In [7]:

```
min max map={}
   file = open('min max map.txt', 'r')
   for line in file.readlines():
3
4
       line = line.strip()
5
       line = line.split(' ')
       min_max_map[line[0]] = (line[1], line[2])
6
7
       # 依旧是关闭文件
8
   file.close()
9
10 print(min max map)
```

```
{'x0': ('0.0', '1.0'), 'x1': ('-2.0', '179.0'), 'x2': ('-100.0', '-100.0')}
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', '112.6340000000547'), '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 -*-
2
3
 4
  import os
5
   import sys
   import json
6
7
   import logging
8
9
   sys.path.append('/home/moegwjawe1/atec project/train/lmf/lib/python3.6/site-pack
10
11
   import numpy as np
   import pandas as pd
12
13
14 from sklearn.preprocessing import StandardScaler
15 from sklearn.model selection import train test split
   from sklearn.model_selection import GridSearchCV
16
17
18
   import joblib
19
   input data path = '/home/admin/workspace/job/input/train.jsonl'
20
   output model path = '/home/admin/workspace/job/output/train.model'
21
   result path = '/home/admin/workspace/job/output/result.json'
22
23
24
   input_data_path = '/mnt/atec/train.jsonl'
25
26 logging.warning("Begin Train.")
   # 1. 读取训练数据进行训练
27
28 import time
29 time start = time.time()
30 | # pd_raw_data = pd.read_json(input_data_path, encoding='utf-8', lines=True, nrow
31 pd raw data = pd.read json(input data path, encoding='utf-8', lines=True)
32 time end=time.time()
33 print('time cost', time end - time start, 's')
34 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
 1
 2
   pd data = pd raw data.copy(deep=True)
 3
 4
   # step2, 去除 label = -1, -9265 row
 5
   pd data = pd data.drop(pd data[(pd data['label'] == -1)].index)
 6
   # print(pd data)
   # print(pd data.describe())
7
   pd_data = pd_data.drop(columns=['memo_polish'],axis=1)
8
9
   # 只保留 int 列 229
10
11
   # pd data = pd data.drop(pd data.columns[np.where((pd data.dtypes!='int64').vall
12
13
14
   min max map={}
15
   # step3, 去除取值只有一个值的列, -15col
16
   columns = pd data.columns.values.tolist()[1:-1] # 去除 id 和 label 的所有列
17
18
   drop list = []
19
   for col in columns:
20
       num = len(pd data[col].unique())
21
       if num == 1:
22
           drop list.append(col)
23
24
   # 去除离群数据,或者离群数据置空
25
   print(pd data.shape)
26
   columns = pd data.columns.values.tolist()[1:-1] # 去除 id 和 label 的所有列
27
   for col in columns:
28
       if pd data.dtypes[col]=='float64':
29
           d \min, d \max = pd data[col].quantile([0.0001, 0.9999])
30
       elif pd data.dtypes[col]=='int64':
31
           d \min, d \max = pd data[col].quantile([0.001, 0.999])
32
       else:
33
           print("=======")
34
       print(col,d min,d max)
       min_max_map[col]=(d_min,d_max)
35
36
       pd data = pd data.drop(pd data[(pd data[col]>d max)].index)
37
       pd data = pd data.drop(pd data[(pd data[col]<d min)].index)</pre>
         pd data[pd data[col]>d_max]=np.nan
38
   #
39
         pd data[pd data[col]<d min]=np.nan
40
41
   print(pd_data.shape)
42
   # step3, 去除取值只有一个值的列, -15col
43
   columns = pd_data.columns.values.tolist()[1:-1] # 去除 id 和 label 的所有列
44
45
   drop list = []
46
   for col in columns:
47
       num = len(pd data[col].unique())
48
       if num == 1:
49
           drop list.append(col)
50
   pd data = pd data.drop(columns=drop list)
51
52
  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 1000000.0

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
```

] [

0 %]

00:

In [46]:

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

00 -> (? left)

Report drop_0.001_0.999_v2.html was generated! NOTEBOOK/COLAB USERS: the 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', 'x215', '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', 'x371', '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
 1
   # pd data = pd raw data.copy(deep=True)
 2
 3 # # 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
 5
  # pd data = pd data.drop(pd data[(pd data['label'] == -1)].index)
 7
   # # step3, 去除取值只有一个值的列, -15col
8 # columns = pd data.columns.values.tolist()
 9
   # drop list = []
10 # for col in columns:
         num = len(pd data[col].unique())
11
   #
         if num == 1:
12
13 #
             drop_list.append(col)
14 # pd data = pd data.drop(columns=drop_list)
15 # # step4, 填充缺失值
16
   # pd data = pd data.fillna(pd data.mean())
17
18
   # print(min max map)
19
   print(drop list)
20
   # 先创建并打开一个文本文件
21
22
   file = open('min max map.txt', 'w')
23
24
   # 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
25
   for k,v in min max map.items():
26
       file.write(str(k)+' '+str(v[0]) + ' ' +str(v[1])+' \setminus n')
27
   # 注意关闭文件
28
29
   file.close()
30
   # 3. 数据转 numpy
31
32 np data = pd data.values
33 # print(np data.shape[1]-2)
34 print(np data.shape)
35 # 去除索引列
36 np data = np data[:, 1:]
   # 去除文字列,分成 input 和 output, output 降维到一维
37
38 np data input = np data[:, :np data.shape[1]-1]
39
   np data input = np data input.astype(np.float64)
40
41
   print(np_data_input.shape)
42
   print(np data input)
43
44
  np_data_output = np_data[:, np_data.shape[1]-1:]
45
   np_data_output = np_data_output.astype(np.int64)
46
   np_data_output = np.squeeze(np_data_output, axis=1)
47
48 print(np_data_output.shape)
49
   print(np_data_output)
50
51 # 4. 数据归一化
52 # sc = StandardScaler()
53 # sc.fit(np data input)
54 # np data input std = sc.transform(np data input)
55 test size=0.2
56
   X_train, X_test, Y_train, Y_test = train_test_split(np_data_input, np_data_outpu
57
   logging.warning("Finish Process.
```

```
'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)
                         0.
   1.
              -2.
                                ... -100.
                                                 0.
                                                           1.
                                                                 ]
] ]
[
     1.
              90.
                         0.
                               ... -100.
                                                 0.
                                                           1.
                                                                 ]
                        0.
                               ... -100.
              69.
                                                 0.
                                                           7.
     1.
                                                                 ]
 [
 . . .
                        2. ... -100.
0. ... -100.
             -2.
     1.
                                                 0.
 [
                                                           1.
                                                                 ]
   1.
             179.
                                                60.42 -100.
                                                               ]
 [
                        -1.001 ... -100.
 ſ
    1.
             -2.
                                                0. 7.
                                                                 ]]
(43031,)
[0 \ 1 \ 1 \ \dots \ 0 \ 1 \ 0]
```

WARNING:root:Finish Process.

```
In [29]:
```

```
logging.warning("Begin Model.")
 2
3 result = {} # 保存一些结果
 4 # lightGBM 模型
5 model name = 'lightGBM'
6 version = 'v0.3 test'
7
   from lightgbm import LGBMClassifier
8
  # 5.1 模型定义
9
10 model = LGBMClassifier(
11
      verbose=1
12
13
14 # 5.2 模型训练
15 import time
16 time start=time.time()
17 model.fit(X train, Y train)
18 time end=time.time()
19
   print('time cost', time end-time start, 's')
20 # 5.3 模型评估
21 result['score_train'] = model.score(X_train, Y_train)
22 result['score test'] = model.score(X test, Y test)
  # 5.4 模型保存
23
24
   joblib.dump(model, f'./model/%s_%s_[%f, %f, %f, %s].model' %
25
             (model name, version,
26
              result['score test'], result['score train'],
27
              test size, model))
28 # 5.5 结果保存
29 print(result)
30 # with open(result path, 'w') as fp:
        json.dump(result, fp)
31
   32
```

WARNING:root:Begin Model.

```
[LightGBM] [Info] Number of positive: 12297, number of negative: 22127 [LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead 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, number 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}
```

```
logging.warning("Begin Model.")
2
3 | result = {} # 保存一些结果
  # XGBoost 模型
4
5
  import xqboost as xqb
6
  # 5.1 模型定义
7
  model = xgb.XGBClassifier(
8
9
      verbose=10
10
11
  # 5.2 模型训练
12
13 import time
14 time start=time.time()
15 model.fit(X train, Y train)
16 time end=time.time()
17 print('time cost', time end-time start, 's')
18 # 5.3 模型评估
19 result['score_train'] = model.score(X_train, Y_train)
  result['score test'] = model.score(X test, Y test)
20
21 # 5.4 模型保存
  joblib.dump(model, f'./model/XGBoost (%f, %f, %f).model'%
22
23
             (test size, result['score train'], result['score test']))
24 # 5.5 结果保存
25 print(result)
26 # with open(result path, 'w') as fp:
27
        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:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if 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.")
2
3 result = {} # 保存一些结果
  # CatBoost 模型
5
  from catboost import CatBoostClassifier
  # 5.1 模型定义
7
8 model = CatBoostClassifier(
9
      verbose=10
10)
11
12 # 5.2 模型训练
13 import time
14 time start=time.time()
15 model.fit(X train, Y train)
16 time end=time.time()
17 print('time cost', time end-time start, 's')
18 # 5.3 模型评估
19 result['score_train'] = model.score(X_train, Y_train)
20 result['score_test'] = model.score(X_test, Y_test)
21 # 5.4 模型保存
22 joblib.dump(model, f'./model/CatBoost (%f, %f, %f).model'%
23
             (test_size, result['score_train'], result['score_test']))
24 # 5.5 结果保存
25 print(result)
26 # with open(result path, 'w') as fp:
        json.dump(result, fp)
27
```

WARNING:root:Begin Model.

In [21]:

```
# logging.warning("Begin Model.")
2
3 # result = {} # 保存一些结果
  # # 1r 模型
5
  # from sklearn import linear model
  # # 5.1 模型定义
7
  # model = linear model.LogisticRegression(
8
9
       C=1e5, max iter=10000000, tol=0.0001,
  #
        verbose=10
10
  # )
11
12
13 # # 5.2 模型训练
14 # import time
15 # time start=time.time()
16 # model.fit(X train, Y train)
17 # time end=time.time()
18 # print('time cost', time end-time start, 's')
19 # # 5.3 模型评估
20 | # result['score train'] = model.score(X train, Y train)
21 # result['score test'] = model.score(X test, Y test)
22 # # 5.4 模型保存
23  # joblib.dump(model, f'./model/lr (%f, %f, %f).model'%
24 #
               (test_size, result['score_train'], result['score_test']))
25 # # 5.5 结果保存
26 # print(result)
27 # # with open(result path, 'w') as fp:
28 # #
         json.dump(result, fp)
```

In [22]:

```
# logging.warning("Begin Model.")
2
3 # result = {} # 保存一些结果
  ## LinearSVC 模型
  # from sklearn.svm import LinearSVC
  # # 5.1 模型定义
7
8 # model = LinearSVC(
9
      verbose=10
10 # )
11
12 # # 5.2 模型训练
13 # import time
14 # time start=time.time()
15 # model.fit(X train, Y train)
16 # time end=time.time()
17 # print('time cost', time end-time start, 's')
18 # # 5.3 模型评估
19 # result['score_train'] = model.score(X_train, Y_train)
20 # result['score_test'] = model.score(X_test, Y_test)
21 # # 5.4 模型保存
22 # joblib.dump(model, f'./model/LinearSVC (%f, %f, %f).model'%
              (test size, result['score train'], result['score test']))
23 #
24 # # 5.5 结果保存
25 # print(result)
26 # # with open(result path, 'w') as fp:
27 # #
         json.dump(result, fp)
```

In [23]:

```
2
  # logging.warning("Begin Model.")
3 # result = {} # 保存一些结果
  # # SVC 模型
4
   # from sklearn.svm import SVC
5
6
  # # 5.1 模型定义
7
  # model = SVC(
8
9
       verbose=10
   #)
10
11
  # # 5.2 模型训练
12
13 # import time
14 # time start=time.time()
15 # model.fit(X train, Y train)
16 # time end=time.time()
17 # print('time cost', time end-time start, 's')
18 # # 5.3 模型评估
19 # result['score train'] = model.score(X train, Y train)
20 # result['score test'] = model.score(X test, Y test)
21 # # 5.4 模型保存
22 # joblib.dump(model, f'./model/SVC (%f, %f, %f).model'%
23 #
               (test size, result['score train'], result['score test']))
24
  # # 5.5 结果保存
25 # print(result)
26 # # with open(result path, 'w') as fp:
27
  # #
         json.dump(result, fp)
```

In [24]:

```
# logging.warning("Begin Model.")
  # result = {} # 保存一些结果
 3
  ## DecisionTreeClassifier 模型
 4
  # from sklearn.neighbors import KNeighborsClassifier
5
7
   # # 5.1 模型定义
  # model = KNeighborsClassifier()
8
9
10 # # 5.2 模型训练
  # import time
11
12 # time start=time.time()
13 # model.fit(X train, Y train)
  # time end=time.time()
14
  # print('time cost', time end-time start, 's')
15
16 # # 5.3 模型评估
17
  # result['score train'] = model.score(X train, Y train)
18 # result['score_test'] = model.score(X_test, Y_test)
  # # 5.4 模型保存
19
20 # joblib.dump(model, f'./model/knc (%f, %f, %f).model'%
21
               (test size, result['score train'], result['score test']))
22
  |# # 5.5 结果保存
23 # print(result)
24 | # # with open(result_path, 'w') as fp:
  # #
25
         ison.dump(result, fp)
```

```
1
2 # logging.warning("Begin Model.")
3 # result = {} # 保存一些结果
4
  ## DecisionTreeClassifier 模型
5 # from sklearn.tree import DecisionTreeClassifier
   # # 5.1 模型定义
7
  # model = DecisionTreeClassifier()
8
9
10 # # 5.2 模型训练
11 # import time
12 # time start=time.time()
13 # model.fit(X train, Y train)
14 | # time end=time.time()
15 # print('time cost', time_end-time_start, 's')
16 # # 5.3 模型评估
17 # result['score train'] = model.score(X train, Y train)
18 # result['score test'] = model.score(X test, Y test)
19 # # 5.4 模型保存
20 # joblib.dump(model, f'./model/dtc (%f, %f, %f).model'%
               (test size, result['score train'], result['score test']))
21 #
22 # # 5.5 结果保存
23 # print(result)
24 | # # with open(result_path, 'w') as fp:
25 # #
         json.dump(result, fp)
```

In [26]:

```
1
  logging.warning("Begin Model.")
2
3 result = {} # 保存一些结果
  # GradientBoostingClassifier 模型
5
  from sklearn.ensemble import GradientBoostingClassifier
  # 5.1 模型定义
7
8 model = GradientBoostingClassifier(
9
      verbose=10
10)
11
12 # 5.2 模型训练
13 import time
14 time start=time.time()
15 model.fit(X train, Y train)
16 time end=time.time()
17 print('time cost', time end-time start, 's')
18 # 5.3 模型评估
19 result['score_train'] = model.score(X_train, Y_train)
20 result['score test'] = model.score(X test, Y test)
21 # 5.4 模型保存
22 joblib.dump(model, f'./model/gbc_(%f, %f, %f).model'%
             (test_size, result['score_train'], result['score_test']))
23
24 # 5.5 结果保存
25 print(result)
26 # with open(result path, 'w') as fp:
27
        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	2.73m
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
	313100	_ 1 J O.M.

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	1.15m
58	0.8010	1.13m
59	0.7988	1.13m
60	0.7982	1.10m
61	0.7977	1.05m
62	0.7977	1.03m
63	0.7971	59.81s
64	0.7955	59.81S 58.19s
	0.7935	56.56s
65		
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

^{[&#}x27;caoro +rain'. 0 0212010/5/0/0600 'caoro +oc+'. 0 0266226622662

```
In [27]:
 1
   logging.warning("Begin Model.")
 2
 3 result = {} # 保存一些结果
   # RandomForestClassifier 模型
 5
   from sklearn.ensemble import RandomForestClassifier
   # 5.1 模型定义
 7
   model = RandomForestClassifier(
 8
 9
      verbose=10
10
11
   # 5.2 模型训练
12
13 import time
14 time start=time.time()
15 model.fit(X train, Y train)
16 time end=time.time()
17 print('time cost', time end-time start, 's')
18 # 5.3 模型评估
19 result['score_train'] = model.score(X_train, Y_train)
20 result['score test'] = model.score(X test, Y test)
21 # 5.4 模型保存
   joblib.dump(model, f'./model/rfc_(%f, %f, %f).model'%
22
              (test size, result['score train'], result['score test']))
23
24 # 5.5 结果保存
25 print(result)
26 # with open(result path, 'w') as fp:
27
         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.
```

```
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
2
3
   4
5
   # logging.warning("Begin Model.")
6 # result = {} # 保存一些结果
7
  |# # lightGBM 模型
  # from lightgbm import LGBMClassifier
8
10 # # 5.1 模型定义
  # param grid = {
11
         'n_estimators': [10, 100, 1000, 10000]
12
13 | # }
14
  # grid search = GridSearchCV(
15 #
       LGBMClassifier(
16
  #
           learning rate=0.1,
  #
17
       ),
18 #
       param grid,
19 #
        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,
20 # #
21 #
        cv=5,
22 #
        verbose=10
23 # )
24
25 # # 5.2 模型训练
26 # import time
   # time start = time.time()
27
28 # grid search.fit(X train, Y train)
29 # time end=time.time()
30 # print('time cost', time_end - time_start, 's')
31
32 # # 5.3 训练记录
33 # result['acc0'] = grid search.score(X test, Y test)
  # result['best_params0'] = grid_search.best_params_
34
35 # result['best_score0'] = grid_search.best_score_
36 | # result['n estimators'] = grid search.best params ['n estimators']
37
38 # # 5.3 模型评估
39 | # result['score_train'] = model.score(X_train, Y_train)
40 # result['score test'] = model.score(X test, Y test)
   # # 5.4 模型保存
41
  # joblib.dump(model, f'./model/lightGBM (%f, %f, %f).model' %
42
43 #
                (test size, result['score train'], result['score test']))
44 # # 5.5 结果保存
45 # print(result)
46 # # with open(result_path, 'w') as fp:
47 # #
          json.dump(result, fp)
```

```
In [29]:
```

```
# # 调参
   # from sklearn.model selection import GridSearchCV
2
3
   4
5
   # logging.warning("Begin Model.")
  | # result = {} # 保存一些结果
6
7
   |# # lightGBM 模型
   # from lightgbm import LGBMClassifier
8
9
10 # # 5.1 参数定义
   # param grid = {
11
          'max depth': [2, 3, 4, 5, 6, 7, 8, 9, 10]
12
13
   #
         'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
14
       # recall: 11
15 #
        # roc auc: 9
  # }
16
  # # 5.2 模型定义
17
18  # grid search = GridSearchCV(
19 #
       LGBMClassifier(
20
   #
            learning rate=0.1,
  #
21
        ),
22 #
        param grid,
        scoring='roc auc', # 'recall', 'roc auc'
23
24
  # #
         n jobs=-1, # 应该为 CPU 内核数 -1, 但是不能加, 否则 memory-leaks,
25 #
        cv=5,
26
   #
        verbose=10
  # )
27
28 # # 5.3 模型训练
29 # import time
30 # time start = time.time()
31 # grid search.fit(X train, Y train)
32 # time end=time.time()
33 # print('time cost', time end - time start, 's')
34
  # # 5.4 训练记录
35 | # result['acc0'] = grid_search.score(X_test, Y_test)
36 # result['best params0'] = grid search.best params
37 # result['best score0'] = grid search.best score
38 | # result['max_depth'] = grid_search.best_params_['max_depth']
39 # # 5.5 结果保存
40 # print(result)
41 # # with open(result path, 'w') as fp:
42 # #
          json.dump(result, fp)
```

```
In [ ]:
```

1

```
In [30]:
```

```
# # 调参
1
 2
   # from sklearn.model selection import GridSearchCV
3
   4
   # logging.warning("Begin Model.")
   | # result = {} # 保存一些结果
   |# # lightGBM 模型
7
   # from lightgbm import LGBMClassifier
8
9
   # # 5.1 参数定义
10
11
   # param grid = {
         'max depth': [9], # < 1
12
13
   #
         'num_leaves': [340], # < 50
14
  #
         'min data in leaf': [50], # <10
  #
         'max bin': [200], # <50
15
16
   #
         'feature fraction': [0.7], # <0.1
  #
17
         'bagging fraction': [0.8], # <0.2
  #
         'bagging freq': [2], # <2
18
   #
         'lambda 11': [0.25], # <0.2
19
   #
         'lambda 12': [0.1], # <0.1
20
  #
21
         'min split gain': [0.9], # <0.1
  # #
22
           'n estimators': [],
   # }
23
24
   #
         # roc_auc: {'max_depth': 9, 'num_leaves': 1002}
25
   #
         # recall: {'max depth': 12, 'num leaves': 1002}
  #
         # roc auc: {'max depth': 9, 'num leaves': 502}
26
         # recall: {'max_depth': 12,
                                     'num leaves': 1002}
27
   #
         # roc auc: {'max depth': 9, 'num leaves': 500}
28 #
29 #
         # recall: {'max depth': 12, 'num leaves': 550}
                                     'num leaves': 400}
30 #
         # roc auc: {'max depth': 9,
         # recall: {'max depth': 12, 'num leaves': 530}
31
   #
32
         # roc auc:
         # {'max depth': 9, 'num leaves': 350}; range(350,450,10),
33
   #
34
   # {'max_bin': 200, 'max_depth': 9, 'min_data_in_leaf': 50, 'num_leaves': 340}
35
   # {'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
37
38
   #
39
         'max depth': [9],
         'num leaves': [350],
40
         'min data in leaf': [10, 20, 30, 40, 50],
41
42
   #
         'max_bin': [10, 50, 100, 200, 300],
         {'max_bin': 200, 'max_depth': 9, 'min_data_in_leaf': 50, 'num leaves': 35
43
44
   #
45
         'max depth': [9],
   #
46
         'num_leaves': [350],
  #
47
         'min data in leaf': [50],
         'max bin': [200],
48
49
   #
         'feature fraction': [0.7, 0.8, 0.9],
  #
50
         'bagging_fraction': [0.7, 0.8, 0.9],
51
   #
         {'bagging_fraction': 0.7, 'feature_fraction': 0.7, 'max_bin': 200, 'max_d
52
   #
53
         'max depth': [9],
  #
54
         'num leaves': [350],
55
  #
         'min_data_in_leaf': [50],
56
         'max bin': [200],
  #
57
         'feature_fraction': [0.7],
58 #
         'bagging fraction': [0.7],
  #
59
         'bagging freq': [3, 4, 5],
```

```
60
    #
          { 'bagging fraction': 0.7, 'bagging freq': 3, 'feature fraction': 0.7, 'ma
 61
          'max depth': [9],
 62
    #
          'num leaves': [350],
 63
   #
 64
          'min data in leaf': [50],
 65
    #
          'max bin': [200],
 66
    #
          'feature fraction': [0.7],
 67
    #
          'bagging fraction': [0.7],
   #
 68
          'bagging freg': [3],
 69
    #
          'lambda 11': [0, 0.1, 0.2],
          'lambda 12': [0, 0.1, 0.2],
 70
    #
    #
          { 'bagging_fraction': 0.7, 'bagging_freq': 3, 'feature_fraction': 0.7, 'la
 71
 72
 73
          'max depth': [9],
 74
    #
          'num leaves': [350],
    #
 75
          'min data in leaf': [50],
 76
    #
          'max bin': [200],
    #
 77
          'feature fraction': [0.7],
 78
    #
          'bagging fraction': [0.7],
 79
    #
          'bagging freg': [3],
   #
          'lambda 11': [0.2],
 80
 81
          'lambda 12': [0.1],
          'min_split_gain': [0.7, 0.8, 0.9],
 82
          { 'bagging_fraction': 0.7, 'bagging_freq': 3, 'feature_fraction': 0.7, 'la
 83
    # """
 84
 85
 86
 87
    # # 5.2 模型定义
 88
 89
   # grid search = GridSearchCV(
 90
          LGBMClassifier(
   #
              learning rate=0.1,
 91
 92
    #
          ),
 93 #
          param grid,
 94
          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,
 95
          cv=5.
 96
    #
 97
    #
          verbose=10
   # )
 98
 99
    # # 5.3 模型训练
100 # import time
101 # time start = time.time()
102 # grid search.fit(X train, Y train)
    # time end=time.time()
103
   # print('time cost', time end - time start, 's')
104
105 # # 5.4 训练记录
    # result['acc0'] = grid search.score(X test, Y test)
106
    # result['best_params0'] = grid_search.best_params_
107
108 | # result['best score0'] = grid search.best score
109 | # result['max_depth'] = grid_search.best_params_['max_depth']
110 | # result['num_leaves'] = grid_search.best_params_['num_leaves']
   # # 5.5 结果保存
111
    # print(result)
112
    # # with open(result path, 'w') as fp:
113
    # #
            json.dump(result, fp)
114
    115
```

```
2
   # logging.warning("Begin Model.")
   |# result = {} # 保存一些结果
 3
 4
   ## lightGBM 模型
 5
   # from lightqbm import LGBMClassifier
 6
   # """
 7
   #
 8
      'max depth': [9], # < 1
 9
   #
         'num leaves': [340], # < 50
   #
         'min data in leaf': [50], # <10
10
         'max_bin': [200], # <50
11
   #
         'feature fraction': [0.7], # <0.1
   #
12
13
   #
         'bagging_fraction': [0.8], # <0.2
  #
14
         'bagging freq': [2], # <2
15 #
         'lambda 11': [0.25], # <0.2
16
         'lambda 12': [0.1], # <0.1
17
   #
         'min split gain': [0.9], # <0.1
   # """
18
19
   # # 5.1 模型定义
20
   ##过拟合,避免过拟合
21
22
  # model = LGBMClassifier(
23
   #
        max depth=9,
24
   #
        num leaves=340,
25
   #
        min data in leaf=50,
        max bin=200,
2.6
27
   #
        feature fraction=0.7,
   #
28
        bagging fraction=0.8,
29
        bagging freg=2,
30 #
        lambda 11=0.25,
        lambda 12=0.1,
   #
31
   # #
         min_split_gain=0.9, # 过拟合
32
33
   #
34
         learning rate=0.01,
   #
35
        n estimators=10000,
36
37
         verbose=10
38
   # )
39
  # # 5.2 模型训练
40
   # import time
41
   # time start=time.time()
42
43 # model.fit(X train, Y train)
44
  # time end=time.time()
  # print('time cost', time end-time start, 's')
45
46 | # # 5.3 模型评估
47 | # result['score train'] = model.score(X train, Y train)
  # result['score test'] = model.score(X test, Y test)
48
   # # 5.4 模型保存
49
50 # joblib.dump(model, f'./model/lightGBM_(%f, %f, %f).model'%
51
                (test size, result['score train'], result['score test']))
52 # # 5.5 结果保存
53
  # print(result)
  # # with open(result path, 'w') as fp:
55
   # #
          json.dump(result, fp)
56
```

In []:	
1	
In []:	
1	
In []:	
1	
In []:	
1	