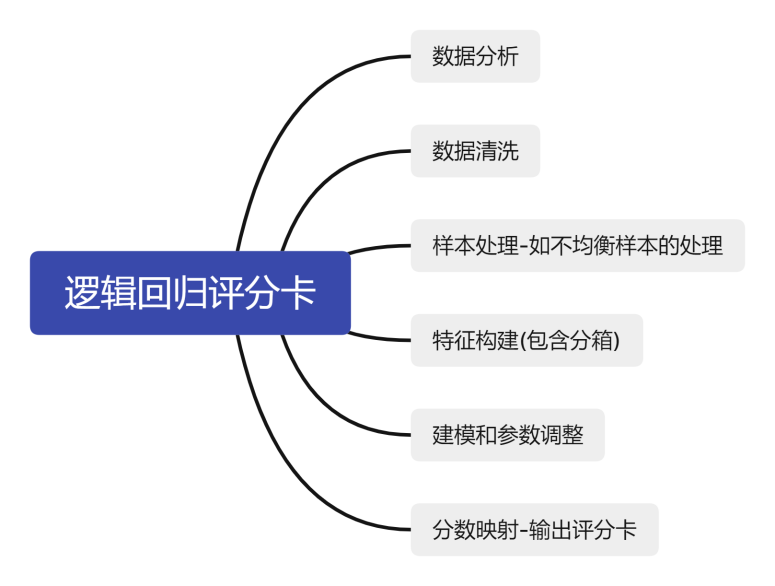
**智能风控学习笔记-基于逻辑回归的评分卡案例**

这次的文章我们不打算进行相关的理论知识的学习，主要是讲述一个基于逻辑回归模型进行构建评分卡模型的的案例，之后的文章我们再进行相关知识点的总结，以这个思路来进行文章的章节布局。

**一、建模思路**

在建模之前，我们先构建一个大纲这样也比较清晰，本次建模的过程思路大致如下：

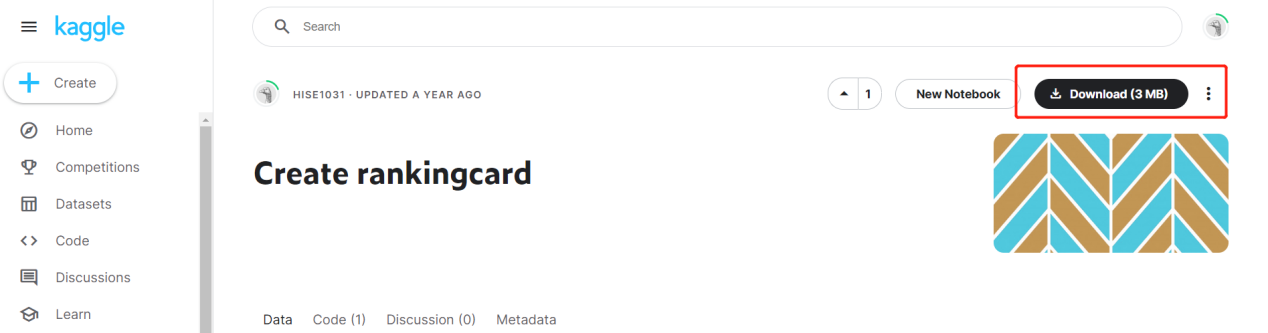


**二、数据集说明**

本次实验的数据集来源于kaggle上的数据集，数据集下载地址为：

**<https://www.kaggle.com/datasets/hise1031/create-rankingcard>**

下载后的文件archive.zip，解压之后就可以得到数据集rankingcard.csv。



字段说明：

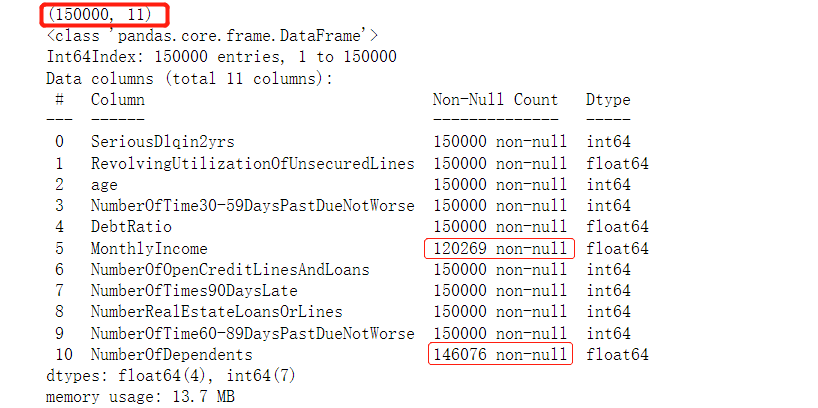


**三、数据分析**

先简单的查看一下数据情况:

from sklearn.ensemble import RandomForestRegressor  
from sklearn.linear\_model import LogisticRegression as LR  
from matplotlib import pyplot as plt  
import numpy as np  
import pandas as pd  
  
df = pd.read\_csv('rankingcard.csv', index\_col=0)  
print(df.shape)  
df.info()

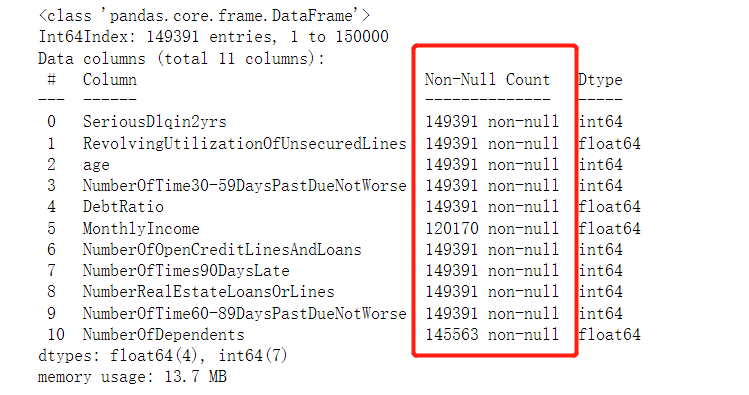
输出如下：



可以看出数据量是15万条数据，且MonthlyIncome、NumberOfDependents特征出现空值的情况，后面将进行统一的处理。

**四、数据清洗**

**去除重复值：**一般来说，我们的数据集中是存在一些重复性的数据的，因此需要进行一定的去重操作，pandas提供了一个非常方便的方法。  
df.drop\_duplicates(inplace=True)  
df.info()



删除一些数据以后，需要进行重置index的操作：

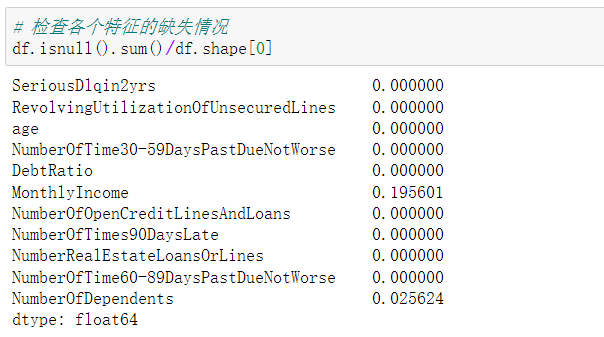
df.index=range(df.shape[0])

当然了我们也可以使用下属方法进行重置索引：

df.reset\_index(drop=True, inplace=True)

**缺失值填充：**样本中存在缺失值的情况也是比较常见的，在数据简单的探索步骤中我们发现特征MonthlyIncome、NumberOfDependents是存在缺失值情况的，我们将进行填充。这里顺便提一下我们在建模过程中常使用的检查缺失值占比的方法。

*# 检查各个特征的缺失情况*df.isnull().sum()/df.shape[0]



MonthlyIncome列缺失值的数量可以这样进行统计：

df.loc[:,"MonthlyIncome"].isnull().sum()

MonthlyIncome和NumberOfDependents填充是不一样的，对于收入这种连续性数据的填充我们一般采用预测方法进行填充，对于家庭人员数量的这种一般采用均值或者众数进行填充。

*# 填充家庭成员缺失值 ，这里使用均值填充*df["NumberOfDependents"].fillna(df["NumberOfDependents"].mean(),inplace=True)

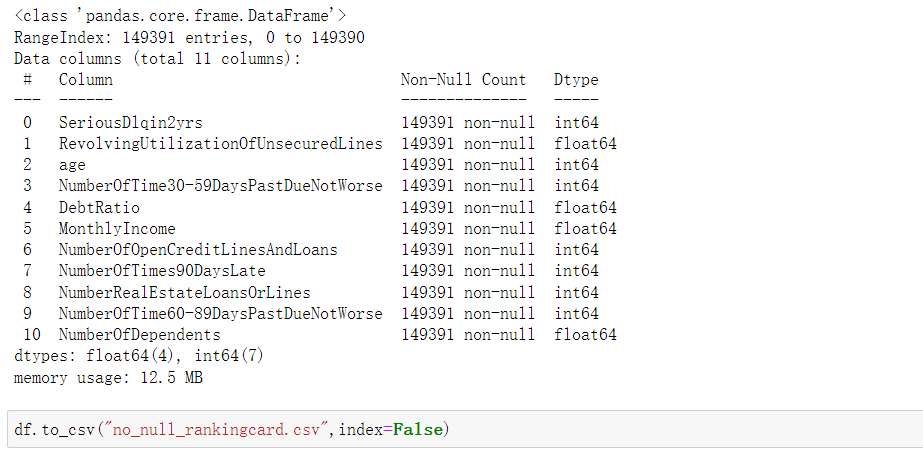
接下来是MonthlyIncome的填充，由于NumberOfDependents已经填充完毕，那么我们就可以使用NumberOfDependents的数据来预测填充MonthlyIncome，为什么不直接使用一般同值填充的原因是因为：缺失的MonthlyIncome都填充同样的值，肯定会对模型的准确率造成一定的影响，随机森林代码如下：

import pandas as pd  
from sklearn.ensemble import RandomForestRegressor  
def fill\_missing\_rf(X, y, fillcolumn):  
 df = X.copy()  
 *# 待预测列* fill\_df = df.loc[:, fillcolumn]  
 *# 去除预测列，将剩下的列与标签组成数据集* df = pd.concat([df.loc[:, df.columns != fillcolumn], pd.DataFrame(y)], axis=1)  
 *# 训练集和测试集* y\_train = fill\_df[fill\_df.notnull()]  
 y\_test = fill\_df[fill\_df.isnull()]  
 X\_train = df.iloc[y\_train.index, :]  
 X\_test = df.iloc[y\_test.index, :]  
 *# 预测* rfr = RandomForestRegressor(n\_estimators=100).fit(X\_train, y\_train)  
 y\_pred = rfr.predict(X\_test)  
 *# 对原数据进行填充并返回* df.loc[df.loc[:, fillcolumn].isnull(), fillcolumn] = y\_pred  
 return df

参数方面其中X为不包含标签类的数据集，y为标签列，本案例调用如下调用如下：

X = df.iloc[:,1:]  
y = df["SeriousDlqin2yrs"]  
new\_df = fill\_missing\_rf(X,y,"MonthlyIncome")

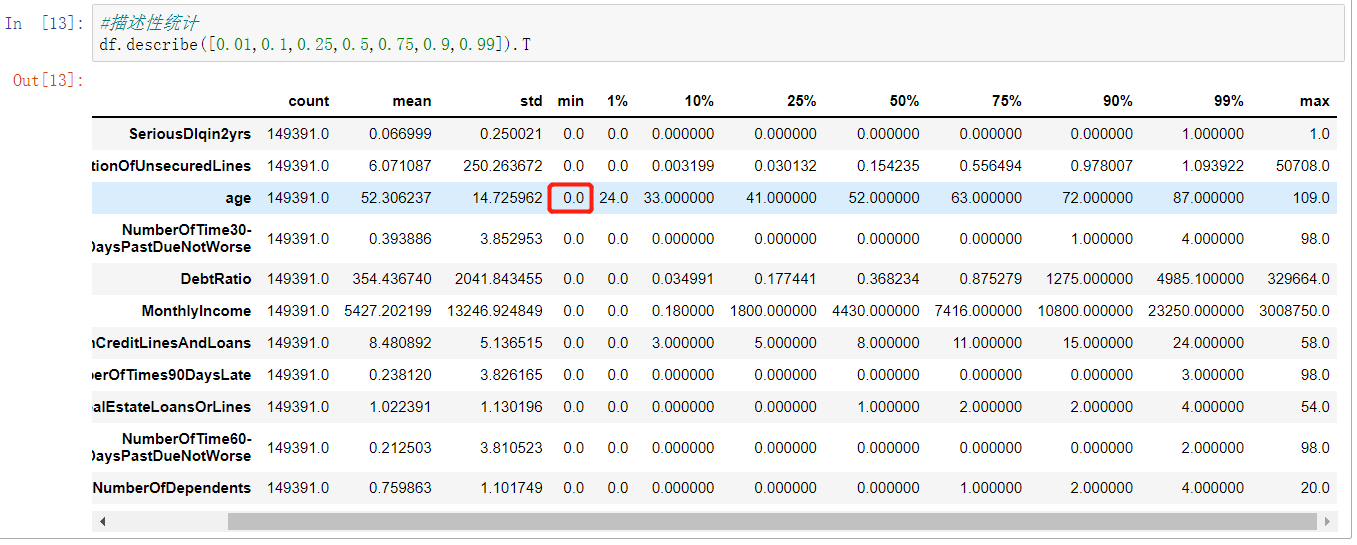
new\_df.info()



这样我们就完成了数据的填充，接下来我们要进行数据当中异常数据的检查和清晰。

**异常值处理：**异常值的处理也是建模过程中必须要进行的一项工作，比如年龄的值域等。

我们可以使用pandas提供的describe查看数据整体的分布情况，其中.T表示对结果进行装置。

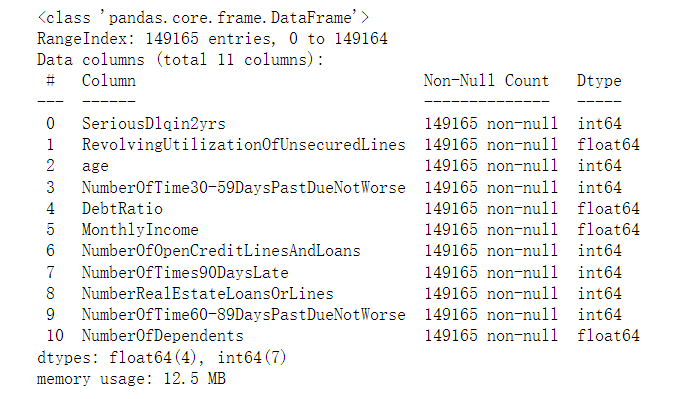


可以发现特征age列存在值为0的情况，这是不符合实际的，我们需要进行处理，这是其中一点，另外值得注意的是在信贷领域，我们要充分考虑客户的逾期次数，一般来说逾期时间过长且次数过多的时候就会被认定为坏客户，实际上各个特征的正常值在什么范围都可以让业务人员给定，这样也比较好指导建模。可以发现特征：

NumberOfTime30-59DaysPastDueNotWorse  
NumberOfTime60-89DaysPastDueNotWorse  
NumberOfTimes90DaysLate

都存在一定的问题，我们这里直接将这三个特征的值大于90的值删除：

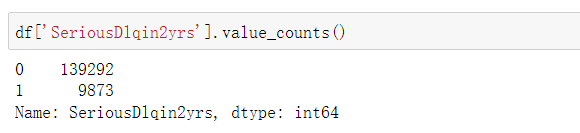
df = df[df['age']>0]  
df = df[df['NumberOfTime30-59DaysPastDueNotWorse']<90]  
df = df[df['NumberOfTime60-89DaysPastDueNotWorse']<90]  
df = df[df['NumberOfTimes90DaysLate']<90]  
df.reset\_index(drop=True, inplace=True)  
df.info()



**五、特征构建-数据处理**

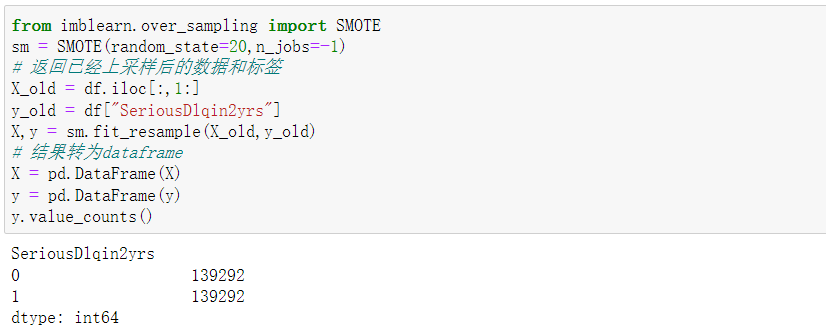
正负样本不均衡的问题也非常常见，这里我们先查看一下样本的分布:

df['SeriousDlqin2yrs'].value\_counts()



很显然样本存在很大的不均衡，这就说明我们在建模的时候需要进行样本的采样操作。对于这种情况我们直接进行**上采样：**

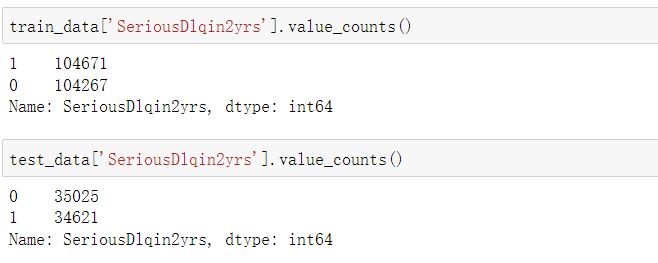
from imblearn.over\_sampling import SMOTE  
sm = SMOTE(random\_state=20,n\_jobs=-1)  
*# 返回已经上采样后的数据和标签*X\_old = df.iloc[:,1:]  
y\_old = df["SeriousDlqin2yrs"]  
X,y = sm.fit\_resample(X\_old,y\_old)  
*# 结果转为dataframe*X = pd.DataFrame(X)   
y = pd.DataFrame(y)   
y.value\_counts()



这样我们就得到了正负样本比例为1:1的数据集，关于采样方法SMOTE算法的原理大家可以参考相关资料进行学习，基于此结果我们来划分数据集：

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25,random\_state=20)   
train\_data = pd.concat([y\_train,X\_train], axis=1)  
train\_data.reset\_index(drop=True, inplace=True)   
test\_data = pd.concat([y\_test, X\_test], axis=1)  
test\_data.reset\_index(drop=True, inplace=True)  
train\_data.to\_csv('train\_data.csv',index = False)  
test\_data.to\_csv('test\_data.csv',index = False)

**这里划分数据集之后，存储为csv文件格式便于后期使用**。我们再查看一下训练集和测试集的数据正负样本的分布情况：



可以看出各个数据集数据的分布情况还是1:1的。接下来我们开始进行数据的分箱，关于数据的分箱的原因和原理以及其他的相关知识我们之后将进行专题讨论，首先我们学习一个知识：两个箱子之间卡方检验的P值很大，则说明他们非常相似，那我们就可以将这两个箱 子合并为一个箱子。

其中注意一个变量IV值的指导意义如下：



**这里插一段，一般在特征很多的时候我们会进行特征的筛选，比如空值和IV值比较低的特征，如果需要进行这样的筛选我们可以进行使用toad模块的select函数即：**

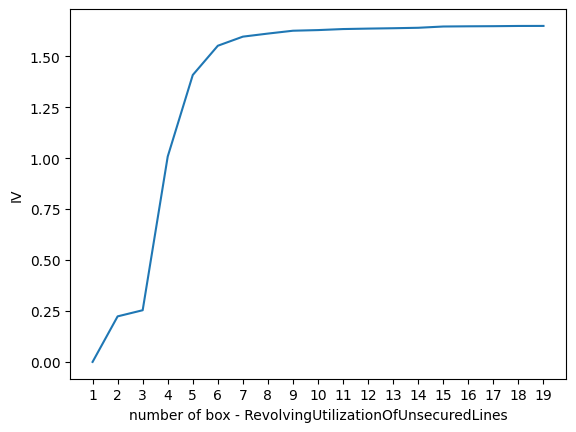
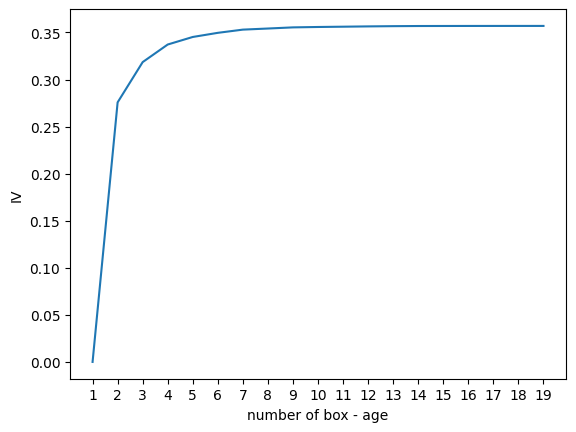
*#缺失率、IV、相关系数进行特征筛选*selected\_data, drop\_list=toad.selection.select(  
 train\_data,  
 train\_data["SeriousDlqin2yrs"],  
 empty=0.7,  
 iv=0.03,  
 corr=1,  
 return\_drop=True,  
 exclude=["SeriousDlqin2yrs"])

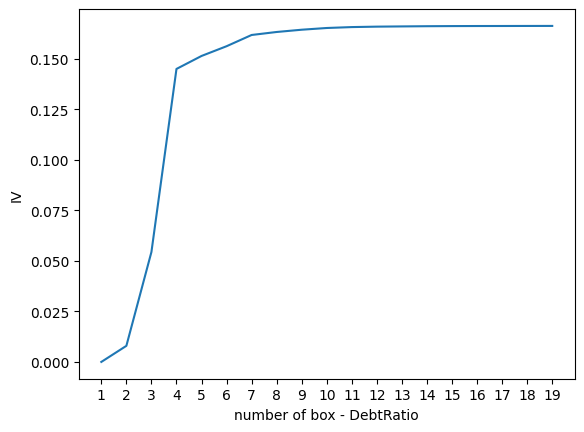
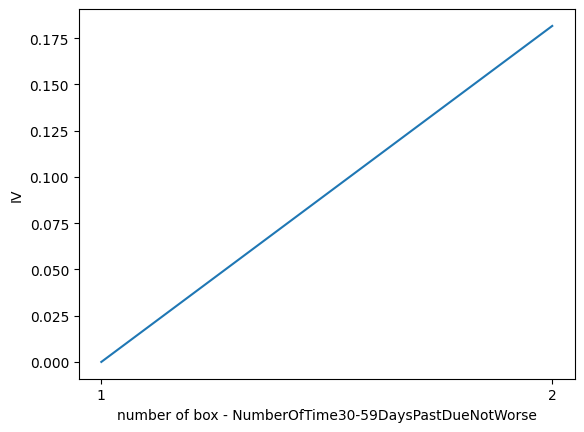
一般我们通过观察IV来认定合理的分箱数是多少，数据分箱的代码在网络上有很多，很多关于风控的书籍也都有封装好的代码可以直接使用，我们一起来看一下。  
import scipy  
def graphforbestbin(df, X, y, n, q=20, graph=True):  
 *'''  
 基于卡方检验的分箱  
 df: 需要输入的数据  
 X: 需要分箱的列名  
 y: 分箱数据对应的标签 y列名  
 n: 保留分箱个数  
 q: 初始分箱的个数  
 graph: 是否要画出IV图像  
 '''* df = df[[X, y]].copy()  
 *# 调用pandas的分箱函数* df["qcut"], bins = pd.qcut(df[X], retbins=True, q=q, duplicates="drop")  
 coount\_y0 = df.loc[df[y] == 0].groupby(by="qcut").count()[y]  
 coount\_y1 = df.loc[df[y] == 1].groupby(by="qcut").count()[y]  
 num\_bins = [\*zip(bins, bins[1:], coount\_y0, coount\_y1)]  
  
 *# 判断每个箱子是否包含正负样本* for i in range(q):  
 if 0 in num\_bins[0][2:]:  
 num\_bins[0:2] = [(  
 num\_bins[0][0],  
 num\_bins[1][1],  
 num\_bins[0][2] + num\_bins[1][2],  
 num\_bins[0][3] + num\_bins[1][3])]  
 continue  
  
 for i in range(len(num\_bins)):  
 if 0 in num\_bins[i][2:]:  
 num\_bins[i - 1:i + 1] = [(  
 num\_bins[i - 1][0],  
 num\_bins[i][1],  
 num\_bins[i - 1][2] + num\_bins[i][2],  
 num\_bins[i - 1][3] + num\_bins[i][3])]  
 break  
 else:  
 break  
  
 *# 定义WOE函数* def get\_woe(num\_bins):  
 columns = ["min", "max", "count\_0", "count\_1"]  
 df = pd.DataFrame(num\_bins, columns=columns)  
 df["total"] = df.count\_0 + df.count\_1  
 df["percentage"] = df.total / df.total.sum()  
 df["bad\_rate"] = df.count\_1 / df.total  
 df["good%"] = df.count\_0 / df.count\_0.sum()  
 df["bad%"] = df.count\_1 / df.count\_1.sum()  
 df["woe"] = np.log(df["good%"] / df["bad%"])  
 return df  
  
 *# 定义IV函数* def get\_iv(df):  
 rate = df["good%"] - df["bad%"]  
 iv = np.sum(rate \* df.woe)  
 return iv  
  
 *# 找最合理的分箱数n* bins\_df = None  
 IV = []  
 axisx = []  
 while len(num\_bins) > n:  
 pvs = []  
 for i in range(len(num\_bins) - 1):  
 x1 = num\_bins[i][2:]  
 x2 = num\_bins[i + 1][2:]  
 pv = scipy.stats.chi2\_contingency([x1, x2])[1]  
 pvs.append(pv)  
  
 i = pvs.index(max(pvs))  
 num\_bins[i:i + 2] = [(  
 num\_bins[i][0],  
 num\_bins[i + 1][1],  
 num\_bins[i][2] + num\_bins[i + 1][2],  
 num\_bins[i][3] + num\_bins[i + 1][3])]  
  
 bins\_df = pd.DataFrame(get\_woe(num\_bins))  
 axisx.append(len(num\_bins))  
 IV.append(get\_iv(bins\_df))  
  
 if graph:  
 plt.figure()  
 plt.plot(axisx, IV)  
 plt.xticks(axisx)  
 plt.xlabel(f"number of box - {X}")  
 plt.ylabel("IV")  
 plt.show()  
 return bins\_df

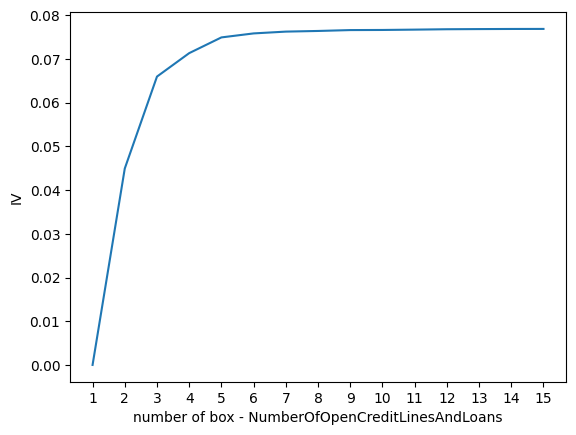
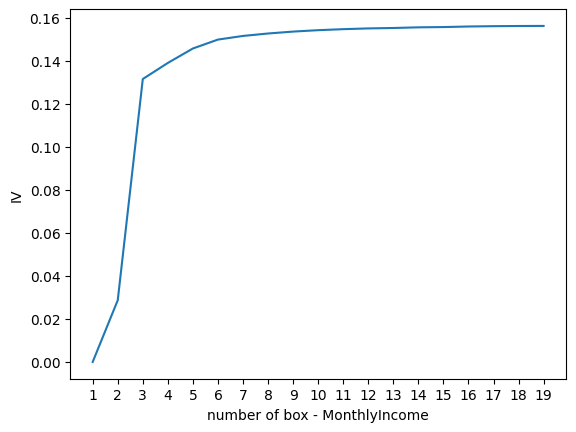
调用如下：

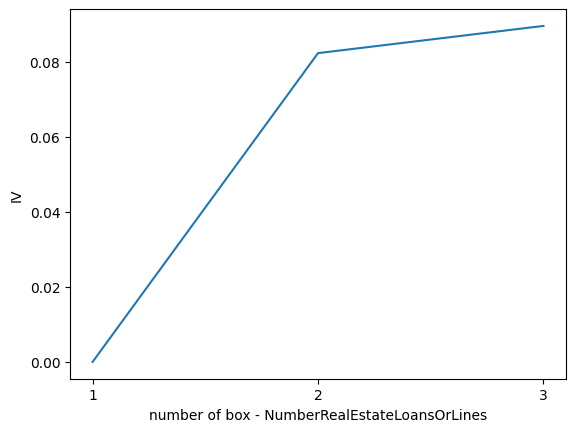
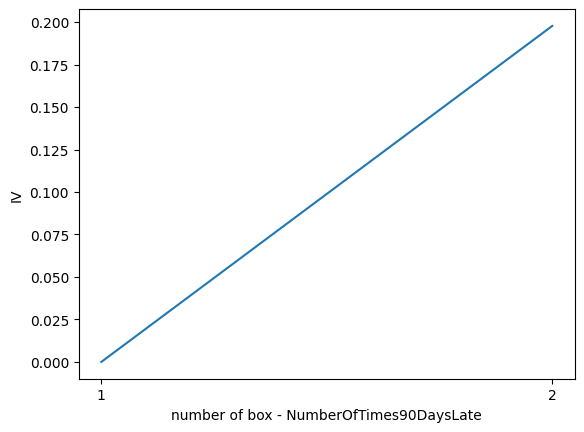
*# 对训练集查看分箱的最优数*col = [each for each in train\_data.columns if each not in [ 'SeriousDlqin2yrs','NumberOfDependents']]  
for columns in col:  
 graphforbestbin(train\_data,columns,"SeriousDlqin2yrs",n=1,q=20,graph=True)

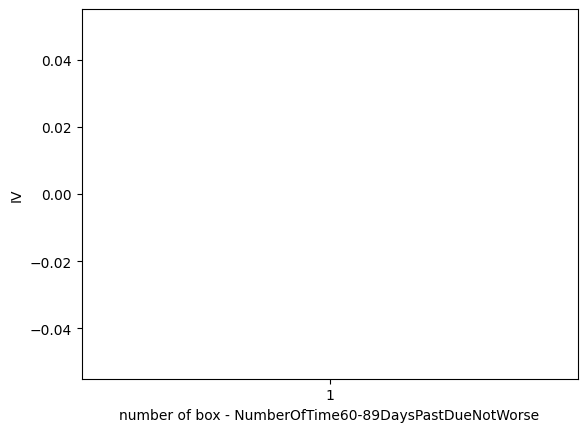
**输出的图像如下：**









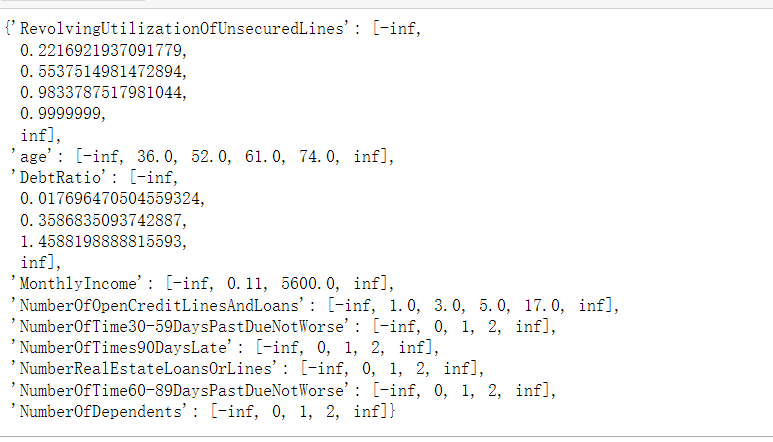
通过图像我们可以清晰的了解到各个IV值与分箱数的情况，那么基于这个图我们怎么确定理想的分箱数呢，理想的分箱数就是上述图像曲线的拐角点对应的x值，比如：RevolvingUtilizationOfUnsecuredLines分5箱就比较合适，没有拐点和只有一种趋势的图像看不出来分箱数，对于这种情况需要依靠人工去确定分箱数，接下来我们确定分箱数：

*# 确定的分箱自动处理*sure\_bins = {"RevolvingUtilizationOfUnsecuredLines":5,  
 "age":5,  
 "DebtRatio":4,  
 "MonthlyIncome":3,  
 "NumberOfOpenCreditLinesAndLoans":5  
 }  
*# 不确定的分箱单独处理*unsure\_bin = {"NumberOfTime30-59DaysPastDueNotWorse":[0,1,2,13],  
 "NumberOfTimes90DaysLate":[0,1,2,17],  
 "NumberRealEstateLoansOrLines":[0,1,2,54],  
 "NumberOfTime60-89DaysPastDueNotWorse":[0,1,2,9],  
 "NumberOfDependents":[0,1,2,3]  
 }

*# 设置分箱区间区间结果使用 np.inf替换最大值，用-np.inf替换最小值*hand\_bins = {k:[-np.inf,\*v[:-1],np.inf] for k,v in unsure\_bin.items()}

*# 训练数据集开始分箱  
# 字典存储分箱结果*bin\_of\_col\_train={}  
*# 生成确定分箱的分箱区间和分箱后的 IV 值*for col in sure\_bins:  
 bins\_df=graphforbestbin(train\_data,  
 col,  
 'SeriousDlqin2yrs',  
 n=sure\_bins[col],  
 q=20,  
 graph=False)  
 bins\_list=sorted(set(bins\_df["min"]).union(bins\_df["max"]))  
 *#保证区间覆盖使用 np.inf 替换最大值 -np.inf 替换最小值* bins\_list[0],bins\_list[-1]=-np.inf,np.inf  
 bin\_of\_col\_train[col]=bins\_list  
*#合并手动分箱数据*bin\_of\_col\_train.update(hand\_bins)  
bin\_of\_col\_train

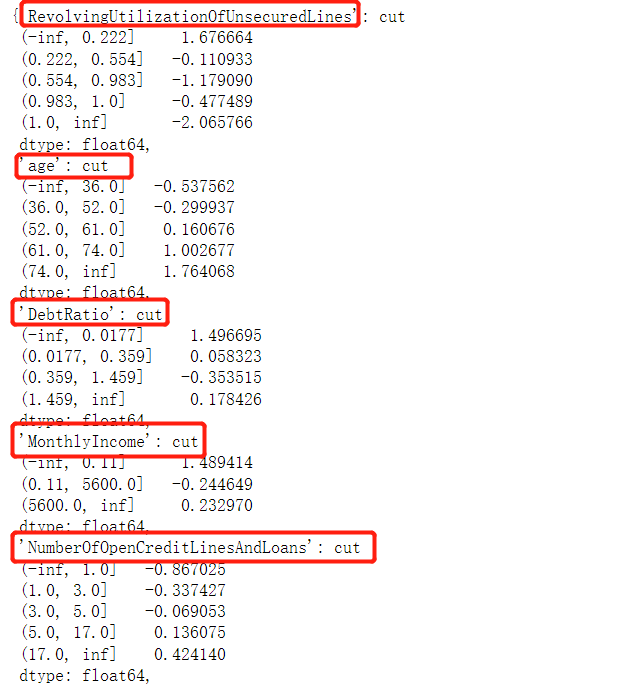
至此步骤的输出如下：



可以看出对于可以自动分箱(确定分箱的情况下)，我们已经得到他们的分箱情况，接下来我们需要计算WOE值，将WOE值作为特征值替代原来的数据值。

*# 获取WOE值*def get\_woe(df,col,y,bins):  
 df=df[[col,y]].copy()  
 df["cut"]=pd.cut(df[col],bins)  
 bins\_df=df.groupby("cut")[y].value\_counts().unstack()  
 woe=bins\_df["woe"]=np.log((bins\_df[0]/bins\_df[0].sum())/(bins\_df[1]/bins\_df[1].sum()))  
 return woe  
*#将所有特征的WOE存储到字典当中*woe={}  
for col in bin\_of\_col\_train:  
 woe[col]=get\_woe(train\_data,col,"SeriousDlqin2yrs",bin\_of\_col\_train[col])  
woe

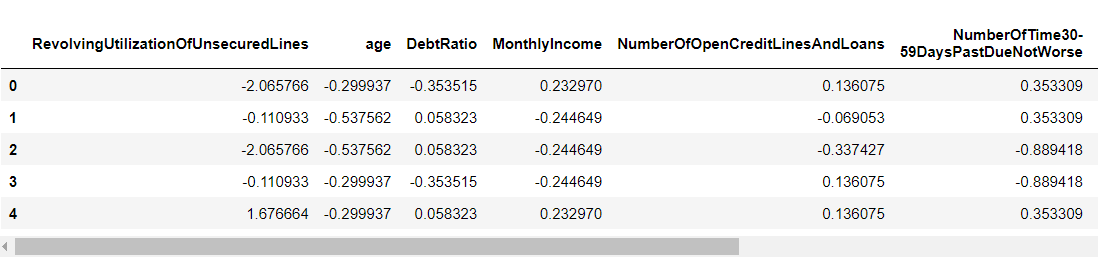
这样就得到所有特征的WOE值，我们这截取一些进行展示：



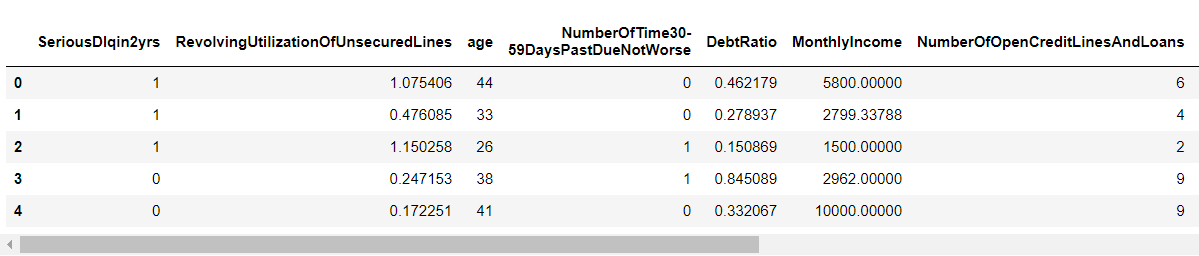
那么接下来的工作就比较简单了，就是将原来的数据基于上述字典进行WOE值得映射，为了不改变原始的数据集，我们新建一个数据集，并完成数据的映射：

*# 创建WOE dataframe*train\_data\_woe=pd.DataFrame(index=train\_data.index)  
*# 对所有特征进行映射：*for col in bin\_of\_col\_train:  
 train\_data\_woe[col]=pd.cut(train\_data[col],bin\_of\_col\_train[col]).map(woe[col])  
*# 添加标签*train\_data\_woe["SeriousDlqin2yrs"]=train\_data["SeriousDlqin2yrs"]  
train\_data\_woe.head()

经过WOE变换的数据如下，这个数据也就是我们进行建模所用的数据集：

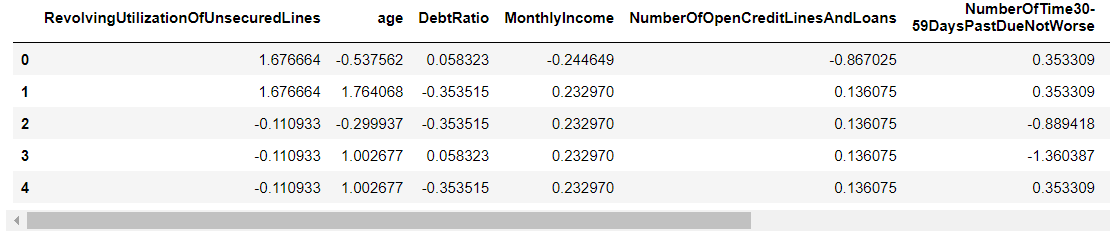


为了进行对比，我们也截图一下原始数据的前五行：



对照分箱后的WOE映射字典可以检查我们的分箱是不存在问题的，这样我们就完成了训练集数据的处理，接下来我们处理一下测试集数据集：

*# 测试集处理*test\_data\_woe = pd.DataFrame(index=test\_data.index)  
*# 训练集和测试集同一个分箱列*for col in bin\_of\_col\_train:  
 test\_data\_woe[col] = pd.cut(test\_data[col],bin\_of\_col\_train[col]).map(woe[col])  
test\_data\_woe["SeriousDlqin2yrs"] = test\_data["SeriousDlqin2yrs"]  
test\_data\_woe.head()



这样我们就完成了整个数据的处理，同样最好也保存一下数据集。

train\_data\_woe.to\_csv('train\_data\_woe.csv',index = False)  
test\_data\_woe.to\_csv('test\_data\_woe.csv',index = False)

**六、模型构建和调参**

在数据集准备好之后，我们就要开始建模了，这里使用逻辑回归模型，我们先整理一下数据集：

*# 模型输入的数据集构建*col = [each for each in train\_data\_woe.columns if each != "SeriousDlqin2yrs" ]  
X\_train = train\_data\_woe[col]  
y\_train=train\_data\_woe['SeriousDlqin2yrs']  
X\_test=test\_data\_woe[col]  
y\_test=test\_data\_woe['SeriousDlqin2yrs']  
print(X\_train.shape)  
print(y\_train.shape)  
print(X\_test.shape)  
print(y\_test.shape)

输出为：

(208938, 10)

(208938,)

(69646, 10)

(69646,)

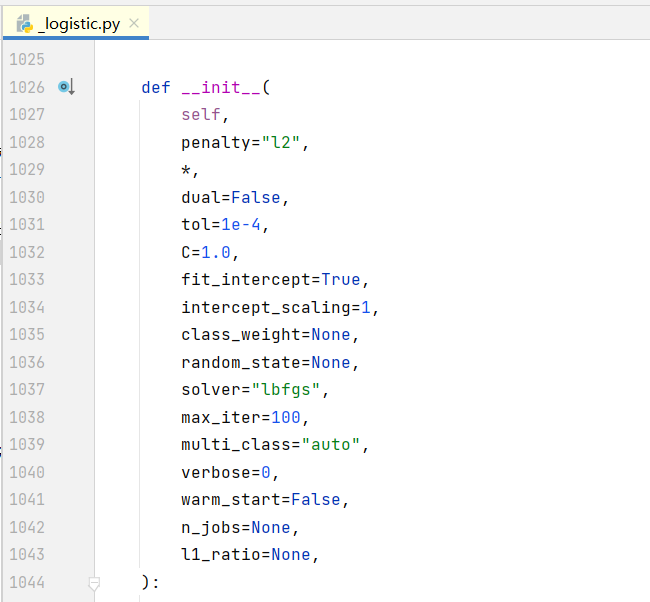
模型整体上是比较简单的，我们直接使用sklearn中的模块进行实现即可：

from sklearn.linear\_model import LogisticRegression  
lr = LogisticRegression().fit(X\_train, y\_train)  
*# 查看得分*print(lr.score(X\_test, y\_test))

# 输出结果为：

0.785127645521638

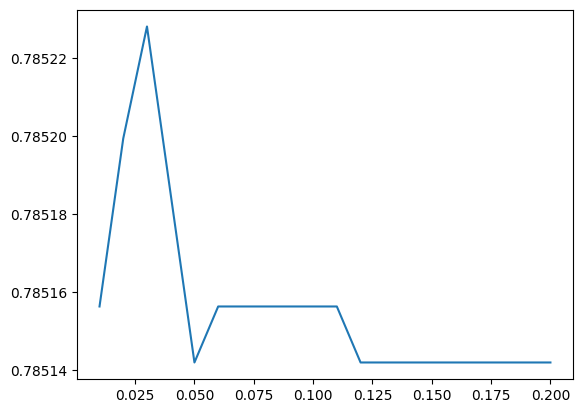
也就是说我们的逻辑回归模型的正确率为78.5%（即预测正确占比，好客户预测为好客户坏客户预测为坏客户的占比），大家可以查看一下机器学习中常见的评价指标的含义，应该就可以明白这个正确率的含义。注意的是我们在构建逻辑回归模型的时候是使用默认的参数的，默认参数我们如下：



接下来我们进行简单的参数调优：

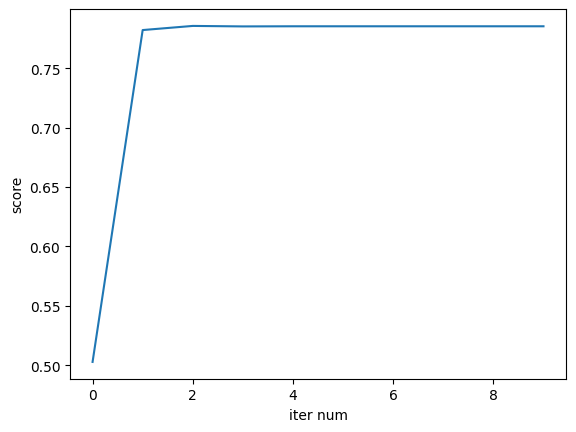
*# 参数的搜索范围*

c\_1 = np.linspace(0.01,0.2,20)  
score = []  
for i in c\_1:  
 lr = LR(solver='liblinear',C=i).fit(X\_train,y\_train)  
 score.append(lr.score(X\_test,y\_test))  
plt.figure()  
plt.plot(c\_1,score)  
plt.show()



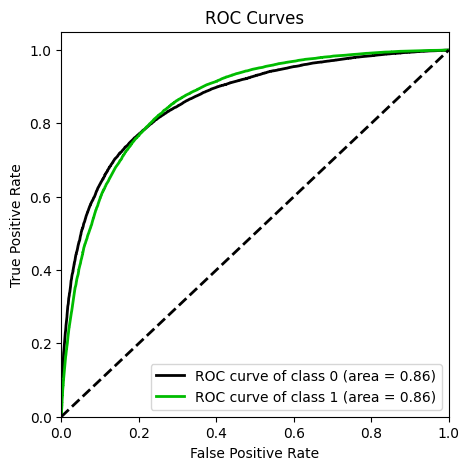
可以看出C的取值大概在0.025的时候准确率是最好的，因此在之后的优化当中我们选择此参数作为模型的参数，另外我们来看一下迭代次数参数max\_iter，这个参数默认是100的，我们来寻优这个参数：

score = []  
find\_list = list(range(10))  
for i in find\_list:  
 lr = LR(solver='liblinear',C=0.025,max\_iter=i).fit(X\_train,y\_train)  
 score.append(lr.score(X\_test,y\_test))  
plt.figure()  
plt.plot(find\_list,score)  
plt.xlabel("iter num")  
plt.ylabel("score")  
plt.show()



从结果图上可以看出，我们的模型迭代次数在max\_iter=2之后得分就不在有提升了，为此也没有必要进行100轮测训练，接下来我们进行ROC曲线的打印：

*# 模型评估*import scikitplot as skplt  
test\_proba\_df = pd.DataFrame(lr.predict\_proba(X\_test))  
skplt.metrics.plot\_roc(y\_test,  
 test\_proba\_df,  
 plot\_micro=False,  
 figsize=(5,5),  
 plot\_macro=False)



可以看出ROC曲线还是比较合适的。

**七、评分映射**

接下来的工作就是评分映射，就好比得到一个具体的支付宝芝麻信用分数一样。因为模型计算出的结果是概率值，即客户后续出现坏账的概率，取值范围是0到1，在实际应用的时候为了构建评分卡，我们需要对概率进行放大，将概率转化为分数。我们的映射公式为：



其中：

1. A，B为待确定的参数
2. log(odds)表示客户违约的可能性

那么确定这个表达式就需要两个条件去确定其中的参数A和B，我们可以根据两个假设条件来确定：

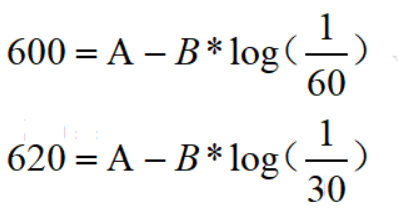
（1）某个特定的违约概率下的预期分值

（2）指定的违约概率翻倍的分数（PDO）

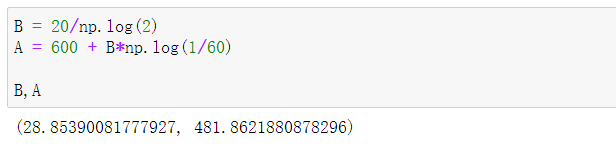
比如当几率为odds时，对应的Score为S1，当设定的几率为2odd时，对应的Score为S1-PDO。

POD一般是由业务人员或者根据一些经验拉起确定的比如20。

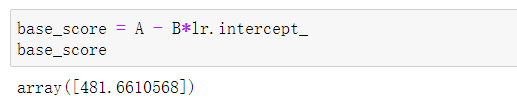
假设PDO=20且当odds = 1/60时设定的分数为600，当odds=1/30的时候得分则为620，据此得到下述表达式：



这样我们就可以直接来计算A和B的值。

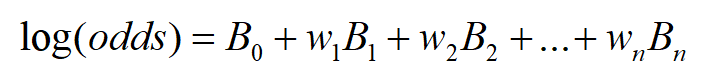


基于A,B的值我们就可以获取本案例中的基础分：



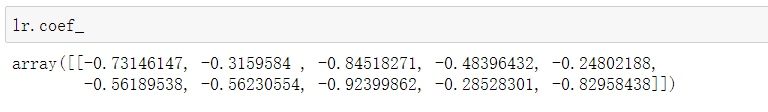
上述**lr.intercept\_**就是逻辑回归模型的截距，这个值也就等于





**其中w为某个样本对应的woe编码的值**，可以看出一个样本的最终得分和A,B的值，逻辑回归系数以及变量对应的woe编码相关。

我们先看一下LR模型中各个变量的权重：



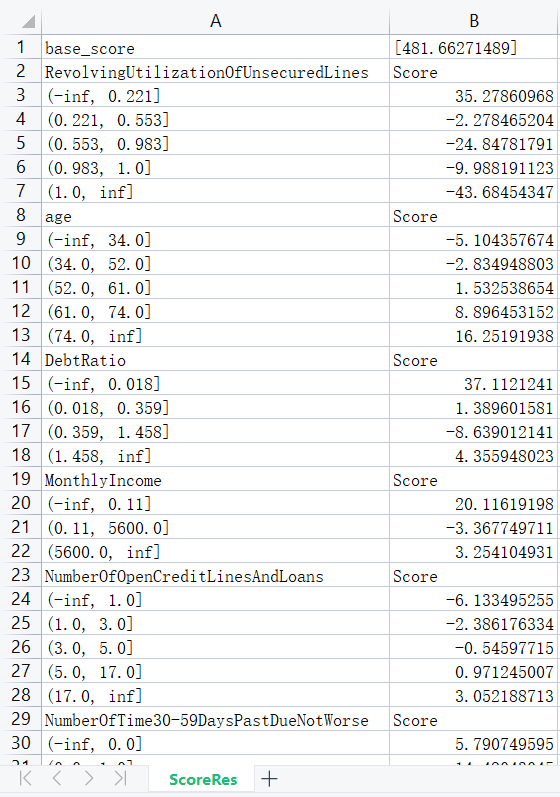
接下来进行特征RevolvingUtilizationOfUnsecuredLines的实际分数映射：



接着我们将上述所有的特征和相应的映射分数写入文件中：

*# 写入所有特征的评分卡内容*file = "ScoreRes.csv"  
with open(file,"w") as f:  
 f.write("base\_score,{}\n".format(base\_score))  
for i,col in enumerate(train\_data.columns):  
 score = woe[col] \* (-B\*lr.coef\_[0][i])  
 score.name = "Score"  
 score.index.name = col  
 score.to\_csv(file,header=True,mode="a")

这样我们就得到一张评分卡：



至此我们的整个生成评分卡的任务就完成了，在实际建模的过程中为了使用模型，可以将模型保存起来并在需要的时候进行加载和预测，我们直接来看代码：

*# 保存训练完结束的模型*import joblib  
joblib.dump(lr, "ScoreCard.pkl")  
  
*# 模型加载和预测  
# 加载和预测*new\_lr = joblib.load("ScoreCard.pkl")  
  
*# proba = new\_lr.predict\_proba(X\_test.loc[0, :].values.reshape(1, -1))  
# proba = new\_lr.predict\_proba([X\_test.loc[0, :]])*proba = new\_lr.predict\_proba(pd.DataFrame([X\_test.loc[0, :]]))  
proba\_class = new\_lr.predict(pd.DataFrame([X\_test.loc[0, :]]))  
*# 预测值*print(proba)  
*# 预测所属类别*print(proba\_class)

# 输出为  
[[0.78863024 0.21136976]]

[0]

new\_lr.predict()返回预测标签

new\_lr.predict()返回预测属于某标签的概率

至此，整个流程就结束了，一般来说特征工程是整个建模过程中最为费时的工作，不仅要考虑业务的需求方的特征，还要考虑一些特征数据自身数据的重要性，另外数据分箱也是比较耗时的过程，不过现在有一些工具包可以直接进行分箱的操作，还是很方便的。对于模型的评估我们可以通过机器学习中的一些常用的指标进行评估，比如AUC和ROC评判模型效果。

**八、参考文档**

https://blog.csdn.net/weixin\_51589123/article/details/120742440

https://www.jianshu.com/p/22d806b892db

https://mp.weixin.qq.com/s/38ZA3PZS\_TzlVTVmcM3\_wg

https://zhuanlan.zhihu.com/p/419822157

|  |  |
| --- | --- |
| **字段名称** | **字段含义** |
| SeriousDlqin2yrs | 出现严重的逾期行为（定义好坏标签） |
| RevolvingUtilizationOfUnsecuredLines | 贷款以及信用卡可用额度与总额比例 |
| age | 借款人年龄 |
| NumberOfTime30-59DaysPastDueNotWorse | 过去两年内出现30-59天逾期但是没有发展得更坏的次数 |
| DebtRatio | 每月偿还债务，生活费等除以月总收入 |
| MonthlyIncome | 月收入 |
| NumberOfOpenCreditLinesAndLoans | 开放式贷款和信贷数量 |
| NumberOfTimes90DaysLate | 过去两年内出现90天逾期或者更坏的次数 |
| NumberRealEstateLoansOrLines | 抵押贷款和房地产贷款数量，包括房屋净值信贷额度 |
| NumberOfTime60-89DaysPastDueNotWorse | 过去两年内出现60-89天逾期但是没有发展得更坏的次数 |
| NumberOfDependents | 家庭人员人数 |