《Kaggle Top 1%方案精讲与实践》

课程说明：

小伙伴好呀~欢迎来到《2021机器学习实战训练营》试学体验课！我是课程主讲老师，九天。

本次体验课为期三天（12月8-10号），期间每晚8点在我的B站直播间公开直播，直播间地址:https://live.bilibili.com/22678166

本期公开课的内容是在上一轮Kaggle竞赛公开课（《Elo Merchant Category Recommendation》）的基础上，进一步探讨如何进一步将排名提升至1%。在接下来的三天内容中，我们将进一步尝试多模型建模与优化、模型融合方法、特征优化与Trick优化等方法，从而在此前的结果基础上大幅提高模型预测准确率。

当然，没有参与上一轮公开课的小伙伴也不用担心，本轮公开课将在开始时我们将快速回顾上一轮公开课内容，并将提供上一轮公开课最终完成的数据处理结果与相关代码，帮助大家无门槛进入本轮课程内容的学习中。当然，如果时间允许，也希望大家能够通过观看上一轮公开课的直播录屏，以巩固相关知识。上一轮公开课直播录屏地址：https://www.bilibili.com/video/BV1QU4y1u7Ph

课程资料/赛题数据/课程代码/付费课程信息，扫码添加客服“小可爱”回复【kaggle】即可领取哦~

另外，双十二年终大促持续进行中，十八周80+课时体系大课限时七折，扫码咨询小可爱回复“优惠”，还可领取额外折上折优惠，课程主页：https://appze9inzwc2314.pc.xiaoe-tech.com

---

【Kaggle】Elo Merchant Category Recommendation

竞赛案例解析公开课 Part II

Day 6.特征优化与模型融合优化

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm,tqdm_notebook
import lightgbm as lgb
import xgboost as xgb
from catboost import CatBoostRegressor

from catboost import CatBoostClassifier
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import BayesianRidge
from catboost import CatBoostRegressor
from sklearn.model_selection import KFold, StratifiedKFold
from scipy import sparse
import warnings
import time
import sys
import os
import gc
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
from sklearn.metrics import log_loss
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore")
pd.set_option('display.max_columns',None)
pd.set_option('max_colwidth',100)
```

一、优化思路梳理

- 课前准备

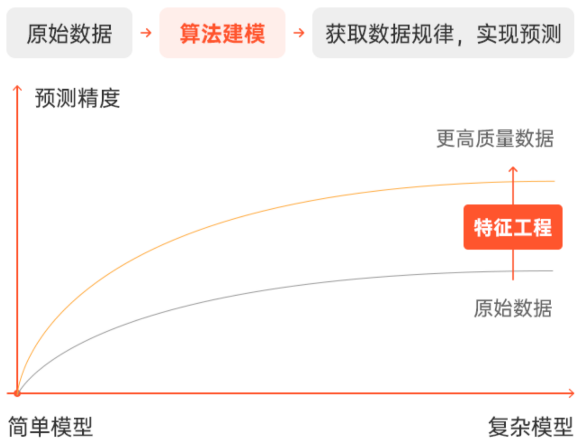
在昨天的内容中，我们通过使用更强的集成模型以及模型融合的方法，已经顺利将比赛分数提高至前20%。但正如此前所说，之前的一系列操作只不过是遵循了常规操作流程进行的数据处理与建模，若希望能够更进一步提高模型分数，则需要因地制宜、活学活用，在考虑到当前数据及特殊情况下进行有针对性的策略调整。本节内容我们将结合此前搜集到的所有数据集信息及业务背景信息，进行最后一轮的特征优化与模型优化，并最终将排名提升至前1%。

当然，这个过程并不简单，若需要跟上本节内容的讨论，需要非常熟悉当前数据集的基本情况，也就是需要深度掌握Day1-2所介绍内容，从而才能理解接下来的特征优化相关内容；此外也需要对Day 5中介绍的集成学习建模策略，即在使用原生算法库情况下，如何配合交叉验证过程、借助贝叶斯优化器进行超参数搜索，并最终输出交叉验证的测试集预测均值作为最终预测结果的一整个流程，从而才能更快速的理解本节开始我们对模型训练流程进行的优化与调整；此外，我还将在本节介绍非常适用于竞赛场景的融合技巧，亦可作为同学日后参与竞赛时的有力工具。不过没有跟上此前的内容的同学也不用担心，本节内容将更加强调优化过程的整体逻辑，并尽可能从一个更加通俗且准确的角度进行解释，大家也可以在听完本节内容后再去回顾此前Day1-5的相关内容，以终为始、未尝不可，通过反复观看，也相信大家会对本节内容有一个更深刻的理解。

- 整体优化思路

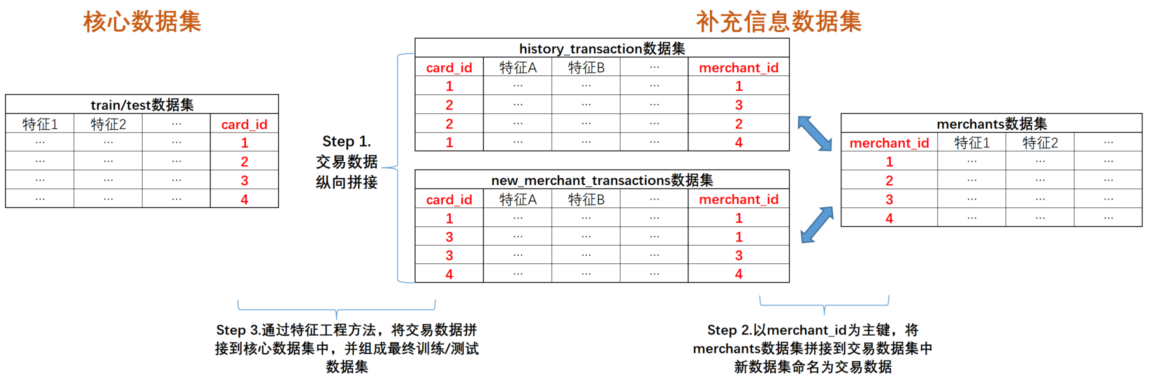
对于机器学习来说，总的来看有两种建模思路，其一是通过特征工程方法进一步提升数据

质量，其二则是通过更加复杂的模型或更加有效的模型融合技巧来提升建模效果，并且就二者的关系来看，正如时下流行的观点所说，特征工程将决定模型效果上界，而建模过程则会不断逼近这个上界。但无论如何，在优化的过程中，需要二者配合执行才能达到更好的效果。



1.特征优化思路

首先，先来看特征优化思路。在此前的建模过程中，我们曾不止一次的对特征进行了处理，首先是在数据聚合时（以card_id进行聚合），为了尽可能提取更多的交易数据信息与商户信息带入进行模型，我们围绕交易数据表和商户数据表进行了工程化批量特征衍生，彼时信息提取流程如下：


核心数据集
补充信息数据集
Step 1.
交易数据
纵向拼接
Step 3.通过特征工程方法，将交易数据拼接到核心数据集中，并组成最终训练/测试数据集
Step 2.以merchant_id为主键，将merchants数据集拼接到交易数据集中
新数据集命名为交易数据

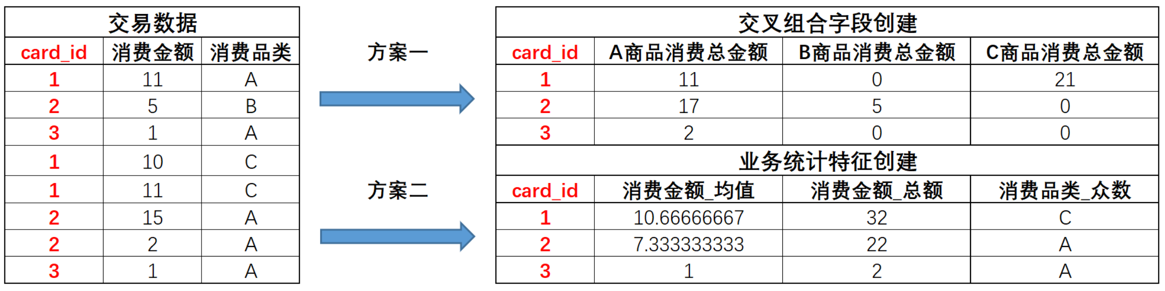
train/test数据集			
特征1	特征2	…	card_id
…	…	…	1
…	…	…	2
…	…	…	3
…	…	…	4

history_transaction数据集				
card_id	特征A	特征B	…	merchant_id
1	…	…	…	1
2	…	…	…	3
2	…	…	…	2
1	…	…	…	4

new_merchant_transactions数据集				
card_id	特征A	特征B	…	merchant_id
1	…	…	…	1
3	…	…	…	1
3	…	…	…	3
4	…	…	…	4

merchants数据集			
merchant_id	特征1	特征2	…
1	…	…	…
2	…	…	…
3	…	…	…
4	…	…	…

具体的特征衍生方案是采用了交叉组合特征创建与业务统计特征创建两种方案：



交易数据		
card_id	消费金额	消费品类
1	11	A
2	5	B
3	1	A
1	10	C
1	11	C
2	15	A
2	2	A
3	1	A

交叉组合字段创建			
card_id	A商品消费总金额	B商品消费总金额	C商品消费总金额
1	11	0	21
2	17	5	0
3	2	0	0

业务统计特征创建			
card_id	消费金额_均值	消费金额_总额	消费品类_众数
1	10.66666667	32	C
2	7.333333333	22	A
3	1	2	A

该过程的详细讲解，可参考Day 3-Day 4的课程内容。总而言之，通过该过程，我们顺利的提取了交易信息表和商户信息表中的数据带入进行建模，并且借助随机森林模型，顺利跑通Baseline。但值得一提的是，在上述流程中，我们其实只是采用了一些工程化的通用做法，这些方法是可以快速适用于任何数据集的特征衍生环节，同时这样的方法也应该是所有建模开始前必须尝试的做法，但既然是“通用”方法，那必然无法帮我们在实际竞赛中脱颖而出。

当然，我们也曾尝试过进行有针对性的特征优化，在Day 5的内容中，我们曾采用NLP方法用于提取特征ID列的信息，并得到了一系列能够更加细致描述用户行为信息与商品偏好的特征，借助该特征，我们最终训练得出了一个效果更好的模型，该结果也进一步验证了特征优化对模型效果提升所能起到的作用。接下来我怕们也将尝试进一步进行有针对性的特征优化。

总体来看，特征优化需要结合数据集当前的实际情况来制定，在已有批量衍生的特征及NLP特征的基础上，针对上述数据集，还可以有以下几点优化方向：

- 用户行为特征

首先，我们注意到，每一笔信用卡的交易记录都有交易时间，而对于时间字段和文本字段，普通的批量创建特征的方法都是无法较好的挖掘其全部信息的，因此我们需要围绕交易字段中的交易时间进行额外的特征衍生。此处我们可以考虑构造一些用于描述用户行为习惯的特征（经过反复验证，用户行为特征是最为有效的提高预测结果的特征类），包括最近一次交易与首次交易的时间差、信用卡激活日期与首次交易的时间差、用户两次交易平均时间间隔、按照不同交易地点/商品品类进行聚合（并统计均值、方差等统计量）。

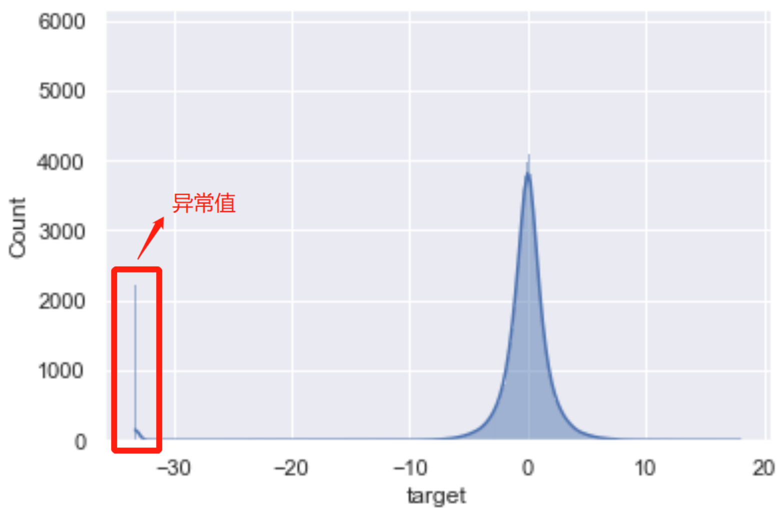
此外，我们也知道越是接近当前时间点的用户行为越有价值，因此我们还需要重点关注用户最近两个月（实际时间跨度可以自行决定）的行为特征，以两个月为跨度，进一步统计该时间周期内用户的上述交易行为特点，并带入模型进行训练。

- 二阶交叉特征

在此前的特征衍生过程中，我们曾进行了交叉特征衍生，但只是进行了一阶交叉衍生，例如交易额在不同商品上的汇总，但实际上还可以进一步构造二阶衍生，例如交易额在不同商品组合上的汇总。通常来说更高阶的衍生会导致特征矩阵变得更加稀疏，并且由于每一阶的衍生都会创造大量特征，因此更高阶的衍生往往也会造成维度爆炸，因此高阶交叉特征衍生需要谨慎。不过正如此前我们考虑的，由于用户行为特征对模型结果有更大的影响，因此我们可以单独围绕用户行为数据进行二阶交叉特征衍生，并在后续建模前进行特征筛选。

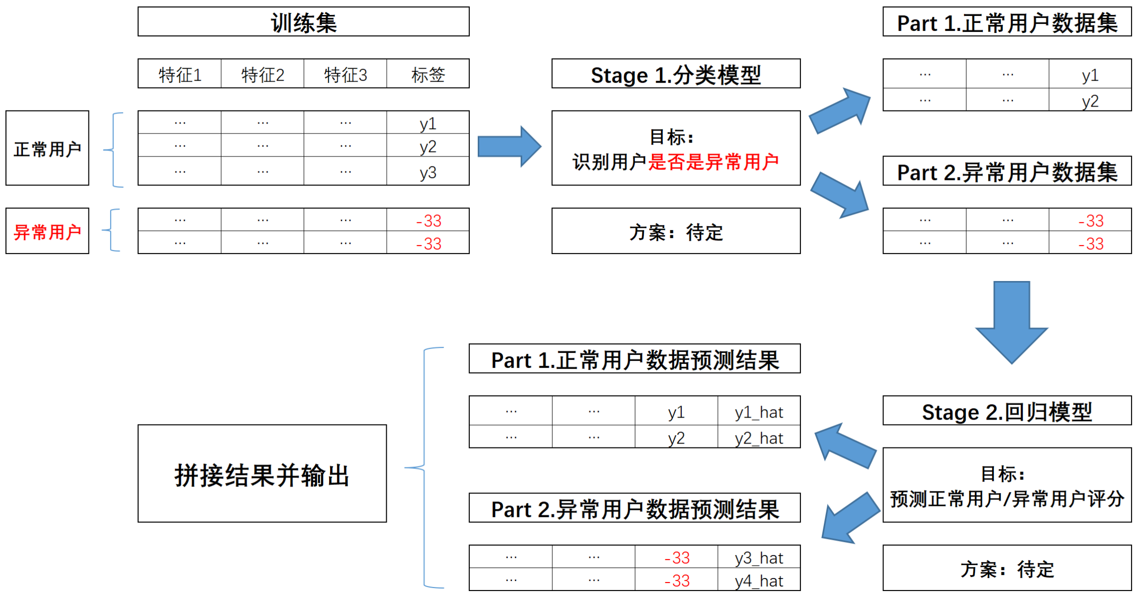
- 异常值识别特征

在Day 1的数据探索中我们就发现，训练数据集的标签中存在少量极端异常值的情况



而若我们更进一步就此前的建模结果展开分析的话，我们会发现，此前的模型误差大多数都源于对异常值用户(card_id)的预测结果。而根据Day 1的讨论我们知道，实际上用户评分是通过某套公式人工计算得出的，因此这些异常值极有可能是某类特殊用户的标记，因此我们不妨在实

际建模过程中进行两阶段建模，即先预测每个输入样本是否是异常样本，并根据分类预测结果进行后续的回归建模，基本流程如下：



而为了保证后续两阶段建模时第一阶段的分类模型能够更加准确的识别异常用户，我们需要创建一些基于异常用户的特征聚合字段，例如异常用户平均消费次数、消费金额等。

2.模型优化思路

- 新增CatBoost模型

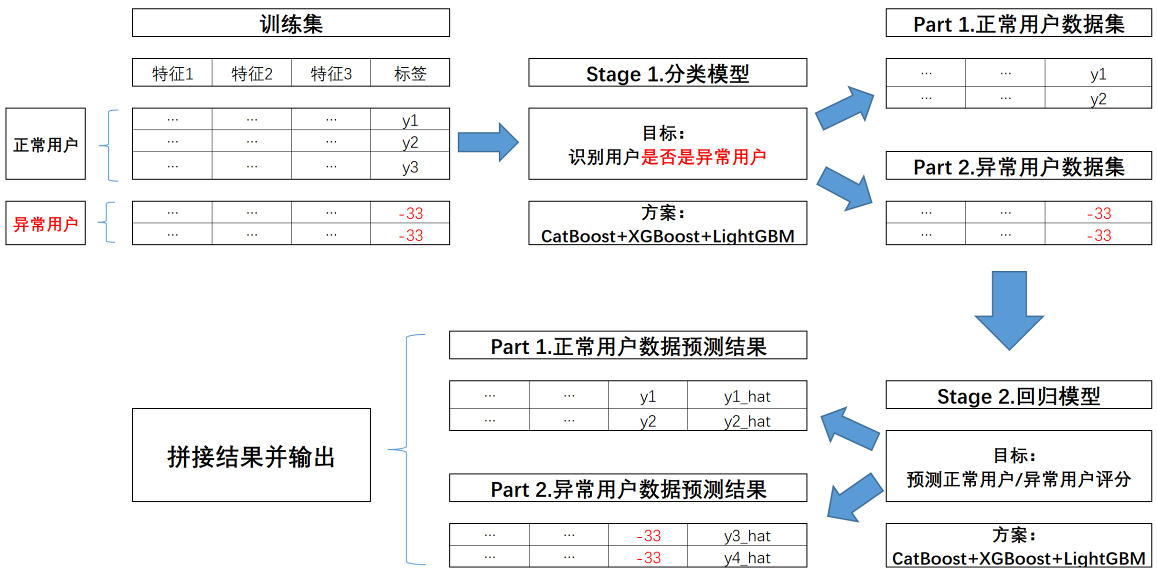
　　相比特征优化思路，模型优化的思路相对简单，首先从模型选择来看，相较LightGBM和XGBoost，随机森林对当前数据集的预测能力较弱，可以考虑将其换成集成算法新秀：CatBoost。

　　CatBoost是由俄罗斯搜索引擎Yandex在2017年7月开源的一个GBM算法，自开源之日起，CatBoost就因为其强大的效果与极快的执行效率广受算法工程人员青睐。在诸多CatBoost的算法优势中，最引人关注的当属该模型能够自主采用独热编码和平均编码的混合策略来处理类别特征，也就是说CatBoost将一些经过实践验证的、普遍有效的特征工程方法融入了实际模型训练过程；此外，CatBoost还提出了一种全新的梯度提升的机制，能够非常好的在经验风险和结构风险中做出权衡（即能够很好的提升精度、同时又能够避免过拟合问题）。

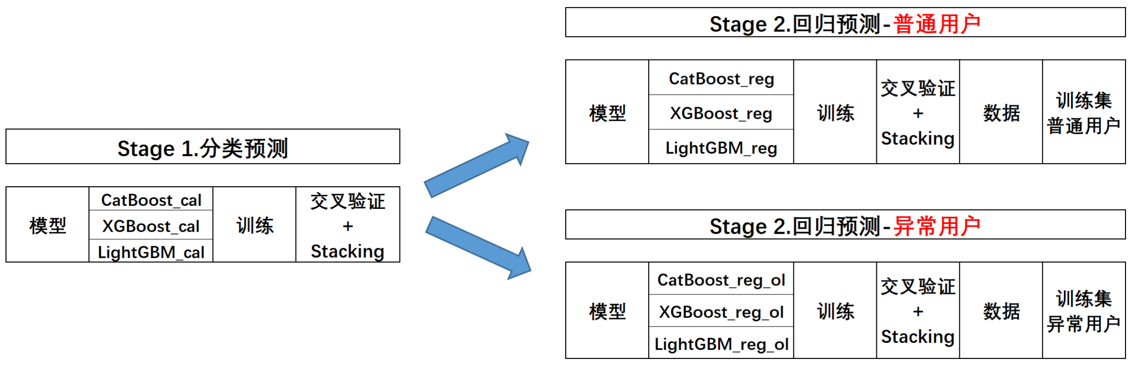
　　而在后续的建模环节中，我们就将使用CatBoost替换随机森林，并最终带入CatBoost、XGBoost和LightGBM三个模型进行模型融合。

- 二阶段建模

　　而从模型训练流程角度出发，则可以考虑进行二阶段建模，基本流程如下：



并且，需要注意的是，在实际二阶段建模过程时，我们需要在每个建模阶段都进行交叉验证与模型融合，才能最大化提升模型效果。也就是说我们需要训练三组模型（以及对应进行三次模型融合），来完成分类预测问题、普通用户回归预测问题和异常用户回归预测问题。三轮建模关系如下：



不难发现，整体建模与融合过程都将变得更加复杂。不过这也是更加贴近真实状态的一种情况，很多时候算法和融合过程都只是元素，如何构建一个更加精准、高效的训练流程，才是进阶的算法工程人员更需要考虑的问题。

3.其他注意事项

- 内存管理

由于接下来我们需要反复读取数据文件并进行计算，因此需要时刻注意进行内存管理，除了可以通过及时删除不用的变量并使用动态垃圾回收机制来清理内存外，还可以使用如下方式在定义数据类型时尽可能在不影响数值运算的前提下给出更加合适的数据类型：

```
In [ ]:
%%time
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            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)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(end_mem,
    return df

new_transactions = pd.read_csv('data/new_merchant_transactions.csv', parse_dates=['purchas
historical_transactions = pd.read_csv('data/historical_transactions.csv', parse_dates=['purchase
for col in ['authorized_flag', 'category_1']:
    historical_transactions[col] = historical_transactions[col].map({'Y':1, 'N':0})
    new_transactions[col]        = new_transactions[col].map({'Y':1, 'N':0})
```

- 代码管理

本部分代码量较大，尽管课上是通过Notebook进行展示，但如果在实际建模过程时，更建议通过自定义模块来存储和调用大段代码。

二、数据预处理与特征优化

1.数据预处理

此处由于需要进行特征优化，因此数据处理工作需要从头开始执行。首先需要进行数据读取，并进行缺失值处理：

In [ ]:

```
%%time
## 加载训练集，测试集，基本处理
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')

target = train['target']
for df in [train, test]:
    df['year']  = df['first_active_month'].fillna('0-0').apply(lambda x:int(str(x).split('-')[0]))
    df['first_active_month'] = pd.to_datetime(df['first_active_month'])
    df['elapsed_time'] = (datetime.date(2018,3, 1) - df['first_active_month'].dt.date).dt.days

    df['first_active_month'] = pd.to_datetime(df['first_active_month'])
    df['weekofyear'] = df['first_active_month'].dt.weekofyear
    df['dayofyear'] = df['first_active_month'].dt.dayofyear
    df['month'] = df['first_active_month'].dt.month

## 交易表合并train test
train_test = pd.concat([train[['card_id','first_active_month']], test[['card_id','first_active_month']]
historical_transactions   = historical_transactions.merge(train_test[['card_id','first_active_month
new_transactions = new_transactions.merge(train_test[['card_id','first_active_month']], on=['ca
```

然后需要进行时间字段的更细粒度的呈现：

```
In [ ]: %%time
        def month_trans(x):
            return x // 30

        def week_trans(x):
            return x // 7

        ## 交易表预处理
        def get_expand_common(df_):
            df = df_.copy()

            df['category_2'].fillna(1.0,inplace=True)
            df['category_3'].fillna('A',inplace=True)
            df['category_3'] = df['category_3'].map({'A':0, 'B':1, 'C':2})
            df['merchant_id'].fillna('M_ID_00a6ca8a8a',inplace=True)
            df['installments'].replace(-1, np.nan,inplace=True)
            df['installments'].replace(999, np.nan,inplace=True)
            df['installments'].replace(0, 1,inplace=True)

            df['purchase_amount'] = np.round(df['purchase_amount'] / 0.00150265118 + 497.06,8)
            df['purchase_amount'] = df.purchase_amount.apply(lambda x: np.round(x))

            df['purchase_date']          =  pd.to_datetime(df['purchase_date'])
            df['first_active_month']     =  pd.to_datetime(df['first_active_month'])
            df['purchase_hour']          =  df['purchase_date'].dt.hour
            df['year']                  = df['purchase_date'].dt.year
            df['month']                  =  df['purchase_date'].dt.month
            df['day']                   = df['purchase_date'].dt.day
            df['hour']                   = df['purchase_date'].dt.hour
            df['weekofyear'] = df['purchase_date'].dt.weekofyear
            df['dayofweek']              =  df['purchase_date'].dt.dayofweek
            df['weekend']                =  (df.purchase_date.dt.weekday >=5).astype(int)
            df                          =  df.sort_values(['card_id','purchase_date'])
            df['purchase_date_floorday'] =  df['purchase_date'].dt.floor('d')  #删除小于day的时间

            # 距离激活时间的相对时间,0, 1,2,3,...,max-act
            df['purchase_day_since_active_day']   = df['purchase_date_floorday'] - df['first_active_month
            df['purchase_day_since_active_day']   = df['purchase_day_since_active_day'].dt.days  #.astyp
            df['purchase_month_since_active_day'] = df['purchase_day_since_active_day'].agg(month_tra
            df['purchase_week_since_active_day']  = df['purchase_day_since_active_day'].agg(week_trans

            # 距离最后一天时间的相对时间,0,1,2,3,...,max-act
            ht_card_id_gp = df.groupby('card_id')
            df['purchase_day_since_reference_day']   =  ht_card_id_gp['purchase_date_floorday'].transfor
            df['purchase_day_since_reference_day']   =  df['purchase_day_since_reference_day'].dt.days
            # 一个粗粒度的特征(距离最近购买过去了几周，几月)
            df['purchase_week_since_reference_day']  = df['purchase_day_since_reference_day'].agg(wee
            df['purchase_month_since_reference_day'] = df['purchase_day_since_reference_day'].agg(mo

            df['purchase_day_diff']    =  df['purchase_date_floorday'].shift()
            df['purchase_day_diff']    =  df['purchase_date_floorday'].values - df['purchase_day_diff'].valu
            df['purchase_day_diff']    =  df['purchase_day_diff'].dt.days
            df['purchase_week_diff']   =  df['purchase_day_diff'].agg(week_trans).values
            df['purchase_month_diff']  =  df['purchase_day_diff'].agg(month_trans).values

            df['purchase_amount_ddgd_98']  = df['purchase_amount'].values * df['purchase_day_since_r
            df['purchase_amount_ddgd_99']  = df['purchase_amount'].values * df['purchase_day_since_r
            df['purchase_amount_wdgd_96']  = df['purchase_amount'].values * df['purchase_week_since
            df['purchase_amount_wdgd_97']  = df['purchase_amount'].values * df['purchase_week_since
            df['purchase_amount_mdgd_90']  = df['purchase_amount'].values * df['purchase_month_sinc
            df['purchase_amount_mdgd_80']  = df['purchase_amount'].values * df['purchase_month_sinc

            df = reduce_mem_usage(df)
```

```
    return df

historical_transactions = get_expand_common(historical_transactions)
new_transactions        = get_expand_common(new_transactions)
```

2.特征优化部分

在执行完数据清洗与时间字段的处理之后，接下来我们需要开始执行特征优化。根据此前介绍的思路，首先我们需要进行基础行为特征字段衍生：

In [ ]:
```
%%time
## 构造基本统计特征
def aggregate_transactions(df_, prefix):

    df = df_.copy()

    df['month_diff'] = ((datetime.datetime.today() - df['purchase_date']).dt.days)//30
    df['month_diff'] = df['month_diff'].astype(int)
    df['month_diff'] += df['month_lag']

    df['price'] = df['purchase_amount'] / df['installments']
    df['duration'] = df['purchase_amount'] * df['month_diff']
    df['amount_month_ratio'] = df['purchase_amount'] / df['month_diff']

    df.loc[:, 'purchase_date'] = pd.DatetimeIndex(df['purchase_date']).\
                                 astype(np.int64) * 1e-9

    agg_func = {
        'category_1':      ['mean'],
        'category_2':      ['mean'],
        'category_3':      ['mean'],
        'installments':    ['mean', 'max', 'min', 'std'],
        'month_lag':       ['nunique', 'mean', 'max', 'min', 'std'],
        'month':           ['nunique', 'mean', 'max', 'min', 'std'],
        'hour':            ['nunique', 'mean', 'max', 'min', 'std'],
        'weekofyear':      ['nunique', 'mean', 'max', 'min', 'std'],
        'dayofweek':       ['nunique', 'mean'],
        'weekend':         ['mean'],
        'year':            ['nunique'],
        'card_id':         ['size','count'],
        'purchase_date':   ['max', 'min'],
        ###
        'price':           ['mean','max','min','std'],
        'duration':        ['mean','min','max','std','skew'],
        'amount_month_ratio':['mean','min','max','std','skew'],
        }

    for col in ['category_2','category_3']:
        df[col+'_mean'] = df.groupby([col])['purchase_amount'].transform('mean')
        agg_func[col+'_mean'] = ['mean']

    agg_df = df.groupby(['card_id']).agg(agg_func)
    agg_df.columns = [prefix + '_'.join(col).strip() for col in agg_df.columns.values]
    agg_df.reset_index(drop=False, inplace=True)

    return agg_df
print('generate statistics features...')
auth_base_stat = aggregate_transactions(historical_transactions[historical_transactions['author
print('generate statistics features...')
hist_base_stat = aggregate_transactions(historical_transactions[historical_transactions['authori
print('generate statistics features...')
new_base_stat  = aggregate_transactions(new_transactions, prefix='new_')
```

In [ ]:

```
%%time
def get_quantile(x, percentiles = [0.1, 0.25, 0.75, 0.9]):
    x_len = len(x)
    x = np.sort(x)
    sts_feas = []
    for per_ in percentiles:
        if per_ == 1:
            sts_feas.append(x[x_len - 1])
        else:
            sts_feas.append(x[int(x_len * per_)])
    return sts_feas

def get_cardf_tran(df_, month = 3, prefix = '_'):

    df = df_.copy()
    if prefix == 'hist_cardf_':
        df['month_to_now']  =  (datetime.date(2018, month, 1) - df['purchase_date_floorday'].dt.d

    df['month_diff'] = ((datetime.datetime.today() - df['purchase_date']).dt.days)//30
    df['month_diff'] = df['month_diff'].astype(int)
    df['month_diff'] += df['month_lag']

    print('*'*30,'Part1, whole data','*'*30)
    cardid_features = pd.DataFrame()
    cardid_features['card_id'] = df['card_id'].unique()
    print( '*' * 30, 'Traditional Features', '*' * 30)
    ht_card_id_gp = df.groupby('card_id')
    cardid_features['card_id_cnt'] = ht_card_id_gp['authorized_flag'].count().values

    if  prefix == 'hist_cardf_':
        cardid_features['card_id_isau_mean'] = ht_card_id_gp['authorized_flag'].mean().values
        cardid_features['card_id_isau_sum'] = ht_card_id_gp['authorized_flag'].sum().values

    cardid_features['month_diff_mean']   = ht_card_id_gp['month_diff'].mean().values
    cardid_features['month_diff_median'] = ht_card_id_gp['month_diff'].median().values

    if prefix == 'hist_cardf_':
        cardid_features['reference_day']           =  ht_card_id_gp['purchase_date_floorday'].max().v
        cardid_features['first_day']               =  ht_card_id_gp['purchase_date_floorday'].min().value
        cardid_features['activation_day']          =  ht_card_id_gp['first_active_month'].max().values

        # first to activation day
        cardid_features['first_to_activation_day']  =  (cardid_features['first_day'] - cardid_features['
        # activation to reference day
        cardid_features['activation_to_reference_day']  =  (cardid_features['reference_day'] - cardic
        # first to last day
        cardid_features['first_to_reference_day']  =  (cardid_features['reference_day'] - cardid_feat
        # reference day to now
        cardid_features['reference_day_to_now']  =  (datetime.date(2018, month, 1) - cardid_featu
        # first day to now
        cardid_features['first_day_to_now']  =  (datetime.date(2018, month, 1) - cardid_features['fi

        print('card_id(month_lag, min to reference day):min')
        cardid_features['card_id_month_lag_min'] = ht_card_id_gp['month_lag'].agg('min').values
        # is_purchase_before_activation,first_to_reference_day_divide_activation_to_reference_day
        cardid_features['is_purchase_before_activation'] = cardid_features['first_to_activation_day'
        cardid_features['is_purchase_before_activation'] = cardid_features['is_purchase_before_act
        cardid_features['first_to_reference_day_divide_activation_to_reference_day'] = cardid_featu
        cardid_features['days_per_count'] = cardid_features['first_to_reference_day'].values / cardi

    if prefix == 'new_cardf_':
        print(' Eight time features, ')
        cardid_features['reference_day']           =  ht_card_id_gp['reference_day'].last().values
        cardid_features['first_day']               =  ht_card_id_gp['purchase_date_floorday'].min().value
        cardid_features['last_day']              = ht_card_id_gp['purchase_date_floorday'].max().valu
        cardid_features['activation_day']         = ht_card_id_gp['first_active_month'].max().values
        # reference to first day
        cardid_features['reference_day_to_first_day']  = (cardid_features['first_day'] - cardid_featu
        # reference to last day
        cardid_features['reference_day_to_last_day']  = (cardid_features['last_day'] - cardid_featur
        # first to last day
        cardid_features['first_to_last_day']  = (cardid_features['last_day'] - cardid_features['first_da
        # activation to first day
        cardid_features['activation_to_first_day']  = (cardid_features['first_day'] - cardid_features['
        # activation to first day
        cardid_features['activation_to_last_day']  = (cardid_features['last_day'] - cardid_features['a
        # last day to now
        cardid_features['reference_day_to_now']  = (datetime.date(2018, month, 1) - cardid_featu
        # first day to now
        cardid_features['first_day_to_now']  = (datetime.date(2018, month, 1) - cardid_features['fi

        print('card_id(month_lag, min to reference day):min')
        cardid_features['card_id_month_lag_max'] = ht_card_id_gp['month_lag'].agg('max').values
        cardid_features['first_to_last_day_divide_reference_to_last_day'] = cardid_features['first_to
        cardid_features['days_per_count'] = cardid_features['first_to_last_day'].values / cardid_fea

    for f in ['reference_day', 'first_day', 'last_day', 'activation_day']:
        try:
            del cardid_features[f]
        except:
            print(f, '不存在！！！')

    print('card id(city_id,installments,merchant_category_id,.......):nunique, cnt/nunique')
    for col in tqdm_notebook(['category_1','category_2','category_3','state_id','city_id','installme
        cardid_features['card_id_%s_nunique'%col]            = ht_card_id_gp[col].nunique().values
        cardid_features['card_id_cnt_divide_%s_nunique'%col] = cardid_features['card_id_cnt'].val

    print('card_id(purchase_amount & degrade version ):mean,sum,std,median,quantile(10,25,75
    for col in tqdm_notebook(['installments','purchase_amount','purchase_amount_ddgd_98','pu
        if col =='purchase_amount':
            for opt in ['sum','mean','std','median','max','min']:
                cardid_features['card_id_' +col+ '_' + opt] = ht_card_id_gp[col].agg(opt).values

            cardid_features['card_id_' +col+ '_range'] = cardid_features['card_id_' +col+ '_max'].val
            percentiles = ht_card_id_gp[col].apply(lambda x:get_quantile(x,percentiles = [0.025, 0.2

            cardid_features[col + '_2.5_quantile']  = percentiles.map(lambda x:x[0]).values
            cardid_features[col + '_25_quantile'] = percentiles.map(lambda x:x[1]).values
            cardid_features[col + '_75_quantile'] = percentiles.map(lambda x:x[2]).values
            cardid_features[col + '_97.5_quantile'] = percentiles.map(lambda x:x[3]).values
            cardid_features['card_id_' +col+ '_range2'] = cardid_features[col+ '_97.5_quantile'].valu
            del cardid_features[col + '_2.5_quantile'],cardid_features[col + '_97.5_quantile']
            gc.collect()
        else:
            for opt in ['sum']:
                cardid_features['card_id_' +col+ '_' + opt] = ht_card_id_gp[col].agg(opt).values

    print( '*' * 30, 'Pivot Features', '*' * 30)
    print('Count  Pivot') #purchase_month_since_reference_day(可能和month_lag重复),百分比降分
    for pivot_col in tqdm_notebook(['category_1','category_2','category_3','month_lag','subsecto

        tmp     = df.groupby(['card_id',pivot_col])['merchant_id'].count().to_frame(pivot_col + '_co
        tmp.reset_index(inplace =True)

        tmp_pivot = pd.pivot_table(data=tmp,index = 'card_id',columns=pivot_col,values=pivot_
        tmp_pivot.columns = [tmp_pivot.columns.names[0] + '_cnt_pivot_'+ str(col) for col in tm
        tmp_pivot.reset_index(inplace = True)
        cardid_features = cardid_features.merge(tmp_pivot, on = 'card_id', how='left')
```

```
        if  pivot_col!='weekend' and  pivot_col!='installments':
            tmp          = df.groupby(['card_id',pivot_col])['purchase_date_floorday'].nunique().to_fr
            tmp1          = df.groupby(['card_id'])['purchase_date_floorday'].nunique().to_frame('pur
            tmp.reset_index(inplace =True)
            tmp1.reset_index(inplace =True)
            tmp  = tmp.merge(tmp1, on ='card_id', how='left')
            tmp[pivot_col + '_day_nunique_pct'] = tmp[pivot_col + '_purchase_date_floorday_nuniq

            tmp_pivot = pd.pivot_table(data=tmp,index = 'card_id',columns=pivot_col,values=pivo
            tmp_pivot.columns = [tmp_pivot.columns.names[0] + '_day_nunique_pct_'+ str(col) for
            tmp_pivot.reset_index(inplace = True)
            cardid_features = cardid_features.merge(tmp_pivot, on = 'card_id', how='left')

    if prefix == 'new_cardf_':
    ######## 在卡未激活之前就有过消费的记录  ##############
        print('*'*30,'Part2，  data with time less than activation day','*'*30)
        df_part = df.loc[df.purchase_date < df.first_active_month]

        cardid_features_part = pd.DataFrame()
        cardid_features_part['card_id'] = df_part['card_id'].unique()
        ht_card_id_part_gp = df_part.groupby('card_id')
        cardid_features_part['card_id_part_cnt'] = ht_card_id_part_gp['authorized_flag'].count().va

        print('card_id(purchase_amount): sum')
        for col in tqdm_notebook(['purchase_amount']):
            for opt in ['sum','mean']:
                cardid_features_part['card_id_part_' +col+ '_' + opt] = ht_card_id_part_gp[col].agg(o

        cardid_features = cardid_features.merge(cardid_features_part, on ='card_id', how='left')
        cardid_features['card_id_part_purchase_amount_sum_percent'] = cardid_features['card_id_

    cardid_features = reduce_mem_usage(cardid_features)

    new_col_names = []
    for col in cardid_features.columns:
        if col == 'card_id':
            new_col_names.append(col)
        else:
            new_col_names.append(prefix + col)
    cardid_features.columns = new_col_names

    return cardid_features
print('auth...')
authorized_transactions = historical_transactions.loc[historical_transactions['authorized_flag'] =
auth_cardf_tran = get_cardf_tran(authorized_transactions, 3, prefix='auth_cardf_')
print('hist...')
hist_cardf_tran = get_cardf_tran(historical_transactions, 3, prefix='hist_cardf_')
print('new...')
reference_days = historical_transactions.groupby('card_id')['purchase_date'].last().to_frame('ref
reference_days.reset_index(inplace = True)
new_transactions = new_transactions.merge(reference_days, on ='card_id', how='left')
new_cardf_tran  = get_cardf_tran(new_transactions, 5, prefix='new_cardf_')
```

然后，需要进一步考虑最近两个月的用户行为特征：

In [ ]:
```
%%time
def get_cardf_tran_last2(df_, month = 3, prefix = 'last2_'):

    df = df_.loc[df_.month_lag >= -2].copy()
    print('*'*30,'Part1, whole data','*'*30)
    cardid_features = pd.DataFrame()
    cardid_features['card_id'] = df['card_id'].unique()
```

```
df['month_diff'] = ((datetime.datetime.today() - df['purchase_date']).dt.days)//30
df['month_diff'] = df['month_diff'].astype(int)
df['month_diff'] += df['month_lag']

print( '*' * 30, 'Traditional Features', '*' * 30)
ht_card_id_gp = df.groupby('card_id')
print(' card id : count')
cardid_features['card_id_cnt'] = ht_card_id_gp['authorized_flag'].count().values

cardid_features['card_id_isau_mean'] = ht_card_id_gp['authorized_flag'].mean().values
cardid_features['card_id_isau_sum']  = ht_card_id_gp['authorized_flag'].sum().values

cardid_features['month_diff_mean']   = ht_card_id_gp['month_diff'].mean().values

print('card id(city_id,installments,merchant_category_id,.......):nunique, cnt/nunique')
for col in tqdm_notebook(['state_id','city_id','installments','merchant_id', 'merchant_category
    cardid_features['card_id_%s_nunique'%col] = ht_card_id_gp[col].nunique().values
    cardid_features['card_id_cnt_divide_%s_nunique'%col] = cardid_features['card_id_cnt'].valu

for col in tqdm_notebook(['purchase_amount','purchase_amount_ddgd_98','purchase_amou
    if col =='purchase_amount':
        for opt in ['sum','mean','std','median']:
            cardid_features['card_id_' +col+ '_' + opt] = ht_card_id_gp[col].agg(opt).values
    else:
        for opt in ['sum']:
            cardid_features['card_id_' +col+ '_' + opt] = ht_card_id_gp[col].agg(opt).values

print( '*' * 30, 'Pivot Features', '*' * 30)
print('Count  Pivot') #purchase_month_since_reference_day(可能和month_lag重复),百分比降

for pivot_col in tqdm_notebook(['category_1','category_2','category_3','month_lag','subsect

    tmp     = df.groupby(['card_id',pivot_col])['merchant_id'].count().to_frame(pivot_col + '_co
    tmp.reset_index(inplace =True)

    tmp_pivot = pd.pivot_table(data=tmp,index = 'card_id',columns=pivot_col,values=pivot_
    tmp_pivot.columns = [tmp_pivot.columns.names[0] + '_cnt_pivot_'+ str(col) for col in tm
    tmp_pivot.reset_index(inplace = True)
    cardid_features = cardid_features.merge(tmp_pivot, on = 'card_id', how='left')

    if  pivot_col!='weekend' and  pivot_col!='installments':
        tmp           = df.groupby(['card_id',pivot_col])['purchase_date_floorday'].nunique().to_fr
        tmp1           = df.groupby(['card_id'])['purchase_date_floorday'].nunique().to_frame('pur
        tmp.reset_index(inplace =True)
        tmp1.reset_index(inplace =True)
        tmp  = tmp.merge(tmp1, on ='card_id', how='left')
        tmp[pivot_col + '_day_nunique_pct'] = tmp[pivot_col + '_purchase_date_floorday_nuniq

        tmp_pivot = pd.pivot_table(data=tmp,index = 'card_id',columns=pivot_col,values=pivo
        tmp_pivot.columns = [tmp_pivot.columns.names[0] + '_day_nunique_pct_'+ str(col) fo
        tmp_pivot.reset_index(inplace = True)
        cardid_features = cardid_features.merge(tmp_pivot, on = 'card_id', how='left')

cardid_features = reduce_mem_usage(cardid_features)

new_col_names = []
for col in cardid_features.columns:
    if col == 'card_id':
        new_col_names.append(col)
    else:
        new_col_names.append(prefix + col)
cardid_features.columns = new_col_names

return cardid_features
```

```
hist_cardf_tran_last2 = get_cardf_tran_last2(historical_transactions, month = 3, prefix = 'hist_la
```

以及进一步进行二阶交叉特征衍生：

In [ ]:
```
%%time
def successive_aggregates(df_, prefix = 'levelAB_'):
    df = df_.copy()
    cardid_features = pd.DataFrame()
    cardid_features['card_id'] = df['card_id'].unique()

    level12_nunique = [('month_lag','state_id'),('month_lag','city_id'),('month_lag','subsector_id'),
                       ('subsector_id','merchant_category_id'),('subsector_id','merchant_id'),('subsector_
                       ('merchant_category_id', 'merchant_id'),('merchant_category_id','purchase_date_
                       ('purchase_date_floorday', 'merchant_id'),('purchase_date_floorday','merchant_ca
    for col_level1,col_level2 in tqdm_notebook(level12_nunique):

        level1  = df.groupby(['card_id',col_level1])[col_level2].nunique().to_frame(col_level2 + '_nu
        level1.reset_index(inplace =True)

        level2 = level1.groupby('card_id')[col_level2 + '_nunique'].agg(['mean', 'max', 'std'])
        level2 = pd.DataFrame(level2)
        level2.columns = [col_level1 + '_' + col_level2 + '_nunique_' + col for col in level2.columr
        level2.reset_index(inplace = True)

        cardid_features = cardid_features.merge(level2, on='card_id', how='left')

    level12_count = ['month_lag','state_id','city_id','subsector_id','merchant_category_id','mercha
    for col_level in tqdm_notebook(level12_count):

        level1  = df.groupby(['card_id',col_level])['merchant_id'].count().to_frame(col_level + '_cou
        level1.reset_index(inplace =True)

        level2 = level1.groupby('card_id')[col_level + '_count'].agg(['mean', 'max', 'std'])
        level2 = pd.DataFrame(level2)
        level2.columns = [col_level + '_count_' + col for col in level2.columns.values]
        level2.reset_index(inplace = True)

        cardid_features = cardid_features.merge(level2, on='card_id', how='left')

    level12_meansum = [('month_lag','purchase_amount'),('state_id','purchase_amount'),('city_id
                       ('merchant_category_id','purchase_amount'),('merchant_id','purchase_amount'),(
    for col_level1,col_level2 in tqdm_notebook(level12_meansum):

        level1  = df.groupby(['card_id',col_level1])[col_level2].sum().to_frame(col_level2 + '_sum')
        level1.reset_index(inplace =True)

        level2 = level1.groupby('card_id')[col_level2 + '_sum'].agg(['mean', 'max', 'std'])
        level2 = pd.DataFrame(level2)
        level2.columns = [col_level1 + '_' + col_level2 + '_sum_' + col for col in level2.columns.va
        level2.reset_index(inplace = True)

        cardid_features = cardid_features.merge(level2, on='card_id', how='left')

    cardid_features = reduce_mem_usage(cardid_features)

    new_col_names = []
    for col in cardid_features.columns:
        if col == 'card_id':
            new_col_names.append(col)
        else:
            new_col_names.append(prefix + col)
    cardid_features.columns = new_col_names
    return cardid_features

print('hist...')
hist_levelAB = successive_aggregates(historical_transactions, prefix = 'hist_levelAB_')
```

接下来，将上述衍生特征进行合并：

In [ ]:
```
%%time
print(train.shape)
print(test.shape)
## 合并到训练集和测试集
print('#_____基础统计特征')
train = pd.merge(train, auth_base_stat, on='card_id', how='left')
test  = pd.merge(test,  auth_base_stat, on='card_id', how='left')
train = pd.merge(train, hist_base_stat, on='card_id', how='left')
test  = pd.merge(test,  hist_base_stat, on='card_id', how='left')
train = pd.merge(train, new_base_stat , on='card_id', how='left')
test  = pd.merge(test,  new_base_stat , on='card_id', how='left')
print(train.shape)
print(test.shape)
print('#_____全局cardid特征')
train = pd.merge(train, auth_cardf_tran, on='card_id', how='left')
test  = pd.merge(test,  auth_cardf_tran, on='card_id', how='left')
train = pd.merge(train, hist_cardf_tran, on='card_id', how='left')
test  = pd.merge(test,  hist_cardf_tran, on='card_id', how='left')
train = pd.merge(train, new_cardf_tran , on='card_id', how='left')
test  = pd.merge(test,  new_cardf_tran , on='card_id', how='left')
print(train.shape)
print(test.shape)
print('#_____最近两月cardid特征')
train = pd.merge(train, hist_cardf_tran_last2, on='card_id', how='left')
test  = pd.merge(test,  hist_cardf_tran_last2, on='card_id', how='left')
print(train.shape)
print(test.shape)
print('#_____补充二阶特征')
train = pd.merge(train, hist_levelAB, on='card_id', how='left')
test  = pd.merge(test,  hist_levelAB, on='card_id', how='left')
print(train.shape)
print(test.shape)
```

并在此基础上补充部分简单四折运算后的衍生特征：

In [ ]:
```
%%time
train['outliers'] = 0
train.loc[train['target'] < -30, 'outliers'] = 1
train['outliers'].value_counts()
for f in ['feature_1','feature_2','feature_3']:
    colname = f+'_outliers_mean'
    order_label = train.groupby([f])['outliers'].mean()
    for df in [train, test]:
        df[colname] = df[f].map(order_label)

for df in [train, test]:

    df['days_feature1'] = df['elapsed_time'] * df['feature_1']
    df['days_feature2'] = df['elapsed_time'] * df['feature_2']
    df['days_feature3'] = df['elapsed_time'] * df['feature_3']

    df['days_feature1_ratio'] = df['feature_1'] / df['elapsed_time']
    df['days_feature2_ratio'] = df['feature_2'] / df['elapsed_time']
    df['days_feature3_ratio'] = df['feature_3'] / df['elapsed_time']

    df['feature_sum'] = df['feature_1'] + df['feature_2'] + df['feature_3']
    df['feature_mean'] = df['feature_sum']/3
    df['feature_max'] = df[['feature_1', 'feature_2', 'feature_3']].max(axis=1)
    df['feature_min'] = df[['feature_1', 'feature_2', 'feature_3']].min(axis=1)
    df['feature_var'] = df[['feature_1', 'feature_2', 'feature_3']].std(axis=1)

    df['card_id_total'] = df['hist_card_id_size']+df['new_card_id_size']
    df['card_id_cnt_total'] = df['hist_card_id_count']+df['new_card_id_count']
    df['card_id_cnt_ratio'] = df['new_card_id_count']/df['hist_card_id_count']
    df['purchase_amount_total'] = df['hist_cardf_card_id_purchase_amount_sum']+df['new_cardf
    df['purchase_amount_ratio'] = df['new_cardf_card_id_purchase_amount_sum']/df['hist_cardf_
    df['month_diff_ratio'] = df['new_cardf_month_diff_mean']/df['hist_cardf_month_diff_mean']
    df['installments_total'] = df['new_cardf_card_id_installments_sum']+df['auth_cardf_card_id_ir
    df['installments_ratio'] = df['new_cardf_card_id_installments_sum']/df['auth_cardf_card_id_in:
    df['price_total'] = df['purchase_amount_total']/df['installments_total']
    df['new_CLV'] = df['new_card_id_count'] * df['new_cardf_card_id_purchase_amount_sum'] / c
    df['hist_CLV'] = df['hist_card_id_count'] * df['hist_cardf_card_id_purchase_amount_sum'] / df[
    df['CLV_ratio'] = df['new_CLV'] / df['hist_CLV']
```

3.特征筛选

在创建完全部特征后即可进行特征筛选了。此处我们考虑手动进行特征筛选，排除部分过于稀疏的特征后即可将数据保存在本地：

In [ ]:
```
%%time
del_cols = []
for col in train.columns:
    if 'subsector_id_cnt_' in col and 'new_cardf':
        del_cols.append(col)
del_cols1 = []
for col in train.columns:
    if 'subsector_id_cnt_' in col and 'hist_last2_' in col:
        del_cols1.append(col)
del_cols2 = []
for col in train.columns:
    if 'subsector_id_cnt_' in col and 'auth_cardf' in col:
        del_cols2.append(col)
del_cols3 = []
for col in train.columns:
    if 'merchant_category_id_month_lag_nunique_' in col and '_pivot_supp' in col:
        del_cols3.append(col)
    if 'city_id' in col and '_pivot_supp' in col:
        del_cols3.append(col)
    if 'month_diff' in col and 'hist_last2_' in col:
        del_cols3.append(col)
    if 'month_diff_std' in col or 'month_diff_gap' in col:
        del_cols3.append(col)
fea_cols = [col for col in train.columns if train[col].dtypes!='object' and train[col].dtypes != '<
          and col not in del_cols and col not in del_cols1 and col not in del_cols2 and col!='targ
print('删除前:',train.shape[1])
print('删除后:',len(fea_cols))

train = train[fea_cols+['target']]
fea_cols.remove('outliers')
test = test[fea_cols]

train.to_csv('./data/all_train_features.csv',index=False)
test.to_csv('./data/all_test_features.csv',index=False)
```

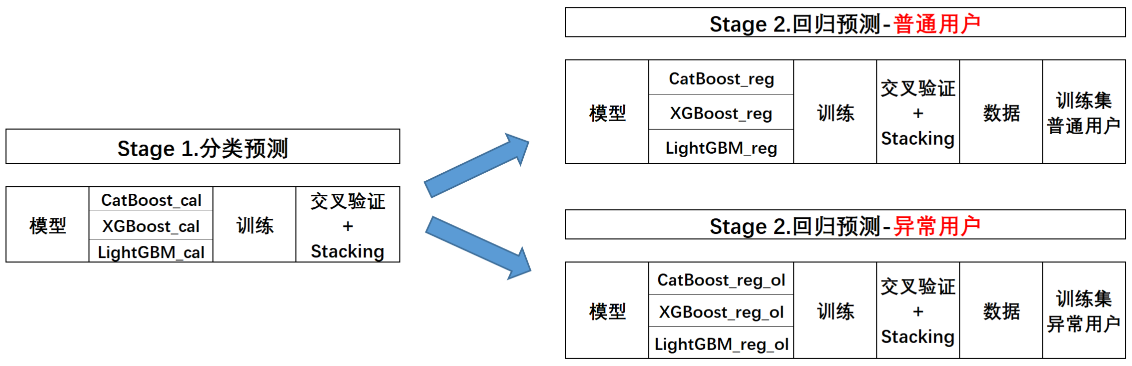
实际执行过程中，可以按照如下方式进行读取：

In [ ]:
```
%%time
## load all features
train = pd.read_csv('./data/all_train_features.csv')
```

```python
test  = pd.read_csv('./data/all_test_features.csv')

inf_cols = ['new_cardf_card_id_cnt_divide_installments_nunique', 'hist_last2_card_id_cnt_divide_
train[inf_cols] = train[inf_cols].replace(np.inf, train[inf_cols].replace(np.inf, -99).max().max())
ntrain[inf_cols] = ntrain[inf_cols].replace(np.inf, ntrain[inf_cols].replace(np.inf, -99).max().max()
test[inf_cols] = test[inf_cols].replace(np.inf, test[inf_cols].replace(np.inf, -99).max().max())

# ## load sparse
# train_tags = sparse.load_npz('train_tags.npz')
# test_tags  = sparse.load_npz('test_tags.npz')

## 获取非异常值的index
normal_index = train[train['outliers']==0].index.tolist()
## without outliers
ntrain = train[train['outliers'] == 0]

target        = train['target'].values
ntarget        = ntrain['target'].values
target_binary = train['outliers'].values
###
y_train        = target
y_ntrain        = ntarget
y_train_binary = target_binary

print('train:',train.shape)
print('ntrain:',ntrain.shape)
```

二、两阶段建模优化

LightGBM+XGBoost+CatBoost两阶段建模



为了方便更快速的调用三种不同的模型，并且同时要求能够完成分类和回归预测，此处通过定义一个函数来完成所有模型的训练过程。

In [ ]:
```python
def train_model(X, X_test, y, params, folds, model_type='lgb', eval_type='regression'):
    oof = np.zeros(X.shape[0])
    predictions = np.zeros(X_test.shape[0])
    scores = []
    for fold_n, (trn_idx, val_idx) in enumerate(folds.split(X, y)):
        print('Fold', fold_n, 'started at', time.ctime())

        if model_type == 'lgb':
            trn_data = lgb.Dataset(X[trn_idx], y[trn_idx])
            val_data = lgb.Dataset(X[val_idx], y[val_idx])
            clf = lgb.train(params, trn_data, num_boost_round=20000,
                            valid_sets=[trn_data, val_data],
                            verbose_eval=100, early_stopping_rounds=300)
            oof[val_idx] = clf.predict(X[val_idx], num_iteration=clf.best_iteration)
            predictions += clf.predict(X_test, num_iteration=clf.best_iteration) / folds.n_splits

        if model_type == 'xgb':
            trn_data = xgb.DMatrix(X[trn_idx], y[trn_idx])
            val_data = xgb.DMatrix(X[val_idx], y[val_idx])
            watchlist = [(trn_data, 'train'), (val_data, 'valid_data')]
            clf = xgb.train(dtrain=trn_data, num_boost_round=20000,
                            evals=watchlist, early_stopping_rounds=200,
                            verbose_eval=100, params=params)
            oof[val_idx] = clf.predict(xgb.DMatrix(X[val_idx]), ntree_limit=clf.best_ntree_limit)
            predictions += clf.predict(xgb.DMatrix(X_test), ntree_limit=clf.best_ntree_limit) / folds.

        if (model_type == 'cat') and (eval_type == 'regression'):
            clf = CatBoostRegressor(iterations=20000, eval_metric='RMSE', **params)
            clf.fit(X[trn_idx], y[trn_idx],
                    eval_set=(X[val_idx], y[val_idx]),
                    cat_features=[], use_best_model=True, verbose=100)
            oof[val_idx] = clf.predict(X[val_idx])
            predictions += clf.predict(X_test) / folds.n_splits

        if (model_type == 'cat') and (eval_type == 'binary'):
            clf = CatBoostClassifier(iterations=20000, eval_metric='Logloss', **params)
            clf.fit(X[trn_idx], y[trn_idx],
                    eval_set=(X[val_idx], y[val_idx]),
                    cat_features=[], use_best_model=True, verbose=100)
            oof[val_idx] = clf.predict_proba(X[val_idx])[:,1]
            predictions += clf.predict_proba(X_test)[:,1] / folds.n_splits
        print(predictions)
        if eval_type == 'regression':
            scores.append(mean_squared_error(oof[val_idx], y[val_idx])**0.5)
        if eval_type == 'binary':
            scores.append(log_loss(y[val_idx], oof[val_idx]))

    print('CV mean score: {0:.4f}, std: {1:.4f}.'.format(np.mean(scores), np.std(scores)))

    return oof, predictions, scores
```

- LightGBM模型训练

  接下来即可进行LGB模型训练：

In [ ]:
```
#### lgb
lgb_params = {'num_leaves': 63,
             'min_data_in_leaf': 32,
             'objective':'regression',
             'max_depth': -1,
             'learning_rate': 0.01,
             "min_child_samples": 20,
             "boosting": "gbdt",
             "feature_fraction": 0.9,
             "bagging_freq": 1,
             "bagging_fraction": 0.9 ,
             "bagging_seed": 11,
             "metric": 'rmse',
             "lambda_l1": 0.1,
             "verbosity": -1}
folds = KFold(n_splits=5, shuffle=True, random_state=4096)
X_ntrain = ntrain[fea_cols].values
X_train  = train[fea_cols].values
X_test   = test[fea_cols].values
print('='*10,'回归模型','='*10)
oof_lgb , predictions_lgb , scores_lgb  = train_model(X_train , X_test, y_train, params=lgb_para
```

In [ ]:
```
print('='*10,'without outliers 回归模型','='*10)
oof_nlgb, predictions_nlgb, scores_nlgb = train_model(X_ntrain, X_test, y_ntrain, params=lgb_p
```

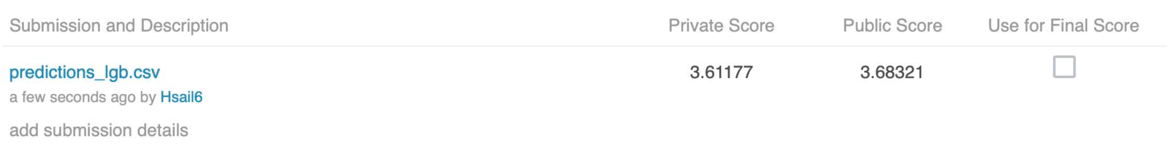
In [ ]:
```
print('='*10,'分类模型','='*10)
lgb_params['objective'] = 'binary'
lgb_params['metric']    = 'binary_logloss'
oof_blgb, predictions_blgb, scores_blgb = train_model(X_train , X_test, y_train_binary, params=
```

然后将所有预测结果进行保存，包括一个分类模型、以及两个回归模型：

In [ ]:
```
sub_df = pd.read_csv('data/sample_submission.csv')
sub_df["target"] = predictions_lgb
sub_df.to_csv('predictions_lgb.csv', index=False)
```

In [ ]:
```
oof_lgb  = pd.DataFrame(oof_lgb)
oof_nlgb = pd.DataFrame(oof_nlgb)
oof_blgb = pd.DataFrame(oof_blgb)

predictions_lgb  = pd.DataFrame(predictions_lgb)
predictions_nlgb = pd.DataFrame(predictions_nlgb)
predictions_blgb = pd.DataFrame(predictions_blgb)

oof_lgb.to_csv('./result/oof_lgb.csv',header=None,index=False)
oof_blgb.to_csv('./result/oof_blgb.csv',header=None,index=False)
oof_nlgb.to_csv('./result/oof_nlgb.csv',header=None,index=False)

predictions_lgb.to_csv('./result/predictions_lgb.csv',header=None,index=False)
predictions_nlgb.to_csv('./result/predictions_nlgb.csv',header=None,index=False)
predictions_blgb.to_csv('./result/predictions_blgb.csv',header=None,index=False)
```

和上一节类似，接下来我们考虑进行结果提交，查看经过特征优化后单模型的建模效果能否得到改善：



Submission and Description	Private Score	Public Score	Use for Final Score
predictions_lgb.csv a few seconds ago by Hsail6 add submission details	3.61177	3.68321	☐

模型	Private Score	Public Score
randomforest	3.65455	3.74969
randomforest+validation	3.65173	3.74954
LightGBM	3.69723	3.80436
LightGBM+validation	3.64403	3.73875
XGBoost	3.62832	3.72358
Voting_avr	3.63650	3.73251
Voting_wei	3.633307	3.72877
Stacking	3.62798	3.72055
第二阶段	特征优化	两阶段建模
LightGBM	3.61177	3.68321

- xgboost模型训练

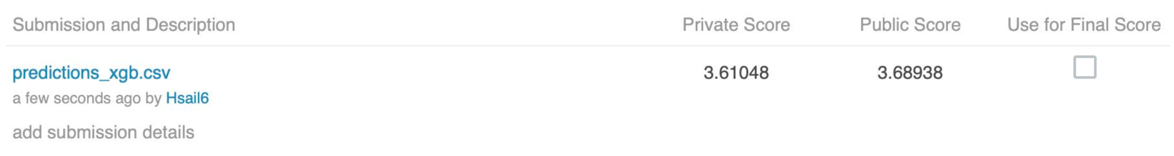
接下来进一步进行XGB模型训练：

In [ ]:
```
#### xgb
xgb_params = {'eta':0.05, 'max_leaves':47, 'max_depth':10, 'subsample':0.8, 'colsample_bytree':0
          'min_child_weight':40, 'max_bin':128, 'reg_alpha':2.0, 'reg_lambda':2.0,
          'objective':'reg:linear', 'eval_metric':'rmse', 'silent': True, 'nthread':4}
folds = KFold(n_splits=5, shuffle=True, random_state=2018)
print('='*10,'回归模型','='*10)
oof_xgb , predictions_xgb , scores_xgb  = train_model(X_train , X_test, y_train , params=xgb_pa
print('='*10,'without outliers 回归模型','='*10)
oof_nxgb, predictions_nxgb, scores_nxgb = train_model(X_ntrain, X_test, y_ntrain, params=xgb
print('='*10,'分类模型','='*10)
xgb_params['objective'] = 'binary:logistic'
xgb_params['metric']    = 'binary_logloss'
oof_bxgb, predictions_bxgb, scores_bxgb = train_model(X_train , X_test, y_train_binary, params
```

然后进行结果保存：

In [ ]:
```
sub_df = pd.read_csv('data/sample_submission.csv')
sub_df["target"] = predictions_xgb
sub_df.to_csv('predictions_xgb.csv', index=False)

oof_xgb  = pd.DataFrame(oof_xgb)
oof_nxgb = pd.DataFrame(oof_nxgb)
oof_bxgb = pd.DataFrame(oof_bxgb)

predictions_xgb  = pd.DataFrame(predictions_xgb)
predictions_nxgb = pd.DataFrame(predictions_nxgb)
predictions_bxgb = pd.DataFrame(predictions_bxgb)

oof_xgb.to_csv('./result/oof_xgb.csv',header=None,index=False)
oof_bxgb.to_csv('./result/oof_bxgb.csv',header=None,index=False)
oof_nxgb.to_csv('./result/oof_nxgb.csv',header=None,index=False)

predictions_xgb.to_csv('./result/predictions_xgb.csv',header=None,index=False)
predictions_nxgb.to_csv('./result/predictions_nxgb.csv',header=None,index=False)
predictions_bxgb.to_csv('./result/predictions_bxgb.csv',header=None,index=False)
```

接下来提交单模型预测结果：



Submission and Description	Private Score	Public Score	Use for Final Score
predictions_xgb.csv a few seconds ago by Hsail6 add submission details	3.61048	3.68938	☐

模型	Private Score	Public Score
randomforest	3.65455	3.74969
randomforest+validation	3.65173	3.74954
LightGBM	3.69723	3.80436
LightGBM+validation	3.64403	3.73875
XGBoost	3.62832	3.72358
Voting_avr	3.63650	3.73251
Voting_wei	3.633307	3.72877
Stacking	3.62798	3.72055
第二阶段	特征优化	两阶段建模

模型	Private Score	Public Score
LightGBM	3.61177	3.68321
XGBoost	3.61048	3.68938

- CatBoost

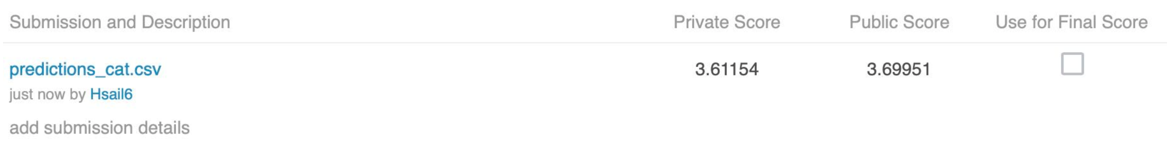
接下来进行CatBoost模型训练

In [ ]:
```
#### cat
cat_params = {'learning_rate': 0.05, 'depth': 9, 'l2_leaf_reg': 10, 'bootstrap_type': 'Bernoulli',
           'od_type': 'Iter', 'od_wait': 50, 'random_seed': 11, 'allow_writing_files': False}
folds = KFold(n_splits=5, shuffle=True, random_state=18)
print('='*10,'回归模型','='*10)
oof_cat , predictions_cat , scores_cat  = train_model(X_train , X_test, y_train , params=cat_parar
print('='*10,'without outliers 回归模型','='*10)
oof_ncat, predictions_ncat, scores_ncat = train_model(X_ntrain, X_test, y_ntrain, params=cat_pa
print('='*10,'分类模型','='*10)
oof_bcat, predictions_bcat, scores_bcat = train_model(X_train , X_test, y_train_binary, params=
```

同时保存模型结果：

In [ ]:
```
sub_df = pd.read_csv('data/sample_submission.csv')
sub_df["target"] = predictions_cat
sub_df.to_csv('predictions_cat.csv', index=False)

oof_cat  = pd.DataFrame(oof_cat)
oof_ncat = pd.DataFrame(oof_ncat)
oof_bcat = pd.DataFrame(oof_bcat)

predictions_cat  = pd.DataFrame(predictions_cat)
predictions_ncat = pd.DataFrame(predictions_ncat)
predictions_bcat = pd.DataFrame(predictions_bcat)

oof_cat.to_csv('./result/oof_cat.csv',header=None,index=False)
oof_bcat.to_csv('./result/oof_bcat.csv',header=None,index=False)
oof_ncat.to_csv('./result/oof_ncat.csv',header=None,index=False)

predictions_cat.to_csv('./result/predictions_cat.csv',header=None,index=False)
predictions_ncat.to_csv('./result/predictions_ncat.csv',header=None,index=False)
predictions_bcat.to_csv('./result/predictions_bcat.csv',header=None,index=False)
```

测试单模型建模效果：



Submission and Description	Private Score	Public Score	Use for Final Score
predictions_cat.csv just now by Hsail6 add submission details	3.61154	3.69951	☐

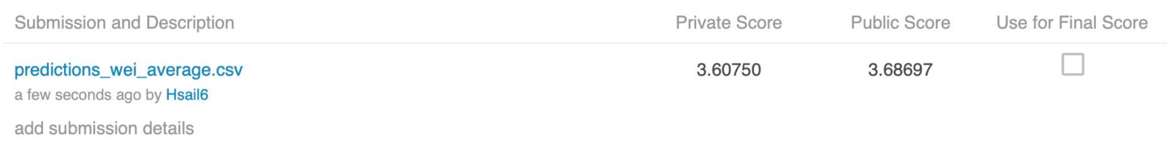
模型	Private Score	Public Score
randomforest	3.65455	3.74969
randomforest+validation	3.65173	3.74954
LightGBM	3.69723	3.80436
LightGBM+validation	3.64403	3.73875
XGBoost	3.62832	3.72358
Voting_avr	3.63650	3.73251

模型	Private Score	Public Score
Voting_wei	3.633307	3.72877
Stacking	3.62798	3.72055
第二阶段	特征优化	两阶段建模
LightGBM	3.61177	3.68321
XGBoost	3.61048	3.68938
CatBoost	3.61154	3.69951

3.融合阶段

- 加权融合

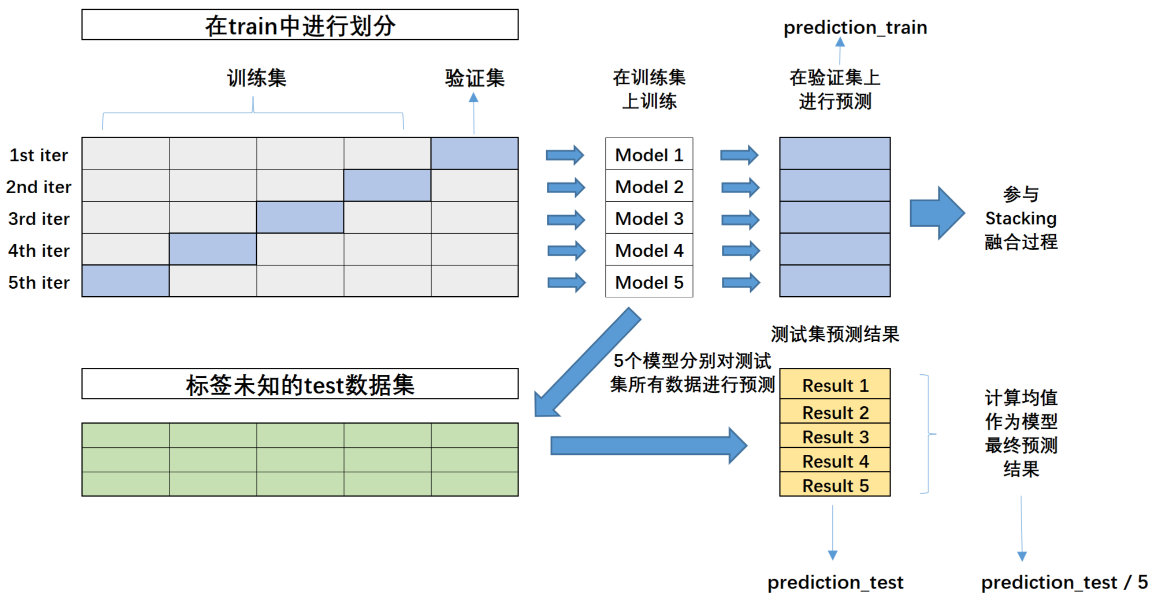
In [ ]:
```
sub_df = pd.read_csv('data/sample_submission.csv')
sub_df["target"] = (predictions_lgb + predictions_xgb.values.flatten() + predictions_cat.values.
sub_df.to_csv('predictions_wei_average.csv', index=False)
```



Submission and Description	Private Score	Public Score	Use for Final Score
predictions_wei_average.csv a few seconds ago by Hsail6 add submission details	3.60750	3.68697	☐

模型	Private Score	Public Score
randomforest	3.65455	3.74969
randomforest+validation	3.65173	3.74954
LightGBM	3.69723	3.80436
LightGBM+validation	3.64403	3.73875
XGBoost	3.62832	3.72358
Voting_avr	3.63650	3.73251
Voting_wei	3.633307	3.72877
Stacking	3.62798	3.72055
第二阶段	特征优化	两阶段建模
LightGBM	3.61177	3.68321
XGBoost	3.61048	3.68938
CatBoost	3.61154	3.69951
Voting	3.60640	3.68697

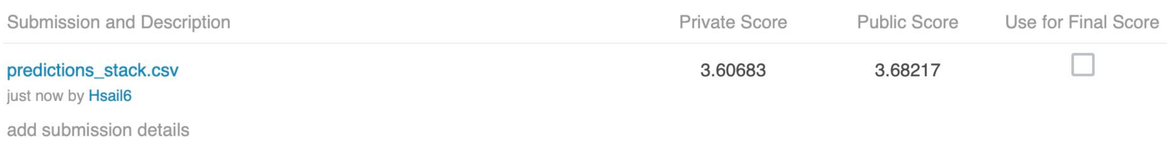
- Stacking融合



In [ ]:
```python
def stack_model(oof_1, oof_2, oof_3, predictions_1, predictions_2, predictions_3, y, eval_type='

    # Part 1.数据准备
    # 按行拼接列，拼接验证集所有预测结果
    # train_stack就是final model的训练数据
    train_stack = np.hstack([oof_1, oof_2, oof_3])
    # 按行拼接列，拼接测试集上所有预测结果
    # test_stack就是final model的测试数据
    test_stack = np.hstack([predictions_1, predictions_2, predictions_3])
    # 创建一个和验证集行数相同的全零数组
    oof = np.zeros(train_stack.shape[0])
    # 创建一个和测试集行数相同的全零数组
    predictions = np.zeros(test_stack.shape[0])

    # Part 2.多轮交叉验证
    from sklearn.model_selection import RepeatedKFold
    folds = RepeatedKFold(n_splits=5, n_repeats=2, random_state=2020)

    # fold_为折数，trn_idx为每一折训练集index，val_idx为每一折验证集index
    for fold_, (trn_idx, val_idx) in enumerate(folds.split(train_stack, y)):
        # 打印折数信息
        print("fold n°{}".format(fold_+1))
        # 训练集中划分为训练数据的特征和标签
        trn_data, trn_y = train_stack[trn_idx], y[trn_idx]
        # 训练集中划分为验证数据的特征和标签
        val_data, val_y = train_stack[val_idx], y[val_idx]
        # 开始训练时提示
        print("-" * 10 + "Stacking " + str(fold_+1) + "-" * 10)
        # 采用贝叶斯回归作为结果融合的模型（final model）
        clf = BayesianRidge()
        # 在训练数据上进行训练
        clf.fit(trn_data, trn_y)
        # 在验证数据上进行预测，并将结果记录在oof对应位置
        # oof[val_idx] = clf.predict(val_data)
        # 对测试集数据进行预测，每一轮预测结果占比额外的1/10
        predictions += clf.predict(test_stack) / (5 * 2)

    if eval_type == 'regression':
        print('mean: ',np.sqrt(mean_squared_error(y, oof)))
    if eval_type == 'binary':
        print('mean: ',log_loss(y, oof))
    # 返回测试集的预测结果
    return oof, predictions
```

In [ ]:
```
print('='*30)
oof_stack , predictions_stack  = stack_model(oof_lgb , oof_xgb , oof_cat , predictions_lgb , prec
print('='*30)
oof_nstack, predictions_nstack = stack_model(oof_nlgb, oof_nxgb, oof_ncat, predictions_nlgb,
print('='*30)
oof_bstack, predictions_bstack = stack_model(oof_blgb, oof_bxgb, oof_bcat, predictions_blgb,
```

In [ ]:
```
sub_df = pd.read_csv('data/sample_submission.csv')
sub_df["target"] = predictions_stack
sub_df.to_csv('predictions_stack.csv', index=False)
```

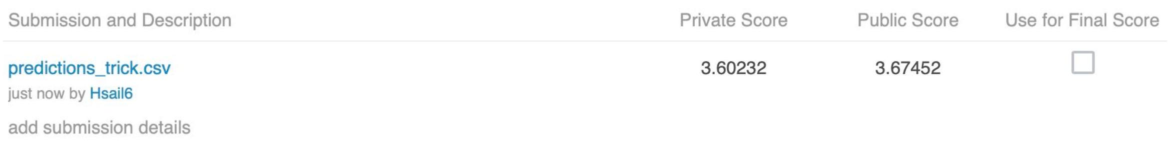


Submission and Description	Private Score	Public Score	Use for Final Score
predictions_stack.csv just now by Hsail6 add submission details	3.60683	3.68217	☐

模型	Private Score	Public Score
randomforest	3.65455	3.74969
randomforest+validation	3.65173	3.74954
LightGBM	3.69723	3.80436
LightGBM+validation	3.64403	3.73875
XGBoost	3.62832	3.72358
Voting_avr	3.63650	3.73251
Voting_wei	3.633307	3.72877
Stacking	3.62798	3.72055
第二阶段	特征优化	两阶段建模
LightGBM	3.61177	3.68321
XGBoost	3.61048	3.68938
CatBoost	3.61154	3.69951
Voting	3.60640	3.68697
Stacking	3.60683	3.68217

- Trick融合

In [ ]:
```
sub_df = pd.read_csv('data/sample_submission.csv')
sub_df["target"] = predictions_bstack*-33.219281 + (1-predictions_bstack)*predictions_nstack
sub_df.to_csv('predictions_trick.csv', index=False)
```



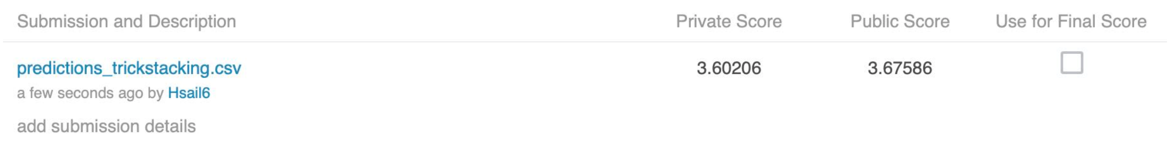
Submission and Description	Private Score	Public Score	Use for Final Score
predictions_trick.csv just now by Hsail6 add submission details	3.60232	3.67452	☐

模型	Private Score	Public Score
randomforest	3.65455	3.74969

模型	Private Score	Public Score
randomforest+validation	3.65173	3.74954
LightGBM	3.69723	3.80436
LightGBM+validation	3.64403	3.73875
XGBoost	3.62832	3.72358
Voting_avr	3.63650	3.73251
Voting_wei	3.633307	3.72877
Stacking	3.62798	3.72055
第二阶段	特征优化	两阶段建模
LightGBM	3.61177	3.68321
XGBoost	3.61048	3.68938
CatBoost	3.61154	3.69951
Voting	3.60640	3.68697
Stacking	3.60683	3.68217
Trick	3.60232	3.67452

- TrickStacking

In [ ]:
```
sub_df = pd.read_csv('data/sample_submission.csv')
sub_df["target"] = (predictions_bstack*-33.219281 + (1-predictions_bstack)*predictions_nstack
sub_df.to_csv('predictions_trick&stacking.csv', index=False)
```

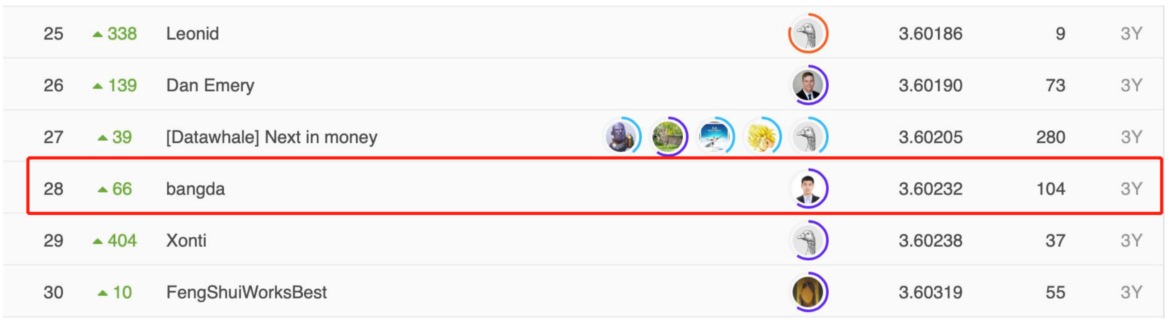


Submission and Description	Private Score	Public Score	Use for Final Score
predictions_trickstacking.csv a few seconds ago by Hsail6 add submission details	3.60206	3.67586	☐

模型	Private Score	Public Score
randomforest	3.65455	3.74969
randomforest+validation	3.65173	3.74954
LightGBM	3.69723	3.80436
LightGBM+validation	3.64403	3.73875
XGBoost	3.62832	3.72358
Voting_avr	3.63650	3.73251
Voting_wei	3.633307	3.72877
Stacking	3.62798	3.72055
第二阶段	特征优化	两阶段建模
LightGBM	3.61177	3.68321
XGBoost	3.61048	3.68938
CatBoost	3.61154	3.69951
Voting	3.60640	3.68697

模型	Private Score	Public Score
Stacking	3.60683	3.68217
Trick	3.60232	3.67452
Trick+Voting	3.60206	3.67580

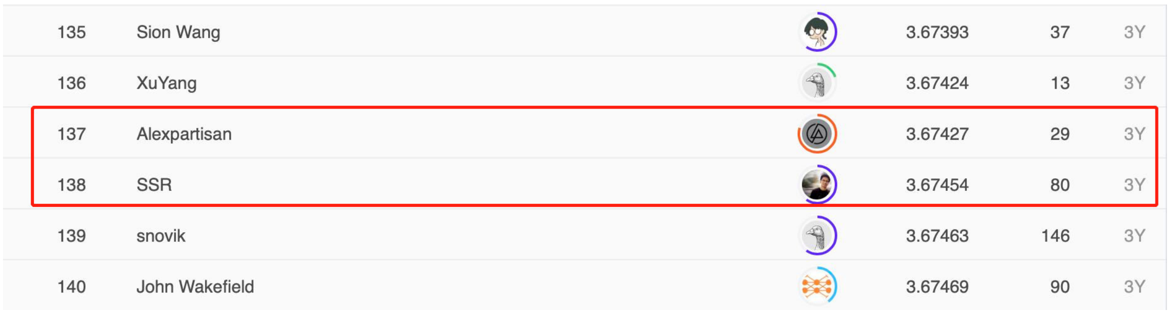
对比目前该比赛的榜单，我们能够看到最后的成绩排名，私榜排名如下：



25	▲338	Leonid	3.60186	9	3Y
26	▲139	Dan Emery	3.60190	73	3Y
27	▲39	[Datawhale] Next in money	3.60205	280	3Y
28	▲66	bangda	3.60232	104	3Y
29	▲404	Xonti	3.60238	37	3Y
30	▲10	FengShuiWorksBest	3.60319	55	3Y

In [ ]:
```
28 / 4127
```

排名约在前0.6%，而公榜排名如下：



135	Sion Wang	3.67393	37	3Y
136	XuYang	3.67424	13	3Y
137	Alexpartisan	3.67427	29	3Y
138	SSR	3.67454	80	3Y
139	snovik	3.67463	146	3Y
140	John Wakefield	3.67469	90	3Y

In [ ]:
```
137 / 4127
```

约在前3%左右。

- 后续优化策略

　　无论目前结果如何，永远可以有更进一步优化的地方。相信细心的小伙伴已经观察到，本节特征构建和建模策略和此前5天的策略略有差异，如果能融合两部分的方法，则能取得更好的效果。

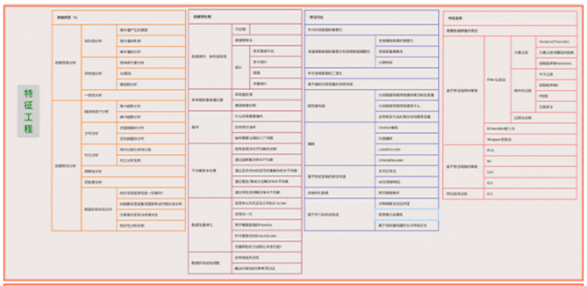
　　当然，如果对机器学习算法或者机器学习竞赛感兴趣，也欢迎各位小伙伴参与付费课程的学习！《机器学习实战训练营》（第二期）即将开课，十八周80+小时体系大课，由我和菜菜老师主讲，从零开始补充机器学习必备基础，深入讲解集成算法与特征工程方法，并包含四项大型Kaggle案例/企业级应用案例！对机器学习感兴趣的小伙伴扫描下方二维码查看课程详情哦！也可以添加客服VX：little_bird0229咨询课程，还有惊喜优惠券哦~

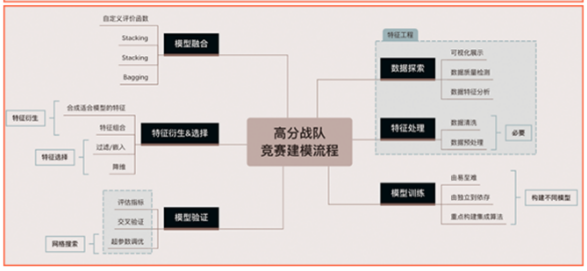
分享
机器学习实战
训练营第2期
18周 80+小时高品质课程
主讲老师
菜菜&九天
专栏

竞赛经验倾囊相授

拒绝纸上谈兵，快速提分有迹可循!

竞赛常用特征工程技巧





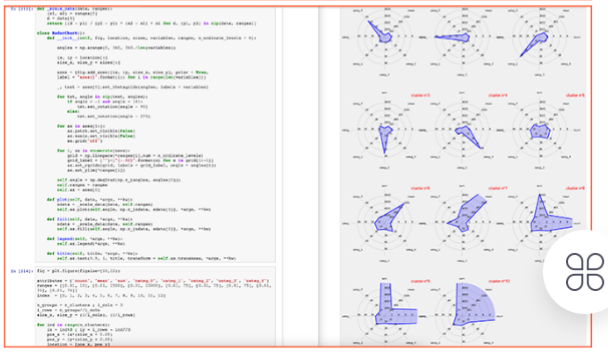
竞赛常用模型融合技巧

处理缺失值

自定义处理缺失值填充函数

处理缺失值

缺失值处理评估



☟扫码查看更多课程详情☟

本次公开课圆满结束！接下来有任何想听的内容，也欢迎在我的B站账号下留言~各位小伙伴，大家下次见！