# LightGBM调参

https://github.com/wenkaicn/ML-000/blob/main/Week16/lending\_ml.ipynb

## 数据探索

对业务逻辑做了充分了解，对特征做了归纳分类，了解了各特征的值分布情况。

个人改进：结合目标loan\_status对特征做相关性分析；使用Confusion Matrix分析结果

## 特征工程

做了很多尝试。有效果的：

1. 本次贷款月还款/月收入；
2. funded\_amnt 与funded\_amnt\_inv相减：（尚未理解业务含义）
3. 离散变量用count编码取代onehot：
4. 还款周期term的onehot编码与信用acc相乘（信用acc按还款周期拆为两列）

没有效果的，如：

1. 按不同地区算收入平均值，构造新特征：个人收入/地区收入平均值
2. one-hot编码还原：grade和sub\_grade都是one-hot编码。推测原始数据中的grade是信用等级，不同等级的用户信用存在排序关系，尝试将one-hot编码转为顺序数字

个人改进：特征工程有些地方凭感觉做决定，希望多掌握些通用原则以提高效率。

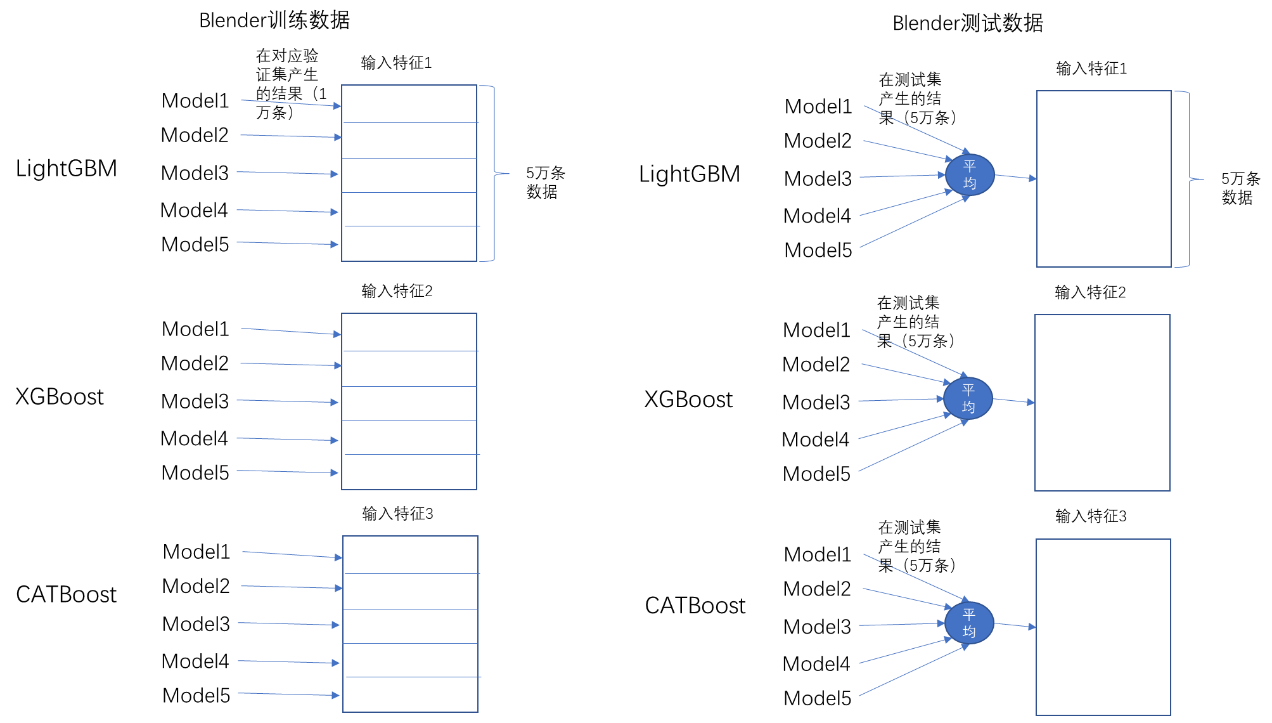
【问题：】

1. 信用acc按还款周期拆为两列，验证有效。其他变量是否也应按不同还款周期拆分？**把所有变量都按还款周期拆分（方案1），是否等价于方案2将目标变量 loan\_status 更改为0，1，2，3**（三年已还、三年未还、五年已还、五年未还）？或者**是否等价于方案3将三年期和五年期的数据拆开分别训练成两个模型？**（备注：方案2multi-class的方法在integration做过验证，结果为0.91732差距较大。方案3在lending\_ml中的“附加验证”中做了验证，初步看效果为 0.92。似乎方案3与方案1等价？）
2. 信用acc按还款周期拆为两列，验证有效，是否应该将原有的acc列删除？
3. 离散变量的onehot改为count编码，选取了“addr\_state，emp\_length，purpose，grade，sub\_grade”等onehot列较多的变量。而对“home\_ownership”、“application\_type”等onehot列较少的变量没做处理。类似编码转换这样的通用操作，对部分有效，是否应该对全部离散变量做相同处理？
4. 未产生效果的两个尝试，尤其是one-hot编码还原，思路是否正确？
5. 5折cv，为什么验证结果总是好于测试结果？验证时用的也是模型训练时没看到的数据

## Stacking

将LightGBM、XGBoost、CATBoost三个模型做Stacking，思路：

1. blender训练数据构建：LightGBM模型是五折cv训练出来的五个子模型，将每个子模型在对应验证集上的预测结果纵向拼在一起，作为Stacking层的1个输入特征。同样处理XGBoost和CATBoost，获得另外2个输入特征。输出特征即为训练数据的目标值。
2. Blender测试：用LightGBM的5个子模型对测试集分别做预测，对预测结果做平均得到1个输入特征。XGBoost和CATBoost做同样处理得到另2个输入特征，使用这3个输入特征用blender得到最终预测结果



【问题】：

使用上述Stacking+cv方法，有效果提升。但不确定是否是最好的方法？疑惑的地方在于：blender训练时，特征1的某个值是LightGBM的某个子模型的预测，而在blender测试时，特征1的某个值是LightGBM的5个子模型的预测的平均值。这两者是有明显差异的。

## 总结：

特征工程汇总，得到测试结果为0.91856,相比baseline 0.91792 ，提升 0.064%

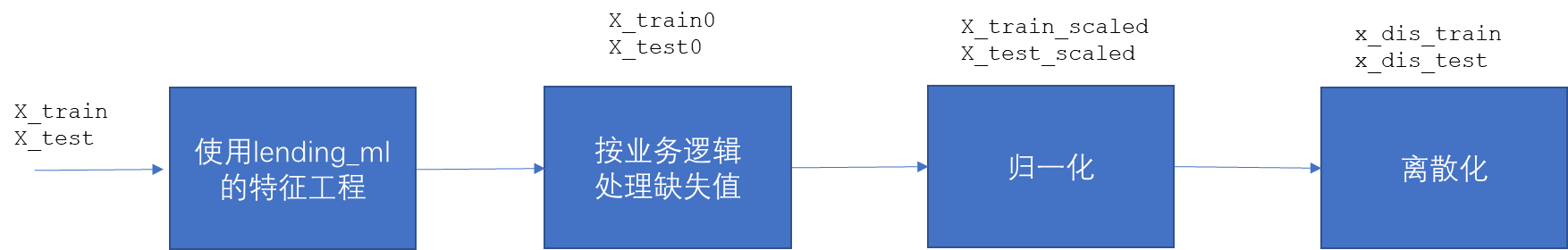
Stacking，比单个模型的最好结果有提升：0.9187-0.91856 = 0.014%。Stacking有初步效果

使用colab会经常遇到实例释放，导致已训练的结果丢失。编写了Utils，但或许TensorBoard有现成的功能，需进一步了解。

# 神经网络训练

https://github.com/wenkaicn/ML-000/blob/main/Week16/lending\_dl.ipynb

## 数据预处理



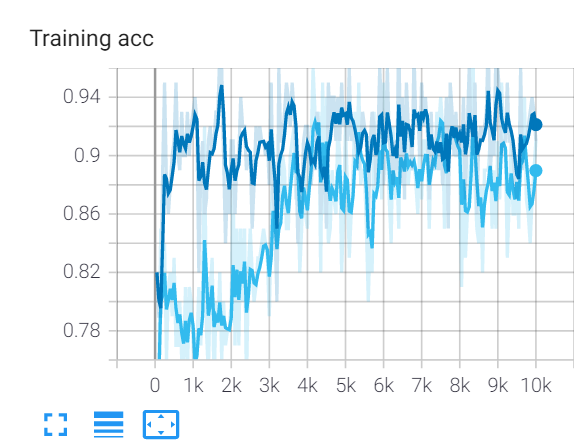
训练神经网络前需处理缺失值，结合业务逻辑填补缺失值的思路：

* 没有联合申请人，等价于联合申请人的年收入为0；
* continuous\_dti的NaN是因为有人年收入为0造成的。将dti改为“收入/贷款”，即1/(dti+1)。然后将NaN改为0。分母为dti+1是因为原有continuous\_dti有0值，变为分母时加1处理。
* continuous\_dti\_joint、continuous\_loan\_income按相同逻辑更改
* 剩下的空值，是类似continuous\_mths\_since\_last\_delinq，距不良记录以来的时间，这个时间越长信用越高。将没有不良记录的统一设置为240个月（20年）

对输入神经网络的数据做归一化处理. 注意：这里是对训练集fit，对测试集transform.

## 神经网络MLP

尝试MLP（之前作业对Tabnet改进不到位，普通MLP模型的效果比Tabnet还好）。同时验证了归一化处理效果明显：如图，有归一化的效果约0.91



## 离散+Embedding

**曲线改为epoch粒度**

Tensorboard以step为粒度曲线波动太大不好比较，通过如下两个方式改为epoch粒度

self.log('train\_loss', loss, on\_epoch=True, logger=True)

val\_check\_interval=500 #设置batch\_size = 100，5万条数据的1个epoch就是500 steps

**离散化和Embedding层**

MLP加上对数据离散化、Embedding处理（这里直接采用之前作业经验，未确认是否有效，在5折cv最后做了补充验证，基本有效）

Embedding处理时，有值过于集中的情况，放弃了Embedding with Distance改进

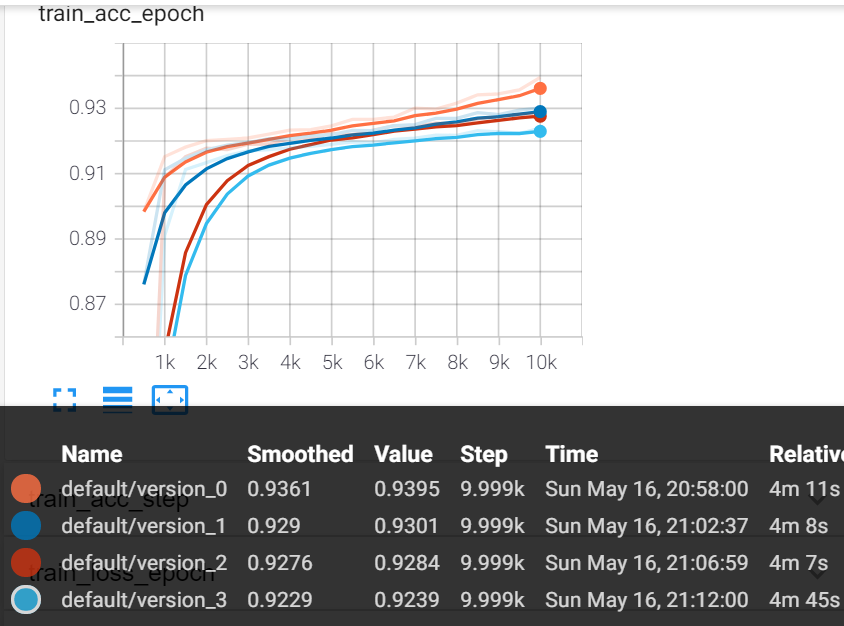
**调参：选取基本上收敛的模型，比较不同参数组合效果差异**

以训练集5万条数据做训练，以测试集5万条数据作为验证。先看看不同参数的大致效果

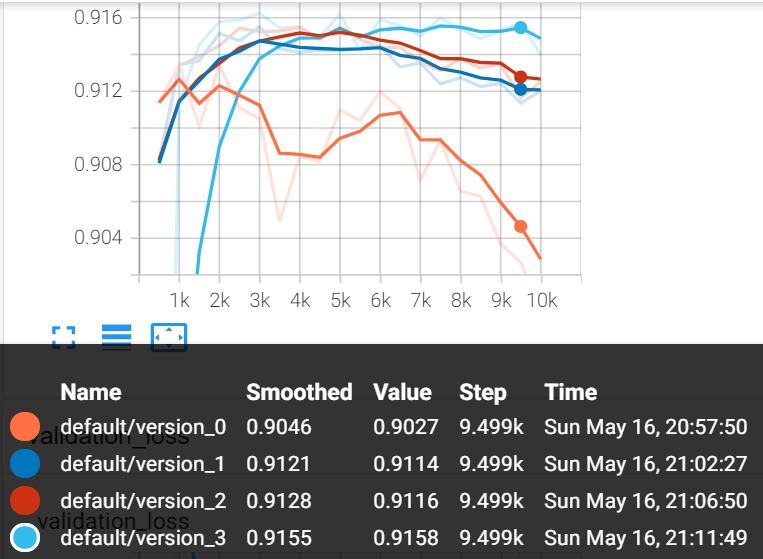
1. 学习率为 1e-3 （网络宽度 64，深度10）
2. 调小学习率为 1e-4
3. 学习率WarmUp
4. 在WarmUp基础上，调网络 宽度32，深度 15

**以上四种情况的训练结果**

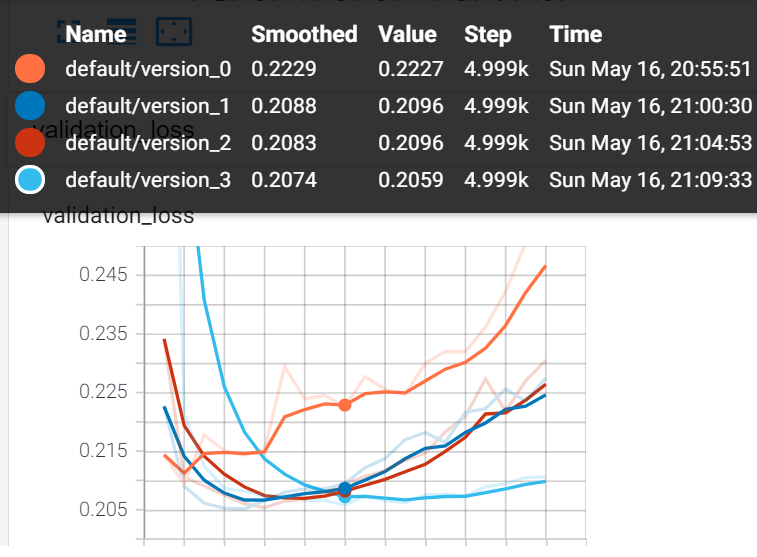
train\_acc



validation acc



 validation\_loss



通过图型，大致认定第4次，即采用warmup并更改网络结构，效果最好

1、学习率调小的效果比较明显

2、第3第4组采用了WarmUp ，效果比第2组好一点

3、第4组更改网络结构，相比第3次没有那么过拟合，更收敛

【问题】：

是否应比较各次训练的validation acc的最高值？例如，如果第3次训练的acc最高值大于第4次的最高值，那么选择第4次训练的参数组合是否还正确？

**使用trainer.test做测试**

基于pytorch lightning文档，使用trainer.test会自动选用最好的权重（？）

# call after training  
trainer = Trainer()  
trainer.fit(model)

# automatically auto-loads the best weights  
trainer.test(test\_dataloaders=test\_dataloader)

基于第4组参数，训练两次并使用测试集5万条数据做测试，结果如下

'test\_acc': 0.9139599800109863, 'test\_loss': 0.2107117772102356,

'test\_acc': 0.9145799875259399, 'test\_loss': 0.21014273166656494,

两次训练参数相同，但由于初始化不同等导致每次跑的结果有差异。相差0.06%有点大，以LightGBM经验来看这个差别很明显，会影响参数选择。

【问题】：

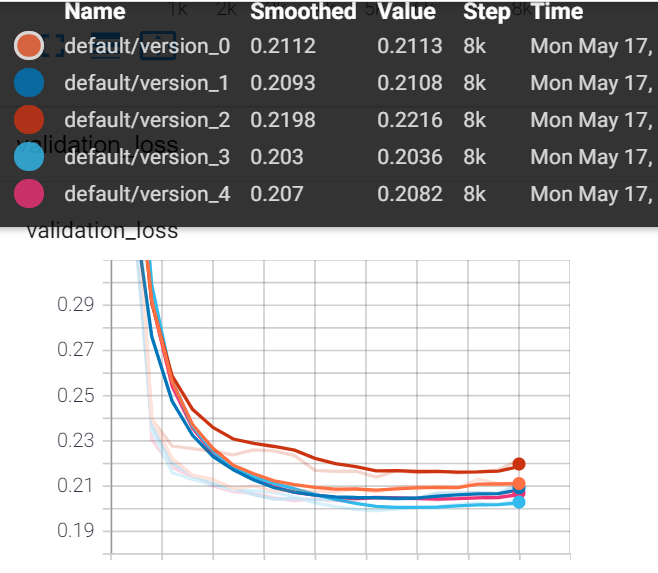
对比tensor board的记录，使用trainer.test的结果并非tensor board中记录的最好结果？而是最后一个step的结果？

## 5折cv

用测试集5万条数据做验证是不合适的（仅仅用于感觉不同参数的效果）。使用5折cv的方法，选取不同参数组合。

用第4组参数(32\*15)跑5折cv得到5个子模型，测试集5万条数据的测试结果：0.91636





调参：使用第五组参数（48\*12），效果好于第四组参数

测试结果 ：0.91678；重复跑第二次测试结果 ：0.91682。只有0.004%的波动，可接受。

【问题】：

1. 从validation acc和loss图型看，到最后有点过拟合，是否还应选择最好的epochs？（而不是固定为20）
2. 参数相同情况下，采用5折cv，测试效果提升很明显。从数据的使用程度看，之前的方式贪心地用了5万条测试数据做验证，单从指标上看按理不应差太多，难道是因为5折cv有5个子模型的Stacking的因素在起作用？

## 总结：

对Learning rate和网络宽度深度做了调试，有效果；使用5折cv也有明显收效**0.91682**

integration中将MLP模型加入Stacking效果为0.91828，相比仅用三个树模型Stacking的0.9187没有提升

经验：别把数据集弄错了（曾把MLP模型的输入搞错导致无效果）

后补齐Pytorch和Pytorch-lightning基本概念，提高效率和提升代码结构