atec初赛

经过了几个月的奋斗，比赛终于结束了。最终的成绩是初赛9/783 ， 复赛13/86。
对于第一次参赛的人来说还算满足了（想想最后再努力下的话可以更往前的），先来总结一下初赛的经验，有时间再写下复赛的经验（感觉更像hive大战）

赛题简介

风控大脑这个题的背景来源于真实的安全风控业务。这个问题的本质是，如果发生了交易风险，我们如何用大量的历史数据去发现异常交易，我们需要根据已有的特征去判别一笔交易是否存在风险。
这样的风控问题，有如下的四个特点：

1. 样本失衡。99.99%的交易都是好人，真正产生案件的交易是非常少的。
2. 数据海量。风控采集非常非常多的数据，这些特征不一定是有用的
3. 攻防激烈。坏人会根据被拒绝的交易不断改变提交的参数，试探系统背后的规则策略。

评价指标

本赛题定义并计算在不同打扰率下对应的覆盖率 TPR1： 当FPR等于0.001时的TPR TPR2：当FPR等于0.005时的TPR TPR3：当FPR等于0.01时的TPR 模型成绩 score=

0.4 * TPR1 + 0.3 * TPR2 + 0.3 * TPR3。这个方法也需要我们在线下实现，方便线下的测试。

数据分析

数据的统计分析

目前来看，这是一个分类任务。我们需要根据特征集合，判定某个交易是安全交易（标签为0）还是是灰色交易（标签为1），训练集中标签为-1表示不确定。
先导入一些必要的数据包

```
import pandas as pd
import numpy as np
from pandas import DataFrame,Series
import matplotlib.pyplot as plt
import sys
path='D:\\libsvm-3.21\\python'
sys.path.append(path)
%matplotlib inline
##算法相关的导入包
from svmutil import *
import seaborn as sns
import lightgbm as lgb
import gc
import time
from sklearn import preprocessing
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
读取数据内容，并保存为dataframe格式
#读取训练数据，并观察数据的格式
data=pd.read_csv('D:/2018/kaggle/anti/atec_anti_fraud_train.csv')
data['id']=data['id'].astype('category')
data['label']=data['label'].astype('category')
data.describe()

#查看样本的标签的分布情况
fre=data.label.value_counts()

#使用图表更加直观的观察分布
fre=data.label.value_counts()
fre.plot(kind='bar')
plt.title(u"样本预测情况") # puts a title on our graph
plt.ylabel(u"人数")

#观察每天的数据量

date_info=data['date'].value_counts().sort_index()
date_info
```

分别运行上面的几段代码可以得到如下的直观印象：一共有994,731条训练数据，对于需要验证的标签，一共有三种，分别是0,1，-1。其中98.3%的用户的标签是0，1.21%的用户是1,0.48%的用户标签是-1。训练样本包含了62天的数据信息,时间区间从20170905到20171105，平均每天的记录规模是16000个

标签为-1的样本的处理

-1是未知标签，因此在这个问题上，合理使用这些样本是一个加分关键点。

1. 完全不考虑label为-1的情况（可以作为baseline）
2. 将label为-1的全部标记为1，放入训练集.因为这些因为异常被中断的交易是蚂蚁风控系统认为风险比较大的交易，因此可能蕴含着异常交易的特征，可以用于补充label 1样本太少的问题。
3. 使用机器学习算法对label为-1样本进行预测学习，将被大概率预测为1的样本标记为1，放入训练集中。

这是一个需要尝试的问题，通过几次尝试对比，我们的团队发现，在初赛阶段，第二种处理方式的效果要优于另外两种方式。

特征分析和选择

赛方一共提供了298个特征列，在使用全部的特征进行模型训练时，发现模型指标只能在0.30左右，陷入局部最优。因此需要对特征列进行分析，选择真正对结果有帮助的特征列。因为特征列的缺失情况不一，为避免**缺失值填充**对特征选择的影响，我们先进行了特征选择。选择的主要依据有

1. 特征的缺失率不能太大。若大部分样本都缺失，则特征无用。
2. 训练集和测试集的特征分布一致，不能有太大的偏差。否则建立的模型不具备泛化能力。
3. 标签在不同的特征值上的分布具有差异性
4. 通过gdbt算法训练出的模型选择重要特征

缺失率分析

对所有的特征检查数据缺失.可以发现很多不同的但是序号连续的特征可能是互相关联的。比如f1-f4,都是三值函数，都没有缺失。f20-f23都缺失了207448条记录。同时根据Data_Analysis_v1中的分析，有如下结论，缺失值为x的特征一定同时缺失，也就是说如果A列特征缺失x条，B列特征也缺失x条，那么一条记录中，A和B要么同时缺失，要么同时不缺失。

```
#检查特征的缺失情况，第一列是缺失数量，第二列是特征列数量
data_lost=data.isnull().sum().reset_index(name='count')
data_lost.columns=['feature','counts']
lost_count=data_lost.groupby('counts',as_index='False').count()
lost_count
```

	feature
counts	
0	21
919	48
136021	69
199825	1
207448	6
207585	24
211036	4
270515	78
273904	14
286285	5
341661	14
341887	4
925781	12

可以发现虽然一共有298个特征列，但是数据的缺失数量只有13个不同的值。可以看到有12个特征他们的缺失情况是925781，大部分的样本都没有数据。只有139个特征的缺失率是低于20%。 对于这样的数据分布，我们尝试了不同的特征删除策略：删除缺失率大于20%，30%，40%，50%等等。
在对测试集进行缺失值分析的时候，我们也查看了测试集的特征。我们发现对于相同的特征，训练集和测试集的缺失情况不尽相同，因此对于缺失率相差比较大的特征列，最好也删除。否则训练集的特征对于测试集没有很好的适应能力。

训练集和测试集的分布情况

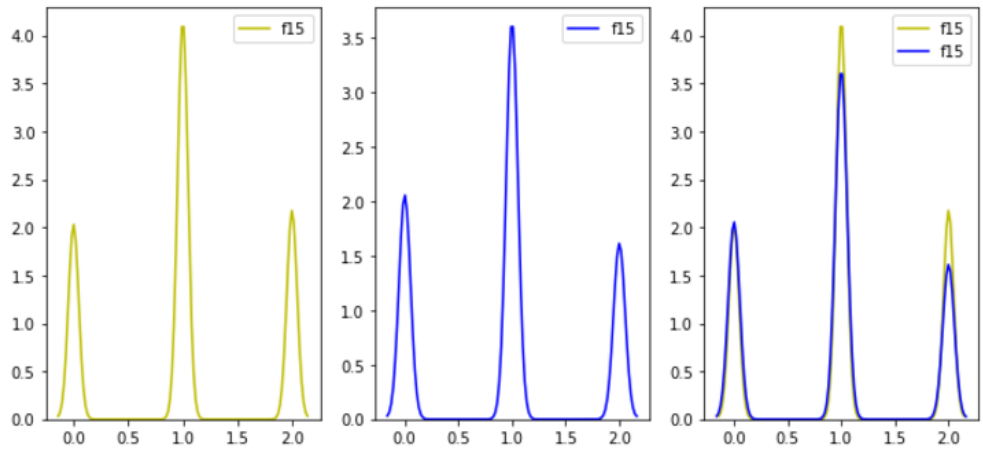
正常的情况下，训练集和测试集的特征分布应该一致，这样根据训练集训练得到的模型才对测试集适用。一致的分布意味着，取值范围一致，概率密度核kde分布曲线类似。

```
def plot_kde(train, test, col, values=True):
fig,ax =plt.subplots(1,4,figsize=(15,5))
sns.kdeplot(train[col][train['label']==0],color='g',ax=ax[0])
sns.kdeplot(train[col][train['label']==1],color='r',ax=ax[0])
sns.kdeplot(train[col][train['label']==-1],color='y',ax=ax[0])
sns.kdeplot(train[col],color='y',ax=ax[1])
sns.kdeplot(test[col],color='b',ax=ax[2])
sns.kdeplot(train[col],color='y',ax=ax[3])
sns.kdeplot(test[col],color='b',ax=ax[3])
plt.show()
del train, col,test
gc.collect()
train_data =
pd.read_csv('E:\\ATEC_FRAUD\\b_data\\atec_anti_fraud_train.csv')
test_data =
pd.read_csv('E:\\ATEC_FRAUD\\b_data\\atec_anti_fraud_test_b.csv'
```

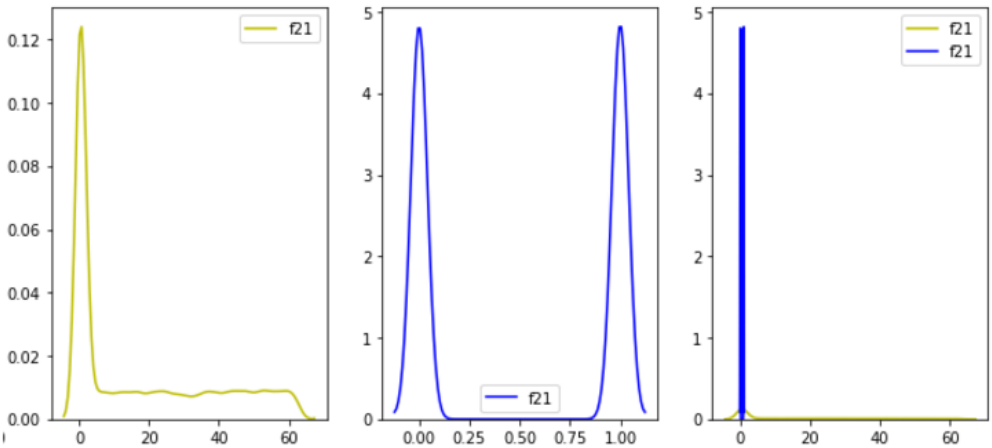
```
col_need =[ 'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10',
'f11', 'f12', 'f13', 'f14', 'f15', 'f16', 'f17','f18', 'f19', 'f20',
'f21','f22', 'f23', 'f24', 'f25', 'f26', 'f27','f28', 'f29', 'f30', 'f31',
'f52', 'f53', 'f161', 'f162', 'f163', 'f164', 'f165', 'f211','f212',
'f213', 'f214', 'f215', 'f216', 'f217', 'f218','f219', 'f220', 'f221',
'f222', 'f223', 'f224', 'f225', 'f226', 'f227','f228', 'f229', 'f230',
'f231', 'f232', 'f233','f234', 'f235', 'f236', 'f237', 'f238', 'f239',
'f240', 'f241', 'f242', 'f243','f244', 'f245', 'f246', 'f247', 'f248',
'f249', 'f250', 'f251', 'f252', 'f253', 'f254', 'f255', 'f256', 'f257',
'f258', 'f259','f260', 'f261', 'f262','f263', 'f264', 'f265', 'f266',
'f267', 'f268','f269', 'f270', 'f271', 'f272', 'f273', 'f274',
'f275','f276', 'f277', 'f278', 'f279', 'f280', 'f281', 'f282', 'f283',
'f284', 'f285', 'f286', 'f287', 'f288', 'f289', 'f290', 'f291','f292',
'f293', 'f294', 'f295', 'f296', 'f297']
for col in train_data.columns:
if col !='id' and col !='label' and col in col_need:
plot_kde(train_data,test_data,col)

```

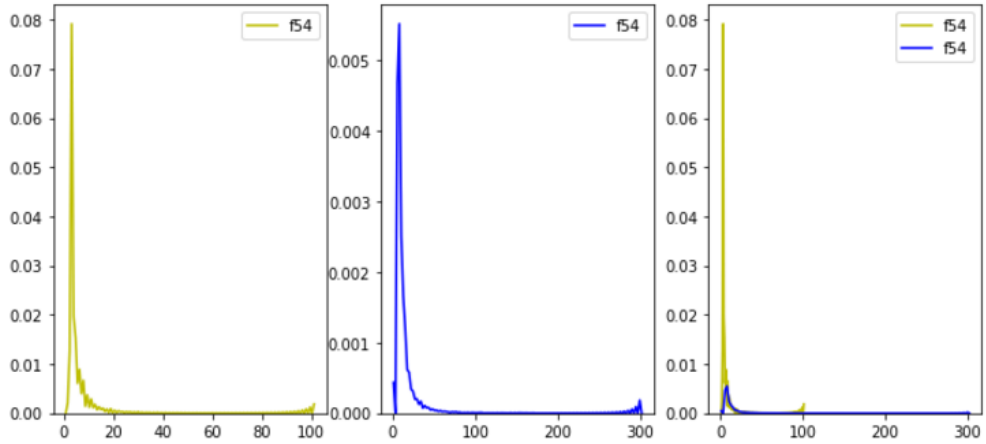
成功运行上面的代码可以得到一系列的如下图所示的对比图，其中左图为训练集的分布kde，中图为测试集的kde，右图为训练集和测试集的合并在一起观察的情况：
以特征f15为例，在训练集和测试集上都只有三个值，且在这三个值上的kde曲线基本重叠。



而f21特征在训练集上是一个多值分布，而在测试集上是二值分布，通过最后一张对比图可以明显的看出分布的异常。



f54虽然在训练集和测试集上都是多值分布，但是训练集分布的范围是[0,100]，而测试机的范围在[0,300]。因此这样的特征不具备训练的意义。



筛选具有辨识能力的特征

对于特定的特征，如果样本标签在各值上的分布没有明显的区别，那么这个特征对于最终的预测能起到的作用很小。相反，如果某个特征值上异常交易占比特别高，那么当新样本出现这个特征值时，系统就会引起警觉。常规的方式有

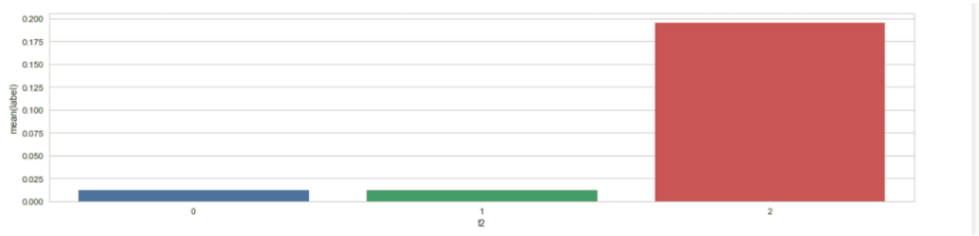
1. 通过训练集特征与label的pearson相关性选择特征
2. 通过观察异常交易在每个特征值上的占比情况选择特征

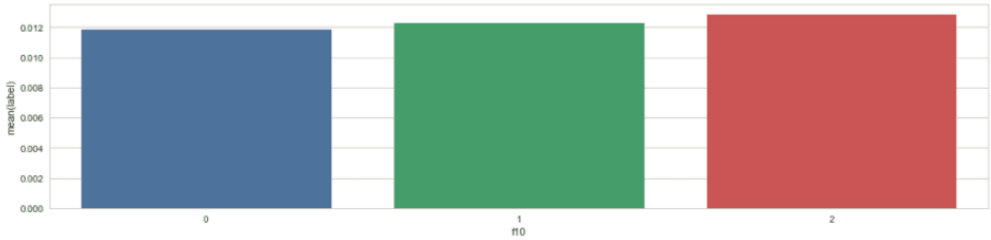
第一个比较容易实现，我们来看一下第二个~

以前四个特征为例，前四个特征是三值特征，即 每个特征的取值只有0,1,2三个值。检查label为0和1，与特征为0,1,2之间的关系.可以明确的发现对于前4个特征来说，当其为2时，危险交易数目/正常交易数目的比率明显高于其他两个值。尤其是f2特征，这个比率达到了0.25.

```
feat=['f1','f2','f3','f4']
fig = plt.figure()
fig.set(alpha=0.2)
i=0
for feature in feat:
safe_user = data[data.label == 0][feature].value_counts()
risk_user= data[data.label == 1][feature].value_counts()
rate=risk_user.div(safe_user)
plt.subplot2grid((2,2),(int(i/2),int(i%2)))
rate.plot(kind='bar')
i=i+1
plt.title(feature + u"危险用户/安全用户")
plt.show(
```

重点是我们要选择区分度好的特征，比如这里的f2,下面一张图的f10就没有这么好的区分度。





f10的三个值占比都很平均，无法通过这个特征单一分析出交易是否是异常交易的概率，对标签不具有敏感性。在特征选择的时候，这样的特征最好删除。
经过上面的三层分析，我们可以删掉很大的一部分特征了。在这里比赛中，我们主要使用了上面的方式进行特征的筛选，而使用随机森林以及GBDT的方式则略作涉猎，并没有系统的作为依据。

```
#筛选之后剩下的需要使用的特征

col_need=['f1', 'f2', 'f3', 'f4', 'f6', 'f7', 'f8', 'f9', 'f11',
'f12', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f19', 'f161',
'f162', 'f163', 'f164', 'f165', 'f211', 'f212', 'f213', 'f214',
'f215', 'f216', 'f217', 'f218', 'f219', 'f220', 'f221', 'f222',
'f223', 'f224', 'f225', 'f226', 'f227', 'f228', 'f229', 'f230',
'f231', 'f232', 'f233', 'f234', 'f235', 'f236', 'f237', 'f238',
'f239', 'f240', 'f241', 'f242', 'f243', 'f244', 'f245', 'f246',
'f247', 'f248', 'f249', 'f250', 'f251', 'f252', 'f253']
#这个函数用于将其余的不需要的特征列删除
def preprocess_likeo(df,col_need):
num =0
drop_cols=[]
for col in df.columns:
if(col not in col_need):
drop_cols.append(col)
num= num +1
df.drop(drop_cols,axis=1,inplace=True)
print('drop ',num, " features in data set")
return df
```

缺失值的填充

在对每个特征进行分析，然后通过实验综合考量得到训练特征之前，我们还需要做特征处理 缺失值填充可以分为连续值的缺失值填充和离散值的缺失值填充。连续值的缺失值填充常见的有中值填充、众数填充、均值填充以及使用randomforest等机器学习算法进行填充。缺失值的填充也有众数填充，-1填充以及使用学习算法进行填充。
在本题中，需要不断验证的是如下两个方面：

1. 如何划分特征是连续值还是离散值。我们通过观察每个特征的分布情况，人为的划分，然后根据赛方反馈的结果进行调整。
2. 用什么方法进行填充。在现阶段我们发现对于连续值，使用中值填充的方式优于均值填充。对于离散值，则是-1填充较好。

将要用到的函数

```
#添加每行数据的缺失值情况
def lost_info(data):
    data_lost=data.isnull().sum().reset_index(name='count')
    data_lost.columns=['feature','counts']
    lost_count=data_lost.groupby('counts',as_index='False').count()
    print("there are ",lost_count.shape[0] ," different lost values in ",data.shape[1]," columns.")
    print("Add ",lost_count.shape[0]," cols behind the origin data...")
    col_len=lost_count.shape[0]
    lost_values=lost_count.index
    lost_info_data=pd.DataFrame()
    for i in np.arange(1,lost_count.shape[0]):
        lost_value=lost_values
        lost_cols_count=lost_count.loc[lost_value,"feature"]
        #same_index=np.array(data_lost[data_lost.counts==lost_value]['feature'])[0]
        same_index=data_lost[data_lost.counts==lost_value]['feature']
        post_fix=np.array(same_index)[0]

data_lost_sample=data[same_index].isnull().sum(axis=1)/lost_cols_count

lost_info_data["lost_"+str(lost_cols_count)+"_"+post_fix]=data_lost_sample
    data["sum"]=lost_info_data.sum(axis=1)
    data["isnull_count"]=data.isnull().sum(axis=1)
    #data=pd.concat([data,lost_info_data],axis=1)
    return data

#分别使用均值填充，插值填充和判定是否空值的方式填充连续型特征
def mean_fillna(df,features):
    for feature in features:
        df[feature+'_mean']=df[feature].fillna(df[feature].mean())
    return df

def interpolate_fillna(df,features):
    for feature in features:
        df[feature]=df[feature].interpolate()
    return df

def isnan_fillna(df,features):
    for feature in features:
        df[feature+'_isnan']=df[feature]
        df.loc[(df[feature].isnull()),feature+'_isnan']=1
```

```
        df.loc[(df[feature].notnull()),feature+'_isnan']=0
    return df
def median_fillna(df,features):
    for feature in features:
        df[feature+'_meadian']=df[feature].median()
    return df

#连续特征的处理方式
def continues_fillna(df,features):
    df=median_fillna(df,features)
    #df=interpolate_fillna(df,features)
    return df


#使用-1填充离散型特征，并生成是否为空的特征列
def category_fillna(df,category_cols):
    for category_col in category_cols:
        df[category_col].fillna(-1,inplace=True)
    return df

#为train和test数据标记是否是离散值，当数据unquie唯一值个数超过threld的时候，则被认为是连续特征，否则是离散特征。
#并且填充缺失值
def set_cate_type_and_fillna(train_x,test,threld=80):
    #给离散列转换类型,得到离散值
    data_clo_unique_num=pd.DataFrame(train_x.apply(lambda x:len(x.unique())),columns=['unique'])
    for i in data_clo_unique_num.index:
        print(i," ",data_clo_unique_num.loc[i,'unique'])
    cate_cols=set(data_clo_unique_num[data_clo_unique_num['unique']<threld].index)
    cont_cols=set(train_x.columns)-cate_cols
    for i in cate_cols:
        train_x=train_x.astype('category')
        test=test.astype('category')
    train_x=category_fillna(train_x,cate_cols)
    test=category_fillna(test,cate_cols)
    train_x=continues_fillna(train_x,cont_cols)
    test=continues_fillna(test,con_cols)

print("the size of train set is "+str(train_x.shape[0])+", the size of test set is "+str(test.shape[0]))
set_cate_type(train_x,test,10)

```

模型训练

在完成了特征分析和处理之后，进入到了最复杂的过程，模型训练。实际上，上面的-1标签的处理、数据倾斜的处理、特征选择和缺失值填充也是模型训练的有机组成部分。在实际的工作中，我们会先确定上面的部分，然后调节实验参数，得到较好的本地结

果，上传系统得到评分，然后再修改特征部分，调节参数，...，一直循环这样的过程，不断分析本地结果和系统结果的得失。有时候会陷入局部最优，一直无法提升成绩，有的时候又会因为一次提升而全组欢庆。
模型训练大致有四个步骤：
交叉验证集的生成
模型选择
参数优化
模型融合
按照赛后的经验，我们发现使用多模型融合可以使得结果获得0.02左右的提升。

交叉验证集的生成

在比赛过程中，发现有同仁提出按照时间排序之后，分成五份，依次取4份作为训练集一份作为测试集进行模型训练，如此交叉验证的结果优于随机选择的5折交叉验证。

模型的选择

Lightgbm是微软去年推出的很好用的机器学习竞赛算法。它的本质也是集成学习的算法，这个方法中可以通过参数选择使用随机森林、GBDT或者是Goss算法。相对于传统的sklearn中的集成算法，这个算法的优点在于：

1. 传统集成算法对于CART树使用pre-sorted的算法。这是一个简单的解决方案，但是不易于优化。LightGBM 利用基于 histogram 的算法 通过将连续特征（属性）值分段为 discrete bins 来加快训练的速度并减少内存的使用。
2. 大部分决策树的学习算法通过level(depth)-wise策略生长树，而lightgbm使用Leaf-wise (Best-first) 的决策树生长策略。它将选取具有最大损失的叶节点来生长。 因此在叶子结点相同的时候，leaf-wise 算法可以比 level-wise算法减少更多的损失。
3. 大多数机器学习工具都无法直接支持类别特征，一般需要把类别特征转化one-hotting特征，降低了空间和时间的效率。LightGBM优化了对类别特征的支持，可以直接输入类别特征，不需要额外操作。并在决策树算法上增加了类别特征的决策规则。

在实践中，我们发现lightgbm算法可以达到xgboost相同的score，但是时间开销更少。同时，因为lightgbm可以同时训练GBDT模型和随机森林模型，因此简化了写代码的时间开销。通过实际训练对比，发现使用GBDT的score要略优于使用随机森林模型。LightGbm的训练参数有很多，主要的有叶子节点数num_leaves，叶子结点中含有的最少样本数min_child_samples，学习速率learning_rate以及训练循环次数num_boost_round。使用5折交叉验证，在最佳的情况下，score可以达到0.43。

参数优化

常用的参数优化方式有两种，第一种是使用基于网格搜索grid-search的方式；第二种是基于贝叶斯优化的参数搜索方式，已有的hyperopt包就是这样的最优参数搜索包。

grid-search参数优化

基于grid-search的参数优化的本质是给定需要调节的参数名称以及参数值，通过遍历这个参数空间，尝试每一种参数之间的搭配，选择score最高的参数值作为返回。sklearn.model_selection中提供了ParameterGrid方法，可以对给定的参数生成遍历之

后的参数组合列表，将这样的参数传入算法中，依次训练即可得到参数空间中最优的参数值。

hyperopt参数优化

grid-search 是全空间扫描，所以比较慢。hyperopt是一种通过贝叶斯优化（贝叶斯优化简介）来调整参数的工具，对于像XGBoost这种参数比较多的算法，可以用它来获取比较好的参数值。
hyperopt需要对每个参数指定搜索空间，而不是如grid-search中那样指定值，比如参数x在0-1区间内均匀取值，参数y在0-1之间对数取值。然后，可以指定参数优化的搜索算法，如随机搜索(对应是hyperopt.rand.suggest)和模拟退火(对应是hyperopt.anneal.suggest)，TPE算法。

模型融合

模型融合就是将多个模型的结果重新整理归纳，得到一个更具有泛化能力和精度的最终模型。多模型融合算法可以比单一模型算法有极为明显的效果提升。
将每个模型结果取均值得到最终的结果，这样的模型融合方式就是均值融合。这种简单的方式，实际效果很很好。通过融合多个成绩在0.40-0.43的lightgbm模型，最终的score可以达到0.4361。
下面是模型训练部分的代码，封装了一些方法，在注释里面进行解释。

```
#评价函数，可用于普通的sklearn包算法的结果分析，如GBDT
def feval_spec(preds, train_data):
    from sklearn.metrics import roc_curve
    fpr, tpr, threshold = roc_curve(train_data, preds)
    tpr0001 = tpr[fpr <= 0.001].max()
    tpr001 = tpr[fpr <= 0.005].max()
    tpr005 = tpr[fpr <= 0.01].max()
    #tpr01 = tpr[fpr.values <= 0.01].max()
    tprcal = 0.4 * tpr0001 + 0.3 * tpr001 + 0.3 * tpr005
    return tprcal

#用于lightgbm的评价函数
def feval_spec_train(preds, train_data):
    from sklearn.metrics import roc_curve
    fpr, tpr, threshold = roc_curve(train_data.get_label(), preds)
    tpr0001 = tpr[fpr <= 0.001].max()
    tpr001 = tpr[fpr <= 0.005].max()
    tpr005 = tpr[fpr <= 0.01].max()
    #tpr01 = tpr[fpr.values <= 0.01].max()
    tprcal = 0.4 * tpr0001 + 0.3 * tpr001 + 0.3 * tpr005
    return 'spec_cal',tprcal,True

#Kfolds将数据根据索引分为k份，可以用于生成第一层输出的时候，保证各折数据之间没有干扰
# GroupSelect将Kfolds中的第i折数据索引作为预测集pre_set，其余的作为训练集train_set。
# TrainSet针对训练数据的i折，得到训练数据和第二层的输入数据
```

```
def Kfolds(x_index,k=5,seed=1):
    np.random.seed(seed)

xL=np.array_split(np.random.choice(x_index,len(x_index),replace=False),k)
    return xL

def GroupSelect(xL,i=0):
    xLc=xL.copy()
    pre_index=list(xLc.pop(i))
    train_index=sum([list(x) for x in xLc],[])
    return train_index,pre_index

def TrainSet(x,y,xL,i=0):
    train_index,pre_index=GroupSelect(xL,i)
    train_x,pre_x=x.loc[train_index],x.loc[pre_index]
    train_y,pre_y=y.loc[train_index],y.loc[pre_index]
    return train_x,train_y,pre_x,pre_y

#K折交叉验证，可用于sklearn传统机器学习算法
def KflodsTrain(x,y,test,k=5,classifier=lgb.LGBMClassifier()):
    xL=Kfolds(np.array(x.index),k)
    #predict_proba=pd.DataFrame()
    test_predict_proba=pd.DataFrame()
    for i in range(k):
        print("begin the "+str(i)+" of "+str(k)+" kflods training...")
        train_x,train_y,pre_x,pre_y=TrainSet(x,y,xL,i)
        model=classifier.fit(train_x,train_y)
        predict_res=pd.Series(model.predict_proba(pre_x)[:,
1],index=pre_x.index)
        train_metric=feval_spec(model.predict_proba(train_x)[:,1],train_y)
        valid_metri=feval_spec(predict_res,pre_y)
        print("The ",i," times res: train set spe_val:",train_metric,",
validation set sep_val: ",valid_metri)
        #predict_proba=pd.concat([predict_proba,predict_res])
        test_predict_proba=pd.Series(model.predict_proba(test)
[:,1],index=test.index)
    #print("The output data of classifier: ",type(classifier), " is ready
for stacking...")
    #print("the size  of data is ",predict_proba.shape[0])
    test_predict_proba_mean=test_predict_proba.mean(axis=1)
    return test_predict_proba_mean

#K折交叉验证算法，用于lightgbm算法
def KflodsTrain_light(x,y,test,k=5,lgb_params=None):
    if lgb_params==None:
        lgb_params = {
            'boosting_type': 'gbdt',
            'objective': 'binary',
            'nthread': -1,
            'metric': 'auc',
            'num_leaves': 7,  # -1 means no limit
```

```
            'min_child_samples': 1000,  # Minimum number of data need in a child(min_data_in_leaf)
        }
    xL=Kfolds(np.array(x.index),k)
    #predict_proba=pd.DataFrame()
    test_predict_proba=pd.DataFrame()
    for i in range(k):
        print("begin the "+str(i)+" of "+str(k)+" kflods training...")
        train_x,train_y,pre_x,pre_y=TrainSet(x,y,xL,i)
        xgtrain = lgb.Dataset(train_x,train_y)
        xgvalid = lgb.Dataset(pre_x,pre_y)
        evals_results={}
        bst1 = lgb.train(lgb_params,xgtrain,valid_sets=[xgtrain,xgvalid],valid_names=['train','valid'],evals_result=evals_results,
                         num_boost_round=200,early_stopping_rounds=30,verbose_eval=False,feval=feval_spec_train)
        print('The ',i,' times running....')
        print('Best ite and score ',bst1.best_iteration,bst1.best_score)
        #predict_proba=pd.concat([predict_proba,predict_res])

test_predict_proba=pd.Series(bst1.predict(test,num_iteration=bst1.best_iteration),index=test.index)
    #print("The output data of classifier: ",type(classifier), " is ready for stacking...")
    #print("the size  of data is ",predict_proba.shape[0])
    test_predict_proba_mean=test_predict_proba.mean(axis=1)
    return test_predict_proba_mean

#设置lightgbm参数之后，调用上面的方法即可
total_res=pd.DataFrame()
total_res['id']=test_id
lgb_params = {
            'boosting_type': 'gbdt',
            'objective': 'binary',
            'nthread': -1,
            'metric': 'auc',
            'bagging_fraction':0.8,
            'feature_fraction':0.8,
            'num_leaves': 250,  # -1 means no limit
            'min_child_samples': 100,  # Minimum number of data need in a child(min_data_in_leaf)
        }
total_res["score"]=KflodsTrain_light(train_x,train_y,test,10,lgb_params)
total_res.to_csv("submission_b_22.csv",index=False)

```