**“魔镜杯”风控算法大赛**

**拍拍贷风控模型**

Author：Zhiwen Gu

**比赛数据：**

https://link.zhihu.com/?target=https%3A//www.kesci.com/home/competition/56cd5f02b89b5bd026cb39c9/content/1

**项目概述：**

1.比赛数据：

"拍拍贷"提供的数据包括信用违约标签（因变量）、建模所需的基础与加工字段（自变量）、相关用户的网络行为原始数据，数据字段已经做脱敏处理。本次实战采用的是初赛数据，包括3万条训练集和2万条测试集。数据文档包括：

* Master：每一行代表一个样本（一笔成功成交借款），每个样本包含200多个各类字段。
* Log\_Info：借款人的登陆信息，每个样本含多条数据。
* Userupdate\_Info：借款人修改信息，每个样本多条数据。

2.比赛规则：

基于训练集数据构建预测模型，使用模型计算测试集的评分（评分数值越高，表示越有可能出现贷款违约），评价标准为AUC。

**建模思路：**

1. 数据清洗工作，主要做一下缺失值处理，常变量处理，空格符处理，字符大小写转换等

2. 特征处理工作，分为特征转换和特征衍生。主要做了以下工作：

* Master数据：地理位置信息的处理（省份，城市），运营商和微博特征的转换，以及对排序特征，periods特征的交叉组合等。
* Log\_Info数据：衍生出"累计登陆次数"，"登录时间的平均间隔"，"最近一次的登录时间距离成交时间差"等特征。
* Userupdate\_Info数据：衍生出"最近的修改时间距离成交时间差"，"修改信息的总次数", "每种信息修改的次数"等特征。

3. 特征筛选工作：利用lightgbm输出特征重要性进行筛选。

4. 建模工作：

* 单模型，选用的机器学习模型是lightgbm。
* bagging模型：基于bagging的思想，通过对模型参数进行随机扰动，构建多个子模型，选用的基模型是lightgbm。

**重要工作：**

1. 数据离群值判定，清洗

2. 利用XGboost，按特征重要性系统衍生新特征，剥离没用的特征

3. 特征的离散化，二值化，数值型特征衍生排序

4. 交叉验证，网格搜索完成lightGBM超参数设置

5. bagging lightGBM模型

**项目流程**

**一.数据清洗（data\_input, data\_EDA\_clean代码模块）**

1. 导入合并好的数据，看一下好坏用户比：

Mater数据包含约50000个样本，200多个字段。

# 样本的好坏比

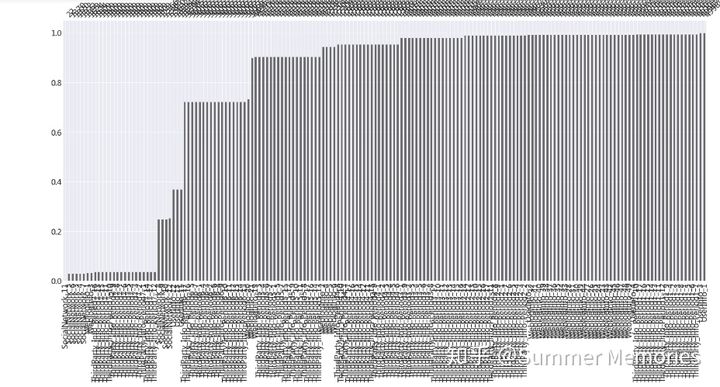
data1.target.value\_counts()



好坏比约11：1，属于不平衡数据集，因为最后模型用的是lightgbm，故本次实战先不采用过采样方法来进行数据抽样，直接用原始数据来建模。

2. 缺失值处理：

缺失变量的数据可视化

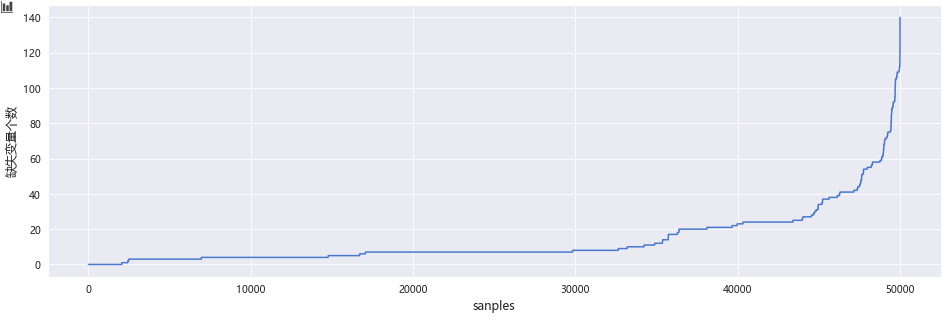


有一些变量缺失率很高，如果将缺失填充为0，这类变量其实可看做一类稀疏特征，由于xgboost，lightgbm等GBDT类树模型对高维稀疏特征处理不太好，并考虑到这些变量的业务含义未知，携带的信息量太少，故对缺失率高的变量作删除处理。

除了要考虑变量的缺失情况，也要考虑样本缺失特征个数的情况，一个样本如果缺失的特征很多，说明用户的信息完善程度很低，通过对缺失样本的可视化，发现了这些离群点，可以考虑删除。

# 样本的趋势个数可视化

sc.plot\_missing\_user(df=data1,plt\_size=(16,5))



# 删除变量缺失个数在100个以上的用户

data1 = sc.missing\_delete\_user(df=data1,threshold=100)

缺失变量个数在100以上的用户数有298个。

4. 常变量处理

当一个变量中单个值所占比例过高（90%以上），说明该变量方差比较小，携带的信息较少，导致区分能力较差，对于这些变量也做删除处理。

5. 字符格式转换

城市特征清洗：

# 计算4个城市特征的非重复项计数，观察是否有数据异常

for col in ['UserInfo\_2','UserInfo\_4','UserInfo\_8','UserInfo\_20']:

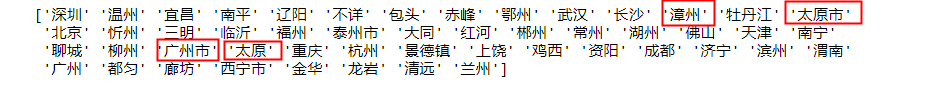
print('{}:{}'.format(col,data1[col].nunique()))

print('\t')



UserInfo\_8相对其他特征nunique较大，发现有些城市有"市"，有些没有，需要做一下格式转换，去掉字符串后缀"市".

print(data1.UserInfo\_8.unique()[:50])



# UserInfo\_8清洗处理，处理后非重复项计数减小到400

data1['UserInfo\_8']=[s[:-1] if s.find('市')>0 else s[:] for s in data1.UserInfo\_8]

UserupdateInfo1特征大小写转换：

'UserupdateInfo1'包含了大小写，如"qQ"和"QQ",属于同一种意思，所以需要对其英文字符统一转换大小写。

# 将UserupdateInfo1里的字符改为小写形式

df2['UserupdateInfo1'] = df2.UserupdateInfo1.map(lambda x:x.lower())

**二.特征处理（feature\_processing代码模块）**

**1.Master数据:**

对Master中的类别型特征（省份，城市，运营商，微博）做特征转换，对数值型变量做排序特征，periods特征的衍生。

类别型特征：

1）省份：

原数据有两个省份字段，推测一个为用户的户籍地址，另一个为用户居住地址所在省份，由此可衍生的字段为：

* 省份二值化，通过违约率将单个省份衍生为二值化特征，分为户籍省份和居住地省份
* 户籍省份和居住地省份是否一致，推测不一致的用户大部分为外来打工群体，相对违约率会高一点。
* 计算违约率时要考虑该省份的借款人数，如果人数太少，参考价值不大

省份二值化



两种省份特征选择出违约率排名前五的省份，并做二值化衍生。

户籍省份和居住地省份不一致衍生

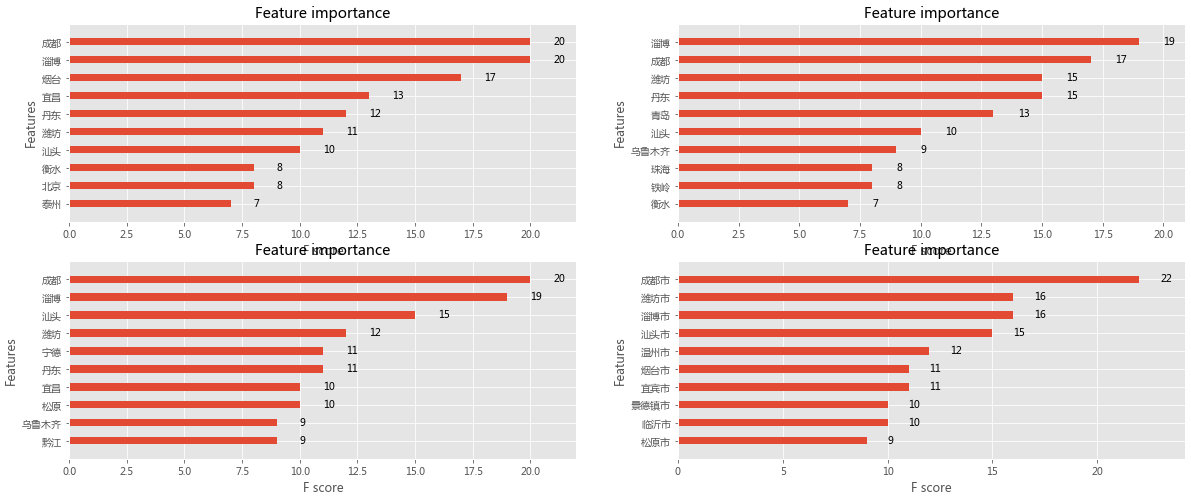
2）城市变量：

原数据中有4个城市信息，推测为用户常登录的IP地址城市，衍生的逻辑为：

* 通过xgboost挑选比较重要的城市变量，进行二值化衍生
* 由4个城市特征的非重复项计数可衍生成 登录IP地址的变更次数

城市二值化衍生

# 根据xgboost变量重要性的输出对城市作二值化衍生



将特征重要性排名前三的城市做二值化衍生：

IP地址变更次数衍生

3）运营商：

因运营商的种类少，直接做亚编码处理即可。

4）微博：

先对微博特征做一下缺失值填充，再做亚编码处理。

数值型特征：

1）排序特征衍生：

对数值型特征按数值从小到大进行排序，衍生成排序特征，排序特征异常值有更强的鲁棒性，可以增强模型的稳定性，降低过拟合风险。

2）periods特征衍生：

# 生成只包含periods的临时表

periods\_col = [i for i in num\_col2 if i.find('Period')>0]

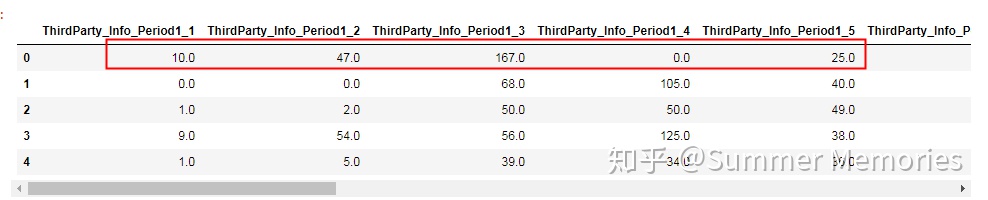
periods\_col2 = periods\_col+['target']

periods\_data = data1.loc[:,periods\_col2]

观察包含period1所有字段的数据，发现字段之间量级差异比较大，可能代表不同的含义，不适合做衍生。

periods1\_col = [col for col in periods\_col if col.find('Period1')>0]

periods\_data.loc[:,periods1\_col].head()



观察后缀都为1的字段，发现字段数据的量级基本一致，可以对其做min,max,avg等统计值的衍生。

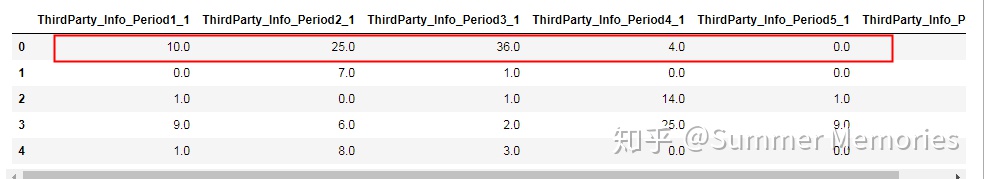
period\_1\_col=[]

for i in range(0,102,17):

col = periods\_col[i]

period\_1\_col.append(col)

periods\_data.loc[:,period\_1\_col].head()



对后缀都为1的periods字段做相应的四则运算（最小，最大，平均），衍生成新的特征。

2. Log\_Info表：

衍生的变量

* 累计登录次数
* 登录时间的平均间隔
* 最近一次的登录时间距离成交时间差

累计登录次数



登录时间的平均间隔



最近一次的登录时间距离成交时间差



将三个衍生好的特征组合成一张表



**3.Userupdate\_Info表**

衍生的变量

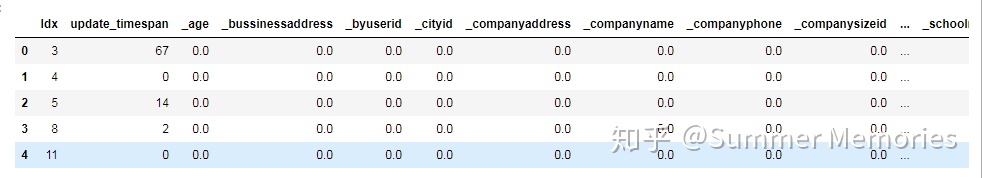
* 最近的修改时间距离成交时间差
* 修改信息的总次数
* 每种信息修改的次数

将衍生特征汇总成一张表：

update\_info = pd.merge(time\_span,cate\_change\_df,on='Idx',how='left')

update\_info = pd.merge(update\_info,update\_cnt,on='Idx',how='left')

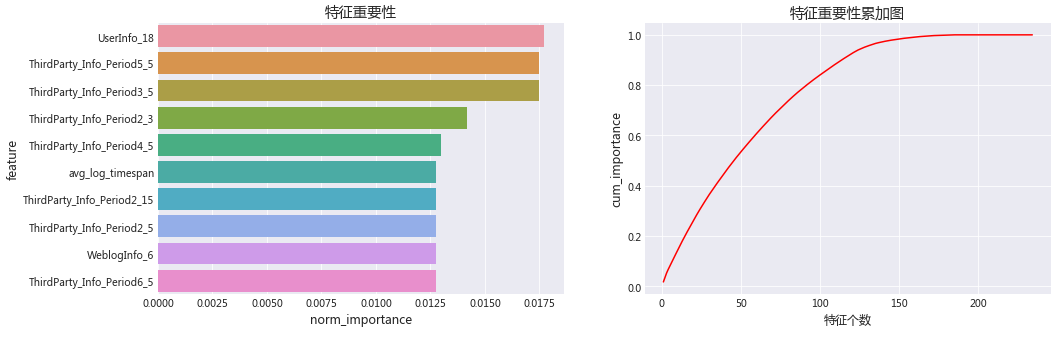
update\_info.head()



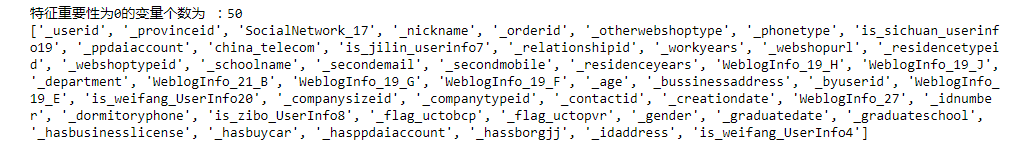
**三.特征筛选（data\_select代码模块)**

合并衍生后的特征，这里将数据集分为data1和data2，data1不包含排序特征和periods衍生特征，而data2包含所有的特征。之所以这么分类是因为实际工作中像排序特征和periods衍生特征在业务上很难解释，且线上部署比较麻烦，基于实际考虑，所以后续主要用data1来做特征筛选和建模，后续有时间也会对data2做一下，看一下是否对模型有没有提升效果。

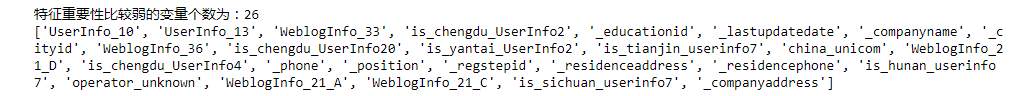
用lightgbm来筛选特征，这里训练了10个lightgbm，并对并对10个模型输出的feature\_importances\_取平均，最后对特征重要性的值进行归一化处理，并做特征重要性的可视化。



剔除特征重要性为0的变量



剔除特征重要性比较弱的变量



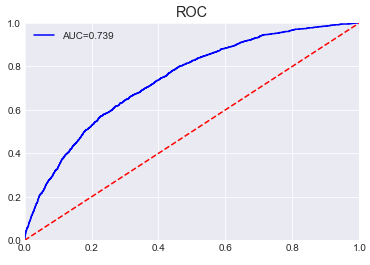
**最后筛选出160个特征。**

**四.建模**

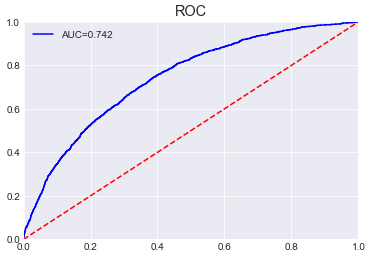
**1. 单模型lightgbm（single\_lightgbm\_model代码模块）**

1）在不调参的情况下，两种版本的lightgbm的默认参数模型

sklearn接口版本



原生版本的lightgbm



调参

1）学习速率先设定为0.1，便于缩短模型的训练时间，最后一步再减小学习速率，先确定最大的迭代次数。

# 确定最大迭代次数，学习率设为0.1

base\_parmas={'boosting\_type':'gbdt',

'learning\_rate':0.1,

'num\_leaves':40,

'max\_depth':-1,

'bagging\_fraction':0.8,

'feature\_fraction':0.8,

'lambda\_l1':0,

'lambda\_l2':0,

'min\_data\_in\_leaf':20,

'min\_sum\_hessian\_inleaf':0.001,

'metric':'auc'}

cv\_result = lgb.cv(train\_set=lgb\_train,

num\_boost\_round=200,

early\_stopping\_rounds=5,

nfold=5,

stratified=True,

shuffle=True,

params=base\_parmas,

metrics='auc',

seed=0)

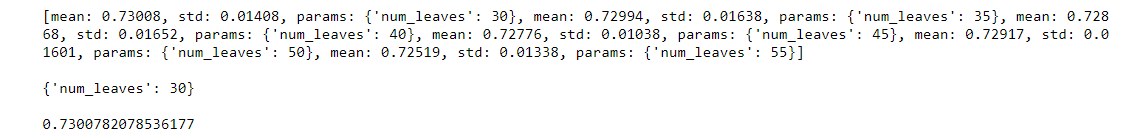
print('最大的迭代次数: {}'.format(len(cv\_result['auc-mean'])))

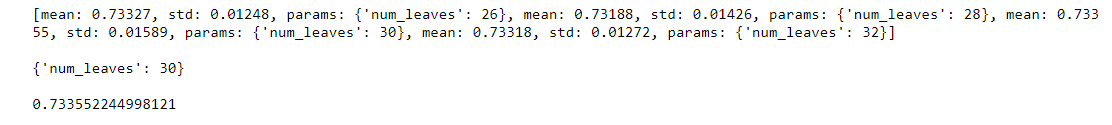
print('交叉验证的AUC: {}'.format(max(cv\_result['auc-mean'])))



**最大迭代次数为51次。**

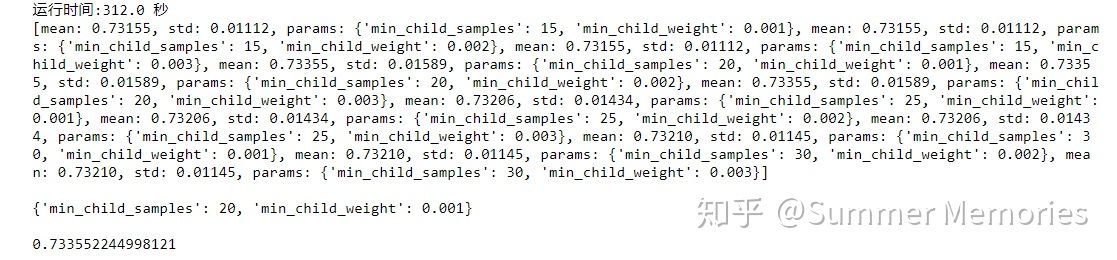
2）调num\_leaves，先粗调（设定步长为5），再细调（设定步长为





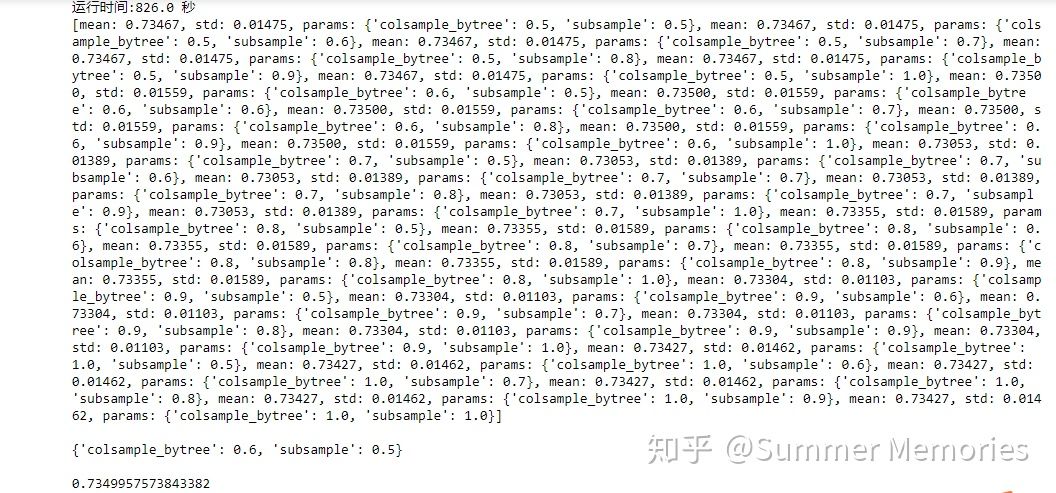
**确定num\_leaves值为30。**

3）min\_child\_samples 和 min\_child\_weight的调参



**确定min\_child\_samples 为20， min\_child\_weight为0.001**

4）subsample和colsample\_bytree进行调参



**确定subsample为0.5，colsample\_bytree为0.6**

5）正则化参数调参



**确定reg\_lambda为0.03，reg\_alpha为0.3**

6）减小学习速率，再计算最佳参数下的迭代次数



**确定最终的迭代次数为889次**

7）最佳参数建模（sklearn接口版本）

lgb\_single\_model = lgb.LGBMClassifier(n\_estimators=900,

learning\_rate=0.005,

min\_child\_weight=0.001,

min\_child\_samples = 20,

subsample=0.5,

colsample\_bytree=0.6,

num\_leaves=30,

max\_depth=-1,

reg\_lambda=0.03,

reg\_alpha=0.3,

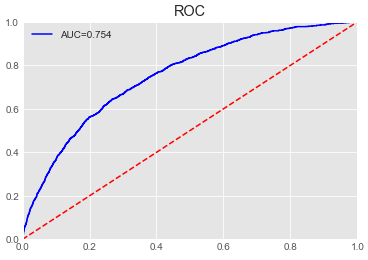
random\_state=0)

lgb\_single\_model.fit(x\_train,y\_train)

pre = lgb\_single\_model.predict\_proba(x\_test)[:,1]

print('lightgbm单模型的AUC：{}'.format(metrics.roc\_auc\_score(y\_test,pre)))

sc.plot\_roc(y\_test,pre)



**调参之后测试集的AUC提升了0.015。**

**2. lightgbm bagging**

主要借鉴了随机森林对于选取样本的随机性和对于特征的随机性的思想。

* 样本随机性的实现方法：random\_seed,bagging\_fraction,feature\_fraction的参数扰动
* 特征随机性的实现方法：在使用所有原生特征基础上，随机抽取一定的排序特征和periods特征至模型中。

因电脑内存有限，故只建立30个子模型，代码跑了4个多小时。。。

对30个子模型的预测结果(概率)进行average，得到bagging模型的预测结果

import time

a= time.time()

result\_df\_list=[]

for i in range(30):

result\_df = bagging\_lightgbm(feature\_fraction=feature\_fraction[i],

n\_feature=n\_feature[i],

ramdom\_seed=random\_seed[i],

bagging\_fraction=bagging\_fraction[i])

result\_df\_list.append(result\_df)

prep\_list = [list(x['prep']) for x in result\_df\_list]

bagging\_prep= list(np.sum(score\_list,axis=0)/30)

b = time.time()

print('运行时间:{}'.format(round(b-a,0)))

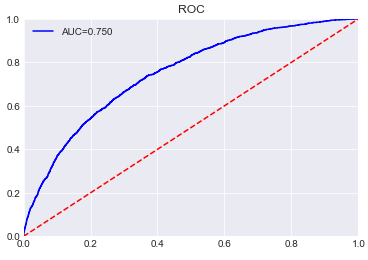


bagging模型的AUC：

import score\_card as sc

test\_y = list(df[df.sample\_status=='test']['target'])

sc.plot\_roc(y\_label=test\_y,y\_pred=bagging\_prep)



可以看到，最终bagging的结果相对于单lightGBM并没有提升，个人理解，目前的XGboost和lightGBM在模型复杂度和过拟合控制上已经有相当的稳定性，当然，只是个人理解，目前没有遇到bagging XGboost能提升明显的情况。。。。