



山东大学



山东大学

# 机器学习 贷款违约预测报告

学院： 计算机科学与技术学院

专业： 人工智能

班级： 19 级智能班

姓名： 陈怡然



## 目 录

一、写在前面.....	错误！未定义书签。
二、赛题介绍.....	2
1. 赛题背景.....	2
2. 任务.....	2
3. 赛题数据 .....	2
4. 评测标准.....	4
三、问题分析.....	5
1. 类型.....	5
2. AUC.....	5
3. 数据.....	5
四、赛题整体流程.....	7
五、数据 EDA.....	8
1. 总体分布.....	10
2. 数据类型分析.....	13
六、特征工程.....	27
1. 重复值处理.....	27
2. 缺失值处理.....	28
3. 异常值处理.....	29
4. 时间数据处理 .....	30
5. 特征交叉 .....	31
6. 特征编码 .....	32
7. 特征选择 .....	35
8. 对训练集处理 .....	37
8. 对测试集处理 .....	39
七、建模分析 .....	40
1. 模型考虑.....	40
2. 尝试过程.....	43
八、结果 .....	51
九、总结与收获 .....	53
十、代码 .....	54



## 一、写在前面

本次机器学习大作业我选择了天池新人赛项目贷款违约预测项目。初看到项目介绍感觉和华为实验指导书上的很多东西是类似的，就是一个数据量大一些的二分类罢了。但是毕竟刚开始接触机器学习，对于一些算法的理解和使用还是不够熟悉，或者说是还没有很好地掌握机器学习，所以在一开始还是很蒙的状态。不知道如何下手，在查询了很多资料后，渐渐地步入正轨。

从看到题目到完成大约用了两个星期的时间，中间走了不少弯路，一开始做的数据处理，特征工程等数据清洗地过于干净，导致最后的模型无论怎么选分数都不是很高。后来发现项目给出的数据就已经很干净了，是很符合实际贷款情况的数据，所以不用清洗太多地数据，即使有一些是在学习过程中通常会处理的情况。最后简化了数据处理和特征选择后，分数提升地就比较大了。也学习到了理论要服务与实际的道理。

总的来说大作业很难，我做的很艰辛，收获很丰富，不想挂科，不想再来一遍了，求求了!!!



## 二、赛题介绍

### 1. 赛事背景

赛题以金融风控中的个人信贷为背景，要求选手根据贷款申请人的数据信息预测其是否有违约的可能，以此判断是否通过此项贷款，这是一个典型的分类问题。

### 2. 任务

预测用户贷款是否违约。

### 3. 赛题数据

赛题以预测用户贷款是否违约为任务，数据集报名后可见并可下载，该数据来自某信贷平台的贷款记录，总数据量超过 120w，包含 47 列变量信息，其中 15 列为匿名变量。为了保证比赛的公平性，将会从中抽取 80 万条作为训练集，20 万条作为测试集 A，20 万条作为测试集 B，同时会对 employmentTitle、purpose、postCode 和 title 等信息进行脱敏。

- 字段表



Field	Description
id	为贷款清单分配的唯一信用证标识
loanAmnt	贷款金额
term	贷款期限 (year)
interestRate	贷款利率
installment	分期付款金额
grade	贷款等级
subGrade	贷款等级之子级
employmentTitle	就业职称
employmentLength	就业年限 (年)
homeOwnership	借款人在登记时提供的房屋所有权状况
annualIncome	年收入
verificationStatus	验证状态
issueDate	贷款发放的月份
purpose	借款人在贷款申请时的贷款用途类别
postCode	借款人在贷款申请中提供的邮政编码的前3位数字
regionCode	地区编码
dti	债务收入比



delinquency_2years	借款人过去2年信用档案中逾期30天以上的违约事件数
ficoRangeLow	借款人在贷款发放时的fico所属的下限范围
ficoRangeHigh	借款人在贷款发放时的fico所属的上限范围
openAcc	借款人信用档案中未结信用额度的数量
pubRec	贬损公共记录的数量
pubRecBankruptcies	公开记录清除的数量
revolBal	信贷周转余额合计
revolUtil	循环额度利用率，或借款人使用的相对于所有可用循环信贷的信贷金额
totalAcc	借款人信用档案中当前的信用额度总数
initialListStatus	贷款的初始列表状态
applicationType	表明贷款是个人申请还是与两个共同借款人的联合申请
earliesCreditLine	借款人最早报告的信用额度开立的月份
title	借款人提供的贷款名称
policyCode	公开可用的策略_代码=1新产品不公开可用的策略_代码=2
n系列匿名特征	匿名特征n0-n14，为一些贷款人行为计数特征的处理

#### 4. 评测标准

提交结果为每个测试样本是 1 的概率，也就是  $y$  为 1 的概率。评价方法为 AUC 评估模型效果（越大越好）。



### 三、问题分析

#### 1、类型

通过对赛事背景的理解，可以很清楚地得知此为一个二分类问题，对于每个样本都需要得到其  $y$  为 1（即贷款违约）的概率。

#### 2、AUC

其评测标准为 AUC。AUC 在机器学习领域中是一种模型评估指标。根据百科的定义，AUC(area under the curve)是 ROC 曲线下的面积。在二分类（0，1）的模型中，一般我们最后的输出是一个概率值，表示结果是 1 的概率。需要一个阈值，超过这个阈值则归类为 1，低于这个阈值就归类为 0。所以，不同的阈值会导致分类的结果不同，也就是混淆矩阵不一样了，FPR 和 TPR 也就不一样了。所以当阈值从 0 开始慢慢移动到 1 的过程，就会形成很多对 (FPR, TPR) 的值，将它们画在坐标系上，就是所谓的 ROC 曲线了。

$$FPR = \frac{FP}{FP + TN} \quad TPR = \frac{TP}{TP + FN}$$

真实情况	预测结果	
	y=1, 良性	y=0, 恶性
y=1, 良性	TP（真正例）	FN（假反例）
y=0, 恶性	FP（假正例）	TN（真反例）

#### 3、数据

数据都是来自信贷平台的贷款记录，所以数据中的很多特征都是在实际中很重要或者是对于结果预测有很大影响的特征，所以在进行特征选择时需要注意不





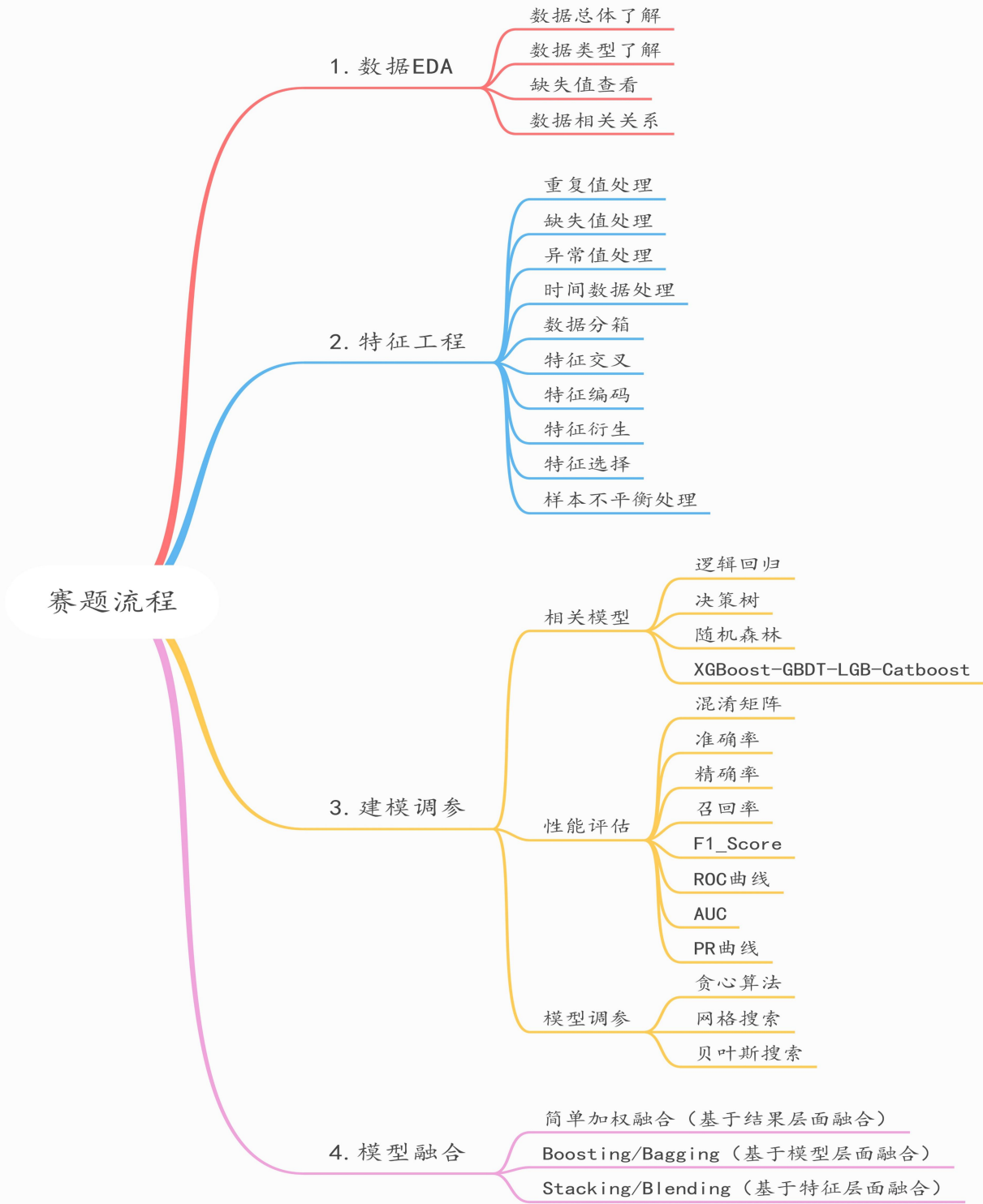
要将数据处理得太过于干净，否则将会影响分数，就像我的分数一样，在一开始特征选择和数据处理地过于干净，反而导致了最后分数较低，且不断地更换模型也无法使分数提升地很快，所以在重新简化特征选择和数据处理后，分数提升地比较快。

- 
- **日期:** 2021-06-09 09:46:04 **排名:** 无  
**score:** 0.7362
  - **日期:** 2021-06-08 16:04:31 **排名:** 无  
**score:** 0.7361
  - **日期:** 2021-06-08 15:13:23 **排名:** 无  
**score:** 0.7176
  - **日期:** 2021-06-08 08:20:07 **排名:** 无  
**score:** 0.7136
  - **日期:** 2021-06-07 22:52:10 **排名:** 无  
**score:** 0.7128
  - **日期:** 2021-06-07 21:22:54 **排名:** 无  
**score:** 0.7111





# 四、赛题整体流程



## 五、数据 EDA

数据探索性分析是对数据进行初步分析，了解数据特征，观察数据类型，分析数据分布等等，为后续特征工程，以及建模分析都特别重要。

分析数据中每个字段的含义、分布、缺失情况；字段表示什么含义、字段的类型是什么、字段的取值空间是什么、字段每个取值表示什么意义；字段整体的分布，分析字段在训练集/测试集中的分布情况；字段缺失值的分布比例，字段在训练集/测试集的缺失情况；分析数据中每个字段的与赛题标签的关系；分析数据字段两两之间，或者主者之间的关系。



首先导入模块和数据集：

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```



```
import datetime
import statsmodels.formula.api as smf

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
from lightgbm import LGBMClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.model_selection import cross_val_score

import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import roc_auc_score, accuracy_score, f1_score, log_loss

plt.rcParams["font.sans-serif"]=["SimHei"]#正常显示中文
plt.rcParams["axes.unicode_minus"]=False#正常显示负号

from jupyterthemes import jtplot
jtplot.style(theme='onedork') #选择一个绘图主题

train=pd.read_csv('G://classStudy/II 2/机器学习/大作业/贷款违约预测/train.csv')
test=pd.read_csv('G://classStudy/II 2/机器学习/大作业/贷款违约预测/testA.csv')
```

查看部分数据：

```
1 train.head()
```

	id	loanAmnt	term	interestRate	installment	grade	subGrade	employmentTitle	employmentLength	homeOwnership	...	n5	n6	n7
0	0	35000.0	5	19.52	917.97	E	E2	320.0	2 years	2	...	9.0	8.0	4.0
1	1	18000.0	5	18.49	461.90	D	D2	219843.0	5 years	0	...	NaN	NaN	NaN
2	2	12000.0	5	16.99	298.17	D	D3	31698.0	8 years	0	...	0.0	21.0	4.0
3	3	11000.0	3	7.26	340.96	A	A4	46854.0	10+ years	1	...	16.0	4.0	7.0
4	4	3000.0	3	12.99	101.07	C	C2	54.0	NaN	1	...	4.0	9.0	10.0

5 rows × 47 columns



## 1. 总体分布

前面提到，整个数据包括 80 万条训练集，20 万条测试集 A，20 万条测试集 B。另外训练集中有 47 列，其中包括 46 个特征列，1 个标签列测试集中只有 46 个特征列。

```
1 train.shape
```

```
(800000, 47)
```

```
1 test.shape
```

```
(200000, 46)
```

查看特征名：

```
1 train.columns
```

```
Index(['id', 'loanAmnt', 'term', 'interestRate', 'installment', 'grade',  
      'subGrade', 'employmentTitle', 'employmentLength', 'homeOwnership',  
      'annualIncome', 'verificationStatus', 'issueDate', 'isDefault',  
      'purpose', 'postCode', 'regionCode', 'dti', 'delinquency_2years',  
      'ficoRangeLow', 'ficoRangeHigh', 'openAcc', 'pubRec',  
      'pubRecBankruptcies', 'revolBal', 'revolUtil', 'totalAcc',  
      'initialListStatus', 'applicationType', 'earliesCreditLine', 'title',  
      'policyCode', 'n0', 'n1', 'n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8',  
      'n9', 'n10', 'n11', 'n12', 'n13', 'n14'],  
      dtype='object')
```

接下来查看数据集的一些基本信息（缺失情况、类型…）：



```
1 train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800000 entries, 0 to 799999
Data columns (total 47 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   id                    800000 non-null  int64  
1   loanAmnt              800000 non-null  float64
2   term                  800000 non-null  int64  
3   interestRate          800000 non-null  float64
4   installment           800000 non-null  float64
5   grade                 800000 non-null  object  
6   subGrade              800000 non-null  object  
7   employmentTitle       799999 non-null  float64
8   employmentLength      753201 non-null  object  
9   homeOwnership         800000 non-null  int64  
10  annualIncome          800000 non-null  float64
11  verificationStatus     800000 non-null  int64  
12  issueDate              800000 non-null  object  
```





```
13  isDefault      800000 non-null  int64
14  purpose        800000 non-null  int64
15  postCode       799999 non-null  float64
16  regionCode     800000 non-null  int64
17  dti            799761 non-null  float64
18  delinquency_2years 800000 non-null  float64
19  ficoRangeLow   800000 non-null  float64
20  ficoRangeHigh  800000 non-null  float64
21  openAcc        800000 non-null  float64
22  pubRec         800000 non-null  float64
23  pubRecBankruptcies 799595 non-null  float64
24  revolBal       800000 non-null  float64
25  revolUtil      799469 non-null  float64
26  totalAcc       800000 non-null  float64
27  initialListStatus 800000 non-null  int64
28  applicationType 800000 non-null  int64
29  earliesCreditLine 800000 non-null  object
30  title          799999 non-null  float64
31  policyCode     800000 non-null  float64
32  n0             759730 non-null  float64
33  n1             759730 non-null  float64
34  n2             759730 non-null  float64
35  n3             759730 non-null  float64
36  n4             766761 non-null  float64
37  n5             759730 non-null  float64
```

```
37  n5             759730 non-null  float64
38  n6             759730 non-null  float64
39  n7             759730 non-null  float64
40  n8             759729 non-null  float64
41  n9             759730 non-null  float64
42  n10            766761 non-null  float64
43  n11            730248 non-null  float64
44  n12            759730 non-null  float64
45  n13            759730 non-null  float64
46  n14            759730 non-null  float64
dtypes: float64(33), int64(9), object(5)
memory usage: 286.9+ MB
```

可以看到，许多特征存在缺失，特征的类型有 `dtypes: float64(33), int64(9), object(5)`。



接下来查看一下数据的描述性分析：

```
1 train.describe()
```

	id	loanAmnt	term	interestRate	installment	employmentTitle	homeOwnership	annualIncome	verificationStat
count	800000.000000	800000.000000	800000.000000	800000.000000	800000.000000	799999.000000	800000.000000	8.000000e+05	800000.000000
mean	399999.500000	14416.818875	3.482745	13.238391	437.947723	72005.351714	0.614213	7.613391e+04	1.009683
std	230940.252015	8716.086178	0.855832	4.765757	261.460393	106585.640204	0.675749	6.894751e+04	0.782716
min	0.000000	500.000000	3.000000	5.310000	15.690000	0.000000	0.000000	0.000000e+00	0.000000
25%	199999.750000	8000.000000	3.000000	9.750000	248.450000	427.000000	0.000000	4.560000e+04	0.000000
50%	399999.500000	12000.000000	3.000000	12.740000	375.135000	7755.000000	1.000000	6.500000e+04	1.000000
75%	599999.250000	20000.000000	3.000000	15.990000	580.710000	117663.500000	1.000000	9.000000e+04	2.000000
max	799999.000000	40000.000000	5.000000	30.990000	1715.420000	378351.000000	5.000000	1.099920e+07	2.000000

8 rows x 42 columns

## 2、数据类型分析

2.1、数值类型（连续变量、离散型变量和单值变量）：

```
1 # 数值类型
2 numerical_feature = list(train.select_dtypes(exclude=['object']).columns)
3 numerical_feature
```

```
['id',
 'loanAmnt',
 'term',
 'interestRate',
 'installment',
 'employmentTitle',
 'homeOwnership',
 'annualIncome',
 'verificationStatus',
 'isDefault',
 'purpose',
 'postCode',
 'regionCode',
 'dti',
 'delinquency_2years',
 'ficoRangeLow',
 'ficoRangeHigh',
 'openAcc',
 'pubRec',
 'pubRecBankruptcies',
```





```
'revolBal',  
'revolUtil',  
'totalAcc',  
'initialListStatus',  
'applicationType',  
'title',  
'policyCode',  
'n0',  
'n1',  
'n2',  
'n3',  
'n4',  
'n5',  
'n6',  
'n7',  
'n8',  
'n9',  
'n10',  
'n11',  
'n12',  
'n13',  
'n14']
```

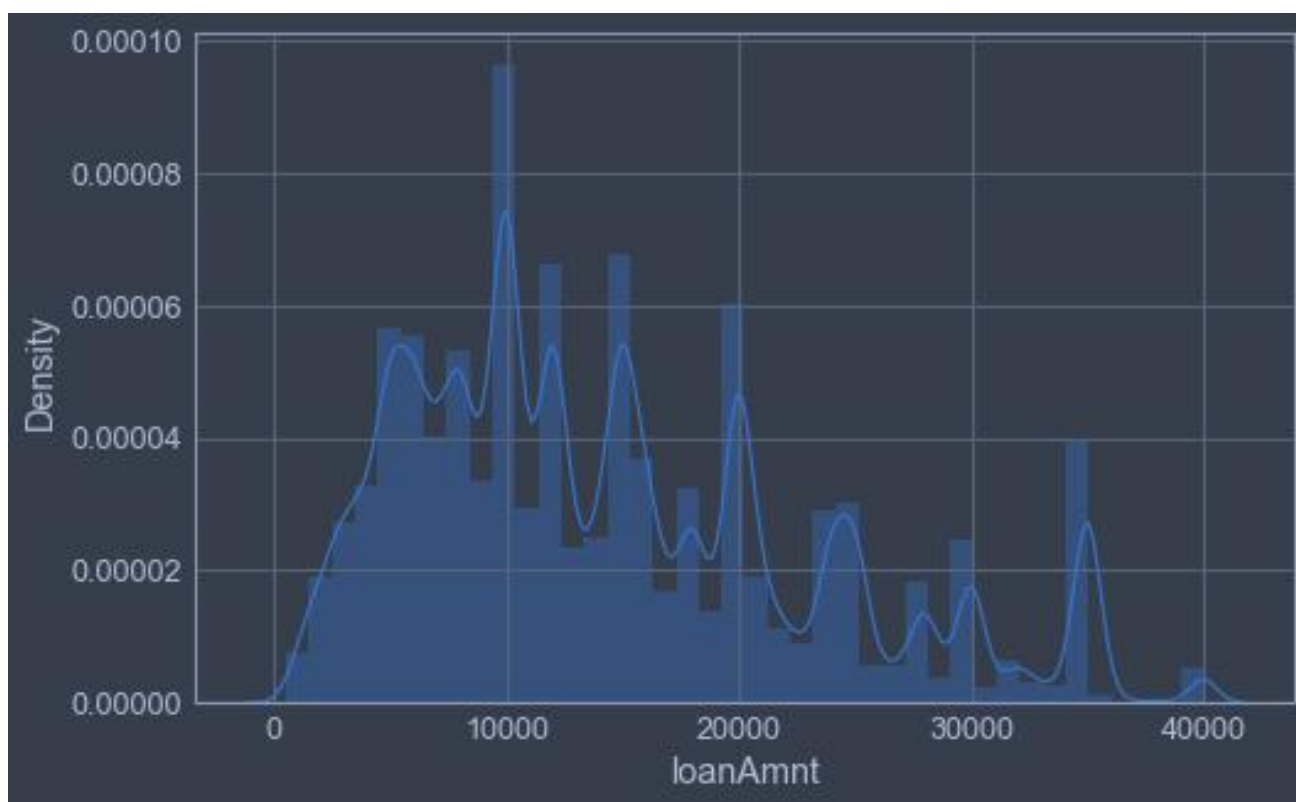
由于数值类型又可以分为连续变量、离散型变量和单值变量因此接下来把数值型中的连续型变量和离散型变量区分开来：

```
# 连续型变量  
serial_feature = []  
# 离散型变量  
discrete_feature = []  
# 单值变量  
unique_feature = []  
  
for fea in numerical_feature:  
    temp = train[fea].nunique()# 返回的是唯一值的个数  
    if temp == 1:  
        unique_feature.append(fea)  
        # 自定义变量的值的取值个数小于 10 就为离散型变量  
    elif temp <= 10:  
        discrete_feature.append(fea)  
    else:  
        serial_feature.append(fea)
```

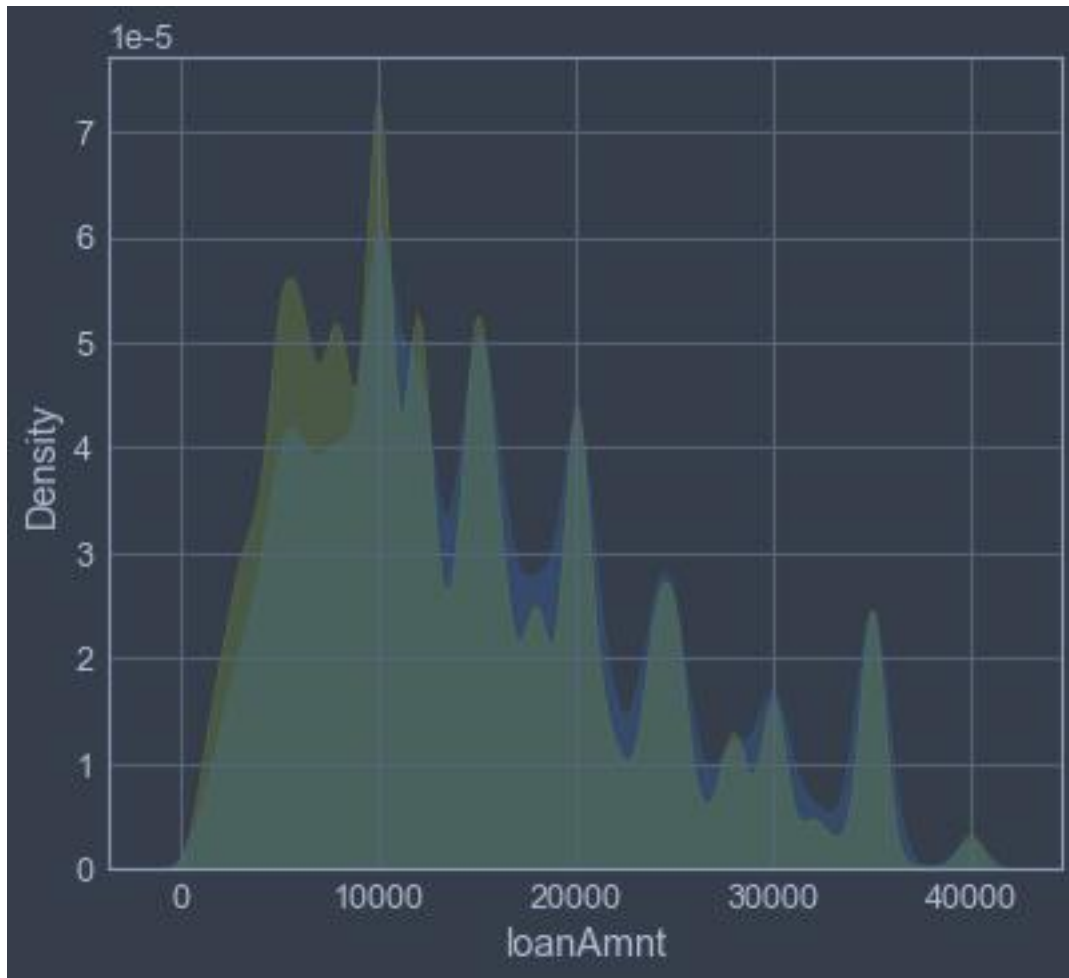
- 对于连续型变量

查看某一个数值型变量的分布，查看变量是否符合正态分布，如果不符合正态分布的变量可以  $\log$  化后再观察下是否符合正态分布。正态化的原因：一些情况下正态非正态可以让模型更快的收敛，一些模型要求数据正态 (eg. GMM、KNN)，保证数据不要过偏态即可，过于偏态可能会影响模型预测结果。

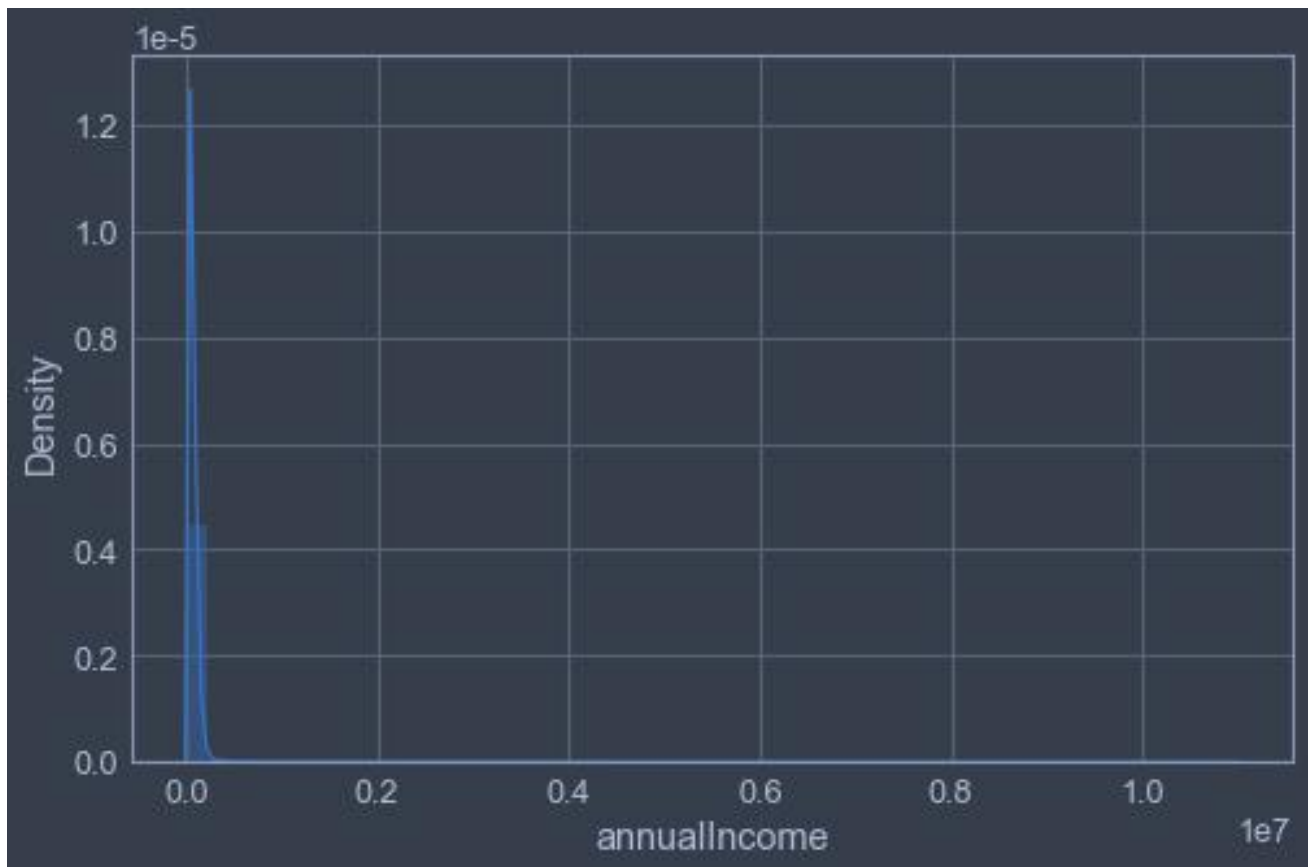
可以单独查看一下贷款金额 `loanAmnt` 的分布情况：



对于违约与不违约两类样本的贷款金额分布情况：



单独查看一下年收入的分布情况：

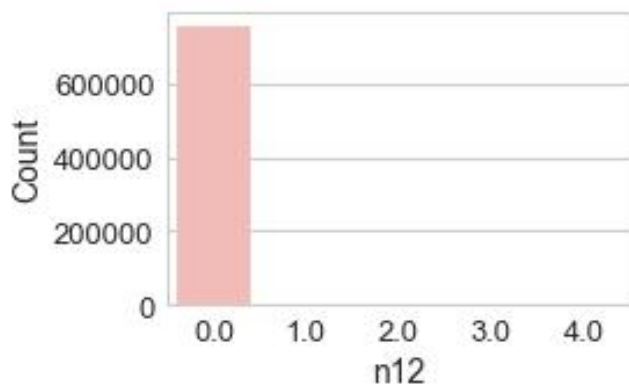
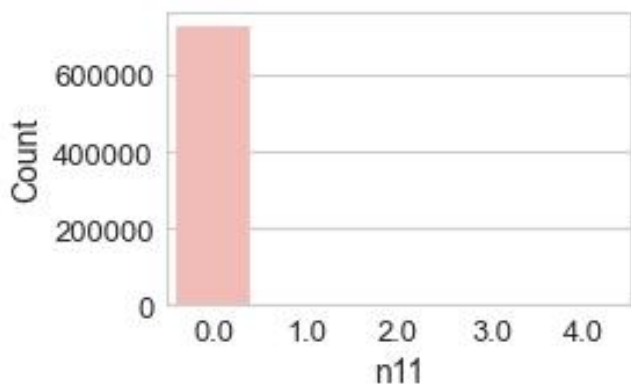
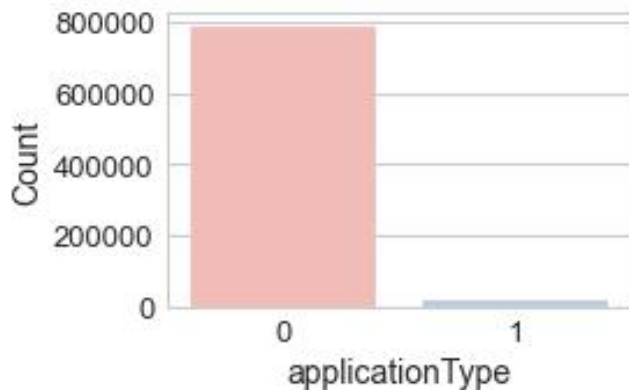
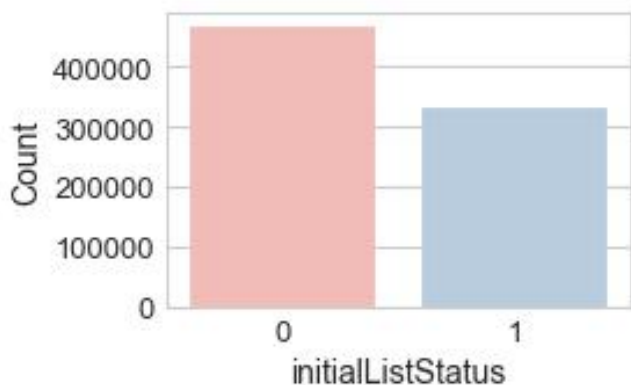
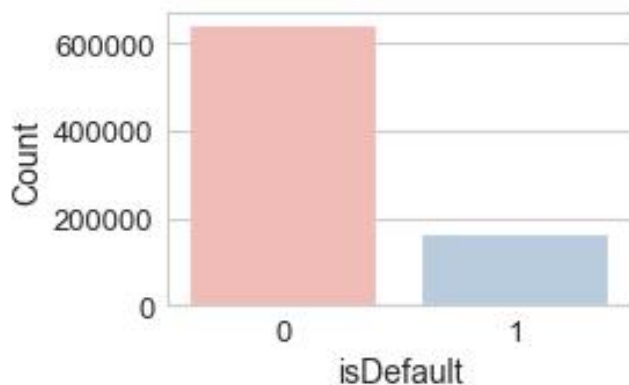
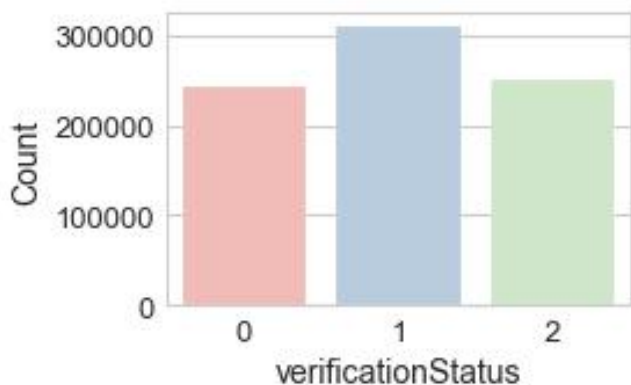
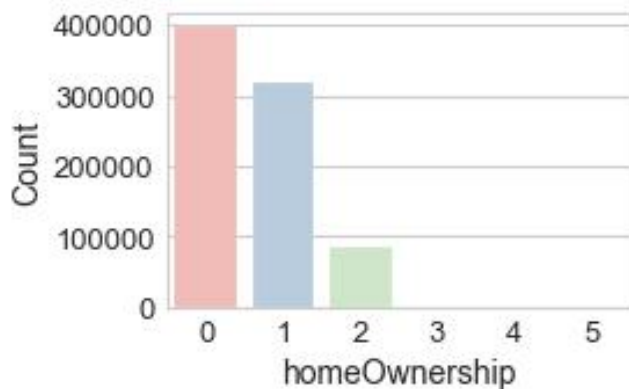
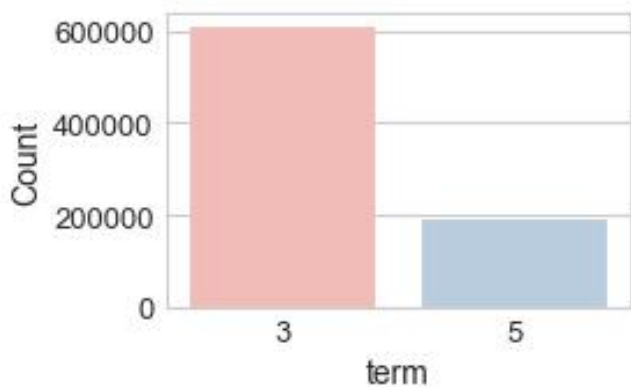


## ● 对于离散数据

离散型特征可视化呈现：

```
df_ = train[discrete_feature]

sns.set_style("whitegrid") # 使用 whitegrid 主题
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(8, 10))
for i, item in enumerate(df_):
    plt.subplot(4, 2, (i+1))
    # ax=df[item].value_counts().plot(kind='bar')
    ax = sns.countplot(item, data=df_, palette="Pastel1")
    plt.xlabel(str(item), fontsize=14)
    plt.ylabel('Count', fontsize=14)
    plt.xticks(fontsize=13)
    plt.yticks(fontsize=13)
    # plt.title("Churn by "+ str(item))
    i=i+1
plt.tight_layout()
plt.show()
```



- 对于单值变量



单值变量表示该特征只有一种类别，对于数值全部都一样的特征，可以考虑直接删除。

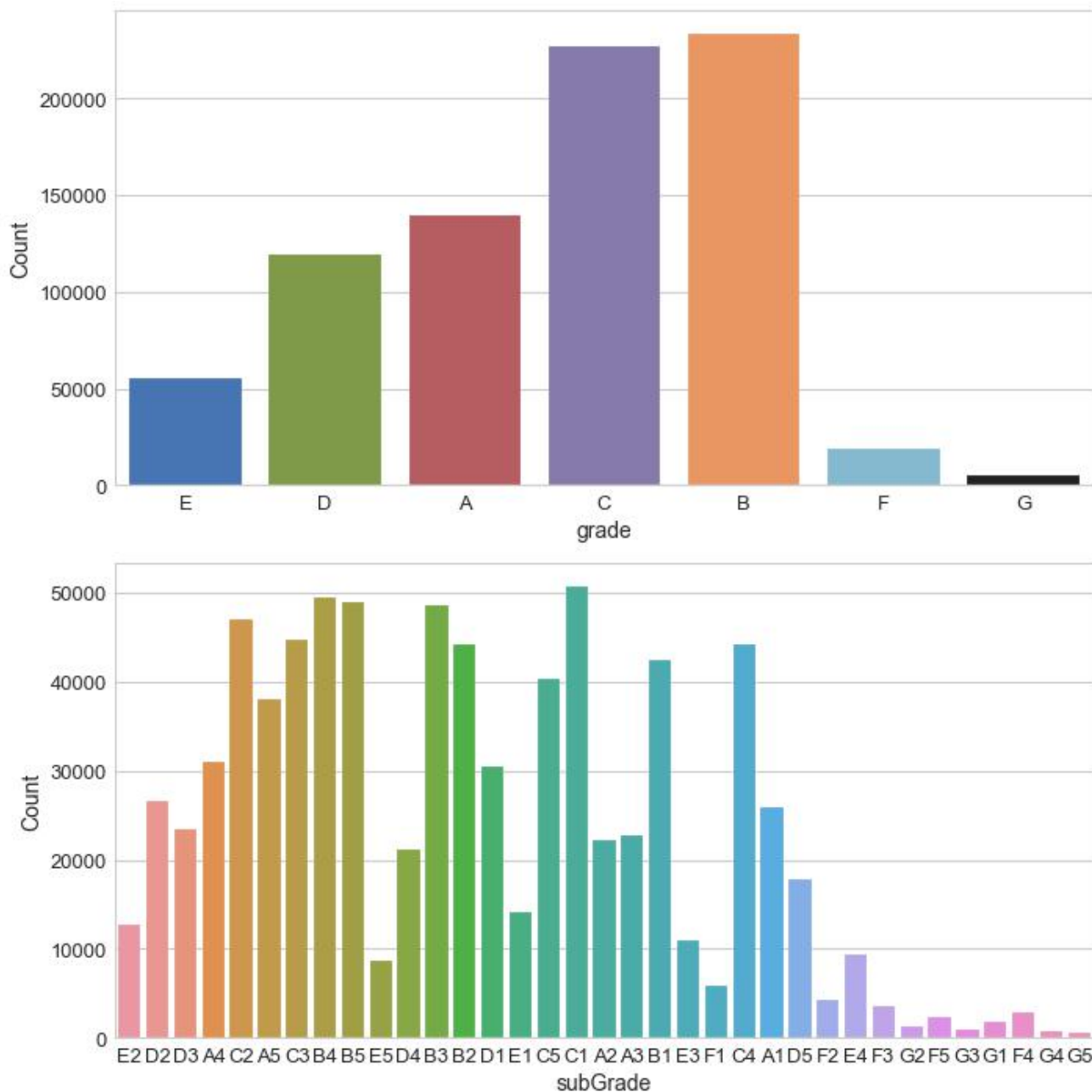
```
1 unique_feature  
  
['policyCode']
```

## 2.2、分类型特征

```
1 # 分类型特征  
2 category_feature = list(filter(lambda x: x not in numerical_feature, list(train.columns)))  
3 category_feature  
  
['grade', 'subGrade', 'employmentLength', 'issueDate', 'earliesCreditLine']
```

这里 "grade" 为贷款等级，"subGrade" 为贷款等级之子级，"employmentLength" 为就业年限，"issueDate" 为贷款发放的月份，"earliesCreditLine" 为借款人最早报告的信用额度开立的月份，共有 5 个分类型特征。

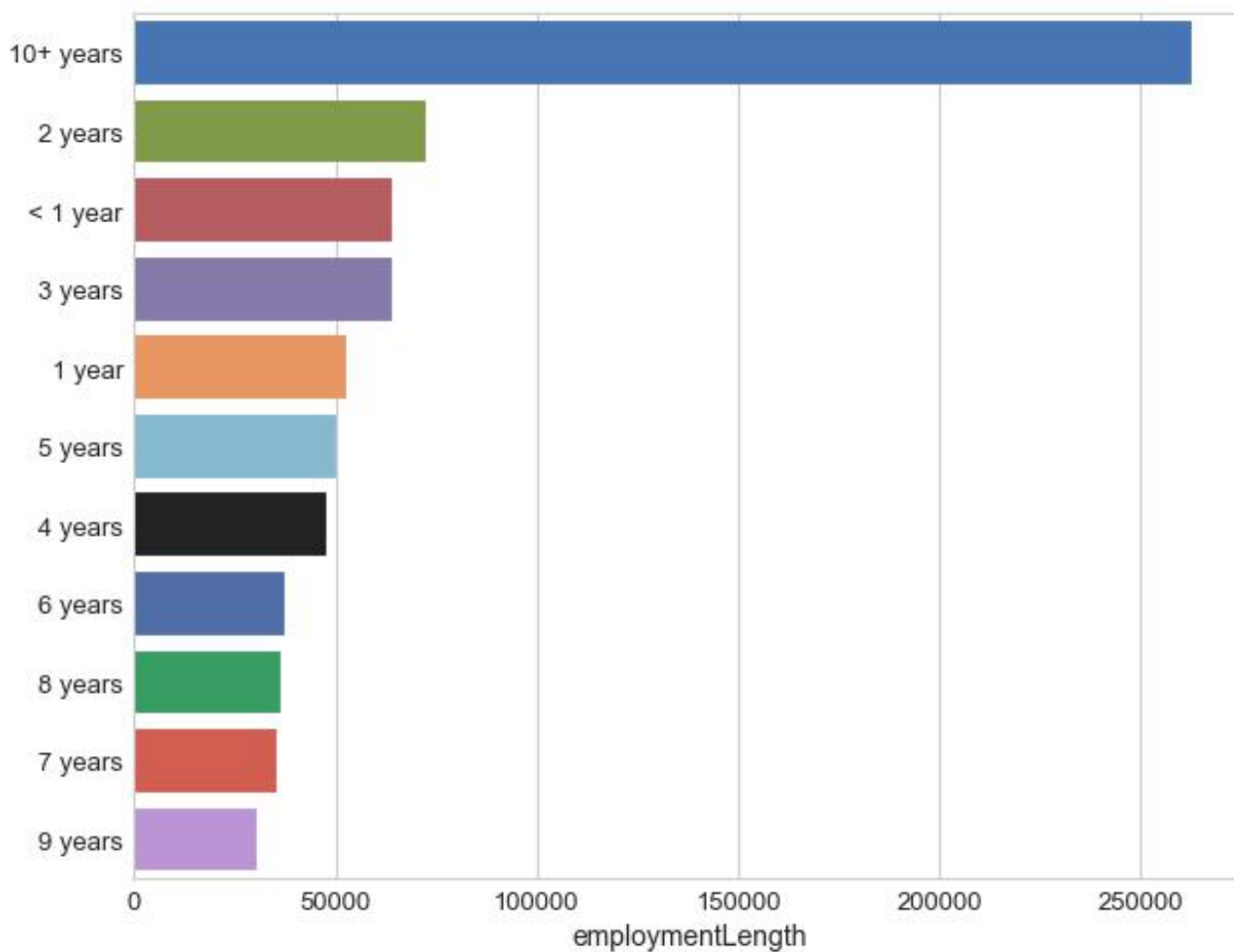
- 分类型特征可视化呈现：



可以看出对于 grade 特征中 A\B\C 等级的贷款占比比较大。

- employmentLength 就业年限可视化呈现



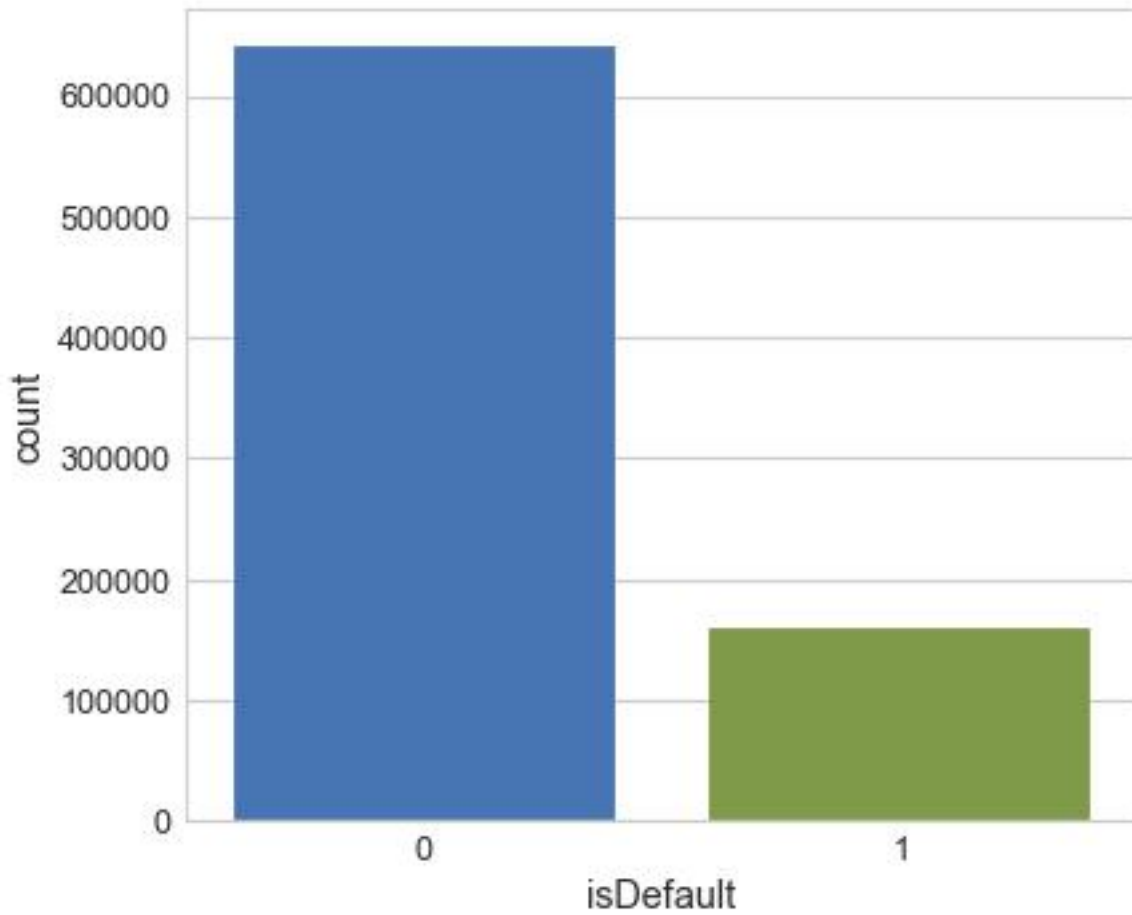


可以看到，就业年限最多是 10+year。

### 2.3、目标变量（标签 y）的分布

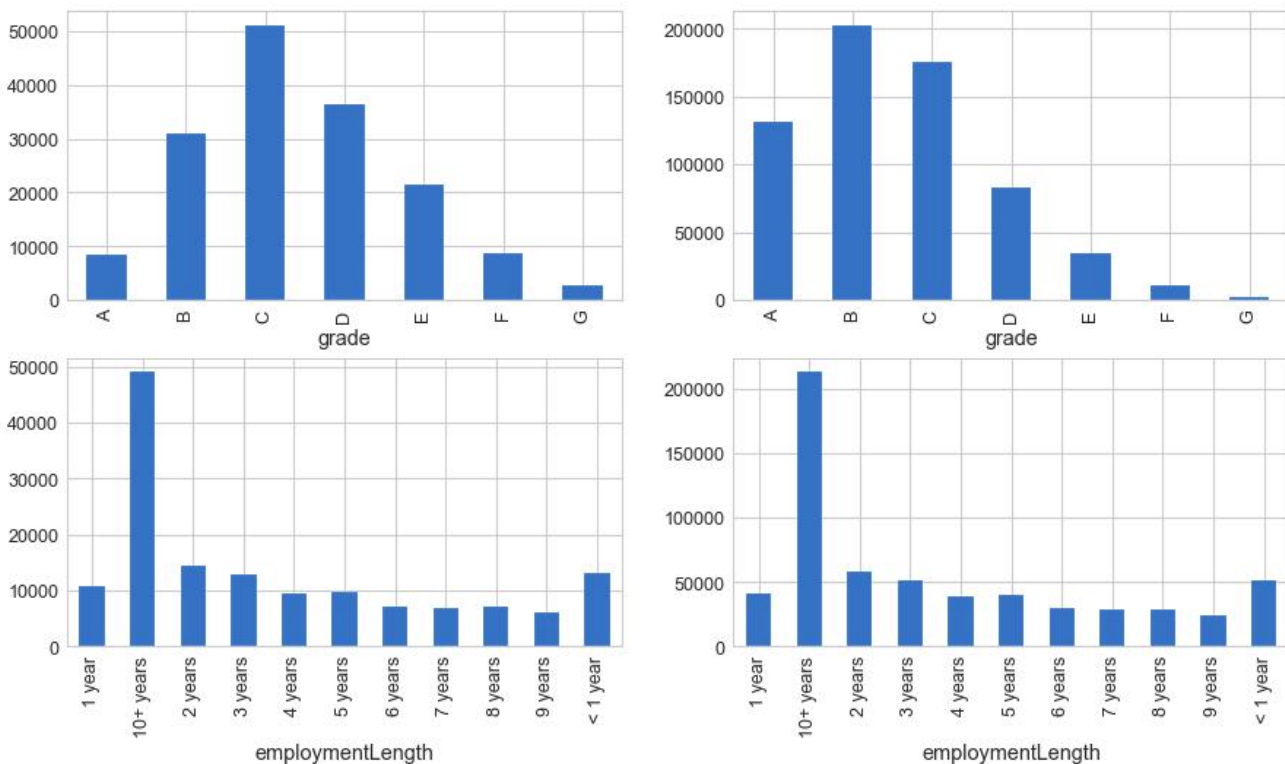
- 查看目标变量（标签）是否平衡。

若分类问题中各类别样本数量差距太大，则会造成样本不均衡的问题。样本不均衡不利于建立与训练出正确的模型，且不能做出合理的评估。



可以看到，贷款违约与不违约的比例大约为 1：4，样本较不平衡，这是金融风控模型评估的中常见的现象，大多数的人都是不会拖欠贷款的。

- 目标变量和分类类别之间的分布关系



### 2.4、缺失值查看

如果缺失值过多会对整体的模型结果产生一定的影响，因此每次在建模之前都需要对数据的缺失值情况就行查看，若有缺失情况，需要在后续特征工程中进行填补。

#### ● 缺省值查看



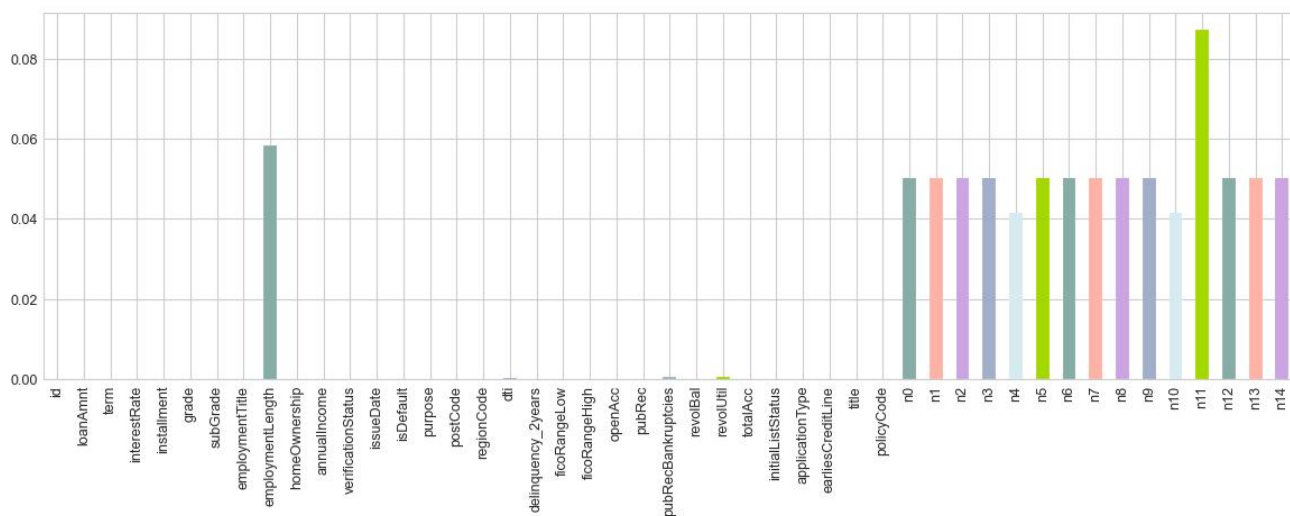
	特征	缺失值个数	缺失比例
7	employmentTitle	1	0.000001
8	employmentLength	46799	0.058499
14	postCode	1	0.000001
16	dti	239	0.000299
22	pubRecBankruptcies	405	0.000506
24	revolUtil	531	0.000664
29	title	1	0.000001
31	n0	40270	0.050338
32	n1	40270	0.050338
33	n2	40270	0.050338
34	n3	40270	0.050338
35	n4	33239	0.041549
36	n5	40270	0.050338
37	n6	40270	0.050338

38	n7	40270	0.050338
39	n8	40271	0.050339
40	n9	40270	0.050338
41	n10	33239	0.041549
42	n11	69752	0.087190
43	n12	40270	0.050338
44	n13	40270	0.050338
45	n14	40270	0.050338

可以看到 employmentTitle、employmentLength、dti 以及匿名特征等字段存在缺省值。从上面的结果可以看出 train 数据集中的 47 个字段有 22 个存在缺省值的情况。

- 可视化一下缺省值数量占比。

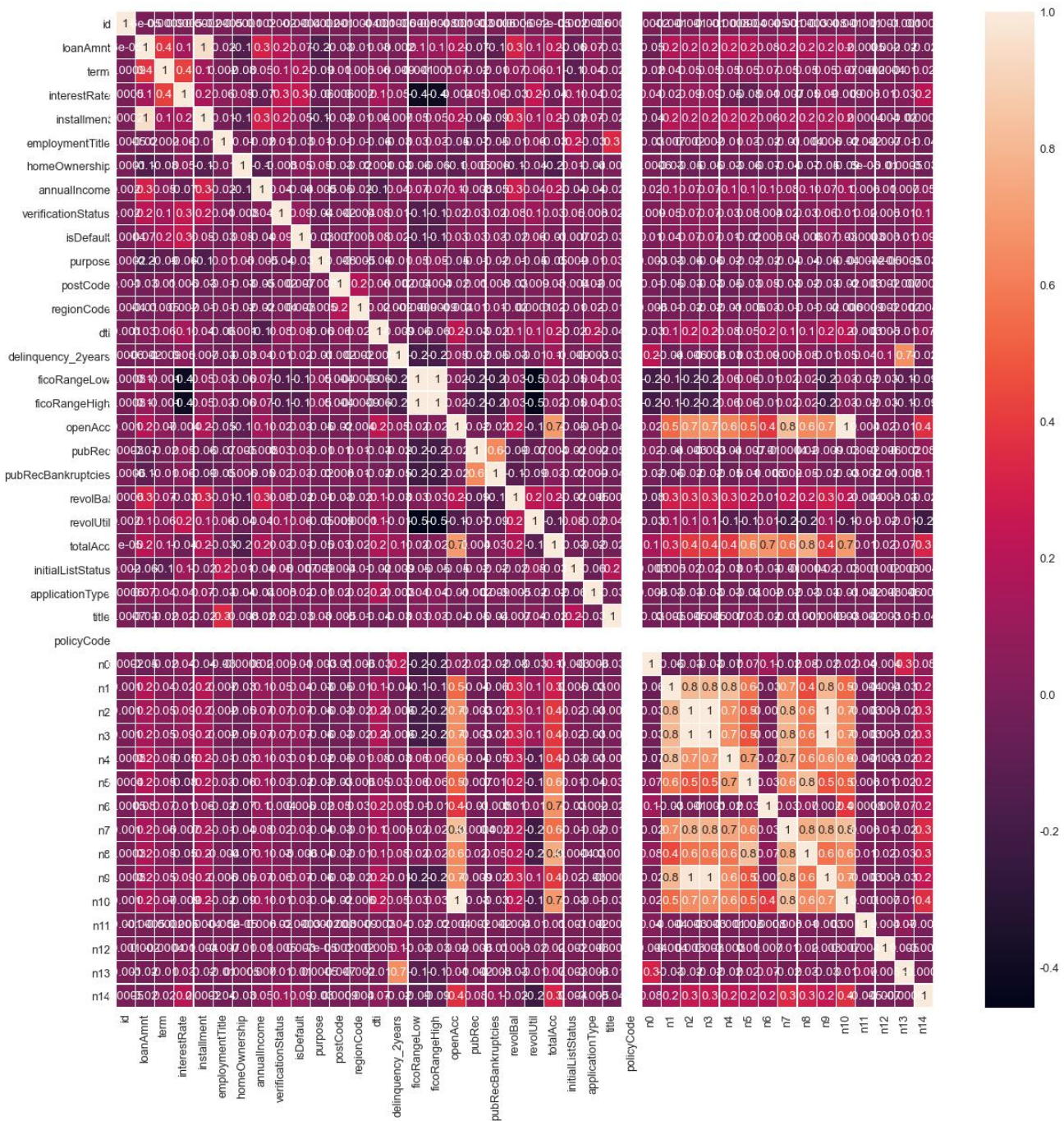
一般对于缺失值，需要进行横纵对比。纵向（从列方向）：如果 nan 存在的过多，说明这一列对 label 的影响几乎不起作用了，可以考虑删掉。如果缺失值很小一般可以选择填充。比如占到总数的 50%，理论上对分析作用不大，这样就可以省略该字段。横向（从行方向）：如果在数据集中，某些样本数据的大部分列都是缺失的且样本足够的情况下可以考虑删除。



可以看到，所有的特征缺失值都在 10%以内，这里考虑全部保留。

## 2.5、数据相关关系

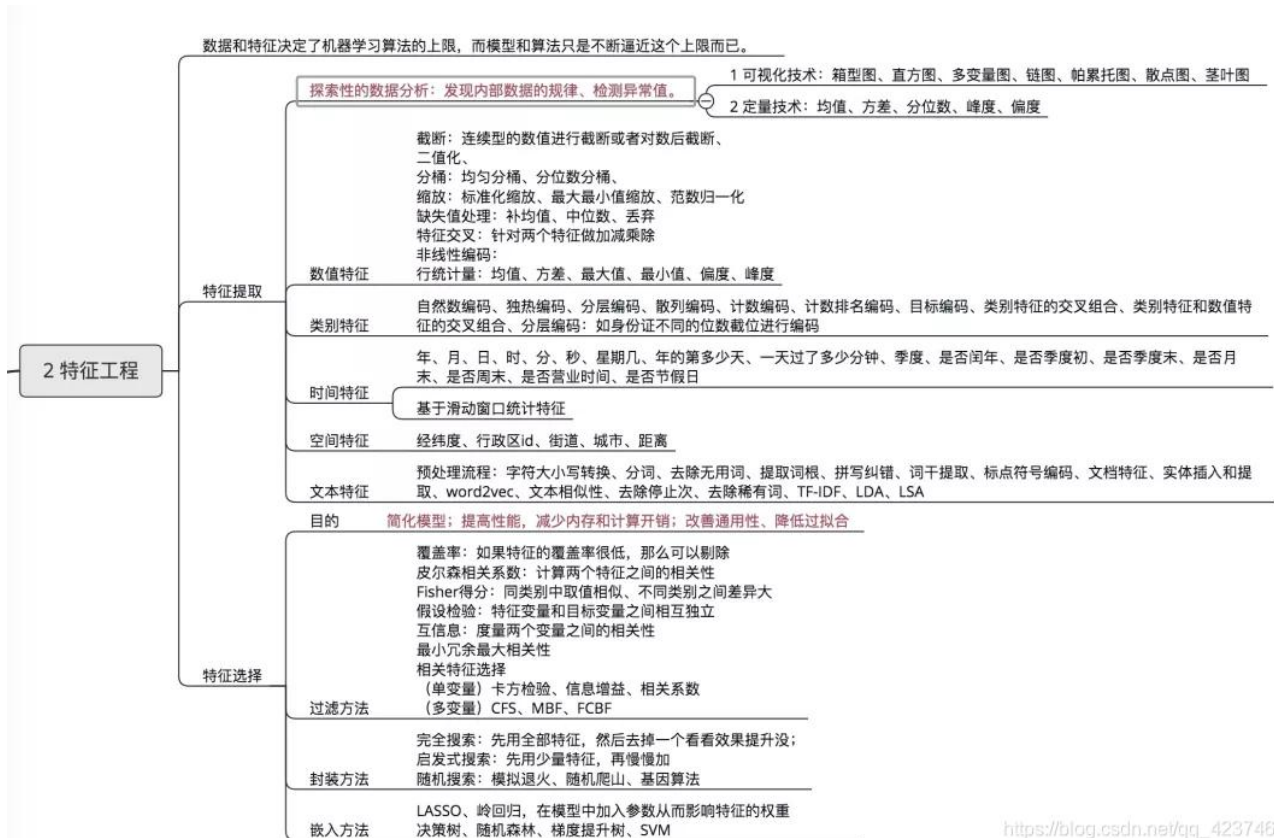




可以看到，有些变量之间的相关性还是很强的，比如贷款总额 `loanAmnt` 和分期付款金额 `installment` 相关性为 1，`ficoRangeLow` 和 `ficoRangeHigh` 相关性为 1...，这种情况后面再特征选择时考虑删除。

## 六、特征工程

基本的 EDA 探索完成后，就可以进行特征工程，在数据挖掘中，大部分时间都是在做特征工程特征工程包括数据预处理、缺失值以及异常值的处理、数据分桶处理以及特征交互、编码、选择。



在特征工程中我走了很多弯路，原因就如前面所说的数据处理和特征选择地过于复杂，导致数据过于干净，使得用于预测的特征偏离了实际，导致分数不高。下面我将会从一开始的处理开始描述，将从头至尾地描述出我在这部分进行的具体操作。

### ➤ 开始时的工作

#### 1. 重复值处理

```
1 train.duplicated().sum()
```





## 2. 缺失值填补

用户在登录界面输入用户名和密码并经过系统验证后，如果没有被管理员封禁，则可以进入主服务界面进行相关操作（新用户注册后使用）。

传统地，如果是分类型特征，采用众数进行填补。如果是连续型特征，采用均值进行填补。还要考虑，均值一般适用于近似正态分布数据，观测值较为均匀散布均值周围；中位数一般适用于偏态分布或者有离群点数据，中位数是更好地代表数据中心趋势；众数一般用于类别变量，无大小、先后顺序之分。所以对于连续变量对于数据近似符合正态分布，用该变量的均值填补缺失。对于数据存在偏态分布的情况，采用中位数进行填补。

### ● 数值型特征用中位数填补：

```
# 训练集
train[numerical_feature] = train[numerical_feature].fillna(train[numerical_feature].median())
# 测试集
test[numerical_feature] = test[numerical_feature].fillna(train[numerical_feature].median())
```

### ● 分类型特征用众数填补：

```
# 训练集
train[category_feature] = train[category_feature].fillna(train[category_feature].mode())
# 测试集
test[category_feature] = test[category_feature].fillna(train[category_feature].mode())
```

### ● 之后 employmentLength 列还存在缺失值，采用决策树来填补

```
from sklearn.tree import DecisionTreeClassifier

empLenNotNullInd = train.employmentLength.notnull() # 不是空的行，返回 True
columns = ['postCode','regionCode','employmentTitle','annualIncome'] # 用四个特征来预测 employmentLength
train_empLen_X = train.loc[empLenNotNullInd,columns]
train_empLen_y = train.employmentLength[empLenNotNullInd]

DTC = DecisionTreeClassifier() # 实例化
DTC.fit(train_empLen_X,train_empLen_y) # 训练
print(DTC.score(train_empLen_X,train_empLen_y))
# 预测
for data in [train,test]:
```



```
empLenIsNullInd = data.employmentLength.isnull()
test_empLen_X = data.loc[empLenIsNullInd, columns]
empLen_pred = DTC.predict(test_empLen_X)
data.employmentLength[empLenIsNullInd] = empLen_pred
```

填补完毕:

```
1 train.isnull().any().sum()
```

```
0
```

### 3. 异常值处理

异常值的存在很可能会影响模型的最终结果，但是当我们发现异常值的时候也不能马上就删除，应该先看看这个异常值是不是有特殊原因造成的，特别是在金融风控问题中，异常值的出现往往是存在意义的。

如果不是因为特殊原因造成的，可以先观察这个异常值出现的频率。若异常值只出现了一次，多半是输入错误，直接把异常值删除即可；若异常值出现了多次，可以和业务人员沟通，可能这是某种特殊表示，如果是人为造成的错误，留着是没有用，只要数据量不是太大，都可以删除；若异常值占到总数据量的 10% 以上，不能轻易删除。可以考虑把异常值替换成非异常但是非干扰的项，比如说用 0 来进行替换，或者把异常当缺失值，用均值或者众数来进行替换。通常，在进行 EDA 的时候会利用描述统计的方法，查看特征的均值、极大值、极小值等信息，结合常识来判断是否存在异常值。

#### ● 均方差 $3\sigma$

在统计学中，如果一个数据分布近似正态，那么大约 68% 的数据值会在均值的一个标准差范围内，大约 95% 会在两个标准差范围内，大约 99.7% 会在三个标准差范围内。



```
def find_outliers_by_3segama(data,fea):
    data_std = np.std(data[fea])
    data_mean = np.mean(data[fea])
    outliers_cut_off = data_std * 3
    lower_rule = data_mean - outliers_cut_off
    upper_rule = data_mean + outliers_cut_off
    data[fea+'_outliers'] = data[fea].apply(lambda x:str('异常值') if x > upper_rule or x < lower_rule else '正常值')
    return data
data_train = train.copy()
for fea in numerical_feature:
    data_train = find_outliers_by_3segama(data_train,fea)
    print(data_train[fea+'_outliers'].value_counts())
    print(data_train.groupby(fea+'_outliers')['isDefault'].sum())
    print('*'*10)
```

## 4. 时间数据处理

对于本赛题，时间数据有 `issueDate`，可以将其转化成时间格式（`issueDateDT` 特征表示数据日期离数据集中日期最早的日期（2007-06-01）的天数）。

### ● 训练集时间数据处理

```
import datetime
# 转化成时间格式 issueDateDT 特征表示数据日期离数据集中日期最早的日期（2007-06-01）的天数
train['issueDate'] = pd.to_datetime(train['issueDate'],format='%Y-%m-%d')
startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
train['issueDateDT'] = train['issueDate'].apply(lambda x: x-startdate).dt.days
```

	issueDate	issueDateDT
0	2014-07-01	2587
1	2012-08-01	1888
2	2015-10-01	3044
3	2015-08-01	2983
4	2016-03-01	3196
...	...	...
799995	2016-07-01	3318
799996	2013-04-01	2131
799997	2015-10-01	3044
799998	2015-02-01	2802
799999	2018-08-01	4079
800000 rows × 2 columns		

## ● 训练集时间数据处理

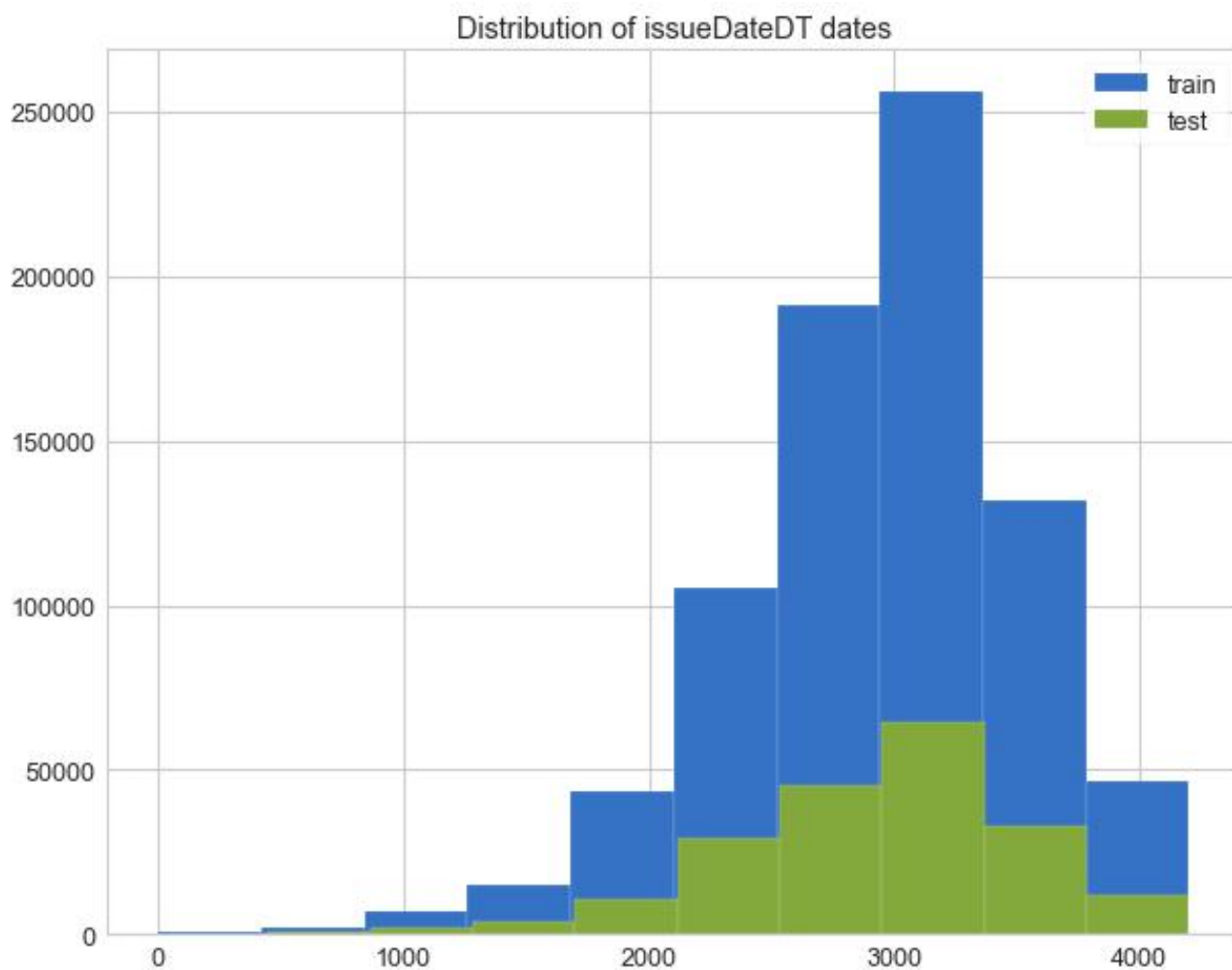
#转化成时间格式

```
test['issueDate'] = pd.to_datetime(train['issueDate'],format='%Y-%m-%d')
```

```
startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
```

```
test['issueDateDT'] = test['issueDate'].apply(lambda x: x-startdate).dt.days
```

## ● 可视化



## 5. 特征交叉

表示特征之间的相互作用，再线性模型中引入非线性性质，提升模型表达能力。

将 issueDate 贷款发放时的年份减去借款人最早报告的信用额度开立的年份，得到新的特征，即开卡年限 CreditLine。

```
train_earliesCreditLine_year = train['earliesCreditLine'].apply(lambda x:x[-4:]).astype('int64')
```



```
test_earliesCreditLine_year = test['earliesCreditLine'].apply(lambda x:x[-4:]).astype('int64')
```

```
train_issueDate_year = train['issueDate'].astype('str').apply(lambda x:x[:4]).astype('int64')
```

```
test_issueDate_year = test['issueDate'].astype('str').apply(lambda x:x[:4]).astype('int64')
```

```
train['CreditLine'] = train_issueDate_year - train_earliesCreditLine_year
```

```
test['CreditLine'] = test_issueDate_year - test_earliesCreditLine_year
```

```
train = train.drop(['earliesCreditLine','issueDate'],axis=1)
```

```
test = test.drop(['earliesCreditLine','issueDate'],axis=1)
```

```
1 train['CreditLine']
```

```
0      13
```

```
1      10
```

```
2       9
```

```
3      16
```

```
4      39
```

```
..
```

```
799995    5
```

```
799996   24
```

```
799997   13
```

```
799998   21
```

```
799999   16
```

```
Name: CreditLine, Length: 800000, dtype: int64
```

## 6. 特征编码

对类别型特征进行转换，使其变为数值特征。具体有以下几种方法：序号编码：适用于类别间存在大小关系的特征。比如级别高中低，可以对应 321；oneHot 编码：适用于不具有大小关系的特征。比如地名；二进制编码：先给每个类别赋予一个序号 ID，然后对 ID 进行二进制编码，最终得到和 OneHot 类似的 0-1 向量，但是维度更小。

首先将 employmentLength 进行简单的处理，再进行编码



这里将就业年限特征转换为数值（把数字后面的 years 去掉并且把 10+ 改成 10，

<1 改成 0）：

```
def employmentLength_to_int(s):
    if pd.isnull(s):
        return s
    else:
        return np.int8(s.split()[0])
for data in [train, test]:
    data['employmentLength'].replace(to_replace='10+ years', value='10 years', inplace=True)
    data['employmentLength'].replace('< 1 year', '0 years', inplace=True)
    data['employmentLength'] = data['employmentLength'].apply(employmentLength_to_int)
```

```
0      2
1      5
2      8
3     10
4      5
5      7
6      9
7      1
8      5
9      6
10     10
11      3
12      2
13     10
14      2
15      2
16      9
17      0
18     10
19      9
Name: employmentLength, dtype: int64
```

- 接下来，对其余分类型特征进行编码，像等级 grade、subGrade 这种类别特征，虽然是表示类别的数据，但是信用评级是有高低大小之分的，是有优先级的，所以可以直接自映射，转化为数值类型。

```
a2z = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
a2z_code = np.arange(1,27)
```



```
a2z_mapping = dict(zip(a2z, a2z_code))
```

```
for data in [train,test]:
```

```
    data.loc[:,['grade','subGrade']] = data.loc[:,['grade','subGrade']].applymap(lambda g:g.replace(g[0],
str(a2z.index(g[0])+1))).astype('int')
```

	grade	subGrade
0	5	52
1	4	42
2	4	43
3	1	14
4	3	32
...	...	...
799995	3	34
799996	1	14
799997	3	33
799998	1	14
799999	2	23
800000 rows × 2 columns		

- 对于离散型特征，可以使用 OneHotEncoder 独热编码

```
from sklearn.preprocessing import OneHotEncoder
```

```
oh = OneHotEncoder(sparse=False)
```

```
oh.fit(train[['homeOwnership','verificationStatus','purpose']])
```

```
OneHot1 = oh.transform(train[['homeOwnership','verificationStatus','purpose']])
```

```
OneHot2 = oh.transform(test[['homeOwnership','verificationStatus','purpose']])
```

```
OneHot1.shape
```

```
train = pd.concat([train, pd.DataFrame(OneHot1)], axis=1)
```

```
test = pd.concat([test, pd.DataFrame(OneHot2)], axis=1)
```

```
train = train.drop(['homeOwnership','verificationStatus','purpose'],axis=1)
```

```
test = test.drop(['homeOwnership','verificationStatus','purpose'],axis=1)
```

```
train.shape
```

## 7. 特征选择





- 人工判断与目标无关联特征为"id"，需删除

```
train=train.drop(["id"],axis=1)
train.shape
test=test.drop(["id"],axis=1)
test.shape
```

- 求出各个特征与目标的相关系数，综合考虑排除 corr 小于 0.01 的特征

```
1 train.corr()["isDefault"].sort_values()
```

```
ficoRangeLow      -0.130994
ficoRangeHigh     -0.130993
6                 -0.085972
0                 -0.069264
annualIncome      -0.042782
...
interestRate      0.259202
grade             0.261858
subGrade          0.266415
isDefault         1.000000
policyCode        NaN
Name: isDefault, Length: 66, dtype: float64
```

```
1 train=train.drop(["initialListStatus","n5","n11","n12","n8","postCode","policyCode"],axis=1)
2 test=test.drop(["initialListStatus","n5","n11","n12","n8","postCode","policyCode"],axis=1)
3
4 train.shape
```

```
(800000, 59)
```

- 特征间高相关过滤

# 显示相关性高于 0.6 的变量

```
def getHighRelatedFeatureDf(corr_matrix, corr_threshold):
```

```
    highRelatedFeatureDf = pd.DataFrame(corr_matrix[corr_matrix>corr_threshold].stack().reset_index())
```

```
    highRelatedFeatureDf.rename({'level_0':'feature_x', 'level_1':'feature_y', 0:'corr'}, axis=1, inplace=True)
```

```
    highRelatedFeatureDf = highRelatedFeatureDf[highRelatedFeatureDf.feature_x != highRelatedFeatureDf.feature_y]
```

```
    highRelatedFeatureDf['feature_pair_key'] = highRelatedFeatureDf.loc[:,['feature_x', 'feature_y']].apply(lambda
```

```
r:'#'.join(np.sort(r.values)), axis=1)
```

```
    highRelatedFeatureDf.drop_duplicates(subset=['feature_pair_key'],inplace=True)
```



```
highRelatedFeatureDf.drop(['feature_pair_key'], axis=1, inplace=True)
return highRelatedFeatureDf
```

```
getHighRelatedFeatureDf(train.corr(),0.6)
```

"loanAmnt" 贷款金额, "installment" 分期付款金额两个特征间相关系数为 0.95;  
 "ficoRangeLow" fico 所属的下限范围, "ficoRangeHigh" fico 所属的上限范围两个特征间相关系数为 1; "openAcc" 未结信用额度的数量, "n10" 两个特征间相关系数为 0.93; "n3", "n2" 两个特征间相关系数为 1; "n3", "n9" 两个特征间相关系数为 0.98。根据高相关特征, 综合考虑他们与目标的相关性, 删除特征 "installment", "ficoRangeHigh", "openAcc", "n3", "n9"。

## ● 低方差过滤

```
train.var().sort_values()
col = ['applicationType']
for data in [train,test]:
    data.drop(col,axis=1,inplace=True)
```

## ➤ 最终进行的处理

### ● 导入模块和数据:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import auc, roc_curve
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler
import datetime
np.random.seed(2020)

train_file_name = 'G://classStudy/II 2/机器学习/大作业/贷款违约预测/train.csv'
test_file_name = 'G://classStudy/II 2/机器学习/大作业/贷款违约预测/testA.csv'

df_train = pd.read_csv(train_file_name)
df_test = pd.read_csv(test_file_name)
```

### 1. 对训练集进行处理

- 对 grade 进行特征编码

因为 grade 特征，虽然是表示类别的数据，但是信用评级是有高低大小之分的，是有优先级的，所以也可以使用 labelencode 编码。

```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
df_train['grade'] = labelEncoder.fit_transform(df_train['grade'])
df_train['grade'] = labelEncoder.fit_transform(df_train['grade'])
```

```
1 df_train['grade']
```

0	4
1	3
2	3
3	0
4	2
	..
799995	2
799996	0
799997	2
799998	0
799999	1

Name: grade, Length: 800000, dtype: int64

- 对 employmentLength 进行自然映射，将其从分类型变为数值型

```
employmentLength = ['< 1 year', '1 year', '2 years',  
                    '3 years', '4 years', '5 years',  
                    '6 years', '8 years', '7 years', '9 years', '10+ years']  
  
j = 0  
for i in employmentLength:  
    df_train['employmentLength'] = df_train['employmentLength'].replace(i, j)  
    j += 1
```

- 对 subGrade 与目标进行特征交互和特征编码

```
_ = pd.crosstab(df_train.subGrade, df_train.isDefault)  
_["yp"] = _[1]/(_[0]+_[1])  
_.reset_index(inplace=True)
```



```
_.sort_values(by="yp", inplace=True)
df_train = pd.merge(df_train, _[["subGrade", "yp"]], on="subGrade", how="left")
df_train['subGrade'] = labelEncoder.fit_transform(df_train['subGrade'])
```

“yp”为不同 subGrade 中正例所占的比例，能够帮助后面更好地训练模型

isDefault	subGrade	0	1	yp
0	A1	25082	827	0.031919
1	A2	21113	1011	0.045697
2	A3	21389	1266	0.055882
3	A4	28849	2079	0.067221
4	A5	34796	3249	0.085399
5	B1	38020	4362	0.102921
6	B2	39262	4965	0.112262
7	B3	42319	6281	0.129239
8	B4	42156	7360	0.148639
9	B5	40854	8111	0.165649
10	C1	41049	9714	0.191360
11	C2	37330	9738	0.206892
12	C3	34701	10050	0.224576

### ● 对于 issueDate、issueDateDT、earliesCreditLine 等时间信息处理

```
df_train['issueDate'] = pd.to_datetime(df_train['issueDate'], format='%Y-%m-%d')
startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
df_train['issueDateDT'] = df_train['issueDate'].apply(lambda x: x-startdate).dt.days
df_train['earliesCreditLine'] = df_train['earliesCreditLine'].apply(lambda s: int(s[-4:]))
```

‘issueDate’使用年月日的格式，‘issueDateDT’为数据时间减去最早时间，‘earlierCreditLine’保留年份。在实际生活中，借款的时间是衡量违约的一重要因素，所以时间这一特征很重要，在一开始的特征工程中，我未保留月日的信息，从而导致预测结果不好。

## 2. 对于测试集进行和训练集一样的特征工程处理



```
df_test['grade'] = labelEncoder.fit_transform(df_test['grade'])
employmentLength = ['< 1 year','1 year','2 years',
                    '3 years', '5 years', '4 years',
                    '6 years', '8 years', '7 years','9 years','10+ years']

j = 0
for i in employmentLength:
    df_test['employmentLength'] = df_test['employmentLength'].replace(i, j)
    j += 1
    df_test = pd.merge(df_test, _[["subGrade", "yp"]], on="subGrade", how="left")
    df_test['subGrade'] = labelEncoder.fit_transform(df_test['subGrade'])
    df_test['issueDate'] = pd.to_datetime(df_test['issueDate'],format='%Y-%m-%d')
    startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
    df_test['issueDateDT'] = df_test['issueDate'].apply(lambda x: x-startdate).dt.days
    df_test['earliesCreditLine'] = df_test['earliesCreditLine'].apply(lambda s: int(s[-4:]))
```

## 七、建模分析

在完成相关的特征处理后，接下来进行建模分析，通过调节参数得到性能更强的模型，再进行模型融合，模型融合是比赛后期上分的重要手段，模型融合后结果会有大幅提升。

## 1. 模型考虑

### ● Xgboost

GBDT 算法只利用了一阶的导数信息，xgboost 对损失函数做了二阶的泰勒展开，并在目标函数之外加入了正则项对整体求最优解，用以权衡目标函数的下降和模型的复杂程度，避免过拟合。xgboost 对应的模型就是一堆 CART 树。主要思想就是弱分类器，一起组合成一个强分类器。

一开始树是 0，然后往里面加树，相当于多了一个函数，再加第二棵树，相当于又多了一个函数…等等，这里需要保证加入新的函数能够提升整体对表达效果。提升表达效果的意思就是说加上新的树之后，目标函数的值会下降。如果叶子结点的个数太多，那么过拟合的风险会越大，所以这里要限制叶子结点的个数，所以在原来目标函数里要加上一个惩罚项。

Obj 代表了当我们指定一个树的结构的时候，在目标上最多会减少多少，我们可以把它叫做结构分数，这个分数越小越好。

xgboost 的优势：正则化——减少过拟合；并行处理——提升速度；高度灵活性——允许用户定义自定义优化目标和评价标准；缺失值处理——内置处理缺失值的规则；内置交叉验证 XGBoost 允许在每一轮 boosting 迭代中使用交叉验证；在已有的模型基础上继续——可以在上一轮的结果上继续训练；剪枝——XGBoost 会一直分裂到指定的最大深，然后回过头来剪枝。如果某个节点之后不再有正值，它会去除这个分裂。

### ● Lightgbm

Xgboost 通过预排序能精确地找到分割点。但是缺点也很明显：空间消耗大、时间上也有较大的开销，在遍历每一个分割点的时候，都需要进行分裂增益的计





算，消耗的代价大。

LightGBM 是一个实现 GBDT 算法的框架，支持高效率的并行训练，并且具有更快的训练速度、更低的内存消耗、更好的准确率、支持分布式可以快速处理海量数据等优点。LightGBM 提出的主要原因就是为了解决 GBDT 在海量数据遇到的问题。本题 120 万条数据就比较符合这一点。

优点：速度更快——采用了直方图算法将遍历样本转变为遍历直方图，极大的降低了时间复杂度；在训练过程中采用单边梯度算法过滤掉梯度小的样本，减少了大量的计算；采用了基于 Leaf-wise 算法的增长策略构建树，减少了很多不必要的计算量；采用优化后的特征并行、数据并行方法加速计算，当数据量非常大的时候还可以采用投票并行的策略；对缓存也进行了优化，增加了缓存命中率；内存更小——采用了直方图算法将存储特征值转变为存储 bin 值，降低了内存消耗；在训练过程中采用互斥特征捆绑算法减少了特征数量，降低了内存消耗。

## ● Catboost

CatBoost 是一种基于对称决策树为基学习器实现的参数较少、支持类别型变量和高准确性的 GBDT 框架，主要解决的痛点是高效合理地处理类别型特征，这一点从它的名字中可以看出，CatBoost 是由 Categorical 和 Boosting 组成。此外，CatBoost 还解决了梯度偏差以及预测偏移的问题，从而减少过拟合的发生，进而提高算法的准确性和泛化能力。

与 XGBoost、LightGBM 相比，CatBoost 的创新点有：嵌入了自动将类别型特征处理为数值型特征的创新算法。首先对计算某个类别特征出现的频率，之后加上超参数，生成新的数值型特征。这一点非常适合本实验的这些特征，这使得不用



对于分类特征做过多的处理，从而避免了数据处理地过于干净。Catboost 还使用了组合类别特征，可以利用到特征之间的联系，这极大的丰富了特征维度。采用排序提升的方法对抗训练集中的噪声点，从而避免梯度估计的偏差，进而解决预测偏移的问题。

其有点为：性能卓越——在性能方面可以匹敌任何先进的机器学习算法；鲁棒性——它减少了对很多超参数调优的需求，并降低了过度拟合的机会，这也使得模型变得更加具有通用性；易于使用——提供与 scikit 集成的 Python 接口，以及 R 和命令行界面；实用——可以处理类别型、数值型特征；可扩展——支持自定义损失函数。

## ● 模型融合之 stacking

stacking 构建多层模型，并利用预测结果再拟合预测。Stacking 将若干基学习器获得的预测结果作为新的训练集训练一个学习器。但是由于直接由多个基学习器获得结果直接带入模型中，容易导致过拟合。所以在使用多个基模型进行预测时，可以考虑 K 折验证，防止过拟合。Stacking 中由于两层使用的数据不同，所以可以避免信息泄露的问题。

## ● 模型融合之 Voting

Voting 针对分类问题，按照少数服从多数的方法进行投票，比较符合本题。

模型1 A-99%； B-1%

模型2 A-49%； B-51%

模型3 A-40%； B-60%

模型4 A-90%； B-10%

模型5 A-30%； B-70%




A-两票； B-三票

最终结果为B

Hard Voting

[https://blog.csdn.net/qq\\_42374697](https://blog.csdn.net/qq_42374697)



模型1 A-99%; B-1%		$A - (0.99 + 0.49 + 0.4 + 0.9 + 0.3) / 5$
模型2 A-49%; B-51%		$= 0.616$
模型3 A-40%; B-60%		$B - (0.01 + 0.51 + 0.6 + 0.1 + 0.7) / 5$
模型4 A-90%; B-10%		$= 0.384$
模型5 A-30%; B-70%		最终结果为A

## 2. 尝试过程

### ➤ 使用开始时处理后的数据进行建模

#### ● Lightgbm

```
X = train.drop(['isDefault'], axis=1)
y = train.loc[:, 'isDefault']

kf = KFold(n_splits=5, shuffle=True, random_state=525)
X_train_split, X_val, y_train_split, y_val = train_test_split(X, y, test_size=0.2)
import lightgbm as lgb
cv_scores = []
for i, (train_index, val_index) in enumerate(kf.split(X, y)):
    X_train, y_train, X_val, y_val = X.iloc[train_index], y.iloc[train_index], X.iloc[val_index], y.iloc[val_index]

    train_matrix = lgb.Dataset(X_train, label=y_train)
    valid_matrix = lgb.Dataset(X_val, label=y_val)

    params = {
        'boosting_type': 'gbdt',
        'objective': 'binary',
        'learning_rate': 0.1,
        'metric': 'auc',
        'min_child_weight': 1e-3,
        'num_leaves': 31,
        'max_depth': -1,
        'seed': 525,
        'nthread': 8,
        'silent': True,
    }

    model = lgb.train(params, train_set=train_matrix, num_boost_round=20000, valid_sets=valid_matrix, verbose_eval=1000,
```



```
early_stopping_rounds=200)
val_pred = model.predict(X_val, num_iteration=model.best_iteration)

cv_scores.append(roc_auc_score(y_val, val_pred))
print(cv_scores)

print("lgb_scotrainre_list: {}".format(cv_scores))
print("lgb_score_mean: {}".format(np.mean(cv_scores)))
print("lgb_score_std: {}".format(np.std(cv_scores)))
from sklearn import metrics
from sklearn.metrics import roc_auc_score

al_pre_lgb = model.predict(X_val, num_iteration=model.best_iteration)
fpr, tpr, threshold = metrics.roc_curve(y_val, val_pred)
roc_auc = metrics.auc(fpr, tpr)
print('AUC: {}'.format(roc_auc))

plt.figure(figsize=(8, 8))
plt.title('Validation ROC')
plt.plot(fpr, tpr, 'b', label = 'Val AUC = %0.4f' % roc_auc)
plt.ylim(0,1)
plt.xlim(0,1)
plt.legend(loc='best')
plt.title('ROC')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
# 画出对角线
plt.plot([0,1],[0,1], 'r--')
plt.show()
```

本机 AUC 得分为 0.7338

## ● Xgboost

```
X = train.drop(['isDefault'], axis=1)
y = train.loc[:, 'isDefault']

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, y, test_size=0.3)
from xgboost.sklearn import XGBClassifier
clf1 = XGBClassifier(n_jobs=-1)
clf1.fit(Xtrain, Ytrain)
```



```
clf1.score(Xtest,Ytest)
from sklearn.metrics import roc_curve, auc

predict_proba = clf1.predict_proba(Xtest)
false_positive_rate, true_positive_rate, thresholds = roc_curve(Ytest, predict_proba[:,1])
auc(false_positive_rate, true_positive_rate)
```

本机计算模型结构的 AUC 面积 0.7326304866618416

进行网格调参找更好的参数

```
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import GridSearchCV

# 其余参数
other_params = {'learning_rate': 0.1,
                 'n_estimators': 100,
                 'max_depth': 5,
                 'min_child_weight': 1,
                 'seed': 0,
                 'subsample': 0.8,
                 'colsample_bytree': 0.8,
                 'gamma': 0,
                 'reg_alpha': 0,
                 'reg_lambda': 1}

# 待调参数
param_test1 = {
    'max_depth':list(range(4,9,2)),
    'min_child_weight':list(range(1,6,2))
}

xgb1 = XGBClassifier(**other_params)
# 网格搜索
gs1 = GridSearchCV(xgb1,param_test1,cv = 5,scoring = 'roc_auc',n_jobs = -1,verbose=2)
best_model1=gs1.fit(Xtrain,Ytrain)
print('最优参数: ',best_model1.best_params_)
print('最佳模型得分: ',best_model1.best_score_)

other_params = {'learning_rate': 0.1,
                 'n_estimators': 100,
                 'max_depth': 4,
                 'min_child_weight': 5,
                 'seed': 0,
                 'subsample': 0.8,
```



```
'colsample_bytree': 0.8,  
'gamma': 0,  
'reg_alpha': 0,  
'reg_lambda': 1}
```

```
param_test = {  
'gamma':[0,0.05,0.1,0.2,0.3]  
}
```

```
xgb = XGBClassifier(**other_params)  
gs = GridSearchCV(xgb,param_test,cv = 5,scoring = 'roc_auc',n_jobs = -1,verbose=2)  
best_model=gs.fit(Xtrain,Ytrain)  
print('最优参数: ',best_model.best_params_)  
print('最佳模型得分: ',best_model.best_score_)
```

```
other_params = {'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 4, 'min_child_weight': 5, 'seed': 0,  
                'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0, 'reg_alpha': 0, 'reg_lambda': 1}
```

```
param_test = {  
'subsample':[0.6,0.7,0.8,0.9],  
'colsample_bytree':[0.6,0.7,0.8,0.9]  
}
```

```
xgb = XGBClassifier(**other_params)  
gs = GridSearchCV(xgb,param_test,cv = 5,scoring = 'roc_auc',n_jobs = -1,verbose=2)  
best_model=gs.fit(Xtrain,Ytrain)  
print('最优参数: ',best_model.best_params_)  
print('最佳模型得分: ',best_model.best_score_)
```

```
other_params = {'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 4, 'min_child_weight': 5, 'seed': 0,  
                'subsample': 0.7, 'colsample_bytree': 0.7, 'gamma': 0, 'reg_alpha': 0, 'reg_lambda': 1}
```

```
param_test = {  
'reg_alpha': [4,5,6,7],  
'reg_lambda': [0,0.01,0.05, 0.1]  
}
```

```
xgb = XGBClassifier(**other_params)  
gs = GridSearchCV(xgb,param_test,cv = 5,scoring = 'roc_auc',n_jobs = -1,verbose=2)  
best_model=gs.fit(Xtrain,Ytrain)  
print('最优参数: ',best_model.best_params_)  
print('最佳模型得分: ',best_model.best_score_)
```





```
other_params = {'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 4, 'min_child_weight': 5, 'seed': 0,
                'subsample': 0.7, 'colsample_bytree': 0.7, 'gamma': 0, 'reg_alpha': 5, 'reg_lambda': 0.01}
```

```
param_test = {
    'learning_rate': [0.01, 0.05, 0.07, 0.1, 0.2],
    'n_estimators': [100, 200, 300, 400, 500]
}
```

```
xgb = XGBClassifier(**other_params)
gs = GridSearchCV(xgb, param_test, cv = 5, scoring = 'roc_auc', n_jobs = -1, verbose=2)
best_model = gs.fit(Xtrain, Ytrain)
print('最优参数: ', best_model.best_params_)
print('最佳模型得分: ', best_model.best_score_)
```

## 通过调参后的模型

```
from xgboost.sklearn import XGBClassifier
clf = XGBClassifier(
    learning_rate=0.05,
    n_estimators=400,
    max_depth=4,
    min_child_weight=5,
    seed=0,
    subsample=0.7,
    colsample_bytree=0.7,
    gamma=0,
    reg_alpha=5,
    reg_lambda=0.01,
    n_jobs=-1)
```

```
clf.fit(Xtrain, Ytrain)
clf.score(Xtest, Ytest)
from sklearn.metrics import roc_curve, auc
```

```
predict_proba = clf.predict_proba(Xtest)
false_positive_rate, true_positive_rate, thresholds = roc_curve(Ytest, predict_proba[:,1])
auc(false_positive_rate, true_positive_rate)
```

本机 AUC 面积为 0.74512067（本机测试很高，但是线上测试只有 0.71，模型融合之后同样提升不高）





## ● Stacking（模型融合）

之后使用之前的训练的 lgb 和 xgb 模型作为基分类器，逻辑回归作为目标分类器做 stacking。

```
from mlxtend.classifier import StackingClassifier

gra=GradientBoostingClassifier()
xgb=XGBClassifier()
lgb=LGBMClassifier()
lr = LogisticRegression()
scf = StackingClassifier(classifiers=[gra, xgb, lgb],
                        use_probab=True,
                        meta_classifier=lr)

scf.fit(Xtrain,Ytrain)
pre =scf.predict_proba(Xtest)[:,-1]
fpr, tpr, thresholds = roc_curve(Ytest, pre)
score = auc(fpr, tpr)
print(score)
```

日期: 2021-06-07 22:52:10 排名: 无

score: 0.7128

模型融合之后效果也不好，分析出的原因就是前面讨论过的数据太干净的问题，特征偏离了实际，如时间只保留了年份，但是实际违约需要更精确的时间。

## ➤ 改变数据处理方式后的建模

使用特征工程后的特征进行建模：

```
tags = ['loanAmnt', 'term', 'interestRate', 'installment', 'grade',
        'subGrade', 'employmentTitle', 'employmentLength', 'homeOwnership',
        'annualIncome', 'verificationStatus', 'issueDateDT', 'earliesCreditLine',
        'purpose', 'postCode', 'regionCode', 'dti', 'delinquency_2years',
        'ficoRangeLow', 'ficoRangeHigh', 'openAcc', 'pubRec',
        'pubRecBankruptcies', 'revolBal', 'revolUtil', 'totalAcc',
        'initialListStatus', 'applicationType', 'title',
        'n0', 'n1', 'n2', 'n4', 'n5', 'n6', 'n7', 'n8',
```



```
'n9', 'n10', 'n11', 'n12', 'n13', 'n14', 'yp']
```

## ● Lightgbm

```
lgbr = LGBMRegressor(num_leaves=30
                      ,max_depth=10
                      ,learning_rate=0.01
                      ,n_estimators=13000
                      ,subsample_for_bin=5000
                      ,min_child_samples=200
                      ,colsample_bytree=.2
                      ,reg_alpha=.1
                      ,reg_lambda=.1
                      ,seed=2020
                      )
```

参数由网格调参得出

## ● Catboost

```
cat = CatBoostRegressor(depth=9,
                        l2_leaf_reg=1,
                        learning_rate=0.01,
                        eval_metric = 'AUC' ,
                        border_count = 128,
                        bagging_temperature = 0.9 ,
                        n_estimators=16000,
                        early_stopping_rounds=500,
                        subsample = 0.9,
                        random_seed=1,
                        verbose = 0)
```

参数由网格调参得出

## ● Voting（模型融合）

```
from sklearn.ensemble import VotingRegressor
rg_model = VotingRegressor([('lgb', lgbr), ('catboost', cat)],n_jobs=12)
```

使用前面训练出来的两个模型进行建模，这一次分数提升很多。由于能力和时间问题也就没有继续改进。



**日期:** 2021-06-09 09:46:04 **排名:** 无

**score:** 0.7362

## 八、结果

1. 通过最后得出的模型，将测试集带进去训练，并得到最后的提交文件。

```
pre = pd.DataFrame(rg_model.predict(x_),columns=['isDefault'])
results = pd.concat([df_test['id'],pre],axis = 1)
results.to_csv('submit.csv', index=False)
```



```
1 id,isDefault
2 800000,0.06318659442621563
3 800001,0.29879428716544154
4 800002,0.5721188684156882
5 800003,0.32322955493146993
6 800004,0.34252539622339845
7 800005,0.0009191142875585342
8 800006,0.31127322180755945
9 800007,0.051008700149821375
10 800008,0.6104296576088584
11 800009,0.031190520169666225
12 800010,0.3955897817378232
13 800011,0.3531153411352013
14 800012,0.3315535747937265
15 800013,0.20738111241741464
16 800014,0.13159407019488192
17 800015,0.22751125725670696
18 800016,0.2159128282413172
19 800017,0.13977483516563194
20 800018,0.04557731135570997
21 800019,0.2590286451068249
```

2. 线上提交文件后，得到成绩



长期赛: 285 /0.7362

长期赛

正式赛

日期: 2021-06-09 09:46:04 排名: 无  
score: 0.7362

## 九、总结与收获





通过大作业学习到了一个机器学习项目的完整过程，而不是平时实验时对每个部分分开处理，所以会感觉一个完整的项目想要做好需要很多的精力和知识。从开始数据处理的顺利到建模时分数死活提升不了，还有中间调参时的等待，感觉已经不是单纯的大作业了，更像是对我耐心的考验。做完完整的项目后，感觉还是很有成就感的，特别是看见分数提升之后，感觉还是很好的。

这个项目还教会了我，知识和实践还是有很大不同的，就像是学习的数据处理和特征工程的很多方法在该项目中不太实用，实用之后还会导致结果不理想，原因就是该实验是实际的贷款平台得到的数据，所以那些特征很多不需要处理的，过渡处理反而导致过拟合。

总之，收获很多，期望不会挂科!!!

## 十、代码



```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import datetime
import statsmodels.formula.api as smf

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
from lightgbm import LGBMClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.model_selection import cross_val_score

import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import roc_auc_score, accuracy_score, f1_score, log_loss

plt.rcParams["font.sans-serif"]=["SimHei"]#正常显示中文
plt.rcParams["axes.unicode_minus"]=False#正常显示负号

from jupyterthemes import jtplot
jtplot.style(theme='onedork') #选择一个绘图主题

train=pd.read_csv('G://classStudy/II 2/机器学习/大作业/贷款违约预测/train.csv')
test=pd.read_csv('G://classStudy/II 2/机器学习/大作业/贷款违约预测/testA.csv')

train.head()
train.shape
test.shape
train.columns
train.info()
train.describe()
```



```
# 数值类型
numerical_feature = list(train.select_dtypes(exclude=['object']).columns)
numerical_feature
len(numerical_feature)

# 连续型变量
serial_feature = []
# 离散型变量
discrete_feature = []
# 单值变量
unique_feature = []

for fea in numerical_feature:
    temp = train[fea].nunique()# 返回的是唯一值的个数
    if temp == 1:
        unique_feature.append(fea)
        # 自定义变量的值的取值个数小于 10 就为离散型变量
    elif temp <= 10:
        discrete_feature.append(fea)
    else:
        serial_feature.append(fea)
serial_feature
plt.figure(1 , figsize = (8 , 5))
sns.distplot(train.loanAmnt,bins=40)
plt.xlabel('loanAmnt')
label=train.isDefault
label.value_counts()/len(label)

sns.kdeplot(train.loanAmnt[label[label==1].index], label='1', shade=True)#违约
sns.kdeplot(train.loanAmnt[label[label==0].index], label='0', shade=True)#没有违约
plt.xlabel('loanAmnt')
plt.ylabel('Density');
plt.figure(1 , figsize = (8 , 5))
sns.distplot(train['annualIncome'])
plt.xlabel('annualIncome')
discrete_feature
for f in discrete_feature:
    print(f, '类型数: ', train[f].nunique())

df_ = train[discrete_feature]

sns.set_style("whitegrid")# 使用 whitegrid 主题
fig,axes=plt.subplots(nrows=4,ncols=2,figsize=(8,10))
for i, item in enumerate(df_):
```



```
plt.subplot(4,2,(i+1))
#ax=df[item].value_counts().plot(kind = 'bar')
ax=sns.countplot(item,data = df_,palette="Pastel1")
plt.xlabel(str(item),fontsize=14)
plt.ylabel('Count',fontsize=14)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
#plt.title("Churn by "+ str(item))
i=i+1
plt.tight_layout()
plt.show()
unique_feature
# 分类型特征
category_feature = list(filter(lambda x: x not in numerical_feature,list(train.columns)))
category_feature
train[category_feature]

df_category = train[['grade', 'subGrade']]

sns.set_style("whitegrid") # 使用 whitegrid 主题
color = sns.color_palette()
fig,axes=plt.subplots(nrows=2,ncols=1,figsize=(10,10))
for i, item in enumerate(df_category):
    plt.subplot(2,1,(i+1))
    #ax=df[item].value_counts().plot(kind = 'bar')
    ax=sns.countplot(item,data = df_category)
    plt.xlabel(str(item),fontsize=14)
    plt.ylabel('Count',fontsize=14)
    plt.xticks(fontsize=13)
    plt.yticks(fontsize=13)
    #plt.title("Churn by "+ str(item))
    i=i+1
    plt.tight_layout()
plt.show()

plt.figure(1 , figsize = (10 , 8))
sns.barplot(train["employmentLength"].value_counts(dropna=False),
            train["employmentLength"].value_counts(dropna=False).keys())
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.xlabel('employmentLength',fontsize=14)
plt.show()
for i in train[['issueDate', 'earliesCreditLine']]:
    print(train[i].value_counts())
```



```
print()

label=train.isDefault
label.value_counts()/len(label)

sns.countplot(label)

train_loan_fr = train.loc[train['isDefault'] == 1]
train_loan_nofr = train.loc[train['isDefault'] == 0]

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 8))
# 目标变量为 1 时候 grade 的分布
train_loan_fr.groupby("grade").size().plot.bar(ax=ax1)
# 目标变量为 0 时候 grade 的分布
train_loan_nofr.groupby("grade")["grade"].count().plot.bar(ax=ax2)
# 目标变量为 1 时候 employmentLength 的分布
train_loan_fr.groupby("employmentLength").size().plot.bar(ax=ax3)
# 目标变量为 0 时候 employmentLength 的分布
train_loan_nofr.groupby("employmentLength")["employmentLength"].count().plot.bar(ax=ax4)
plt.xticks(rotation=90);

train_positve = train[train['isDefault'] == 1]
train_negative = train[train['isDefault'] != 1]
f, ax = plt.subplots(len(numerical_feature),2,figsize = (10,80))
for i,col in enumerate(numerical_feature):
    sns.distplot(train_positve[col],ax = ax[i,0],color = "blue")
    ax[i,0].set_title("positive")
    sns.distplot(train_negative[col],ax = ax[i,1],color = 'red')
    ax[i,1].set_title("negative")
plt.subplots_adjust(hspace = 1)
# 去掉标签
X_missing = train.drop(['isDefault'],axis=1)

# 查看缺失情况
missing = X_missing.isna().sum()
missing = pd.DataFrame(data={'特征': missing.index,'缺失值个数':missing.values})
#通过~取反，选取不包含数字 0 的行
missing = missing[~missing['缺失值个数'].isin([0])]
# 缺失比例
missing['缺失比例'] = missing['缺失值个数']/X_missing.shape[0]
missing

# 可视化
(train.isnull().sum()/len(train)).plot.bar(figsize = (20,6),color=['#d6ecf0','#a3d900','#88ada6','#ffb3a7','#cca4e3','#a1afc9'])
```



```
f, ax = plt.subplots(1,1, figsize = (20,20))
cor = train[numerical_feature].corr()
sns.heatmap(cor, annot = True, linewidth = 0.2, linecolor = "white", ax = ax, fnt = ".lg" )

import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import auc, roc_curve
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler
import datetime
np.random.seed(2020)

train_file_name = 'G://classStudy/II 2/机器学习/大作业/贷款违约预测/train.csv'
test_file_name = 'G://classStudy/II 2/机器学习/大作业/贷款违约预测/testA.csv'

df_train = pd.read_csv(train_file_name)
df_test = pd.read_csv(test_file_name)

from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
df_train['grade'] = labelEncoder.fit_transform(df_train['grade'])
df_train['grade'] = labelEncoder.fit_transform(df_train['grade'])
df_train['grade']
df_train['employmentLength']
employmentLength = ['< 1 year','1 year','2 years',
                    '3 years', '4 years', '5 years',
                    '6 years', '8 years', '7 years','9 years','10+ years']

j = 0
for i in employmentLength:
    df_train['employmentLength'] = df_train['employmentLength'].replace(i, j)
    j += 1
_ = pd.crosstab(df_train.subGrade, df_train.isDefault)
_["yp"] = _[1]/(_[0]+_[1])
_.reset_index(inplace=True)
_.sort_values(by="yp", inplace=True)
-
df_train = pd.merge(df_train, _[["subGrade", "yp"]], on="subGrade", how="left")
df_train['subGrade'] = labelEncoder.fit_transform(df_train['subGrade'])
df_train['issueDate'] = pd.to_datetime(df_train['issueDate'],format='%Y-%m-%d')
startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
df_train['issueDateDT'] = df_train['issueDate'].apply(lambda x: x-startdate).dt.days
df_train['earliesCreditLine'] = df_train['earliesCreditLine'].apply(lambda s: int(s[-4:]))
```





```

df_train['earliesCreditLine']
tags = ['loanAmnt', 'term', 'interestRate', 'installment', 'grade',
        'subGrade', 'employmentTitle', 'employmentLength', 'homeOwnership',
        'annualIncome', 'verificationStatus', 'issueDateDT', 'earliesCreditLine',
        'purpose', 'postCode', 'regionCode', 'dti', 'delinquency_2years',
        'ficoRangeLow', 'ficoRangeHigh', 'openAcc', 'pubRec',
        'pubRecBankruptcies', 'revolBal', 'revolUtil', 'totalAcc',
        'initialListStatus', 'applicationType', 'title',
        'n0', 'n1', 'n2', 'n4', 'n5', 'n6', 'n7', 'n8',
        'n9', 'n10', 'n11', 'n12', 'n13', 'n14', 'yp']
df_test['grade'] = labelEncoder.fit_transform(df_test['grade'])
employmentLength = ['< 1 year', '1 year', '2 years',
                    '3 years', '5 years', '4 years',
                    '6 years', '8 years', '7 years', '9 years', '10+ years']

j = 0
for i in employmentLength:
    df_test['employmentLength'] = df_test['employmentLength'].replace(i, j)
    j += 1
df_test = pd.merge(df_test, _[["subGrade", "yp"]], on="subGrade", how="left")
df_test['subGrade'] = labelEncoder.fit_transform(df_test['subGrade'])
df_test['issueDate'] = pd.to_datetime(df_test['issueDate'], format='%Y-%m-%d')
startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
df_test['issueDateDT'] = df_test['issueDate'].apply(lambda x: x-startdate).dt.days
df_test['earliesCreditLine'] = df_test['earliesCreditLine'].apply(lambda s: int(s[-4:]))
Standard_scaler = StandardScaler()
Standard_scaler.fit(df_train[tags].values)
x = Standard_scaler.transform(df_train[tags].values)
x_ = Standard_scaler.transform(df_test[tags].values)
y = df_train['isDefault'].values
lgbr = LGBMRegressor(num_leaves=30
                    ,max_depth=10
                    ,learning_rate=0.01
                    ,n_estimators=13000
                    ,subsample_for_bin=5000
                    ,min_child_samples=200
                    ,colsample_bytree=.2
                    ,reg_alpha=.1
                    ,reg_lambda=.1
                    ,seed=2020
                    )
cat = CatBoostRegressor(depth=9,
                        l2_leaf_reg=1,
                        learning_rate=0.01,
                        eval_metric = 'AUC' ,

```



```
border_count = 128,
bagging_temperature = 0.9 ,
n_estimators=16000,
early_stopping_rounds=500,
subsample = 0.9,
random_seed=1,
verbose = 0)

from sklearn.ensemble import VotingRegressor
rg_model = VotingRegressor([('lgb', lgbr), ('catboost', cat)],n_jobs=12)
rg_model.fit(x,y)
pre = pd.DataFrame(rg_model.predict(x_),columns=['isDefault'])
results = pd.concat([df_test['id'],pre],axis = 1)
results.to_csv('submit.csv', index=False)
```