In [7]:

```python
min_max_map={}
file = open('min_max_map.txt', 'r')
for line in file.readlines():
    line = line.strip()
    line = line.split(' ')
    min_max_map[line[0]] = (line[1], line[2])
    # 依旧是关闭文件
file.close()

print(min_max_map)
```

```
{'x0': ('0.0', '1.0'), 'x1': ('-2.0', '179.0'), 'x2': ('-100.0', '-10
0.0'), 'x3': ('-1.001', '2623.970399996324'), 'x4': ('-1.001', '300.
0'), 'x5': ('-1.001', '300.0'), 'x6': ('-100.0', '0.0'), 'x7': ('-100.
0', '100000.0'), 'x8': ('-1.0', '1.0'), 'x9': ('-100.0', '1.0'), 'x1
0': ('0.0', '23.0'), 'x11': ('-1.001', '2.451644724085354'), 'x12':
('0.0', '136.15200000003097'), 'x13': ('0.0', '2.0'), 'x14': ('2.0',
'8.0'), 'x15': ('-1.0', '112.63400000000547'), 'x16': ('-1.0', '25.
0'), 'x17': ('0.0', '2030.7250000000058'), 'x18': ('-1.0', '5113.98044
9986075'), 'x19': ('-100.0', '465.0'), 'x20': ('-1.0', '1526.0'), 'x2
1': ('-100.0', '1.0'), 'x22': ('-1.0', '33087.14999992662'), 'x23':
('1.0', '5858.876000000004'), 'x24': ('0.0', '31.0'), 'x25': ('0.0',
'2.0'), 'x26': ('-1.001', '22.817361304343535'), 'x27': ('0.0', '38700
3224.2956824'), 'x28': ('0.0', '8583943.171076408'), 'x29': ('-1.001',
'575.5990999999922'), 'x30': ('-1.001', '1.0'), 'x31': ('-1.0', '0.
0'), 'x32': ('-100.0', '0.012227292684181549'), 'x33': ('-100.0', '0.
0'), 'x34': ('-100.0', '15.0'), 'x35': ('-100.0', '568.797606421916'),
'x36': ('-10.0', '5531.085000000006'), 'x37': ('-1.001', '2.0'), 'x3
8': ('0.0', '300.0'), 'x39': ('0.0', '0.10914139809981968'), 'x40':
('0.0', '220561.48436187883'), 'x41': ('-1.0', '46.26271309659049'),
```

In [13]:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os
import sys
import json
import logging

sys.path.append('/home/moegwjawe1/atec_project/train/lmf/lib/python3.6/site-pack

import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

import joblib

input_data_path = '/home/admin/workspace/job/input/train.jsonl'
output_model_path = '/home/admin/workspace/job/output/train.model'
result_path = '/home/admin/workspace/job/output/result.json'

input_data_path = '/mnt/atec/train.jsonl'

logging.warning("Begin Train.")
# 1. 读取训练数据进行训练
import time
time_start = time.time()
# pd_raw_data = pd.read_json(input_data_path, encoding='utf-8', lines=True, nrow
pd_raw_data = pd.read_json(input_data_path, encoding='utf-8', lines=True)
time_end=time.time()
print('time cost', time_end - time_start, 's')
logging.warning("Finish Read.")
```

```
WARNING:root:Begin Train.
WARNING:root:Finish Read.

time cost 20.19862461090088 s
```

In [24]:

```
# 2. 数据预处理
pd_data = pd_raw_data.copy(deep=True)

# step2, 去除 label = -1, -9265 row
pd_data = pd_data.drop(pd_data[(pd_data['label'] == -1)].index)
# print(pd_data)
# print(pd_data.describe())
pd_data = pd_data.drop(columns=['memo_polish'],axis=1)

# 只保留 int 列 229
# pd_data = pd_data.drop(pd_data.columns[np.where((pd_data.dtypes!='int64').valu


min_max_map={}

# step3, 去除取值只有一个值的列, -15col
columns = pd_data.columns.values.tolist()[1:-1] # 去除 id 和 label 的所有列
drop_list = []
for col in columns:
    num = len(pd_data[col].unique())
    if num == 1:
        drop_list.append(col)

# 去除离群数据, 或者离群数据置空
print(pd_data.shape)
columns = pd_data.columns.values.tolist()[1:-1] # 去除 id 和 label 的所有列
for col in columns:
    if pd_data.dtypes[col]=='float64':
        d_min, d_max = pd_data[col].quantile([0.0001, 0.9999])
    elif pd_data.dtypes[col]=='int64':
        d_min, d_max = pd_data[col].quantile([0.001, 0.999])
    else:
        print("==========================")
    print(col,d_min,d_max)
    min_max_map[col]=(d_min,d_max)
    pd_data = pd_data.drop(pd_data[(pd_data[col]>d_max)].index)
    pd_data = pd_data.drop(pd_data[(pd_data[col]<d_min)].index)
#     pd_data[pd_data[col]>d_max]=np.nan
#     pd_data[pd_data[col]<d_min]=np.nan

print(pd_data.shape)

# step3, 去除取值只有一个值的列, -15col
columns = pd_data.columns.values.tolist()[1:-1] # 去除 id 和 label 的所有列
drop_list = []
for col in columns:
    num = len(pd_data[col].unique())
    if num == 1:
        drop_list.append(col)
pd_data = pd_data.drop(columns=drop_list)

print(pd_data.shape)
```

```
(482 ,48500)
x0 0.0 1.0
x1 -2.0 179.0
x2 -100.0 -100.0
x3 -1.001 2623.970399996324
x4 -1.001 300.0
```

```
                                                    x5 -1.001 300.0
                                                      x6 -100.0 0.0
                                                   x7 -100.0 100000.0
                                                          x8 -1.0 1.0
                                                       x9 -100.0 1.0
                                                        x10 0.0 23.0
                                              x11 -1.001 2.451644724085354
                                                x12 0.0 136.15200000003097
                                                          x13 0.0 2.0
                                                          x14 2.0 8.0
                                               x15 -1.0 112.63400000000547
                                                        x16 -1.0 25.0
                                                x17 0.0 2030.7250000000058
```

In [46]:

```
# import sweetviz as sv
# html = sv.analyze(pd_data, pairwise_analysis='off')
# html.show_html('drop_0.001_0.999_v2.html')
```

```
                                                    |          | [  0%]   00:
00 -> (? left)

Report drop_0.001_0.999_v2.html was generated! NOTEBOOK/COLAB USERS: t
he web browser MAY not pop up, regardless, the report IS saved in your
notebook/colab files.
```

In [25]:

```
print(len(drop_list))
print(drop_list)
print(pd_data.shape)
# print(pd_data)
print(pd_data.values.shape)
```

```
82
['x2', 'x8', 'x13', 'x18', 'x25', 'x32', 'x33', 'x42', 'x48', 'x55',
'x68', 'x70', 'x71', 'x72', 'x77', 'x79', 'x91', 'x96', 'x97', 'x100',
'x107', 'x110', 'x111', 'x113', 'x121', 'x130', 'x135', 'x141', 'x14
5', 'x152', 'x155', 'x178', 'x184', 'x198', 'x202', 'x207', 'x209', 'x
215', 'x224', 'x225', 'x228', 'x232', 'x239', 'x248', 'x249', 'x258',
'x261', 'x264', 'x265', 'x277', 'x279', 'x280', 'x281', 'x289', 'x29
2', 'x300', 'x319', 'x325', 'x341', 'x343', 'x347', 'x351', 'x367', 'x
371', 'x373', 'x384', 'x385', 'x389', 'x394', 'x400', 'x419', 'x427',
'x431', 'x436', 'x441', 'x442', 'x451', 'x452', 'x456', 'x461', 'x46
2', 'x475']
(43031, 400)
(43031, 400)
```

In [26]:

```
# # 2. 数据预处理
# pd_data = pd_raw_data.copy(deep=True)
# # step1, 去除 -1111, -26 row
# pd_data = pd_data.drop(pd_data[(pd_data['x0'] == -1111)].index)
# # step2, 去除 label = -1, -9265 row
# pd_data = pd_data.drop(pd_data[(pd_data['label'] == -1)].index)
# # step3, 去除取值只有一个值的列, -15col
# columns = pd_data.columns.values.tolist()
# drop_list = []
# for col in columns:
#     num = len(pd_data[col].unique())
#     if num == 1:
#         drop_list.append(col)
# pd_data = pd_data.drop(columns=drop_list)
# # step4, 填充缺失值
# pd_data = pd_data.fillna(pd_data.mean())

# print(min_max_map)
print(drop_list)

# 先创建并打开一个文本文件
file = open('min_max_map.txt', 'w')

# 遍历字典的元素，将每项元素的key和value分拆组成字符串，注意添加分隔符和换行符
for k,v in min_max_map.items():
    file.write(str(k)+' '+str(v[0]) + ' ' +str(v[1])+'\n')

# 注意关闭文件
file.close()

# 3. 数据转 numpy
np_data = pd_data.values
# print(np_data.shape[1]-2)
print(np_data.shape)
# 去除索引列
np_data = np_data[:, 1:]
# 去除文字列, 分成 input 和 output, output 降维到一维
np_data_input = np_data[:, :np_data.shape[1]-1]
np_data_input = np_data_input.astype(np.float64)

print(np_data_input.shape)
print(np_data_input)

np_data_output = np_data[:, np_data.shape[1]-1:]
np_data_output = np_data_output.astype(np.int64)
np_data_output = np.squeeze(np_data_output, axis=1)

print(np_data_output.shape)
print(np_data_output)

# 4. 数据归一化
# sc = StandardScaler()
# sc.fit(np_data_input)
# np_data_input_std = sc.transform(np_data_input)
test_size=0.2
X_train, X_test, Y_train, Y_test = train_test_split(np_data_input, np_data_outpu
logging.warning("Finish Process.")
```

['x2', 'x8', 'x13', 'x18', 'x25', 'x32', 'x33', 'x42', 'x48', 'x55',

```
'x68', 'x70', 'x71', 'x72', 'x77', 'x79', 'x91', 'x96', 'x97', 'x100',
'x107', 'x110', 'x111', 'x113', 'x121', 'x130', 'x135', 'x141', 'x14
5', 'x152', 'x155', 'x178', 'x184', 'x198', 'x202', 'x207', 'x209', 'x
215', 'x224', 'x225', 'x228', 'x232', 'x239', 'x248', 'x249', 'x258',
'x261', 'x264', 'x265', 'x277', 'x279', 'x280', 'x281', 'x289', 'x29
2', 'x300', 'x319', 'x325', 'x341', 'x343', 'x347', 'x351', 'x367', 'x
371', 'x373', 'x384', 'x385', 'x389', 'x394', 'x400', 'x419', 'x427',
'x431', 'x436', 'x441', 'x442', 'x451', 'x452', 'x456', 'x461', 'x46
2', 'x475']
(43031, 400)
(43031, 398)
[[   1.      -2.       0.    ... -100.       0.       1.   ]
 [   1.      90.       0.    ... -100.       0.       1.   ]
 [   1.      69.       0.    ... -100.       0.       7.   ]
 ...
 [   1.      -2.       2.    ... -100.       0.       1.   ]
 [   1.     179.       0.    ... -100.      60.42  -100.   ]
 [   1.      -2.      -1.001 ... -100.       0.       7.   ]]
(43031,)
[0 1 1 ... 0 1 0]

WARNING:root:Finish Process.
```

In [29]:

```
#########################################################
logging.warning("Begin Model.")
result = {}  # 保存一些结果
# lightGBM 模型
model_name = 'lightGBM'
version = 'v0.3_test'
from lightgbm import LGBMClassifier

# 5.1 模型定义
model = LGBMClassifier(
    verbose=1
)

# 5.2 模型训练
import time
time_start=time.time()
model.fit(X_train, Y_train)
time_end=time.time()
print('time cost',time_end-time_start,'s')
# 5.3 模型评估
result['score_train'] = model.score(X_train, Y_train)
result['score_test'] = model.score(X_test, Y_test)
# 5.4 模型保存
joblib.dump(model, f'./model/%s_%s_[%f, %f, %f, %s].model' %
            (model_name, version,
             result['score_test'], result['score_train'],
             test_size, model))
# 5.5 结果保存
print(result)
# with open(result_path, 'w') as fp:
#     json.dump(result, fp)
#########################################################
```

```
WARNING:root:Begin Model.

[LightGBM] [Info] Number of positive: 12297, number of negative: 22127
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overh
ead of testing was 0.031496 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 35246
[LightGBM] [Info] Number of data points in the train set: 34424, numbe
r of used features: 395
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.357222 -> initscore=
-0.587443
[LightGBM] [Info] Start training from score -0.587443
time cost 2.1668572425842285 s
{'score_train': 0.8687834069254009, 'score_test': 0.8416405251539445}
```

In [12]:

```
#######################################################
logging.warning("Begin Model.")
result = {}  # 保存一些结果
# XGBoost 模型
import xgboost as xgb

# 5.1 模型定义
model = xgb.XGBClassifier(
    verbose=10
)

# 5.2 模型训练
import time
time_start=time.time()
model.fit(X_train, Y_train)
time_end=time.time()
print('time cost',time_end-time_start,'s')
# 5.3 模型评估
result['score_train'] = model.score(X_train, Y_train)
result['score_test'] = model.score(X_test, Y_test)
# 5.4 模型保存
joblib.dump(model, f'./model/XGBoost_(%f, %f, %f).model'%
            (test_size, result['score_train'], result['score_test']))
# 5.5 结果保存
print(result)
# with open(result_path, 'w') as fp:
#     json.dump(result, fp)
#######################################################
```

```
WARNING:root:Begin Model.
/home/moegwjawe1/atec_project/train/lmf/lib/python3.6/site-packages/xg
boost/sklearn.py:1224: UserWarning: The use of label encoder in XGBCla
ssifier is deprecated and will be removed in a future release. To remo
ve this warning, do the following: 1) Pass option use_label_encoder=Fa
lse when constructing XGBClassifier object; and 2) Encode your labels
(y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[22:38:05] WARNING: ../src/learner.cc:576:
Parameters: { "verbose" } might not be used.

  This could be a false alarm, with some parameters getting used by la
nguage bindings but
  then being mistakenly passed down to XGBoost core, or some parameter
actually being used
  but getting flagged wrongly here. Please open an issue if you find a
ny such cases.


[22:38:06] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
time cost 44.68465065956116 s
{'score_train': 0.9202591215431094, 'score_test': 0.8263041710235854}
```

In [20]:

```
##########################################################
logging.warning("Begin Model.")
result = {}  # 保存一些结果
# CatBoost 模型
from  catboost  import  CatBoostClassifier

# 5.1 模型定义
model = CatBoostClassifier(
    verbose=10
)

# 5.2 模型训练
import time
time_start=time.time()
model.fit(X_train, Y_train)
time_end=time.time()
print('time cost',time_end-time_start,'s')
# 5.3 模型评估
result['score_train'] = model.score(X_train, Y_train)
result['score_test'] = model.score(X_test, Y_test)
# 5.4 模型保存
joblib.dump(model, f'./model/CatBoost_(%f, %f, %f).model'%
            (test_size, result['score_train'], result['score_test']))
# 5.5 结果保存
print(result)
# with open(result_path, 'w') as fp:
#     json.dump(result, fp)
##########################################################
```

```
WARNING:root:Begin Model.
```

In [21]:

```
# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # lr 模型
# from sklearn import linear_model

# # 5.1 模型定义
# model = linear_model.LogisticRegression(
#     C=1e5, max_iter=10000000, tol=0.0001,
#     verbose=10
# )

# # 5.2 模型训练
# import time
# time_start=time.time()
# model.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost',time_end-time_start,'s')
# # 5.3 模型评估
# result['score_train'] = model.score(X_train, Y_train)
# result['score_test'] = model.score(X_test, Y_test)
# # 5.4 模型保存
# joblib.dump(model, f'./model/lr_(%f, %f, %f).model'%
#             (test_size, result['score_train'], result['score_test']))
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [22]:

```
# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # LinearSVC 模型
# from sklearn.svm import LinearSVC

# # 5.1 模型定义
# model = LinearSVC(
#     verbose=10
# )

# # 5.2 模型训练
# import time
# time_start=time.time()
# model.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost',time_end-time_start,'s')
# # 5.3 模型评估
# result['score_train'] = model.score(X_train, Y_train)
# result['score_test'] = model.score(X_test, Y_test)
# # 5.4 模型保存
# joblib.dump(model, f'./model/LinearSVC_(%f, %f, %f).model'%
#             (test_size, result['score_train'], result['score_test']))
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [23]:

```
# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # SVC 模型
# from sklearn.svm import SVC

# # 5.1 模型定义
# model = SVC(
#     verbose=10
# )

# # 5.2 模型训练
# import time
# time_start=time.time()
# model.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost',time_end-time_start,'s')
# # 5.3 模型评估
# result['score_train'] = model.score(X_train, Y_train)
# result['score_test'] = model.score(X_test, Y_test)
# # 5.4 模型保存
# joblib.dump(model, f'./model/SVC_(%f, %f, %f).model'%
#             (test_size, result['score_train'], result['score_test']))
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [24]:

```
# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # DecisionTreeClassifier 模型
# from sklearn.neighbors import KNeighborsClassifier

# # 5.1 模型定义
# model = KNeighborsClassifier()

# # 5.2 模型训练
# import time
# time_start=time.time()
# model.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost',time_end-time_start,'s')
# # 5.3 模型评估
# result['score_train'] = model.score(X_train, Y_train)
# result['score_test'] = model.score(X_test, Y_test)
# # 5.4 模型保存
# joblib.dump(model, f'./model/knc_(%f, %f, %f).model'%
#             (test_size, result['score_train'], result['score_test']))
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [25]:

```
# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # DecisionTreeClassifier 模型
# from sklearn.tree import DecisionTreeClassifier

# # 5.1 模型定义
# model = DecisionTreeClassifier()

# # 5.2 模型训练
# import time
# time_start=time.time()
# model.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost',time_end-time_start,'s')
# # 5.3 模型评估
# result['score_train'] = model.score(X_train, Y_train)
# result['score_test'] = model.score(X_test, Y_test)
# # 5.4 模型保存
# joblib.dump(model, f'./model/dtc_(%f, %f, %f).model'%
#             (test_size, result['score_train'], result['score_test']))
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [26]:

```
#######################################################
logging.warning("Begin Model.")
result = {}  # 保存一些结果
# GradientBoostingClassifier 模型
from sklearn.ensemble import GradientBoostingClassifier

# 5.1 模型定义
model = GradientBoostingClassifier(
    verbose=10
)

# 5.2 模型训练
import time
time_start=time.time()
model.fit(X_train, Y_train)
time_end=time.time()
print('time cost',time_end-time_start,'s')
# 5.3 模型评估
result['score_train'] = model.score(X_train, Y_train)
result['score_test'] = model.score(X_test, Y_test)
# 5.4 模型保存
joblib.dump(model, f'./model/gbc_(%f, %f, %f).model'%
            (test_size, result['score_train'], result['score_test']))
# 5.5 结果保存
print(result)
# with open(result_path, 'w') as fp:
#     json.dump(result, fp)
#######################################################
```

```
WARNING:root:Begin Model.
      Iter       Train Loss   Remaining Time
         1           1.2435            2.83m
         2           1.1950            2.81m
         3           1.1558            2.73m
         4           1.1147            2.68m
         5           1.0871            2.63m
         6           1.0568            2.59m
         7           1.0318            2.55m
         8           1.0064            2.51m
         9           0.9909            2.48m
        10           0.9707            2.45m
        11           0.9562            2.42m
        12           0.9414            2.39m
        13           0.9305            2.37m
        14           0.9183            2.34m
        15           0.9081            2.31m
        16           0.9007            2.28m
        17           0.8950            2.25m
        18           0.8875            2.23m
        19           0.8806            2.20m
        20           0.8768            2.17m
        21           0.8708            2.14m
        22           0.8672            2.11m
        23           0.8623            2.09m
        24           0.8580            2.06m
        25           0.8543            2.03m
        26           0.8515            2.00m
        27           0.8483            1.98m
```

28	0.8456	1.95m
29	0.8427	1.93m
30	0.8402	1.90m
31	0.8376	1.87m
32	0.8356	1.84m
33	0.8334	1.82m
34	0.8309	1.79m
35	0.8294	1.76m
36	0.8271	1.73m
37	0.8258	1.70m
38	0.8242	1.68m
39	0.8223	1.65m
40	0.8207	1.62m
41	0.8196	1.60m
42	0.8185	1.57m
43	0.8166	1.54m
44	0.8151	1.51m
45	0.8141	1.49m
46	0.8132	1.46m
47	0.8115	1.43m
48	0.8106	1.40m
49	0.8094	1.38m
50	0.8085	1.35m
51	0.8077	1.32m
52	0.8064	1.29m
53	0.8055	1.27m
54	0.8047	1.24m
55	0.8040	1.21m
56	0.8028	1.19m
57	0.8010	1.16m
58	0.8004	1.13m
59	0.7988	1.10m
60	0.7982	1.08m
61	0.7977	1.05m
62	0.7971	1.02m
63	0.7961	59.81s
64	0.7955	58.19s
65	0.7946	56.56s
66	0.7934	54.96s
67	0.7929	53.35s
68	0.7920	51.73s
69	0.7912	50.11s
70	0.7908	48.50s
71	0.7902	46.89s
72	0.7896	45.26s
73	0.7891	43.65s
74	0.7887	42.04s
75	0.7883	40.44s
76	0.7876	38.82s
77	0.7872	37.23s
78	0.7869	35.63s
79	0.7861	34.02s
80	0.7856	32.39s
81	0.7848	30.78s
82	0.7839	29.16s
83	0.7825	27.54s
84	0.7819	25.92s
85	0.7816	24.30s
86	0.7811	22.68s
87	0.7807	21.06s
88	0.7802	19.45s

```
        89           0.7796          17.82s
        90           0.7789          16.20s
        91           0.7782          14.58s
        92           0.7779          12.96s
        93           0.7775          11.34s
        94           0.7771           9.72s
        95           0.7769           8.10s
        96           0.7767           6.48s
        97           0.7762           4.86s
        98           0.7759           3.24s
        99           0.7756           1.62s
       100           0.7753           0.00s
time cost 162.2033224105835 s
{'score_train': 0.8312819454949689, 'score_test': 0.8366336633663366}
```

In [27]:

```
#########################################################
logging.warning("Begin Model.")
result = {}  # 保存一些结果
# RandomForestClassifier 模型
from sklearn.ensemble import RandomForestClassifier

# 5.1 模型定义
model = RandomForestClassifier(
    verbose=10
)

# 5.2 模型训练
import time
time_start=time.time()
model.fit(X_train, Y_train)
time_end=time.time()
print('time cost',time_end-time_start,'s')
# 5.3 模型评估
result['score_train'] = model.score(X_train, Y_train)
result['score_test'] = model.score(X_test, Y_test)
# 5.4 模型保存
joblib.dump(model, f'./model/rfc_(%f, %f, %f).model'%
            (test_size, result['score_train'], result['score_test']))
# 5.5 结果保存
print(result)
# with open(result_path, 'w') as fp:
#     json.dump(result, fp)
#########################################################
```

```
WARNING:root:Begin Model.

building tree 1 of 100

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remainin
g:    0.0s

building tree 2 of 100

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.5s remainin
g:    0.0s

building tree 3 of 100

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.7s remainin
g:    0.0s

building tree 4 of 100
```

In [28]:

```
# # 调参
# from sklearn.model_selection import GridSearchCV

# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # lightGBM 模型
# from lightgbm import LGBMClassifier

# # 5.1 模型定义
# param_grid = {
#     'n_estimators': [10, 100, 1000, 10000]
# }
# grid_search = GridSearchCV(
#     LGBMClassifier(
#         learning_rate=0.1,
#     ),
#     param_grid,
#     scoring='roc_auc', # or 'recall'
# #     n_jobs=-1, # 应该为 CPU 内核数 -1, 但是不能加, 否则 memory-leaks,
#     cv=5,
#     verbose=10
# )

# # 5.2 模型训练
# import time
# time_start = time.time()
# grid_search.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost', time_end - time_start, 's')

# # 5.3 训练记录
# result['acc0'] = grid_search.score(X_test, Y_test)
# result['best_params0'] = grid_search.best_params_
# result['best_score0'] = grid_search.best_score_
# result['n_estimators'] = grid_search.best_params_['n_estimators']

# # 5.3 模型评估
# result['score_train'] = model.score(X_train, Y_train)
# result['score_test'] = model.score(X_test, Y_test)
# # 5.4 模型保存
# joblib.dump(model, f'./model/lightGBM_(%f, %f, %f).model' %
#             (test_size, result['score_train'], result['score_test']))
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [29]:

```
# # 调参
# from sklearn.model_selection import GridSearchCV

# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # lightGBM 模型
# from lightgbm import LGBMClassifier

# # 5.1 参数定义
# param_grid = {
# #     'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10]
#     'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
#     # recall: 11
#     # roc_auc: 9
# }
# # 5.2 模型定义
# grid_search = GridSearchCV(
#     LGBMClassifier(
#         learning_rate=0.1,
#     ),
#     param_grid,
#     scoring='roc_auc', # 'recall', 'roc_auc'
# #     n_jobs=-1, # 应该为 CPU 内核数 -1，但是不能加，否则 memory-leaks,
#     cv=5,
#     verbose=10
# )
# # 5.3 模型训练
# import time
# time_start = time.time()
# grid_search.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost', time_end - time_start, 's')
# # 5.4 训练记录
# result['acc0'] = grid_search.score(X_test, Y_test)
# result['best_params0'] = grid_search.best_params_
# result['best_score0'] = grid_search.best_score_
# result['max_depth'] = grid_search.best_params_['max_depth']
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [ ]:

```

```

In [30]:

```
# # 调参
# from sklearn.model_selection import GridSearchCV

# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # lightGBM 模型
# from lightgbm import LGBMClassifier

# # 5.1 参数定义
# param_grid = {
#     'max_depth': [9], # < 1
#     'num_leaves': [340], # < 50
#     'min_data_in_leaf': [50], # <10
#     'max_bin': [200], # <50
#     'feature_fraction': [0.7], # <0.1
#     'bagging_fraction': [0.8], # <0.2
#     'bagging_freq': [2], # <2
#     'lambda_l1': [0.25], # <0.2
#     'lambda_l2': [0.1], # <0.1
#     'min_split_gain': [0.9], # <0.1
# #     'n_estimators': [],
# }
#     # roc_auc: {'max_depth': 9, 'num_leaves': 1002}
#     # recall: {'max_depth': 12, 'num_leaves': 1002}
#     # roc_auc: {'max_depth': 9, 'num_leaves': 502}
#     # recall: {'max_depth': 12, 'num_leaves': 1002}
#     # roc_auc: {'max_depth': 9, 'num_leaves': 500}
#     # recall: {'max_depth': 12, 'num_leaves': 550}
#     # roc_auc: {'max_depth': 9, 'num_leaves': 400}
#     # recall: {'max_depth': 12, 'num_leaves': 530}
#     # roc_auc:
#     # {'max_depth': 9, 'num_leaves': 350}; range(350,450,10),
# """
# {'max_bin': 200, 'max_depth': 9, 'min_data_in_leaf': 50, 'num_leaves': 340}
# {'bagging_fraction': 0.8, 'bagging_freq': 2, 'feature_fraction': 0.7, 'max_bi
# {'bagging_fraction': 0.8, 'bagging_freq': 2, 'feature_fraction': 0.7, 'lambda

#     'max_depth': [9],
#     'num_leaves': [350],
#     'min_data_in_leaf': [10, 20, 30, 40, 50],
#     'max_bin': [10, 50, 100, 200, 300],
#     {'max_bin': 200, 'max_depth': 9, 'min_data_in_leaf': 50, 'num_leaves': 35

#     'max_depth': [9],
#     'num_leaves': [350],
#     'min_data_in_leaf': [50],
#     'max_bin': [200],
#     'feature_fraction': [0.7, 0.8, 0.9],
#     'bagging_fraction': [0.7, 0.8, 0.9],
#     {'bagging_fraction': 0.7, 'feature_fraction': 0.7, 'max_bin': 200, 'max_d

#     'max_depth': [9],
#     'num_leaves': [350],
#     'min_data_in_leaf': [50],
#     'max_bin': [200],
#     'feature_fraction': [0.7],
#     'bagging_fraction': [0.7],
#     'bagging_freq': [3, 4, 5],
```

```
#     {'bagging_fraction': 0.7, 'bagging_freq': 3, 'feature_fraction': 0.7, 'ma

#     'max_depth': [9],
#     'num_leaves': [350],
#     'min_data_in_leaf': [50],
#     'max_bin': [200],
#     'feature_fraction': [0.7],
#     'bagging_fraction': [0.7],
#     'bagging_freq': [3],
#     'lambda_l1': [0, 0.1, 0.2],
#     'lambda_l2': [0, 0.1, 0.2],
#     {'bagging_fraction': 0.7, 'bagging_freq': 3, 'feature_fraction': 0.7, 'la

#     'max_depth': [9],
#     'num_leaves': [350],
#     'min_data_in_leaf': [50],
#     'max_bin': [200],
#     'feature_fraction': [0.7],
#     'bagging_fraction': [0.7],
#     'bagging_freq': [3],
#     'lambda_l1': [0.2],
#     'lambda_l2': [0.1],
#     'min_split_gain': [0.7, 0.8, 0.9],
#     {'bagging_fraction': 0.7, 'bagging_freq': 3, 'feature_fraction': 0.7, 'la
# """



# # 5.2 模型定义
# grid_search = GridSearchCV(
#     LGBMClassifier(
#         learning_rate=0.1,
#     ),
#     param_grid,
#     scoring='roc_auc', # 'recall', 'roc_auc'
# #     n_jobs=-1, # 应该为 CPU 内核数 -1，但是不能加，否则 memory-leaks,
#     cv=5,
#     verbose=10
# )
# # 5.3 模型训练
# import time
# time_start = time.time()
# grid_search.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost', time_end - time_start, 's')
# # 5.4 训练记录
# result['acc0'] = grid_search.score(X_test, Y_test)
# result['best_params0'] = grid_search.best_params_
# result['best_score0'] = grid_search.best_score_
# result['max_depth'] = grid_search.best_params_['max_depth']
# result['num_leaves'] = grid_search.best_params_['num_leaves']
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ##########################################################
```

In [31]:

```
# ########################################################
# logging.warning("Begin Model.")
# result = {}  # 保存一些结果
# # lightGBM 模型
# from lightgbm import LGBMClassifier

# """
#  'max_depth': [9], # < 1
#     'num_leaves': [340], # < 50
#     'min_data_in_leaf': [50], # <10
#     'max_bin': [200], # <50
#     'feature_fraction': [0.7], # <0.1
#     'bagging_fraction': [0.8], # <0.2
#     'bagging_freq': [2], # <2
#     'lambda_l1': [0.25], # <0.2
#     'lambda_l2': [0.1], # <0.1
#     'min_split_gain': [0.9], # <0.1
# """

# # 5.1 模型定义
# # 过拟合，避免过拟合
# model = LGBMClassifier(
#     max_depth=9,
#     num_leaves=340,
#     min_data_in_leaf=50,
#     max_bin=200,
#     feature_fraction=0.7,
#     bagging_fraction=0.8,
#     bagging_freq=2,
#     lambda_l1=0.25,
#     lambda_l2=0.1,
# #     min_split_gain=0.9, # 过拟合

#     learning_rate=0.01,
#     n_estimators=10000,

#     verbose=10
# )

# # 5.2 模型训练
# import time
# time_start=time.time()
# model.fit(X_train, Y_train)
# time_end=time.time()
# print('time cost',time_end-time_start,'s')
# # 5.3 模型评估
# result['score_train'] = model.score(X_train, Y_train)
# result['score_test'] = model.score(X_test, Y_test)
# # 5.4 模型保存
# joblib.dump(model, f'./model/lightGBM_(%f, %f, %f).model'%
#             (test_size, result['score_train'], result['score_test']))
# # 5.5 结果保存
# print(result)
# # with open(result_path, 'w') as fp:
# #     json.dump(result, fp)
# ########################################################
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```