1 实验概况

本实验任务是车辆贷款违约预测，具体要求如下：

给定某机构实际业务中的相关借款人信息，包含53个与客户相关的字段，其中loan_default字段表明借款人是否会拖欠付款。任务目标是通过训练集训练模型，来预测测试集中loan_default字段的具体值，即借款人是否会拖欠付款，以此为依据，降低贷款风险。

1.1 实验环境

名称	规格
CPU	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz 2.50 GHz
Python	3.6.5 (anaconda)
Numpy	1.19.5
Matplotlib	3.3.4
pandas	1.1.5
seaborn	0.11.2
scikit-learn	0.19.1
xgboost	1.5.1

1.2 数据集说明

赛题数据由训练集和测试集组成，总数据量超过25w，包含52个特征字段。为了保证比赛的公平性，将会从中抽取15万条作为训练集，3万条作为测试集，同时会对部分字段信息进行脱敏。

特征字段	字段描述
customer_id	客户标识符
main_account_loan_no	主账户申请贷款数量
main_account_active_loan_no	主账户申请的有效贷款数量
main_account_overdue_no	主账号逾期数量
main_account_outstanding_loan	主账户未偿还的贷款余额
main_account_sanction_loan	主账户所有贷款被批准的贷款金额
main_account_disbursed_loan	主账户所有贷款已发放的贷款金额
sub_account_loan_no	二级账户申请贷款数量
sub_account_active_loan_no	二级账户申请的有效贷款数量
sub_account_overdue_no	二级账户逾期数量
sub_account_outstanding_loan	二级账户未偿还的贷款金额
sub_account_sanction_loan	二级账户所有贷款被批准的贷款金额
sub_account_disbursed_loan	二级账户所有贷款已发放的贷款金额
disbursed_amount	已发放贷款金额
asset_cost	资产成本
branch_id	发放贷款的分行
supplier_id	发放贷款的车辆经销商
manufacturer_id	汽车制造商
year_of_birth	客户出生日期
disbursed_date	贷款日期
area_id	付款区域
employee_code_id	记录付款的对接员工
mobileno_flag	是否填写手机号
idcard_flag	是否填写身份证
Driving_flag	是否出具驾驶证
passport_flag	是否填写护照
credit_score	信用评分
main_account_monthly_payment	主账户月供金额
sub_account_monthly_payment	二级账户的月供金额
last_six_month_new_loan_no	过去六个月客户的新贷款申请数量

特征字段	字段描述
last_six_month_defaulted_no	过去六个月客户的违约数量
average_age	平均贷款期限
credit_history	信用记录
enquirie_no	客户查询贷款次数
loan_to_asset_ratio	贷款与资产比例
total_account_loan_no	所有账户申请贷款数量
main_account_inactive_loan_no	主账户申请的无效贷款数量
sub_account_inactive_loan_no	二级账户申请的无效贷款数量
total_inactive_loan_no	所有账户申请的无效贷款数量
total_overdue_no	所有账户的逾期次数
total_outstanding_loan	所有账户的未结余额的总额
total_sanction_loan	来自所有账户的所有贷款被批准的贷款金额
total_disbursed_loan	为所有账户的所有贷款支付的贷款金额
total_monthly_payment	所有账户的月供金额
outstanding_disburse_ratio	已发放贷款总额/未偿还贷款总额（两者比例）
main_account_tenure	主账户还款期数
sub_account_tenure	二级账户还款期数
disburse_to_sactioned_ratio	已发放贷款/批准贷款（两者比例）
active_to_inactive_act_ratio	有效贷款次数/无效贷款次数（两者比例）
Credit_level	信用评分
employment_type	工作类型
age	年龄
loan_default	1表示客户逾期，0表示客户未逾期

1.3 评价指标

本次竞赛的评价标准采用F1 score 指标，具体地为macro-f1指标；F1指标的计算公式为：

$$F1 = 2 * (precision * recall) / (precision + recall)$$

而macro-f1指标是在多分类或者多标签问题上，对多个类别F1指标的平均的一种方式，其中macro-f1指的是直接求F1值的平均，意味着不考虑多个类别之间的类别不平衡问题。

$$macro - f1 = \sum_{i=1}^{N} F1_i$$

2 数据集分析

2.1 数据导入

首先加载必要的相关python库，包括numpy，matplotlib，pandas，seaborn等

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#显示所有列
pd.set_option('display.max_columns', None)
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

然后通过`pandas.read_csv`函数加载数据集，并查看前五行信息

```
df = pd.read_csv("train_2.csv")
df.head() #查看前5行信息
```

In [2]:
```
df = pd.read_csv("train_2.csv")
df.head() #查看前5行信息
```

Out[2]:

	customer_id	main_account_loan_no	main_account_active_loan_no	main_account_overdue_no	main_account_outstanding_loan	main_account_sanction_loan	r
0	105691	4	3	0	384989	666207	
1	24938	7	2	0	268670	387994	
2	104389	5	4	1	3519013	3613854	
3	54688	43	13	6	1867106	2484678	
4	63894	0	0	0	0	0	

2.2 类别分布统计

数据集中的loan_default代表类别信息，0表示没有违约记录，1表示存在违约记录。我们通过`value_counts`函数简单统计下两个类别的数量，分布情况。

In [3]:
```
df["loan_default"].value_counts() #查看数据集类别分布
```

Out[3]:
```
0    98808
1    21192
Name: loan_default, dtype: int64
```

发现该数据集存在一个比较严重的类别不平衡情况，未违约与存在违约的数量比例大概是5:1,这也是本次实验比较大的挑战之一。

2.3 异常值分析

首先是查找是否存在缺失值,使用`isnull()`函数查找缺失值，结果如下图所示。发现并不存在缺失值。

In [7]:
```
#查看缺失值
df.isnull().sum()
```

Out[7]:

```
customer_id                         0
main_account_loan_no                0
main_account_active_loan_no         0
main_account_overdue_no             0
main_account_outstanding_loan       0
main_account_sanction_loan          0
main_account_disbursed_loan         0
sub_account_loan_no                 0
sub_account_active_loan_no          0
sub_account_overdue_no              0
sub_account_outstanding_loan        0
sub_account_sanction_loan           0
sub_account_disbursed_loan          0
disbursed_amount                    0
asset_cost                          0
branch_id                           0
supplier_id                         0
manufacturer_id                     0
area_id                             0
employee_code_id                    0
mobileno_flag                       0
idcard_flag                         0
Driving_flag                        0
passport_flag                       0
credit_score                        0
main_account_monthly_payment        0
sub_account_monthly_payment         0
last_six_month_new_loan_no          0
last_six_month_defaulted_no         0
average_age                         0
credit_history                      0
enquirie_no                         0
loan_to_asset_ratio                 0
total_account_loan_no               0
sub_account_inactive_loan_no        0
total_inactive_loan_no              0
main_account_inactive_loan_no       0
total_overdue_no                    0
total_outstanding_loan              0
total_sanction_loan                 0
total_disbursed_loan                0
total_monthly_payment               0
outstanding_disburse_ratio          0
main_account_tenure                 0
sub_account_tenure                  0
disburse_to_sactioned_ratio         0
active_to_inactive_act_ratio        0
year_of_birth                       0
disbursed_date                      0
Credit_level                        0
employment_type                     0
age                                 0
loan_default                        0
dtype: int64
```

In [4]:
```
df.dtypes #查看各个特征的数据类型
```

使用`df.describe()`来展示各个特征值的数值信息，该函数将返回均值、方差、最小值、中位数、最大值等信息。

In [6]:
```
df.describe()
```

Out[6]:

›d_loan	total_monthly_payment	outstanding_disburse_ratio	main_account_tenure	sub_account_tenure	disburse_to_sactioned_ratio	active_to_inactive_act_ratio	y
00e+05	1.200000e+05	120000.00	1.200000e+05	1.200000e+05	1.200000e+05	120000.000000	12
03e+05	1.287669e+04	inf	5.387051e+04	2.774115e+03	7.533143e+02	1.439262	
25e+06	1.442883e+05	NaN	2.914938e+06	1.112462e+05	1.465863e+05	0.790481	
00e+00	0.000000e+00	-70847.67	0.000000e+00	0.000000e+00	0.000000e+00	1.000000	
00e+00	0.000000e+00	1.00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000	
00e+00	0.000000e+00	1.00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000	
00e+04	2.045000e+03	1.26	2.500000e+01	0.000000e+00	1.000000e+00	1.670000	
00e+09	2.076655e+07	inf	1.000000e+09	1.980000e+07	5.000000e+07	18.000000	

在结果统计中，发现在outstanding_disburse_ratio该特征上存在一个特殊值inf,考虑到模型无法处理inf。所以需要将inf替换成一个有限大的值。首先我查找所有inf所在的位置，

```
np.where(df == np.inf)
```

In [37]:
```
np.where(df == np.inf)
```

Out[37]:
```
(array([ 16195,  17293,  21512,  23965,  26116,  37087,  51544,  59870,
         64117,  87064, 116054, 117807], dtype=int64),
 array([42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42], dtype=int64))
```

发现仅有outstanding_disburse_ratio这一列出现了inf,然后我决定查找该列除inf外之外最大的数（500001.0）,所以我将inf替换成了稍大于500001的600001。

In [12]:
```
maxvalue = max(set(df["outstanding_disburse_ratio"])- {np.inf})
print(maxvalue)
df = df.replace(np.inf,6000001.0)
```

```
5000001.0
```

In [52]:
```
max(df["outstanding_disburse_ratio"]) #检查是否替换成功
```

Out[52]:
```
6000001.0
```

2.4 数据类型分析

其次是数据类型分析，使用`df.dtypes`打印所有特征的数值类型，结果如下：

```
In [4]: df.dtypes #查看各个特征的数据类型

Out[4]: customer_id                        int64
        main_account_loan_no               int64
        main_account_active_loan_no        int64
        main_account_overdue_no            int64
        main_account_outstanding_loan      int64
        main_account_sanction_loan         int64
        main_account_disbursed_loan        int64
        sub_account_loan_no                int64
        sub_account_active_loan_no         int64
        sub_account_overdue_no             int64
        sub_account_outstanding_loan       int64
        sub_account_sanction_loan          int64
        sub_account_disbursed_loan         int64
        disbursed_amount                   int64
        asset_cost                         int64
        branch_id                          int64
        supplier_id                        int64
        manufacturer_id                    int64
        area_id                            int64
        employee_code_id                   int64
        mobileno_flag                      int64
        idcard_flag                        int64
        Driving_flag                       int64
        passport_flag                      int64
        credit_score                       int64
        main_account_monthly_payment       int64
        sub_account_monthly_payment        int64
        last_six_month_new_loan_no         int64
        last_six_month_defaulted_no        int64
        average_age                        int64
        credit_history                     int64
        enquirie_no                        int64
        loan_to_asset_ratio              float64
        total_account_loan_no              int64
        sub_account_inactive_loan_no       int64
        total_inactive_loan_no             int64
        main_account_inactive_loan_no      int64
        total_overdue_no                   int64
        total_outstanding_loan             int64
        total_sanction_loan                int64
        total_disbursed_loan               int64
        total_monthly_payment              int64
        outstanding_disburse_ratio       float64
        main_account_tenure                int64
        sub_account_tenure                 int64
        disburse_to_sactioned_ratio      float64
        active_to_inactive_act_ratio     float64
        year_of_birth                      int64
        disbursed_date                     int64
        Credit_level                       int64
        employment_type                    int64
        age                                int64
        loan_default                       int64
        dtype: object
```

结果显示，数据集仅存在数值型特征，包括浮点型float类型，以及整数型int类型。由于不存在非数值型数据，所以特征编码这步可以省去。

In [12]:
```
#各个特征的特征值的个数
df.nunique()
```

Out[12]:

customer_id	120000
main_account_loan_no	98
main_account_active_loan_no	35
main_account_overdue_no	19
main_account_outstanding_loan	39796
main_account_sanction_loan	25212
main_account_disbursed_loan	27031
sub_account_loan_no	36
sub_account_active_loan_no	20
sub_account_overdue_no	8
sub_account_outstanding_loan	1688
sub_account_sanction_loan	1241
sub_account_disbursed_loan	1405
disbursed_amount	16888
asset_cost	35284
branch_id	82
supplier_id	2835
manufacturer_id	10
area_id	22
employee_code_id	3229
mobileno_flag	1
idcard_flag	1
Driving_flag	2
passport_flag	2
credit_score	568
main_account_monthly_payment	18608
sub_account_monthly_payment	1063
last_six_month_new_loan_no	24
last_six_month_defaulted_no	14
average_age	100
credit_history	100
enquirie_no	22
loan_to_asset_ratio	111726
total_account_loan_no	97
sub_account_inactive_loan_no	84
total_inactive_loan_no	26
main_account_inactive_loan_no	86
total_overdue_no	19
total_outstanding_loan	40423
total_sanction_loan	25723
total_disbursed_loan	27588
total_monthly_payment	18894
outstanding_disburse_ratio	3830
main_account_tenure	10718
sub_account_tenure	1016
disburse_to_sactioned_ratio	330
active_to_inactive_act_ratio	206
year_of_birth	48
disbursed_date	1
Credit_level	14
employment_type	3
age	48
loan_default	2

dtype: int64

然后需要将所有特征进一步区分成一下三种类型，特征值唯一型，连续型数据，离散型数据。特征值唯一型数据，顾名思义指的是该特征仅有一个特征值，正因该特征仅有一个特征值，所以没有额外信息，故后续试验中可以将这些特征删除。结果如下:

```
In [18]: #划分1：该特征仅有一个取值,这些特征在后续的实验中会考虑去除
         unique_feas = [col for col in df.columns if df[col].nunique()==1]
         unique_feas
Out[18]: ['mobileno_flag', 'idcard_flag', 'disbursed_date']
```

第二个划分是连续型数值型数据，该特征有两个条件，一是该特征取值数量较多，另外如果该特征取值数量为n,那么该特征并不为取值为(0,1,.....,n-1)，即取值并不连续有断层，结果如下：

```
In [8]: #划分2：数值型连续型特征，该特征有两个条件，一是特征取值数量较多，另外如果该特征取值数量为n,那么该特征并不为取值为(0,1,……,n-1)，即取值并不
        numerical_serial_feas = [col for col in df.columns if df[col].nunique()>40 or list(set(df[col])) != list(range(df[col].nunique()))]
        numerical_serial_feas = [fea for fea in numerical_serial_feas if fea not in unique_feas]
        print(numerical_serial_feas)
```

```
['customer_id', 'main_account_loan_no', 'main_account_active_loan_no', 'main_account_overdue_no', 'main_account_outstanding_loan', 'main
_account_sanction_loan', 'main_account_disbursed_loan', 'sub_account_loan_no', 'sub_account_active_loan_no', 'sub_account_outstanding_lo
an', 'sub_account_sanction_loan', 'sub_account_disbursed_loan', 'disbursed_amount', 'asset_cost', 'branch_id', 'supplier_id', 'employee_
code_id', 'credit_score', 'main_account_monthly_payment', 'sub_account_monthly_payment', 'last_six_month_new_loan_no', 'last_six_month_d
efaulted_no', 'average_age', 'credit_history', 'enquirie_no', 'loan_to_asset_ratio', 'total_account_loan_no', 'sub_account_inactive_loan
_no', 'total_inactive_loan_no', 'main_account_inactive_loan_no', 'total_overdue_no', 'total_outstanding_loan', 'total_sanction_loan', 't
otal_disbursed_loan', 'total_monthly_payment', 'outstanding_disburse_ratio', 'main_account_tenure', 'sub_account_tenure', 'disburse_to_s
actioned_ratio', 'active_to_inactive_act_ratio', 'year_of_birth', 'Credit_level', 'age']
```

剩下的一个划分是离散型数值型数据，结果如下：

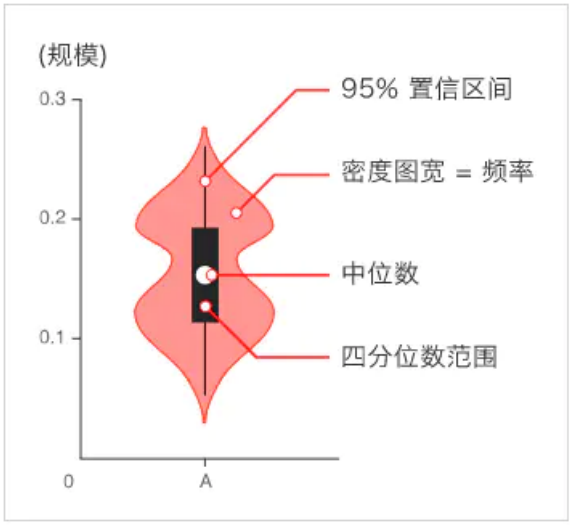
```
In [9]: # 划分三,为数值型离散型变量，除划分一和划分二和标签外其他所有特征
        numerical_noserial_feas = [col for col in df.columns if col not in unique_feas+numerical_serial_feas+["loan_default"]]
        numerical_noserial_feas
Out[9]: ['sub_account_overdue_no',
         'manufacturer_id',
         'area_id',
         'Driving_flag',
         'passport_flag',
         'employment_type']
```

离散型数据后续可能会做的一个处理是，将该特征one-hot化。

3 数据可视化分析

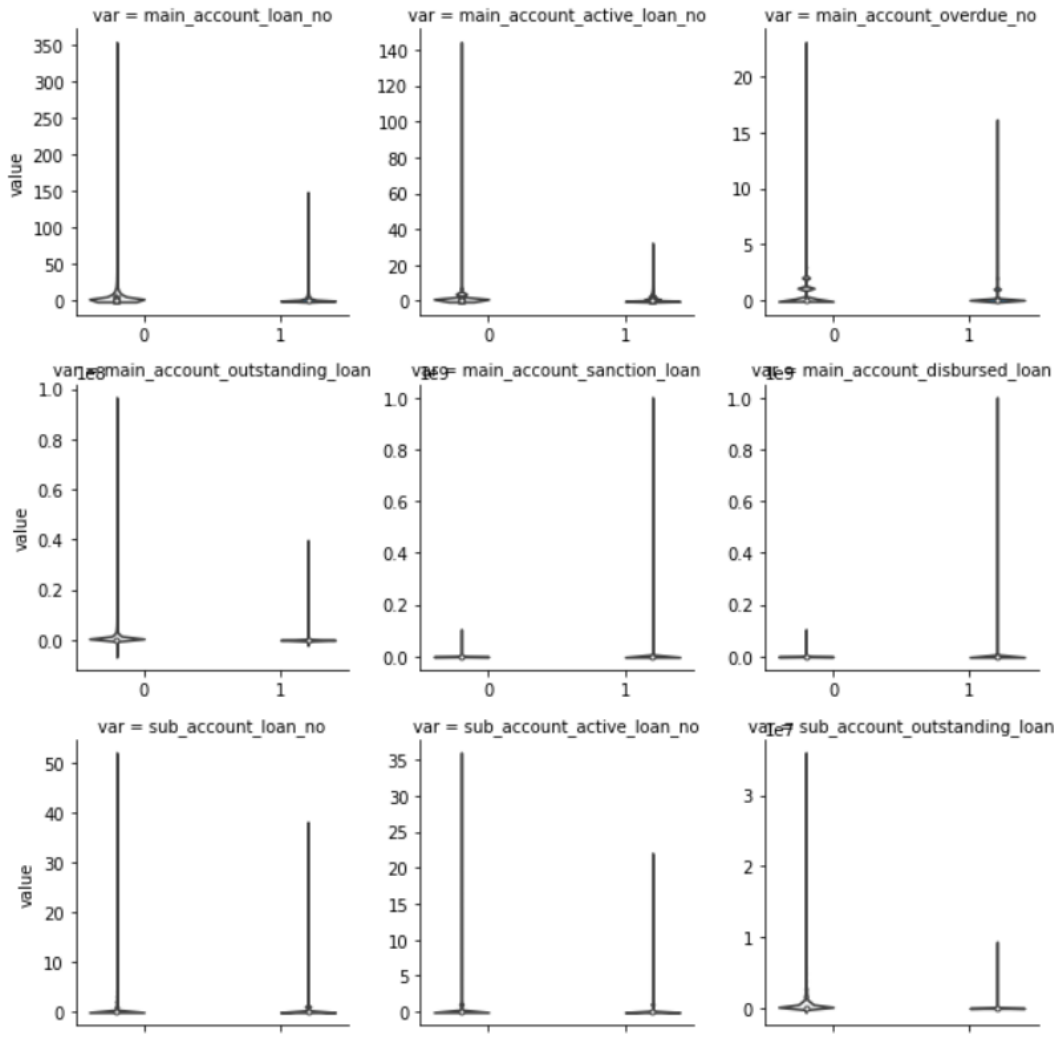
3.1 连续型数据小提琴图

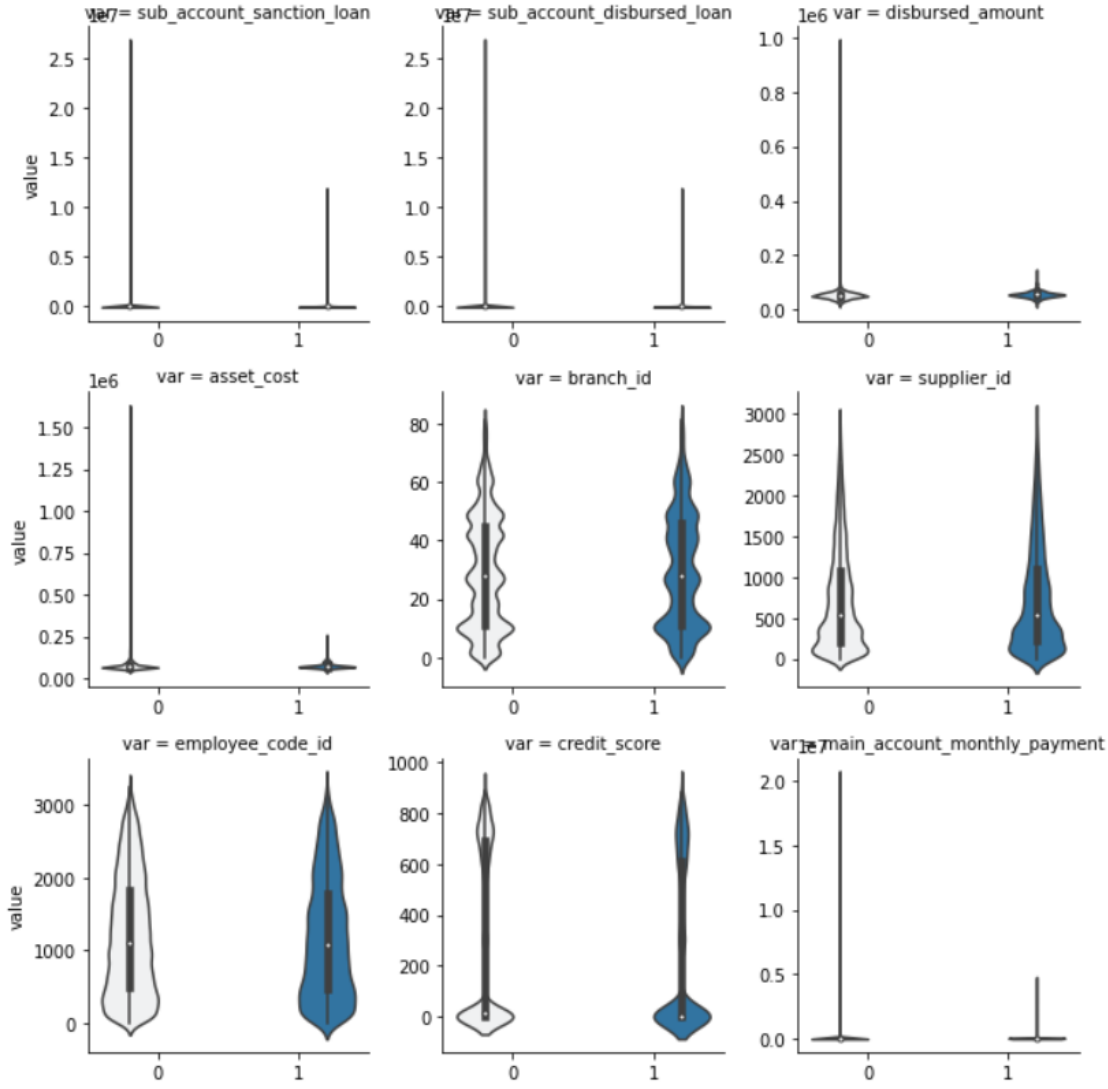
采用小提琴图来查看连续型数据的分布情况，小提琴图结合了箱线图和密度图的特征，用来显示数据的分布形状。如下图所示，中间的黑色粗条表示四分位数范围，从其延伸的幼细黑线代表 95% 置信区间，而白点则为中位数。

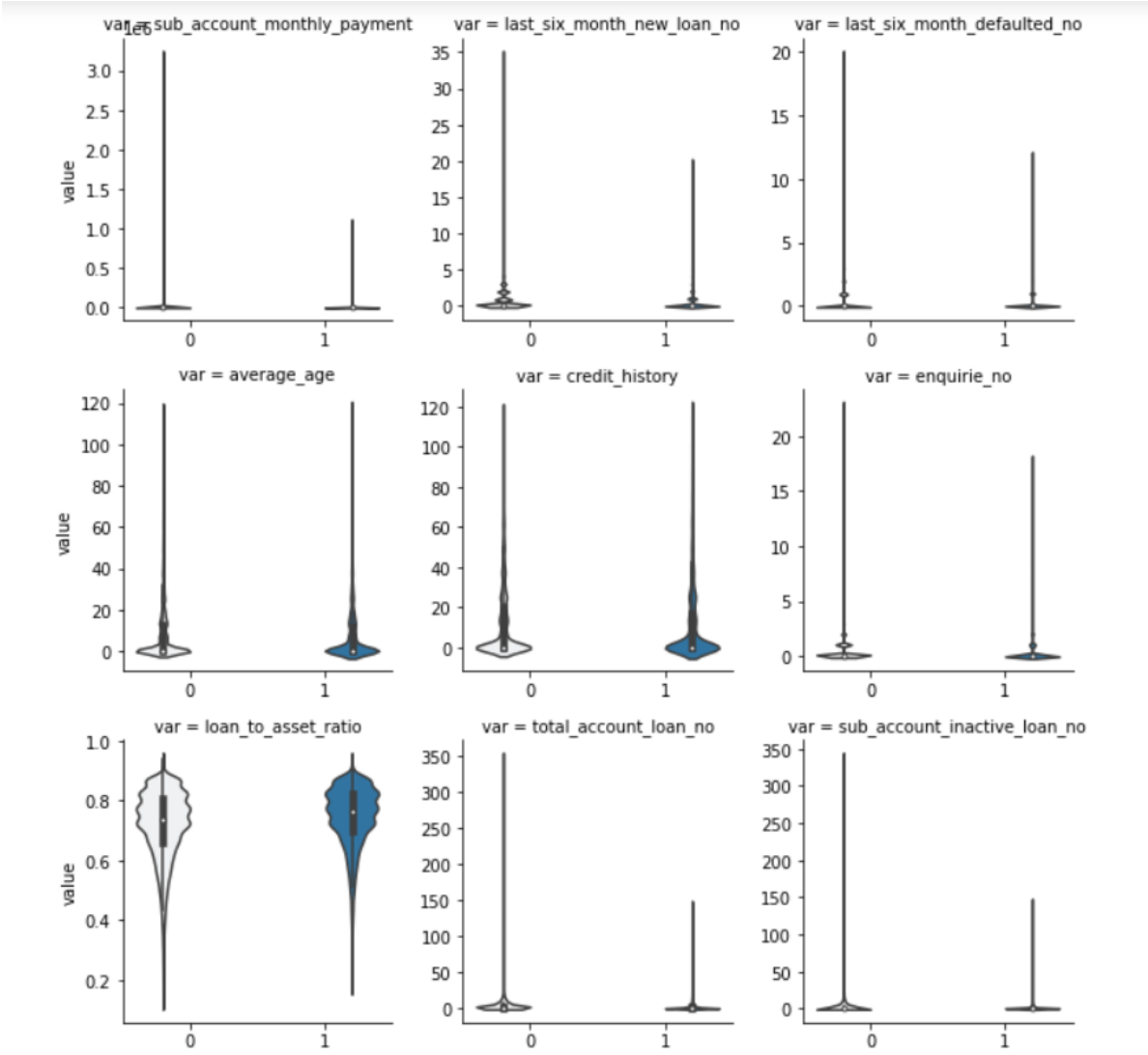


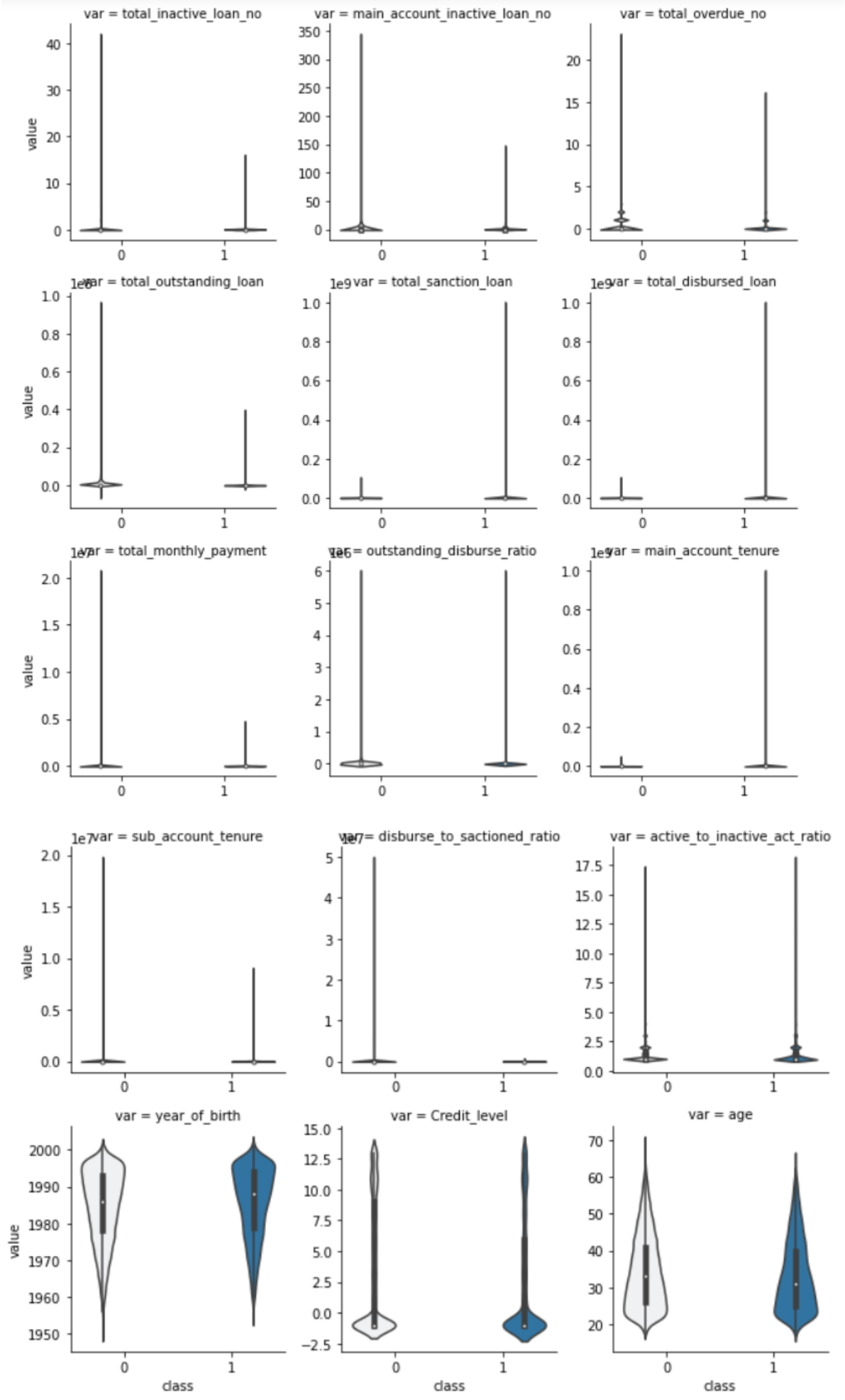
在本次实验中，采用小提琴图的目的是直观地显示在各个特征中，不同label的特征值是否存在着分布的差异，如果存在较大差异，那么该特征则是我们需要重点关注的特征，结果如下图所示：

```
#将宽数据类型转换为长数据类型，目的是方便后面画图
arr = [list(df["loan_default"]) * len(numerical_serial_feas)]
arr = [y for x in arr for y in x]
f = pd.melt(df,value_vars=numerical_serial_feas,var_name="var")
f["class"] = arr
#画多视图violin图
g = sns.FacetGrid(data=f, col="var",col_wrap=3, sharex=False, sharey=False)
g = g.map_dataframe(sns.violinplot,x="class",y="value",hue="class")
```




var = sub_account_sanction_loan
var = sub_account_disbursed_loan
var = disbursed_amount
var = asset_cost
var = branch_id
var = supplier_id
var = employee_code_id
var = credit_score
var = main_account_monthly_payment
value
1e7
1e6
0
1


var = sub_account_monthly_payment
1e6
var = last_six_month_new_loan_no
var = last_six_month_defaulted_no
var = average_age
var = credit_history
var = enquirie_no
var = loan_to_asset_ratio
var = total_account_loan_no
var = sub_account_inactive_loan_no
value


var = total_inactive_loan_no
var = main_account_inactive_loan_no
var = total_overdue_no
var = total_outstanding_loan
var = total_sanction_loan
var = total_disbursed_loan
var = total_monthly_payment
var = outstanding_disburse_ratio
var = main_account_tenure
var = sub_account_tenure
var = disburse_to_sactioned_ratio
var = active_to_inactive_act_ratio
var = year_of_birth
var = Credit_level
var = age
value
class

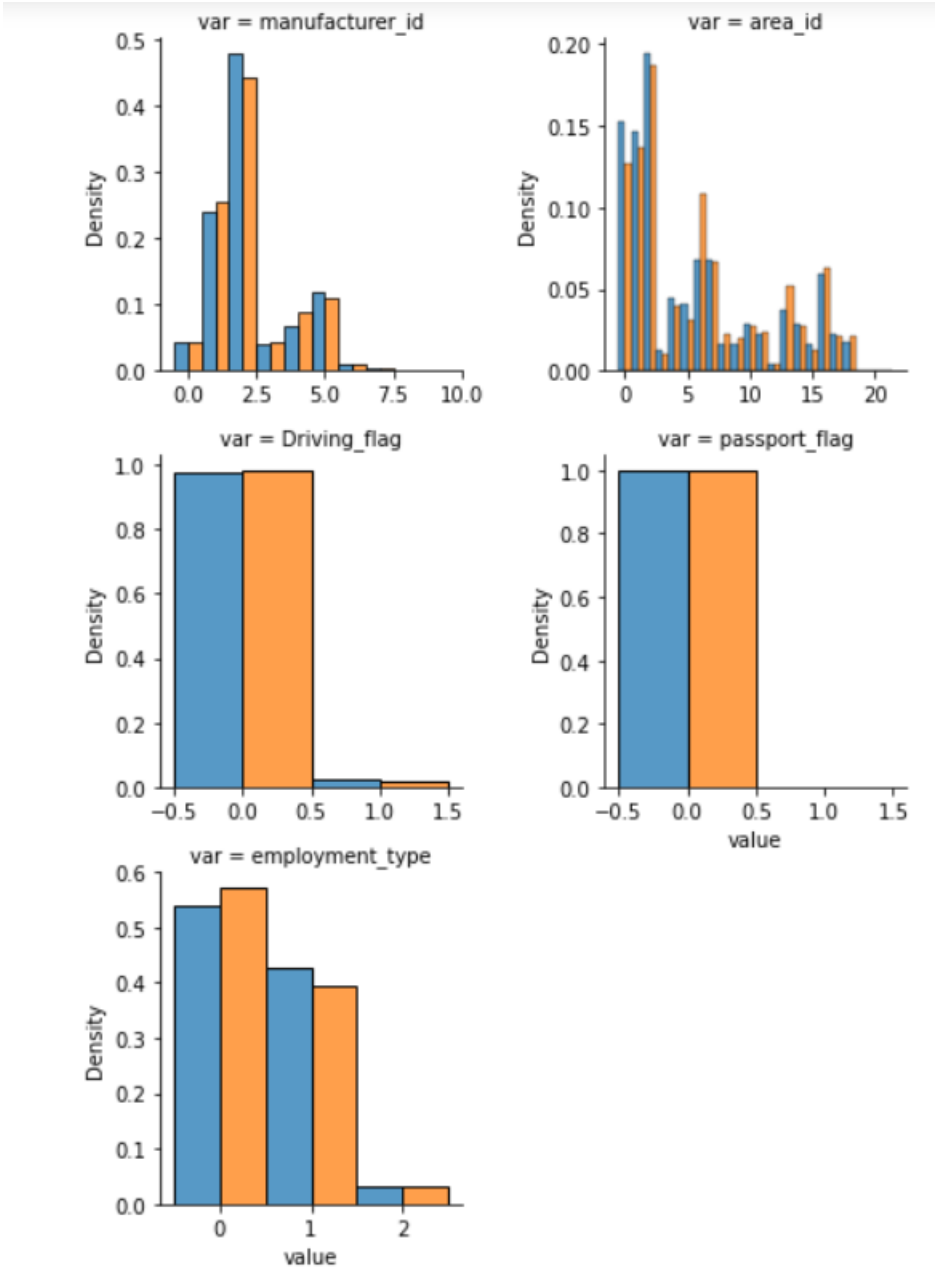
从图上可以看出很多特征的小提琴图呈"钉字型"，这表明该特征的特征值都集中分布在某个很小的区间内；另外很多特征的class0的class1的小提琴图非常相似，这表明该特征并不存在显著的（能区分类别的）差异性。

3.2 离散型数据直方图

对于离散型数据，画出直方图，代码如下:

```
arr = [list(df["loan_default"]) * len(numerical_noserial_feas)]
arr = [y for x in arr for y in x]
f = pd.melt(df,value_vars=numerical_noserial_feas,var_name="var")
f["class"] = arr
g = sns.FacetGrid(data=f, col="var",col_wrap=2, sharex=False, sharey=False)
g =
g.map_dataframe(sns.histplot,x="value",hue="class",multiple="dodge",discrete=Tru
e,stat="density", common_norm=False)
```

结果如下:



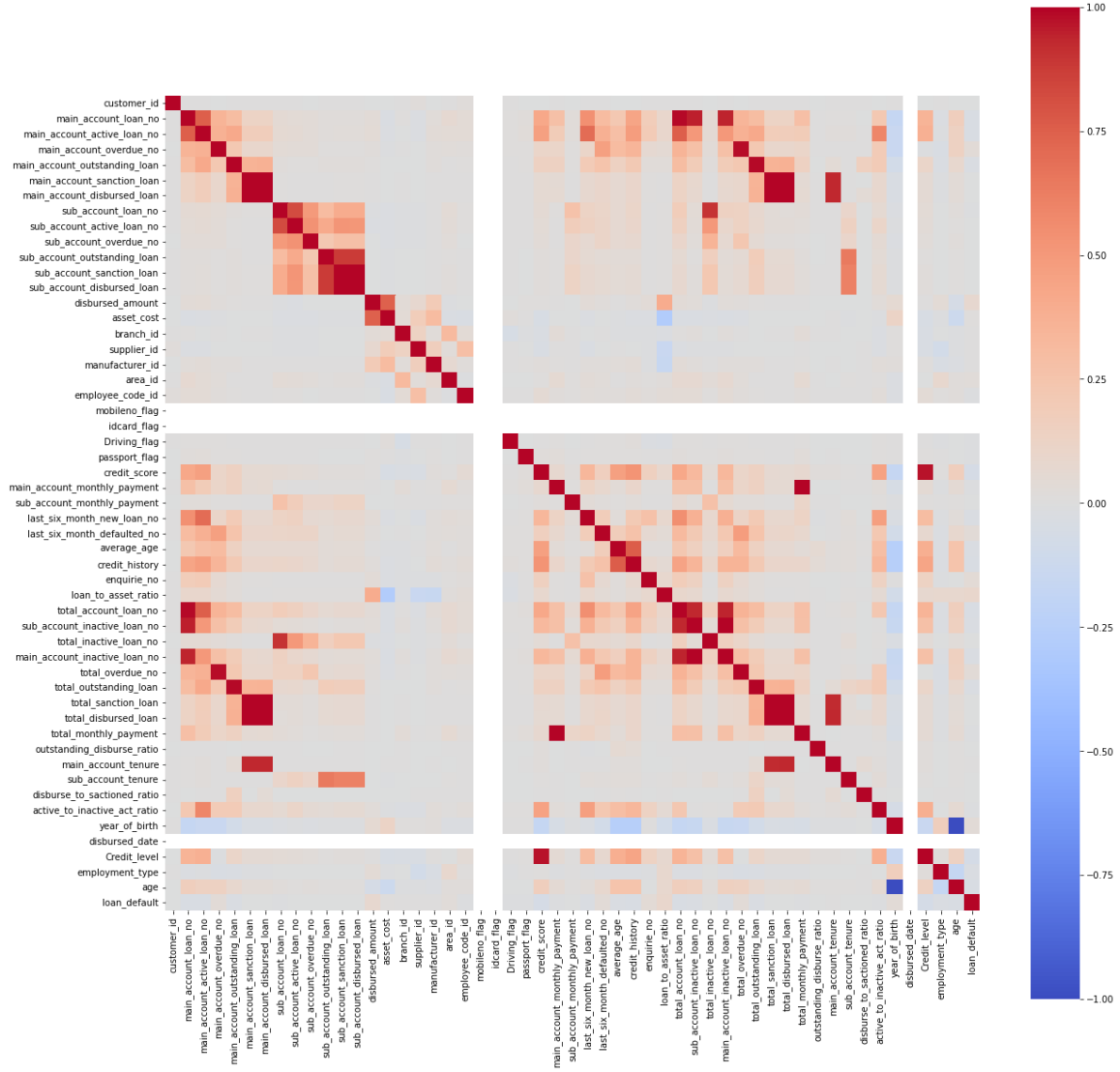
根据图中显示的结果，我们发现所有的离散型特征都不具备很好的区分度。反应在图中就是蓝色的柱状图和橙色的柱状图十分相像。

3.3 特征相关度的热力图

使用`df.corr()`计算各个特征之间的pearson；接着使用`seaborn.heatmap`来可视化特征之间的相关度。

```
#查看特征之间的相关性
fig = plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), square=True, cmap='coolwarm', annot_kws={'size': 14})
```

结果，如下图所示:



根据图中结果发现，有很多对特征的相关度非常高；于是我统计了相关度超过0.99，以及小于-0.99的特征对；结果如下：

```
#统计一些相关度非常高的特征对，后续可以仅保持一个特征
index = np.where(np.logical_or(df.corr() > 0.99,df.corr() <-0.99))
feature_pair = [(df.columns[i],df.columns[j]) for (i,j) in zip(*index) if i<j]
feature_pair
```

In [132]:
```
#统计一些相关度非常高的特征对，后续可以仅保持一个特征
index = np.where(np.logical_or(df.corr() > 0.99,df.corr() <-0.99))
feature_pair = [(df.columns[i],df.columns[j]) for (i,j) in zip(*index) if i<j]
feature_pair
```

Out[132]:
```
[('main_account_loan_no', 'total_account_loan_no'),
 ('main_account_sanction_loan', 'main_account_disbursed_loan'),
 ('main_account_sanction_loan', 'total_sanction_loan'),
 ('main_account_sanction_loan', 'total_disbursed_loan'),
 ('main_account_disbursed_loan', 'total_sanction_loan'),
 ('main_account_disbursed_loan', 'total_disbursed_loan'),
 ('sub_account_sanction_loan', 'sub_account_disbursed_loan'),
 ('main_account_monthly_payment', 'total_monthly_payment'),
 ('sub_account_inactive_loan_no', 'main_account_inactive_loan_no'),
 ('total_sanction_loan', 'total_disbursed_loan'),
 ('year_of_birth', 'age')]
```

4 模型训练和推理

4.1 逻辑回归模型

首先进行训练集和测试集划分，数据集中一共有120000条数据；划分100000条数据为训练集，20000条数据为测试集

```
from sklearn.model_selection import train_test_split
X,y = df.iloc[:,1:-1],df.iloc[:,-1]
X_train,X_test,y_train,y_test =
train_test_split(df.iloc[:,1:-1],df.iloc[:,-1],test_size=20000,random_state=222,
stratify=y)
```

这边我设置了`stratify=y`，这条语句的作用在于保持训练集和测试集的类别分布与原来一致，测试集的类别分布为：

In [15]:
```
np.bincount(y_test)
```

Out[15]:
```
array([16468,  3532], dtype=int64)
```

首先测试最简单的逻辑回归模型,结果如下图

```
# test logistic regression
from sklearn.linear_model import LogisticRegression,LogisticRegressionCV
lr = LogisticRegressionCV(class_weight="balanced",scoring="f1_macro")
lr.fit(X_train,y_train)
y_lr_pred = lr.predict(X_test)
print(f"macro-f1 score: {metrics.f1_score(y_lr_pred,
y_test,average='macro'):.3f}")
print("confusion_matrix:")
print(confusion_matrix(y_lr_pred,y_test))
print("classification_report:")
print(classification_report(y_lr_pred,y_test))
```

```
macro-f1 score: 0.447
confusion_matrix:
[[7003 1041]
 [9465 2491]]
classification_report:
             precision    recall  f1-score   support

          0       0.43      0.87      0.57      8044
          1       0.71      0.21      0.32     11956

avg / total       0.59      0.47      0.42     20000
```

macro-f1的指标仅为0.447,需要注意的是因为训练数据存在类别不平衡的问题；所以我在设置分类时，将class_weight参数设置为"balanced",这样会一定程度上缓解类别不平衡的问题。

4.2 支持向量机模型

支持向量机也是一个非常经典的机器学习算法，它的主要思想是最大化分类间隔。

```
# test SVC
from sklearn.svm import SVC
clf = SVC(class_weight="balanced")
clf.fit(X_train,y_train)
y_svc_pred = clf.predict(X_test)
print(f"macro-f1 score: {metrics.f1_score(y_svc_pred,
y_test,average='macro'):.3f}")
print("confusion_matrix:")
print(confusion_matrix(y_svc_pred,y_test))
print("classification_report:")
print(classification_report(y_svc_pred,y_test))
```

同样是将class_weight设置成balanced，结果为

```
macro-f1 score: 0.457
confusion_matrix:
[[16429  3511]
 [   39    21]]
classification_report:
             precision    recall  f1-score   support

          0       1.00      0.82      0.90     19940
          1       0.01      0.35      0.01        60

avg / total       0.99      0.82      0.90     20000
```

尽管设置了class_weight，但是SVM还是会非常倾向于给出"0"的预测结果，可以看出在类别为0的样本上，模型的查准率和召回率都很高。但是在类别为1的样本上，模型无论是查准率还是召回率都非常低，因为在2w个测试集中，模型只认为60个样本的类别为1，而实际上真实数量有4千左右。

4.3 随机森林模型

随机森林是集成学习的一种，它在决策树的基础上，通过随机采样部分数据和部分特征来达到模型集成的目的，有利于减少过拟合的风险。

```
rf0 = RandomForestClassifier(oob_score=True,
random_state=222,class_weight="balanced")
rf0.fit(X_train,y_train)
print(f"oob_score: {rf0.oob_score_:.3f}")
y_rf_pred = rf0.predict(X_test)
print(f"macro-f1 score: {metrics.f1_score(y_rf_pred,
y_test,average='macro'):.3f}")
print("confusion_matrix:")
print(confusion_matrix(y_rf_pred,y_test))
print("classification_report:")
print(classification_report(y_rf_pred,y_test))
```

随机森林的结果如下，和SVM模型一样，非常倾向于给出0的预测结果。macro-f1指标稍好于svm模型。

```
macro-f1 score: 0.479
confusion_matrix:
[[16230  3416]
 [  238   116]]
classification_report:
             precision    recall  f1-score   support

          0       0.99      0.83      0.90     19646
          1       0.03      0.33      0.06       354

avg / total       0.97      0.82      0.88     20000
```

4.4 xgboost模型

Xgboost是梯度提升树(GBDT)的一种实现，它的主要思路是boosting；组合多个弱分类器，通过boosting的方式组成一个强分类器。

```
import xgboost as xgb
model = xgb.XGBClassifier()
model.fit(X_train, y_train)
y_xgb_pred = model.predict(X_test)
print(f"macro-f1 score: {metrics.f1_score(y_xgb_pred,
y_test,average='macro'):.3f}")
print("confusion_matrix:")
print(confusion_matrix(y_xgb_pred,y_test))
print("classification_report:")
print(classification_report(y_xgb_pred,y_test))
```

首先使用不带任何参数的xgboost拟合训练数据，结果如下：

```
macro-f1 score: 0.466
confusion_matrix:
[[16391  3479]
 [   77    53]]
classification_report:
             precision    recall  f1-score   support

          0       1.00      0.82      0.90     19870
          1       0.02      0.41      0.03       130

avg / total       0.99      0.82      0.90     20000
```

这个结果与随机森林和SVM相差无几，考虑到类别不平衡的因素。与上述模型类似，xgboostClassifier中有一个参数`scale_pos_weight`可以调整类别权重，在本例中，尝试设置成类别0的数量与类别1数量之比。

```
size0,size1 = np.bincount(y_train)
model = xgb.XGBClassifier(scale_pos_weight=size0/size1)
```

再次训练xgboost模型，得到结果如下

```
macro-f1 score: 0.541
confusion_matrix:
[[10419  1511]
 [ 6049  2021]]
classification_report:
             precision    recall  f1-score   support

          0       0.63      0.87      0.73     11930
          1       0.57      0.25      0.35      8070

avg / total       0.61      0.62      0.58     20000
```

macro-f1指标达到了0.541,在上述所有模型中最高。

4.5 交叉验证和参数搜索

由于测试集仅能片面反应模型的好坏，于是我采用5折交叉验证的方式提升指标结果的可靠性。同时对上述表现最佳的xgboost模型进行参数调优，采用`sklearn`的`GridSearchCV`函数进行参数搜索，找出在交叉验证中表现最好的模型超参。首先是交叉验证：

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5,random_state=222)
```

StratifiedKFold相比于普通KFold的优势在于，划分测试集和训练集时保持了原来的类别比例。然后时参数搜索，首先用一个字典记录所有待搜索的参数：

```
scale_pos_weight = [1+(4.6-1)/3*i for i in range(5)] #[1.0, 2.2, 3.4, 4.6, 5.8]
max_depth = [4,5,6,7,8]
n_estimators = [50,100,150]
params =
dict(scale_pos_weight=scale_pos_weight,max_depth=max_depth,n_estimators=n_estima
tors)
```

- scale_pos_weight指类别权重，这里我给出了从1.0到5.8五种权重。
- max_depth指的是树的最大高度
- n_estimators指的是元分类器个数

然后进行参数搜索

```
base_xgb = xgb.XGBClassifier()
grid_clf = GridSearchCV(estimator = base_xgb, param_grid = params, cv =
skf,scoring="f1_macro",n_jobs=-1, verbose=4)
grid_clf.fit(X,y)
```

参数搜索的结果如下：

```
In [64]: grid_clf.best_score_
Out[64]: 0.5769158375556602

In [65]: grid_clf.best_params_
Out[65]: {'max_depth': 4, 'n_estimators': 100, 'scale_pos_weight': 3.4}

In [66]: grid_clf.best_index_
Out[66]: 7
```

经过简单的参数搜索后，模型的macro-f1指标来到了0.576超过了上述所有的模型。

4.6 测试集推理

首先依然是导入测试集，代码如下：

```
df_test = pd.read_csv("test_2.csv")
df_test = df_test.drop(unique_feas,axis=1)
X_test = df_test.iloc[:,1:-1]
y_test = df_test.iloc[:,-1]
print(X_test.shape)
print(y_test.shape)
```

使用最优的xgboost模型进行模型推理

```
y_pred = grid_clf.predict(X_test)
print(f"macro-f1 score: {metrics.f1_score(y_pred, y_test,average='macro'):.3f}")
print("confusion_matrix:")
print(confusion_matrix(y_pred,y_test))
print("classification_report:")
print(classification_report(y_pred,y_test))
```

最终的macro-f1指标，混淆矩阵，分类结果报告如下

```
macro-f1 score: 0.571
confusion_matrix:
[[19291  3271]
 [ 5356  2082]]
classification_report:
             precision    recall  f1-score   support

          0       0.78      0.86      0.82     22562
          1       0.39      0.28      0.33      7438

avg / total       0.69      0.71      0.70     30000
```