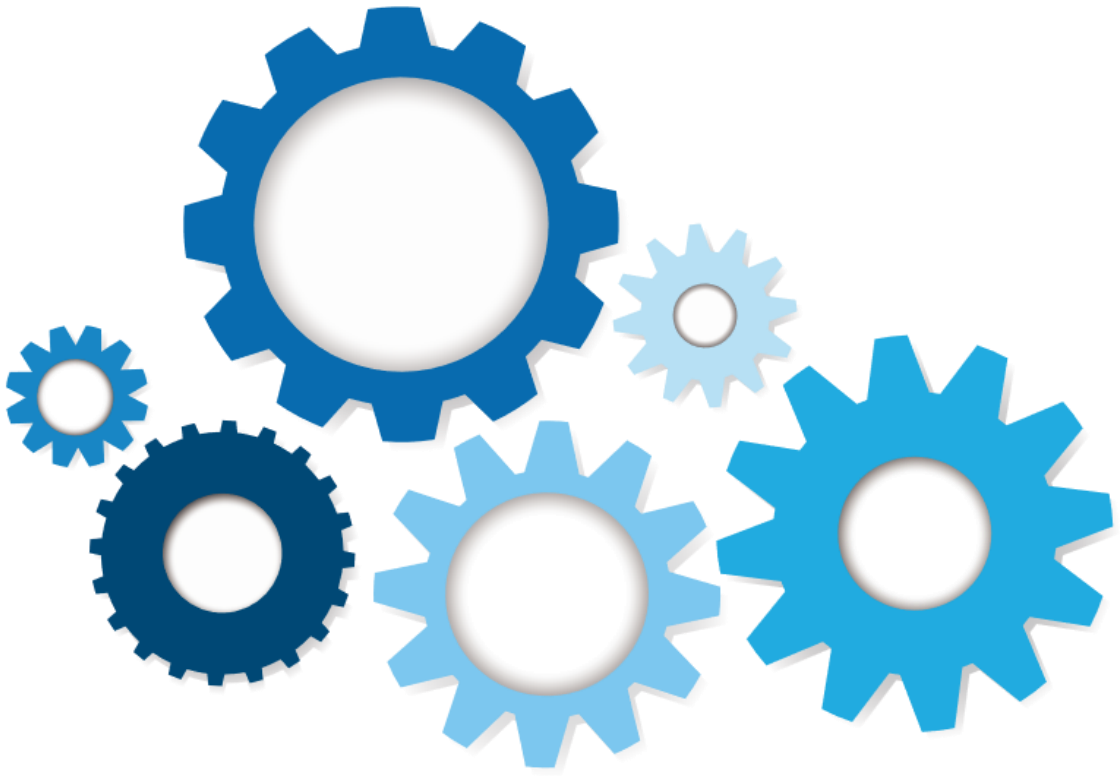
Datawhale

零基础入门金融风控

——贷款违约预测挑战赛



Datawhale 出品

1 Task1 赛题理解

Tip:本次新人赛是Datawhale与天池联合发起的0基础入门系列赛事第四场 —— 零基础入门金融风控之贷款违约预测挑战赛。

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。通过这道赛题来引导大家了解金融风控中的一些业务背景，解决实际问题，帮助竞赛新人进行自我练习、自我提高。

1.1 学习目标

理解赛题数据和目标，清楚评分体系。

完成相应报名，下载数据和结果提交打卡（可提交示例结果），熟悉比赛流程

1.1.1 了解赛题

1. 赛题概况
2. 数据概况
3. 预测指标
4. 分析赛题

1.2 赛题概况

比赛要求参赛选手根据给定的数据集，建立模型，预测金融风险。

赛题以预测金融风险为任务，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。

通过这道赛题来引导大家走进金融风控数据竞赛的世界，主要针对于于竞赛新人进行自我练习、自我提高。

1.3 数据概况

一般而言，对于数据在比赛界面都有对应的数据概况介绍（匿名特征除外），说明列的性质特征。了解列的性质会有助于我们对于数据的理解和后续分析。 Tip:匿名特征，就是未告知数据列所属的性质的特征列。

train.csv

1. id 为贷款清单分配的唯一信用证标识
2. loanAmnt　贷款金额
3. term 贷款期限（year）
4. interestRate 贷款利率
5. installment　分期付款金额
6. grade　贷款等级
7. subGrade 贷款等级之子级
8. employmentTitle 就业职称
9. employmentLength 就业年限（年）
10. homeOwnership 借款人在登记时提供的房屋所有权状况
11. annualIncome 年收入
12. verificationStatus　验证状态
13. issueDate　贷款发放的月份
14. purpose　借款人在贷款申请时的贷款用途类别
15. postCode 借款人在贷款申请中提供的邮政编码的前3位数字
16. regionCode 地区编码
17. dti　债务收入比
18. 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
19. ficoRangeLow　借款人在贷款发放时的fico所属的下限范围
20. ficoRangeHigh　借款人在贷款发放时的fico所属的上限范围
21. openAcc 借款人信用档案中未结信用额度的数量
22. pubRec　贬损公共记录的数量
23. pubRecBankruptcies　公开记录清除的数量
24. revolBal 信贷周转余额合计

25. 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
26. totalAcc 借款人信用档案中当前的信用额度总数
27. initialListStatus 贷款的初始列表状态
28. 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
29. earliesCreditLine 借款人最早报告的信用额度开立的月份
30. title 借款人提供的贷款名称
31. 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
32. 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

1.4 预测指标

竞赛采用AUC作为评价指标。AUC（Area Under Curve）被定义为 ROC曲线 下与坐标轴围成的面积。

1.4.1 分类算法常见的评估指标

1、混淆矩阵（Confuse Matrix）

1. （1）若一个实例是正类，并且被预测为正类，即为真正类TP(True Positive)
2. （2）若一个实例是正类，但是被预测为负类，即为假负类FN(False Negative)
3. （3）若一个实例是负类，但是被预测为正类，即为假正类FP(False Positive)
4. （4）若一个实例是负类，并且被预测为负类，即为真负类TN(True Negative)

2、准确率（Accuracy）

准确率是常用的一个评价指标，但是不适合样本不均衡的情况。

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

3、精确率（Precision）

又称查准率，正确预测为正样本（TP）占预测为正样本(TP+FP)的百分比。

$$Precision = \frac{TP}{TP+FP}$$

4、召回率（Recall）

又称为查全率，正确预测为正样本（TP）占正样本(TP+FN)的百分比。

$$Recall = \frac{TP}{TP+FN}$$

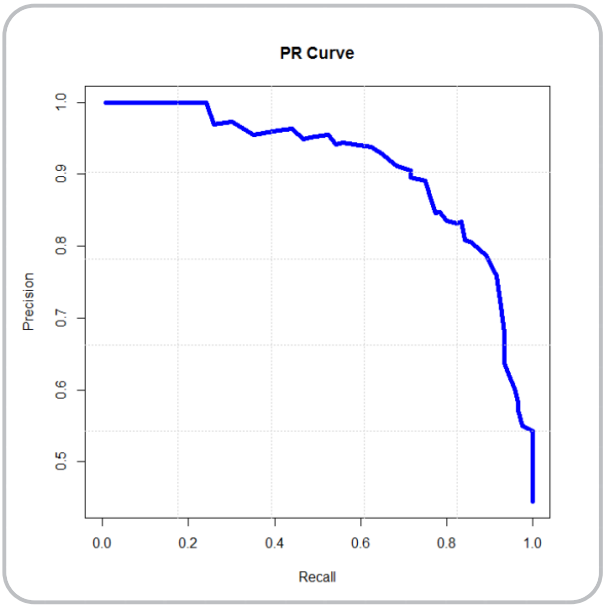
5、F1 Score

精确率和召回率是相互影响的，精确率升高则召回率下降，召回率升高则精确率下降，如果需要兼顾二者，就需要精确率、召回率的结合F1 Score。

$$F1-Score=\frac{2}{\frac{1}{Precision}+\frac{1}{Recall}}$$

6、P-R曲线（Precision-Recall Curve）

P-R曲线是描述精确率和召回率变化的曲线



7、ROC（Receiver Operating Characteristic）

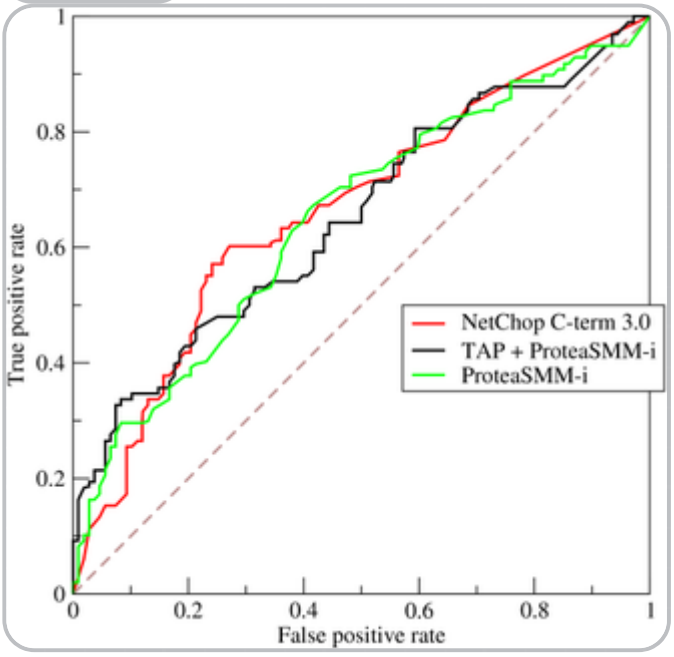
1. ROC空间将假正例率（FPR）定义为 X 轴，真正例率（TPR）定义为 Y 轴。

TPR：在所有实际为正例的样本中，被正确地判断为正例之比率。

$$TPR=\frac{TP}{TP+FN}$$

FPR：在所有实际为负例的样本中，被错误地判断为正例之比率。

$$FPR=\frac{FP}{FP+TN}$$



8、AUC(Area Under Curve)

AUC（Area Under Curve）被定义为 ROC曲线　下与坐标轴围成的面积，显然这个面积的数值不会大于1。又由于ROC曲线一般都处于y=x这条直线的上方，所以AUC的取值范围在0.5和1之间。AUC越接近1.0，检测方法真实性越高;等于0.5时，则真实性最低，无应用价值。

1.4.2 金融风控预测类常见的评估指标

1、KS(Kolmogorov-Smirnov)

K-S曲线与ROC曲线类似，不同在于

1. ROC曲线将真正例率和假正例率作为横纵轴
2. K-S曲线将真正例率和假正例率都作为纵轴，横轴则由选定的阈值来充当。

   公式如下：

   $KS = max(TPR - FPR)$

   KS不同代表的不同情况，一般情况KS值越大，模型的区分能力越强，但是也不是越大模型效果就越好，如果KS过大，模型可能存在异常，所以当KS值过高可能需要检查模型是否过拟合。以下为KS值对应的模型情况，但此对应不是唯一的，只代表大致趋势。
3. KS值<0.2,一般认为模型没有区分能力。
4. KS值[0.2,0.3],模型具有一定区分能力，勉强可以接受
5. KS值[0.3,0.5],模型具有较强的区分能力。
6. KS值大于0.75，往往表示模型有异常。

   2、ROC

   3、AUC

1.5 赛题流程

1.6 代码示例

本部分为对于数据读取和指标评价的示例。

1.6.1 数据读取pandas

```
import pandas as pd
```

```
train = pd.read_csv('train.csv')
testA = pd.read_csv('testA.csv')
```

```
print('Train data shape:',train.shape)
print('TestA data shape:',testA.shape)
```

```
Train data shape: (800000, 47)
TestA data shape: (200000, 48)
```

```
train.head()
```

	id	loanAmnt	term	interestRate	installment	grade	subGrade	employmentTitle	employmentLength	homeOwnership	...	n5	n6	n7	n8	n9	n10	n11	n12	n13	n14
0	0	35000.0	5	19.52	917.97	E	E2	320.0	2 years	2	...	9.0	8.0	4.0	12.0	2.0	7.0	0.0	0.0	0.0	2.0
1	1	18000.0	5	18.49	461.90	D	D2	219843.0	5 years	0	...	NaN	NaN	NaN	NaN	NaN	13.0	NaN	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	5.0	3.0	11.0	0.0	0.0	0.0	4.0
3	3	11000.0	3	7.26	340.96	A	A4	46854.0	10+ years	1	...	16.0	4.0	7.0	21.0	6.0	9.0	0.0	0.0	0.0	1.0
4	4	3000.0	3	12.99	101.07	C	C2	54.0	NaN	1	...	4.0	9.0	10.0	15.0	7.0	12.0	0.0	0.0	0.0	4.0

1.6.2 分类指标评价计算示例

```
## 混淆矩阵
import numpy as np
from sklearn.metrics import confusion_matrix
y_pred = [0, 1, 0, 1]
y_true = [0, 1, 1, 0]
print('混淆矩阵:\n',confusion_matrix(y_true, y_pred))
```

```
混淆矩阵:
 [[1 1]
 [1 1]]
```

```
## accuracy
from sklearn.metrics import accuracy_score
y_pred = [0, 1, 0, 1]
y_true = [0, 1, 1, 0]
print('ACC:',accuracy_score(y_true, y_pred))
```

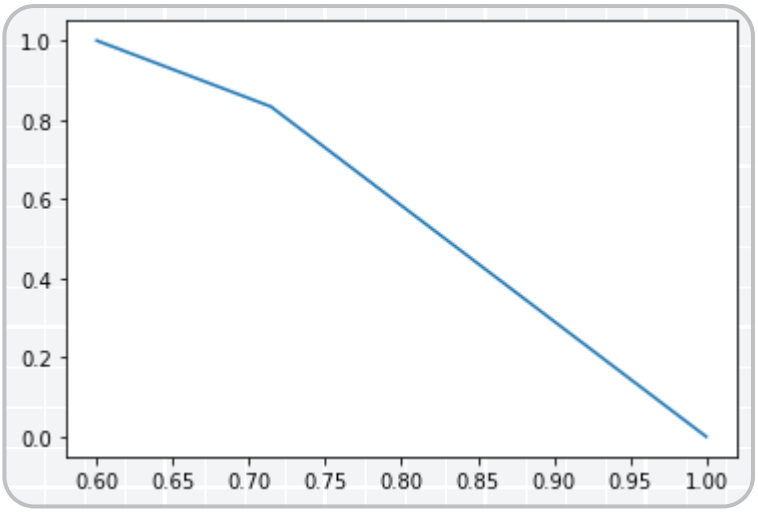
```
ACC: 0.5
```

```
## Precision,Recall,F1-score
from sklearn import metrics
y_pred = [0, 1, 0, 1]
y_true = [0, 1, 1, 0]
print('Precision',metrics.precision_score(y_true, y_pred))
print('Recall',metrics.recall_score(y_true, y_pred))
print('F1-score:',metrics.f1_score(y_true, y_pred))
```

```
Precision 0.5
Recall 0.5
F1-score: 0.5
```

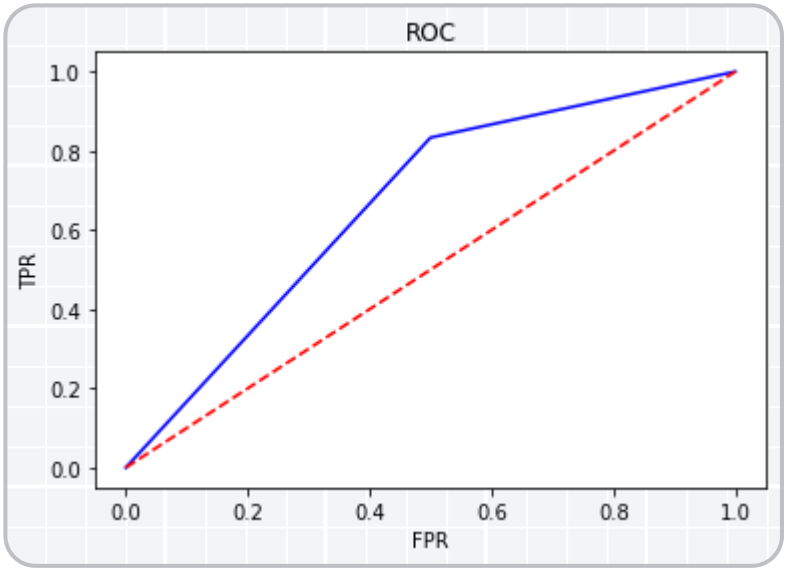
```
## P-R曲线
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
y_pred = [0, 1, 1, 0, 1, 1, 0, 1, 1, 1]
y_true = [0, 1, 1, 0, 1, 0, 1, 1, 0, 1]
precision, recall, thresholds = precision_recall_curve(y_true, y_pred)
plt.plot(precision, recall)
```

```
[<matplotlib.lines.Line2D at 0x2170d0d6108>]
```




```
## ROC曲线
from sklearn.metrics import roc_curve
y_pred = [0, 1, 1, 0, 1, 1, 0, 1, 1, 1]
y_true = [0, 1, 1, 0, 1, 0, 1, 1, 0, 1]
FPR,TPR,thresholds=roc_curve(y_true, y_pred)
plt.title('ROC')
plt.plot(FPR, TPR,'b')
plt.plot([0,1],[0,1],'r--')
plt.ylabel('TPR')
plt.xlabel('FPR')
```

```
Text(0.5, 0, 'FPR')
```




```
## AUC
import numpy as np
from sklearn.metrics import roc_auc_score
y_true = np.array([0, 0, 1, 1])
y_scores = np.array([0.1, 0.4, 0.35, 0.8])
print('AUC socre:',roc_auc_score(y_true, y_scores))
```

```
AUC socre: 0.75
```

```
## KS值 在实际操作时往往使用ROC曲线配合求出KS值
from sklearn.metrics import roc_curve
y_pred = [0, 1, 1, 0, 1, 1, 0, 1, 1, 1]
y_true = [0, 1, 1, 0, 1, 0, 1, 1, 1, 1]
FPR,TPR,thresholds=roc_curve(y_true, y_pred)
KS=abs(FPR-TPR).max()
print('KS值：',KS)
```

```
KS值： 0.5238095238095237
```

1.7 经验总结

赛题理解是开始比赛的第一步，赛题的理解有助于对竞赛全局的把握。通过赛题理解有助于对赛题的业务逻辑把握，对于后期的特征工程构建和模型选择都尤为重要。

1. 在开始比赛之前要对赛题进行充分的了解。
2. 比赛什么时候开始，什么时候结束，什么时候换B榜数据。
3. 和该比赛有没有类似的比赛可以参考借鉴。

4. 线上提交结果的次数往往是有限的，提前了解每日可以提交的次数。

5. 比赛使用的是什么评价指标，可以选择相同的评价指标作为线下验证的方式。

1.8 拓展知识——评分卡

评分卡是一张拥有分数刻度会让相应阈值的表。信用评分卡是用于用户信用的一张刻度表。以下代码是一个非标准评分卡的代码流程，用于刻画用户的信用评分。评分卡是金融风控中常用的一种对于用户信用进行刻画的手段哦！

```
#评分卡 不是标准评分卡
def Score(prob,P0=600,PDO=20,badrate=None,goodrate=None):
    P0 = P0
    PDO = PDO
    theta0 = badrate/goodrate
    B = PDO/np.log(2)
    A = P0 + B*np.log(2*theta0)
    score = A-B*np.log(prob/(1-prob))
    return score
```

END.

【 杨冰楠：Datawhale成员，金融风控爱好者。】

关于Datawhale：

Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

2 Task2 数据分析

此部分为零基础入门金融风控的 Task2 数据分析部分，带你来了解数据，熟悉数据，为后续的特征工程做准备，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 零基础入门金融风控之贷款违约

目的：

1. 1.EDA价值主要在于熟悉了解整个数据集的基本情况（缺失值，异常值），对数据集进行验证是否可以进行接下来的机器学习或者深度学习建模.
2. 2.了解变量间的相互关系、变量与预测值之间的存在关系。
3. 3.为特征工程做准备

预测地址：

2.1 学习目标

1. 学习如何对数据集整体概况进行分析，包括数据集的基本情况（缺失值，异常值）
2. 学习了解变量间的相互关系、变量与预测值之间的存在关系
3. 完成相应学习打卡任务

2.2 内容介绍

1. 数据总体了解：
    a. 读取数据集并了解数据集大小，原始特征维度；
    b. 通过info熟悉数据类型；
    c. 粗略查看数据集中各特征基本统计量；
2. 缺失值和唯一值：
    a. 查看数据缺失值情况

b. 查看唯一值特征情况

3. 深入数据-查看数据类型
   a. 类别型数据
   b. 数值型数据
      - 离散数值型数据
      - 连续数值型数据
4. 数据间相关关系
   a. 特征和特征之间关系
   b. 特征和目标变量之间关系
5. 用pandas_profiling生成数据报告

2.3 代码示例

2.3.1 导入数据分析及可视化过程需要的库

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import warnings
warnings.filterwarnings('ignore')
```

```
/Users/exudingtao/opt/anaconda3/lib/python3.7/site-
packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated.
Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

以上库都是pip install 安装就好，如果本机有python2,python3两个python环境傻傻分不清哪个的话,可以pip3 install 。或者直接在notebook中'!pip3 install '安装。

说明：

本次数据分析探索，尤其可视化部分均选取某些特定变量进行了举例，所以它只是一个方法的展示而不是整个赛题数据分析的解决方案。

2.3.2 读取文件

```
data_train = pd.read_csv('./train.csv')
```

```
data_test_a = pd.read_csv('./testA.csv')
```

读取文件的拓展知识

1. pandas读取数据时相对路径载入报错时，尝试使用os.getcwd()查看当前工作目录。
2. TSV与CSV的区别：
   a. 从名称上即可知道，TSV是用制表符（Tab,'\t'）作为字段值的分隔符；CSV是用半角逗号（','）作为字段值的分隔符；
   b. Python对TSV文件的支持：
   Python的csv模块准确的讲应该叫做dsv模块，因为它实际上是支持范式的分隔符分隔值文件（DSV，delimiter-separated values）的。
   delimiter参数值默认为半角逗号，即默认将被处理文件视为CSV。当delimiter='\t'时，被处理文件就是TSV。
3. 读取文件的部分（适用于文件特别大的场景）
   a. 通过nrows参数，来设置读取文件的前多少行，nrows是一个大于等于0的整数。
   b. 分块读取

```
data_train_sample = pd.read_csv("./train.csv",nrows=5)
```

```
#设置chunksize参数，来控制每次迭代数据的大小
chunker = pd.read_csv("./train.csv",chunksize=5)
for item in chunker:
    print(type(item))
    #<class 'pandas.core.frame.DataFrame'>
    print(len(item))
    #5
```

2.3.3 总体了解

查看数据集的样本个数和原始特征维度

```
data_test_a.shape
```

```
(200000, 48)
```

```
data_train.shape
```

```
(800000, 47)
```

```
data_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', 'n2.1', 'n4', 'n5', 'n6', 'n7', 'n8',
       'n9', 'n10', 'n11', 'n12', 'n13', 'n14'],
      dtype='object')
```

查看一下具体的列名，赛题理解部分已经给出具体的特征含义，这里方便阅读再给一下：

1. id 为贷款清单分配的唯一信用证标识
2. loanAmnt　贷款金额
3. term 贷款期限（year）
4. interestRate 贷款利率
5. installment　分期付款金额
6. grade　贷款等级
7. subGrade 贷款等级之子级
8. employmentTitle 就业职称
9. employmentLength 就业年限（年）
10. homeOwnership 借款人在登记时提供的房屋所有权状况
11. annualIncome 年收入
12. verificationStatus　验证状态
13. issueDate　贷款发放的月份
14. purpose　借款人在贷款申请时的贷款用途类别
15. postCode 借款人在贷款申请中提供的邮政编码的前3位数字
16. regionCode 地区编码
17. dti　债务收入比
18. 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
19. ficoRangeLow　借款人在贷款发放时的fico所属的下限范围
20. ficoRangeHigh　借款人在贷款发放时的fico所属的上限范围
21. openAcc 借款人信用档案中未结信用额度的数量
22. pubRec　贬损公共记录的数量
23. pubRecBankruptcies　公开记录清除的数量
24. revolBal 信贷周转余额合计
25. 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
26. totalAcc 借款人信用档案中当前的信用额度总数

27. initialListStatus 贷款的初始列表状态

28. 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

29. earliesCreditLine 借款人最早报告的信用额度开立的月份

30. title 借款人提供的贷款名称

31. 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

32. 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

通过info()来熟悉数据类型

```
data_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  n2.1                 759730 non-null  float64
 36  n4                   766761 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
```

总体粗略的查看数据集各个特征的一些基本统计量

```
data_train.describe()
```

	id	loanAmnt	term	interestRate	installment	employmentTitle	homeOwnership	annualIncome	verificationStatus	isDefault	...	n5	n6	n7	n8	n9	n10	n11	n12	n13	n14
count	800000.000000	800000.000000	800000.000000	800000.000000	800000.000000	799999.000000	800000.000000	8.000000e+05	800000.000000	800000.000000	...	759730.000000	759730.000000	759730.000000	759729.000000	759730.000000	766761.000000	730248.000000	759730.000000	759730.000000	759730.000000
mean	399999.500000	14416.818875	3.482745	13.238391	437.947723	72005.351714	0.614213	7.613391e+04	1.009683	0.199513	...	8.107937	8.575994	8.282953	14.622488	5.592345	11.643896	0.000815	0.003384	0.089366	2.178606
std	230940.252015	8716.086178	0.855832	4.765757	261.460393	106585.640204	0.675749	6.894751e+04	0.782716	0.399634	...	4.799210	7.400536	4.561689	8.124610	3.216184	5.484104	0.030075	0.062041	0.509069	1.844377
min	0.000000	500.000000	3.000000	5.310000	15.690000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	...	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	199999.750000	8000.000000	3.000000	9.750000	248.450000	427.000000	0.000000	4.560000e+04	0.000000	0.000000	...	5.000000	4.000000	5.000000	9.000000	3.000000	8.000000	0.000000	0.000000	0.000000	1.000000
50%	399999.500000	12000.000000	3.000000	12.740000	375.135000	7755.000000	1.000000	6.500000e+04	1.000000	0.000000	...	7.000000	7.000000	7.000000	13.000000	5.000000	11.000000	0.000000	0.000000	0.000000	2.000000
75%	599999.250000	20000.000000	3.000000	15.990000	580.710000	117663.500000	1.000000	9.000000e+04	2.000000	0.000000	...	11.000000	11.000000	10.000000	19.000000	7.000000	14.000000	0.000000	0.000000	0.000000	3.000000
max	799999.000000	40000.000000	5.000000	30.990000	1715.420000	378351.000000	5.000000	1.099920e+07	2.000000	1.000000	...	70.000000	132.000000	79.000000	128.000000	45.000000	82.000000	4.000000	4.000000	39.000000	30.000000

```
data_train.head(3).append(data_train.tail(3))
```

	id	loanAmnt	term	interestRate	installment	grade	subGrade	employmentTitle	employmentLength	homeOwnership	...	n5	n6	n7	n8	n9	n10	n11	n12	n13	n14
0	0	35000.0	5	19.52	917.97	E	E2	320.0	2 years	2	...	9.0	8.0	4.0	12.0	2.0	7.0	0.0	0.0	0.0	2.0
1	1	18000.0	5	18.49	461.90	D	D2	219843.0	5 years	0	...	NaN	NaN	NaN	NaN	NaN	13.0	NaN	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	5.0	3.0	11.0	0.0	0.0	0.0	4.0
799997	799997	6000.0	3	13.33	203.12	C	C3	2582.0	10+ years	1	...	4.0	26.0	4.0	10.0	4.0	5.0	0.0	0.0	1.0	4.0
799998	799998	19200.0	3	6.92	592.14	A	A4	151.0	10+ years	0	...	10.0	6.0	12.0	22.0	8.0	16.0	0.0	0.0	0.0	5.0
799999	799999	9000.0	3	11.06	294.91	B	B3	13.0	5 years	0	...	3.0	4.0	4.0	8.0	3.0	7.0	0.0	0.0	0.0	2.0

2.3.4 查看数据集中特征缺失值，唯一值等

查看缺失值

```
print(f'There are {data_train.isnull().any().sum()} columns in train dataset with missing values.')
```

```
There are 22 columns in train dataset with missing values.
```

上面得到训练集有22列特征有缺失值，进一步查看缺失特征中缺失率大于50%的特征

```
have_null_fea_dict = (data_train.isnull().sum()/len(data_train)).to_dict()
fea_null_moreThanHalf = {}
for key,value in have_null_fea_dict.items():
    if value > 0.5:
        fea_null_moreThanHalf[key] = value
```

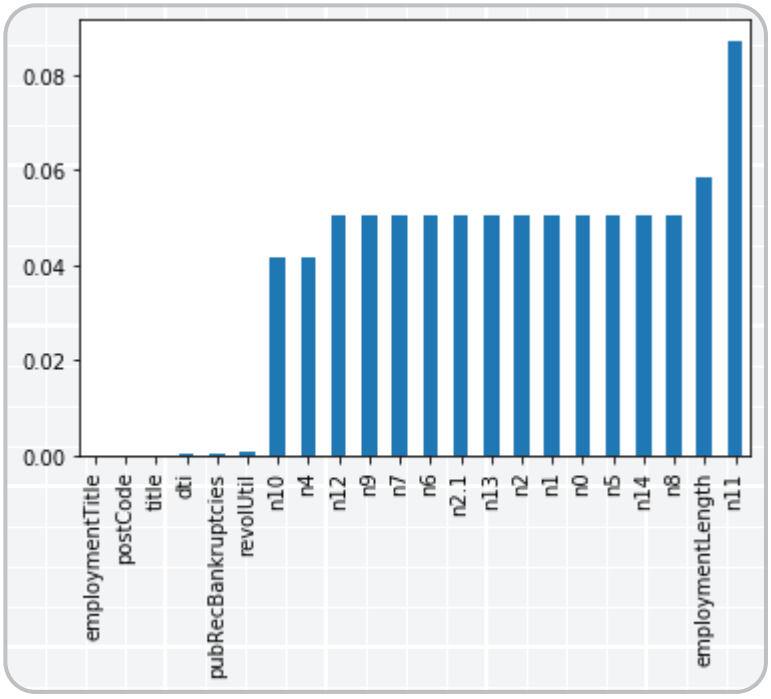
```
fea_null_moreThanHalf
```

```
{}
```

具体的查看缺失特征及缺失率

```
# nan可视化
missing = data_train.isnull().sum()/len(data_train)
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1229ab890>
```



了解哪些列存在 “nan”, 并可以把nan的个数打印，主要的目的在于 nan存在的个数是否真的很大，如果很小一般选择填充，如果使用lgb等树模型可以直接空缺，让树自己去优化，但如果nan存在的过多、可以考虑删掉

查看训练集测试集中特征属性只有一值的特征

```
one_value_fea = [col for col in data_train.columns if data_train[col].nunique() <= 1]
```

```
one_value_fea_test = [col for col in data_test_a.columns if data_test_a[col].nunique() <= 1]
```

```
one_value_fea
```

```
['policyCode']
```

```
one_value_fea_test
```

```
['policyCode']
```

```
print(f'There are {len(one_value_fea)} columns in train dataset with one unique value.')
print(f'There are {len(one_value_fea_test)} columns in test dataset with one unique value.')
```

```
There are 1 columns in train dataset with one unique value.
There are 1 columns in test dataset with one unique value.
```

总结：

47列数据中有22列都缺少数据，这在现实世界中很正常。‘policyCode’具有一个唯一值（或全部缺失）。有很多连续变量和一些分类变量。

2.3.5 查看特征的数值类型有哪些，对象类型有哪些

1. 特征一般都是由类别型特征和数值型特征组成
2. 类别型特征有时具有非数值关系，有时也具有数值关系。比如‘grade’中的等级A，B，C等，是否只是单纯的分类，还是A优于其他要结合业务判断。
3. 数值型特征本是可以直接入模的，但往往风控人员要对其做分箱，转化为WOE编码进而做标准评分卡等操作。从模型效果上来看，特征分箱主要是为了降低变量的复杂性，减少变量噪音对模型的影响，提高自变量和因变量的相关度。从而使模型更加稳定。

```
numerical_fea = list(data_train.select_dtypes(exclude=['object']).columns)
category_fea = list(filter(lambda x: x not in numerical_fea,list(data_train.columns)))
```

```
numerical_fea
```

```
['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',
```

```
 'n2.1',
 'n4',
 'n5',
 'n6',
 'n7',
 'n8',
 'n9',
 'n10',
 'n11',
 'n12',
 'n13',
 'n14']
```

```
category_fea
```

```
['grade', 'subGrade', 'employmentLength', 'issueDate', 'earliesCreditLine']
```

```
data_train.grade
```

```
0         E
1         D
2         D
3         A
4         C
         ..
799995    C
799996    A
799997    C
799998    A
799999    B
Name: grade, Length: 800000, dtype: object
```

数值型变量分析，数值型肯定是包括连续型变量和离散型变量的，找出来

1. 划分数值型变量中的连续变量和分类变量

```python
#过滤数值型类别特征
def get_numerical_serial_fea(data,feas):
    numerical_serial_fea = []
    numerical_noserial_fea = []
    for fea in feas:
        temp = data[fea].nunique()
        if temp <= 10:
            numerical_noserial_fea.append(fea)
            continue
        numerical_serial_fea.append(fea)
    return numerical_serial_fea,numerical_noserial_fea
numerical_serial_fea,numerical_noserial_fea =
get_numerical_serial_fea(data_train,numerical_fea)
```

```python
numerical_serial_fea
```

```
['id',
 'loanAmnt',
 'interestRate',
 'installment',
 'employmentTitle',
 'annualIncome',
 'purpose',
 'postCode',
 'regionCode',
 'dti',
 'delinquency_2years',
 'ficoRangeLow',
 'ficoRangeHigh',
 'openAcc',
 'pubRec',
 'pubRecBankruptcies',
 'revolBal',
 'revolUtil',
 'totalAcc',
 'title',
 'n0',
 'n1',
 'n2',
 'n2.1',
 'n4',
 'n5',
 'n6',
 'n7',
 'n8',
```

```
  'n9',
  'n10',
  'n13',
  'n14']
```

```
numerical_noserial_fea
```

```
['term',
 'homeOwnership',
 'verificationStatus',
 'isDefault',
 'initialListStatus',
 'applicationType',
 'policyCode',
 'n11',
 'n12']
```

1. 数值类别型变量分析

```
data_train['term'].value_counts()#离散型变量
```

```
3    606902
5    193098
Name: term, dtype: int64
```

```
data_train['homeOwnership'].value_counts()#离散型变量
```

```
0    395732
1    317660
2     86309
3       185
5        81
4        33
Name: homeOwnership, dtype: int64
```

```
data_train['verificationStatus'].value_counts()#离散型变量
```

```
1    309810
2    248968
0    241222
Name: verificationStatus, dtype: int64
```

```
data_train['initialListStatus'].value_counts()#离散型变量
```

```
0    466438
1    333562
Name: initialListStatus, dtype: int64
```

```
data_train['applicationType'].value_counts()#离散型变量
```

```
0    784586
1     15414
Name: applicationType, dtype: int64
```

```
data_train['policyCode'].value_counts()#离散型变量，无用，全部一个值
```

```
1.0    800000
Name: policyCode, dtype: int64
```

```
data_train['n11'].value_counts()#离散型变量，相差悬殊，用不用再分析
```

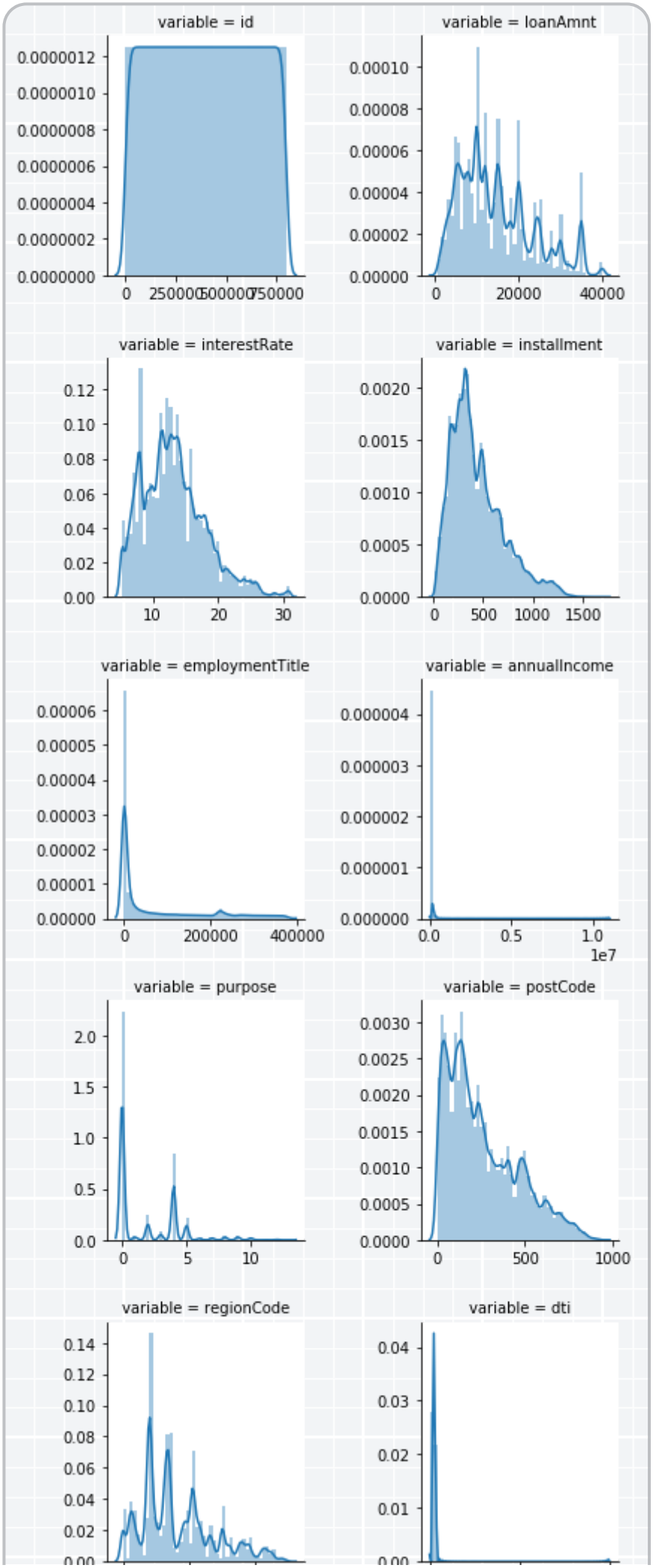
```
0.0    729682
1.0       540
2.0        24
4.0         1
3.0         1
Name: n11, dtype: int64
```

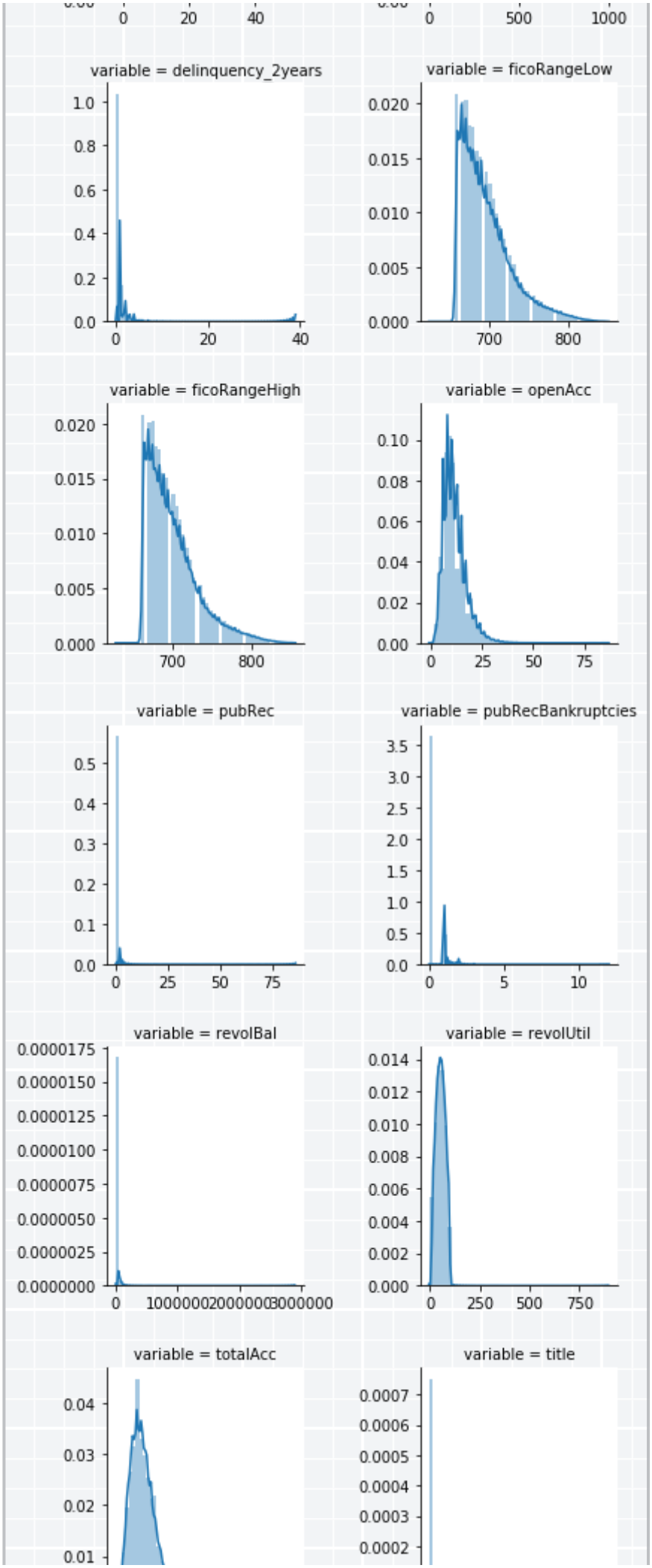
```
data_train['n12'].value_counts()#离散型变量，相差悬殊，用不用再分析
```

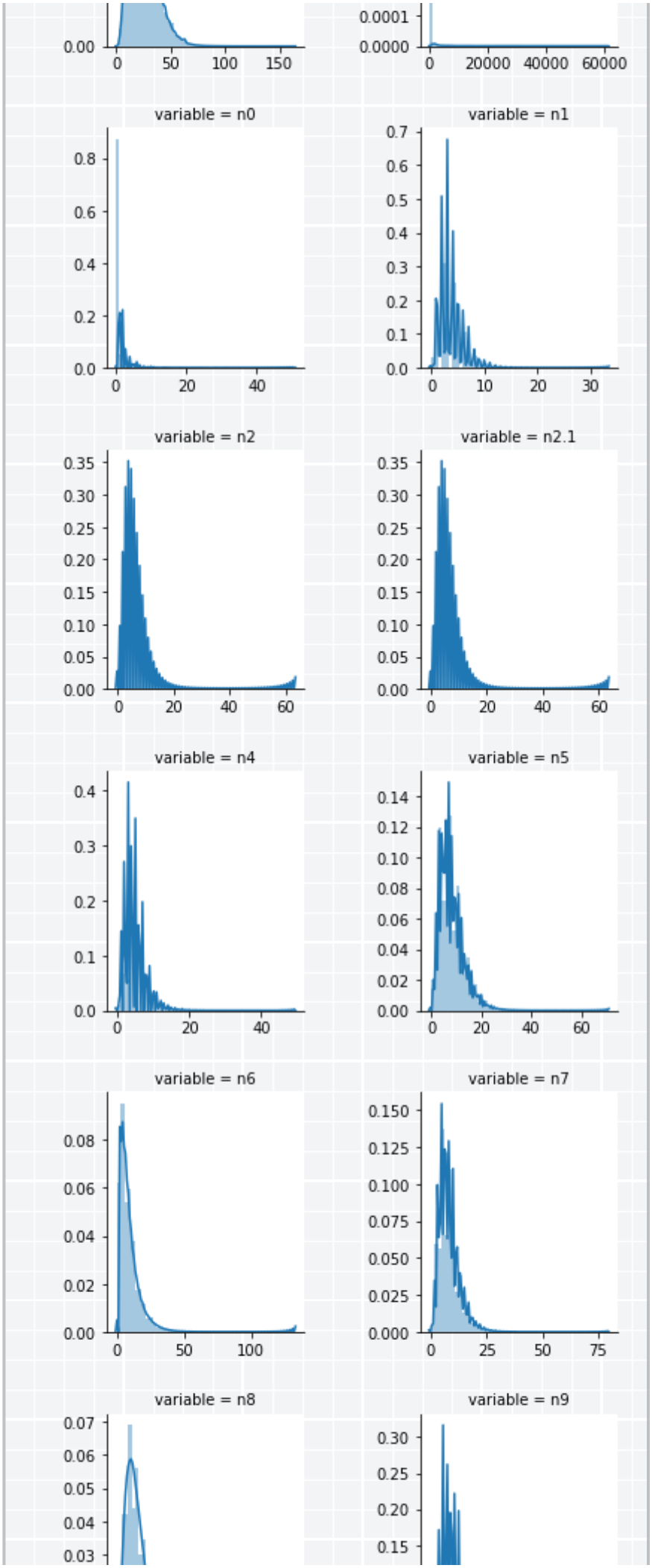
```
0.0    757315
1.0      2281
2.0       115
3.0        16
4.0         3
Name: n12, dtype: int64
```

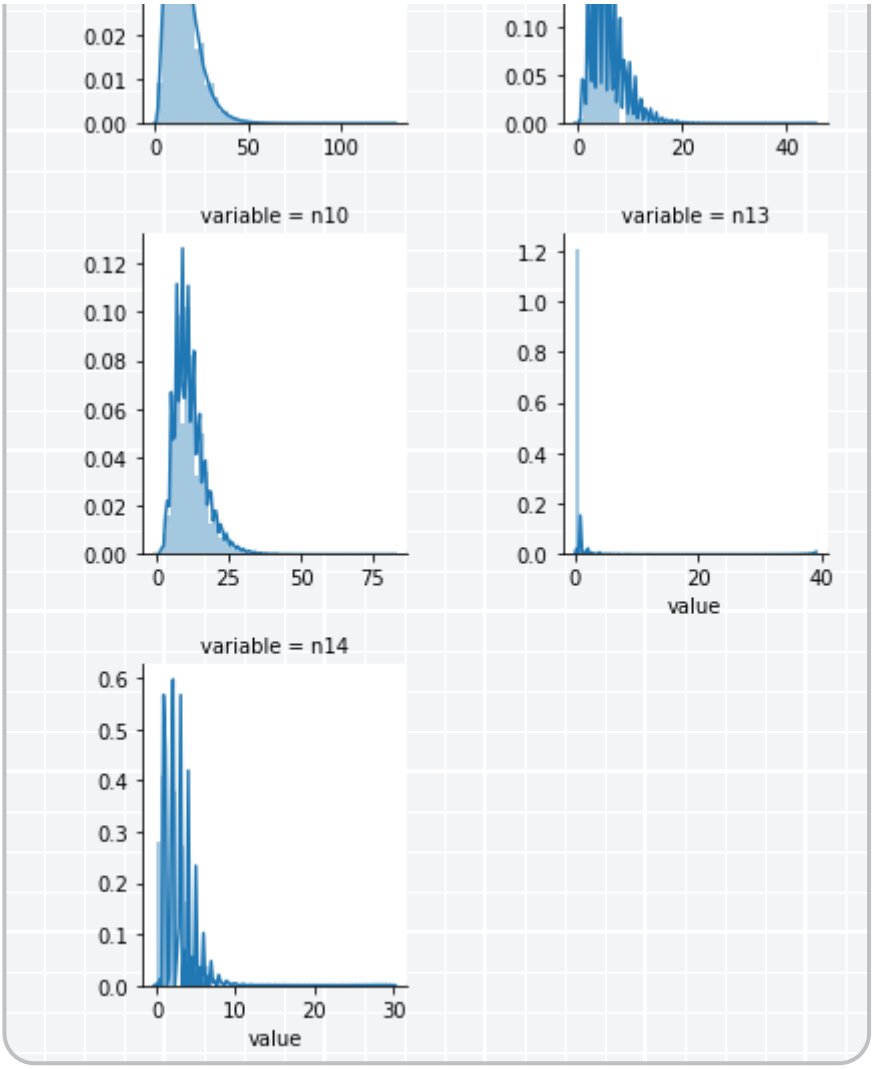
1. 数值连续型变量分析

```
#每个数字特征得分布可视化
f = pd.melt(data_train, value_vars=numerical_serial_fea)
g = sns.FacetGrid(f, col="variable",  col_wrap=2, sharex=False, sharey=False)
g = g.map(sns.distplot, "value")
```


variable = id
variable = loanAmnt
variable = interestRate
variable = installment
variable = employmentTitle
variable = annualIncome
1e7
variable = purpose
variable = postCode
variable = regionCode
variable = dti


variable = delinquency_2years
variable = ficoRangeLow
variable = ficoRangeHigh
variable = openAcc
variable = pubRec
variable = pubRecBankruptcies
variable = revolBal
variable = revolUtil
variable = totalAcc
variable = title

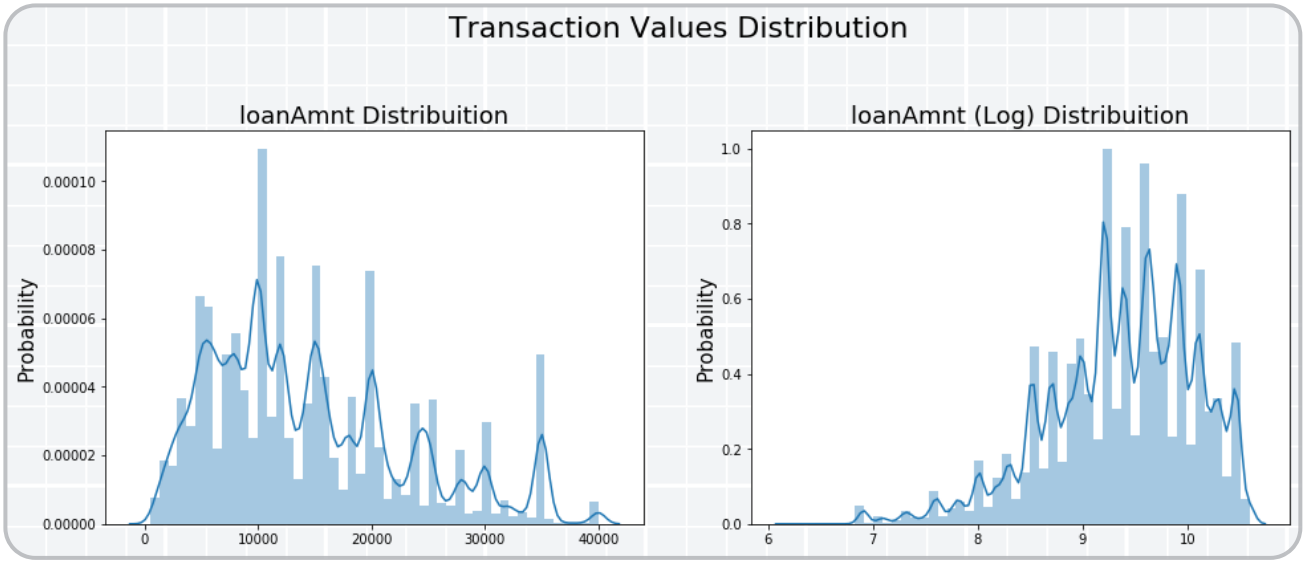

variable = n0
variable = n1
variable = n2
variable = n2.1
variable = n4
variable = n5
variable = n6
variable = n7
variable = n8
variable = n9



1. 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，如果不符合正太分布的变量可以log化后再观察下是否符合正态分布。
2. 如果想统一处理一批数据变标准化 必须把这些之前已经正态化的数据提出

```
#Ploting Transaction Amount Values Distribution
plt.figure(figsize=(16,12))
plt.suptitle('Transaction Values Distribution', fontsize=22)
plt.subplot(221)
sub_plot_1 = sns.distplot(data_train['loanAmnt'])
sub_plot_1.set_title("loanAmnt Distribuition", fontsize=18)
sub_plot_1.set_xlabel("")
sub_plot_1.set_ylabel("Probability", fontsize=15)

plt.subplot(222)
sub_plot_2 = sns.distplot(np.log(data_train['loanAmnt']))
sub_plot_2.set_title("loanAmnt (Log) Distribuition", fontsize=18)
sub_plot_2.set_xlabel("")
sub_plot_2.set_ylabel("Probability", fontsize=15)
```

```
Text(0, 0.5, 'Probability')
```



1. 非数值类别型变量分析

```
category_fea
```

```
['grade', 'subGrade', 'employmentLength', 'issueDate', 'earliesCreditLine']
```

```
data_train['grade'].value_counts()
```

```
B    233690
C    227118
A    139661
D    119453
E     55661
F     19053
G      5364
Name: grade, dtype: int64
```

```
data_train['subGrade'].value_counts()
```

```
C1    50763
B4    49516
B5    48965
B3    48600
C2    47068
C3    44751
```

```
C4    44272
B2    44227
B1    42382
C5    40264
A5    38045
A4    30928
D1    30538
D2    26528
A1    25909
D3    23410
A3    22655
A2    22124
D4    21139
D5    17838
E1    14064
E2    12746
E3    10925
E4     9273
E5     8653
F1     5925
F2     4340
F3     3577
F4     2859
F5     2352
G1     1759
G2     1231
G3      978
G4      751
G5      645
Name: subGrade, dtype: int64
```

```
data_train['employmentLength'].value_counts()
```

```
10+ years    262753
2 years       72358
< 1 year      64237
3 years       64152
1 year        52489
5 years       50102
4 years       47985
6 years       37254
8 years       36192
7 years       35407
9 years       30272
Name: employmentLength, dtype: int64
```

```
data_train['issueDate'].value_counts()
```

```
2016-03-01    29066
2015-10-01    25525
2015-07-01    24496
2015-12-01    23245
2014-10-01    21461
              ...
2007-08-01       23
2007-07-01       21
2008-09-01       19
2007-09-01        7
2007-06-01        1
Name: issueDate, Length: 139, dtype: int64
```

```
data_train['earliesCreditLine'].value_counts()
```

```
Aug-2001    5567
Sep-2003    5403
Aug-2002    5403
Oct-2001    5258
Aug-2000    5246
            ...
May-1960       1
Apr-1958       1
Feb-1960       1
Aug-1946       1
Mar-1958       1
Name: earliesCreditLine, Length: 720, dtype: int64
```

```
data_train['isDefault'].value_counts()
```

```
0    640390
1    159610
Name: isDefault, dtype: int64
```

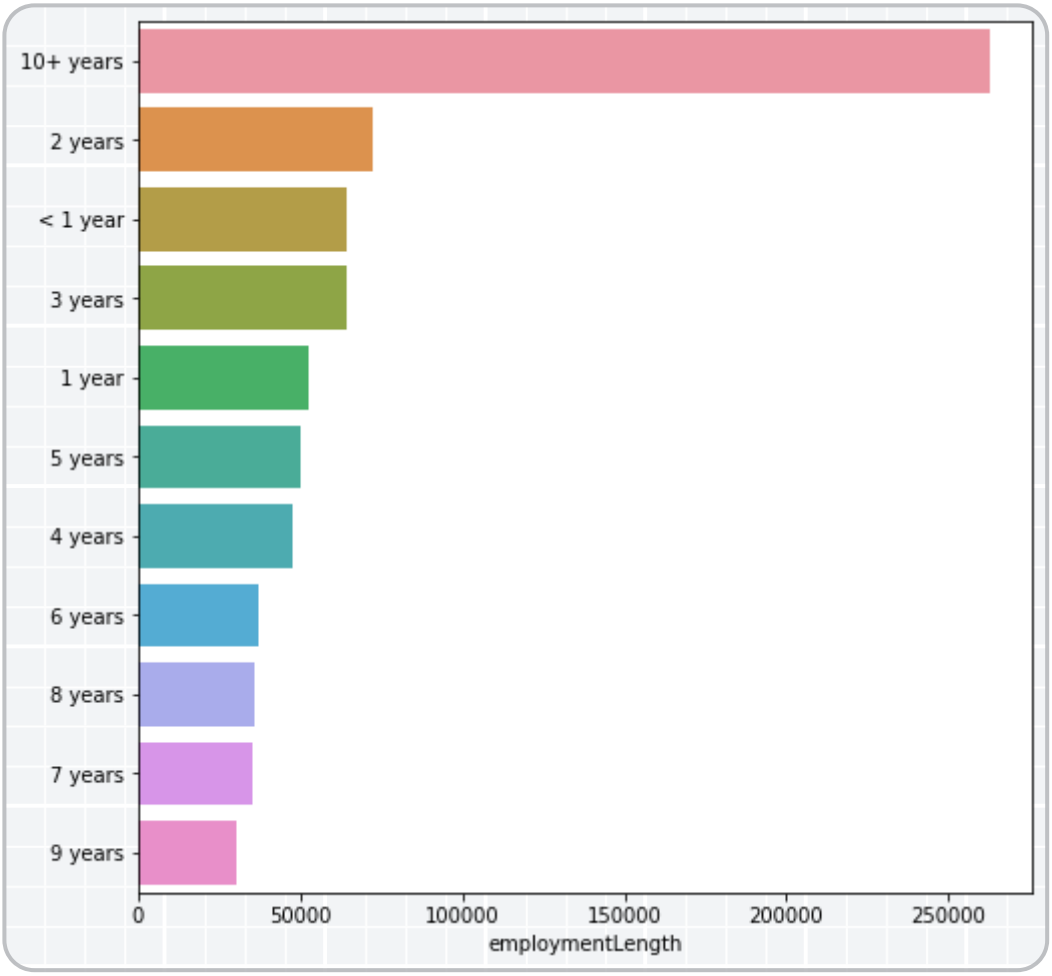
总结：

1. 上面我们用value_counts()等函数看了特征属性的分布，但是图表是概括原始信息最便捷的方式。
2. 数无形时少直觉。
3. 同一份数据集，在不同的尺度刻画上显示出来的图形反映的规律是不一样的。python将数据转化成图表，但结论是否正确需要由你保证。

2.3.6 变量分布可视化

2.3.6.1 单一变量分布可视化

```
plt.figure(figsize=(8, 8))
sns.barplot(data_train["employmentLength"].value_counts(dropna=False)[:20],
            data_train["employmentLength"].value_counts(dropna=False).keys()[:20])
plt.show()
```

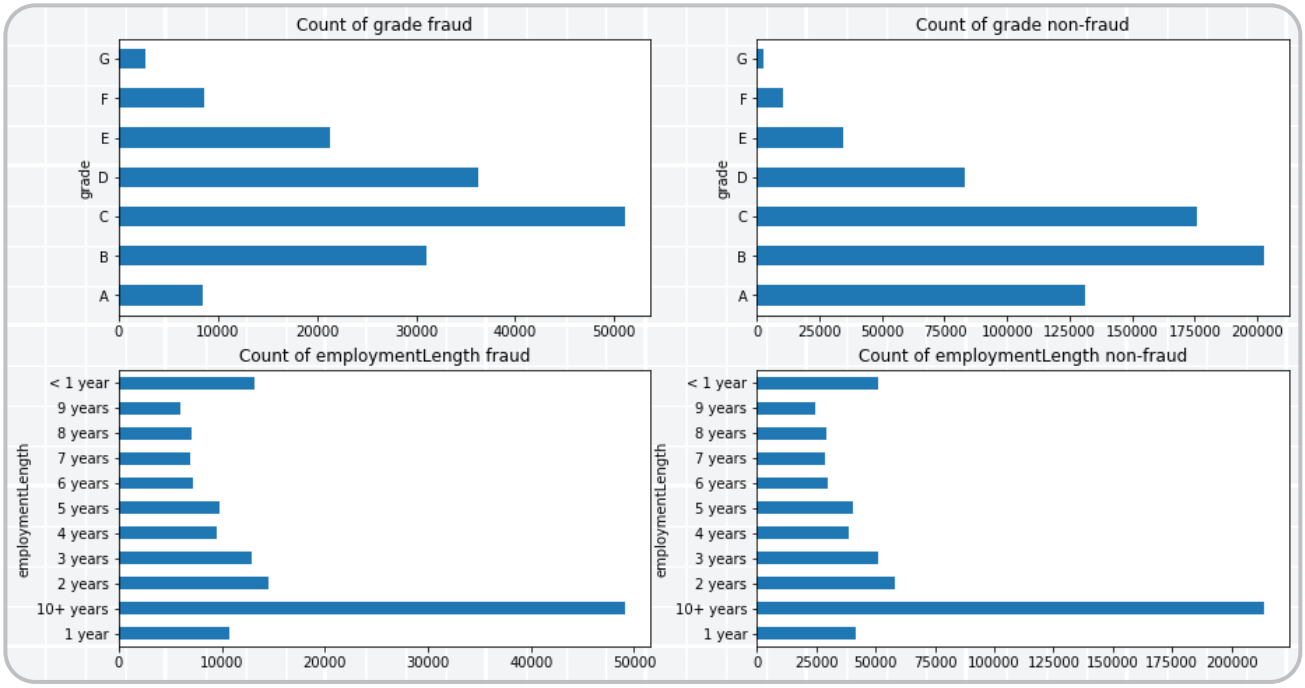


2.3.6.2 根绝y值不同可视化x某个特征的分布

1. 首先查看类别型变量在不同y值上的分布

```
train_loan_fr = data_train.loc[data_train['isDefault'] == 1]
train_loan_nofr = data_train.loc[data_train['isDefault'] == 0]
```

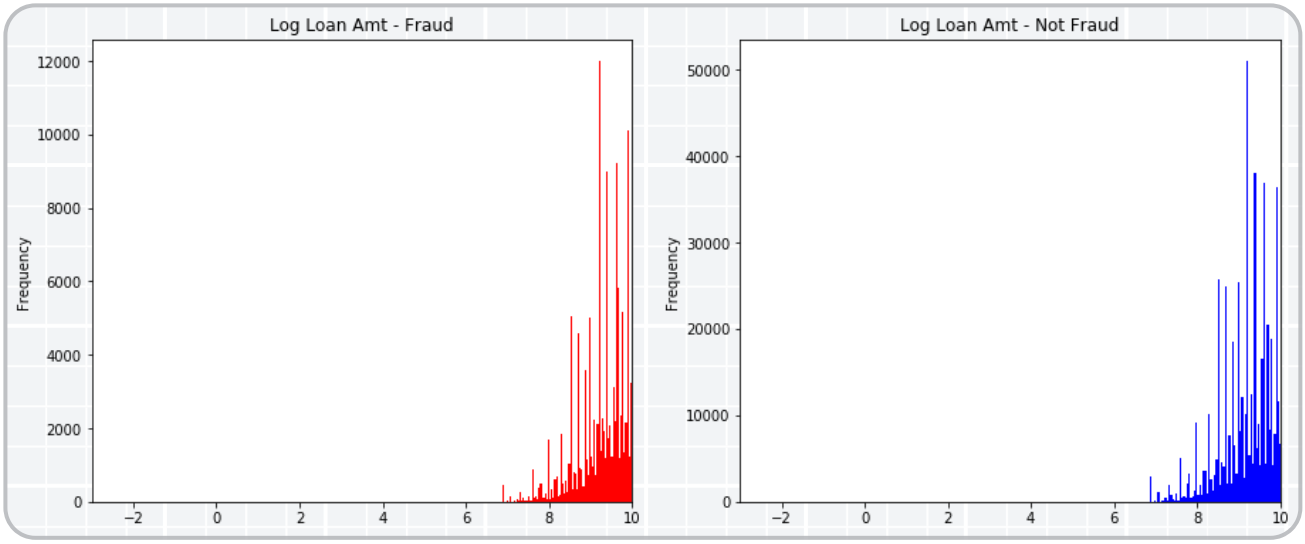
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 8))
train_loan_fr.groupby('grade')['grade'].count().plot(kind='barh', ax=ax1, title='Count of grade fraud')
train_loan_nofr.groupby('grade')['grade'].count().plot(kind='barh', ax=ax2, title='Count of grade non-fraud')
train_loan_fr.groupby('employmentLength')['employmentLength'].count().plot(kind='barh', ax=ax3, title='Count of employmentLength fraud')
train_loan_nofr.groupby('employmentLength')['employmentLength'].count().plot(kind='barh', ax=ax4, title='Count of employmentLength non-fraud')
plt.show()
```



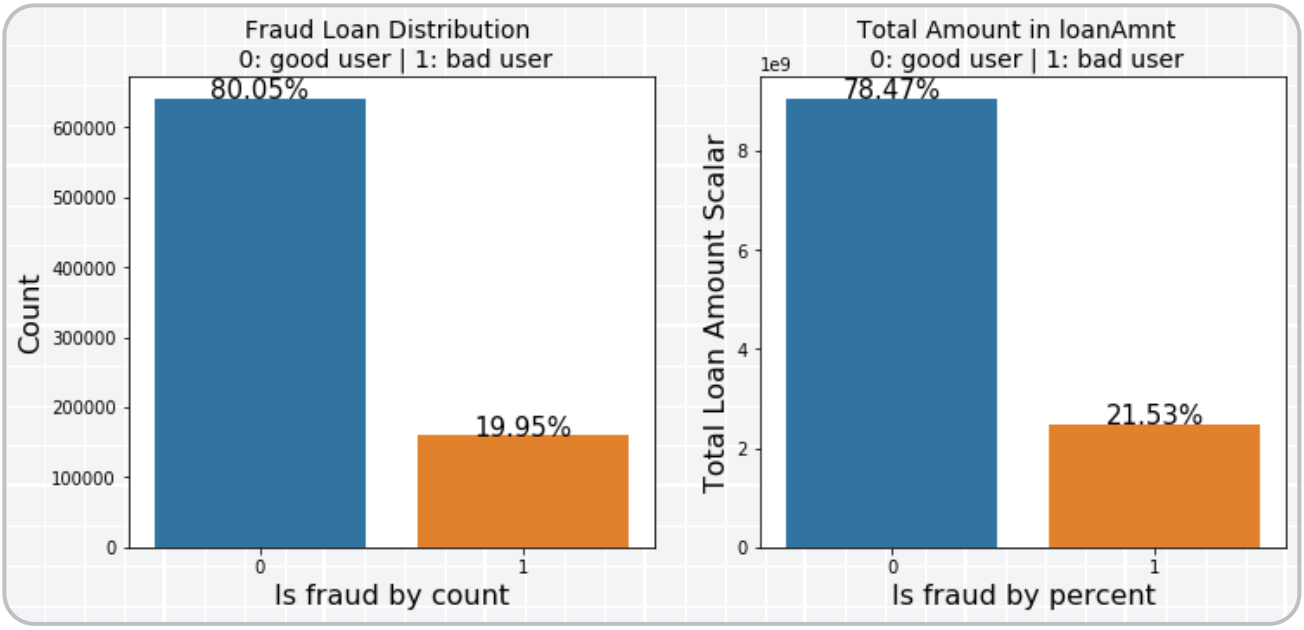
1. 其次查看连续型变量在不同y值上的分布

```
fig, ((ax1, ax2)) = plt.subplots(1, 2, figsize=(15, 6))
data_train.loc[data_train['isDefault'] == 1] \
    ['loanAmnt'].apply(np.log) \
    .plot(kind='hist',
          bins=100,
          title='Log Loan Amt - Fraud',
          color='r',
          xlim=(-3, 10),
         ax= ax1)
data_train.loc[data_train['isDefault'] == 0] \
    ['loanAmnt'].apply(np.log) \
    .plot(kind='hist',
          bins=100,
          title='Log Loan Amt - Not Fraud',
          color='b',
          xlim=(-3, 10),
         ax=ax2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x126a44b50>
```




```
total = len(data_train)
total_amt = data_train.groupby(['isDefault'])['loanAmnt'].sum().sum()
plt.figure(figsize=(12,5))
plt.subplot(121)##1代表行，2代表列，所以一共有2个图，1代表此时绘制第一个图。
plot_tr = sns.countplot(x='isDefault',data=data_train)#data_train‘isDefault’这个特征每种类别的数量**
plot_tr.set_title("Fraud Loan Distribution \n 0: good user | 1: bad user", fontsize=14)
plot_tr.set_xlabel("Is fraud by count", fontsize=16)
plot_tr.set_ylabel('Count', fontsize=16)
for p in plot_tr.patches:
    height = p.get_height()
    plot_tr.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total*100),
            ha="center", fontsize=15)

percent_amt = (data_train.groupby(['isDefault'])['loanAmnt'].sum())
percent_amt = percent_amt.reset_index()
plt.subplot(122)
plot_tr_2 = sns.barplot(x='isDefault', y='loanAmnt',  dodge=True, data=percent_amt)
plot_tr_2.set_title("Total Amount in loanAmnt  \n 0: good user | 1: bad user", fontsize=14)
plot_tr_2.set_xlabel("Is fraud by percent", fontsize=16)
plot_tr_2.set_ylabel('Total Loan Amount Scalar', fontsize=16)
for p in plot_tr_2.patches:
    height = p.get_height()
    plot_tr_2.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}%'.format(height/total_amt * 100),
            ha="center", fontsize=15)
```

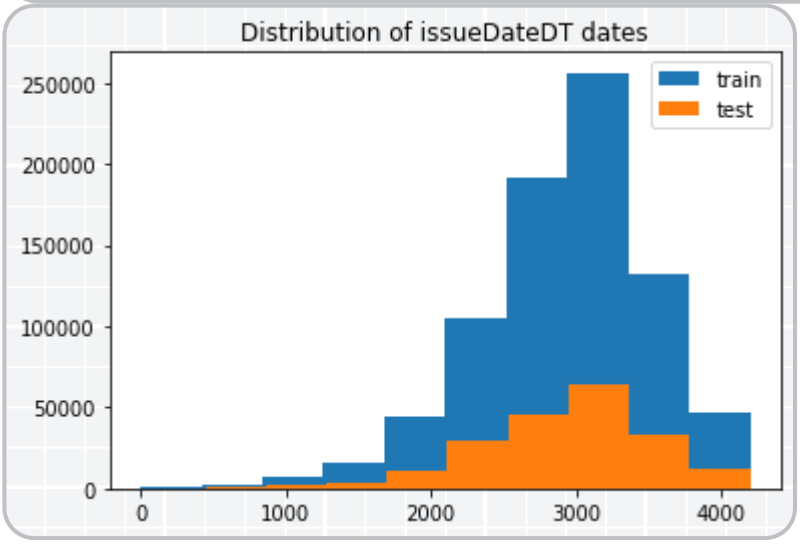


2.3.7 时间格式数据处理及查看

```
#转化成时间格式
data_train['issueDate'] = pd.to_datetime(data_train['issueDate'],format='%Y-%m-%d')
startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
data_train['issueDateDT'] = data_train['issueDate'].apply(lambda x: x-startdate).dt.days
```

```
#转化成时间格式
data_test_a['issueDate'] = pd.to_datetime(data_train['issueDate'],format='%Y-%m-%d')
startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
data_test_a['issueDateDT'] = data_test_a['issueDate'].apply(lambda x: x-startdate).dt.days
```

```
plt.hist(data_train['issueDateDT'], label='train');
plt.hist(data_test_a['issueDateDT'], label='test');
plt.legend();
plt.title('Distribution of issueDateDT dates');
#train 和 test issueDateDT 日期有重叠 所以使用基于时间的分割进行验证是不明智的
```



2.3.8 掌握透视图可以让我们更好的了解数据

```
#透视图 索引可以有多个，“columns（列）”是可选的，聚合函数aggfunc最后是被应用到了变量“values”中你所列举的项目上。
pivot = pd.pivot_table(data_train, index=['grade'], columns=['issueDateDT'], values=['loanAmnt'], aggfunc=np.sum)
```

```
pivot
```

	loanAmnt
issueDateDT	0
grade	
A	NaN
B	NaN
C	NaN
D	NaN
E	7500.0
F	NaN
G	NaN

2.3.9 用pandas_profiling生成数据报告

```
import pandas_profiling
```

```
pfr = pandas_profiling.ProfileReport(data_train)
pfr.to_file("./example.html")
```

2.4 总结

数据探索性分析是我们初步了解数据，熟悉数据为特征工程做准备的阶段，甚至很多时候EDA阶段提取出来的特征可以直接当作规则来用。可见EDA的重要性，这个阶段的主要工作还是借助于各个简单的统计量来对数据整体的了解，分析各个类型变量相互之间的关系，以及用合适的图形可视化出来直观观察。希望本节内容能给初学者带来帮助，更期待各位学习者对其中的不足提出建议。

END.

【 言溪：Datawhale成员，金融风控爱好者。知乎地址：https://www.zhihu.com/people/exuding】

关于Datawhale：

Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

3 Task3 特征工程

此部分为零基础入门金融风控的 Task3 特征工程部分，带你来了解各种特征工程以及分析方法，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 零基础入门金融风控之贷款违约

预测地址：

3.1 学习目标

1. 学习特征预处理、缺失值、异常值处理、数据分桶等特征处理方法
2. 学习特征交互、编码、选择的相应方法
3. 完成相应学习打卡任务，两个选做的作业不做强制性要求，供学有余力同学自己探索

3.2 内容介绍

1. 数据预处理：
    a. 缺失值的填充
    b. 时间格式处理
    c. 对象类型特征转换到数值
2. 异常值处理：
    a. 基于3segama原则
    b. 基于箱型图
3. 数据分箱
    a. 固定宽度分箱
    b. 分位数分箱
        - 离散数值型数据分箱
        - 连续数值型数据分箱

c. 卡方分箱（选做作业）

4. 特征交互

a. 特征和特征之间组合

b. 特征和特征之间衍生

c. 其他特征衍生的尝试（选做作业）

5. 特征编码

a. one-hot编码

b. label-encode编码

6. 特征选择

a. 1 Filter

b. 2 Wrapper （RFE）

c. 3 Embedded

3.3 代码示例

3.3.1 导入包并读取数据

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
import warnings
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, log_loss
warnings.filterwarnings('ignore')
```

```
data_train =pd.read_csv('../train.csv')
data_test_a = pd.read_csv('../testA.csv')
```

3.3.2 特征预处理

1. 数据EDA部分我们已经对数据的大概和某些特征分布有了了解，数据预处理部分一般我们要处理一些EDA阶段分析出来的问题，这里介绍了数据缺失值的填充，时间格式特征的转化处理，某些对象类别特征的处理。

首先我们查找出数据中的对象特征和数值特征

```
numerical_fea = list(data_train.select_dtypes(exclude=['object']).columns)
category_fea = list(filter(lambda x: x not in numerical_fea,list(data_train.columns)))
label = 'isDefault'
numerical_fea.remove(label)
```

在比赛中数据预处理是必不可少的一部分，对于缺失值的填充往往会影响比赛的结果，在比赛中不妨尝试多种填充然后比较结果选择结果最优的一种；

比赛数据相比真实场景的数据相对要“干净”一些，但是还是会有一定的“脏”数据存在，清洗一些异常值往往会获得意想不到的效果。

3.3.2.1 缺失值填充

1. 把所有缺失值替换为指定的值0

   data_train = data_train.fillna(0)

1. 向用缺失值上面的值替换缺失值

   data_train = data_train.fillna(axis=0,method='ffill')

1. 纵向用缺失值下面的值替换缺失值,且设置最多只填充两个连续的缺失值

   data_train = data_train.fillna(axis=0,method='bfill',limit=2)

```
#查看缺失值情况
data_train.isnull().sum()
```

```
id                     0
loanAmnt               0
term                   0
interestRate           0
installment            0
grade                  0
subGrade               0
employmentTitle        1
employmentLength   46799
homeOwnership          0
annualIncome           0
verificationStatus     0
issueDate              0
```

```
isDefault                0
purpose                  0
postCode                 1
regionCode               0
dti                    239
delinquency_2years       0
ficoRangeLow             0
ficoRangeHigh            0
openAcc                  0
pubRec                   0
pubRecBankruptcies     405
revolBal                 0
revolUtil              531
totalAcc                 0
initialListStatus        0
applicationType          0
earliesCreditLine        0
title                    1
policyCode               0
n0                   40270
n1                   40270
n2                   40270
n2.1                 40270
n4                   33239
n5                   40270
n6                   40270
n7                   40270
n8                   40271
n9                   40270
n10                  33239
n11                  69752
n12                  40270
n13                  40270
n14                  40270
dtype: int64
```

```
#按照平均数填充数值型特征
data_train[numerical_fea] =
data_train[numerical_fea].fillna(data_train[numerical_fea].median())
data_test_a[numerical_fea] =
data_test_a[numerical_fea].fillna(data_train[numerical_fea].median())
#按照众数填充类别型特征
data_train[category_fea] = data_train[category_fea].fillna(data_train[category_fea].mode())
data_test_a[category_fea] =
data_test_a[category_fea].fillna(data_train[category_fea].mode())
```

```
data_train.isnull().sum()
```

```
id                     0
loanAmnt               0
term                   0
interestRate           0
installment            0
grade                  0
subGrade               0
employmentTitle        0
employmentLength   46799
homeOwnership          0
annualIncome           0
verificationStatus     0
issueDate              0
isDefault              0
purpose                0
postCode               0
regionCode             0
dti                    0
delinquency_2years     0
ficoRangeLow           0
ficoRangeHigh          0
openAcc                0
pubRec                 0
pubRecBankruptcies     0
revolBal               0
revolUtil              0
totalAcc               0
initialListStatus      0
applicationType        0
earliesCreditLine      0
title                  0
policyCode             0
n0                     0
```

```
n1                    0
n2                    0
n2.1                  0
n4                    0
n5                    0
n6                    0
n7                    0
n8                    0
n9                    0
n10                   0
n11                   0
n12                   0
n13                   0
n14                   0
dtype: int64
```

```
#查看类别特征
category_fea
```

```
['grade', 'subGrade', 'employmentLength', 'issueDate', 'earliesCreditLine']
```

1. category_fea：对象型类别特征需要进行预处理，其中['issueDate']为时间格式特征。

3.3.2.2 时间格式处理

```
#转化成时间格式
for data in [data_train, data_test_a]:
    data['issueDate'] = pd.to_datetime(data['issueDate'],format='%Y-%m-%d')
    startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
    #构造时间特征
    data['issueDateDT'] = data['issueDate'].apply(lambda x: x-startdate).dt.days
```

```
data_train['employmentLength'].value_counts(dropna=False).sort_index()
```

```
1 year        52489
10+ years    262753
2 years       72358
3 years       64152
4 years       47985
5 years       50102
6 years       37254
7 years       35407
8 years       36192
9 years       30272
< 1 year      64237
NaN           46799
Name: employmentLength, dtype: int64
```

3.3.2.3 对象类型特征转换到数值

```
def employmentLength_to_int(s):
    if pd.isnull(s):
        return s
    else:
        return np.int8(s.split()[0])
for data in [data_train, data_test_a]:
    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)
```

```
data['employmentLength'].value_counts(dropna=False).sort_index()
```

```
0.0     15989
1.0     13182
2.0     18207
3.0     16011
4.0     11833
5.0     12543
6.0      9328
7.0      8823
8.0      8976
9.0      7594
10.0    65772
NaN     11742
Name: employmentLength, dtype: int64
```

1. 对earliesCreditLine进行预处理

```
data_train['earliesCreditLine'].sample(5)
```

```
519915    Sep-2002
564368    Dec-1996
768209    May-2004
453092    Nov-1995
763866    Sep-2000
Name: earliesCreditLine, dtype: object
```

```
for data in [data_train, data_test_a]:
    data['earliesCreditLine'] = data['earliesCreditLine'].apply(lambda s: int(s[-4:]))
```

3.3.2.4 类别特征处理

```
# 部分类别特征
cate_features = ['grade', 'subGrade', 'employmentTitle', 'homeOwnership',
'verificationStatus', 'purpose', 'postCode', 'regionCode', \
                 'applicationType', 'initialListStatus', 'title', 'policyCode']
for f in cate_features:
    print(f, '类型数：', data[f].nunique())
```

```
grade 类型数： 7
subGrade 类型数： 35
employmentTitle 类型数： 79282
homeOwnership 类型数： 6
verificationStatus 类型数： 3
purpose 类型数： 14
postCode 类型数： 889
regionCode 类型数： 51
applicationType 类型数： 2
initialListStatus 类型数： 2
title 类型数： 12058
policyCode 类型数： 1
```

像等级这种类别特征，是有优先级的可以labelencode或者自映射

```
for data in [data_train, data_test_a]:
    data['grade'] = data['grade'].map({'A':1,'B':2,'C':3,'D':4,'E':5,'F':6,'G':7})
```

```
# 类型数在2之上，又不是高维稀疏的,且纯分类特征
for data in [data_train, data_test_a]:
    data = pd.get_dummies(data, columns=['subGrade', 'homeOwnership', 'verificationStatus', 'purpose', 'regionCode'], drop_first=True)
```

3.3.3 异常值处理

1. 当你发现异常值后，一定要先分清是什么原因导致的异常值，然后再考虑如何处理。首先，如果这一异常值并不代表一种规律性的，而是极其偶然的现象，或者说你并不想研究这种偶然的现象，这时可以将其删除。其次，如果异常值存在且代表了一种真实存在的现象，那就不能随便删除。在现有的欺诈场景中很多时候欺诈数据本身相对于正常数据勒说就是异常的，我们要把这些异常点纳入，重新拟合模型，研究其规律。能用监督的用监督模型，不能用的还可以考虑用异常检测的算法来做。
2. 注意test的数据不能删。

3.3.3.1 检测异常的方法一：均方差

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
```

1. 得到特征的异常值后可以进一步分析变量异常值和目标变量的关系

```
data_train = data_train.copy()
for fea in numerical_fea:
    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)
```

```
正常值    800000
Name: id_outliers, dtype: int64
id_outliers
正常值    159610
```

```
Name: isDefault, dtype: int64
**********
正常值    800000
Name: loanAmnt_outliers, dtype: int64
loanAmnt_outliers
正常值    159610
Name: isDefault, dtype: int64
**********
正常值    800000
Name: term_outliers, dtype: int64
term_outliers
正常值    159610
Name: isDefault, dtype: int64
**********
正常值    794259
异常值      5741
Name: interestRate_outliers, dtype: int64
interestRate_outliers
异常值      2916
正常值    156694
Name: isDefault, dtype: int64
**********
正常值    792046
异常值      7954
Name: installment_outliers, dtype: int64
installment_outliers
异常值      2152
正常值    157458
Name: isDefault, dtype: int64
**********
正常值    800000
Name: employmentTitle_outliers, dtype: int64
employmentTitle_outliers
正常值    159610
Name: isDefault, dtype: int64
**********
正常值    799701
异常值       299
Name: homeOwnership_outliers, dtype: int64
homeOwnership_outliers
异常值        62
正常值    159548
Name: isDefault, dtype: int64
**********
正常值    793973
```

```
异常值      6027
Name: annualIncome_outliers, dtype: int64
annualIncome_outliers
异常值       756
正常值    158854
Name: isDefault, dtype: int64
**********
正常值    800000
Name: verificationStatus_outliers, dtype: int64
verificationStatus_outliers
正常值    159610
Name: isDefault, dtype: int64
**********
正常值    783003
异常值     16997
Name: purpose_outliers, dtype: int64
purpose_outliers
异常值      3635
正常值    155975
Name: isDefault, dtype: int64
**********
正常值    798931
异常值      1069
Name: postCode_outliers, dtype: int64
postCode_outliers
异常值       221
正常值    159389
Name: isDefault, dtype: int64
**********
正常值    799994
异常值         6
Name: regionCode_outliers, dtype: int64
regionCode_outliers
异常值         1
正常值    159609
Name: isDefault, dtype: int64
**********
正常值    798440
异常值      1560
Name: dti_outliers, dtype: int64
dti_outliers
异常值       466
正常值    159144
Name: isDefault, dtype: int64
**********
```

```
正常值    778245
异常值     21755
Name: delinquency_2years_outliers, dtype: int64
delinquency_2years_outliers
异常值      5089
正常值    154521
Name: isDefault, dtype: int64
**********
正常值    788261
异常值     11739
Name: ficoRangeLow_outliers, dtype: int64
ficoRangeLow_outliers
异常值       778
正常值    158832
Name: isDefault, dtype: int64
**********
正常值    788261
异常值     11739
Name: ficoRangeHigh_outliers, dtype: int64
ficoRangeHigh_outliers
异常值       778
正常值    158832
Name: isDefault, dtype: int64
**********
正常值    790889
异常值      9111
Name: openAcc_outliers, dtype: int64
openAcc_outliers
异常值      2195
正常值    157415
Name: isDefault, dtype: int64
**********
正常值    792471
异常值      7529
Name: pubRec_outliers, dtype: int64
pubRec_outliers
异常值      1701
正常值    157909
Name: isDefault, dtype: int64
**********
正常值    794120
异常值      5880
Name: pubRecBankruptcies_outliers, dtype: int64
pubRecBankruptcies_outliers
异常值      1423
```

```
正常值    158187
Name: isDefault, dtype: int64
**********
正常值    790001
异常值      9999
Name: revolBal_outliers, dtype: int64
revolBal_outliers
异常值      1359
正常值    158251
Name: isDefault, dtype: int64
**********
正常值    799948
异常值        52
Name: revolUtil_outliers, dtype: int64
revolUtil_outliers
异常值        23
正常值    159587
Name: isDefault, dtype: int64
**********
正常值    791663
异常值      8337
Name: totalAcc_outliers, dtype: int64
totalAcc_outliers
异常值      1668
正常值    157942
Name: isDefault, dtype: int64
**********
正常值    800000
Name: initialListStatus_outliers, dtype: int64
initialListStatus_outliers
正常值    159610
Name: isDefault, dtype: int64
**********
正常值    784586
异常值     15414
Name: applicationType_outliers, dtype: int64
applicationType_outliers
异常值      3875
正常值    155735
Name: isDefault, dtype: int64
**********
正常值    775134
异常值     24866
Name: title_outliers, dtype: int64
title_outliers
```

```
异常值      3900
正常值    155710
Name: isDefault, dtype: int64
**********
正常值    800000
Name: policyCode_outliers, dtype: int64
policyCode_outliers
正常值    159610
Name: isDefault, dtype: int64
**********
正常值    782773
异常值     17227
Name: n0_outliers, dtype: int64
n0_outliers
异常值      3485
正常值    156125
Name: isDefault, dtype: int64
**********
正常值    790500
异常值      9500
Name: n1_outliers, dtype: int64
n1_outliers
异常值      2491
正常值    157119
Name: isDefault, dtype: int64
**********
正常值    789067
异常值     10933
Name: n2_outliers, dtype: int64
n2_outliers
异常值      3205
正常值    156405
Name: isDefault, dtype: int64
**********
正常值    789067
异常值     10933
Name: n2.1_outliers, dtype: int64
n2.1_outliers
异常值      3205
正常值    156405
Name: isDefault, dtype: int64
**********
正常值    788660
异常值     11340
Name: n4_outliers, dtype: int64
```

```
n4_outliers
异常值      2476
正常值    157134
Name: isDefault, dtype: int64
**********
正常值    790355
异常值      9645
Name: n5_outliers, dtype: int64
n5_outliers
异常值      1858
正常值    157752
Name: isDefault, dtype: int64
**********
正常值    786006
异常值     13994
Name: n6_outliers, dtype: int64
n6_outliers
异常值      3182
正常值    156428
Name: isDefault, dtype: int64
**********
正常值    788430
异常值     11570
Name: n7_outliers, dtype: int64
n7_outliers
异常值      2746
正常值    156864
Name: isDefault, dtype: int64
**********
正常值    789625
异常值     10375
Name: n8_outliers, dtype: int64
n8_outliers
异常值      2131
正常值    157479
Name: isDefault, dtype: int64
**********
正常值    786384
异常值     13616
Name: n9_outliers, dtype: int64
n9_outliers
异常值      3953
正常值    155657
Name: isDefault, dtype: int64
**********
```

```
正常值    788979
异常值     11021
Name: n10_outliers, dtype: int64
n10_outliers
异常值      2639
正常值    156971
Name: isDefault, dtype: int64
**********
正常值    799434
异常值       566
Name: n11_outliers, dtype: int64
n11_outliers
异常值       112
正常值    159498
Name: isDefault, dtype: int64
**********
正常值    797585
异常值      2415
Name: n12_outliers, dtype: int64
n12_outliers
异常值       545
正常值    159065
Name: isDefault, dtype: int64
**********
正常值    788907
异常值     11093
Name: n13_outliers, dtype: int64
n13_outliers
异常值      2482
正常值    157128
Name: isDefault, dtype: int64
**********
正常值    788884
异常值     11116
Name: n14_outliers, dtype: int64
n14_outliers
异常值      3364
正常值    156246
Name: isDefault, dtype: int64
**********
```

1. 例如可以看到异常值在两个变量上的分布几乎复合整体的分布，如果异常值都属于为1的用户数据里面代表什么呢？

```
#删除异常值
for fea in numerical_fea:
    data_train = data_train[data_train[fea+'_outliers']=='正常值']
    data_train = data_train.reset_index(drop=True)
```

3.3.3.2 检测异常的方法二：箱型图

1. 总结一句话：四分位数会将数据分为三个点和四个区间，IQR = Q3 -Q1，下触须=Q1 − 1.5x IQR，上触须=Q3 + 1.5x IQR；

3.3.4 数据分桶

1. 特征分箱的目的：
    a. 从模型效果上来看，特征分箱主要是为了降低变量的复杂性，减少变量噪音对模型的影响，提高自变量和因变量的相关度。从而使模型更加稳定。
2. 数据分桶的对象：
    a. 将连续变量离散化
    b. 将多状态的离散变量合并成少状态
3. 分箱的原因：
    a. 数据的特征内的值跨度可能比较大，对有监督和无监督中如k-均值聚类它使用欧氏距离作为相似度函数来测量数据点之间的相似度。都会造成大吃小的影响，其中一种解决方法是对计数值进行区间量化即数据分桶也叫做数据分箱，然后使用量化后的结果。
4. 分箱的优点：
    a. 处理缺失值：当数据源可能存在缺失值，此时可以把null单独作为一个分箱。
    b. 处理异常值：当数据中存在离群点时，可以把其通过分箱离散化处理，从而提高变量的鲁棒性（抗干扰能力）。例如，age若出现200这种异常值，可分入“age > 60”这个分箱里，排除影响。
    c. 业务解释性：我们习惯于线性判断变量的作用，当x越来越大，y就越来越大。但实际x与y之间经常存在着非线性关系，此时可经过WOE变换。
5. 特别要注意一下分箱的基本原则：
    a. （1）最小分箱占比不低于5%
    b. （2）箱内不能全部是好客户
    c. （3）连续箱单调

1. 固定宽度分箱

当数值横跨多个数量级时，最好按照 10 的幂（或任何常数的幂）来进行分组：0_9、$_{10}99$、100_{999}、$_{1000}9999$，等等。固定宽度分箱非常容易计算，但如果计数值中有比较大的缺口，就会产生很多没有任何数据的空箱子。

```
# 通过除法映射到间隔均匀的分箱中，每个分箱的取值范围都是loanAmnt/1000
data['loanAmnt_bin1'] = np.floor_divide(data['loanAmnt'], 1000)
```

```
## 通过对数函数映射到指数宽度分箱
data['loanAmnt_bin2'] = np.floor(np.log10(data['loanAmnt']))
```

2. 分位数分箱

```
data['loanAmnt_bin3'] = pd.qcut(data['loanAmnt'], 10, labels=False)
```

3. 卡方分箱及其他分箱方法的尝试

1. 这一部分属于进阶部分，学有余力的同学可以自行搜索尝试。

3.3.5 特征交互

1. 交互特征的构造非常简单，使用起来却代价不菲。如果线性模型中包含有交互特征对，那它的训练时间和评分时间就会从 O(n) 增加到 O(n2)，其中 n 是单一特征的数量。

```
for col in ['grade', 'subGrade']:
    temp_dict = data_train.groupby([col])['isDefault'].agg(['mean']).reset_index().rename(columns={'mean': col + '_target_mean'})
    temp_dict.index = temp_dict[col].values
    temp_dict = temp_dict[col + '_target_mean'].to_dict()

    data_train[col + '_target_mean'] = data_train[col].map(temp_dict)
    data_test_a[col + '_target_mean'] = data_test_a[col].map(temp_dict)
```

```
# 其他衍生变量 mean 和 std
for df in [data_train, data_test_a]:
    for item in ['n0','n1','n2','n2.1','n4','n5','n6','n7','n8','n9','n10','n11','n12','n13','n14']:
        df['grade_to_mean_' + item] = df['grade'] / df.groupby([item])['grade'].transform('mean')
        df['grade_to_std_' + item] = df['grade'] / df.groupby([item])['grade'].transform('std')
```

这里给出一些特征交互的思路，但特征和特征间的交互衍生出新的特征还远远不止于此，抛砖引玉，希望大家多多探索。请学习者尝试其他的特征交互方法

3.3.6 特征编码

3.3.6.1 labelEncode 直接放入树模型中

```
#label-encode:subGrade,postCode,title
# 高维类别特征需要进行转换
for col in tqdm(['employmentTitle', 'postCode', 'title','subGrade']):
    le = LabelEncoder()
    le.fit(list(data_train[col].astype(str).values) +
list(data_test_a[col].astype(str).values))
    data_train[col] = le.transform(list(data_train[col].astype(str).values))
    data_test_a[col] = le.transform(list(data_test_a[col].astype(str).values))
print('Label Encoding 完成')
```

100%|██████████| 4/4 [00:08<00:00, 2.04s/it]

Label Encoding 完成

3.3.6.2 逻辑回归等模型要单独增加的特征工程

1. 对特征做归一化，去除相关性高的特征
2. 归一化目的是让训练过程更好更快的收敛，避免特征大吃小的问题
3. 去除相关性是增加模型的可解释性，加快预测过程。

```
# 举例归一化过程
#伪代码
for fea in [要归一化的特征列表]:
    data[fea] = ((data[fea] - np.min(data[fea])) / (np.max(data[fea]) - np.min(data[fea])))
```

3.3.7 特征选择

1. 特征选择技术可以精简掉无用的特征，以降低最终模型的复杂性，它的最终目的是得到一个简约模型，在不降低预测准确率或对预测准确率影响不大的情况下提高计算速度。特征选择不是为了减少训练时间（实际上，一些技术会增加总体训练时间），而是为了减少模型评分时间。

特征选择的方法：

1. 1 Filter
    a. 方差选择法
    b. 相关系数法（pearson 相关系数）
    c. 卡方检验
    d. 互信息法
2. 2 Wrapper （RFE）
    a. 递归特征消除法

3. 3 Embedded

   a. 基于惩罚项的特征选择法

   b. 基于树模型的特征选择

3.3.7.1 Filter

1. 基于特征间的关系进行筛选

方差选择法

1. 方差选择法中，先要计算各个特征的方差，然后根据设定的阈值，选择方差大于阈值的特征

```
from sklearn.feature_selection import VarianceThreshold
#其中参数threshold为方差的阈值
VarianceThreshold(threshold=3).fit_transform(train,target_train)
```

相关系数法

1. Pearson 相关系数

   皮尔森相关系数是一种最简单的，可以帮助理解特征和响应变量之间关系的方法，该方法衡量的是变量之间的线性相关性。

   结果的取值区间为 [-1，1] ， -1 表示完全的负相关， +1表示完全的正相关，0 表示没有线性相关。

```
from sklearn.feature_selection import SelectKBest
from scipy.stats import pearsonr
#选择K个最好的特征，返回选择特征后的数据
#第一个参数为计算评估特征是否好的函数，该函数输入特征矩阵和目标向量，
#输出二元组（评分，P值）的数组，数组第i项为第i个特征的评分和P值。在此定义为计算相关系数
#参数k为选择的特征个数

SelectKBest(k=5).fit_transform(train,target_train)
```

卡方检验

1. 经典的卡方检验是用于检验自变量对因变量的相关性。 假设自变量有N种取值，因变量有M种取值，考虑自变量等于i且因变量等于j的样本频数的观察值与期望的差距。 其统计量如下： $\chi 2=\sum(A-T)2T$，其中A为实际值，T为理论值
2. (注：卡方只能运用在正定矩阵上，否则会报错Input X must be non-negative)

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
#参数k为选择的特征个数

SelectKBest(chi2, k=5).fit_transform(train,target_train)
```

互信息法

1. 经典的互信息也是评价自变量对因变量的相关性的。 在feature_selection库的SelectKBest类结合最大信息系数法可以用于选择特征，相关代码如下：

```
from sklearn.feature_selection import SelectKBest
from minepy import MINE
#由于MINE的设计不是函数式的，定义mic方法将其为函数式的，
#返回一个二元组，二元组的第2项设置成固定的P值0.5
def mic(x, y):
    m = MINE()
    m.compute_score(x, y)
    return (m.mic(), 0.5)
#参数k为选择的特征个数
SelectKBest(lambda X, Y: array(map(lambda x:mic(x, Y), X.T)).T,
k=2).fit_transform(train,target_train)
```

3.3.7.2 Wrapper （Recursive feature elimination，RFE）

1. 递归特征消除法 递归消除特征法使用一个基模型来进行多轮训练，每轮训练后，消除若干权值系数的特征，再基于新的特征集进行下一轮训练。 在feature_selection库的RFE类可以用于选择特征，相关代码如下（以逻辑回归为例）：

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
#递归特征消除法，返回特征选择后的数据
#参数estimator为基模型
#参数n_features_to_select为选择的特征个数

RFE(estimator=LogisticRegression(),
n_features_to_select=2).fit_transform(train,target_train)
```

3.3.7.3 Embedded

1. 基于惩罚项的特征选择法 使用带惩罚项的基模型，除了筛选出特征外，同时也进行了降维。 在feature_selection库的SelectFromModel类结合逻辑回归模型可以用于选择特征，相关代码如下：

```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
#带L1惩罚项的逻辑回归作为基模型的特征选择

SelectFromModel(LogisticRegression(penalty="l1", C=0.1)).fit_transform(train,target_train)
```

1. 基于树模型的特征选择 树模型中GBDT也可用来作为基模型进行特征选择。 在feature_selection库的SelectFromModel类结合GBDT模型可以用于选择特征，相关代码如下：

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import GradientBoostingClassifier
#GBDT作为基模型的特征选择
SelectFromModel(GradientBoostingClassifier()).fit_transform(train,target_train)
```

3.3.7.4 数据处理

本数据集中我们删除非入模特征后，并对缺失值填充，然后用计算协方差的方式看一下特征间相关性，然后进行模型训练

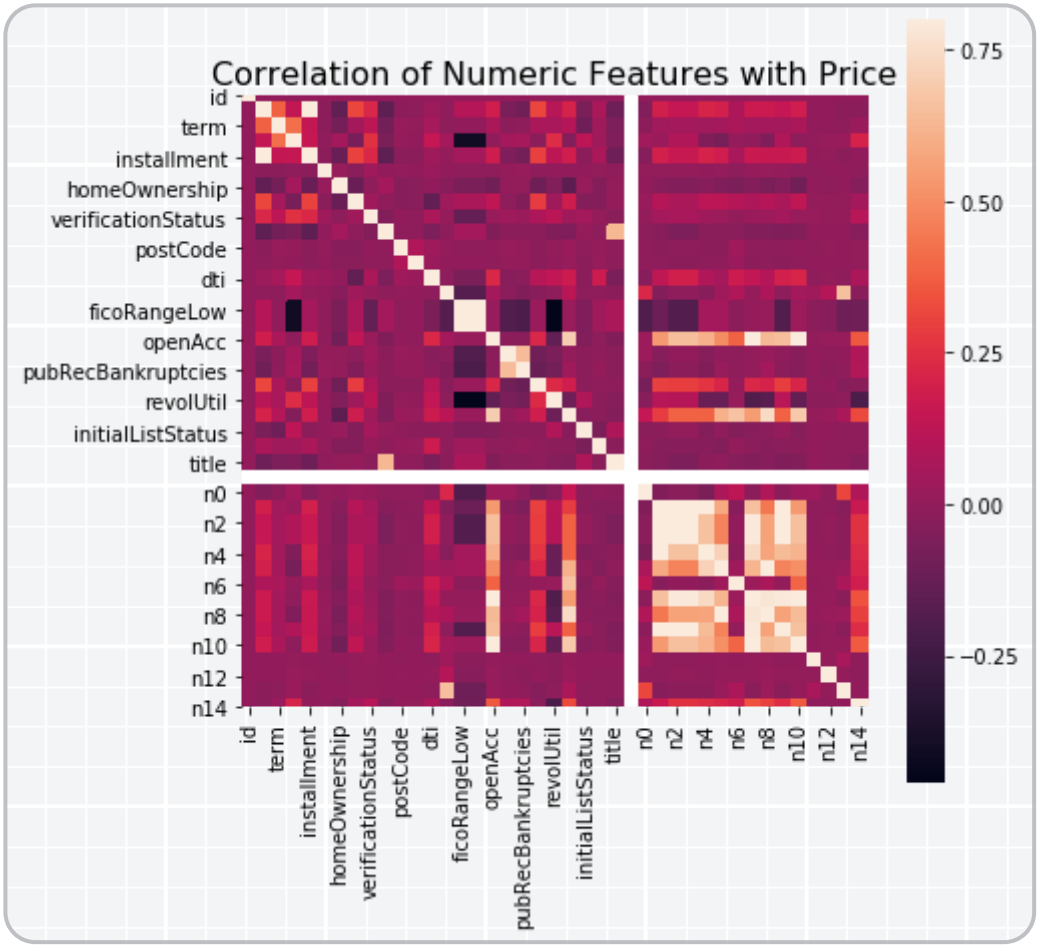
```
# 删除不需要的数据
for data in [data_train, data_test_a]:
    data.drop(['issueDate','id'], axis=1,inplace=True)
```

```
"纵向用缺失值上面的值替换缺失值"
data_train = data_train.fillna(axis=0,method='ffill')
```

```
x_train = data_train.drop(['isDefault','id'], axis=1)
#计算协方差
data_corr = x_train.corrwith(data_train.isDefault) #计算相关性
result = pd.DataFrame(columns=['features', 'corr'])
result['features'] = data_corr.index
result['corr'] = data_corr.values
```

```
# 当然也可以直接看图
data_numeric = data_train[numerical_fea]
correlation = data_numeric.corr()

f , ax = plt.subplots(figsize = (7, 7))
plt.title('Correlation of Numeric Features with Price',y=1,size=16)
sns.heatmap(correlation,square = True,  vmax=0.8)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12d88ad10>
```




```
features = [f for f in data_train.columns if f not in ['id','issueDate','isDefault'] and '_outliers' not in f]
x_train = data_train[features]
x_test = data_test_a[features]
y_train = data_train['isDefault']
```

```
def cv_model(clf, train_x, train_y, test_x, clf_name):
    folds = 5
    seed = 2020
    kf = KFold(n_splits=folds, shuffle=True, random_state=seed)

    train = np.zeros(train_x.shape[0])
    test = np.zeros(test_x.shape[0])

    cv_scores = []

    for i, (train_index, valid_index) in enumerate(kf.split(train_x, train_y)):
        print('************************************ {} ************************************'.format(str(i+1)))
        trn_x, trn_y, val_x, val_y = train_x.iloc[train_index], train_y[train_index], train_x.iloc[valid_index], train_y[valid_index]

        if clf_name == "lgb":
```

```
            train_matrix = clf.Dataset(trn_x, label=trn_y)
            valid_matrix = clf.Dataset(val_x, label=val_y)

            params = {
                'boosting_type': 'gbdt',
                'objective': 'binary',
                'metric': 'auc',
                'min_child_weight': 5,
                'num_leaves': 2 ** 5,
                'lambda_l2': 10,
                'feature_fraction': 0.8,
                'bagging_fraction': 0.8,
                'bagging_freq': 4,
                'learning_rate': 0.1,
                'seed': 2020,
                'nthread': 28,
                'n_jobs':24,
                'silent': True,
                'verbose': -1,
            }

            model = clf.train(params, train_matrix, 50000, valid_sets=[train_matrix, valid_matrix], verbose_eval=200,early_stopping_rounds=200)
            val_pred = model.predict(val_x, num_iteration=model.best_iteration)
            test_pred = model.predict(test_x, num_iteration=model.best_iteration)

            # print(list(sorted(zip(features, model.feature_importance("gain")), key=lambda x: x[1], reverse=True))[:20])

        if clf_name == "xgb":
            train_matrix = clf.DMatrix(trn_x , label=trn_y)
            valid_matrix = clf.DMatrix(val_x , label=val_y)

            params = {'booster': 'gbtree',
                      'objective': 'binary:logistic',
                      'eval_metric': 'auc',
                      'gamma': 1,
                      'min_child_weight': 1.5,
                      'max_depth': 5,
                      'lambda': 10,
                      'subsample': 0.7,
                      'colsample_bytree': 0.7,
                      'colsample_bylevel': 0.7,
                      'eta': 0.04,
                      'tree_method': 'exact',
```

```
                    'seed': 2020,
                    'nthread': 36,
                    "silent": True,
                    }

            watchlist = [(train_matrix, 'train'),(valid_matrix, 'eval')]

            model = clf.train(params, train_matrix, num_boost_round=50000, evals=watchlist,
verbose_eval=200, early_stopping_rounds=200)
            val_pred  = model.predict(valid_matrix, ntree_limit=model.best_ntree_limit)
            test_pred = model.predict(test_x , ntree_limit=model.best_ntree_limit)

        if clf_name == "cat":
            params = {'learning_rate': 0.05, 'depth': 5, 'l2_leaf_reg': 10,
'bootstrap_type': 'Bernoulli',
                    'od_type': 'Iter', 'od_wait': 50, 'random_seed': 11,
'allow_writing_files': False}

            model = clf(iterations=20000, **params)
            model.fit(trn_x, trn_y, eval_set=(val_x, val_y),
                      cat_features=[], use_best_model=True, verbose=500)

            val_pred  = model.predict(val_x)
            test_pred = model.predict(test_x)

        train[valid_index] = val_pred
        test = test_pred / kf.n_splits
        cv_scores.append(roc_auc_score(val_y, val_pred))

        print(cv_scores)

    print("%s_scotrainre_list:" % clf_name, cv_scores)
    print("%s_score_mean:" % clf_name, np.mean(cv_scores))
    print("%s_score_std:" % clf_name, np.std(cv_scores))
    return train, test
```

```
def lgb_model(x_train, y_train, x_test):
    lgb_train, lgb_test = cv_model(lgb, x_train, y_train, x_test, "lgb")
    return lgb_train, lgb_test

def xgb_model(x_train, y_train, x_test):
    xgb_train, xgb_test = cv_model(xgb, x_train, y_train, x_test, "xgb")
    return xgb_train, xgb_test

def cat_model(x_train, y_train, x_test):
    cat_train, cat_test = cv_model(CatBoostRegressor, x_train, y_train, x_test, "cat")
```

```
lgb_train, lgb_test = lgb_model(x_train, y_train, x_test)
```

```
********************************** 1 **********************************
Training until validation scores don't improve for 200 rounds
[200]   training's auc: 0.749225    valid_1's auc: 0.729679
[400]   training's auc: 0.765075    valid_1's auc: 0.730496
[600]   training's auc: 0.778745    valid_1's auc: 0.730435
Early stopping, best iteration is:
[455]   training's auc: 0.769202    valid_1's auc: 0.730686
[0.7306859913754798]
********************************** 2 **********************************
Training until validation scores don't improve for 200 rounds
[200]   training's auc: 0.749221    valid_1's auc: 0.731315
[400]   training's auc: 0.765117    valid_1's auc: 0.731658
[600]   training's auc: 0.778542    valid_1's auc: 0.731333
Early stopping, best iteration is:
[407]   training's auc: 0.765671    valid_1's auc: 0.73173
[0.7306859913754798, 0.7317304414673989]
********************************** 3 **********************************
Training until validation scores don't improve for 200 rounds
[200]   training's auc: 0.748436    valid_1's auc: 0.732775
[400]   training's auc: 0.764216    valid_1's auc: 0.733173
Early stopping, best iteration is:
[386]   training's auc: 0.763261    valid_1's auc: 0.733261
[0.7306859913754798, 0.7317304414673989, 0.7332610441015461]
********************************** 4 **********************************
Training until validation scores don't improve for 200 rounds
[200]   training's auc: 0.749631    valid_1's auc: 0.728327
[400]   training's auc: 0.765139    valid_1's auc: 0.728845
Early stopping, best iteration is:
[286]   training's auc: 0.756978    valid_1's auc: 0.728976
[0.7306859913754798, 0.7317304414673989, 0.7332610441015461, 0.7289759386807912]
********************************** 5 **********************************
Training until validation scores don't improve for 200 rounds
[200]   training's auc: 0.748414    valid_1's auc: 0.732727
[400]   training's auc: 0.763727    valid_1's auc: 0.733531
[600]   training's auc: 0.777489    valid_1's auc: 0.733566
Early stopping, best iteration is:
[524]   training's auc: 0.772372    valid_1's auc: 0.733772
[0.7306859913754798, 0.7317304414673989, 0.7332610441015461, 0.7289759386807912,
0.7337723979789789]
lgb_scotrainre_list: [0.7306859913754798, 0.7317304414673989, 0.7332610441015461,
0.7289759386807912, 0.7337723979789789]
lgb_score_mean: 0.7316851627208389
lgb_score_std: 0.0017424259863954693
```

```
testA_result = pd.read_csv('../testA_result.csv')
```

```
roc_auc_score(testA_result['isDefault'].values, lgb_test)
```

```
0.7290917729487896
```

3.4 总结

特征工程是机器学习，甚至是深度学习中最为重要的一部分，在实际应用中往往也是所花费时间最多的一步。各种算法书中对特征工程部分的讲解往往少得可怜，因为特征工程和具体的数据结合的太紧密，很难系统地覆盖所有场景。本章主要是通过一些常用的方法来做介绍，例如缺失值异常值的处理方法详细对任何数据集来说都是适用的。但对于分箱等操作本章给出了具体的几种思路，需要读者自己探索。在特征工程中比赛和具体的应用还是有所不同的，在实际的金融风控评分卡制作过程中，由于强调特征的可解释性，特征分箱尤其重要。学有余力同学可以自行多尝试，希望大家在本节学习中有所收获。

END.

【 言溪：Datawhale成员，金融风控爱好者。知乎地址：https://www.zhihu.com/people/exuding 】

关于Datawhale： Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。 本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

4 Task4 建模与调参

此部分为零基础入门金融风控的 Task4 建模调参部分，带你来了解各种模型以及模型的评价和调参策略，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 零基础入门金融风控之贷款违约预测

地址：

4.1 学习目标

1. 学习在金融分控领域常用的机器学习模型
2. 学习机器学习模型的建模过程与调参流程
3. 完成相应学习打卡任务

4.2 内容介绍

1. 逻辑回归模型：
    a. 理解逻辑回归模型；
    b. 逻辑回归模型的应用；
    c. 逻辑回归的优缺点；
2. 树模型：
    a. 理解树模型；
    b. 树模型的应用；
    c. 树模型的优缺点；
3. 集成模型

a. 基于bagging思想的集成模型
    - 随机森林模型
b. 基于boosting思想的集成模型
    - XGBoost模型
    - LightGBM模型
    - CatBoost模型

4. 模型对比与性能评估：
    a. 回归模型/树模型/集成模型；
    b. 模型评估方法；
    c. 模型评价结果；
5. 模型调参：
    a. 贪心调参方法；
    b. 网格调参方法；
    c. 贝叶斯调参方法；

4.3 模型相关原理介绍

由于相关算法原理篇幅较长，本文推荐了一些博客与教材供初学者们进行学习。

4.3.1 逻辑回归模型

https://blog.csdn.net/han_xiaoyang/article/details/49123419

4.3.2 决策树模型

https://blog.csdn.net/c406495762/article/details/76262487

4.3.3 GBDT模型

https://zhuanlan.zhihu.com/p/45145899

4.3.4 XGBoost模型

https://blog.csdn.net/wuzhongqiang/article/details/104854890

4.3.5 LightGBM模型

https://blog.csdn.net/wuzhongqiang/article/details/105350579

4.3.6 Catboost模型

https://mp.weixin.qq.com/s/xloTLr5NJBgBspMQtxPoFA

4.3.7 时间序列模型(选学)

RNN：https://zhuanlan.zhihu.com/p/45289691

LSTM：https://zhuanlan.zhihu.com/p/83496936

4.3.8 推荐教材：

《机器学习》 https://book.douban.com/subject/26708119/

《统计学习方法》 https://book.douban.com/subject/10590856/

《面向机器学习的特征工程》 https://book.douban.com/subject/26826639/

《信用评分模型技术与应用》https://book.douban.com/subject/1488075/

《数据化风控》https://book.douban.com/subject/30282558/

4.4 模型对比与性能评估

4.4.1 逻辑回归

1. 优点
    a. 训练速度较快，分类的时候，计算量仅仅只和特征的数目相关；
    b. 简单易理解，模型的可解释性非常好，从特征的权重可以看到不同的特征对最后结果的影响；
    c. 适合二分类问题，不需要缩放输入特征；
    d. 内存资源占用小，只需要存储各个维度的特征值；
2. 缺点
    a. **逻辑回归需要预先处理缺失值和异常值【可参考task3特征工程】；**
    b. 不能用Logistic回归去解决非线性问题，因为Logistic的决策面是线性的；
    c. 对多重共线性数据较为敏感，且很难处理数据不平衡的问题；
    d. 准确率并不是很高，因为形式非常简单，很难去拟合数据的真实分布；

4.4.2 决策树模型

1. 优点
    a. 简单直观，生成的决策树可以可视化展示
    b. **数据不需要预处理，不需要归一化，不需要处理缺失数据**
    c. 既可以处理离散值，也可以处理连续值
2. 缺点
    a. 决策树算法非常容易过拟合，导致泛化能力不强（可进行适当的剪枝）
    b. 采用的是贪心算法，容易得到局部最优解

4.4.3 集成模型集成方法（ensemble method）

通过组合多个学习器来完成学习任务，通过集成方法，可以将多个弱学习器组合成一个强分类器，因此集成学习的泛化能力一般比单一分类器要好。

集成方法主要包括Bagging和Boosting，Bagging和Boosting都是将已有的分类或回归算法通过一定方式组合起来，形成一个更加强大的分类。两种方法都是把若干个分类器整合为一个分类器的方法，只是整合的方式不一样，最终得到不一样的效果。常见的基于Baggin思想的集成模型有：随机森林、基于Boosting思想的集成模型有：Adaboost、GBDT、XgBoost、LightGBM等。

Baggin和Boosting的区别总结如下：

1. **样本选择上：** Bagging方法的训练集是从原始集中有放回的选取，所以从原始集中选出的各轮训练集之间是独立的；而Boosting方法需要每一轮的训练集不变，只是训练集中每个样本在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整
2. **样例权重上：** Bagging方法使用均匀取样，所以每个样本的权重相等；而Boosting方法根据错误率不断调整样本的权值，错误率越大则权重越大
3. **预测函数上：** Bagging方法中所有预测函数的权重相等；而Boosting方法中每个弱分类器都有相应的权重，对于分类误差小的分类器会有更大的权重
4. **并行计算上：** Bagging方法中各个预测函数可以并行生成；而Boosting方法各个预测函数只能顺序生成，因为后一个模型参数需要前一轮模型的结果。

4.4.4 模型评估方法

对于模型来说，其在训练集上面的误差我们称之为**训练误差**或者**经验误差**，而在测试集上的误差称之为**测试误差**。

对于我们来说，我们更关心的是模型对于新样本的学习能力，即我们希望通过对已有样本的学习，尽可能的将所有潜在样本的普遍规律学到手，而如果模型对训练样本学的太好，则有可能把训练样本自身所具有的一些特点当做所有潜在样本的普遍特点，这时候我们就会出现**过拟合**的问题。

因此我们通常将已有的数据集划分为训练集和测试集两部分，其中训练集用来训练模型，而测试集则是用来评估模型对于新样本的判别能力。

对于数据集的划分，我们通常要保证满足以下两个条件：

1. 训练集和测试集的分布要与样本真实分布一致，即训练集和测试集都要保证是从样本真实分布中独立同分布采样而得；
2. 训练集和测试集要互斥

对于数据集的划分有三种方法：留出法，交叉验证法和自助法，下面挨个介绍：

1. **①留出法**

   留出法是直接将数据集D划分为两个互斥的集合，其中一个集合作为训练集S，另一个作为测试集T。需要注意的是在划分的时候要尽可能保证数据分布的一致性，即避免因数据划分过程引入额外的偏差而对最终结果产生影响。为了保证数据分布的一致性，通常我们采用**分层采样**的方式来对数据进行采样。

   Tips： 通常，会将数据集D中大约2/3~4/5的样本作为训练集，其余的作为测试集。
2. **②交叉验证法**

k折交叉验证通常将数据集D分为k份，其中k-1份作为训练集，剩余的一份作为测试集，这样就可以获得k组训练/测试集，可以进行k次训练与测试，最终返回的是k个测试结果的均值。交叉验证中数据集的划分依然是依据**分层采样**的方式来进行。

对于交叉验证法，其k值的选取往往决定了评估结果的稳定性和保真性，**通常k值选取10。**

当k=1的时候，我们称之为**留一法**

3. **③自助法**

   我们每次从数据集D中取一个样本作为训练集中的元素，然后把该样本放回，重复该行为m次，这样我们就可以得到大小为m的训练集，在这里面有的样本重复出现，有的样本则没有出现过，我们把那些没有出现过的样本作为测试集。

   进行这样采样的原因是因为在D中约有36.8%的数据没有在训练集中出现过。留出法与交叉验证法都是使用**分层采样**的方式进行数据采样与划分，而自助法则是使用**有放回重复采样**的方式进行数据采样

数据集划分总结

1. 对于数据量充足的时候，通常采用**留出法**或者**k折交叉验证法**来进行训练/测试集的划分；
2. 对于数据集小且难以有效划分训练/测试集时使用**自助法**；
3. 对于数据集小且可有效划分的时候最好使用**留一法**来进行划分，因为这种方法最为准确

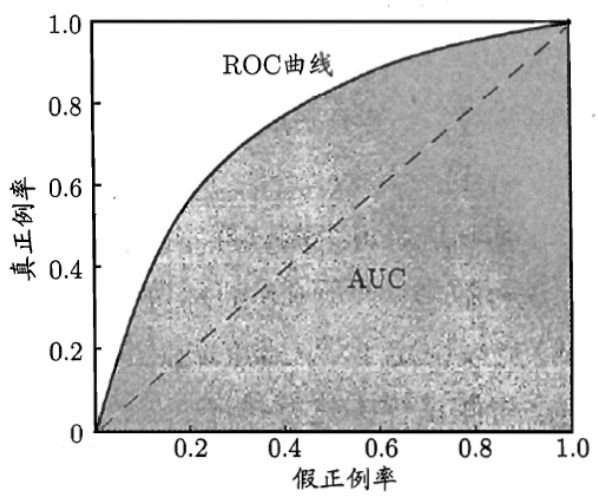
4.4.5 模型评价标准

对于本次比赛，我们选用auc作为模型评价标准，类似的评价标准还有ks、f1-score等，具体介绍与实现大家可以回顾下task1中的内容。

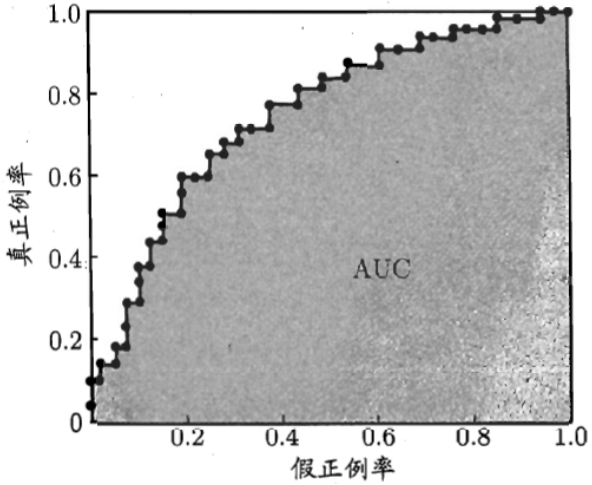
一起来看一下auc到底是什么？

在逻辑回归里面，对于正负例的界定，通常会设一个阈值，大于阈值的为正类，小于阈值为负类。如果我们减小这个阀值，更多的样本会被识别为正类，提高正类的识别率，但同时也会使得更多的负类被错误识别为正类。为了直观表示这一现象，引入ROC。

根据分类结果计算得到ROC空间中相应的点，连接这些点就形成ROC curve，横坐标为False Positive Rate(FPR：假正率)，纵坐标为True Positive Rate(TPR：真正率)。 一般情况下，这个曲线都应该处于(0,0)和(1,1)连线的上方,如图：



(a) ROC 曲线与 AUC



(b) 基于有限样例绘制的 ROC 曲线与 AUC

ROC曲线中的四个点：

1. 点(0,1)：即FPR=0, TPR=1，意味着FN＝0且FP＝0，将所有的样本都正确分类；
2. 点(1,0)：即FPR=1，TPR=0，最差分类器，避开了所有正确答案；
3. 点(0,0)：即FPR=TPR=0，FP＝TP＝0，分类器把每个实例都预测为负类；
4. 点(1,1)：分类器把每个实例都预测为正类

总之：ROC曲线越接近左上角，该分类器的性能越好，其泛化性能就越好。而且一般来说，如果ROC是光滑的，那么基本可以判断没有太大的overfitting。

但是对于两个模型，我们如何判断哪个模型的泛化性能更优呢？这里我们有主要以下两种方法：

如果模型A的ROC曲线完全包住了模型B的ROC曲线，那么我们就认为模型A要优于模型B；

如果两条曲线有交叉的话，我们就通过比较ROC与X，Y轴所围得曲线的面积来判断，面积越大，模型的性能就越优，**这个面积我们称之为AUC(area under ROC curve)**

4.5 代码示例

4.5.1 导入相关关和相关设置

```
import pandas as pd
import numpy as np
import warnings
import os
import seaborn as sns
```

```
import matplotlib.pyplot as plt
"""
sns 相关设置
@return:
"""
# 声明使用 Seaborn 样式
sns.set()
# 有五种seaborn的绘图风格，它们分别是：darkgrid, whitegrid, dark, white, ticks。默认的主题是darkgrid。
sns.set_style("whitegrid")
# 有四个预置的环境，按大小从小到大排列分别为：paper, notebook, talk, poster。其中，notebook是默认的。
sns.set_context('talk')
# 中文字体设置-黑体
plt.rcParams['font.sans-serif'] = ['SimHei']
# 解决保存图像是负号'-'显示为方块的问题
plt.rcParams['axes.unicode_minus'] = False
# 解决Seaborn中文显示问题并调整字体大小
sns.set(font='SimHei')
```

4.5.2 读取数据

reduce_mem_usage 函数通过调整数据类型，帮助我们减少数据在内存中占用的空间

```
def reduce_mem_usage(df):
    start_mem = df.memory_usage().sum()
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
```

```
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum()
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df
```

```
# 读取数据
data = pd.read_csv('dataset/data_for_model.csv')
data = reduce_mem_usage(data)
```

```
Memory usage of dataframe is 928000128.00 MB
Memory usage after optimization is: 165006456.00 MB
Decreased by 82.2%
```

4.5.3 简单建模

Tips1：金融风控的实际项目多涉及到信用评分，因此需要模型特征具有较好的可解释性，所以目前在实际项目中多还是以逻辑回归作为基础模型。但是在比赛中以得分高低为准，不需要严谨的可解释性，所以大多基于集成算法进行建模。

Tips2：因为逻辑回归的算法特性，需要提前对异常值、缺失值数据进行处理【参考task3部分】

Tips3：基于树模型的算法特性，异常值、缺失值处理可以跳过，但是对于业务较为了解的同学也可以自己对缺失异常值进行处理，效果可能会更优于模型处理的结果。

注：以下建模的源数据参考baseline进行了相应的特征工程，对于异常缺失值未进行相应的处理操作。

建模之前的预操作

```
from sklearn.model_selection import KFold
# 分离数据集，方便进行交叉验证
X_train = data.loc[data['sample']=='train', :].drop(['id','issueDate','isDefault',
'sample'], axis=1)
X_test = data.loc[data['sample']=='test', :].drop(['id','issueDate','isDefault', 'sample'],
axis=1)
y_train = data.loc[data['sample']=='train', 'isDefault']

# 5折交叉验证
folds = 5
seed = 2020
kf = KFold(n_splits=folds, shuffle=True, random_state=seed)
```

使用Lightgbm进行建模

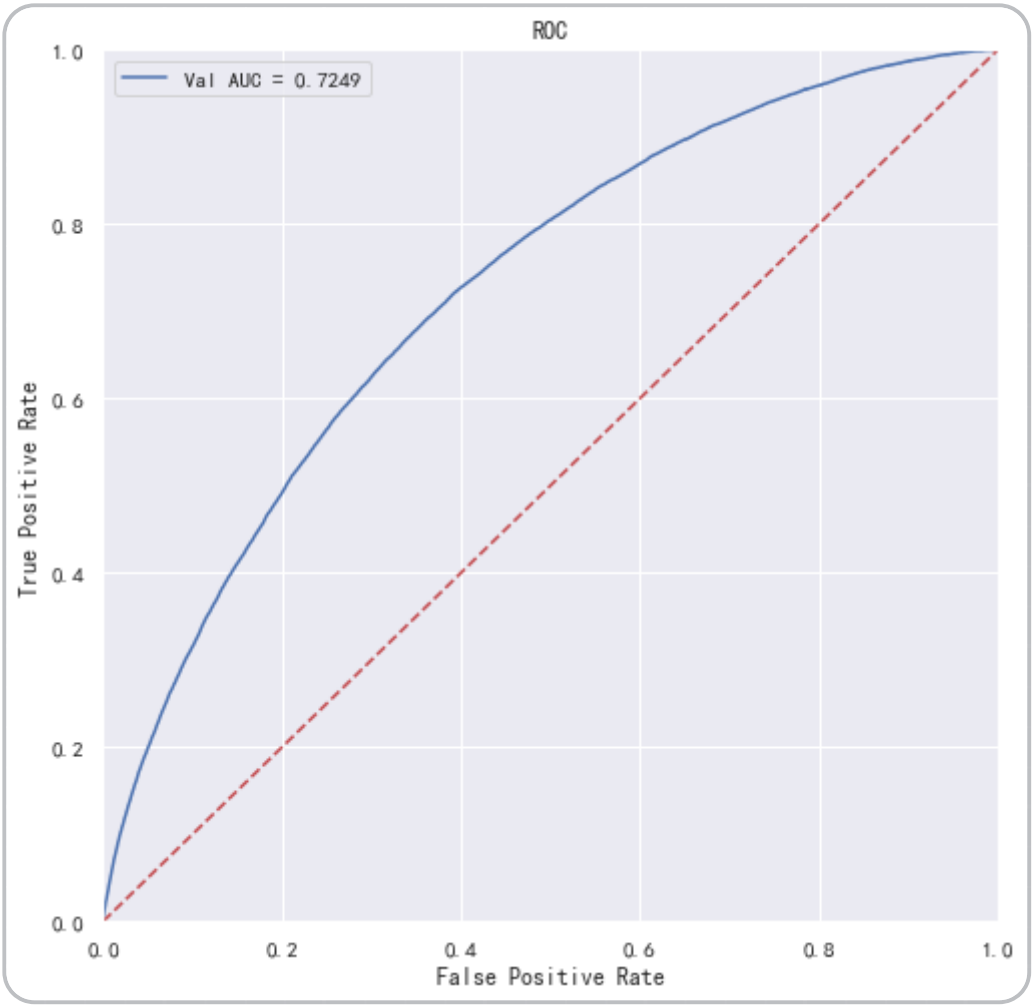
```
"""对训练集数据进行划分，分成训练集和验证集，并进行相应的操作"""
from sklearn.model_selection import train_test_split
import lightgbm as lgb
# 数据集划分
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_train,
test_size=0.2)
train_matrix = lgb.Dataset(X_train_split, label=y_train_split)
valid_matrix = lgb.Dataset(X_val, 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,
            'reg_lambda': 0,
            'reg_alpha': 0,
            'feature_fraction': 1,
            'bagging_fraction': 1,
            'bagging_freq': 0,
            'seed': 2020,
            'nthread': 8,
            'silent': True,
            'verbose': -1,
}

"""使用训练集数据进行模型训练"""
model = lgb.train(params, train_set=train_matrix, valid_sets=valid_matrix,
num_boost_round=20000, verbose_eval=1000, early_stopping_rounds=200)
```

```
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[427]   valid_0's auc: 0.724947
```

对验证集进行预测

```
from sklearn import metrics
from sklearn.metrics import roc_auc_score

"""预测并计算roc的相关指标"""
val_pre_lgb = model.predict(X_val, num_iteration=model.best_iteration)
fpr, tpr, threshold = metrics.roc_curve(y_val, val_pre_lgb)
roc_auc = metrics.auc(fpr, tpr)
print('未调参前lightgbm单模型在验证集上的AUC：{}'.format(roc_auc))
"""画出roc曲线图"""
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()
```

```
未调参前lightgbm单模型在验证集上的AUC：0.7249469360631181
```



更进一步的，使用5折交叉验证进行模型性能评估

```
import lightgbm as lgb
"""使用lightgbm 5折交叉验证进行建模预测"""
cv_scores = []
for i, (train_index, valid_index) in enumerate(kf.split(X_train, y_train)):
    print('********************************** {}
**********************************'.format(str(i+1)))
    X_train_split, y_train_split, X_val, y_val = X_train.iloc[train_index],
y_train[train_index], X_train.iloc[valid_index], y_train[valid_index]

    train_matrix = lgb.Dataset(X_train_split, label=y_train_split)
    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,
```

```
                'reg_lambda': 0,
                'reg_alpha': 0,
                'feature_fraction': 1,
                'bagging_fraction': 1,
                'bagging_freq': 0,
                'seed': 2020,
                'nthread': 8,
                'silent': True,
                'verbose': -1,
    }

    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)))
```

```
...
lgb_scotrainre_list:[0.7303837315833632, 0.7258692125145638, 0.7305149209921737,
0.7296117869375041, 0.7294438695369077]
lgb_score_mean:0.7291647043129024
lgb_score_std:0.0016998349834934656
```

4.5.4 模型调参

1. **1. 贪心调参**

先使用当前对模型影响最大的参数进行调优，达到当前参数下的模型最优化，再使用对模型影响次之的参数进行调优，如此下去，直到所有的参数调整完毕。

这个方法的缺点就是可能会调到局部最优而不是全局最优，但是只需要一步一步的进行参数最优化调试即可，容易理解。

需要注意的是在树模型中参数调整的顺序，也就是各个参数对模型的影响程度，这里列举一下日常调参过程中常用的参数和调参顺序：

1. ①：max_depth、num_leaves
2. ②：min_data_in_leaf、min_child_weight
3. ③：bagging_fraction、 feature_fraction、bagging_freq

4. ④：reg_lambda、reg_alpha
5. ⑤：min_split_gain

```
from sklearn.model_selection import cross_val_score

# 调objective
best_obj = dict()
for obj in objective:
    model = LGBMRegressor(objective=obj)
    """预测并计算roc的相关指标"""
    score = cross_val_score(model, X_train, y_train, cv=5, scoring='roc_auc').mean()
      best_obj[obj] = score

# num_leaves
best_leaves = dict()
for leaves in num_leaves:
    model = LGBMRegressor(objective=min(best_obj.items(), key=lambda x:x[1])[0],
num_leaves=leaves)
    """预测并计算roc的相关指标"""
    score = cross_val_score(model, X_train, y_train, cv=5, scoring='roc_auc').mean()
    best_leaves[leaves] = score

# max_depth
best_depth = dict()
for depth in max_depth:
    model = LGBMRegressor(objective=min(best_obj.items(), key=lambda x:x[1])[0],
                          num_leaves=min(best_leaves.items(), key=lambda x:x[1])[0],
                          max_depth=depth)
    """预测并计算roc的相关指标"""
    score = cross_val_score(model, X_train, y_train, cv=5, scoring='roc_auc').mean()
    best_depth[depth] = score

"""
可依次将模型的参数通过上面的方式进行调整优化，并且通过可视化观察在每一个最优参数下模型的得分情况
"""
```

可依次将模型的参数通过上面的方式进行调整优化，并且通过可视化观察在每一个最优参数下模型的得分情况

1. **2. 网格搜索**

sklearn 提供GridSearchCV用于进行网格搜索，只需要把模型的参数输进去，就能给出最优化的结果和参数。相比起贪心调参，网格搜索的结果会更优，但是网格搜索只适合于小数据集，一旦数据的量级上去了，很难得出结果。

同样以Lightgbm算法为例，进行网格搜索调参：

```
"""通过网格搜索确定最优参数"""
```

```
from sklearn.model_selection import GridSearchCV

def get_best_cv_params(learning_rate=0.1, n_estimators=581, num_leaves=31, max_depth=-1, bagging_fraction=1.0,
                       feature_fraction=1.0, bagging_freq=0, min_data_in_leaf=20, min_child_weight=0.001,
                       min_split_gain=0, reg_lambda=0, reg_alpha=0, param_grid=None):
    # 设置5折交叉验证
    cv_fold = StratifiedKFold(n_splits=5, random_state=0, shuffle=True, )

    model_lgb = lgb.LGBMClassifier(learning_rate=learning_rate,
                                   n_estimators=n_estimators,
                                   num_leaves=num_leaves,
                                   max_depth=max_depth,
                                   bagging_fraction=bagging_fraction,
                                   feature_fraction=feature_fraction,
                                   bagging_freq=bagging_freq,
                                   min_data_in_leaf=min_data_in_leaf,
                                   min_child_weight=min_child_weight,
                                   min_split_gain=min_split_gain,
                                   reg_lambda=reg_lambda,
                                   reg_alpha=reg_alpha,
                                   n_jobs= 8
                                  )
    grid_search = GridSearchCV(estimator=model_lgb,
                               cv=cv_fold,
                               param_grid=param_grid,
                               scoring='roc_auc'
                              )
    grid_search.fit(X_train, y_train)

    print('模型当前最优参数为:{}'.format(grid_search.best_params_))
    print('模型当前最优得分为:{}'.format(grid_search.best_score_))
```

```
"""以下代码未运行，耗时较长，请谨慎运行，且每一步的最优参数需要在下一步进行手动更新，请注意"""

"""
需要注意一下的是，除了获取上面的获取num_boost_round时候用的是原生的lightgbm（因为要用自带的cv）
下面配合GridSearchCV时必须使用sklearn接口的lightgbm。
"""
"""设置n_estimators 为581，调整num_leaves和max_depth，这里选择先粗调再细调"""
lgb_params = {'num_leaves': range(10, 80, 5), 'max_depth': range(3,10,2)}
get_best_cv_params(learning_rate=0.1, n_estimators=581, num_leaves=None, max_depth=None, min_data_in_leaf=20,
                   min_child_weight=0.001,bagging_fraction=1.0, feature_fraction=1.0, bagging_freq=0,
```

```
                    min_split_gain=0, reg_lambda=0, reg_alpha=0, param_grid=lgb_params)

"""num_leaves为30，max_depth为7，进一步细调num_leaves和max_depth"""
lgb_params = {'num_leaves': range(25, 35, 1), 'max_depth': range(5,9,1)}
get_best_cv_params(learning_rate=0.1, n_estimators=85, num_leaves=None, max_depth=None,
min_data_in_leaf=20,
                    min_child_weight=0.001,bagging_fraction=1.0, feature_fraction=1.0,
bagging_freq=0,
                    min_split_gain=0, reg_lambda=0, reg_alpha=0, param_grid=lgb_params)

"""
确定min_data_in_leaf为45，min_child_weight为0.001 ，下面进行bagging_fraction、
feature_fraction和bagging_freq的调参
"""
lgb_params = {'bagging_fraction': [i/10 for i in range(5,10,1)],
              'feature_fraction': [i/10 for i in range(5,10,1)],
              'bagging_freq': range(0,81,10)
             }
get_best_cv_params(learning_rate=0.1, n_estimators=85, num_leaves=29, max_depth=7,
min_data_in_leaf=45,
                    min_child_weight=0.001,bagging_fraction=None, feature_fraction=None,
bagging_freq=None,
                    min_split_gain=0, reg_lambda=0, reg_alpha=0, param_grid=lgb_params)

"""
确定bagging_fraction为0.4、feature_fraction为0.6、bagging_freq为 ，下面进行reg_lambda、
reg_alpha的调参
"""
lgb_params = {'reg_lambda': [0,0.001,0.01,0.03,0.08,0.3,0.5], 'reg_alpha':
[0,0.001,0.01,0.03,0.08,0.3,0.5]}
get_best_cv_params(learning_rate=0.1, n_estimators=85, num_leaves=29, max_depth=7,
min_data_in_leaf=45,
                    min_child_weight=0.001,bagging_fraction=0.9, feature_fraction=0.9,
bagging_freq=40,
                    min_split_gain=0, reg_lambda=None, reg_alpha=None,
param_grid=lgb_params)

"""
确定reg_lambda、reg_alpha都为0，下面进行min_split_gain的调参
"""
lgb_params = {'min_split_gain': [i/10 for i in range(0,11,1)]}
get_best_cv_params(learning_rate=0.1, n_estimators=85, num_leaves=29, max_depth=7,
min_data_in_leaf=45,
                    min_child_weight=0.001,bagging_fraction=0.9, feature_fraction=0.9,
bagging_freq=40,
```

```
                    min_split_gain=None, reg_lambda=0, reg_alpha=0, param_grid=lgb_params)
```

```
"""
参数确定好了以后，我们设置一个比较小的learning_rate 0.005，来确定最终的num_boost_round
"""
# 设置5折交叉验证
# cv_fold = StratifiedKFold(n_splits=5, random_state=0, shuffle=True, )
final_params = {
                 'boosting_type': 'gbdt',
                 'learning_rate': 0.01,
                 'num_leaves': 29,
                 'max_depth': 7,
                 'min_data_in_leaf':45,
                 'min_child_weight':0.001,
                 'bagging_fraction': 0.9,
                 'feature_fraction': 0.9,
                 'bagging_freq': 40,
                 'min_split_gain': 0,
                 'reg_lambda':0,
                 'reg_alpha':0,
                 'nthread': 6
                }

cv_result = lgb.cv(train_set=lgb_train,
                   early_stopping_rounds=20,
                   num_boost_round=5000,
                   nfold=5,
                   stratified=True,
                   shuffle=True,
                   params=final_params,
                   metrics='auc',
                   seed=0,
                  )

print('迭代次数{}'.format(len(cv_result['auc-mean'])))
print('交叉验证的AUC为{}'.format(max(cv_result['auc-mean'])))
```

在实际调整过程中，可先设置一个较大的学习率（上面的例子中0.1），通过Lgb原生的cv函数进行树个数的确定，之后再通过上面的实例代码进行参数的调整优化。

最后针对最优的参数设置一个较小的学习率（例如0.05），同样通过cv函数确定树的个数，确定最终的参数。

需要注意的是，针对大数据集，上面每一层参数的调整都需要耗费较长时间，

1. **贝叶斯调参**

在使用之前需要先安装包bayesian-optimization，运行如下命令即可：

```
pip install bayesian-optimization
```

贝叶斯调参的主要思想是：给定优化的目标函数(广义的函数，只需指定输入和输出即可，无需知道内部结构以及数学性质)，通过不断地添加样本点来更新目标函数的后验分布(高斯过程,直到后验分布基本贴合于真实分布）。简单的说，就是考虑了上一次参数的信息，从而更好的调整当前的参数。

贝叶斯调参的步骤如下：

1. 定义优化函数(rf_cv）
2. 建立模型
3. 定义待优化的参数
4. 得到优化结果，并返回要优化的分数指标

```
from sklearn.model_selection import cross_val_score

"""定义优化函数"""
def rf_cv_lgb(num_leaves, max_depth, bagging_fraction, feature_fraction, bagging_freq, min_data_in_leaf,
              min_child_weight, min_split_gain, reg_lambda, reg_alpha):
    # 建立模型
    model_lgb = lgb.LGBMClassifier(boosting_type='gbdt', bjective='binary', metric='auc',
                                   learning_rate=0.1, n_estimators=5000,
                                   num_leaves=int(num_leaves), max_depth=int(max_depth),
                                   bagging_fraction=round(bagging_fraction, 2), feature_fraction=round(feature_fraction, 2),
                                   bagging_freq=int(bagging_freq), min_data_in_leaf=int(min_data_in_leaf),
                                   min_child_weight=min_child_weight, min_split_gain=min_split_gain,
                                   reg_lambda=reg_lambda, reg_alpha=reg_alpha,
                                   n_jobs= 8
                                  )

    val = cross_val_score(model_lgb, X_train_split, y_train_split, cv=5, scoring='roc_auc').mean()

    return val
```

```
from bayes_opt import BayesianOptimization
"""定义优化参数"""
bayes_lgb = BayesianOptimization(
    rf_cv_lgb,
    {
        'num_leaves':(10, 200),
        'max_depth':(3, 20),
        'bagging_fraction':(0.5, 1.0),
        'feature_fraction':(0.5, 1.0),
```

```
        'bagging_freq':(0, 100),
        'min_data_in_leaf':(10,100),
        'min_child_weight':(0, 10),
        'min_split_gain':(0.0, 1.0),
        'reg_alpha':(0.0, 10),
        'reg_lambda':(0.0, 10),
    }
)

"""开始优化"""
bayes_lgb.maximize(n_iter=10)
```

```
  |   iter    |  target   | baggin... | baggin... | featur... | max_depth | min_ch... | min_da... | min_sp... | num_le... | reg_alpha | reg_la... |
  -------------------------------------------------------------------------------------------------------------------------------------
  | •[0m 1       •[0m | •[0m 0.7263  •[0m | •[0m 0.7196  •[0m | •[0m 80.73   •[0m | •[0m 0.7988  •[0m | •[0m 19.17   •[0m | •[0m 5.751   •[0m | •[0m 40.71   •[0m | •[0m 0.9548  •[0m | •[0m 176.2   •[0m | •[0m 2.939   •[0m | •[0m 7.212   •[0m |
  | •[95m 2       •[0m | •[95m 0.7279  •[0m | •[95m 0.8997  •[0m | •[95m 74.72   •[0m | •[95m 0.5904  •[0m | •[95m 7.259   •[0m | •[95m 6.175   •[0m | •[95m 92.03   •[0m | •[95m 0.4027  •[0m | •[95m 51.65   •[0m | •[95m 6.404   •[0m | •[95m 4.781   •[0m |
  | •[0m 3       •[0m | •[0m 0.7207  •[0m | •[0m 0.5133  •[0m | •[0m 16.53   •[0m | •[0m 0.9536  •[0m | •[0m 4.974   •[0m | •[0m 2.37    •[0m | •[0m 98.08   •[0m | •[0m 0.7909  •[0m | •[0m 52.12   •[0m | •[0m 4.443   •[0m | •[0m 4.429   •[0m |
  | •[0m 4       •[0m | •[0m 0.7276  •[0m | •[0m 0.6265  •[0m | •[0m 53.12   •[0m | •[0m 0.7307  •[0m | •[0m 10.67   •[0m | •[0m 1.824   •[0m | •[0m 18.98   •[0m | •[0m 0.954   •[0m | •[0m 60.47   •[0m | •[0m 6.963   •[0m | •[0m 1.999   •[0m |
  | •[0m 5       •[0m | •[0m 0.6963  •[0m | •[0m 0.6509  •[0m | •[0m 11.58   •[0m | •[0m 0.5386  •[0m | •[0m 11.21   •[0m | •[0m 7.85    •[0m | •[0m 11.4    •[0m | •[0m 0.4269  •[0m | •[0m 153.0   •[0m | •[0m 0.5227  •[0m | •[0m 2.257   •[0m |
  | •[0m 6       •[0m | •[0m 0.7276  •[0m | •[0m 0.6241  •[0m | •[0m 49.76   •[0m | •[0m 0.6057  •[0m | •[0m 10.34   •[0m | •[0m 1.718   •[0m | •[0m 22.43   •[0m | •[0m 0.8294  •[0m | •[0m 55.68   •[0m | •[0m 6.759   •[0m | •[0m 2.6     •[0m |
  | •[95m 7       •[0m | •[95m 0.7283  •[0m | •[95m 0.9815  •[0m | •[95m 96.15   •[0m | •[95m 0.6961  •[0m | •[95m 19.45   •[0m | •[95m 1.627   •[0m | •[95m 37.7    •[0m | •[95m 0.4185  •[0m | •[95m 14.22   •[0m | •[95m 7.057   •[0m | •[95m 9.924   •[0m |
  | •[0m 8       •[0m | •[0m 0.7278  •[0m | •[0m 0.7139  •[0m | •[0m 96.83   •[0m | •[0m 0.5063  •[0m | •[0m 3.941   •[0m | •[0m 1.469   •[0m | •[0m 97.28   •[0m | •[0m 0.07553 •[0m | •[0m 196.9   •[0m | •[0m 7.988   •[0m | •[0m 2.159   •[0m |
  | •[0m 9       •[0m | •[0m 0.7195  •[0m | •[0m 0.5352  •[0m | •[0m 98.72   •[0m | •[0m 0.9699  •[0m | •[0m 4.445   •[0m | •[0m 1.767   •[0m | •[0m 13.91   •[0m | •[0m 0.1647  •[0m | •[0m 191.5   •[0m | •[0m 4.003   •[0m | •[0m 2.027   •[0m |
  | •[0m 10      •[0m | •[0m 0.7281  •[0m | •[0m 0.7281  •[0m | •[0m 73.63   •[0m | •[0m 0.5598  •[0m | •[0m 19.29   •[0m | •[0m 0.5344  •[0m | •[0m 99.66   •[0m | •[0m 0.933   •[0m | •[0m 101.4   •[0m | •[0m 8.836   •[0m | •[0m 0.9222  •[0m |
```

```
  | •[0m 11      •[0m | •[0m 0.7279  •[0m | •[0m 0.8213  •[0m | •[0m 0.05856 •[0m | •[0m 0.7626  •[0m | •[0m 17.49   •[0m | •[0m 8.447   •[0m | •[0m 10.71   •[0m | •[0m 0.3252 •[0m | •[0m 13.64   •[0m | •[0m 9.319   •[0m | •[0m 0.4747  •[0m |
  | •[0m 12      •[0m | •[0m 0.7281  •[0m | •[0m 0.8372  •[0m | •[0m 95.71   •[0m | •[0m 0.9598  •[0m | •[0m 10.32   •[0m | •[0m 8.394   •[0m | •[0m 15.23   •[0m | •[0m 0.4909 •[0m | •[0m 94.48   •[0m | •[0m 9.486   •[0m | •[0m 9.044   •[0m |
  | •[0m 13      •[0m | •[0m 0.6993  •[0m | •[0m 0.5183  •[0m | •[0m 99.02   •[0m | •[0m 0.542   •[0m | •[0m 15.5    •[0m | •[0m 8.35    •[0m | •[0m 38.15   •[0m | •[0m 0.4079 •[0m | •[0m 58.01   •[0m | •[0m 0.2668  •[0m | •[0m 1.652   •[0m |
  | •[0m 14      •[0m | •[0m 0.7267  •[0m | •[0m 0.7933  •[0m | •[0m 4.459   •[0m | •[0m 0.79    •[0m | •[0m 7.557   •[0m | •[0m 2.43    •[0m | •[0m 27.91   •[0m | •[0m 0.8725 •[0m | •[0m 28.32   •[0m | •[0m 9.967   •[0m | •[0m 9.885   •[0m |
  | •[0m 15      •[0m | •[0m 0.6979  •[0m | •[0m 0.9419  •[0m | •[0m 1.22    •[0m | •[0m 0.835   •[0m | •[0m 11.56   •[0m | •[0m 9.962   •[0m | •[0m 93.79   •[0m | •[0m 0.018  •[0m | •[0m 197.6   •[0m | •[0m 9.711   •[0m | •[0m 3.78    •[0m |
=========================================================================================================
=========================================================
```

```
"""显示优化结果"""
bayes_lgb.max
```

```
{'target': 0.7282530196283977,
 'params': {'bagging_fraction': 0.9815471914843896,
  'bagging_freq': 96.14757648686668,
  'feature_fraction': 0.6961281791730929,
  'max_depth': 19.45450235568963,
  'min_child_weight': 1.6266132496156782,
  'min_data_in_leaf': 37.697878831472295,
  'min_split_gain': 0.4184947943942168,
  'num_leaves': 14.221122487200399,
  'reg_alpha': 7.056502173310882,
  'reg_lambda': 9.924023764203156}}
```

参数优化完成后，我们可以根据优化后的参数建立新的模型，降低学习率并寻找最优模型迭代次数

```
"""调整一个较小的学习率，并通过cv函数确定当前最优的迭代次数"""
base_params_lgb = {
                    'boosting_type': 'gbdt',
                    'objective': 'binary',
                    'metric': 'auc',
                    'learning_rate': 0.01,
                    'num_leaves': 14,
                    'max_depth': 19,
                    'min_data_in_leaf': 37,
                    'min_child_weight':1.6,
                    'bagging_fraction': 0.98,
                    'feature_fraction': 0.69,
                    'bagging_freq': 96,
```

```
                    'reg_lambda': 9,
                    'reg_alpha': 7,
                    'min_split_gain': 0.4,
                    'nthread': 8,
                    'seed': 2020,
                    'silent': True,
                    'verbose': -1,
}

cv_result_lgb = lgb.cv(
    train_set=train_matrix,
    early_stopping_rounds=1000,
    num_boost_round=20000,
    nfold=5,
    stratified=True,
    shuffle=True,
    params=base_params_lgb,
    metrics='auc',
    seed=0
)

print('迭代次数{}'.format(len(cv_result_lgb['auc-mean'])))
print('最终模型的AUC为{}'.format(max(cv_result_lgb['auc-mean'])))
```

```
迭代次数14269
最终模型的AUC为0.7315032037635779
```

模型参数已经确定，建立最终模型并对验证集进行验证

```
import lightgbm as lgb
"""使用lightgbm 5折交叉验证进行建模预测"""
cv_scores = []
for i, (train_index, valid_index) in enumerate(kf.split(X_train, y_train)):
    print('********************************** {}
**********************************'.format(str(i+1)))
    X_train_split, y_train_split, X_val, y_val = X_train.iloc[train_index],
y_train[train_index], X_train.iloc[valid_index], y_train[valid_index]

    train_matrix = lgb.Dataset(X_train_split, label=y_train_split)
    valid_matrix = lgb.Dataset(X_val, label=y_val)

    params = {
                'boosting_type': 'gbdt',
                'objective': 'binary',
                'metric': 'auc',
                'learning_rate': 0.01,
                'num_leaves': 14,
```

```
                'max_depth': 19,
                'min_data_in_leaf': 37,
                'min_child_weight':1.6,
                'bagging_fraction': 0.98,
                'feature_fraction': 0.69,
                'bagging_freq': 96,
                'reg_lambda': 9,
                'reg_alpha': 7,
                'min_split_gain': 0.4,
                'nthread': 8,
                'seed': 2020,
                'silent': True,
    }

    model = lgb.train(params, train_set=train_matrix, num_boost_round=14269, 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)))
```

```
...
lgb_scotrainre_list:[0.7329726464187137, 0.7294292852806246, 0.7341505801564857, 0.7328331383185244, 0.7317405262608612]
lgb_score_mean:0.732225235287042
lgb_score_std:0.0015929470575114753
```

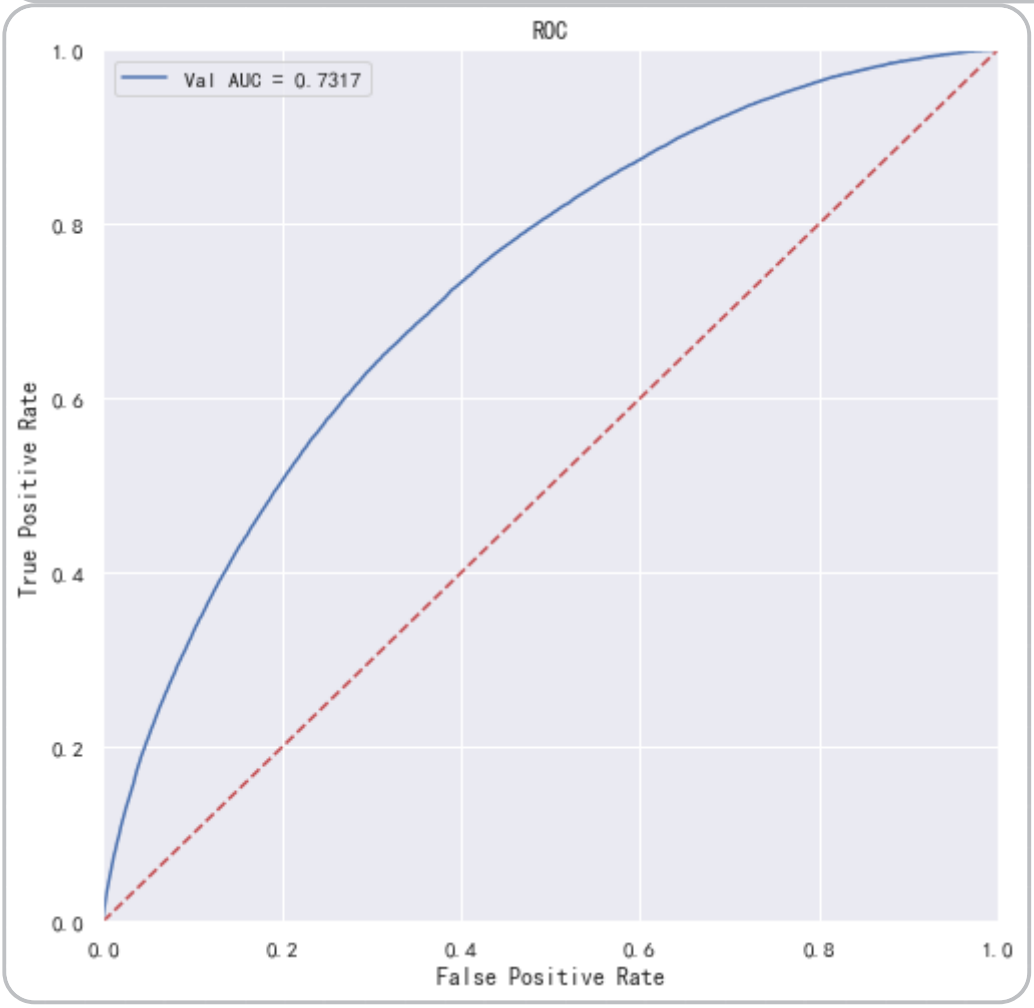
通过5折交叉验证可以发现，模型迭代次数在13000次的时候会停之，那么我们在建立新模型时直接设置最大迭代次数，并使用验证集进行模型预测

```
"""""""
base_params_lgb = {
                    'boosting_type': 'gbdt',
                    'objective': 'binary',
                    'metric': 'auc',
                    'learning_rate': 0.01,
                    'num_leaves': 14,
                    'max_depth': 19,
                    'min_data_in_leaf': 37,
                    'min_child_weight':1.6,
                    'bagging_fraction': 0.98,
                    'feature_fraction': 0.69,
                    'bagging_freq': 96,
```

```
                    'reg_lambda': 9,
                    'reg_alpha': 7,
                    'min_split_gain': 0.4,
                    'nthread': 8,
                    'seed': 2020,
                    'silent': True,
}

"""使用训练集数据进行模型训练"""
final_model_lgb = lgb.train(base_params_lgb, train_set=train_matrix,
valid_sets=valid_matrix, num_boost_round=13000, verbose_eval=1000,
early_stopping_rounds=200)

"""预测并计算roc的相关指标"""
val_pre_lgb = final_model_lgb.predict(X_val)
fpr, tpr, threshold = metrics.roc_curve(y_val, val_pre_lgb)
roc_auc = metrics.auc(fpr, tpr)
print('调参后lightgbm单模型在验证集上的AUC：{}'.format(roc_auc))
"""画出roc曲线图"""
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()
```

```
Training until validation scores don't improve for 200 rounds
[1000]  valid_0's auc: 0.723676
[2000]  valid_0's auc: 0.727282
[3000]  valid_0's auc: 0.728593
[4000]  valid_0's auc: 0.729493
[5000]  valid_0's auc: 0.730087
[6000]  valid_0's auc: 0.730515
[7000]  valid_0's auc: 0.730872
[8000]  valid_0's auc: 0.731121
[9000]  valid_0's auc: 0.731351
[10000] valid_0's auc: 0.731502
[11000] valid_0's auc: 0.731707
Early stopping, best iteration is:
[11192] valid_0's auc: 0.731741
```

```
15  调参后lightgbm单模型在验证集上的AUC：0.7317405262608612
```



可以看到相比最早的原始参数，模型的性能还是有提升的

```
1  """保存模型到本地"""
2  # 保存模型
3  import pickle
4  pickle.dump(final_model_lgb, open('dataset/model_lgb_best.pkl', 'wb'))
```

1. **模型调参小总结**
    a. 集成模型内置的cv函数可以较快的进行单一参数的调节，一般可以用来优先确定树模型的迭代次数
    b. 数据量较大的时候（例如本次项目的数据），网格搜索调参会特别特别慢，不建议尝试
    c. 集成模型中原生库和sklearn下的库部分参数不一致，需要注意，具体可以参考xgb和lgb的官方API

    xgb原生库API， sklearn库下xgbAPI;
    lgb原生库API， sklearn库下lgbAPI

4.6 经验总结

在本节中，我们主要完成了建模与调参的工作，首先在建模的过程中通过划分数据集、交叉验证等方式对模型的性能进行评估验证，并通过可视化方式绘制模型ROC曲线

最后我们对模型进行调参，这部分介绍了贪心调参、网格搜索调参、贝叶斯调参共三种调参手段，重点使用贝叶斯调参对本次项目进行简单优化，大家在实际操作的过程中可以参考调参思路进行优化，不必拘泥于以上教程所写的具体实例。

END.

【 小一 ：Datawhale成员，金融风控爱好者。】

关于Datawhale：

Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

5 Task5 模型融合

Tip:此部分为零基础入门金融风控的 Task5 模型融合部分，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 零基础入门金融风控之贷款违约预测

地址：

5.1 学习目标

将之前建模调参的结果进行模型融合。

尝试多种融合方案，提交融合结果并打卡。（模型融合一般用于A榜比赛的尾声和B榜比赛的全程）

5.2 内容介绍

模型融合是比赛后期上分的重要手段，特别是多人组队学习的比赛中，将不同队友的模型进行融合，可能会收获意想不到的效果哦，往往模型相差越大且模型表现都不错的前提下，模型融合后结果会有大幅提升，以下是模型融合的方式。

1. 平均：
    a. 简单平均法
    b. 加权平均法
2. 投票：
    a. 简单投票法
    b. 加权投票法
3. 综合：
    a. 排序融合
    b. log融合
4. stacking:
    a. 构建多层模型，并利用预测结果再拟合预测。

5. blending：

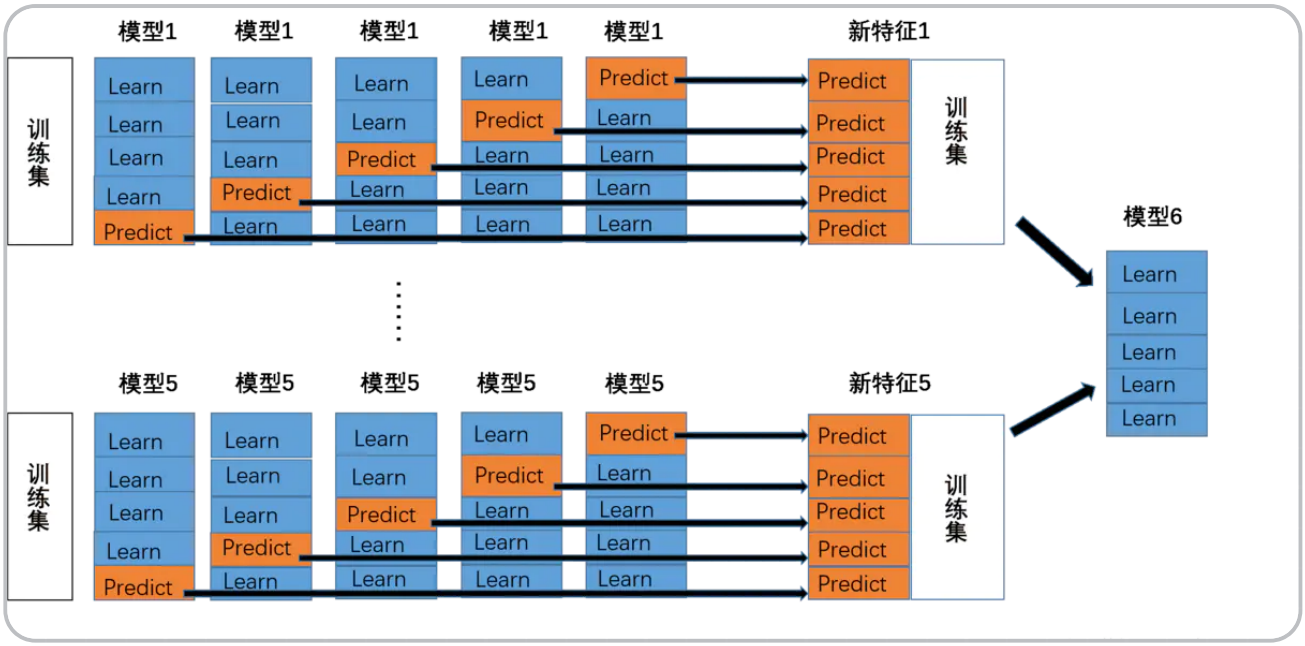
   a. 选取部分数据预测训练得到预测结果作为新特征，带入剩下的数据中预测。Blending只有一层，而Stacking有多层

6. boosting/bagging（在Task4中已经提及，就不再赘述）

5.3 stacking\blending详解

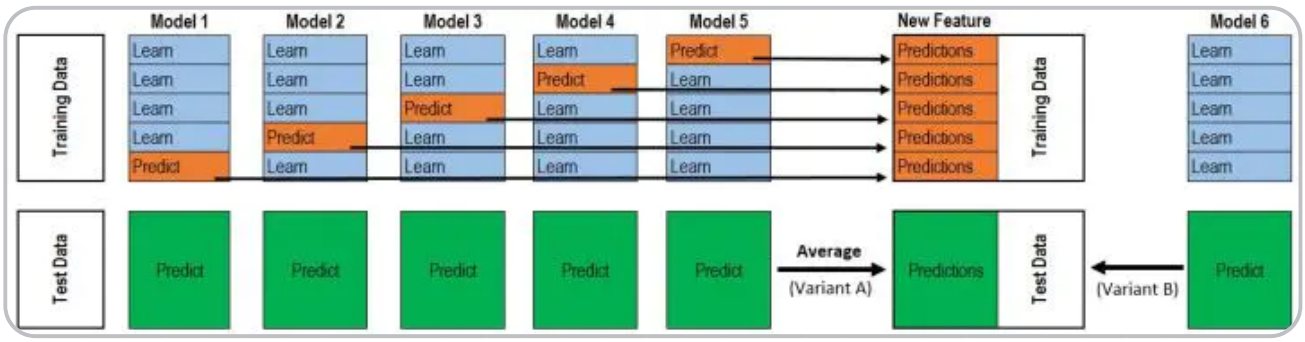
1. stacking

将若干基学习器获得的预测结果，将预测结果作为新的训练集来训练一个学习器。如下图 假设有五个基学习器，将数据带入五基学习器中得到预测结果，再带入模型六中进行训练预测。但是由于直接由五个基学习器获得结果直接带入模型六中，容易导致过拟合。所以在使用五个及模型进行预测的时候，可以考虑使用K折验证，防止过拟合。



1. blending

与stacking不同，blending是将预测的值作为新的特征和原特征合并，构成新的特征值，用于预测。为了防止过拟合，将数据分为两部分d1、d2，使用d1的数据作为训练集，d2数据作为测试集。预测得到的数据作为新特征使用d2的数据作为训练集结合新特征，预测测试集结果。



1. Blending与stacking的不同

   a. stacking

   - 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。
   - 在组队竞赛的过程中，不需要给队友分享自己的随机种子。

   b. Blending

   - blending比stacking简单，不需要构建多层模型。
   - 由于blending对将数据划分为两个部分，在最后预测时有部分数据信息将被忽略。
   - 同时在使用第二层数据时可能会因为第二层数据较少产生过拟合现象。

参考资料：还是没有理解透彻吗？可以查看参考资料进一步了解哦!

https://blog.csdn.net/wuzhongqiang/article/details/105012739

5.4 代码示例

5.4.1 平均

1. 简单加权平均，结果直接融合

   求多个预测结果的平均值。pre1-pren分别是n组模型预测出来的结果，将其进行加权融

```
pre = (pre1 + pre2 + pre3 +...+pren )/n
```

1. 加权平均法

   一般根据之前预测模型的准确率，进行加权融合，将准确性高的模型赋予更高的权重。

```
pre = 0.3pre1 + 0.3pre2 + 0.4pre3
```

5.4.2 投票

1. 简单投票

```
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = XGBClassifier(learning_rate=0.1, n_estimators=150, max_depth=4, min_child_weight=2,
subsample=0.7,objective='binary:logistic')

vclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('xgb', clf3)])
vclf = vclf .fit(x_train,y_train)
print(vclf .predict(x_test))
```

```
- 加权投票
在VotingClassifier中加入参数 voting='soft', weights=[2, 1, 1]，weights用于调节基模型的权重
```

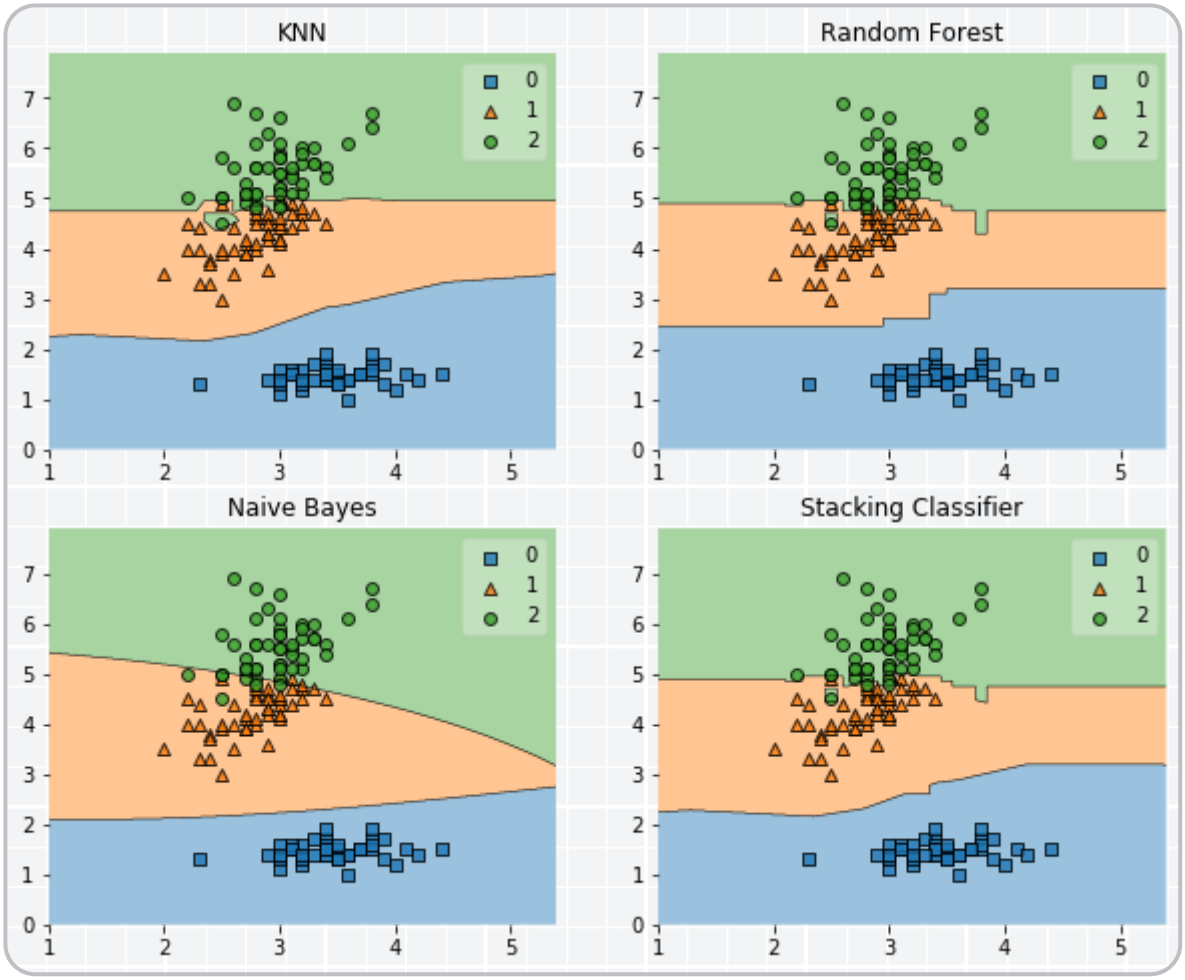
```
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = XGBClassifier(learning_rate=0.1, n_estimators=150, max_depth=4, min_child_weight=2,
subsample=0.7,objective='binary:logistic')

vclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('xgb', clf3)],
voting='soft', weights=[2, 1, 1])
vclf = vclf .fit(x_train,y_train)
print(vclf .predict(x_test))
```

5.4.3 Stacking：

```
import warnings
warnings.filterwarnings('ignore')
import itertools
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import cross_val_score, train_test_split
```

```
from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions


# 以python自带的鸢尾花数据集为例
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target


clf1 = KNeighborsClassifier(n_neighbors=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          meta_classifier=lr)


label = ['KNN', 'Random Forest', 'Naive Bayes', 'Stacking Classifier']
clf_list = [clf1, clf2, clf3, sclf]

fig = plt.figure(figsize=(10,8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)


clf_cv_mean = []
clf_cv_std = []
for clf, label, grd in zip(clf_list, label, grid):

    scores = cross_val_score(clf, X, y, cv=5, scoring='accuracy')
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
    clf_cv_mean.append(scores.mean())
    clf_cv_std.append(scores.std())

    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=clf)
    plt.title(label)


plt.show()
```

```
Accuracy: 0.91 (+/- 0.07) [KNN]
Accuracy: 0.94 (+/- 0.04) [Random Forest]
Accuracy: 0.91 (+/- 0.04) [Naive Bayes]
Accuracy: 0.94 (+/- 0.04) [Stacking Classifier]
```



5.4.4 blending

```
# 以python自带的鸢尾花数据集为例
data_0 = iris.data
data = data_0[:100,:]


target_0 = iris.target
target = target_0[:100]

#模型融合中基学习器
clfs = [LogisticRegression(),
        RandomForestClassifier(),
        ExtraTreesClassifier(),
        GradientBoostingClassifier()]

#切分一部分数据作为测试集
X, X_predict, y, y_predict = train_test_split(data, target, test_size=0.3,
random_state=914)
```

```
17
18
19  #切分训练数据集为d1,d2两部分
20  X_d1, X_d2, y_d1, y_d2 = train_test_split(X, y, test_size=0.5, random_state=914)
21  dataset_d1 = np.zeros((X_d2.shape[0], len(clfs)))
22  dataset_d2 = np.zeros((X_predict.shape[0], len(clfs)))
23
24  for j, clf in enumerate(clfs):
25      #依次训练各个单模型
26      clf.fit(X_d1, y_d1)
27      y_submission = clf.predict_proba(X_d2)[:, 1]
28      dataset_d1[:, j] = y_submission
29      #对于测试集，直接用这k个模型的预测值作为新的特征。
30      dataset_d2[:, j] = clf.predict_proba(X_predict)[:, 1]
31      print("val auc Score: %f" % roc_auc_score(y_predict, dataset_d2[:, j]))
32
33
34  #融合使用的模型
35  clf = GradientBoostingClassifier()
36  clf.fit(dataset_d1, y_d2)
37  y_submission = clf.predict_proba(dataset_d2)[:, 1]
38  print("Val auc Score of Blending: %f" % (roc_auc_score(y_predict, y_submission)))
39
```

5.5 经验总结

1. 简单平均和加权平均是常用的两种比赛中模型融合的方式。其优点是快速、简单。
2. stacking在众多比赛中大杀四方，但是跑过代码的小伙伴想必能感受到速度之慢，同时stacking多层提升幅度并不能抵消其带来的时间和内存消耗，所以实际环境中应用还是有一定的难度，同时在有答辩环节的比赛中，主办方也会一定程度上考虑模型的复杂程度，所以说并不是模型融合的层数越多越好的。
3. 当然在比赛中将加权平均、stacking、blending等混用也是一种策略，可能会收获意想不到的效果哦！