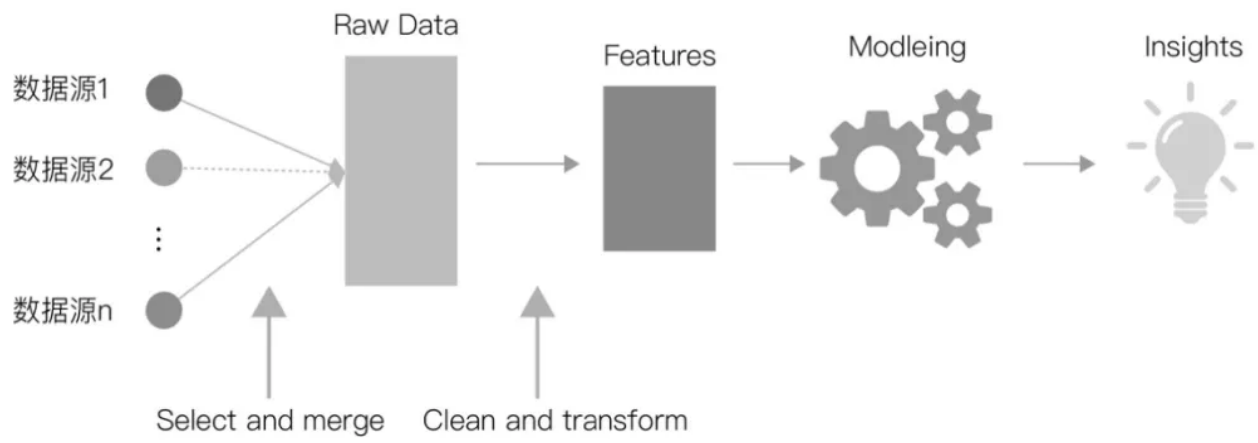
天池项目总结，特征工程了解一下!

阿里云天池 SAMshare 2020-12-22

来源：阿里云天池，案例：机器学习实践

业界广泛流传着这样一句话：“数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已”，由此可见特征工程在机器学习中的重要性，今天我们将通过《阿里云天池大赛赛题解析——机器学习篇》中的【天猫用户重复购买预测】案例来深入解析特征工程在实际商业场景中的应用。

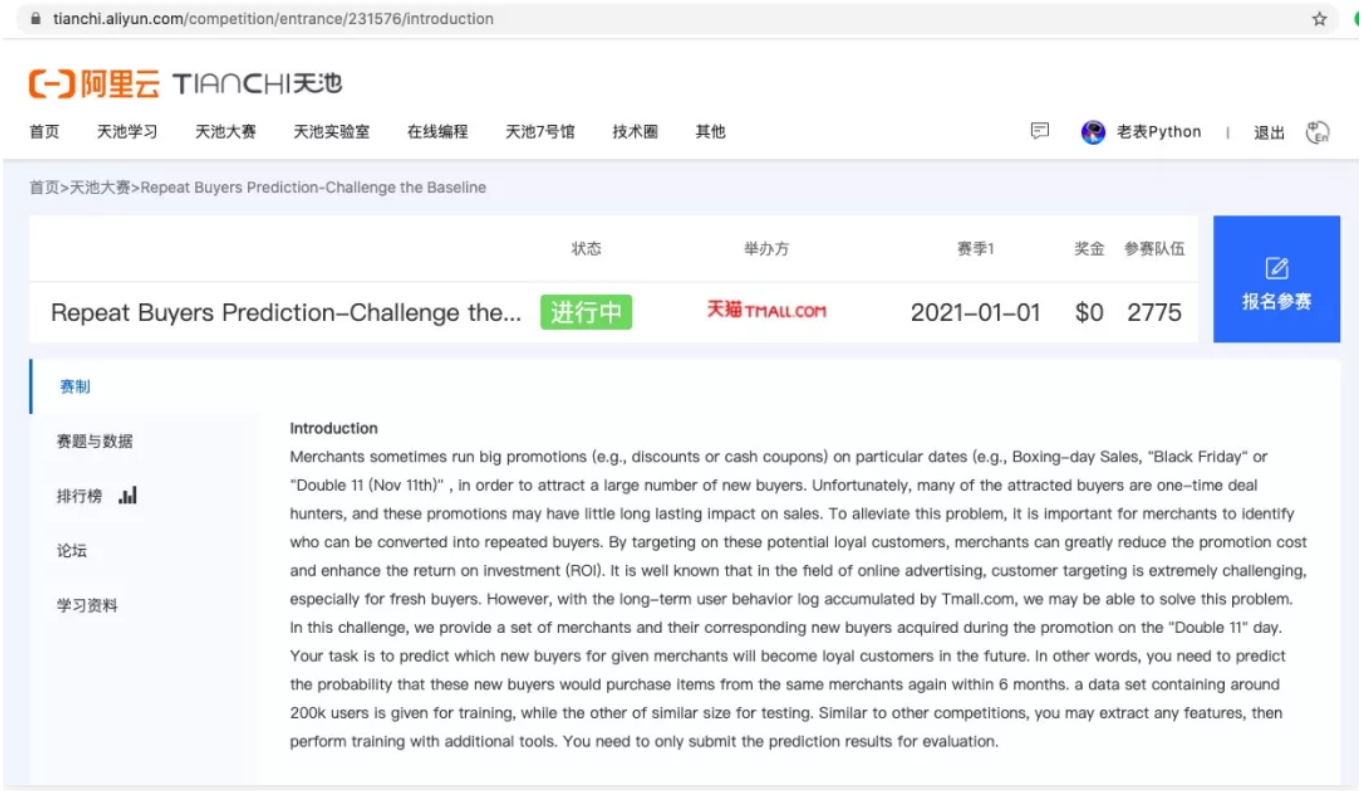


学习前须知

（1）本文特征工程讲解部分参考自图书《阿里云天池大赛赛题解析——机器学习篇》中的第二个赛题：天猫用户重复购买预测。

（2）本文相关数据可以在阿里云天池竞赛平台下载，数据地址：

https://tianchi.aliyun.com/competition/entrance/231576/information



一 数据集介绍

按照上面方法下载好数据集后，我们来看看具体数据含义。

test_format1.csv和train_format1.csv里分别存放着测试数据和训练数据，测试数据集最后一个字段为prob，表示测试结果，训练数据集最后一个字段为label，训练数据各字段信息如下图所示：

Data Fields	Definition
user_id	顾客（用户）ID
merchant_id	商家（店铺）ID
label	取值范围：{0，1}，1 表示重复购买，0 表示非重复购买，测试集数据的这个字段为 Null

训练数据集

user_log_format1.csv里存放着用户行为日志，字段信息如下图所示：

Data Fields	Definition
user_id	顾客（用户）ID
item_id	商品 ID
cat_id	商品类目 ID
seller_id	商家（店铺）ID merchantid
brand_id	品牌 ID
time_tamp	行为发生时间
action_type	取值范围：{0，1，2，3}，0 表示点击，1 表示加入购物车，2 表示购买，3 表示收藏

用户行为日志数据

user_info_format1.csv里存放着用户行基本信息，字段信息如下图所示：

Data Fields	Definition
user_id	顾客（用户）ID
age_range	顾客（用户）年龄范围：1 表示<18；2 表示[18，24]；3 表示[25，29]；4 表示[30，34]；5 表示[35，39]；6 表示[40，49]；7 and 8 表示>=50；0and Null 表示未知
gender	顾客性别：0 表示女性；1 表示男性；2and Null 表示未知

用户基本信息数据

二 特征构造

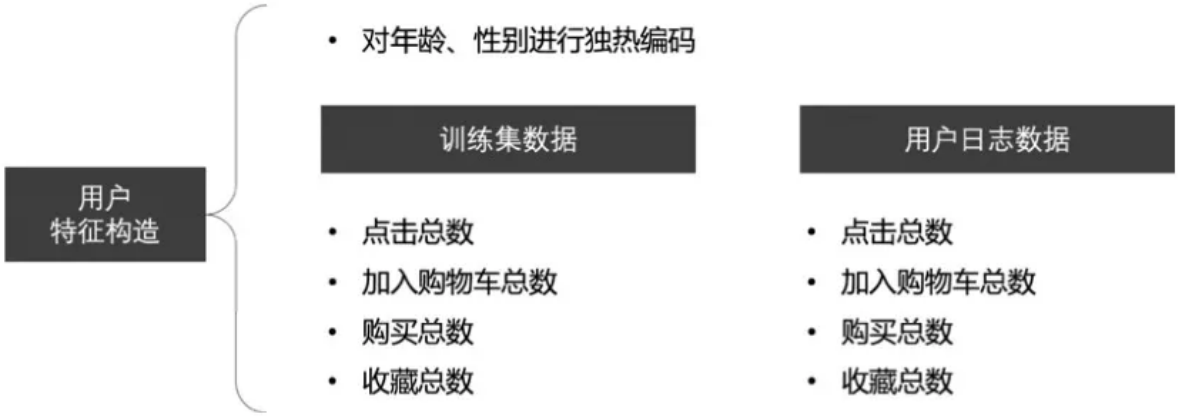
本赛题基于天猫电商数据，主要关心用户、店铺和商家这三个实体，所以特征构造上也以用户、店铺和商家为核心，可以分为以下几部分：

用户-店铺特征构造

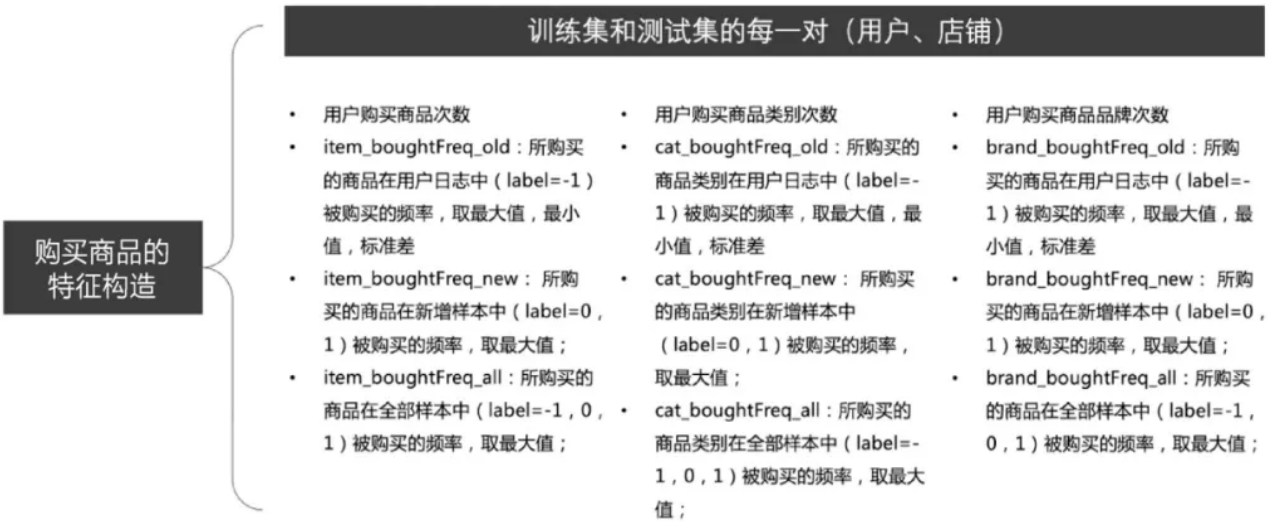
店铺特征构造

对店铺特征选取可以使用，如 Numpy 的 corrcoef(x,y)函数计算相关系数，保留相关系数小于0.9的特征组合，具体内容如图 2-3。

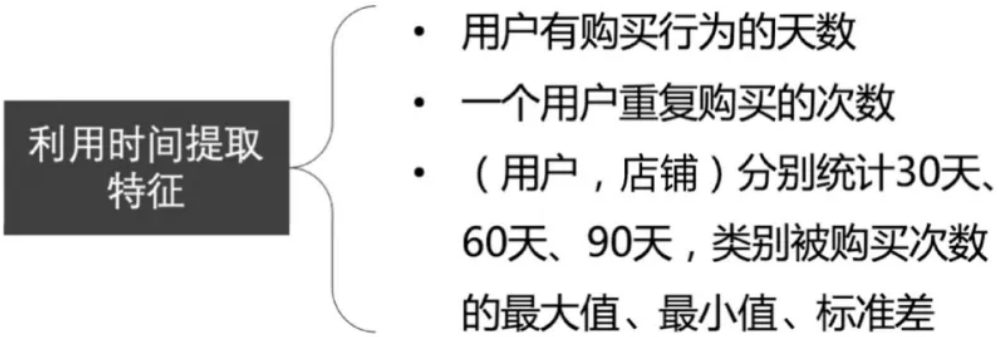
商家特征选取



用户特征构造

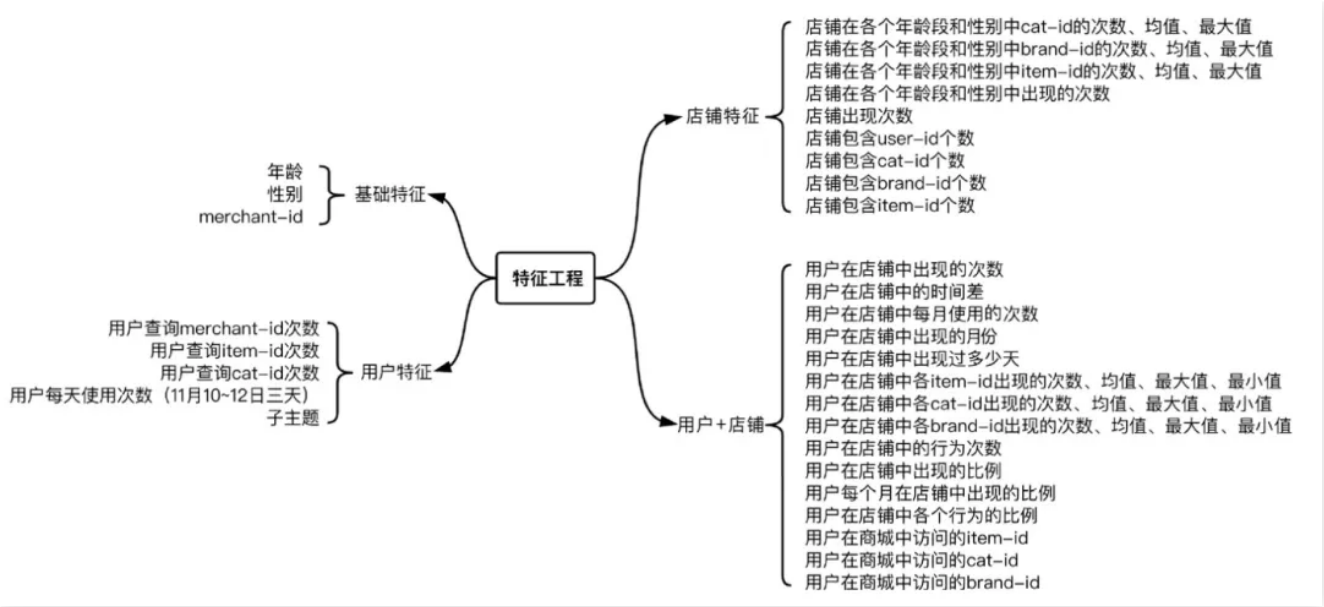


用户购买商品特征构造



利用时间提取特征

总结以上内容，特征主要基于基础特征、用户特征、店铺特征、用户+店铺四个方面，如下图所示：



特征总结

三 特征提取

首先我们导入需要的工具包，进行数据分析和特征提取。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt import seaborn as sns
from scipy import stats
import gc
from collections import Counter import copy
import warnings warnings.filterwarnings("ignore")
%matplotlib inline
```

接下来我们将按以下步骤进行特征提取。

特征提取步骤

1 读取数据

直接调用Pandas的read_csv函数读取训练集和测试集及用户信息、用户日志数据。

```
test_data = pd.read_csv('./data_format1/test_format1.csv') train_data = pd.rea
```

2 数据预处理

对数据内存进行压缩：

```
def reduce_mem_usage(df, verbose=True):
    start_mem = df.memory_usage().sum() / 1024**2
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    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).m
                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.int3
                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( n
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
                    end_mem = df.memory_usage().sum() / 1024**2
                    print('Memory usage after optimization is: {:.2f} MB'.for
```

首先测试数据添加到训练数据后，然后将用户基本信息合并到训练数据左边，并删除不需要的变量，释放内存。

```
all_data = train_data.append(test_data)
all_data = all_data.merge(user_info,on=['user_id'],how='left')
del train_data, test_data, user_info
gc.collect()
```

将用户日志数据各字段合并成一个新的字段item_id，并将其插入到用户信息数据之后。

```
# 用户日志数据按时间排序
user_log = user_log.sort_values(['user_id', 'time_stamp'])
# 合并用户日志数据各字段，新字段名为item_id
list_join_func = lambda x: " ".join([str(i) for i in x])
agg_dict = {
    'item_id': list_join_func,
    'cat_id': list_join_func,
```

```
    'seller_id': list_join_func,
    'brand_id': list_join_func,
    'time_stamp': list_join_func,
    'action_type': list_join_func
}
rename_dict = {
    'item_id': 'item_path',
    'cat_id': 'cat_path',
    'seller_id': 'seller_path',
    'brand_id': 'brand_path',
    'time_stamp': 'time_stamp_path',
    'action_type': 'action_type_path'
}

def merge_list(df_ID, join_columns, df_data, agg_dict, rename_dict):
    df_data = df_data.groupby(join_columns).agg(agg_dict).reset_index().renam
        columns=rename_dict)
    df_ID = df_ID.merge(df_data, on=join_columns, how="left")
    return df_ID
all_data = merge_list(all_data, 'user_id', user_log, agg_dict, rename_dict)
del user_log
gc.collect()
```

3 特征统计函数定义

基于之前的特征构造图，我们提前编写一些统计相关函数，依次有：数据总数、数据唯一值总数、数据最大值、数据最小值、数据标准差、数据中top N数据以及数据中top N数据的总数。

```
def cnt_(x):
    try:
        return len(x.split(' '))
    except:
        return -1

def nunique_(x):
    try:
        return len(set(x.split(' ')))
```

```
    except:
        return -1

def max_(x):
    try:
        return np.max([float(i) for i in x.split(' ')])
    except:
        return -1

def min_(x):
    try:
        return np.min([float(i) for i in x.split(' ')])
    except:
        return -1

def std_(x):
    try:
        return np.std([float(i) for i in x.split(' ')])
    except:
        return -1

def most_n(x, n):
    try:
        return Counter(x.split(' ')).most_common(n)[n-1][0]
    except:
        return -1

def most_n_cnt(x, n):
    try:
        return Counter(x.split(' ')).most_common(n)[n-1][1]
    except:
        return -1
```

基于上面编写的基本统计方法，我们可以针对数据进行特征统计。

```
def user_cnt(df_data, single_col, name):
    df_data[name] = df_data[single_col].apply(cnt_)

```

```
    return df_data

def user_nunique(df_data, single_col, name):
    df_data[name] = df_data[single_col].apply(nunique_)
    return df_data

def user_max(df_data, single_col, name):
    df_data[name] = df_data[single_col].apply(max_)
    return df_data

def user_min(df_data, single_col, name):
    df_data[name] = df_data[single_col].apply(min_)
    return df_data

def user_std(df_data, single_col, name):
    df_data[name] = df_data[single_col].apply(std_)
    return df_data

def user_most_n(df_data, single_col, name, n=1):
    func = lambda x: most_n(x, n)
    df_data[name] = df_data[single_col].apply(func)
    return df_data

def user_most_n_cnt(df_data, single_col, name, n=1):
    func = lambda x: most_n_cnt(x, n)
    df_data[name] = df_data[single_col].apply(func)
    return df_data
```

4 提取统计特征

基于上一步中编写的用户数据统计函数，以店铺特征统计为例，统计与店铺特点有关的特征，如店铺、商品、品牌等。

```
# 取2000条数据举例
all_data_test = all_data.head(2000)
# 总次数
all_data_test = user_cnt(all_data_test, 'seller_path', 'user_cnt')
```

```
# 不同店铺个数
all_data_test = user_nunique(all_data_test, 'seller_path', 'seller_nunique ')
# 不同品类个数
all_data_test = user_nunique(all_data_test, 'cat_path', 'cat_nunique')
# 不同品牌个数
all_data_test = user_nunique(all_data_test, 'brand_path',
                             'brand_nunique')
# 不同商品个数
all_data_test = user_nunique(all_data_test, 'item_path', 'item_nunique')
# 活跃天数
all_data_test = user_nunique(all_data_test, 'time_stamp_path',
                             'time_stamp _nunique')
# 不同用户行为种数
all_data_test = user_nunique(all_data_test, 'action_type_path',
                             'action_ty pe_nunique')
```

此外还可以统计用户最喜欢的店铺、最喜欢的类目、最喜欢的品牌、最长见的行为动作等数据。

```
# 用户最喜欢的店铺
all_data_test = user_most_n(all_data_test, 'seller_path', 'seller_most_1', n=1
# 最喜欢的类目
all_data_test = user_most_n(all_data_test, 'cat_path', 'cat_most_1', n=1)
# 最喜欢的品牌
all_data_test = user_most_n(all_data_test, 'brand_path', 'brand_most_1', n= 1)
# 最常见的行为动作
all_data_test = user_most_n(all_data_test, 'action_type_path', 'action_type _1
```

5 利用countvector和tfidf提取特征

CountVectorizer与TfidfVectorizer是Scikit-learn的两个特征数值计算的类，接下来我们将结合两者进行特征提取。

```
from sklearn.feature_extraction.text import CountVectorizer

```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
from scipy import sparse
tfidfVec = TfidfVectorizer(stop_words=ENGLISH_STOP_WORDS,
                           ngram_range=(1, 1),
                           max_features=100)
columns_list = ['seller_path']
for i, col in enumerate(columns_list):
    tfidfVec.fit(all_data_test[col])
    data_ = tfidfVec.transform(all_data_test[col])
    if i == 0:
        data_cat = data_
    else:
        data_cat = sparse.hstack((data_cat, data_))
```

6 嵌入特征

词嵌入是一类将词向量化的模型的统称，核心思想是将每个词都映射到低维空间(K 为50~300)上的一个稠密向量。

```
import gensim

model = gensim.models.Word2Vec(
    all_data_test['seller_path'].apply(lambda x: x.split(' ')),
    size=100,
    window=5,
    min_count=5,
    workers=4)

def mean_w2v_(x, model, size=100):
    try:
        i = 0
        for word in x.split(' '):
            if word in model.wv.vocab:
                i += 1
            if i == 1:
                vec = np.zeros(size)
```

```
                vec += model.wv[word]
            return vec / i
        except:
            return np.zeros(size)

def get_mean_w2v(df_data, columns, model, size):
    data_array = []
    for index, row in df_data.iterrows():
        w2v = mean_w2v_(row[columns], model, size)
        data_array.append(w2v)
    return pd.DataFrame(data_array)

df_embeeding = get_mean_w2v(all_data_test, 'seller_path', model, 100)
df_embeeding.columns = ['embeeding_' + str(i) for i in df_embeeding.columns]
```

7 Stacking 分类特征

以使用 lgb 和 xgb 分类模型构造 Stacking特征为例，实现方式如下：

```
# 1、使用 5 折交叉验证
from sklearn.model_selection import StratifiedKFold, KFold
folds = 5
seed = 1
kf = KFold(n_splits=5, shuffle=True, random_state=0)
# 2、选择 lgb 和 xgb 分类模型作为基模型
clf_list = [lgb_clf, xgb_clf]
clf_list_col = ['lgb_clf', 'xgb_clf']
# 3、获取 Stacking 特征
clf_list = clf_list
column_list = []
train_data_list=[]
test_data_list=[]
for clf in clf_list:
  train_data,test_data,clf_name=clf(x_train, y_train, x_valid, kf, label_ spl
  train_data_list.append(train_data)
  test_data_list.append(test_data)
train_stacking = np.concatenate(train_data_list, axis=1)
```

```
test_stacking = np.concatenate(test_data_list, axis=1)
```

运行结果：

```
[1] valid_0's multi_logloss: 0.240875
Training until validation scores don't improve for 100 rounds.
[2] valid_0's multi_logloss: 0.240675
[226] train-mlogloss:0.123211 eval-mlogloss:0.226966
Stopping. Best iteration:
[126] train-mlogloss:0.172219 eval-mlogloss:0.218029
xgb now score is: [2.4208301225770263, 2.2433633135072886, 2.51909203146584 34
xgb_score_list: [2.4208301225770263, 2.2433633135072886, 2.5190920314658434, 2
xgb_score_mean: 2.4506746084485203
```