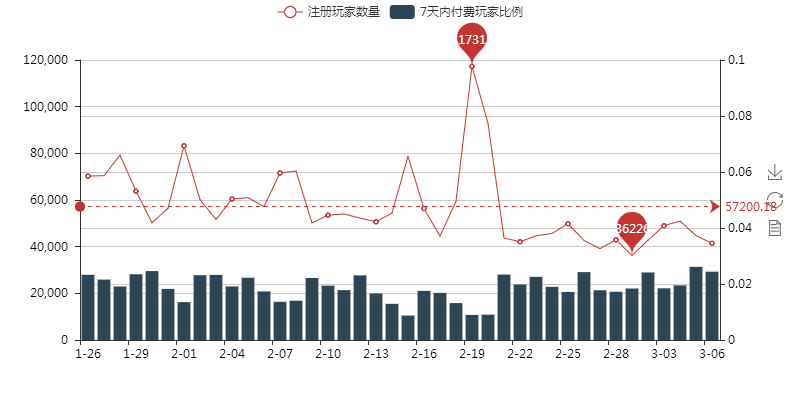
*# 新增一个 y 轴，此时 y 轴的数量为 2，第二个 y 轴的索引为 1（索引从 0 开始），所以设置 yaxis\_index = 1*

*# 由于使用的是同一个 x 轴，所以 x 轴部分不用做出改变*

overlap.add(bar, yaxis\_index=1, is\_add\_yaxis=True)

overlap.render()

overlap



注册用户的7天付费占比平均值是1.811%，分拆到每天的注册用户中，有起伏，但是波动不算大。

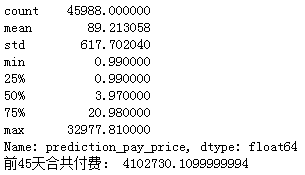
反而是某天（2月1、7、8、15、19、20号等）的注册用户越多，成功转化付费的好像不会“水涨船高”，也就是付费的还是那么多，促销回来的客户还是不付费？

   》会给多少钱？

data\_pay\_45 = copy.copy(data[data['prediction\_pay\_price']!=0])

print(data\_pay\_45['prediction\_pay\_price'].describe())

print('前45天合共付费：',data\_pay\_45['prediction\_pay\_price'].sum())



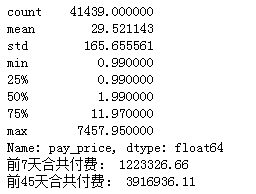
(上图)有4.6万客户前45天付款了，合计给了410万，最土豪的花了3.3万，土豪你不懂。

data\_pay\_7 = copy.copy(data[data['pay\_price']!=0])

print(data\_pay\_7['pay\_price'].describe())

print('前7天合共付费：',data\_pay\_7['pay\_price'].sum())

print('前45天合共付费：',data\_pay\_7['prediction\_pay\_price'].sum())



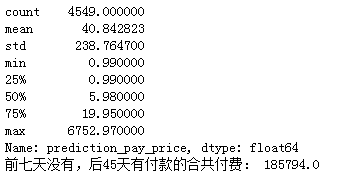
(上图)有4.14万客户前7天就有付款行为了，占了4.6万的90.1%（有9.9%前七天没给钱，后来给了）；

前七天合计花了122万，前45天花了391.7万，占4.6万客户前45天总付款410万的95.5%。

data\_nopay\_7\_pay\_45 = copy.copy(data\_pay\_45[data\_pay\_45['pay\_price']==0])

print(data\_nopay\_7\_pay\_45['prediction\_pay\_price'].describe())

print('前七天没有，后45天有付款的合共付费：',data\_nopay\_7\_pay\_45['prediction\_pay\_price'].sum())



(上图)有4549个客户前7天没付款，但是前45天付款的，占了9.9%，金额18.6万，占比4.53%。

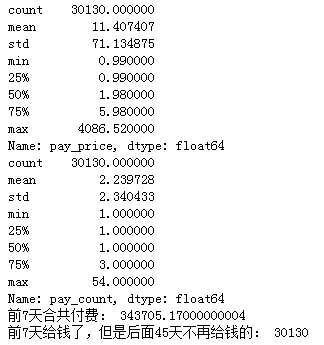
data\_pay\_7\_nopay\_45 = copy.copy(data\_pay\_7[data\_pay\_7['pay\_price']==data\_pay\_7['prediction\_pay\_price']])

print(data\_pay\_7\_nopay\_45['pay\_price'].describe())

print(data\_pay\_7\_nopay\_45['pay\_count'].describe())

print('前7天合共付费：',data\_pay\_7\_nopay\_45['pay\_price'].sum())

print('前7天给钱了，但是后面45天不再给钱的：',data\_pay\_7\_nopay\_45.shape[0])



(上图)前7天付款后，45天内不再付款的有3万，占4.6万的65.5%。前7天付费34.4万，占前七天的28.1%，占前45天410万的8.4%。---（放弃不玩了？）

data\_pay\_7\_pay\_45 = copy.copy(data\_pay\_7[data\_pay\_7['pay\_price']<data\_pay\_7['prediction\_pay\_price']])

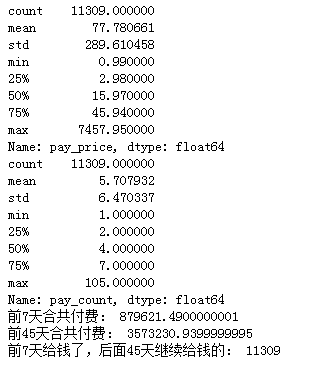
print(data\_pay\_7\_pay\_45['pay\_price'].describe())

print(data\_pay\_7\_pay\_45['pay\_count'].describe())

print('前7天合共付费：',data\_pay\_7\_pay\_45['pay\_price'].sum())

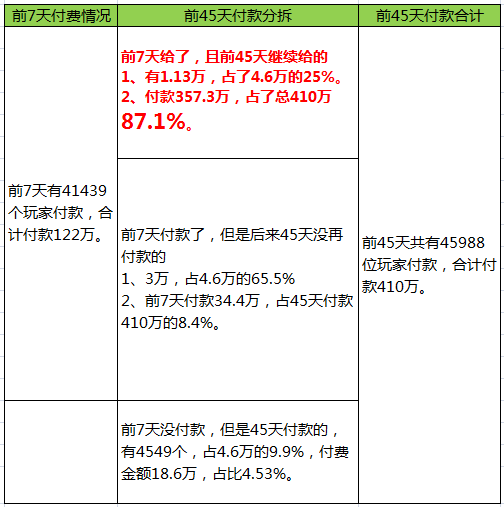
print('前45天合共付费：',data\_pay\_7\_pay\_45['prediction\_pay\_price'].sum())

print('前7天给钱了，后面45天继续给钱的：',data\_pay\_7\_pay\_45.shape[0])



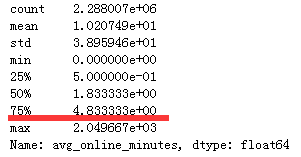
(上图)前7天付款，后面继续付款的，有1.13万，占4.6万的25%，付款占410万的87.1%，而且在7天内比较快的付款第二次了。

**小结：红色字体才是核心人群啊。**



### 4、在线时长如何？

data['avg\_online\_minutes'].describe()

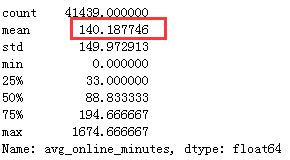


* (上图)整体来说，在线时长75%在5分钟内，看来都是下载看一下就不玩了。

data\_pay = copy.copy(data[data['pay\_price']!=0])

*# data\_pay.shape*

data\_pay['avg\_online\_minutes'].describe()



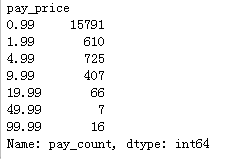
* (上图)付费用户平均在线140分钟，除以7天，每天在线20分钟。

### 5、单次付费有什么额度的？

data\_once = copy.copy(data[data['pay\_count']==1])

*# data\_once.shape*

data\_once.groupby("pay\_price")["pay\_count"].sum()



付费次数只有一次的客户，可以看出有几种单词付费的额度，都是0.99结尾。

### 6、研究评估标准

评分标准是RMSE，均方根误差

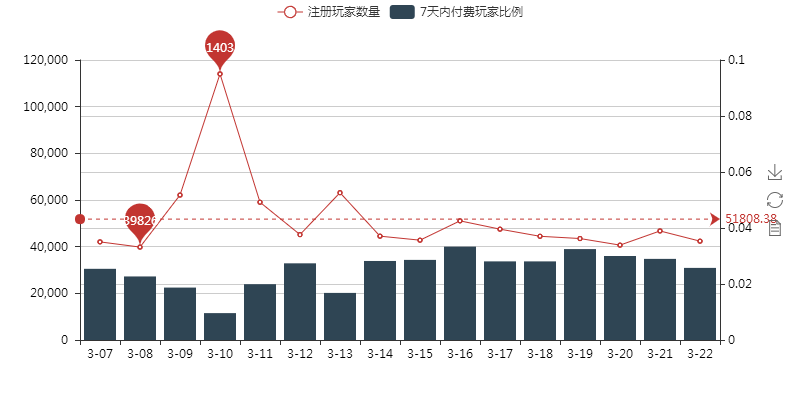
如果有一个土豪客户是付费15000，但是模型只猜测是付费1000，那么(15000-1000)^2/828934=236，开根号得到15.38，一个土豪客户的误判，就可以令评分标准飙升。

如果客户是给0.99的，但是预判了1.99，那么(1.99-0.99)^2=1，开根号也是1，如果每个客户都误判1元，RMSE也就+1。

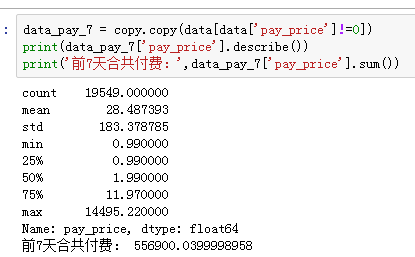
成败看土豪？？？

### 7、test数据集

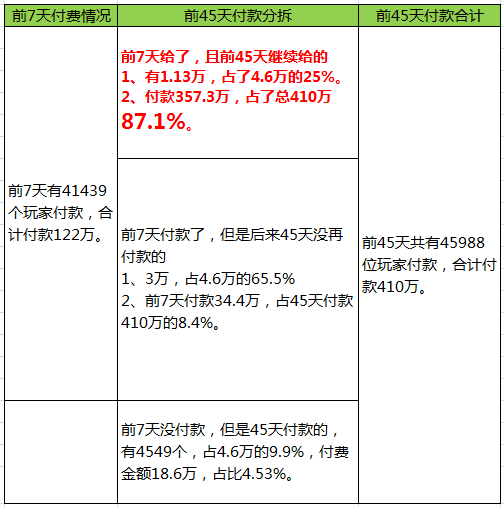
3月10号有类似2月19号的推广活动？



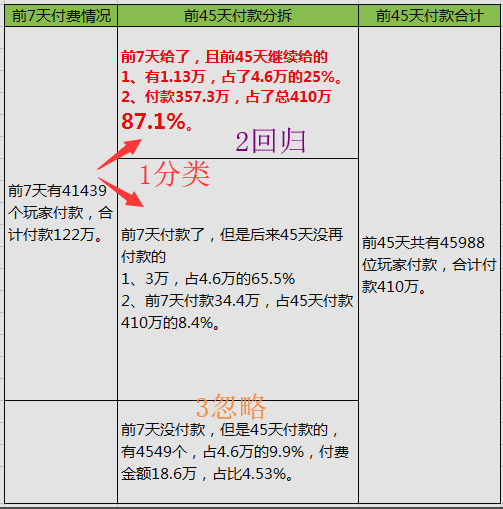
消费分布跟train值非常类似



## 二、解决策略



盯着上面的这个图，看着看着就想到： 使用前7天付费的数据，分两步，先分类出谁会继续付款，然后回归分析继续给钱的玩家会付款多少？还有第三部分，就是忽略掉前7天没付款，但是后面45天会付款的，然后就变成以下的图了。



### 1、分类

》准备数据

先对数据进行一下预处理，主要是对object类的feature处理掉，没有什么归一化之类的处理的。

data = pd.read\_csv("tap\_fun\_train.csv", parse\_dates=True)

*#提取object及其对应的数据*

object\_columns\_df = data.select\_dtypes(include=["object"])

*#object罗列出来了。*

print(object\_columns\_df.iloc[1])

*# 有一个object特征：register\_time，处理掉*

data['register\_time\_month'] = data.register\_time.str[5:7]

data['register\_time\_day'] = data.register\_time.str[8:10]

data = data.drop(['register\_time'],axis=1)

*# object转换float*

data[['register\_time\_month','register\_time\_day']] = data[['register\_time\_month','register\_time\_day']].apply(pd.to\_numeric)

*# data=pd.DataFrame(data,dtype=np.float)*

*# 对于注册时间，拆分开月和日之后，再合并一个数值，更好反馈时间的前后*

data['register\_time\_count'] = data['register\_time\_month'] \* 31 + data['register\_time\_day']

data.shape

*# (2288007, 111)*

*#保存前7天会给钱的客户*

data\_7\_pay = copy.copy(data[data['pay\_price']>0])

data\_7\_pay.shape

*# (41439, 111)*

data\_7\_pay.to\_csv ("tap\_fun\_train\_7\_pay.csv")

打标签label，删掉不该有的特征等。

*# -------------------------读取train set中前7天有付款的玩家明细*

data = pd.read\_csv("tap\_fun\_train\_7\_pay.csv", index\_col=0, parse\_dates=True)

print(data.shape) #(41439, 111)

*# -------------------------打标签*

data['7\_45\_same\_pay\_label'] = (data['pay\_price'] == data['prediction\_pay\_price'])

data['7\_45\_same\_pay\_label']=data['7\_45\_same\_pay\_label'].map({True:1,False:0})

data['7\_45\_same\_pay\_label'].value\_counts()

*# 1 30130 前7天给，后面都不付款了*

*# 0 11309 前7天给了，后续还给的。*

*# 删掉不需要的字段，比如45天的付费金额（对于test set是没有的字段），user\_id*

data = data.drop(['prediction\_pay\_price', 'user\_id'],axis=1)

data.shape

**》**训练分类模型

分拆数据，训练和验证部分。

from sklearn.model\_selection import train\_test\_split

label = '7\_45\_same\_pay\_label'

*# 将X和Y拆分开*

X = data.loc[:, data.columns != label]

y = data.loc[:, data.columns == label]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.22, random\_state = 0)

print("100% data")

print("Number transactions train dataset: ", len(X\_train))

print("Number transactions test dataset: ", len(X\_test))

print("Total number of transactions: ", len(X\_train)+len(X\_test))

*# print(y\_train.info())*

*# train和test拆分后，把train部分重新组合成data\_train，也就是把test部分完整保留下来，除了test用，不参加任何处理了。*

data\_train = pd.concat( [X\_train, y\_train], axis=1 )

print(data.shape)

print('---------------------------------------')

print(data\_train.shape)

# show\_class(data\_train,label) 自己写的函数，屏蔽了吧

#创建一个dataframe，然后对模型的效果进行记录。最后小结。

thresholds = [0.1,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.6,0.65,0.7,0.75,0.8,0.85,0.9]

thresholds\_2 = thresholds[:] #= thresholds,如果这样复杂是，浅复制，映射同一块内存

thresholds\_2.append('time')

print(thresholds\_2)

result\_model\_f1 = pd.DataFrame(index=thresholds\_2)

print(result\_model\_f1)

训练，用GradientBoostingClassifier训练试试先，然后在留下的22%数据中，进行验证，f1有87.55%，挺不错的了。

import time

start = time.time()

print('start time:',time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())))

from sklearn.ensemble import GradientBoostingClassifier

gradient\_boosting\_classifier = GradientBoostingClassifier()

gradient\_boosting\_classifier.fit(X\_train,y\_train.values.ravel())

y\_pred = gradient\_boosting\_classifier.predict(X\_test.values)

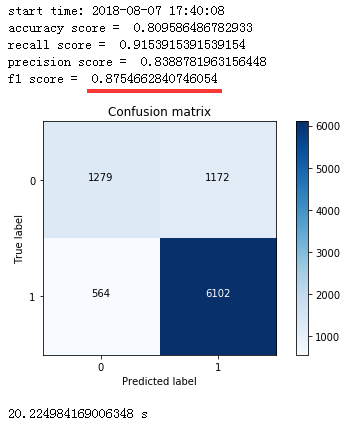
*# Plot non-normalized confusion matrix*

plt.figure()

plot\_confusion\_matrix(y\_test,y\_pred,title='Confusion matrix')

end = time.time()

print(end-start,'s')



试试用不同的threshold value来划分，看看有没更优的threshold value，0.45会好一点点，87.69%，比0.5，高0.14%，小幅度提升。

*#2、----------------------------------预测结果----------------------------------*

y\_pred\_proba = gradient\_boosting\_classifier.predict\_proba(X\_test.values) #array of shape = [n\_samples, n\_classes], or a list of n\_outputs

#3、----------------------------------记录各种threshold下的结果----------------------------------

result\_model\_f1['GradientBoostingClassifier'] = 0 #扩充列，全为0.

print(result\_model\_f1)

for i in thresholds:

y\_test\_predictions\_high\_recall = y\_pred\_proba[:,1] > i

print('Threshold >= %s'%i)

print\_recall\_precision\_f1(y\_test,y\_test\_predictions\_high\_recall)

print("------------------------------------")

for i in thresholds:

y\_test\_predictions\_high\_recall = y\_pred\_proba[:,1] > i

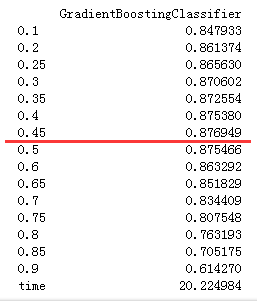
plt.figure(figsize=(4,4))

plot\_confusion\_matrix(y\_test,y\_test\_predictions\_high\_recall, title='Threshold >= %s'%i)

result\_model\_f1.loc[i,'GradientBoostingClassifier'] = f1\_score(y\_test.values,y\_test\_predictions\_high\_recall) #记录f1

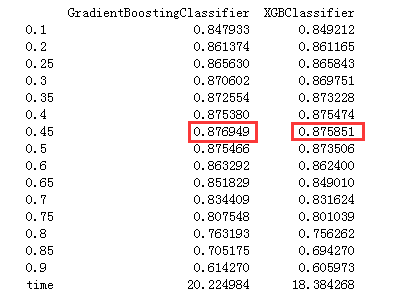
result\_model\_f1.loc['time','GradientBoostingClassifier'] = end-start #记录时间

print(result\_model\_f1)



备注下：

用xgboost也试过，默认参数下，差一点点。



导出模型

from sklearn.externals import joblib

print(gradient\_boosting\_classifier)

joblib.dump(gradient\_boosting\_classifier, 'gradient\_boosting\_classifier.model')

### 2、回归

》准备数据

data = pd.read\_csv("tap\_fun\_train\_7\_pay.csv", index\_col=0, parse\_dates=True)

data\_pay\_more = copy.copy(data[data['pay\_price']<data['prediction\_pay\_price']])

data\_pay\_more.shape

*# (11309, 111) 前7天付款，且后来继续付款的有1.1万人。用来做回归。*

*#删掉user\_id*

data\_pay\_more = data\_pay\_more.drop([ 'user\_id'],axis=1)

data = copy.copy(data\_pay\_more)

data.shape

》训练模型

分拆数据，训练与验证部分

from sklearn.model\_selection import train\_test\_split

Quantity = 'prediction\_pay\_price'

*# 将X和Y拆分开*

X = data.loc[:, data.columns =='pay\_price']

*# X = data.loc[:, data.columns != Quantity]*

y = data.loc[:, data.columns == Quantity]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.22, random\_state = 0)

print("100% data")

print("Number transactions train dataset: ", len(X\_train))

print("Number transactions test dataset: ", len(X\_test))

print("Total number of transactions: ", len(X\_train)+len(X\_test))

来，训练吧，还是GradientBoosting，但是是Regressor版，GradientBoostingRegressor。

import time

start = time.time()

print('start time:',time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time())))

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

gradient\_boosting\_regression = GradientBoostingRegressor()

gradient\_boosting\_regression.fit(X\_train,y\_train.values.ravel())

y\_pred = gradient\_boosting\_regression.predict(X\_test.values)

*# The mean squared error*

print("Root Mean squared error: %.2f"

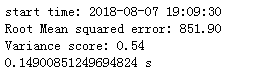
% mean\_squared\_error(y\_test, y\_pred) \*\* 0.5)

*# Explained variance score: 1 is perfect prediction*

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))

end = time.time()

print(end-start,'s')



其实Root Mean Squared error，都去到851了，爆表啊，先不管，上系统看看先

### 3、用模型预测，上系统看看排名

》把test set处理得跟train set训练前的样子，要不模型不认啊

data = pd.read\_csv("tap\_fun\_test.csv", parse\_dates=True)

data.shape #(828934, 108)

*#提取object及其对应的数据*

object\_columns\_df = data.select\_dtypes(include=["object"])

*#object罗列出来了。*

print(object\_columns\_df.iloc[1])

*# 有一个object的feature，register\_time ，处理掉*

data['register\_time\_month'] = data.register\_time.str[5:7]

data['register\_time\_day'] = data.register\_time.str[8:10]

data = data.drop(['register\_time'],axis=1)

*# object转换float*

data[['register\_time\_month','register\_time\_day']] = data[['register\_time\_month','register\_time\_day']].apply(pd.to\_numeric)

*# 对于注册时间，拆分开月和日之后，再合并一个数值，更好反馈时间的前后*

data['register\_time\_count'] = data['register\_time\_month'] \* 31 + data['register\_time\_day']

data.shape

*#(828934, 110)*

*# 保存前7天会给钱的客户*

data\_7\_pay = copy.copy(data[data['pay\_price']>0])

data\_7\_pay.shape # (19549, 110)

data\_7\_pay.to\_csv ("tap\_fun\_test\_7\_pay.csv")

*# 保存前7天没有给钱的客户*

data\_7\_NOT\_pay = copy.copy(data[data['pay\_price']==0])

data\_7\_NOT\_pay.shape #(809385, 110)

data\_7\_NOT\_pay.to\_csv ("tap\_fun\_test\_7\_NOT\_pay.csv")

》把test set里面前7天没付款的直接预测后续不会给钱。

data\_test = pd.read\_csv("tap\_fun\_test\_7\_NOT\_pay.csv", index\_col=0, parse\_dates=True)

print(data\_test.shape)# (809385, 110) ， 80万的客户

*# 把user\_id和前7天的0元pay\_price提炼出来。*

data\_test\_part1 = data\_test[['user\_id','pay\_price']]

data\_test\_part1.rename(columns={'pay\_price':'prediction\_pay\_price'}, inplace = True)

data\_test\_part1.to\_csv('tap\_fun\_test\_part1\_still0.csv')

》在前7天有付费的客户中分类出后续不再付费的客户，并保存。

data\_test = pd.read\_csv("tap\_fun\_test\_7\_pay.csv", index\_col=0, parse\_dates=True)

print(data\_test.shape) # (19549, 110)

data\_test\_model = data\_test.drop([ 'user\_id'],axis=1)，# 跑模型前要删掉user\_id

*#跑分类模型*

y\_test\_pred = gradient\_boosting\_classifier.predict(data\_test\_model.values)

*# 结果转ndarray换成dataframe*

y\_test\_pred = pd.DataFrame(y\_test\_pred, columns= {'pred\_label'})

*# 为了把index清零，重头开始，先转成ndarray，再转回来dataframe*

columns\_test = data\_test.columns

data\_test = data\_test.values

data\_test = pd.DataFrame(data\_test, columns = columns\_test )

*# 重新把预测结果放回去原始数据*

y\_test\_pred = pd.concat([data\_test, y\_test\_pred], axis=1)

y\_test\_pred.shape # (19549, 111)

y\_test\_pred['pred\_label'].value\_counts()

*# 1 15587 不会再给钱的有这么多*

*# 0 3962 会继续给钱的。*

*# part2：把1的留出来，并预测为原来的值。*

*# part3：把0的放到回归里面去猜。*

y\_test\_pred\_part2 = copy.copy(y\_test\_pred[y\_test\_pred['pred\_label']==1])

y\_test\_pred\_part3 = copy.copy(y\_test\_pred[y\_test\_pred['pred\_label']==0])

y\_test\_pred\_part2\_user\_id = pd.DataFrame(y\_test\_pred\_part2,columns ={'user\_id'})

y\_test\_pred\_part2\_pay = pd.DataFrame(y\_test\_pred\_part2,columns ={'pay\_price'})

y\_test\_pred\_part2 = pd.concat([y\_test\_pred\_part2\_user\_id, y\_test\_pred\_part2\_pay], axis=1)

y\_test\_pred\_part2.rename(columns={'pay\_price':'prediction\_pay\_price'}, inplace = True)

y\_test\_pred\_part2.to\_csv('tap\_fun\_test\_part2\_nopaymore.csv')

y\_test\_pred\_part3.to\_csv('tap\_fun\_test\_part3\_paymore.csv')

》前7天付费的，【分类】预判会继续付费的，【回归】出付费金额。

y\_test\_pred\_part3 = pd.read\_csv("tap\_fun\_test\_part3\_paymore.csv", index\_col=0, parse\_dates=True)

y\_test\_pred\_part3.shape #(3962, 111)

*# 把user\_id保留出来。*

user\_id\_pay\_more = y\_test\_pred\_part3['user\_id'].values

user\_id\_pay\_more[:10]

*# 提出需要的字段，也就是pay\_price*

y\_test\_pred\_part3\_test = pd.DataFrame(y\_test\_pred\_part3,columns=['pay\_price'])

y\_test\_pred\_part3\_test.shape

*# 跑模型*

y\_test\_pred\_part3\_howmuch = gradient\_boosting\_regression.predict(y\_test\_pred\_part3\_test.values)

*# 把预测结果和user\_id合并回去。*

y\_test\_pred\_part3\_user\_id = pd.DataFrame(user\_id\_pay\_more,columns = {'user\_id'})

y\_test\_pred\_part3\_howmuch = pd.DataFrame(y\_test\_pred\_part3\_howmuch,columns = {'prediction\_pay\_price'})

y\_test\_pred\_part3 = pd.concat([y\_test\_pred\_part3\_user\_id, y\_test\_pred\_part3\_howmuch], axis=1)

y\_test\_pred\_part3.shape # (3962, 2)

y\_test\_pred\_part3.to\_csv('tap\_fun\_test\_part3\_paymore\_result.csv')

》重新合并三块的结果

pred\_part1 = pd.read\_csv("tap\_fun\_test\_part1\_still0.csv", index\_col=0, parse\_dates=True)

print(pred\_part1.shape)

pred\_part2 = pd.read\_csv("tap\_fun\_test\_part2\_nopaymore.csv", index\_col=0, parse\_dates=True)

print(pred\_part2.shape)

pred\_part3 = pd.read\_csv("tap\_fun\_test\_part3\_paymore\_result.csv", index\_col=0, parse\_dates=True)

print(pred\_part3.shape)

*# (809385, 2)*

*# (15587, 2)*

*# (3962, 2)*

pred = pred\_part1.append(pred\_part2)

pred = pred.append(pred\_part3)

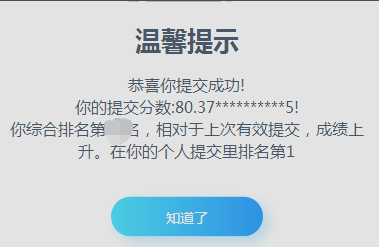
pred.shape # (828934, 2)

pred.to\_csv('result.csv')

幸好最后的数量跟test set的客户规模对得上。

》上系统看分数

有个小插曲，打开文件，把第一列index删掉，然后保存为逗号作为分割的csv文件。



改用RandomForestClassifier+GradientBoostingregression，基本没变化。



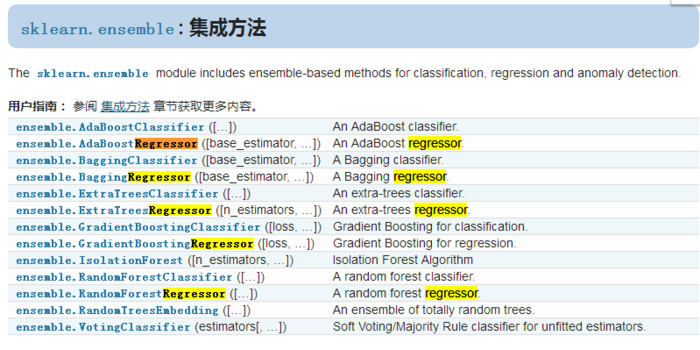
改用RandomForestClassifier和LinearRegression，效果突然降低了10。



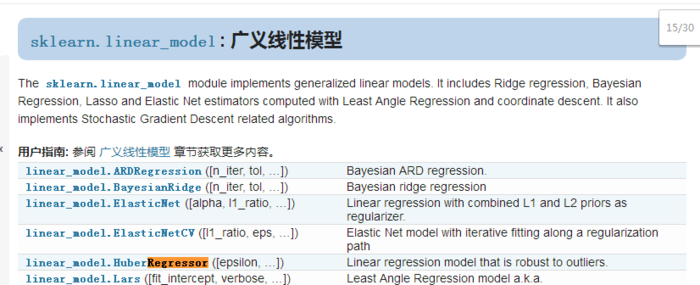
### 4、结果分析

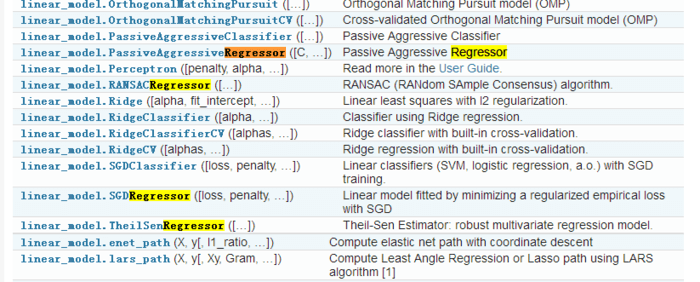
从上面三次结果看，回归部分才是提升的关键所在。

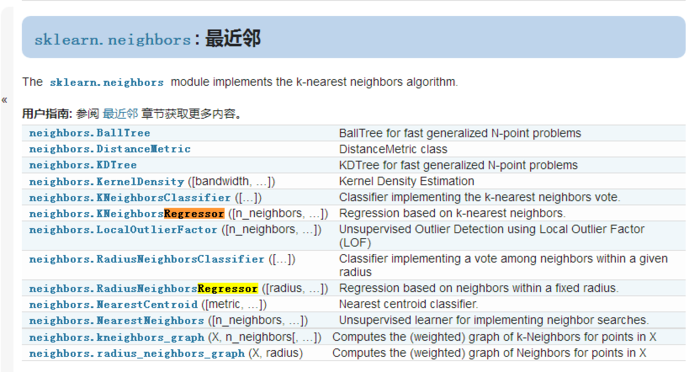
sklearn里面有哪些regressor，[点这里](http://sklearn.apachecn.org/cn/stable/modules/classes.html" \l "module-sklearn.tree" \t "_blank)

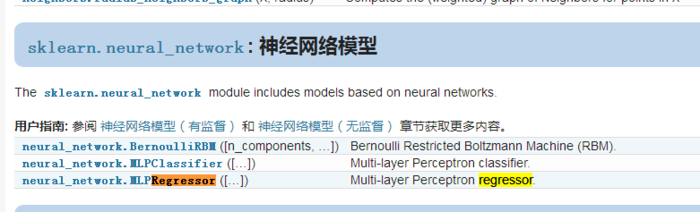


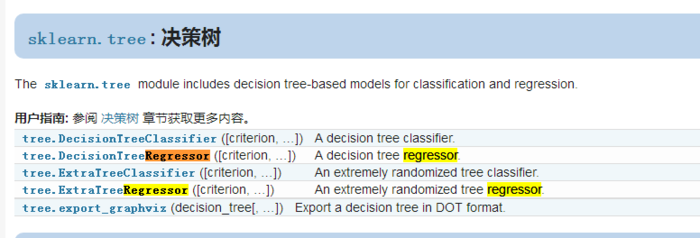












改用RandomForestClassifier和huber\_regressor



迟点再试试其他的regressor吧

### 5、优化方向

1、后续试试对三个部分的RMES看看。

2、归回部分、被忽略的后来付费用户部分

用xgb简单预测pay\_price不等于0的那些项，pay\_price等于0的项直接预测为0，调参后我记得好像可以到65+

2018年8月9日