信用违约预测

|  |  |
| --- | --- |
| 姓名 | 学号 |
| 余若涵 | 3180105412 |
| 范源颢 | 3180103574 |

格式：小四 1.25倍行距 10页以上 （意思是最后需要转成word格式？）

# 一、研究背景与意义

TODO

估计客户的信用评级是银行风险管控非常重要的组成部分，银行借给债务人的借款如果无法按时收回，银行的资产将面临不必要的风险，同时债务人的信用也会受到影响。为此，银行需要在借款之前根据数据估计客户的还款能力，评价信用风险等级。

# 二、国内外研究现状与存在问题

TODO

# 三、研究目标与研究内容

TODO

在本项目中，我们的目的是：根据客户的历史表现判断客户是否有能力及时还款。

# 四、数据集分析

## **4.1** 数据集基本信息

本项目中，我们使用的是由Home Credit公司提供的贷款信息数据集。数据集包含application\_train.csv和application\_test.csv两个文件。训练集共提供307511条记录，每条记录包含121个特征和1个Target 列；测试集包含48744条记录。

数据集中的每条记录都表示了一次贷款，Target的0/1值反映了借款人是否按时还款。121个特征中，有

106个数值特征和16个标称特征，这些特征可分为以下四类(只列出部分)：

1. 借款人基本信息

CODE\_GENDER - 性别，

DAYS\_BIRTH - 客户年龄，

CNT\_CHILDREN - 子女数，

CNT\_FAM\_MEMBERS - 家人数，

OCCUPATION\_TYPE - 职业，

……

1. 借款人经济背景

FLAG\_OWN\_CAR - 是否拥有轿车，

FLAG\_OWN\_REALTY - 是否拥有不动产，

AMT\_INCOME\_TOTAL - 借款人总收入，

APARTMENT\_AVG - 住房评分，

……

1. 本次贷款相关信息

AMT\_CREDIT - 信用额度，

AMT\_ANNUITY - 贷款年金，

WEEKDAY\_APPR\_PROCESS\_START - 申请贷款的星期，

HOUR\_APPR\_PROCESS\_START - 申请贷款的小时，

……

1. 其他信息

FLAG\_MOBIL - 是否提供手机号码，

FLAG\_CONT\_MOBILE - 手机是否可打通，

REGION\_RATING\_CLIENT - 所在地区评分，

EXT\_SOURCE\_1/EXT\_SOURCE\_2/EXT\_SOURCE\_3 - 通过其他信息计算的该客户评分，

……

## **4.2** 可视化分析

以下可视化分析针对未经预处理的训练集aplication\_train.csv

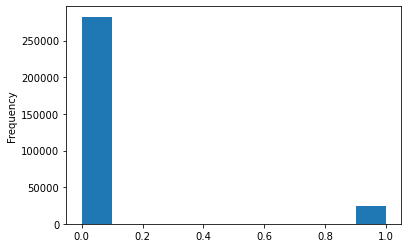
### 4.2.1 Target列的分布

app\_train**[**'TARGET'**].**value\_counts**()**

app\_train**[**'TARGET'**].**astype**(int).**plot**.**hist**()**

输出：

1. 282686
2. 24825



从上图可以看出，训练数据集中的违约记录数远大于未违约记录数，整个训练集是极度不平衡的，这种情况下，模型会花费更多的时间和“精力”去拟合未违约记录(target为0)，所以为了提供模型的效率，在之后我们将采取欠采样或过采样的方法来平衡数据集。

### 4.2.2 特征与Target的相关性

1. 计算特征与**Target**的相关性

# Calculate the correlations and sort

correlations **=** app\_train**.**corr**()[**'TARGET'**]**

correlations **=** correlations**.**drop**([**"TARGET"**]).abs().**sort\_values**()**

1. **10**个相关性最小的特征

# 10 least relevant features

correlations**.**head**(**10**)**

输出：

FLAG\_DOCUMENT\_20 0.000215

FLAG\_DOCUMENT\_5 0.000316

FLAG\_CONT\_MOBILE 0.000370

FLAG\_MOBIL 0.000534

FLAG\_DOCUMENT\_12 0.000756

AMT\_REQ\_CREDIT\_BUREAU\_WEEK 0.000788

AMT\_REQ\_CREDIT\_BUREAU\_HOUR 0.000930

FLAG\_DOCUMENT\_19 0.001358

FLAG\_DOCUMENT\_10 0.001414

FLAG\_DOCUMENT\_7 0.001520

Name: TARGET, dtype: float64

上面输出结果中的FLAG\_DOCUMENT\_X特征表示的是在贷款时是否提交某份文件，这类特征有20个，大部分相关性都很低。FLAG\_MOBIL和FLAG\_CONT\_MOBILE表示的分别是客户是否拥有移动电话、电话是否可打通，AMT\_REQ特征表示的是申请贷款前客户向信贷局进行查询的次数。

可以看出，这些特征的与Target列的相关性都在万分之一数量级，可以归为very weak级别。在之后的特征工程中我们将会试试它们是否可以衍生出足够好的新特征，或者直接删除这些特征，以减少运算量并提高预测的准确度。

**2) 20**个相关性最大的特征

# Display 20 most relevant featrues

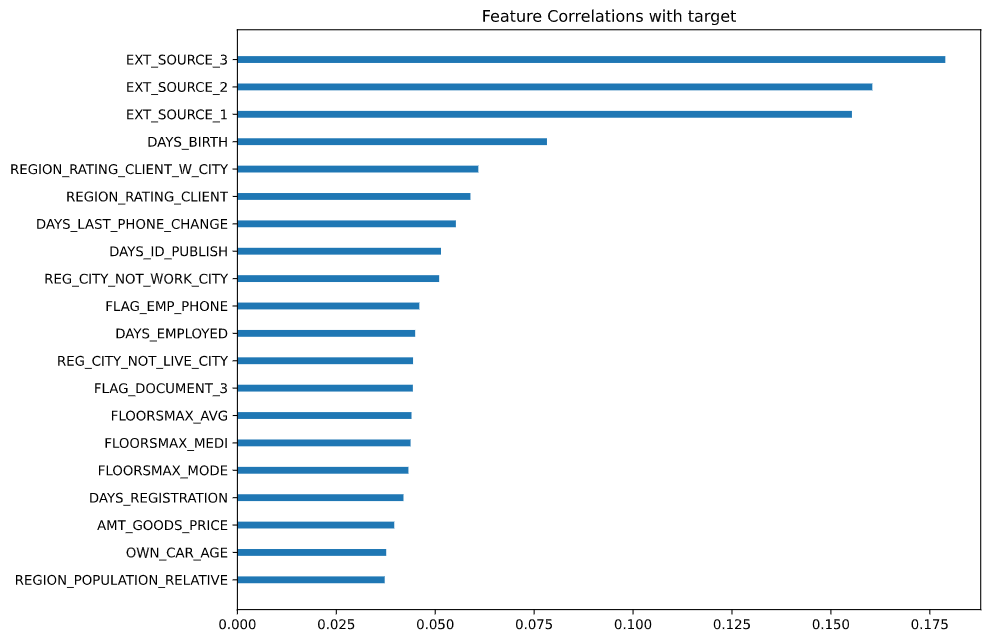
corrs **=** correlations**.**tail**(**20**)**

plt**.**figure**(**figsize **=** **(**10**,** 8**))**

plt**.**bar**(** x**=**0**,** bottom**=**corrs**.**index**.**astype**(str),**

height**=**0.25**,** width**=abs(**corrs**.**values**),** orientation**=** "horizontal"**)**

plt**.**title**(**'Feature Correlations with target'**)**



上图中（相关性已取绝对值）排在前三的EXT\_SOURCE特征是专业人士依据外部数据对该客户进行的评分，值域为[0, 1]，它们的相关性都超过了0.15；DAY\_BIRTH表示的是客户的年龄；两个

REGION\_RATING特征是对客户所在地的评分，分为三档，用{1, 2, 3}表示。接下来我们将对这些特征进行可视化分析。

### 4.2.3 客户年龄分布

plt**.**figure**(**figsize **=** **(**10**,** 8**))**

# KDE plot of loans that were repaid on time

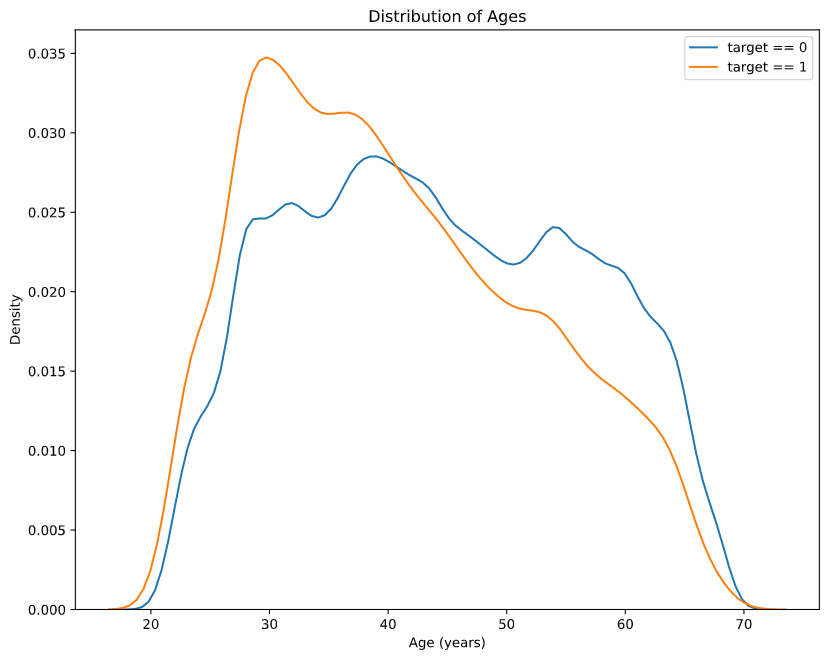
sns**.**kdeplot**(**app\_train**.**loc**[**app\_train**[**'TARGET'**]** **==** 0**,** 'DAYS\_BIRTH'**]** **/** **-**365**,** label **=** 'target == 0'**)**

# KDE plot of loans which were not repaid on time

sns**.**kdeplot**(**app\_train**.**loc**[**app\_train**[**'TARGET'**]** **==** 1**,** 'DAYS\_BIRTH'**]** **/** **-**365**,** label **=** 'target == 1'**)**

# Labeling of plot

plt**.**xlabel**(**'Age (years)'**);** plt**.**ylabel**(**'Density'**);** plt**.**title**(**'Distribution of Ages'**);**



上图中，橘线展示的是违约用户的年龄分布，蓝线为未违约用户的年龄分布。从总可以看出，年轻用户

([25, 40]年龄区间内)更容易违约，而年纪较大的用户([50, 65]年轻区间内)则更可能按时付清欠款。

为了更好地分析客户年龄与违约的关系，我们以5年为一个区间宽度进行分箱处理，并用柱状图展示各个区间内的违约率。如下所示。

# Age information into a separate dataframe

age\_data **=** app\_train**[[**'TARGET'**,** 'DAYS\_BIRTH'**]]**

age\_data**[**'YEARS\_BIRTH'**]** **=** age\_data**[**'DAYS\_BIRTH'**]** **/** **-**365

# Bin the age data

age\_data**[**'YEARS\_BINNED'**]** **=** pd**.**cut**(**age\_data**[**'YEARS\_BIRTH'**],** bins **=** np**.**linspace**(**20**,** 70**,** num **=** 11**))**

age\_data**.**head**(**10**)**

# Group by the bin and calculate averages

age\_groups **=** age\_data**.**groupby**(**'YEARS\_BINNED'**).**mean**()**

plt**.**figure**(**figsize **=** **(**8**,** 8**))**

# Graph the age bins and the average of the target as a bar plot

plt**.**bar**(**age\_groups**.**index**.**astype**(str),** 100 **\*** age\_groups**[**'TARGET'**])**

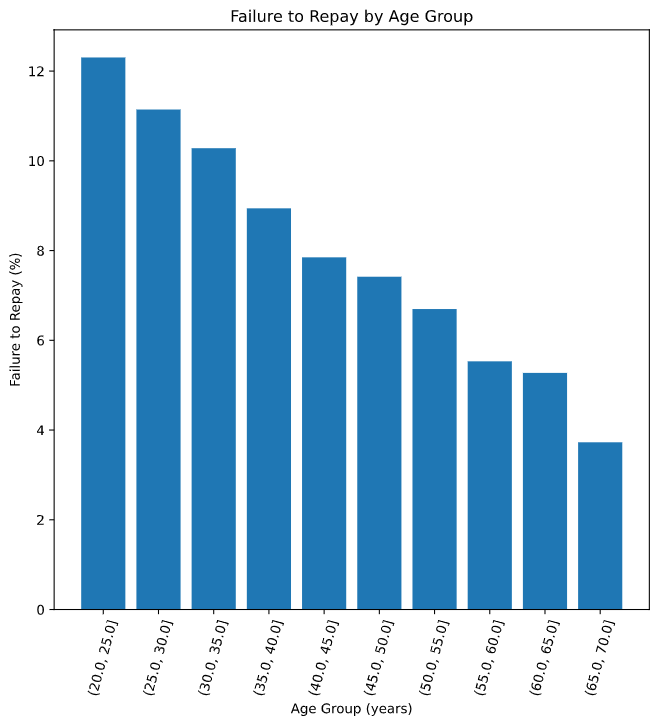
# Plot labeling

plt**.**xticks**(**rotation **=** 75**);**

plt**.**xlabel**(**'Age Group (years)'**);**

plt**.**ylabel**(**'Failure to Repay (%)'**)**

plt**.**title**(**'Failure to Repay by Age Group'**);**



从图中可以看出，随着年龄的增高，违约率一直在下降。年纪最小的三个群体有超过10%的违约率，而年纪最大群体的违约率则低于5%. 这提示我们，年龄将会是一个非常重要的特征，在之后的特征工程中，我们也许可以基于年龄衍生出新的重要特征。

### 4.2.4 附加特征(EXT\_SOURCE)的分布

上文4.2.2的分析显示，三个EXT\_SOURCE列与Target列有很大的相关性。下面探究它们的分布。

plt**.**figure**(**figsize **=** **(**10**,** 12**))**

# iterate through the sources

**for** i**,** source **in** **enumerate([**'EXT\_SOURCE\_1'**,** 'EXT\_SOURCE\_2'**,** 'EXT\_SOURCE\_3'**]):**

# create a new subplot for each source

plt**.**subplot**(**3**,** 1**,** i **+** 1**)**

# plot repaid loans

sns**.**kdeplot**(**app\_train**.**loc**[**app\_train**[**'TARGET'**]** **==** 0**,** source**],** label **=** 'target == 0'**)**

# plot loans that were not repaid

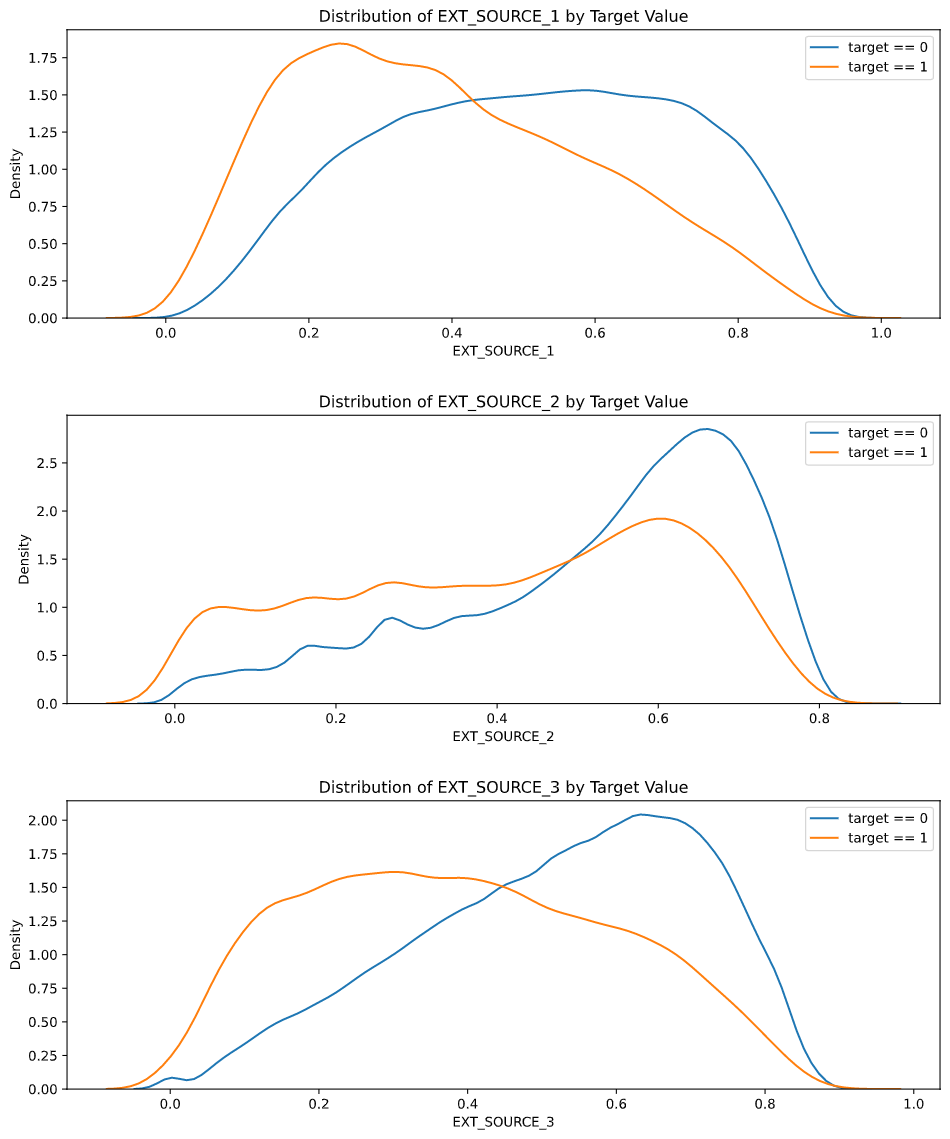
sns**.**kdeplot**(**app\_train**.**loc**[**app\_train**[**'TARGET'**]** **==** 1**,** source**],** label **=** 'target == 1'**)**

# Label the plots

plt**.**title**(**'Distribution of %s by Target Value' **%** source**)**

plt**.**xlabel**(**'%s' **%** source**);** plt**.**ylabel**(**'Density'**);**

plt**.**tight\_layout**(**h\_pad **=** 2.5**)**



可以看出，在以上三张图中都能很好地区分违约与未违约这两条曲线。其中EXT\_SOURCE\_3的效果最好，当其值较小时，有更大概率发生违约。而EXT\_SOURCE\_2虽然也能很好区分两条曲线，但是两曲线的整体趋势非常相似，所以也许它在模型的效果会不如另两个特征。

### 4.2.5 地区评分(REGION\_RATING)的分布

**for** i**,** source **in** **enumerate([**'REGION\_RATING\_CLIENT'**,** 'REGION\_RATING\_CLIENT\_W\_CITY'**]):**

plt**.**figure**(**figsize **=** **(**5**,** 3**))**

tot **=** app\_train**[**app\_train**[**'TARGET'**]!=-**1**].**groupby**(**source**).**count**()[**'TARGET'**]**

default **=** app\_train**[**app\_train**[**'TARGET'**]==**1**].**groupby**(**source**).**count**()[**'TARGET'**]**

plt**.**bar**(**tot**.**index**,** default **/** tot**,** width**=**0.35**,** color**=**'lightskyblue'**)**

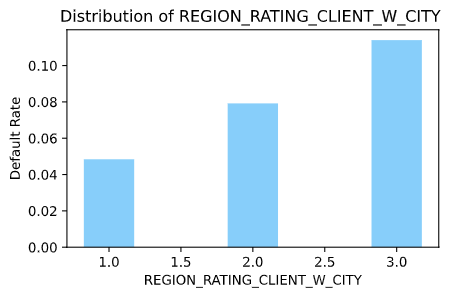
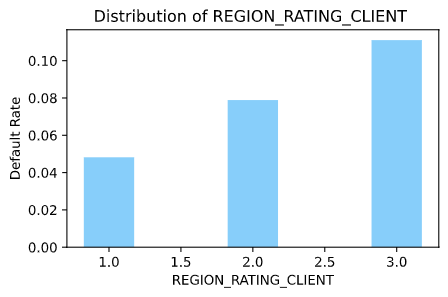
# Label the plots

plt**.**title**(**'Distribution of %s ' **%** source**)**

plt**.**xlabel**(**'%s' **%** source**);**

plt**.**ylabel**(**'Default Rate'**);**

plt**.**show**()**



图中柱状图表示的是违约率即违约客户占总客户的比例。两个特征都呈现出一样的趋势，即评分越高则违约率越大。可以看出这两张柱状图的趋势和数值都非常相似，所以我们推测这两个特征的相关性应该非常高，接下来进行具体分析。

**4.2.6** 特征间的相关度

由于本数据集特征过多，不适合对所有特征进行相关度可视化，所以这一部分我们只选取了上面讨论过的六个特征进行分析。

# Extract the EXT\_SOURCE variables and show correlations

ext\_data **=** app\_train**[[**'TARGET'**,** 'EXT\_SOURCE\_1'**,** 'EXT\_SOURCE\_2'**,** 'EXT\_SOURCE\_3'**,** 'DAYS\_BIRTH'**,** 'R

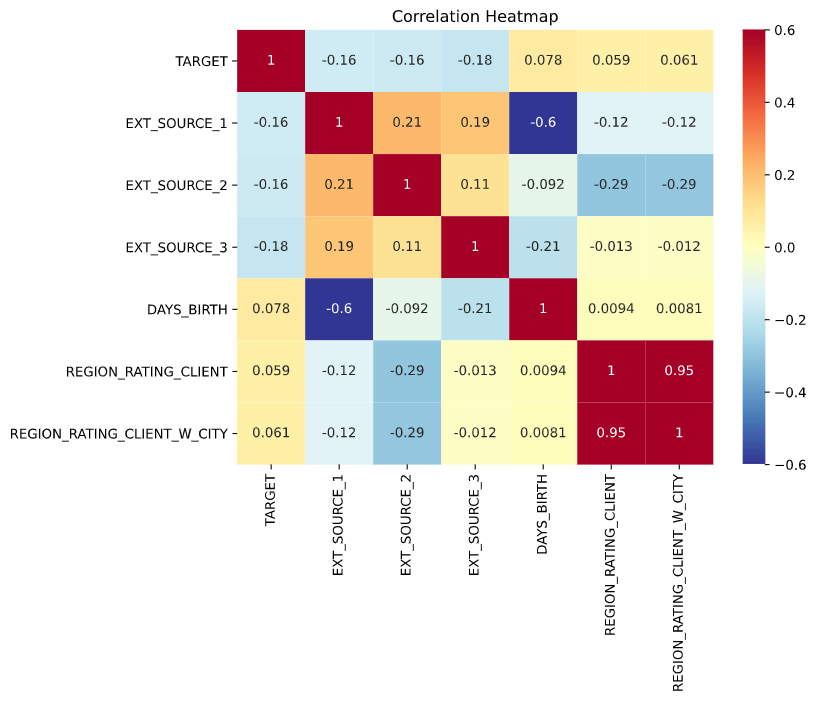
ext\_data\_corrs **=** ext\_data**.**corr**()**

# Heatmap of correlations

plt**.**figure**(**figsize **=** **(**6**,** 6**))**

sns**.**heatmap**(**ext\_data\_corrs**,** cmap **=** plt**.**cm**.**RdYlBu\_r**,** vmin **=** **-**0.6**,** annot **=** **True,** vmax **=** 0.6**)**

plt**.**title**(**'Correlation Heatmap'**);**



上面的热力图中，颜色越深表示二者之间相关度越高。可以看出两个REGION\_RATING类特征有高达 0.95的相关性，这两个特征都是对客户所在地区的评分，二者的差异仅在于：客户所在城市是否在考虑范围内，于是我们可以认为，它们包含的信息是基本一致的，在特征工程中我们会考虑删除这两个特征中的一个，以提高效率。此外，EXT\_SOURCE\_1和年龄的相关度也有0.6，这说明年龄可能是影响 EXT\_SOURCE\_1得分的一个重要因素。

## **4.3** 数据预处理

TODO

编码

缺失

异常

# 五、特征工程

本项目中的特征工程，主要分为新特征衍生和特征降维两部分。在新特征衍生部分，我们将尝试使用多项式特征构造法、专业知识特征构造法以及特征自动生成工具来衍生出新特征，并保留其中效果较好的参与建模。在特征降维中，我们会去除一些贡献度较小的特征：如缺失值过多的特征、与Target相关度过小的特征、共线特征等，以提高模型的效率。

## **5.1** 新特征衍生

### 5.1.1 多项式特征

多项式特征是一种非常简单的特征构造方法，其衍生的新特征为原有特征的n次多项式，形如f1 \* f2 ^

2，f3 ^ 3。

实操中，我们参考了[GitHub上的一篇文章。](https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-engineering.html)

Dcikit-Learn直接为我们提供了一个PolynomialFeatures的类来构造多项式特征，创建该类的实例时需要传入一个参数degree即为多项式的最高幂次。为了避免模型出现过拟合，degree不应取得过高。项目中我们将degree设为3，使用上文分析过的年龄、EXT\_SOURCE作为原始特征，尝试构造多项式特征。具体操作如下。

**from** sklearn**.**preprocessing **import** PolynomialFeatures

# Select features

poly\_features **=** app\_train**[[**'EXT\_SOURCE\_1'**,** 'EXT\_SOURCE\_2'**,** 'EXT\_SOURCE\_3'**,** 'DAYS\_BIRTH'**,** 'TARGET

# Create the polynomial object with specified degree

poly\_transformer **=** PolynomialFeatures**(**degree **=** 3**)**

# Train the polynomial features

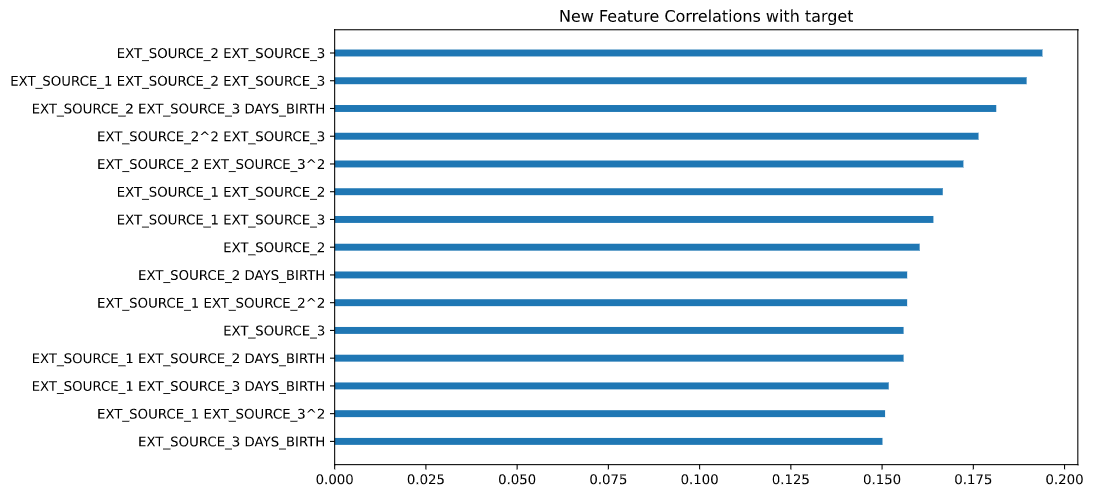
poly\_transformer**.**fit**(**poly\_features**)**

# Transform the features

poly\_features **=** poly\_transformer**.**transform**(**poly\_features**)**

poly\_features\_test **=** poly\_transformer**.**transform**(**poly\_features\_test**)**

计算新特征与Target的相关性并进行可视化，结果如下。（相关性大小已取绝对值）



可以看出，许多特征的相关性超过了原有特征。排在前几的特征均为EXT\_SOURCE特征的乘积，新特

征EXT\_SOUCE\_2 \* EXT\_SOURCE\_3 的相关性达到了0.19，相比原特征EXT\_SOUCE\_2的0.16略有提升。但也需注意，EXT\_SOURCE\_2 在于 DAYS\_BIRTH相乘后，新特征相关性反而略有下降。

在建模前，我们会选择一部分新特征加入到数据集中，并尝试不同的特征组合，根据效果决定最终选择。

### 5.1.2 专业知识特征

专业知识特征是利用金融知识和经验，人为地构造一些我们主观上认为可能会影响客户违约情况的新特征。

由于我们的金融知识有限，所以这一部分我们参考了[Kaggle上的一篇文章](https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction)，构造出以下四个特征。

* CREDIT\_INCOME\_PERCENT ：信用额度(credit)占客户收入(income)的百分比
* ANNUITY\_INCOME\_PERCENT ：贷款年金(annuity)占客户收入(income)的百分比
* ANNUITY\_CREDIT\_PERCENT ：贷款年金(annuity)占信用额度(credit)的百分比
* DAYS\_EMPLOYED\_PERCENT ：客户工作天数(days\_employed)占年龄(days\_birth)的百分比

app\_train\_domain **=** app\_train**.**copy**()**

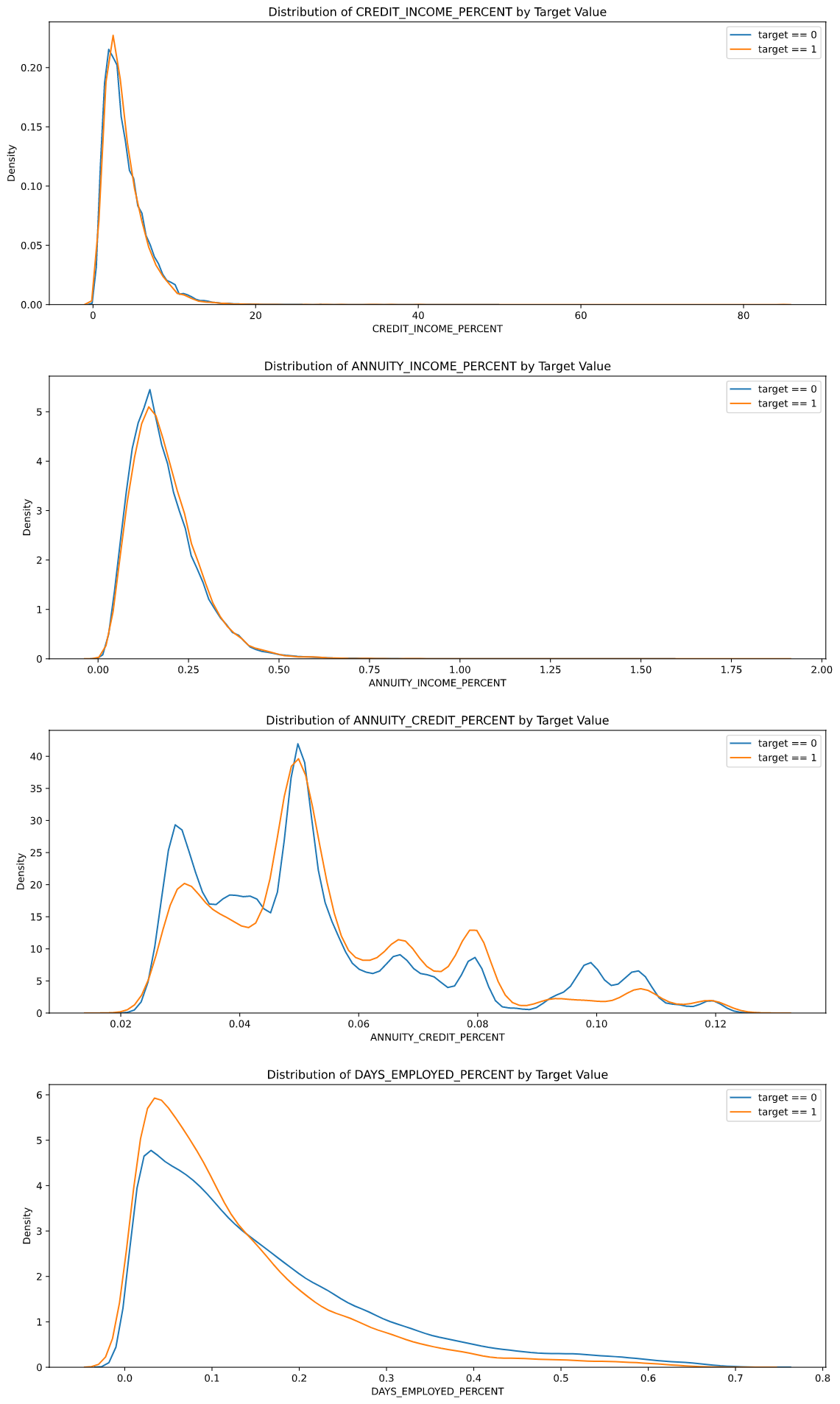
app\_train\_domain**[**'CREDIT\_INCOME\_PERCENT'**]** **=** app\_train\_domain**[**'AMT\_CREDIT'**]** **/** app\_train\_domain**[**'AMT\_INCOME\_TOTAL'**]**

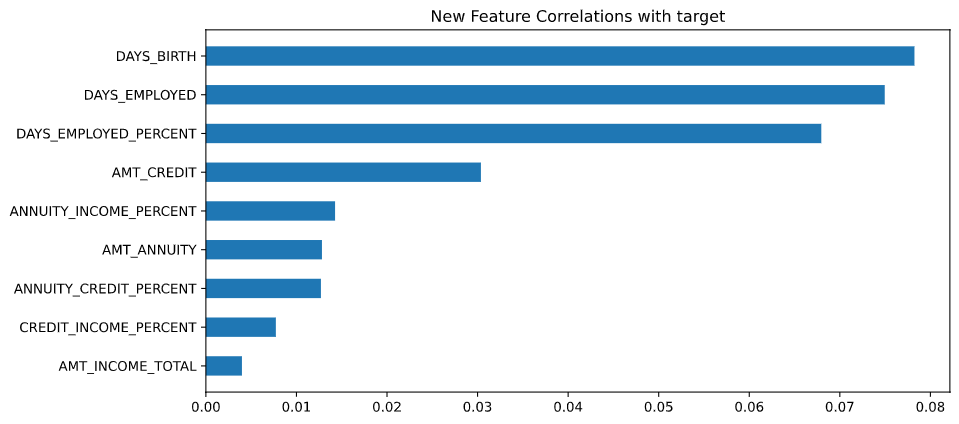
app\_train\_domain**[**'ANNUITY\_INCOME\_PERCENT'**]** **=** app\_train\_domain**[**'AMT\_ANNUITY'**]** **/** app\_train\_domain**[**'AMT\_INCOME\_TOTAL'**]**

app\_train\_domain**[**'ANNUITY\_CREDIT\_PERCENT'**]** **=** app\_train\_domain**[**'AMT\_ANNUITY'**]** **/** app\_train\_domain**[**'AMT\_CREDIT'**]**

app\_train\_domain**[**'DAYS\_EMPLOYED\_PERCENT'**]** **=** app\_train\_domain**[**'DAYS\_EMPLOYED'**]** **/** app\_train\_domain**[**'DAYS\_BIRTH'**]**

对上面四个特征进行可视化分析，得到分布图和相关性柱状图如下：





从分布图来看，前两个特征(CREDIT\_INCOME, ANNUITY\_INCOME)的违约未违约曲线拟合度非常高，所以效果应该不是很好。后两个特征(ANNUITY\_CREDIT, DAYS\_EMPLOYED)的曲线有较明显的差异，但是不同客户的曲线趋势仍是一致的。从相关性来看，各个新特征的相关性都较高，或基于原特征有部分提升。至于这些特征对我们的模型具体有多大的贡献，还需要在建模时进行实验。

### 5.1.3 特征生成工具 Feature Tool

## **5.2** 特征降维

### 5.2.1 共线特征去除

### 5.2.2 缺失值过多特征去除

### 5.3.3 重要性较低特征去除

# 六、研究方法与模型思路

这一章偏重于理论介绍

总体思路：我们这里希望使用机器学习的方法，通过学习训练集（application\_train.csv）中，自变量（借款人信息）和因变量（违约与否）的关系，拟合出一个可能的学习模型，然后依此对于测试集（application\_test.csv，只包含借款人信息）中的自变量估计出一个相对合理的因变量的值（即借款人的违约概率），由于我们的application\_test.csv中不包含真正的因变量值，所以仅仅根据测试集我们没法评估这个学习模型的好坏，为此，我们可以拆分我们的训练集（application\_train.csv），即，将训练集中70%的记录用于训练学习模型，后30%的记录用于测试模型，这样训练集的后30%的记录是已经知道真实的违约情况的，所以我们可以比较预测的违约情况和真实的违约情况，计算二者的差距，从而判别这个学习模型的好坏。

## 6.1 逻辑斯蒂回归（Logistic Regression）

虽然名字中有回归，但是模型最初是为了解决二分类问题。

线性回归模型帮助我们用最简单的线性方程实现了对数据的拟合，但只实现了回归而无法进行分类，因此LR就是在线性回归的基础上，构造的一种分类模型。对线性模型进行分类。在本题中，我们将违约与否分别标记为零和一两类，于是其实上我们可以将自变量的值首先按照线性的关系映射到一个中间变量x上，之后利用sigmoid函数

将中间变量映射到一个0-1之间的数值，作为违约概率，以此和违约与否的实际情况比较，即可得到学习模型和实际情况的差距（损失函数），之后我们利用一定的算法，调节自变量到中间变量的线性映射的参数，尽量减小这个损失函数的大小，就可以得到一个相对合理的学习模型，以此来进行预测。

### 6.1.1 sigmoid函数

如果要把一个实数域上的变量映射到两个类别上，似乎使用阶跃函数（也即符号函数sgn(x)）更为恰当，

但是在机器学习中，我们会发现sigmoid函数比起阶跃函数要有更加好的性质：sigmoid是连续无限阶可导的，而阶跃函数在x = 0时有一个明显的不连续点，不连续必定不可导，这样阶跃函数的损失函数也不可导，于是非常不利于我们优化自变量到中间变量的参数，而sigmoid函数很好的规避了这个问题。

### 6.1.2 sigmoid函数的损失函数

逻辑斯蒂回归问题的损失函数有如下形式：

其中, Y 为输出变量, X为输入变量, L 为损失函数. N为输入样本量, M为可能的类别数, yij 是一个二值指标, 表示类别 j 是否是输入实例 xi 的真实类别. pij 为模型或分类器预测输入实例 xi 属于类别 j 的概率

显然，这个公式在本题中可以进一步化简，因为本题中，可能的类别数为2，也就是最多只有两种可能的类，那么M = 2时，我们有：

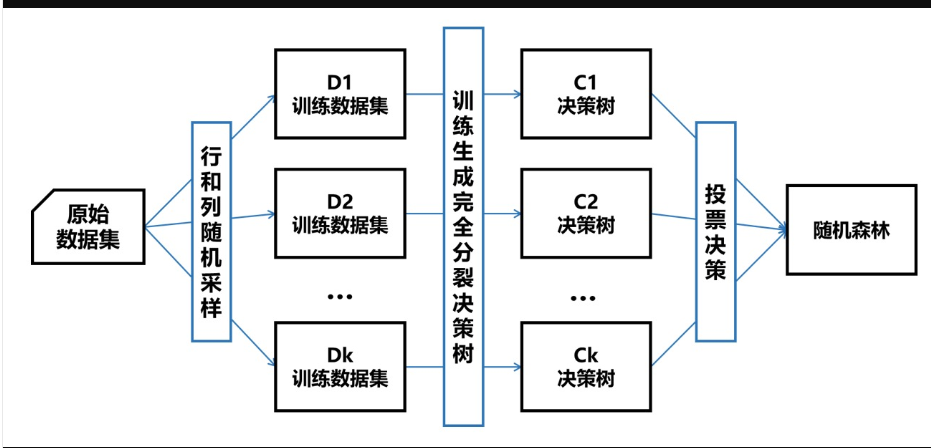
　这时, yi 为输入实例 xi 的真实类别, pi 为预测输入实例 xi 属于类别 1 的概率. 对所有样本的对数损失表示对每个样本的对数损失的平均值, 对于完美的分类器, 对数损失为 0 .

### 6.1.3 实现的思路

我们知道在python的机器学习包中，有完整的逻辑斯蒂回归的学习模块，在sklearn（scikit-learn）之下，为此，我们可以借助python实现上文所述的机器学习方法，不过需要注意的是，逻辑斯蒂回归要求真实值不能为空，因此，我们在数据清洗的时候需要更深一层的清洗，也就是需要将所有NaN进行合理的填充，我们采用中位数填充法，不使用平均值填充是为了避免出现极值单方面影响均值的情况，填充完成后，我们使用归一化方法将所有的数据都压缩到0-1之间。

具体的代码和实验结果见第七章。

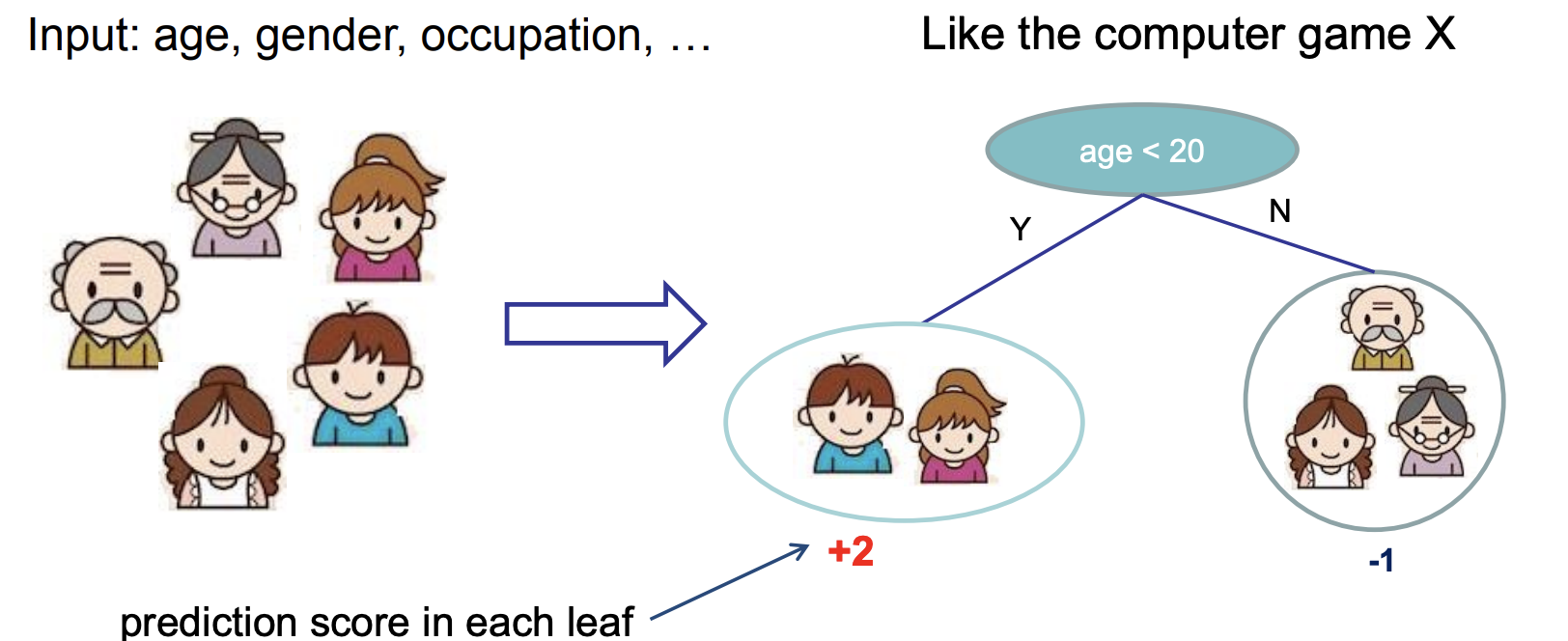
## 6.2 随机森林（Random Forest）

 随机森林的主要思想可以用下图表示：

可以看到，随机森林是一种以决策树为基分类器的集成算法，通过组合多棵独立的决策树后根据投票或取均值的方式得到最终预测结果的机器学习方法。

随机森林相比于决策树拥有出色的性能，这主要取决于他的“随机抽取样本特征”和“集成算法”，前者让它具有更稳定的抗过拟合能力，后者让它有更高的准确率。

### 6.2.1 分类回归树（CART）

随机森林的决策树一般被称为CART（Classification and Regress tree，分类回归树，是决策树【decision tree】的一种，一般而言，用于数据挖掘的决策树都可以称为CART，然后根据更进一步的用途，即，用于分类还是回归，分为分类树和回归树）本文中，我们需要的是一种分类算法，也就是根据大量自变量（借贷者信息）对这些记录进行分类（会违约的和不会违约的）。而分类树的功能，简单来说，就是将这些自变量进行一定的配置之后作为树的节点，然后按照二叉树的二分法给出一个分类结果，如果和实际的分类结果差距过大，那就重新调节这个配置，直到和实际结果相差不大为止，但是，单独的决策树经常会遇到过拟合程度太大和准确率太低无法取舍的问题，为此，可以改进为随机森林（Random Forest，简称RF）来解决此题。

上图给出了一个分类树的例子，已知五个记录（分别包含年龄、性别、职业等属性），和分类的结果，我们于是根据这些信息配置一个分类回归树，内结点是分类依据的属性，叶结点是分类的结果，我们需要变更内结点的配置来使得叶结点的分类结果尽量和真实值相同，上例中只有一个内结点年龄，实际上，更加复杂的分类可能会让叶结点包含所有的属性。

### 6.2.2 随机森林的优化：随机化

随机森林为了保证较强的抗过拟合和抗噪声能力，在构建每一棵CART决策树的时候采用了行抽样和列抽样的随机化方法。

**行抽样：**

假设训练集的数据行数为，对于每一棵CART树，我们从个原始样本中有放回地随机抽取个作为单棵树的训练集。

假设随机森林中CART树数目为，那么我们通过该办法生成个独立的训练集用于CART的训练。

对于单独一个样本而言，它在次有放回地随机抽样中都不被抽中的概率是：

当足够大时，该式的结果约等于1/e，即在每一轮行抽样大概有的36.8% 数据始终不会被采集到。

**列抽样：**

假设原始数据集的特征数为，在通过行采样获取每棵CART树的训练集后，随机森林会随机选取个特征（）训练用于每一棵CART树的生成。

当越小时，模型的抗干扰性和抗过拟合性越强，但是模型的准确率会下降，因此在实际建模过程中，常需要用交叉验证等方式选择合适的值

### 6.2.3 实现的思路

我们知道在python的机器学习包中，也有完整的随机森林的学习模块，在sklearn（scikit-learn）之下，为此，我们可以借助python实现上文所述的机器学习方法，同样需要注意的是，随机森林要求真实值不能为空，因此，我们在数据清洗的时候也将所有NaN进行合理的填充，我们同样采用中位数填充法，填充完成后，我们同样使用归一化方法将所有的数据都压缩到0-1之间。

具体的代码和实验结果见第七章。

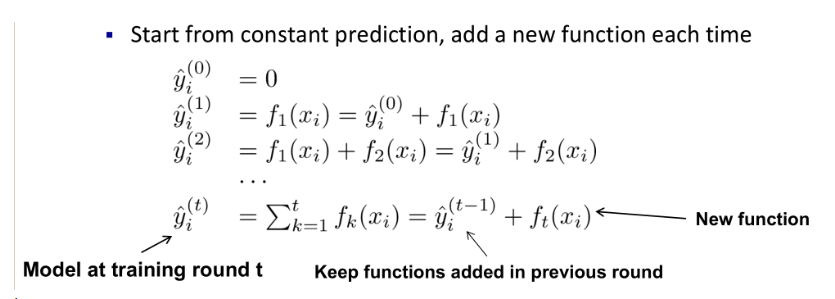
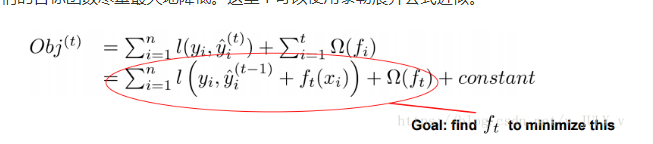
## 6.3 XGBoost和LGBM

### 6.3.1 XGboost

XGboost是Extreme Gradient Boosting （Decision Tree）（极限梯度提升决策树）的简称，和随机森林一样，他也是通过改进单一决策树而来的，不过他的想法不是简单的增加树的数量，而是通过泰勒展开式，对之前决策树的优化过程做一些改进，使得优化更加有效，具体的算法思想如下：

1. 不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数f(x)，去拟合上次预测的残差。
2. 当我们训练完成得到k棵树，我们要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，每个叶子节点就对应一个分数
3. 最后只需要将每棵树对应的分数加起来就是该样本的预测值。

XGboost的关键在于优化误差项，我们这里每一次迭代，都是增加一棵树去拟合之前树的预测结果和真实值之间的残差，如下图所示，一个f(x\_i )可以看作是一个决策树的配置函数。

 每一次加入的f(x)可以用泰勒公式近似：

其中，是用户定义的残差。

同时，XGboost还定义了树的复杂度

其中T是叶子结点的个数，而ω^2表示树上叶子节点的得分w的L2模平方（对w进行L2正则化，相当于针对每个叶结点的得分增加L2平滑，目的是为了避免过拟合），γ和λ是用户自定参数，可以用于调节树复杂度的定义。优化树的配置就是需要优化下面的XGboost目标函数（损失函数揭示训练误差 + 正则化定义复杂度）：

### 6.3.2 lightGBM

lightGBM 也是用于改进GBDT的一个算法，全称是light gradient boost machine，相较于xgboost，他其实更加快捷，而且更加适合在较小型的计算机（比如PC机）上运行，在实现中，我们主要实现lightGBM，而XGboost则仅作上文的理论介绍。

比起XGboost，lightGBM的改进点有如下几个：

**直方图算法：**

直方图算法是先把连续的浮点特征值离散化成k个整数，同时构造一个宽度为k的直方图。遍历数据时，根据离散化后的值作为索引在直方图中积累统计量，当遍历一次数据后，直方图积累了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点。

**梯度下降的优化：GOSS（Gradient-based One-Side Sampling）**

GOSS是通过区分不同梯度的实例，保留较大梯度实例同时对较小梯度随机采样的方式减少计算量，从而达到提升效率的目的。

只对梯度小的样本进行采样，是因为在提升树训练过程中目标函数学习的就是负梯度（近似残差），梯度小说明训练误差已经很小了，对这部分数据的进一步学习的效果不如对梯度大的样本进行学习的效果好或者说对梯度小的样本进行进一步学习对改善结果精度帮助其实并不大。

**特征合并的优化：EFB（Exclusive Feature Bundling[独立特征合并]）**

EFB是通过特征捆绑的方式减少特征维度（其实是降维技术）的方式来提升计算效率。通常被捆绑的特征都是互斥的（一个特征值为零一个特征值不为零），这样两个特征捆绑起来才不会丢失信息。如果两个特征并不是完全互斥（部分情况下两个特征都是非零值），可以用一个指标对特征不互斥程度进行衡量，称之为冲突比率，当这个值较小时，可以选择把不完全互斥的两个特征捆绑，而不影响最后的精度。

### 6.3.3 lightGBM实现：

我们借助XXX包进行实现，需要注意的是，lightGBM不需要要求所有数据的真实标签已知，因此，我们在数据清洗的时候，即使有一些残留的NaN值也没有很大的关系。

代码见第七章

# 七、实验与分析

这一章偏重于的对(五)的实现，以及对比不同实现方法的效果好坏

## 7.1 逻辑斯蒂回归

我们首先引入需要的包，并且读入数据：

**import** pandas **as** pd

**from** sklearn**.**preprocessing **import** MinMaxScaler

**from** sklearn**.**impute **import** SimpleImputer

FILE\_NAME **=** "preprocess.ipynb"

PARENT\_DIR **=** os**.**path**.**abspath**(**os**.**path**.**join**(**os**.**path**.**dirname**(**FILE\_NAME**),** "."**))**

app\_train **=** pd**.**read\_csv**(** PARENT\_DIR **+** '/data/processed\_train.csv'**)**

app\_test **=** pd**.**read\_csv**(** PARENT\_DIR **+** '/data/processed\_test.csv'**)**

之后我们按照上文所述进行数据预处理，首先读取数据和特征名，然后拷贝一份数据，利用中位数填充缺失值（借助sklearn.impute实现），之后我们再使用sklearn.scaler进行数据归一化，将所有数据压缩到0-1之间，代码如下：

# Drop the target from training data

**if** 'TARGET' **in** app\_train**:**

train **=** app\_train**.**drop**(**columns **=** **[**'TARGET'**])**

**else:**

train **=** app\_train**.**copy**()**

# Feature names

features **=** **list(**train**.**columns**)**

# Copy of the testing data

test **=** app\_test**.**copy**()**

# Median imputation of missing values

imputer **=** SimpleImputer**(**strategy**=**'median'**)**

# Scale each feature to 0-1 (Normalization)

scaler **=** MinMaxScaler**(**feature\_range**=(**0**,**1**))**

# Fit on the training data

imputer**.**fit**(**train**)**

# Transform both training data and testing data

train **=** imputer**.**transform**(**train**)**

test **=** imputer**.**transform**(**app\_test**)**

# Repeate with the scaler

scaler**.**fit**(**train**)**

train **=** scaler**.**transform**(**train**)**

test **=** scaler**.**transform**(**test**)**

**print(**'Training data shape:'**,** train**.**shape**)**

**print(**'Testing data shape:'**,** test**.**shape**)**

之后，如前所述，我们为了估计模型学习的准确率，需要对数据集进行切分，这个步骤需要我们使用sklearn.model\_selection实现，从中调用函数train\_test\_split

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

# 70% training and 30% testing

X\_train**,** X\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**train**,** train\_labels**,** test\_size **=** 0.3**,** stratify **=** train\_labels**)**

X\_train

然后我们调用sklearn中的学习方法LogisticRegression

**from** sklearn**.**linear\_model **import** LogisticRegression

# Make the model with the specified regularization parameter

log\_reg **=** LogisticRegression**()**

log\_reg**.**fit**(**X\_train**,** y\_train**)**

y\_pred **=** log\_reg**.**predict\_proba**(**X\_test**)[:,**1**]**

为了评价我们的学习模型，我们这里设定了ROC曲线下的面积作为评价标准：

ROC曲线是指“接受者操作特性曲线（**R**eceiver **O**perating **C**haracteristic curve）”，又称为感受性曲线（sensitivity curve）。，以被试在不同判断标准下所得的虚报概率fpr(**f**alse **p**ositive **r**ate)为横坐标，以击中概率tpr(**t**rue **p**ositive **r**ate)为纵坐标，画得的各点的连线。假设一个算法根本就是随机的决定分类，那么他的击中概率和虚报概率应该相等，表现在图像上就是在y=x这条一三对角线上，而可用的分类算法，其ROC曲线一定是上凸的（所有点都在y=x之上），最好的分类方法其虚报率为0，击中率为100%（=1）,应该是x=0,y=1这条折线，在这两者之间的话，一个学习模型的ROC曲线越偏向左上方，学习能力越高。

为了定量的评判这个能力，我们可以使用ROC曲线下的面积来度量，sklearn中有专门的模块用于度量曲线下的面积：AUC（**A**rea **U**nder **C**urve）显然，根据之前的论述，曲线下的面积越大，这个模型越好。

我们给出作图的代码：

**from** sklearn**.**metrics **import** roc\_curve

**from** sklearn**.**metrics **import** auc

**import** matplotlib**.**pyplot **as** plt

fpr\_lr**,** tpr\_lr**,** thresholds\_lr **=** roc\_curve**(**y\_test**,** y\_pred**)**

auc\_lr **=** auc**(**fpr\_lr**,** tpr\_lr**)**

plt**.**plot**([**0**,** 1**],** **[**0**,** 1**],** 'k--'**)**

plt**.**plot**(**fpr\_rf**,** tpr\_rf**,** label**=**'LR (area = {:.3f})'**.format(**auc\_rf**))**

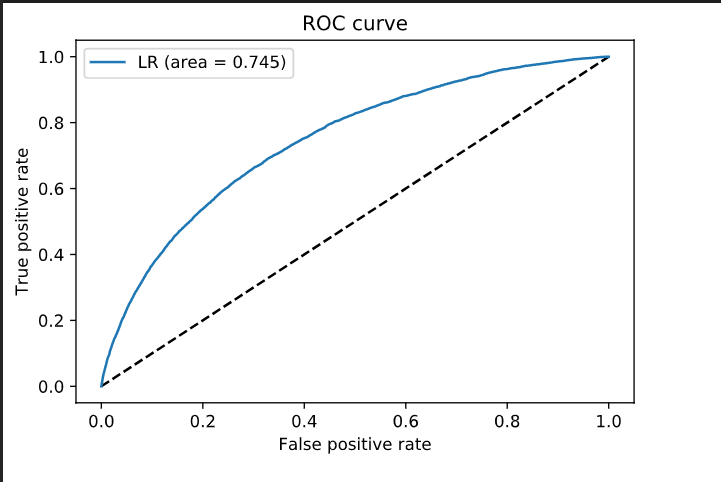
plt**.**xlabel**(**'False positive rate'**)**

plt**.**ylabel**(**'True positive rate'**)**

plt**.**title**(**'ROC curve'**)**

plt**.**legend**(**loc**=**'best'**)**

plt**.**show**()**

结果：

我们也可以使用LogsiticRegression内置的.score功能给出分类的结果

log\_reg**.**score**(**X\_test**,**y\_test**)**

结果为：

0.9192230147202289

## 7.2 随机森林

随机森林的数据清洗过程和7.1的逻辑斯蒂回归是完全一致的，数据集的切分也和7.1保持一致，不相同的地方仅仅在于我们的学习模型在制定的时候指定为sklearn.ensemble下的RandomForestClassifier，如下：

**from** sklearn**.**ensemble **import** RandomForestClassifier

**import** pandas **as** pd

# Make the random forest classifier

random\_forest **=** RandomForestClassifier**()**

# training

random\_forest **=** RandomForestClassifier**()**

random\_forest**.**fit**(**X\_train**,** y\_train**)**

# Extract feature importances

feature\_importance\_values **=** random\_forest**.**feature\_importances\_

feature\_importances **=** pd**.**DataFrame**({**'feature'**:** features**,**

'importance'**:** feature\_importance\_values**})**

# Make predictions on the test data

predictions\_val **=** random\_forest**.**predict\_proba**(**X\_test**)[:,** 1**]**

之后我们用相近的代码进行作图和分数计算

**from** sklearn**.**metrics **import** roc\_curve

**from** sklearn**.**metrics **import** auc

**import** matplotlib**.**pyplot **as** plt

fpr\_rf**,** tpr\_rf**,** thresholds\_rf **=** roc\_curve**(**y\_test**,** predictions\_val**)**

auc\_rf **=** auc**(**fpr\_rf**,** tpr\_rf**)**

plt**.**plot**([**0**,** 1**],** **[**0**,** 1**],** 'k--'**)**

plt**.**plot**(**fpr\_rf**,** tpr\_rf**,** label**=**'RF (area = {:.3f})'**.format(**auc\_rf**))**

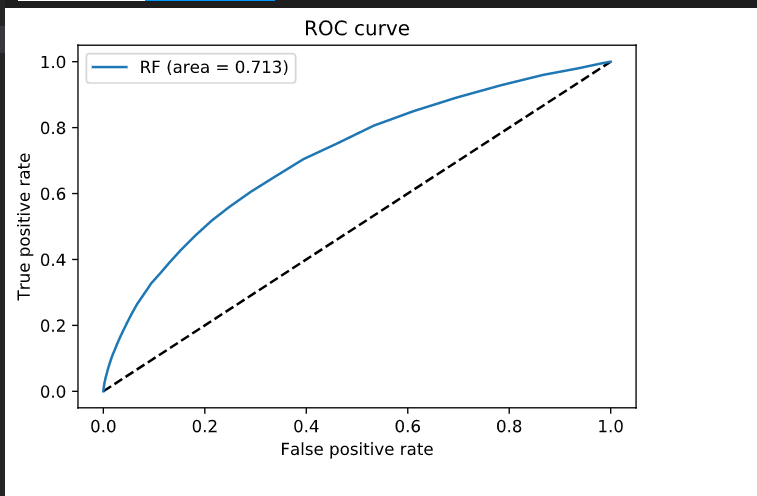
plt**.**xlabel**(**'False positive rate'**)**

plt**.**ylabel**(**'True positive rate'**)**

plt**.**title**(**'ROC curve'**)**

plt**.**legend**(**loc**=**'best'**)**

plt**.**show**()**

图像如下：

然后计分的函如下：

random\_forest**.**score**(**X\_test**,**y\_test**)**

结果为：

0.9192338543586186

可以看到，随机森林在单纯的ROC作图上似乎不如简单的逻辑斯蒂回归，但是其得分更高，也许是因为随机森林在面对一些阈值比较极端的情况下其效果不如逻辑斯蒂回归（鲁棒性不够），表现在ROC的图像上就是曲线的两段和对角线比较接近，所以才会出现面积小于逻辑斯蒂回归的情况。

## 7.3 LightGBM

lightGBM的实现相对来说更加复杂，因为在经典的机器学习模型sklearn之中，并没有提供lightgbm模型，所以我们需要另外引入lightgbm包，同时，在度量学习效果的时候，我们仍然需要借助sklearn中的度量模块（sklearn.metrics），这就需要注意这两个模块的函数的兼容问题，下文中我们会在度量结果这章中详细说明。

### 7.3.1 模型搭建

我们定义一个lightgbm的封装模块model,这个封装模块需要的参数是features、test\_features,也就是测试集和训练集，这里我们同时引入交叉校验模块，让训练更加有效，交叉校验的分组默认是五组，另外，在给变量分类的时候，我们使用的编码方式默认是独热码（one-hot-encoding，encoding = ‘ohe’），也可以改成顺序标签编码（‘le’），不过效果应该是独热码比较好。

搭建模型的思路如下：首先我们将数据记录的预测目标和身份标识号（‘SK\_ID\_CURR’）和预测目标（因变量‘TARGET’）取出，然后我们将余下的数据进行独热码或者顺序编码，这样算完成了数据预处理。之后我们创建交叉校验的KFold，从训练的数据集中随机抽取一部分做训练集（如果是默认的KFold= 5,则抽取的是4/5）,余下的（默认是1/5）,做测试集，通过lgb.LGBMClassifier进行训练，重复这个“训练、测试”的过程（如果是默认的切分率，那么要重复五次，来确保每个数据都被训练和测试到），分别记录测试的结果（结果也会打印在屏幕上），我们寻找最好的测试结果，保留此时的模型配置，然后以这个配置对用户给出的测试集做预测，然后给出结果（返回submission）,以及一些运行结果的参数（最优情况下的正确率），以及每个单独的属性对预测结果的影响。

代码如下（绿色汉字对于代码的内容作了分割）：

**def** model**(**features**,** test\_features**,** encoding **=** 'ohe'**,** n\_folds**=** 5**):**

"""Train and test a light gradient boosting model using

cross validation.

Parameters

--------

features (pd.DataFrame):

dataframe of training features to use

for training a model. Must include the TARGET column.

test\_features (pd.DataFrame):

dataframe of testing features to use

for making predictions with the model.

encoding (str, default = 'ohe'):

method for encoding categorical variables. Either 'ohe' for one-hot encoding or 'le' for integer label encoding

n\_folds (int, default = 5): number of folds to use for cross validation

Return

--------

submission (pd.DataFrame):

dataframe with `SK\_ID\_CURR` and `TARGET` probabilities

predicted by the model.

feature\_importances (pd.DataFrame):

dataframe with the feature importances from the model.

valid\_metrics (pd.DataFrame):

dataframe with training and validation metrics (ROC AUC) for each fold and overall.

"""

# Extract the ids

train\_ids **=** features**[**'SK\_ID\_CURR'**]**

test\_ids **=** test\_features**[**'SK\_ID\_CURR'**]**

# Extract the labels for training

labels **=** features**[**'TARGET'**]**

# Remove the ids and target

features **=** features**.**drop**(**columns **=** **[**'SK\_ID\_CURR'**,** 'TARGET'**])**

test\_features **=** test\_features**.**drop**(**columns **=** **[**'SK\_ID\_CURR'**])**

##数据读入部分如上------------------------------------------

# One Hot Encoding

**if** encoding **==** 'ohe'**:**

features **=** pd**.**get\_dummies**(**features**)**

test\_features **=** pd**.**get\_dummies**(**test\_features**)**

# Align the dataframes by the columns

features**,** test\_features **=** features**.**align**(**test\_features**,** join **=** 'inner'**,** axis **=** 1**)**

# No categorical indices to record

cat\_indices **=** 'auto'

# Integer label encoding

**elif** encoding **==** 'le'**:**

# Create a label encoder

label\_encoder **=** LabelEncoder**()**

# List for storing categorical indices

cat\_indices **=** **[]**

# Iterate through each column

**for** i**,** col **in** **enumerate(**features**):**

**if** features**[**col**].**dtype **==** 'object'**:**

# Map the categorical features to integers

features**[**col**]** **=** label\_encoder**.**fit\_transform**(**np**.**array**(**features**[**col**].**astype**(str)).**reshape**((-**1**,)))**

test\_features**[**col**]** **=** label\_encoder**.**transform**(**np**.**array**(**test\_features**[**col**].**astype**(str)).**reshape**((-**1**,)))**

# Record the categorical indices

cat\_indices**.**append**(**i**)**

# Catch error if label encoding scheme is not valid

**else:**

**raise** **ValueError(**"Encoding must be either 'ohe' or 'le'"**)**

**print(**'Training Data Shape: '**,** features**.**shape**)**

**print(**'Testing Data Shape: '**,** test\_features**.**shape**)**

## 编码处理部分如上--------------------------------------

# Extract feature names

feature\_names **=** **list(**features**.**columns**)**

# Convert to np arrays

features **=** np**.**array**(**features**)**

test\_features **=** np**.**array**(**test\_features**)**

# Create the kfold object

k\_fold **=** KFold**(**n\_splits **=** n\_folds**,** shuffle **=** **True,** random\_state **=** 50**)**

# Empty array for feature importances

feature\_importance\_values **=** np**.**zeros**(len(**feature\_names**))**

# Empty array for test predictions

test\_predictions **=** np**.**zeros**(**test\_features**.**shape**[**0**])**

# Empty array for out of fold validation predictions

out\_of\_fold **=** np**.**zeros**(**features**.**shape**[**0**])**

# Lists for recording validation and training scores

valid\_scores **=** **[]**

train\_scores **=** **[]**

## 创建交叉校验模型如上-----------------------------------

# Iterate through each fold

**for** train\_indices**,** valid\_indices **in** k\_fold**.**split**(**features**):**

# Training data for the fold

train\_features**,** train\_labels **=** features**[**train\_indices**],** labels**[**train\_indices**]**

# Validation data for the fold

valid\_features**,** valid\_labels **=** features**[**valid\_indices**],** labels**[**valid\_indices**]**

# Create the model

model **=** lgb**.**LGBMClassifier**(**n\_estimators**=**10000**,** objective **=** 'binary'**,**

class\_weight **=** 'balanced'**,** learning\_rate **=** 0.05**,**

reg\_alpha **=** 0.1**,** reg\_lambda **=** 0.1**,**

subsample **=** 0.8**,** n\_jobs **=** **-**1**,** random\_state **=** 50**)**

# Train the model

model**.**fit**(**train\_features**,** train\_labels**,** eval\_metric **=** 'auc'**,**

eval\_set **=** **[(**valid\_features**,** valid\_labels**),** **(**train\_features**,** train\_labels**)],**

eval\_names **=** **[**'valid'**,** 'train'**],** categorical\_feature **=** cat\_indices**,**

early\_stopping\_rounds **=** 100**,** verbose **=** 200**)**

## 模型多次训练如上---------------------------------------

# Record the best iteration

best\_iteration **=** model**.**best\_iteration\_

# Record the feature importances

feature\_importance\_values **+=** model**.**feature\_importances\_ **/** k\_fold**.**n\_splits

# Make predictions

test\_predictions **+=** model**.**predict\_proba**(**test\_features**,** num\_iteration **=** best\_iteration**)[:,** 1**]** **/** k\_fold**.**n\_splits

# Record the out of fold predictions

out\_of\_fold**[**valid\_indices**]** **=** model**.**predict\_proba**(**valid\_features**,** num\_iteration **=** best\_iteration**)[:,** 1**]**

# Record the best score

valid\_score **=** model**.**best\_score\_**[**'valid'**][**'auc'**]**

train\_score **=** model**.**best\_score\_**[**'train'**][**'auc'**]**

valid\_scores**.**append**(**valid\_score**)**

train\_scores**.**append**(**train\_score**)**

# Clean up memory

gc**.**enable**()**

**del** model**,** train\_features**,** valid\_features

gc**.**collect**()**

## 记录最优结果和最优配置如上------------------------------

# Make the submission dataframe

submission **=** pd**.**DataFrame**({**'SK\_ID\_CURR'**:** test\_ids**,** 'TARGET'**:** test\_predictions**})**

# Make the feature importance dataframe

feature\_importances **=** pd**.**DataFrame**({**'feature'**:** feature\_names**,** 'importance'**:** feature\_importance\_values**})**

# Overall validation score

valid\_auc **=** roc\_auc\_score**(**labels**,** out\_of\_fold**)**

# Add the overall scores to the metrics

valid\_scores**.**append**(**valid\_auc**)**

train\_scores**.**append**(**np**.**mean**(**train\_scores**))**

# Needed for creating dataframe of validation scores

fold\_names **=** **list(range(**n\_folds**))**

fold\_names**.**append**(**'overall'**)**

# Dataframe of validation scores

metrics **=** pd**.**DataFrame**({**'fold'**:** fold\_names**,**

'train'**:** train\_scores**,**

'valid'**:** valid\_scores**})**

## 创建返回值如上---------------------------------------

**return** submission**,** feature\_importances**,** metrics

### 7.3.2数据读入和训练

用和7.1、7.2一样的方法读入数据，但是，如前所述，lightgbm不要求数据没有缺失值，于是我们不必用中位数填充，也不需要对数据进行归一化。

之后和7.1、7.2一致，我们也是把数据划分为70%用于训练，30%用于测试。（注意这里的划分不同于7.3.1训练过程中的KFold的划分，KFold的划分是暂时的，最终每个数据都会被训练和测试，但是这里的划分中，后30%的数据从未被训练到，是直接用于预测和检验模型的可靠性的）不过，我们这里不用之前使用的sklearn.model\_selection. train\_test\_split实现分割，而是使用更加直接的numpy的数据选择方法，理由是，sklearn的模型会自动把数据的标签（预测值‘TARGET’）和其他值分开，但是我们是在7.3.1定义的模型“model”中完成这步操作的，所以如果用sklearn.model\_selection.train\_test\_split的话，会由于新的数据在model中找不到‘TARGET’而报错，于是我们使用numpy的数据选取方法即可：

num\_train **=** **int(**app\_train**.**shape**[**0**]** **\*** 0.7**)**

X\_train **=** app\_train**[:**num\_train**]**

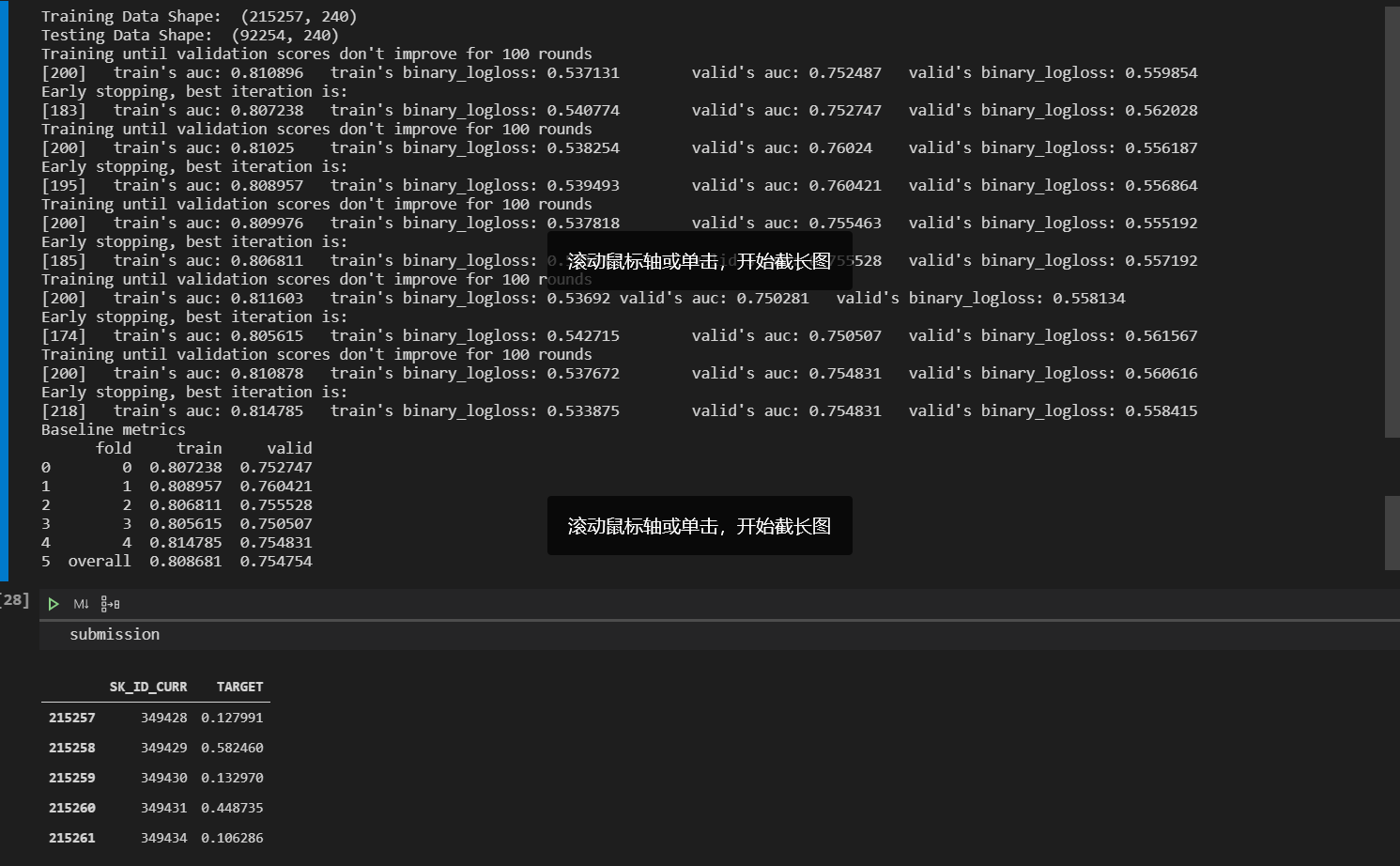
X\_test **=** app\_train**[**num\_train**:]**

X\_train

submission**,** fi**,** metrics **=** model**(**X\_train**,** X\_test**)**

**print(**'Baseline metrics'**)**

**print(**metrics**)**

实验结果：

最终的实验结果大概在75%-80%之间，表示对于模型对于训练集的拟合优度。

### 7.3.3模型测试

为了得到模型对于测试集的拟合优度，也就是模型真正的准确率，我们同样采用上文的ROC曲线的方法，代码如下：

**from** sklearn**.**metrics **import** roc\_curve

**from** sklearn**.**metrics **import** auc

**import** matplotlib**.**pyplot **as** plt

fpr\_lgbm**,** tpr\_lgbm**,** thresholds\_lgbm **=** roc\_curve**(**X\_test**[**'TARGET'**],** submission**[**'TARGET'**])**

auc\_lgbm **=** auc**(**fpr\_lgbm**,** tpr\_lgbm**)**

plt**.**plot**([**0**,** 1**],** **[**0**,** 1**],** 'k--'**)**

plt**.**plot**(**fpr\_lgbm**,** tpr\_lgbm**,** label**=**'LGBM (area = {:.3f})'**.format(**auc\_lgbm**))**

plt**.**xlabel**(**'False positive rate'**)**

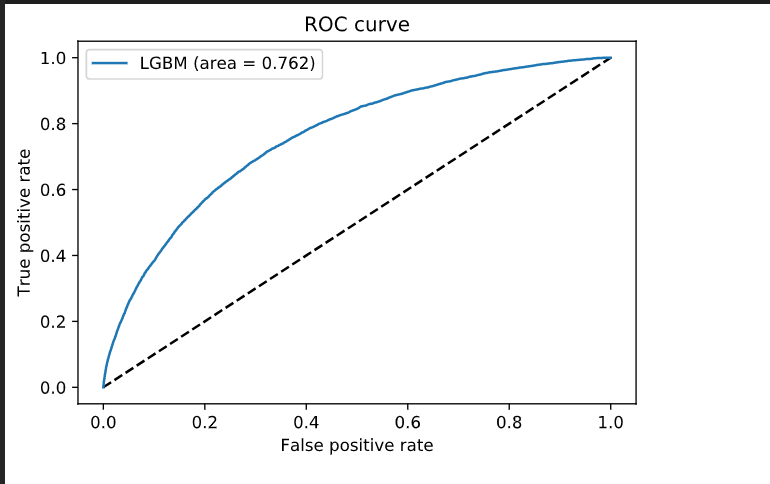
plt**.**ylabel**(**'True positive rate'**)**

plt**.**title**(**'ROC curve'**)**

plt**.**legend**(**loc**=**'best'**)**

plt**.**show**()**

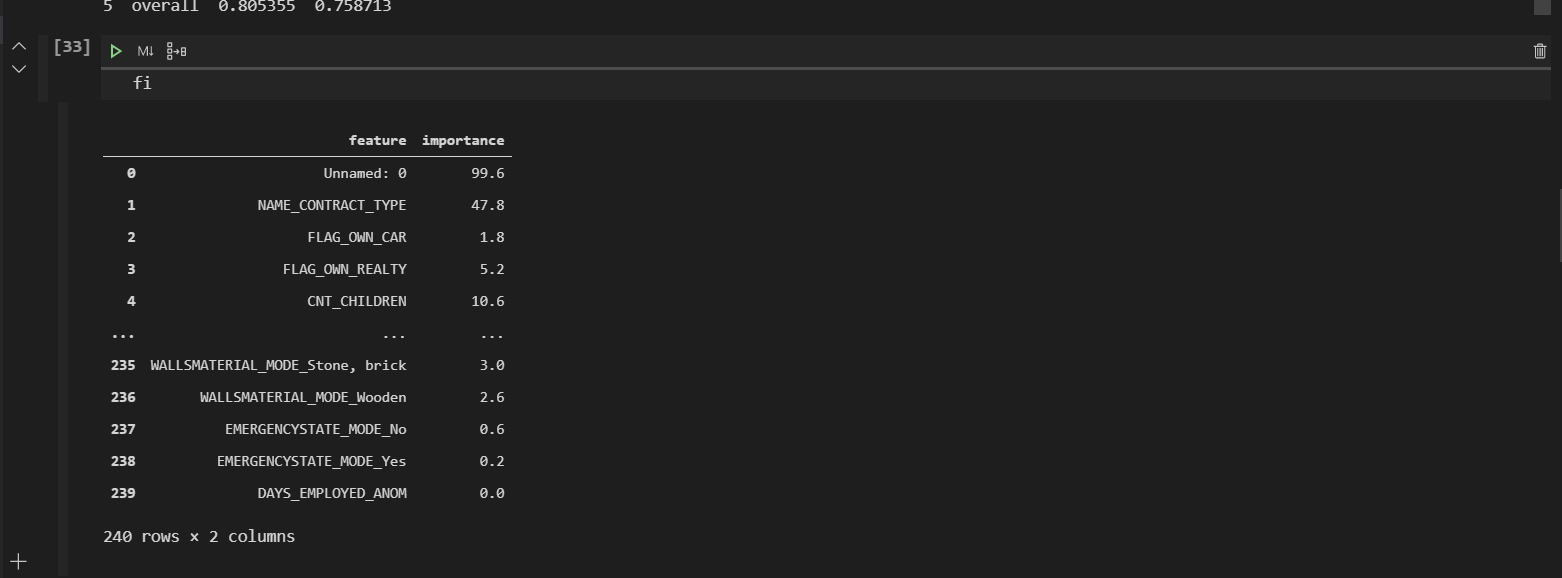
结果：



发现lightgbm的效果比起随机森林和逻辑斯蒂确实好一点，说明模型抗过拟合的能力要高很多。

### 7.3.4副产品：特征重要性

我们打印模型中的fi，得到的结果大致如下：

 这里列出了每个属性对于最后的违约与否的贡献度，贡献度越大，说明这个属性越能够决定这名借款人是否违约。我们使用如下的代码画出图像，首先做一个直观的认识。

**import** matplotlib**.**pyplot **as** plt

fi\_x **=** fi**[**'feature'**]**

fi\_y **=** fi**[**'importance'**]**

plt**.**plot**([**0**,** 1**],** **[**0**,** 1**],** 'k--'**)**

plt**.**plot**(**fi\_x**,** fi\_y**)**

plt**.**title**(**'feature importance'**)**

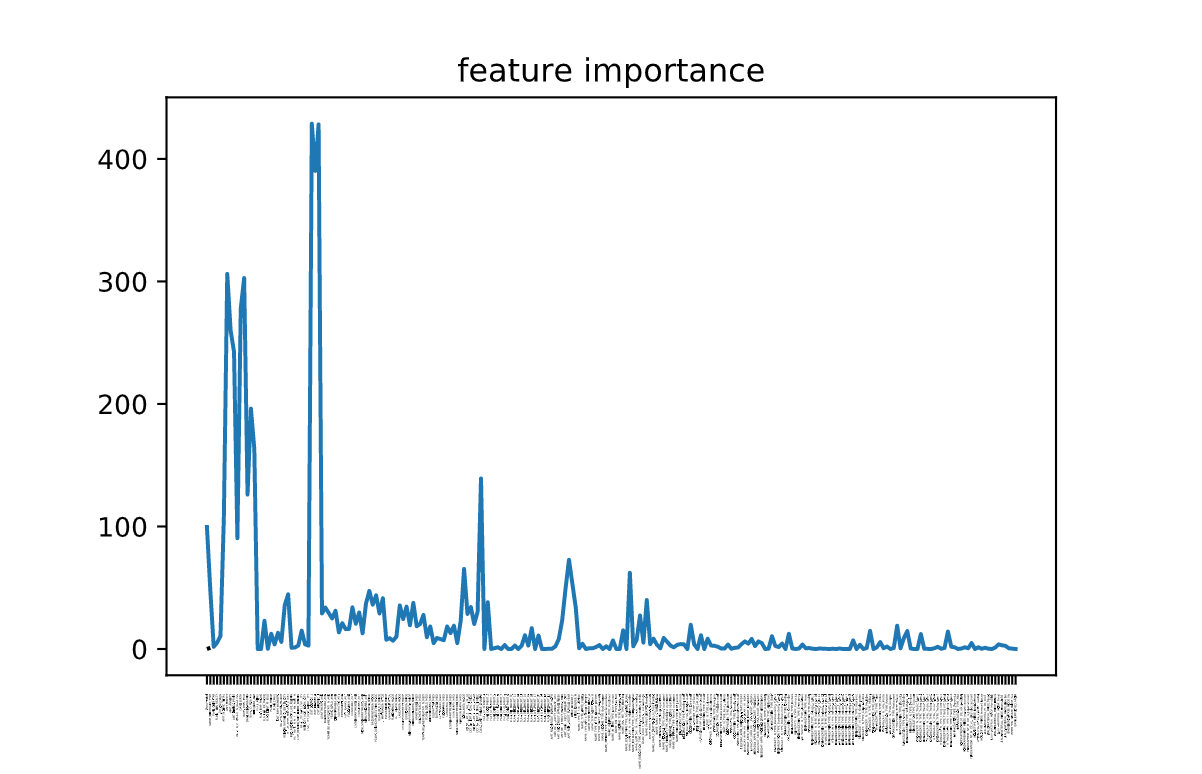
plt**.**xticks**(**rotation**=**90**)**

plt**.**xticks**(**fontsize**=** 1**)**

plt**.**savefig**(**'featrue importance.png'**)**

plt**.**show**()**

结果如下：

 因为属性太多，字体调节的比较小（避免重叠）,直观上看，前面的几个属性对于违约与否的影响非常大，前面有两个峰值值得关注，后面虽然也有三个小峰值，但是绝大多数的影响力都非常小。

通过查阅fi，我们看到，第一个较大峰值的属性是：AMT\_INCOME\_TOTAL（用户的总收入），AMT\_CREDIT（用户的总信贷），NAME\_INCOME\_TYPE（收入类型）等等，而FLAG\_OWN\_CAR和FLAG\_OWN\_REALTY（是否拥有汽车和不动产）其实影响远不如前述三个大。第二个峰值是数据集没有标明来源的三个外部数据（EXT\_SOURCE\_1、EXT\_SOURCE\_2、EXT\_SOURCE\_3），可以看到，这三分属性的影响力几乎是最大的，遗憾的是数据集的说明中没有给出这三个数据的来源，只是说了Normalized score from external data source，（其他外部数据）

# 八、结论与展望

本实验通过三种不同的学习模式，完成了对于银行风险管理的研究，在完成了特征工程之后，最终模型的学习效果比较好，可以满足比较基础的预测工作，同时，在lightgbm中，我们也给出了一个含有经济意义的预测模型，指出了哪些特征对于借款人违约与否有较大的影响，银行在管理借款时可以额外注意。

实验中，模型的准确率其实可以进一步提升，因为我们在使用各个模型进行学习的时候，各个模型的参数没有经过严格的调整，大多数都是使用了推荐值，因此，下一步的实验可以设计程序调节各个学习模型的参数，研究一下这些参数和模型学习结果的关系，这将进一步提高我们预测借款人是否会违约的精度。

# 参考文献

代码部分：

<https://www.kaggle.com/c/home-credit-default-risk/data>

<https://www.kaggle.com/acnqccc/start-here-a-gentle-introduction#Just-for-Fun:-Light-Gradient-Boosting-Machine>

<https://www.kaggle.com/acnqccc/feature-selection/edit>