AI 引论报告——信用卡评分模型

1. 问题定义

银行在决定是否向客户发放贷款时，会首先对贷款者的信用进行评分。一类简单的信用评分算法是根据贷款者的各项特征行计算，得到贷款者的违约概率，最终银行倾向于贷款给那些违约概率较低的贷款者。我们的任务是，使用某种自己感兴趣的方法（我们最终使用了基于 WOE 的逻辑回归），分析数据不同特征对于用户信用评分的重要程度，以及不同特征之间的相关关系。完成信用评分（300 – 900分）算法的建模以及算法实现（方法不限，例如：传统机器学习方法）。

本小组利用Kaggle上的**15**万条数据，对数据加以清洗，利用随机森林法填补缺失值，采用模型变量WOE编码方式离散化，并运用logistic回归模型，建立并训练二分类变量的广义线性模型，最终建立具有良好评估效果的信用评分卡。

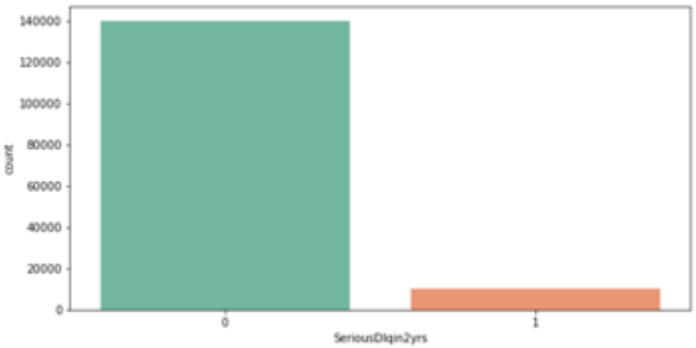
2. 数据集分析

2.1 综述

首先，我们来阐述一下本小节的逻辑。我们获得的数据是一份大型的数据，由10个特征（自变量）和一个结果（因变量）组成。在这10个特征中，每个特征有一定的范围限制，如年龄必须大于0，可用额度比值介于0和1之间等等。然而数据集中确实存在不合理的取值，甚至存在部分数据缺失的现象，这值得引起我们的额外注意。因此，我们需要通过python的函数来进行数据集分析的工作。我们拥有几个有力的武器：核密度曲线、箱型图、柱状图等。核密度曲线能够展示数据的聚集情况，而箱型图能够很好地展示出数据中的上界、下界有无异常值存在。通过这些直观的展示，我们可以较为轻松的分析数据的异常或缺失现象，进而为后面的数据预处理部分做好准备。我们的判断异常值基于的总原则是：情况一：若存在明显不符合逻辑的数据，则必然为异常值；情况二：若存在明显与其他数据相距甚远的数据，则极有可能为异常值。

2.2 查看好坏客户分布

绘制柱状图：

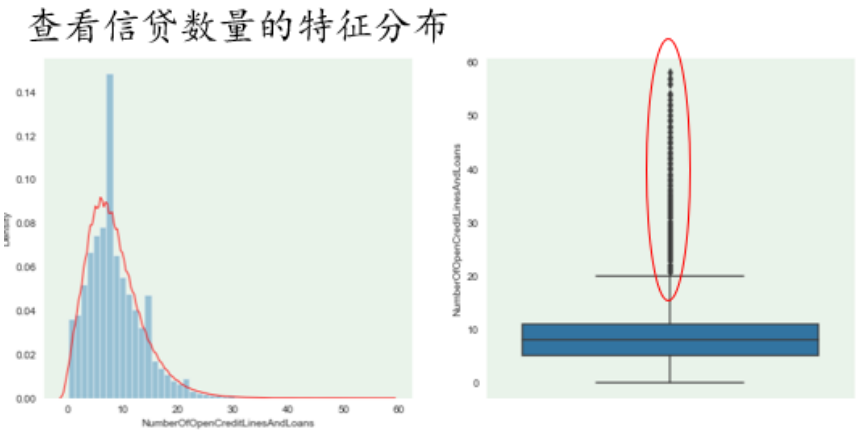


可以看出，好、坏客户的比例约为93：7，坏客户较少，符合预期。

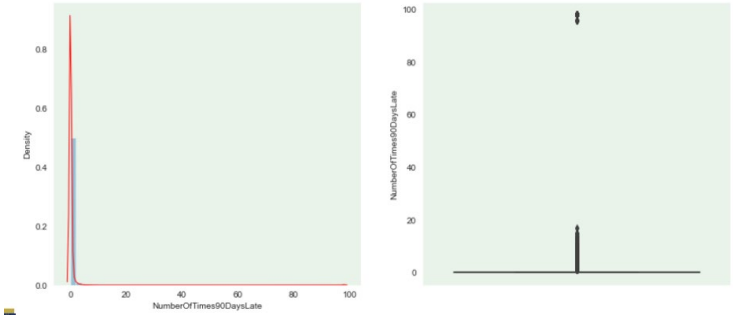
2.3 查看各维度数据的特征分布

读入数据后，我们需要通过绘制相关的核密度曲线与箱型曲线便于发现数据分布特点与异常，以供后续处理。

下面，我们以信贷数量和逾期90天笔数为例加以分析：



以信贷数量为例，上界值连续，没有出现单个离群的数据，不符合情况一、二，不是异常值，不用删除。



逾期90天笔数得人数分布，在上方有极少的两个点，说明有两个明显偏离的数据，符合情况二，极有可能为异常值，后续应删除。

2.4 本部分的结论

通过按照上述逻辑对10个特征的依次分析，我们得出了结论：可用额度比值大于1的为异常值；年龄小于18的（实际上数据集中仅有一个，年龄为0岁）为异常值；逾期30－60/60－90/90天笔数大于80的为异常值；家属数量大于20的为异常值；信贷数量大于50的为异常值。

此外，我们还发现：家属数量和月收入存在缺失值，后续需要处理。家属数量的缺失比率约为2.62%，月收入的缺失比率约为19.82%。

3. 数据预处理

根据上一部分的分析，我们需要对数据进行一定的处理。这里分为三个部分进行数据预处理。

首先我们需要处理缺失值。由于二中已经分析，家属数量的缺失量比较少，因此选择直接删除：

```
#3.2.1 对家属数量缺失的数据进行删除
trainingData=trainingData[trainingData['NumberOfDependents'].notnull()]
testData=testData[testData['NumberOfDependents'].notnull()]
```

而月收入的数据缺失比率接近20%，一般来说对于这种数据，我们有随机森林填补和直接删除两种做法。随机森林，即random forest，是一种新兴的机器学习算法，而且近些年他在随机森林领域的运用十分广泛，当然也是因为他本身非常灵活实用。它是一种以决策树作为基本单元，将多棵树集成的算法，具有当下算法中非常高的准确度，并且对于大数据的适用性很好。实际上在有的情况下直接删除得到结果反而比随机森林填充得到的结果更好，但是考虑到这里删除元素其实达到了五分之一，而且之前一直在删除，所以这里换一种方式。

另外这里还要注意，若缺失样本占总数很大，我们也会直接删除，因为如果在这种情况下作为特征加入的话，可能会带入噪音，影响结果。而月收入这栏缺失数据大小适中，也就是说它满足随机森林使用的条件，所以我们再运用python中的RandomForestRegressor等函数。于是就有了如下操作：

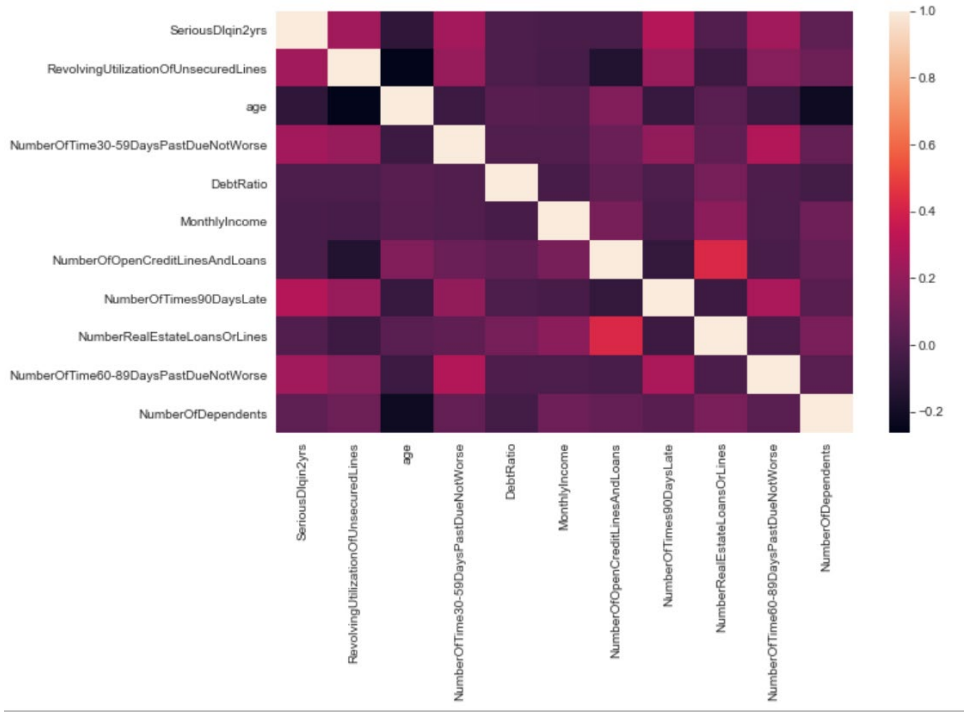
```
def myFiller(data):
    haveKnown=data[data['MonthlyIncome'].notnull()]
    haveNotKnown=data[data['MonthlyIncome'].isnull()]
    x_0=haveKnown.iloc[:,[1,2,3,4,6,7,8,9,10]]
    y_0=haveKnown.iloc[:,5]
    x_1=haveNotKnown.iloc[:,[1,2,3,4,6,7,8,9,10]]
    randomForest=RandomForestRegressor(random_state=0,n_estimators=200,max_depth=3,n_jobs=-1)
    y_2=randomForest.fit(x_0,y_0).predict(x_1)
    return y_2
```

其次，我们需要对于2.3中分析的异常值进行删除处理。我们编写了删除myDelete：

```
def myDelete(data):
    data=data[data['RevolvingUtilizationOfUnsecuredLines']<1]
    data=data[data['age']>18]
    data=data[data['NumberOfTime30-59DaysPastDueNotWorse']<80]
    data=data[data['NumberOfTime60-89DaysPastDueNotWorse']<80]
    data=data[data['NumberOfTimes90DaysLate']<80]
    data=data[data['NumberOfDependents']<20]
    data=data[data['NumberRealEstateLoansOrLines']<50]
    return data
trainingData=myDelete(trainingData)
testData=myDelete(testData)
```

最后，我们需要额外判断特征之间的共线性。当两个特征之间的相关系数> 0.8时，可以认为两个特征拥有很强的相关性，即共线性。在进行特征选择时，共线性的特征不仅会增加计算成本，还有可能降低模型的稳定性，因此需要筛选。

通过创建共线性表格和绘制热力图（如下），我们可以发现特征之间共线性不强，相关性最高不超过0.5，因此不需要额外操作。但是，判断共线性的这一步骤是不可缺少的。



4. 具体实现方法

4.1特征选择

所谓特征选择，就是需要选择出10个特征中可以用来建立模型的特征。

首先，我们进行准备工作——划分数据。我们将训练集数据划分成training集和testing集，分别用于训练和评估。

之后，我们要进行分箱工作。这是因为我们需要对变量进行WOE编码。WOE的计算公式如下：

$$WOE_i = \ln(\frac{P(y_i)}{P(y_n)}) = \ln(\frac{y_i/y_T}{n_i/n_T})$$

WOE表示的实际上是“当前分组中响应客户（即坏客户）占所有响应客户的比例”和“当前分组中没有响应的客户（即好客户）占所有没有响应的客户的比例”的差异。

我们分箱计算WOE的目的是为了进一步计算IV值。IV值，即信息价值，是用来衡量自变量的预测能力。因此，一个变量的IV值越高，其对于因变量的预测性能就越好，我们可能就会考虑将其选择出来用于建立模型。

IV值的计算公式如下：

$$IV_i = (P(y_i) - P(n_i)) \times WOE_i = (\frac{y_i}{y_T} - \frac{n_i}{n_T}) \times \ln(\frac{y_i/y_T}{n_i/n_T})$$

有了一个变量各分组的IV值，把各分组的IV相加就可得整个变量的IV值：

$$IV = \sum_{i=1}^{n} IV_i$$

在上述理论知识的指导下，我们需要对10个特征分别分箱、计算WOE和IV值。对于分箱的操作来说，具体又有两种操作方式。对于连续性变量，我们利用qcut函数进行自动的最优分箱，编写了autoBin函数：

```
#采取定义自动分箱函数的方式--最优分箱
def autoBin(target,data,n=10): #data为待分箱变量，n为分箱数量
    r=0 #斯皮尔曼初始值
    bad=target.sum() #计算坏样本的数量
    good=target.count()-bad #计算好样本的数量
    #以下是机器来自动选择指定最优的分箱节点，代替我们自己来设置
    while np.abs(r)<1:
        #用pd.qcut实现最优分箱，Bucket：将data分为n段（n由斯皮尔曼系数决定）
        d1=pd.DataFrame({"data":data,"target":target,"Bucket":pd.qcut(data,n)})
        #按照分箱结果进行分组聚合
        d2=d1.groupby('Bucket',as_index=True)
        #以斯皮尔曼系数作为分箱终止条件
        r,p=stats.spearmanr(d2.mean().data,d2.mean().target)
        n-=1
    d3=pd.DataFrame(d2.data.min(),columns=['min'])
    d3['min']=d2.min().data #左边界
    d3['max']=d2.max().data #右边界
    d3['bad']=d2.sum().target #每个箱体中坏的样本的数量
    d3['total']=d2.count().target #每个箱体的总样本数
    d3['rate']=d2.mean().target
    print(d3['rate'])
    print('----------------------')
```

```
#计算每个箱体的WOE值
d3['woe']=np.log((d3['bad']/bad)/((d3['total']-d3['bad'])/good))
#计算每个箱体中坏样本所占坏样本总数的比例
d3['badattr']=d3['bad']/bad
#计算每个箱体中好样本所占好样本总数的比例
d3['goodattr']=(d3['total']-d3['bad'])/good
#计算变量的IV值
iv=((d3['badattr']-d3['goodattr'])*d3['woe']).sum()
#对箱体从大到小进行排序
d4=(d3.sort_values(by = 'min')).reset_index(drop=True)
print('分箱结果：')
print(d4)
print('IV值为：')
print(iv)
print('\n')
woe=list(d4['woe'].round(3))
cut=[] #cut用来存放箱段节点
cut.append(float('-inf'))#在列表前加上-inf
for i in range(1,n+1):# n是前面的分箱的分割数，所以分成n+1份
    qua=data.quantile(i/(n+1))#quantile 分为数,得到分箱的节点
    cut.append(round(qua,4))#保留4位小数，返回cut
cut.append(float('inf'))#在列表后加上inf
return d4,iv,cut,woe
```

我们以月收入为例查看分箱结果：

```
Bucket
(-0.001, 3416.0]       0.068875
(3416.0, 6900.0]       0.064987
(6900.0, 1794060.0]    0.046244
Name: rate, dtype: float64
----------------------
分箱结果：
      min        max   bad  total      rate       woe   badattr  goodattr
0     0.0     3416.0  2620  38040  0.068875  0.146609  0.382593  0.330420
1  3417.0     6900.0  2473  38054  0.064987  0.084332  0.361127  0.331922
2  6902.0  1794060.0  1755  37951  0.046244 -0.275768  0.256279  0.337659
IV值为：
0.032554004505139844
```

可以看出，我们对月收入分了三个箱，分别计算了WOE值，并算出总的IV值约为0.03。

对于离散型的变量，我们则不能采用自动分箱的方式。我们可以手动设置分箱边界，进行手动分箱，编写myBin函数：

```
#手动分箱法
def myBin(target,data,cut):
    bad=target.sum() #坏样本数量
    good=target.count()-bad #好样本数量
    #建立个数据框 data-- 各个特征变量 ， target--用户好坏标签 ， Bucket--各个分箱
    d1=pd.DataFrame({"data":data,"target":target,"Bucket":pd.cut(data,cut)})
    #按照分箱结果进行分组聚合
    d2=d1.groupby('Bucket',as_index=True)
    #添加 min 列 ,不用管里面的 d2.X.min()
    d3=pd.DataFrame(d2.data.min(),columns=['min'])
    d3['min']=d2.min().data
    d3['max']=d2.max().data
    d3['bad']=d2.sum().target
    d3['total']=d2.count().target
    d3['rate']=d2.mean().target
    #计算每个箱体的WOE值
    d3['woe']=np.log((d3['bad']/bad)/((d3['total']-d3['bad'])/good))
    #每个箱体中坏样本所占坏样本总数的比例
    d3['badattr']=d3['bad']/bad
    #每个箱体中好样本所占好样本总数的比例
    d3['goodattr']=(d3['total']-d3['bad'])/good
    #计算变量的N值
    iv=((d3['badattr']-d3['goodattr'])*d3['woe']).sum()
    #对箱体从大到小进行排序
    d4=(d3.sort_values(by='min')).reset_index(drop=True)
    woe=list(d4['woe'].round(3))
```

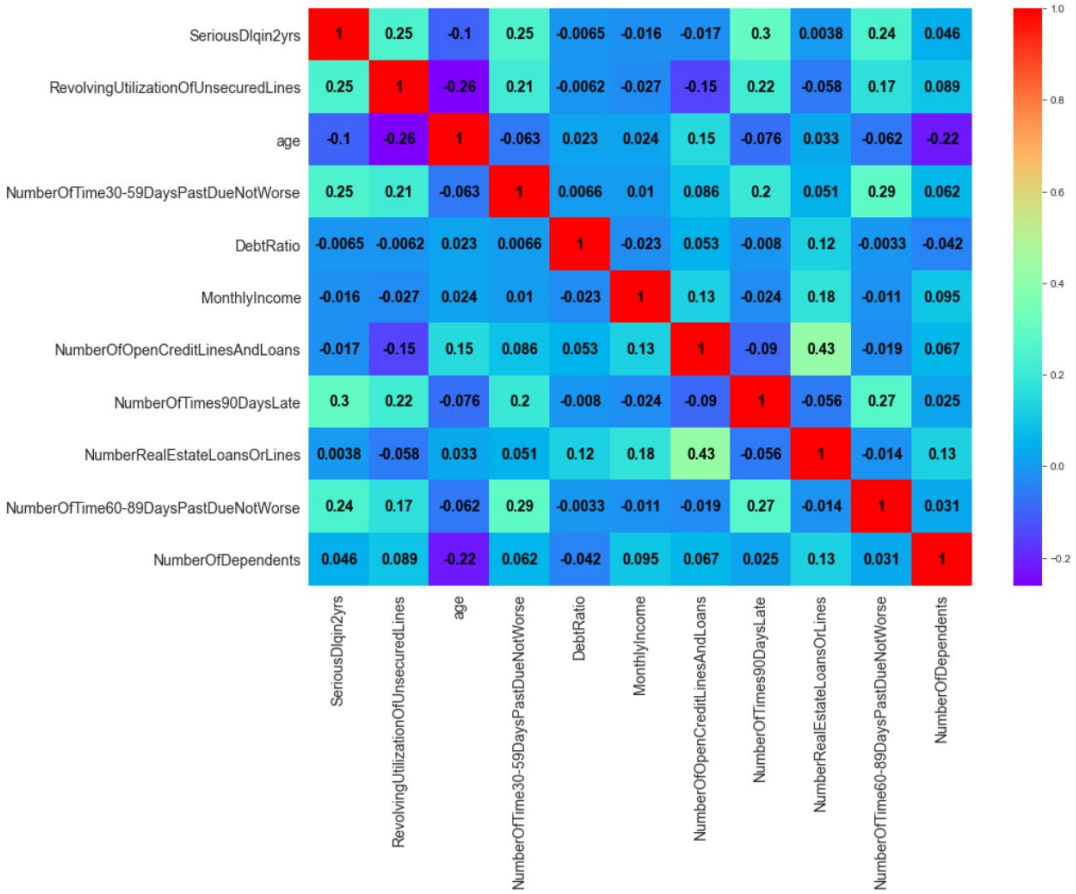
我们以逾期 30-59 天次数这一变量为例，查看分箱结果：

```
Bucket
(-inf, 0.0]    0.038333
(0.0, 1.0]     0.133780
(1.0, 3.0]     0.263004
(3.0, 5.0]     0.400549
(5.0, inf]     0.496732
Name: rate, dtype: float64
------------------------
分箱结果:
   min  max   bad  total      rate       woe   badattr  goodattr
0    0    0  3709  96758  0.038333 -0.471652  0.541618  0.868019
1    1    1  1598  11945  0.133780  0.882768  0.233353  0.096523
2    2    3  1173   4460  0.263004  1.720301  0.171291  0.030663
3    4    5   292    729  0.400549  2.347532  0.042640  0.004077
4    6   12    76    153  0.496732  2.737640  0.011098  0.000718
IV值为:
0.6356038610056689
```

可以看出，我们分了5个箱，计算出WOE值，其总的IV值约为0.64。

做好了上述准备后，我们可以开始进行特征选择了。我们的特征选择基于两项指标，一项是特征和因变量的相关性，另一项是IV值。

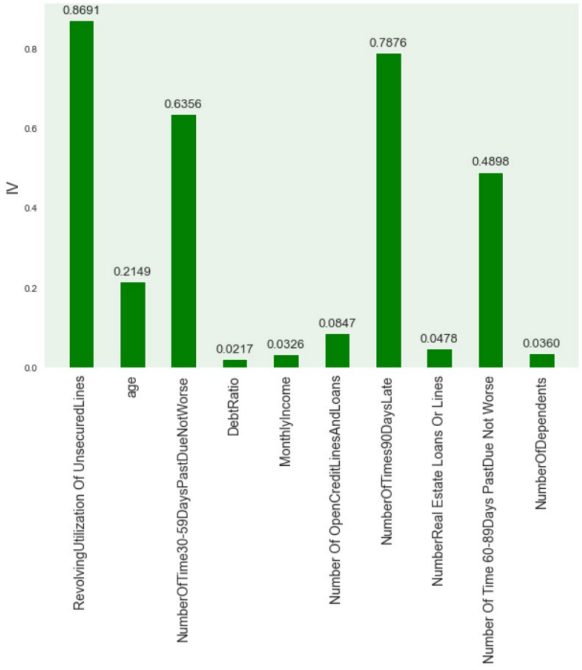
首先来看相关性。通过绘制热力图，并标出相关系数如下：



值得注意的是，这张图似乎与3中的一张热力图很相似。事实上，这两张图是相同的。但是，我们两次绘制热力图背后的逻辑是完全不同的。在3的共线性判断部分，我们考察的是除因变量外，10个自变量之间的相关性关系，以期找出是否存在两个自变量关联程度较高。在这里，我们考察的是10个自变量同1个因变量之间的相关性关系，以期找出能够显著决定因变量取值的自变量，将其选择出来用以后期建模。这两者的逻辑有着本质的不同，不可混为一谈。

我们可以发现，可用额度比值、逾期30－59天次数、逾期60－89天次数、逾期90天次数这四项特征与因变量（好坏客户）的相关性相对较高，但是也不是非常高，仅在0.25~0.3左右。这也间接说明，利用相关系数来进行特征选择不是一种非常有效的方式。

更为有效的方式是利用IV值的筛选。由于我们前面已经做足了准备，因此我们只需要将每个变量的IV值利用柱状图表示出来，即可进行选择：



IV值的筛选标准为：

< 0.02: unpredictive $0.02 - 0.1$: weak $0.1 - 0.3$: medium > 0.3: strong

因此我们通过上图可以发现，除了可用额度比值、逾期30－59天次数、逾期60－89天次数、逾期90天次数这四项特征外，年龄的IV值也达到了了medium水平，因此也可以将其选择出来。

综上，我们选择出了五项特征，用于接下来的模型建立。它们分别是：

可用额度比值、年龄、逾期30－59天次数、逾期60－89天次数、逾期90天次数。

4.2模型建立

特征选择完毕，我们需要在前述数据分析的基础上建立我们的逻辑回归模型。

首先，我们需要在已有分箱的基础上将相关性较大的几个变量转化为WOE（Weight of Evidence）。前面WOE已经算好，我们只需将所有相关性较大的变量转化为其对应分箱的WOE。

In [50]:
```
#5、模型建立
#5.1 一些准备
#将筛选后的变量转换为WOE值，便于信用评分
def toWOE(var,var_name,woe,cut):
    woe_name=var_name+'woe'
    for i in range(len(woe)): # len(woe) 得到woe里 有多少个数值
        if i==0:
            #将woe的值按 cut分箱的下节点，顺序赋值给var的woe_name列,分箱的第一段
            var.loc[(var[var_name]<=cut[i+1]),woe_name]=woe[i]
        elif(i>0)and(i<=len(woe)-2):
            #处理中间的分箱区间
            var.loc[((var[var_name]>cut[i])&(var[var_name]<=cut[i+1])),woe_name]=woe[i]
        else:
            #大于最后一个分箱区间的上限值,最后一个值是正无穷
            var.loc[(var[var_name]>cut[len(woe)-1]),woe_name]=woe[len(woe)-1]
    return var
```

```
In [51]: x1_name='RevolvingUtilizationOfUnsecuredLines'
         x2_name='age'
         x3_name='NumberOfTime30-59DaysPastDueNotWorse'
         x7_name='NumberOfTimes90DaysLate'
         x9_name='NumberOfTime60-89DaysPastDueNotWorse'
         training=toWOE(training,x1_name,x1_woe,x1_cut)
         training=toWOE(training,x2_name,x2_woe,x2_cut)
         training=toWOE(training,x3_name,woex3,cutx3)
         training=toWOE(training,x7_name,woex7,cutx7)
         training=toWOE(training,x9_name,woex9,cutx9)

In [52]: Y=training['SeriousDlqin2yrs']   #因变量
         #剔除对因变量影响不明显的变量
         X=training.drop(['SeriousDlqin2yrs','DebtRatio','Month
         X=training.iloc[:,-5:]
         X.head(5)
```

然后在训练之前设置好因变量，并抛弃所有相关性不大的其他变量。

```
In [52]: Y=training['SeriousDlqin2yrs']   #因变量
         #剔除对因变量影响不明显的变量
         X=training.drop(['SeriousDlqin2yrs','DebtRatio','MonthlyIncome', 'NumberOfOpenCreditLinesAndLoans','NumberRealEstateLoansOrLines','NumberOfDepe
         X=training.iloc[:,-5:]
         X.head(5)
```

Out[52]:

Unnamed: 0	RevolvingUtilizationOfUnsecuredLineswoe	agewoe	NumberOfTime30-59DaysPastDueNotWorsewoe	NumberOfTimes90DaysLatewoe	NumberOfTime60-89DaysPastDueNotWorsewoe
13017	1.047	-0.207	-0.472	-0.347	-0.242
60720	-1.138	-0.986	-0.472	-0.347	-0.242
67395	1.047	0.130	0.883	-0.347	-0.242
56109	-1.197	-0.986	-0.472	-0.347	-0.242
130935	-1.197	-0.504	-0.472	-0.347	-0.242

```
In [53]: #5.2 利用STATSMODEL包来建立逻辑回归模型得到回归系数，后面可用于建立标准评分卡
         X1=sm.add_constant(X)
         logit=sm.Logit(Y,X1)
         result=logit.fit()
         print(result)
         print(result.summary())
```

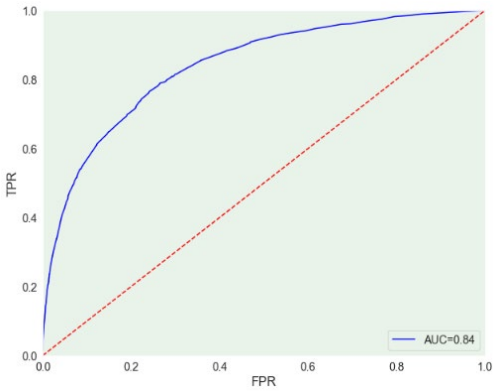
最后利用STATSMODEL包建立逻辑回归模型，并得出相应的回归系数。在“coef”这一栏下的数值，就是我们训练得到的回归系数。在后续步骤中，还会再次用到这些回归系数。

```
Optimization terminated successfully.
         Current function value: 0.176636
         Iterations 8
<statsmodels.discrete.discrete_model.BinaryResultsWrapper object at 0x0000023A091EB880>
                           Logit Regression Results
==============================================================================
Dep. Variable:       SeriousDlqin2yrs   No. Observations:               114045
Model:                          Logit   Df Residuals:                   114039
Method:                           MLE   Df Model:                            5
Date:                Sun, 30 May 2021   Pseudo R-squ.:                 0.2222
Time:                        10:29:45   Log-Likelihood:                -20144.
converged:                       True   LL-Null:                       -25899.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==========================================================================================
                                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                                     -2.7251      0.015   -183.611      0.000      -2.754      -2.696
RevolvingUtilizationOfUnsecuredLineswoe    0.6565      0.016     41.094      0.000       0.625       0.688
agewoe                                     0.5041      0.033     15.082      0.000       0.439       0.570
NumberOfTime30-59DaysPastDueNotWorsewoe    0.5567      0.016     34.615      0.000       0.525       0.588
NumberOfTimes90DaysLatewoe                 0.5965      0.013     44.353      0.000       0.570       0.623
NumberOfTime60-89DaysPastDueNotWorsewoe    0.4276      0.018     24.052      0.000       0.393       0.462
==========================================================================================
```

4.3模型评估

模型建立完毕，我们通过AUC和KS两个指标来评估上一步建立的模型是否可靠。

我们使用建模初期保留的testing测试集，利用sklearn.metrics，比较两个分类器，自动计算ROC和AUC。

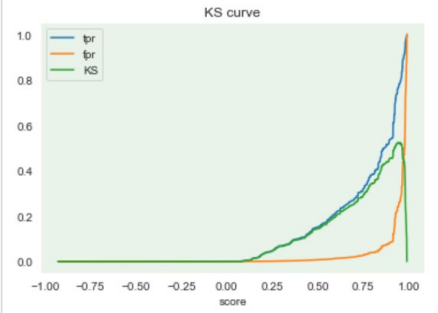



```
In [55]: #测试集的特征和标签
test_X=testing.iloc[:,-5:]
test_Y=testing.iloc[:,0]
#评估
X3=sm.add_constant(test_X)
resu=result.predict(X3)
print(resu)
fpr,tpr,threshold=metrics.roc_curve(test_Y,resu)    #评估算法
rocauc=metrics.auc(fpr,tpr)    #计算AUC
#绘图
plt.figure(figsize=(10,8))   #只能在这里面设置
plt.plot(fpr,tpr,'b',label='AUC=%0.2f'% rocauc)
plt.legend(loc='lower right',fontsize=14)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.ylabel('TPR',fontsize=16)
plt.xlabel('FPR',fontsize=16)
plt.show()
```

ROC图像中，横轴为FPR（False Positive Rate），即假正判率，这个数值越大说明模型敏感度越高；纵轴为TPR（True Positive Rate），即真正判率，这个数值越大说明模型精确度越高。ROC曲线向我们展示了，随着模型敏感度提高，模型精确度的变化情况。其中AUC为ROC曲线下方的面积，故而ROC曲线向上拱的越明显，即AUC越大，模型拟合效果越好，当然过高的AUC值表明该模型存在过拟合风险。这里我们作图得到AUC值高达0.84，说明这个模型的拟合效果不错。

下面我们来看看KS值。

```
In [56]: #KS指标：用以评估模型对好、坏客户的判别区分能力
#计算累计坏客户与累计好客户百分比的最大差距。
fig,ax = plt.subplots()
ax.plot(1-threshold,tpr,label='tpr')
ax.plot(1-threshold,fpr,label='fpr')
ax.plot(1-threshold,tpr-fpr,label='KS')
plt.xlabel('score')
plt.title('KS curve')
plt.xlim=([0.0,1.0])
plt.ylim=([0.0,1.0])
plt.figure(figsize=(20,20))
legend=ax.legend(loc='upper left')
plt.show()
max(tpr-fpr)
```




```
<Figure size 1440x1440 with 0 Axes>
Out[56]: 0.5241713995900088
```

KS曲线的计算方式为TPR－FPR这个差值，而KS值即为KS曲线上的最大值。结合KS曲线的计算方式，我们可得知KS值越大说明模型对好坏客户（阳性和阴性）的区分能力越强。与AUC类似，过高的KS同样表明模型存在一定风险。这里我们看到KS值在0.524左右，说明这个模型的好坏客户区分能力比较可靠。

4.4 建立信用评分卡

我们使用了查资料[1]得出的公式，来构造我们的评分函数：

$$\text{Score} = \sum_{1}^{n}\left(\text{WOE}_i \times \text{coef}_i + \frac{\text{coef}_0}{n}\right) \times \text{factor} + \text{offset}$$

$$\text{factor} = p/\log(2)$$

$$\text{offset} = b - p \times \frac{\log(o)}{\log(2)}$$

[1] https://blog.csdn.net/gxhzoe/article/details/80428560

In [58]:
```
#6、创建信用评分卡
#6.1 在建立标准评分卡之前，还需要设定几个评分卡参数：基础分值、 PDO（比率翻倍的分值）和好坏比。
#我们取600分为基础分值b，取20为PDO （每高20分好坏比翻一倍），好坏比0取20。
p=20/np.log(2) #比例因子
q=600-20*np.log(20)/np.log(2) #偏移量
x_coe=[-2.7251,0.6565,0.5041,0.5576,0.5965,0.4276] #上面计算得出的回归系数
base=round(q+p*x_coe[0],0) #基础得分
#总分=基础分+各部分得分
def Score(coe,woe,factor):
    scores=[]
    for k in woe:
        score=round(coe*k*factor,0)
        scores.append(score)
    return scores
#每一项得分
x1_score=Score(x_coe[1],x1_woe,p) #注：'RevolvingUtilizationOfUnsecuredLines'
x2_score=Score(x_coe[2],x2_woe,p) #注：'age'
x3_score=Score(x_coe[3],woex3,p) #注：'NumberOfTime30-59DaysPastDueNotWorse'
x7_score=Score(x_coe[4],woex7,p) #注：'NumberOfTimes90DaysLate'
x9_score=Score(x_coe[5],woex9,p) #注：'NumberOfTime60-89DaysPastDueNotWorse'
```

其中，p为好坏比翻一倍时评分增加的分数，b为基础分值，o为基础分值对应的好坏比。

由于题设要求，我们的得分在300~900之间，我们自然选择600分作为基础分b。基于前述数据分析，15万数据中好客户占比超过90%，故我们将基础得分对应的好坏比o设置为 20。为了将得分限制在300~900分， p数值设置为20。

得分计算函数设置完毕，我们来计算每条数据，即每个客户的得分。

In [60]:
```
#计算得分
trainingData['BaseScore']=np.zeros(len(trainingData))+base
trainingData['x1']=compute(trainingData['RevolvingUtilizationOfUnsecuredLines'],x1_cut,x1_score)
trainingData['x2']=compute(trainingData['age'], x2_cut, x2_score)
trainingData['x3']=compute(trainingData['NumberOfTime30-59DaysPastDueNotWorse'], cutx3, x3_score)
trainingData['x7']=compute(trainingData['NumberOfTimes90DaysLate'], cutx7, x7_score)
trainingData['x9']=compute(trainingData['NumberOfTime60-89DaysPastDueNotWorse'],cutx9,x9_score)
trainingData['Score']=trainingData['x1']+trainingData['x2']+trainingData['x3']+trainingData['x7']+trainingData['x9']+base
#选取需要的列，就是评分列
scoreTable1=trainingData.iloc[:,[0,-7,-6,-5,-4,-3,-2,-1]]
scoreTable1.head(5)
```

	A	B	C	D	E	F	G	H	I
1	nnamed:	eriousDlqin2yr	BaseScore	ilizationOfUns	age	e30-59DaysPast	rOfTimes90Days	e60-89DaysPast	Score
2	1	1	435	20	3	28	34	23	543
3	2	0	435	20	4	14	34	23	530
4	3	0	435	20	5	28	47	23	558
5	4	0	435	-5	7	14	34	23	508
6	5	0	435	20	3	28	34	23	543
7	6	0	435	-5	-14	14	34	23	487
8	7	0	435	-5	-3	14	34	23	498
9	8	0	435	20	5	14	34	23	531
10	10	0	435	-5	-3	14	34	23	498
11	11	0	435	20	7	14	34	23	533
12	12	0	435	-23	2	14	34	23	485
13	13	0	435	-23	3	14	34	23	486
14	14	1	435	20	4	38	58	33	588
15	15	0	435	-23	-14	14	34	23	469
16	16	0	435	20	-12	14	34	23	514

4.5 对测试集进行预测，转化为信用评分卡

我们已经确认自己得到的模型可靠，故此处只需对测试集重复前述的部分流程。

首先，将测试集数据转化为WOE，剔除掉和因变量相关性差的自变量，然后用已经建立好的模型对其进行预测。预测完毕后，再调用建立好的评分函数进行评分。

In [62]:
```
#7、利用模型预测测试集
#7.1 测试集转化为WOE值
testData=toWOE(testData,x1_name,x1_woe,x1_cut)
testData=toWOE(testData,x2_name,x2_woe,x2_cut)
testData=toWOE(testData,x3_name,woex3,cutx3)
testData=toWOE(testData,x7_name,woex7,cutx7)
testData=toWOE(testData,x9_name,woex9,cutx9)
#自变量，剔除对因变量影响不明显的变量
testData=testData.drop(['DebtRatio','MonthlyIncome', 'NumberOf(
#测试集的特征和标签
test_X=testData.iloc[:,-5:]
test_Y=testData.iloc[:,0]
#7.2 评估
X_=sm.add_constant(test_X)
list=result.predict(X_)
testData['predict']=list
```

In [63]:
```
#7.3 对测试集进行评分
testData['BaseScore']=np.zeros(len(testData))+base
testData['x1'] = compute(testData['RevolvingUtilizationOfUnsecuredLines'], x1_cut, x1_score)
testData['x2'] = compute(testData['age'], x2_cut, x2_score)
testData['x3'] = compute(testData['NumberOfTime30-59DaysPastDueNotWorse'], cutx3, x3_score)
testData['x7'] = compute(testData['NumberOfTimes90DaysLate'], cutx7, x7_score)
testData['x9'] = compute(testData['NumberOfTime60-89DaysPastDueNotWorse'],cutx9,x9_score)
testData['Score'] = testData['x1'] + testData['x2'] + testData['x3'] + testData['x7'] +testData['x9']  + base
#选取需要的列，就是评分列
scoretable2=testData.iloc[:,[0,-8,-7,-6,-5,-4,-3,-2,-1]]
print(scoretable2.head(5))
```

```
            SeriousDlqin2yrs   predict  BaseScore    x1   x2    x3    x7  \
Unnamed: 0
1                        NaN  0.076739      435.0  20.0  4.0  14.0  34.0
2                        NaN  0.027417      435.0  -5.0 -3.0  14.0  34.0
3                        NaN  0.015523      435.0 -22.0 -7.0  14.0  34.0
4                        NaN  0.073192      435.0  -5.0  5.0  28.0  34.0
5                        NaN  0.086396      435.0  20.0  7.0  14.0  34.0

              x9  Score
Unnamed: 0
1           23.0  530.0
2           23.0  498.0
3           23.0  477.0
4           23.0  520.0
5           23.0  533.0
```

最后，将预测并转换为得分的结果输出到目标excel文件中。

In [64]:
```
#输出结果
colNameDict={'x1': 'RevolvingUtilizationOfUnsecuredLines' ,'x2':'age','x3':'NumberOfTime30-59DaysPastDueNotWorse',
             'x7':'NumberOfTimes90DaysLate', 'x9':'NumberOfTime60-89DaysPastDueNotWorse'}
scoretable2=scoretable2.rename(columns=colNameDict,inplace=False)
```

predict	BaseScore	ilizationOfUns	age	e30-59DaysPast	rOfTimes90Days	e60-89DaysPast	Score
0.076739446	435	20	4	14	34	23	530
0.027417396	435	-5	-3	14	34	23	498
0.015523241	435	-22	-7	14	34	23	477
0.073192089	435	-5	5	28	34	23	520
0.08639633	435	20	7	14	34	23	533
0.023695521	435	-5	-7	14	34	23	494
0.076383066	435	20	2	14	34	23	528
0.022138259	435	-22	-14	28	34	23	484
0.010910252	435	-23	-12	14	34	23	471
0.019593406	435	-23	5	14	34	23	488
0.018344733	435	-22	2	14	34	23	486
0.010910252	435	-23	-12	14	34	23	471
0.03872072	435	-5	7	14	34	23	508
0.06207429	435	20	-3	14	34	23	523
0.022034074	435	-22	7	14	34	23	491
0.042777829	435	20	-14	14	34	23	512
0.020351458	435	-22	5	14	34	23	489
0.186177157	435	20	7	14	34	33	543
0.076739446	435	20	4	14	34	23	530
0.019322574	435	-22	3	14	34	23	487
0.023695521	435	-5	-7	14	34	23	494
0.010910252	435	-23	-12	14	34	23	471
0.615888459	435	20	-12	28	47	33	551
0.069681822	435	-5	2	28	34	23	517
0.018344733	435	-22	2	14	34	23	486
0.076739446	435	20	3	14	34	23	529

在predict一栏，我们看到的数值即为客户的违约概率。如果这个数值超过0.5，我们认为这个客户很可能是坏客户。在上图中，红色框内的数据的predict值达到了0.616，因此我们认为这位客户为坏客户，银行在为这位客户办理信用卡业务时需要格外谨慎。

5. 总结和展望

本次建模完整实现了数据读入、数据分析、数据预处理（即数据清洗）、特征选择、模型建立、模型评估、创建信用评分卡、预测测试集等八个步骤，建立起了一个具有良好预测效果的信用评分模型。通过本次建模过程，我们可以深刻地感受到，身处一个数据爆炸的时代，对数据进行合理的筛选、划分、建模显得尤为重要。我们的工作由于时间原因也有不足之处，比如我们在模型的选择上比较保守，后期我们将继续尝试梯度上升法等方式进行更好地分类。