《数据分析课程设计》

基于 ICR 数据的主要疾病的分类预测分析

摘要

项目来源：来自于 Kaggle 上正在进行的竞赛项目。

比赛主办方：比赛主办方：竞赛主办方 InVitro Cell Research, LLC (ICR)成立于 2015 年，是一家专注于再生和预防性个性化医疗的私募公司。他们在大纽约地区的办公室和实验室提供最先进的研究空间。

比赛目标：预测一个人是否患有三种疾病中的任何一种；患有一种或多种（class 1），没有这三种疾病中的一种（class 0）。

比赛数据：包含与三种与年龄相关的状况相关联的五十多个匿名健康特征（脱敏）。目标是预测受试者是否被诊断出患有这些病症之一，即是二元分类问题。

- train.csv---训练集。
  - ID 每个观察的唯一标识符。
  - AB-GL 五十六个匿名的健康特征。除了 EJ，是类别型变量，其余都是数值型。
  - Class 二元目标：1 表示已被诊断为三种情况之一，0 表示没有。
- test.csv ---测试集。目标是预测该集合中的受试者属于两个类别中的每个类别的概率。
- greeks.csv---补充元数据，仅适用于训练集。
  - Alpha：标识与年龄相关的状况的类型（如果存在）。

    A 没有与年龄有关的情况。对应 class 0

    B,D,G 三种与年龄有关的情况。对应 class 1
  - Beta,Gamma,Delta 三个实验特性
  - Epsilon 收集该主题数据的日期。请注意，测试集中的数据都是在收集训练集之后收集的。
- sample_submission.csv ---格式正确的示例提交文件。

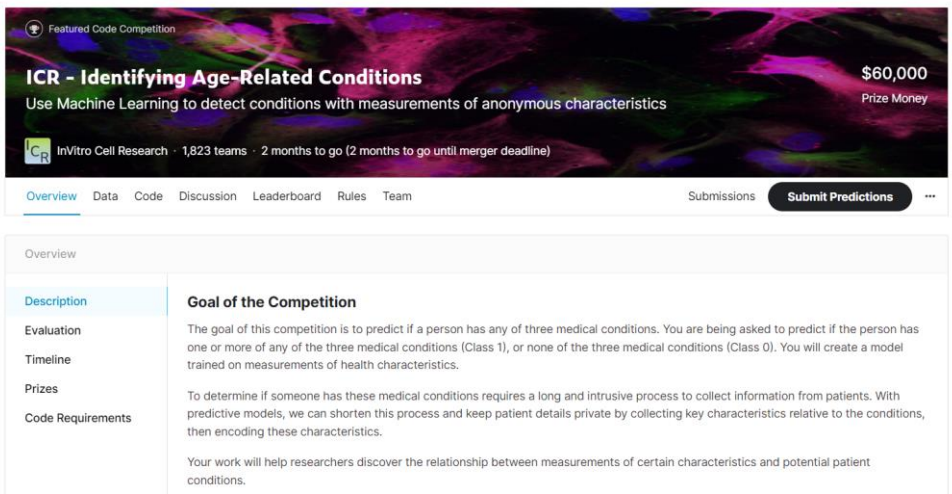


图 1-1 项目介绍

一、提出问题

1. 训练数据少的前提下，如何有效进行数据的各项处理处理？

2. 在数据集当中的所有特征变量都是脱敏的情况下，如何进行特征选择？

3. 目前，XGBoost 和随机森林等模型用于预测医疗状况，但模型的性能还不够好，如何进行调整优化？

二、获取数据

1. 导包

Loading Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib

from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, GridSearchCV
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay
#分类模型评价
from sklearn.metrics import classification_report
#混淆矩阵
from sklearn.metrics import confusion_matrix
#roc,P-R
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score
plt.rcParams["font.sans-serif"]=['SimHei']
plt.rcParams["axes.unicode_minus"]=False
plt.rcParams.update({'font.size':13 }) #全局字体大小
plt.rcParams["font.family"]=['SimHei']
plt.rcParams['axes.unicode_minus']=False
```

图 2-1 导包

2. 读入数据集

load dataset

```
train = pd.read_csv('.//data/train.csv')
greeks = pd.read_csv('.//data/greeks.csv')
test = pd.read_csv('.//data/test.csv')
sample_submission = pd.read_csv('.//data/sample_submission.csv')
```

图 2-2 读入数据集

3. 查看数据基本情况

show dataset

```
print('train:',train.shape)
print('test:',test.shape)
print('greeks:',greeks.shape)
```

```
train: (617, 58)
test: (5, 57)
greeks: (617, 6)
```

图 2-3 数据基本情况

三、数据预处理和探索性分析

1. 训练集信息

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 617 entries, 0 to 616
Data columns (total 58 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Id      617 non-null    object 
 1   AB      617 non-null    float64
 2   AF      617 non-null    float64
 3   AH      617 non-null    float64
 4   AM      617 non-null    float64
 5   AR      617 non-null    float64
 6   AX      617 non-null    float64
 7   AY      617 non-null    float64
 8   AZ      617 non-null    float64
 9   BC      617 non-null    float64
 10  BD      617 non-null    float64
 11  BN      617 non-null    float64
 12  BP      617 non-null    float64
 13  BQ      617 non-null    float64
 14  BR      617 non-null    float64
 15  BZ      617 non-null    float64
 16  CB      617 non-null    float64
 17  CC      617 non-null    float64
 18  CD      617 non-null    float64
 19  CF      617 non-null    float64
 20  CH      617 non-null    float64
 21  CL      617 non-null    float64
 22  CR      617 non-null    float64
 23  CS      617 non-null    float64
 24  CU      617 non-null    float64
 25  CW      617 non-null    float64
 26  DA      617 non-null    float64
 27  DE      617 non-null    float64
 28  DF      617 non-null    float64
 29  DH      617 non-null    float64
 30  DI      617 non-null    float64
 31  DL      617 non-null    float64
 32  DN      617 non-null    float64
 33  DU      617 non-null    float64
 34  DV      617 non-null    float64
 35  DY      617 non-null    float64
 36  EB      617 non-null    float64
 37  EE      617 non-null    float64
 38  EG      617 non-null    float64
 39  EH      617 non-null    float64
 40  EJ      617 non-null    object 
 41  EL      617 non-null    float64
 42  EP      617 non-null    float64
 43  EU      617 non-null    float64
 44  FC      617 non-null    float64
 45  FD      617 non-null    float64
 46  FE      617 non-null    float64
 47  FI      617 non-null    float64
 48  FL      617 non-null    float64
 49  FR      617 non-null    float64
 50  FS      617 non-null    float64
 51  GB      617 non-null    float64
 52  GE      617 non-null    float64
 53  GF      617 non-null    float64
 54  GH      617 non-null    float64
 55  GI      617 non-null    float64
 56  GL      617 non-null    float64
 57  Class   617 non-null    int64  
dtypes: float64(55), int64(1), object(2)
memory usage: 284.4+ KB
```

图 3-1 训练集信息

2. 查看训练集的缺失情况

```
# Missing training data
Missing_col = train.isnull().sum()[train.isnull().sum().values!=0]
Missing_col
```

```
BQ    60
CB     2
CC     3
DU     1
EL    60
FC     1
FL     1
FS     2
GL     1
dtype: int64
```

图 3-2 训练集缺失情况

分析：由于训练集数据量有限，故而不能简单采取删除缺失列的方法。

3. 处理缺失值

处理思路：

- 合并 train 和 greeks(训练集的补充元数据)
- 查看变量列中缺少值的 Alpha 情况
- 根据步骤 1 中获得的“Alpha”计算变量列的平均值
- 替换与此列相关的缺失值

1） 拼接 train 和 greeks 两个数据集

```
# 拼接train和greeks两个数据集
train = pd.merge(train,greeks,on='Id')
```

```
train.sample(5)
```

Id	AB	AF	AH	AM	AR	AX	AY	AZ	BC	...	GF	GH	GI	GL	Class	Alpha
c214d487e2db	0.235015	2648.34515	85.200147	10.474054	8.138688	2.919040	0.033495	3.396778	2.371950	...	16045.77406	19.665485	20.851204	0.260780	0	A
14a3f3cc04bd	0.559763	3263.59506	85.200147	85.083881	8.138688	13.208769	0.025578	11.980102	1.229900	...	13150.30853	19.092451	69.901816	21.978000	0	A
a46b439e1eac	0.615312	5379.22198	117.667299	30.477577	8.138688	5.590029	0.025578	11.129332	1.229900	...	20268.14279	27.286093	48.770264	0.173909	0	A
c5e5456ccaa2	0.247834	2925.54768	85.200147	25.565735	8.138688	4.704129	0.025578	3.396778	1.229900	...	28431.62111	26.839573	26.805384	0.133562	0	A
8b9d72eec14e	4.277273	7314.15234	85.200147	630.518230	59.390922	6.936597	0.025578	18.805168	4.649022	...	14702.79859	39.427716	74.479976	0.013500	0	A

ns × 63 columns

图 3-3 拼接 train 和 greeks

2)缺失值处理

'BQ'

```
# 查看变量列'BQ'缺失值的 Alpha 情况
train[['BQ','Alpha']][train.BQ.isnull()]['Alpha'].unique()
```

```
array(['A'], dtype=object)
```

分析：缺失值的'Alpha'都为'A';表示:没有与年龄相关的疾病。对应目标变量：类别 0

```
# 采用'Alpha'=='A'的'BQ'均值替换此列缺失值
mean = train[train.Alpha=='A']['BQ'].mean()
train.BQ.fillna(mean,inplace=True)
```

'CB'

```
# 查看变量列'CB'缺失值的 Alpha 情况
train[['CB','Alpha']][train.CB.isnull()]['Alpha'].unique()
```

```
array(['A'], dtype=object)
```

分析：缺失值的'Alpha'都为'A';表示:没有与年龄相关的疾病。对应目标变量：类别 0

```
# 采用'Alpha'=='A'的'CB'均值替换此列缺失值
mean = train[train.Alpha=='A']['CB'].mean()
train.CB.fillna(mean,inplace=True)
```

'CC'

```
# 查看变量列'CC'缺失值的 Alpha 情况
train[['CC','Alpha']][train.CC.isnull()]['Alpha'].unique()
```

```
array(['A', 'D'], dtype=object)
```

分析：缺失值的'Alpha'为'A'和'D';表示:既有没有与年龄相关的疾病；也有与年龄相关的疾病。对应目标变量：类别 0 和类别 1

```
# 采用'Alpha'=='A'/'D'的'CC'均值替换此列缺失值
mean1 = train[train.Alpha=='A']['CC'].mean()
mean2 = train[train.Alpha=='D']['CC'].mean()

for i in train[(train.Alpha=='A') & (train.CC.isnull())].index:
    temp = train[train.index==i].copy()
    temp.CC=mean1
    train[train.index==i] = temp
for i in train[(train.Alpha=='D') & (train.CC.isnull())].index:
    temp = train[train.index==i].copy()
    temp.CC=mean2
    train[train.index==i] = temp
```

'DU'

```
# 查看变量列'DU'缺失值的 Alpha 情况
train[['DU','Alpha']][train.DU.isnull()]['Alpha'].unique()
```

```
array(['A'], dtype=object)
```

分析：缺失值的'Alpha'都为'A';表示:没有与年龄相关的疾病。对应目标变量：类别 0

```
# 采用'Alpha'=='A'的'DU'均值替换此列缺失值
mean = train[train.Alpha=='A']['DU'].mean()
train.DU.fillna(mean,inplace=True)
```

'EL'

```
# 查看变量列'DU'缺失值的 Alpha 情况
train[['EL','Alpha']][train.EL.isnull()]['Alpha'].unique()
```

```
array(['A', 'B', 'G', 'D'], dtype=object)
```

分析：缺失值的'Alpha'为'A'和'D';表示:既有没有与年龄相关的疾病；也有与年龄相关的疾病。对应目标变量：类别 0 和类别 1

```
# 采用'Alpha'=='A'/'D'的'CC'均值替换此列缺失值
mean1 = train[train.Alpha=='A']['EL'].mean()
mean2 = train[train.Alpha=='B']['EL'].mean()
mean3 = train[train.Alpha=='D']['EL'].mean()
mean4 = train[train.Alpha=='G']['EL'].mean()

for i in train[(train.Alpha=='A') & (train.EL.isnull())].index:
    temp = train[train.index==i].copy()
    temp.EL = mean1
    train[train.index==i] = temp
for i in train[(train.Alpha=='B') & (train.EL.isnull())].index:
    temp = train[train.index==i].copy()
    temp.EL = mean2
    train[train.index==i] = temp
for i in train[(train.Alpha=='D') & (train.EL.isnull())].index:
    temp = train[train.index==i].copy()
    temp.EL = mean3
    train[train.index==i] = temp
for i in train[(train.Alpha=='G') & (train.EL.isnull())].index:
    temp = train[train.index==i].copy()
    temp.EL = mean4
    train[train.index==i] = temp
```

'FC'

```
# 查看变量列'FC'缺失值的 Alpha 情况
train[['FC','Alpha']][train.FC.isnull()]['Alpha'].unique()
```

```
array(['D'], dtype=object)
```

分析：缺失值的'Alpha'为'A'和'D';表示:有与年龄相关的疾病。对应目标变量：类别 1

```
# 采用'Alpha'=='D'的'FC'均值替换此列缺失值
mean = train[train.Alpha=='D']['FC'].mean()
train.FC.fillna(mean,inplace=True)
```

'FL'

```
# 查看变量列'FC'缺失值的 Alpha 情况
train[['FL','Alpha']][train.FL.isnull()]['Alpha'].unique()
```

```
array(['A'], dtype=object)
```

分析：缺失值的'Alpha'都为'A';表示:没有与年龄相关的疾病。对应目标变量：类别 0

```
# 采用'Alpha'=='A'的'DU'均值替换此列缺失值
mean = train[train.Alpha=='A']['FL'].mean()
train.FL.fillna(mean,inplace=True)
```

'FS'

```
# 查看变量列'FC'缺失值的 Alpha 情况
train[['FS','Alpha']][train.FS.isnull()]['Alpha'].unique()
```

```
array(['D', 'A'], dtype=object)
```

分析：缺失值的'Alpha'为'A'和'D';表示:既有没有与年龄相关的疾病；也有与年龄相关的疾病。对应目标变量：类别 0 和类别 1

```
# 采用'Alpha'=='A'/'D'的'CC'均值替换此列缺失值
mean1 = train[train.Alpha=='A']['FS'].mean()
mean2 = train[train.Alpha=='D']['FS'].mean()

for i in train[(train.Alpha=='A') & (train.FS.isnull())].index:
    temp = train[train.index==i].copy()
    temp.FS=mean1
    train[train.index==i] = temp
for i in train[(train.Alpha=='D') & (train.FS.isnull())].index:
    temp = train[train.index==i].copy()
    temp.FS=mean2
    train[train.index==i] = temp
```

'GL'

```
# 查看变量列'GL'缺失值的 Alpha 情况
train[['GL','Alpha']][train.GL.isnull()]['Alpha'].unique()
```

```
array(['A'], dtype=object)
```

分析：缺失值的'Alpha'都为'A';表示:没有与年龄相关的疾病。对应目标变量：类别 0

```
# 采用'Alpha'=='A'的'GL'均值替换此列缺失值
mean = train[train.Alpha=='A']['GL'].mean()
train.GL.fillna(mean,inplace=True)
```

图 3-4 缺失值处理

3)再次检查缺失值处理

```
# Recheck missing value handling(再次检查缺失值处理)
train.isnull().sum()[train.isnull().sum().values!=0]
```

```
Series([], dtype: int64)
```

图 3-5 再次查看缺失值处理情况

4) 移除 greeks 数据集相关列

```
# 去除缺失值后，移除greeks数据集相关列
train = train.iloc[:,0:58]
train.sample(5)
```

	Id	AB	AF	AH	AM	AR	AX	AY	AZ	BC	...
505	cc9f47d89979	0.769140	2199.85260	213.675315	30.973097	8.138688	4.580103	0.025578	3.396778	1.229900	...
4	044fb8a146ec	0.380297	3733.04844	85.200147	14.103738	8.138688	3.942255	0.054810	3.396778	102.151980	...
130	3493c79c8f35	0.202967	3301.82017	88.618430	13.473498	8.138688	3.946684	0.103225	10.789024	7.983808	...
226	5a87a6061488	0.730683	1704.55725	152.921331	94.944729	8.138688	5.031912	0.036845	10.952876	1.229900	...
292	7416fea10b6b	1.196440	4231.60958	85.200147	54.110784	8.138688	5.439426	0.025578	3.396778	7.150990	...

5 rows × 58 columns

图 3-6 移除 greeks 数据集

4.探索性分析（EDA）

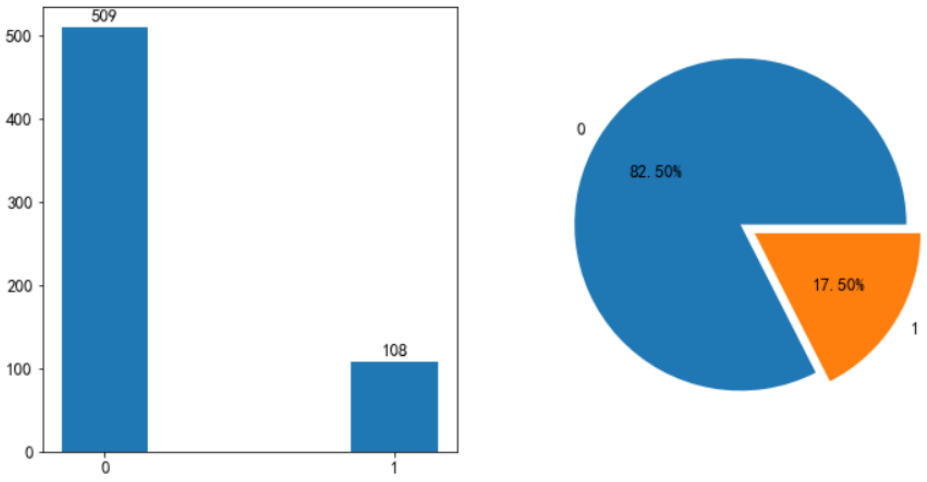
1)目标变量’Class’分布情况

```
print("distribution of 'Class':")
plt.figure(figsize=(12,6))
ax1 = plt.subplot(1,2,1)
rects = ax1.bar(x=train['Class'].value_counts().index,
       height=train['Class'].value_counts().values,
       width=0.3)
ax1.set_xticks(ticks=[0,1],labels=['0','1'])
ax1.bar_label(rects,padding=3,size=13)

ax2 = plt.subplot(1,2,2)
ax2.pie(train['Class'].value_counts().values,
        labels=train['Class'].value_counts().index,
        autopct='%.2f%%',explode=[0.05,0.05])

plt.show()
```

```
distribution of 'Class':
```



分析：训练集当中目标变量的分布情况并不均衡，因此在后续应当考虑变量平衡！

图 3-7 目标变量的分布情况

2)数值型特征的分布

取出数值型特征变量名到 numerical 变量中

```
numerical = train.dtypes[train.dtypes=='float64'].index.to_list()
df_numerical = train.loc[:,numerical]
df_numerical
```

	AB	AF	AH	AM	AR	AX	AY	AZ	BC	BD	...
0	0.209377	3109.03329	85.200147	22.394407	8.138688	0.699861	0.025578	9.812214	5.555634	4126.58731	...
1	0.145282	978.76416	85.200147	36.968889	8.138688	3.632190	0.025578	13.517790	1.229900	5496.92824	...
2	0.470030	2635.10654	85.200147	32.360553	8.138688	6.732840	0.025578	12.824570	1.229900	5135.78024	...
3	0.252107	3819.65177	120.201618	77.112203	8.138688	3.685344	0.025578	11.053708	1.229900	4169.67738	...
4	0.380297	3733.04844	85.200147	14.103738	8.138688	3.942255	0.054810	3.396778	102.151980	5728.73412	...
...	...	...	...	...	...	...	...	...	...	...	...

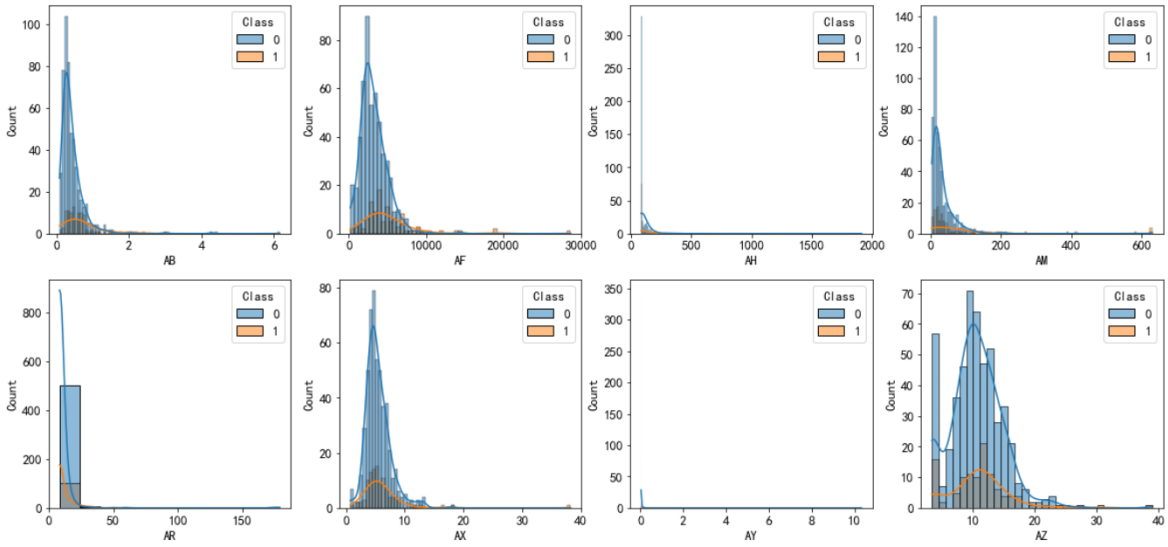
图 3-8 数值型特征变量

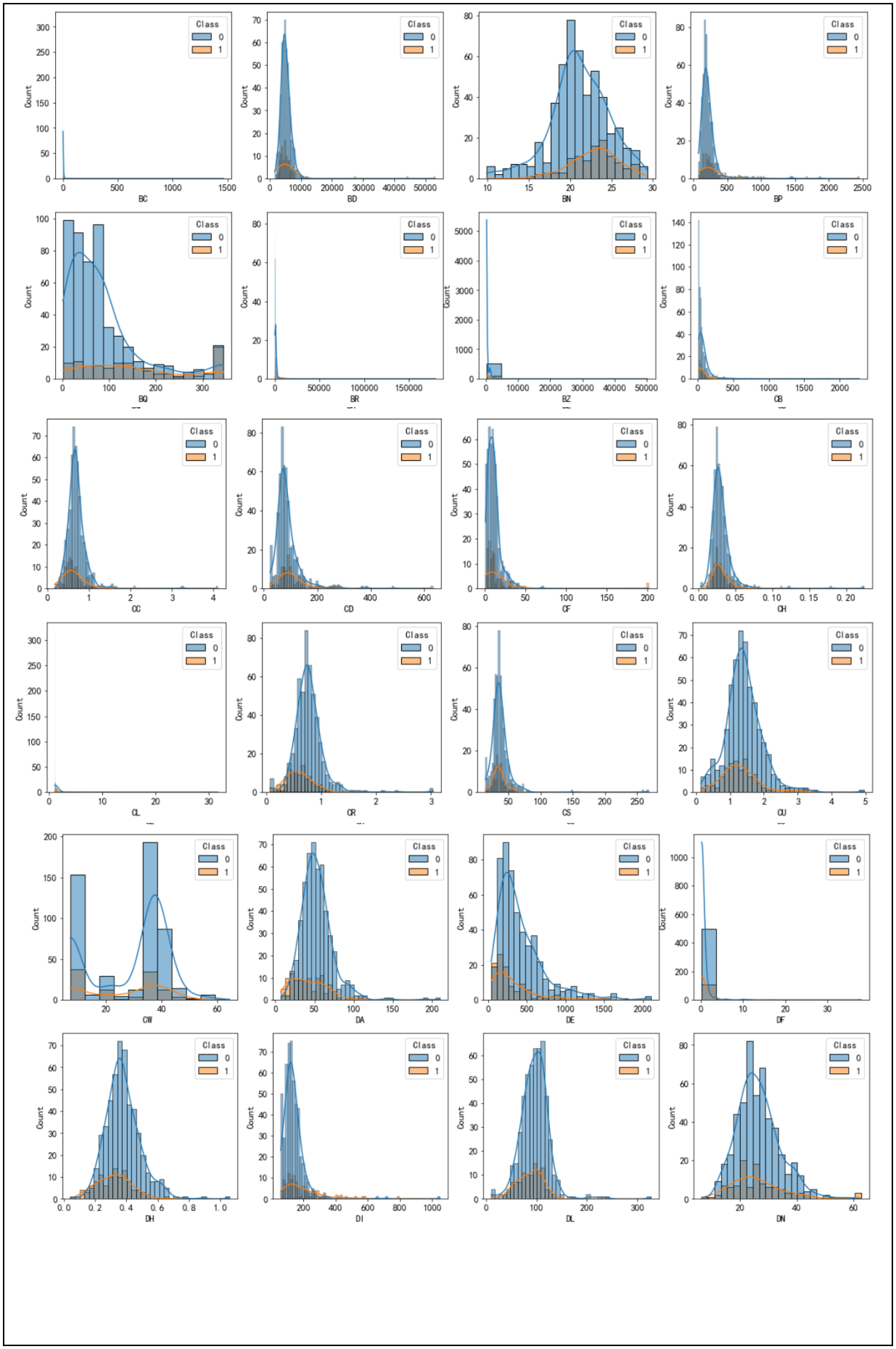
数值型特征变量分布情况(For 循环画图查看)

分析：由于此次的主题是疾病预测，同时数据集的各个特征变量都是脱敏的情况，因此对于数据集当中所谓的’异常值’,并没有采取处理，因为它可能是预测受试者是否患病的重要一环。故而并没有采取绘制箱线图查看数据分布情况。

distribution(变量分布情况)

```
#数值型变量
plt.figure(figsize=(20,70))
i=1
for temp in numerical:
    plt.subplot(14,4,i)
    sns.histplot(data=train,x=temp,hue='Class',kde=True)
    i+=1
plt.show()
```




Class
0
1
Count
BC
BD
BN
BP
BQ
BR
BZ
CB
CC
CD
CF
CH
CL
CR
CS
CU
CW
DA
DE
DF
DH
DI
DL
DN

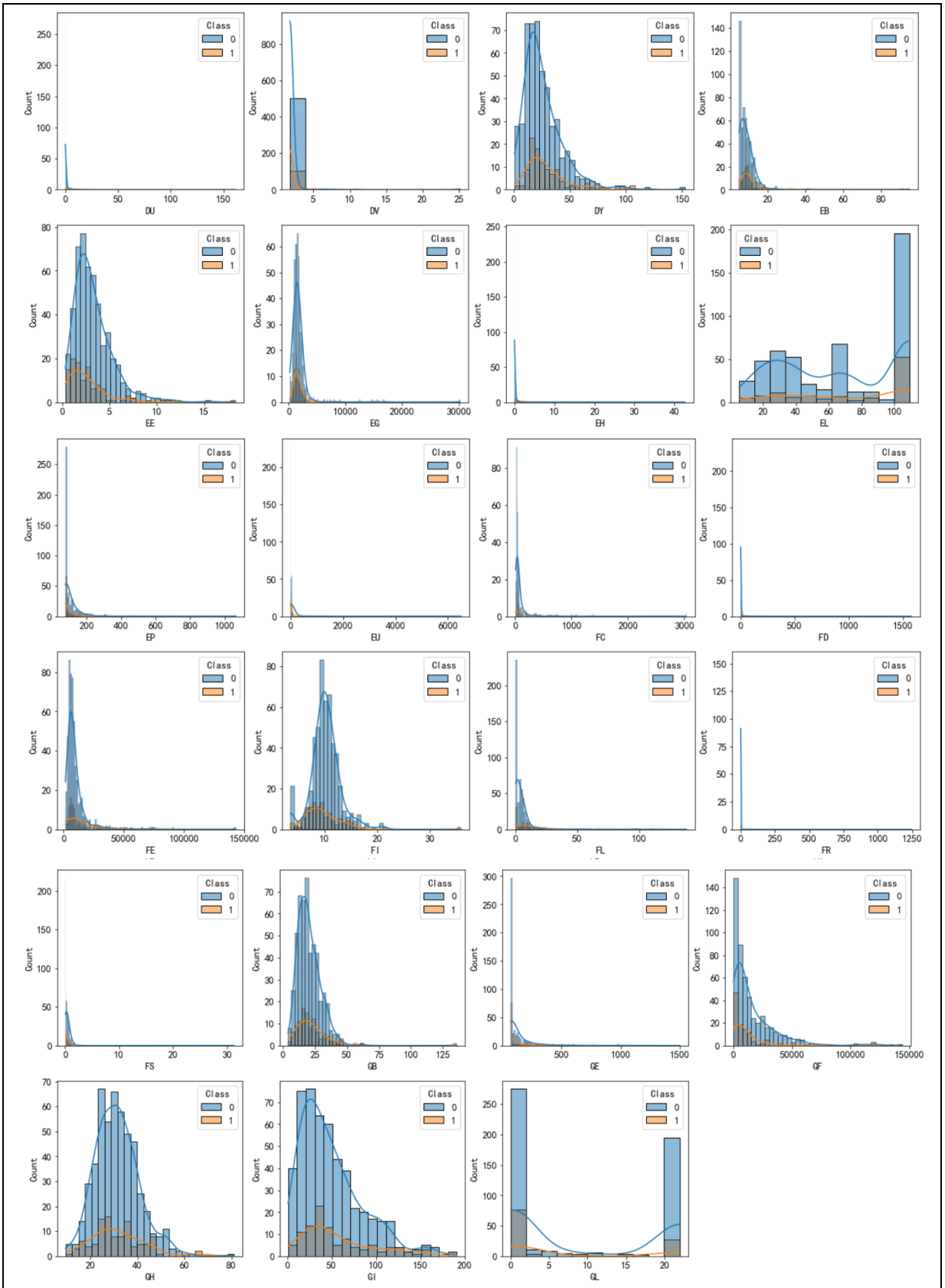


图 3-9 数值型变量的分布直方图

分析：从上述各图可看出，各个特征存在量纲差异，而且并不是所有特征都近似服从正态分布；从整体上看，目标变量的两种对应类别的特征变量分布大体呈现一致趋势。

类别型变量’EJ’的分布情况

```
# 类别型变量分布情况
plt.figure(figsize=(8,5))
sns.countplot(data=train,x='EJ',hue='Class')
plt.title("The distribution of 'EJ'")
plt.show()
```

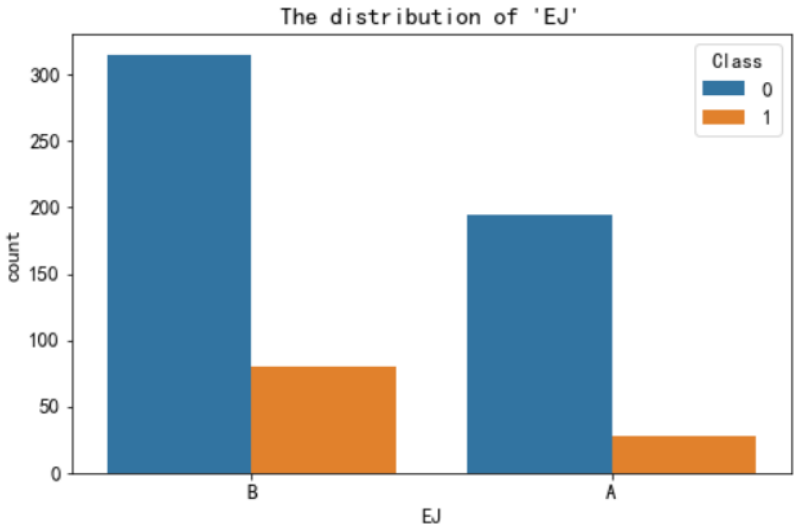


图 3-10 类别型变量 EJ 的分布情况

四、建模和模型评价（可视化）

1. 特征工程

1) 备份数据集

```
# 备份数据集
train_ = train.copy()
test_ = test.copy()
```

图 4-1 备份

2) 独热编码转换

```
# 对 train_和 test_ 的 'EJ' 进行独热编码转换
train_ = pd.get_dummies(train_,columns=['EJ'],prefix_sep='_')
test_ = pd.get_dummies(test_,columns=['EJ'],prefix_sep='_')
```

图 4-2 独热编码转换

3) 划分数据集

```
# 将训练集的目标变量和特征变量进行划分
X_train_ = train_.drop(['Class','Id'],axis=1)
Y = train_['Class']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test,Y_train, Y_test = train_test_split(X_train_,Y,train_size=0.70, random_state=10)
```

图 4-3 划分数据集

2. 建模

1)随机森林分类

```
# RandomForest
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(X_train,Y_train)

print(f'Accuracy on train set: {RF.score(X_train, Y_train)*100}%')
print(f'Accuracy on test set: {RF.score(X_test, Y_test)*100}%')
# The evaluation of RandomForest
print(f"model evaluation report:\n{classification_report(Y_test,RF.predict(X_test))}")
```

```
Accuracy on train set: 100.0%
Accuracy on test set: 92.47311827956989%
model evaluation report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       157
           1       0.78      0.72      0.75        29

    accuracy                           0.92       186
   macro avg       0.86      0.84      0.85       186
weighted avg       0.92      0.92      0.92       186
```

图 4-4 随机森林分类初始建模

2)XGBoost 分类

```
# xgboost
from xgboost import XGBClassifier
XGB = XGBClassifier()
XGB.fit(X_train,Y_train)
print(f'Accuracy on train set: {XGB.score(X_train, Y_train)*100}%')
print(f'Accuracy on test set: {XGB.score(X_test, Y_test)*100}%')
# The evaluation of XGBClassifier
print(f"model evaluation report:\n{classification_report(Y_test,XGB.predict(X_test))}")
```

```
Accuracy on train set: 100.0%
Accuracy on test set: 93.01075268817203%
model evaluation report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       157
           1       0.77      0.79      0.78        29

    accuracy                           0.93       186
   macro avg       0.86      0.87      0.87       186
weighted avg       0.93      0.93      0.93       186
```

图 4-5 XGBoost 分类初始建模

分析：结合此次竞赛主题是疾病预测领域，通常情况下更多关注的是召回率，即是查全率；因为高召回率意味着能将更多患病的预测出来，从上述两种建模算法的模型评价来看，XGBoost 的召回率和准确率相较于随机森林分类都更胜一筹。

综上，选取 XGBoostClassifer 进行模型优化。

3) SelectFromModel 变量筛选

a. XGBClassifier 作为基模型的特征选择，先进行探索阈值

```
from numpy import sort
from sklearn.metrics import recall_score
# 初始建模
XGB = XGBClassifier()
XGB.fit(X_train,Y_train)

Y_pred = XGB.predict(X_test.values)
recall = recall_score(Y_test, Y_pred)
print("Recall_score: %.2f%%" % (recall * 100.0))
# Fit model using each importance as a threshold
thresholds = sort(XGB.feature_importances_)
thresh_lt=[]
n_lt = []
recall_lt=[]
for thresh in thresholds:
    # select features using threshold
    selection = SelectFromModel(XGB, threshold=thresh, prefit=True)
    select_X_train = selection.transform(X_train.values)
    # train model
    selection_model = XGBClassifier()
    selection_model.fit(select_X_train, Y_train)
    # eval model
    select_X_test = selection.transform(X_test.values)
    y_pred = selection_model.predict(select_X_test)
    predictions = [round(value) for value in y_pred]
    recall = recall_score(Y_test, predictions)
    thresh_lt.append(thresh)
    n_lt.append(select_X_train.shape[1])
    recall_lt.append(recall*100.0)
df_temp = pd.DataFrame({'threshold':thresholds,'n':n_lt,'recall(%)':recall_lt})
df_temp
```

```
Recall_score: 84.95%
```

	threshold	n	recall(%)
0	0.000000	57	79.310345
1	0.000000	57	79.310345
2	0.000000	57	79.310345
3	0.000000	57	79.310345
4	0.000559	53	79.310345
5	0.000961	52	79.310345
6	0.002898	51	79.310345
7	0.002978	50	79.310345
8	0.003224	49	82.758621
9	0.003710	48	79.310345
10	0.003846	47	75.862069
11	0.004294	46	79.310345
12	0.004872	45	79.310345
13	0.005608	44	82.758621
14	0.006593	43	82.758621
15	0.006868	42	86.206897
16	0.007065	41	82.758621
17	0.007168	40	82.758621
18	0.007290	39	82.758621
19	0.007820	38	82.758621
20	0.009618	37	86.206897
21	0.009764	36	75.862069
22	0.010355	35	75.862069
23	0.010606	34	82.758621
24	0.010908	33	72.413793
25	0.011092	32	82.758621
26	0.011465	31	82.758621
27	0.011724	30	72.413793
28	0.011822	29	79.310345
29	0.012477	28	79.310345
30	0.012943	27	79.310345
31	0.013110	26	79.310345
32	0.014570	25	79.310345
33	0.015998	24	75.862069
34	0.016551	23	79.310345
35	0.017737	22	82.758621
36	0.018214	21	82.758621
37	0.018373	20	75.862069
38	0.018743	19	82.758621
39	0.019161	18	79.310345
40	0.020217	17	68.965517
41	0.020322	16	72.413793
42	0.021240	15	72.413793
43	0.022211	14	62.068966
44	0.024034	13	62.068966
45	0.026504	12	72.413793
46	0.029045	11	72.413793
47	0.029416	10	75.862069
48	0.030007	9	75.862069
49	0.038757	8	79.310345
50	0.039064	7	75.862069
51	0.046040	6	68.965517
52	0.050107	5	72.413793
53	0.052874	4	75.862069
54	0.056684	3	55.172414
55	0.057746	2	31.034483
56	0.084749	1	27.586207

图 4-6 XGBClassifier 作为基模型的特征选择探索

分析：从上可得，当阈值取到中位数时，recall 分数趋于稳定。

```
from sklearn.feature_selection import SelectFromModel
# XGBClassifier作为基模型的特征选择
selector_XGBC = SelectFromModel(XGBClassifier(),threshold='median').fit(X_train,Y_train)
```

b. 执行筛选并展示

```
print('SelectFromModel筛选前的特征变量维度：',X_train.shape)
print('SelectFromModel筛选后的特征变量维度：',selector_XGBC.transform(X_train).shape)
```

```
SelectFromModel筛选前的特征变量维度： (431, 57)
SelectFromModel筛选后的特征变量维度： (431, 29)
```

```
selector_X = X_train.columns[selector_XGBC.get_support()].to_list()
print('筛选后的特征变量个数:',len(selector_X))
```

```
筛选后的特征变量个数: 29
```

c. 按照筛选成功的变量进行划分训练集和测试集

```
# 按照筛选的结果对train_进行划分训练集和测试集
X = train_.loc[:,selector_X]
Y = train_['Class']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test,Y_train, Y_test = train_test_split(X,Y,train_size=0.70, random_state=10)
```

图 4-7 XGBClassifier 作为基模型嵌入法的特征选择

d. 筛选后的变量的相关性分析(重要变量相关性热力图)

```
train_corr_data = train_.loc[:,selector_X].corr(method='pearson')
plt.figure(figsize=(10,6))
sns.heatmap(train_corr_data,cmap=plt.cm.Blues)
plt.show()
```

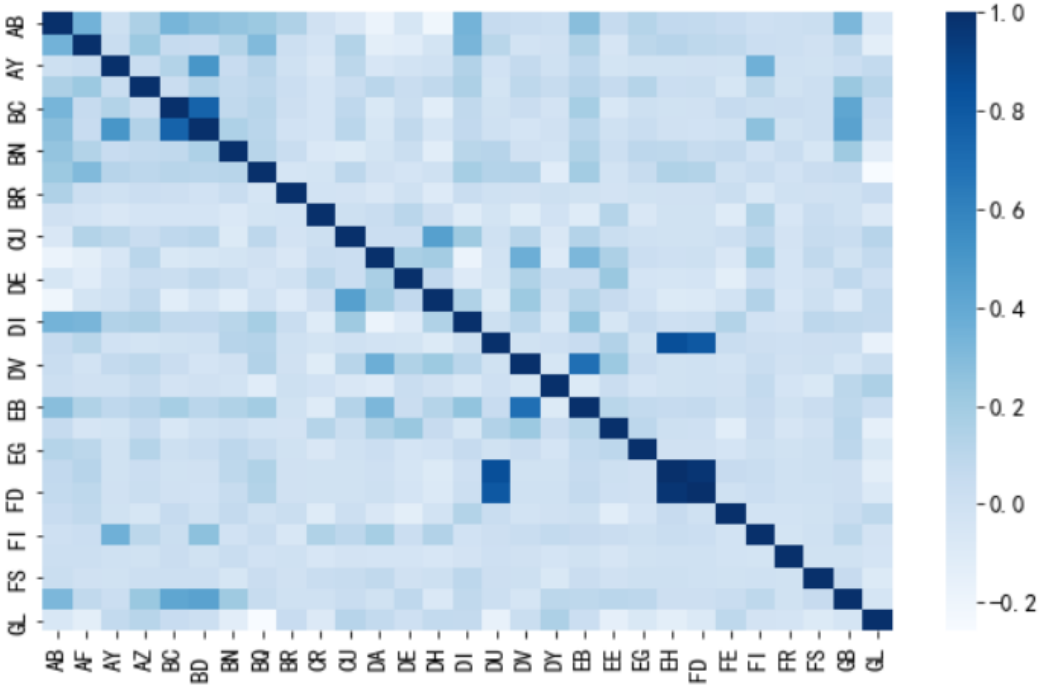


图 4-8 重要变量相关性热力图

五、模型优化

XGBoostClassifer 参数调优，采取 GridSearchCV 进行调参，对 XGBoost 的重要参数进行选择优化，如 n_estimators(树的个数)， max_depth(树的深度),learning_rate(学习率),min_child_weight(叶子节点中样本的权值，避免欠拟合)等；

由于训练集数据量有限，故而只采取 5 折交叉验证，避免模型出现过拟合。

```
from sklearn.model_selection import GridSearchCV,StratifiedKFold
from xgboost import XGBClassifier
from sklearn.model_selection import KFold,StratifiedKFold
XGB_model = XGBClassifier()
#params
parameters = {'learning_rate':[0.01,0.1,0.3],'subsample':[0.3,0.5],
              'max_depth':[3,6,9],'n_estimators':[300,800,1500],
              'min_child_weight':[2,4,6],'max_delta_step':[3,5,7],
              'eta':[0.1,0.3]
             }
#5fold
kfold = KFold(n_splits=5, shuffle=True, random_state=10)
#GridSearchCV
grid_search = GridSearchCV(XGB_model, parameters, scoring="recall",n_jobs=-1,cv=kfold)
grid_result = grid_search.fit(X_train,Y_train)
#best results
print(f"Best:{grid_result.best_score_};\nusing:\n{grid_result.best_params_}")
```

调优结果：

```
Best:0.7238789280894544;
using:
{'eta': 0.1, 'learning_rate': 0.01, 'max_delta_step': 3, 'max_depth': 6, 'min_child_weight': 2, 'n_estimators': 1500, 'subsample': 0.5}
```

图 4-9 模型调优

1)在网格调优的最优参数下探索模型复杂度与训练误差，测试误差以及交叉验证误差之间的情况同时探索竞赛评价指标对数损失的情况。

计算对数损失函数：

$$\text{Log Loss} = \frac{-\frac{1}{N_0}\sum_{i=1}^{N_0} y_{0i}\log p_{0i} - \frac{1}{N_1}\sum_{i=1}^{N_1} y_{1i}\log p_{1i}}{2}$$

```
# 对数损失函数
def competition_log_loss(y_true, y_pred):
    # y_true: 真实值 标签 0, 1
    # y_pred: 预测患病（class=1）的概率
    # 使用实现评估方程式 w_0 = w_1 = 1.
    #计算每个类别的观测次数
    N_0 = np.sum(1 - y_true)
    N_1 = np.sum(y_true)
    # 计算每个类别的预测概率
    p_1 = np.clip(y_pred, 1e-15, 1 - 1e-15)
    p_0 = 1 - p_1
    # 计算每类的平均对数损失
    log_loss_0 = -np.sum((1 - y_true) * np.log(p_0)) / N_0
    log_loss_1 = -np.sum(y_true * np.log(p_1)) / N_1
    # 返回
    return (log_loss_0 + log_loss_1)/2
```

图 4-10 对数损失函数

循环探索不同模型复杂度下的各个误差并且进行可视化：

```
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
```

```
trainErr = []  #训练误差
testErr = []   #测试误差
CV_recall_Err = []     #交叉验证召回率误差
CV_f1_Err = []        #交叉验证f1误差
CV_precision_Err = []#交叉验证精确率误差
log_loss_lt = []
for k in np.arange(100,2000,100):  #模型复杂度 K
    # model
    XGBC_model = XGBClassifier(n_estimators=k,max_depth=6,learning_rate=0.01,
                           subsample=0.5,eta=0.1,max_delta_step=3,min_child_weight=2,
                           random_state=10)
    XGBC_model.fit(X_train,Y_train)
    y_pred = XGBC_model.predict(X_test)
    test_prob = XGBC_model.predict_proba(X_test)
    test_prob_df = pd.DataFrame(test_prob, columns=['class_0', 'class_1'])
    log_loss = competition_log_loss(Y_test,test_prob_df.class_1.to_list())
    log_loss_lt.append(log_loss)
    trainErr.append(1-XGBC_model.score(X_train,Y_train))
    testErr.append(1-(XGBC_model.score(X_test,Y_test)))
    recall_Err = 1-cross_val_score(XGBC_model,X,Y,cv=5,scoring='recall')
    f1_Err = 1-cross_val_score(XGBC_model,X,Y,cv=5,scoring='f1')
    precision_Err = 1-cross_val_score(XGBC_model,X,Y,cv=5,scoring='precision')
    CV_recall_Err.append(recall_Err.mean())
    CV_f1_Err.append(f1_Err.mean())
    CV_precision_Err.append(precision_Err.mean())
```

图 4-11 for 循环计算不同模型复杂度下的误差

```
K = np.arange(100,2000,100)
plt.figure(figsize=(8,5))
plt.plot(K,log_loss_lt,label='log_loss',marker='o',linestyle = '-')
bestK = K[log_loss_lt.index(np.min(log_loss_lt))]
plt.xlabel('模型复杂程度')
plt.ylabel('对数损失')
plt.legend()
plt.title(f'模型复杂度和对数损失\n(Best K={bestK})',fontsize=18)
plt.show()
```

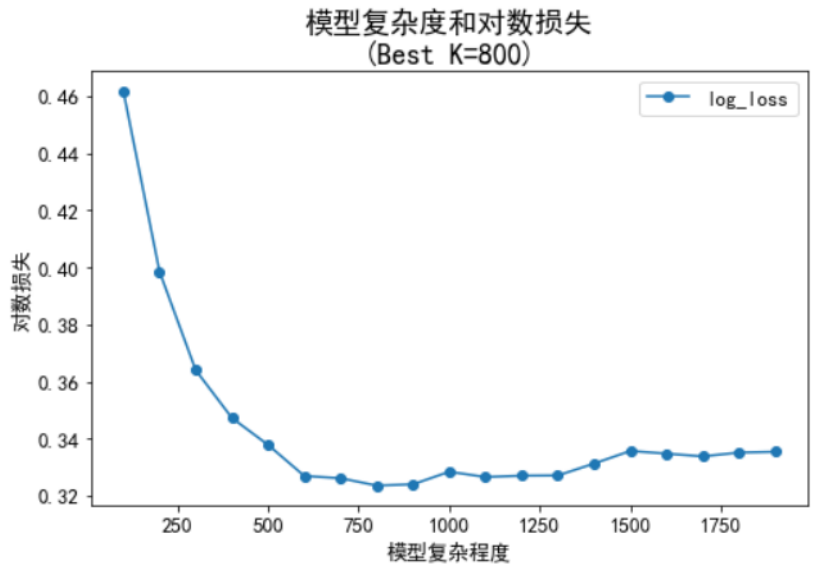


图 4-12 XGBoost 模型的对数损失折线图

绘制折线图显示模型复杂度与各误差之间的情况

```
K = np.arange(100,2000,100)
plt.figure(figsize=(20,6))
ax1 = plt.subplot(1,2,1)
ax1.grid(True,linestyle = '-.')
ax1.plot(K,trainErr,label='训练误差',marker='o',linestyle = '-')
ax1.plot(K,testErr,label='测试误差',marker='o',linestyle = '-.')
bestK = K[testErr.index(np.min(testErr))]
ax1.set_xlabel('模型复杂程度')
ax1.set_ylabel('误差')
ax1.legend()
ax1.set_title(f'模型复杂度和误差\n(Best K={bestK})',fontsize=18)
ax2 = plt.subplot(1,2,2)
ax2.grid(True,linestyle = '-.')
ax2.plot(K,CV_recall_Err,label='5折交叉验证_recall',marker='o',linestyle = '-')
ax2.plot(K,CV_f1_Err,label='5折交叉验证_f1',marker='o',linestyle = '-.')
ax2.plot(K,CV_precision_Err,label='5折交叉验证_precision',marker='o',linestyle = '--')
ax2.set_xlabel('模型复杂程度')
ax2.set_ylabel('误差')
ax2.legend()
ax2.set_title(f'模型复杂度和误差\n交叉验证',fontsize=18)
plt.show()
```

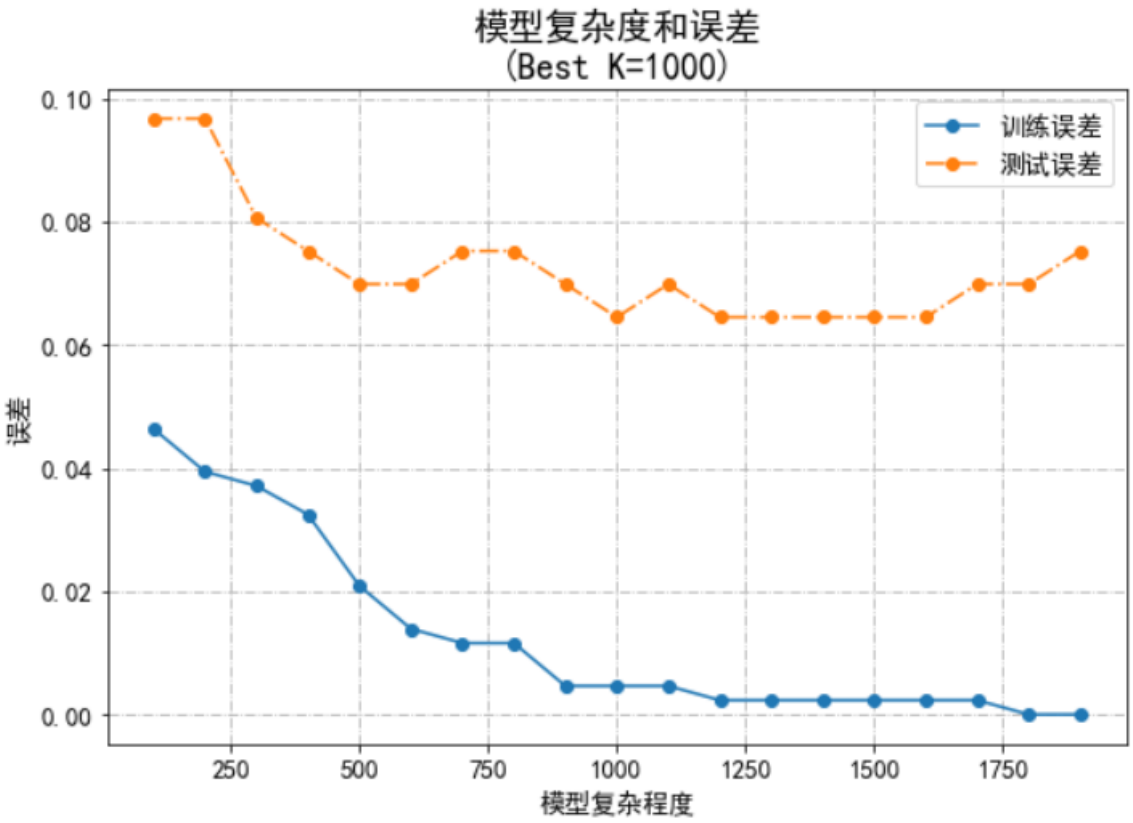


图 4-13 XGBoost 模型的训练与测试误差折线图

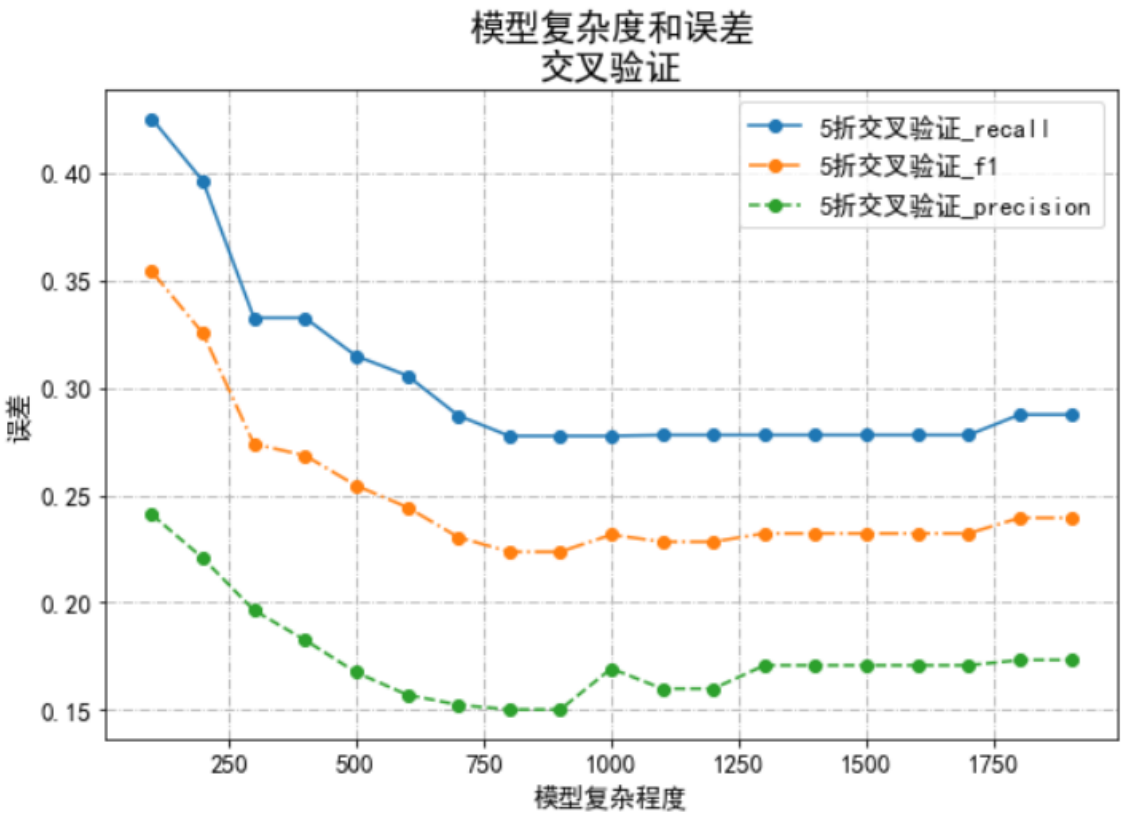


图 4-14 XGBoost 模型的交叉验证误差折线图

分析：从上图得出，当模型复杂度为 900 左右时，模型测试误差已经趋于平稳且变化较小，而且此时的训练误差已经趋于平稳，而此时评估指标对数损失也是较小而且趋于平稳。

2)XGBoost 模型建立

```
from xgboost import XGBClassifier
# model
XGBC_model = XGBClassifier(max_depth=6, learning_rate=0.01,n_estimators=900,
                           subsample=0.5,max_delta_step=3,eta=0.1,
                           min_child_weight=2,random_state=10)
```

```
#把train_划分的训练集进行模型训练
XGBC_model.fit(X_train,Y_train)
```

图 4-15 XGBoost 模型的建立

采用如上方法对于随机森林模型进行同样的探索

```
K = np.arange(100,2000,100)
plt.figure(figsize=(8,5))
plt.plot(K,log_loss_lt,label='log_loss',marker='o',linestyle = '-')
bestK = K[log_loss_lt.index(np.min(log_loss_lt))]
plt.xlabel('模型复杂程度')
plt.ylabel('对数损失')
plt.legend()
plt.title(f'随机森林模型复杂度和对数损失\n(Best K={bestK})',fontsize=18)
plt.show()
```

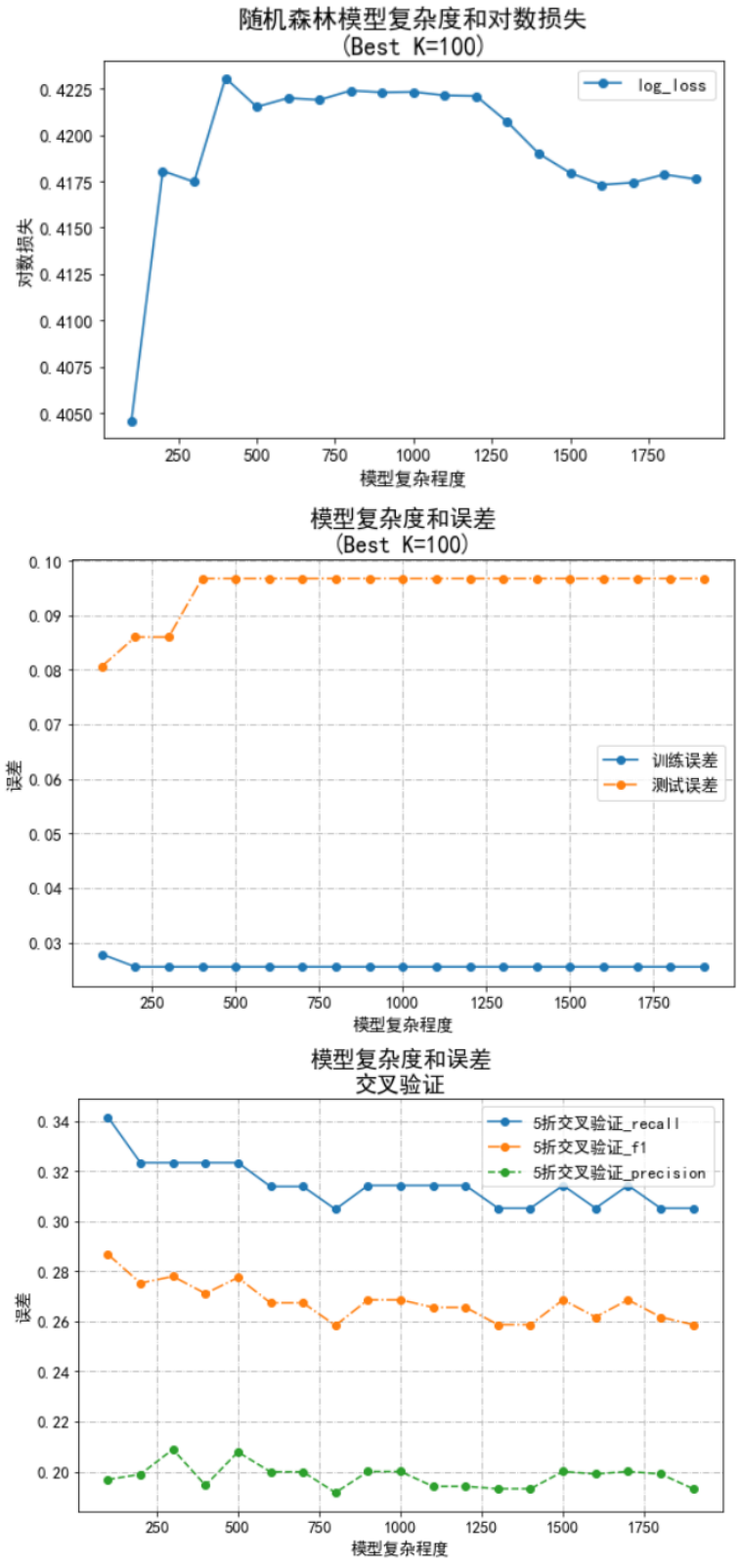


图 4-16 随机森林模型的探索结果可视化

结果分析：随机森林模型的模型探索情况表示，评估指标相较于 XGBoost 较大，而且随着模型复杂度的增加，对数损失呈现先上升，平稳，稍微下降的趋势，结合初步探索以及此次的调优模型探索，选定模型为 XGBoost

3. 模型评价

1) 报告

```
# 模型的评价
print(f"model evaluation report:\n{classification_report(Y_test,XGBC_model.predict(X_test))}")
```

```
model evaluation report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       157
           1       0.75      0.83      0.79        29

    accuracy                           0.93       186
   macro avg       0.86      0.89      0.87       186
weighted avg       0.93      0.93      0.93       186
```

图 4-16 XGBoost 模型评价报告

2) 混淆矩阵

```
# 混淆矩阵
cnf_matrix = confusion_matrix(Y_test,XGBC_model.predict(X_test))
print(f'confusion matrix: \n{cnf_matrix}\n')
plt.figure(figsize=(8,4))
sns.heatmap(cnf_matrix,annot=True,cmap=plt.cm.Blues,fmt='g')
plt.title('the heatmap of confusion matrix')
plt.show()
```

```
confusion matrix:
[[149   8]
 [  5  24]]
```

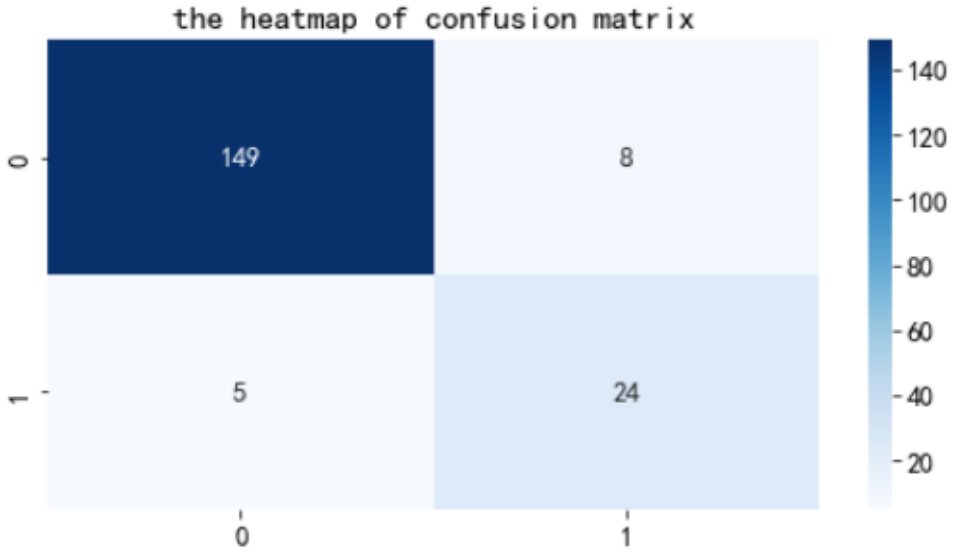


图 4-17 XGBoost 模型混淆矩阵

3) 显示特征变量重要性

```
# 显示特征变量重要性
from xgboost import plot_importance
plt.rcParams["figure.figsize"] = (12,10)
plot_importance(XGBC_model)
plt.show()
```

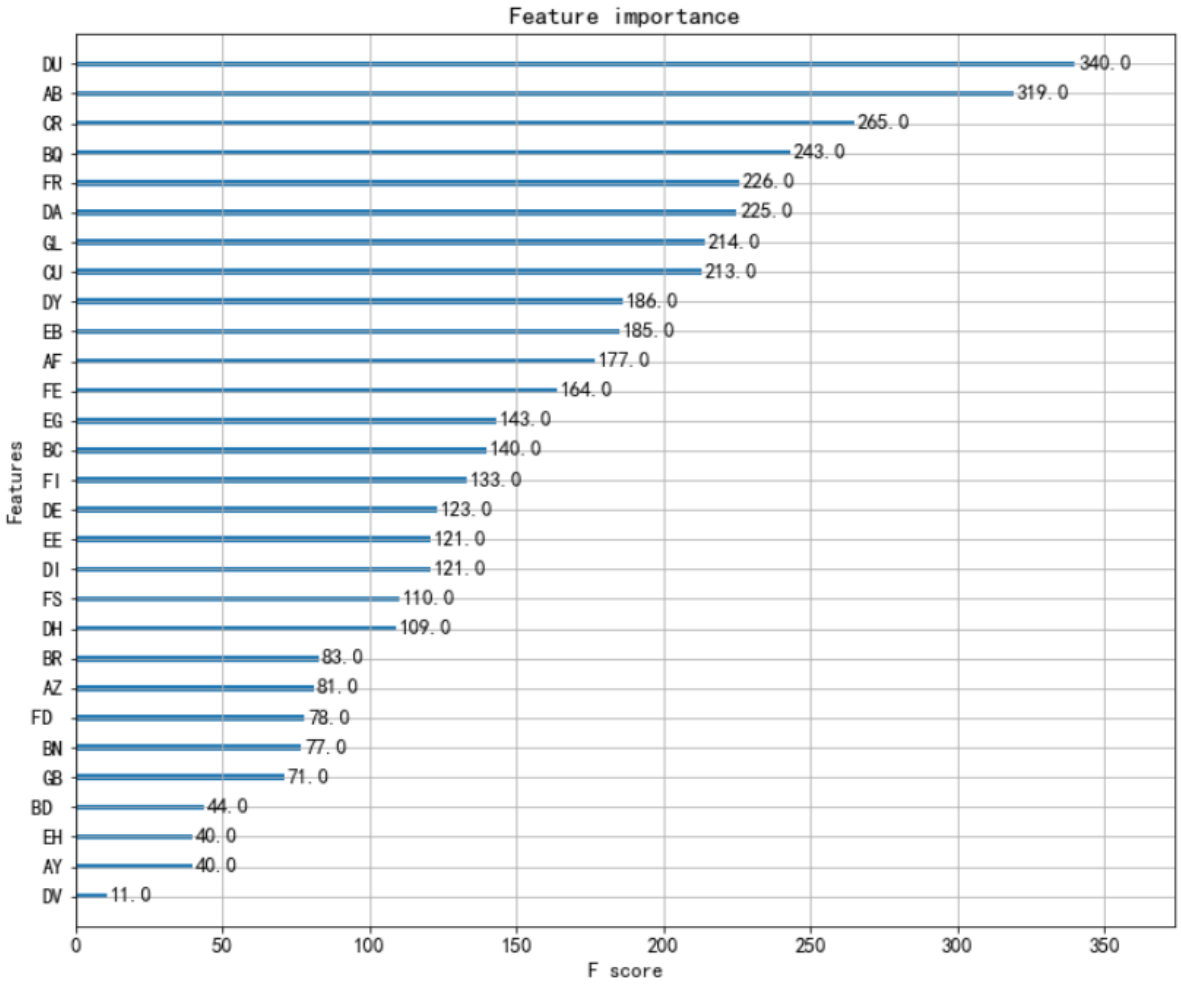


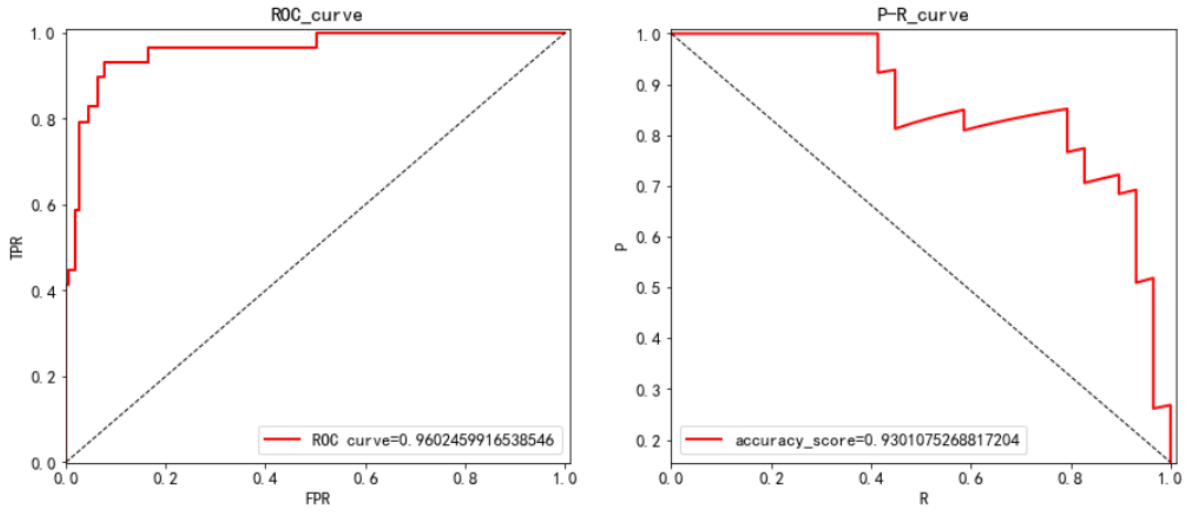
图 4-18 XGBoost 模型变量重要性

4)模型的 ROC 曲线和 P-R 曲线

```
#ROC曲线和P-R曲线
#FTR:假阳性率，TPR：真阳性率
FPR,TPR,thresholds = roc_curve(Y_test,XGBC_model.predict_proba(X_test)[:,1],pos_label=1)
roc_auc = auc(FPR,TPR)  #auc值
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
plt.plot(FPR,TPR,color = 'r',linewidth=2,label=f'ROC curve={roc_auc}')
plt.plot([0,1],[0,1],color='k',linewidth=1,linestyle='--')
plt.xlim([-0.001,1.01])
plt.ylim([-0.001,1.01])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC_curve')
plt.legend(loc = 'best')

plt.subplot(1,2,2)
PRE,REC,thresholds = precision_recall_curve(Y_test,XGBC_model.predict_proba(X_test)[:,1],pos_label=1)
plt.plot(REC,PRE,color = 'r',linewidth=2,label=f'accuracy_score={accuracy_score(Y_test,XGBC_model.predict(X_test))}')
plt.plot([0,1],[1,PRE.min()],color='k',linewidth=1,linestyle='--')
plt.xlim([-0.001,1.01])
plt.ylim([PRE.min()-0.001,1.01])
plt.xlabel('R')
plt.ylabel('P')
plt.title('P-R_curve')
plt.legend(loc = 'best')
plt.show()

print("AUC：",roc_auc)
print("The total accuracy is：",accuracy_score(Y_test,XGBC_model.predict(X_test)))
```




```
AUC：  0.9602459916538546
The total accuracy is：  0.9301075268817204
```

图 4-19 XGBoost ROC 曲线,P_R 曲线

5) 模型可视化（部分），以及最终模型建立

```
#模型可视化（部分）
import xgboost
digraph1 = xgboost.to_graphviz(XGBC_model,num_trees=8)
digraph1
```

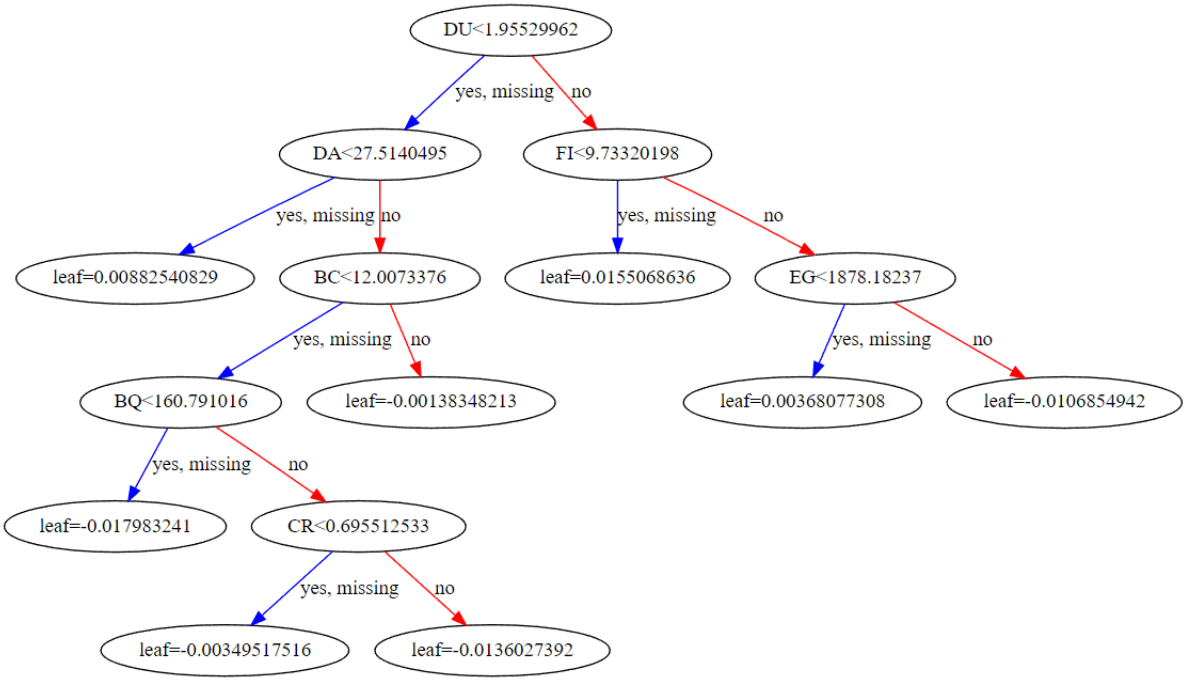


图 4-19 XGBoost 模型可视化(部分)

六、模型应用

1)将模型用于预测测试集

```
# 按照筛选的变量取出测试集数据
test_X = test_.loc[:,selector_X]
# 模型预测测试集
test_pred = XGBC_model.predict(test_X)
# 预测概率
test_pred_prob = XGBC_model.predict_proba(test_X)
```

```
# 构造预测概率dataframe
test_pred_prob_df = pd.DataFrame(test_pred_prob, columns=['class_0', 'class_1'])
test_pred_prob_df
```

	class_0	class_1
0	0.660142	0.339858
1	0.660142	0.339858
2	0.660142	0.339858
3	0.660142	0.339858
4	0.660142	0.339858

```
print('预测结果：',test_pred)
```

预测结果： [0 0 0 0 0]

图 6-1 模型应用预测测试集

2) 构造提交结果

```
submission = pd.DataFrame(columns=['Id','class_0','class_1'])
submission.Id = test_.Id
submission.class_0 = test_pred_prob_df.class_0
submission.class_1 = test_pred_prob_df.class_1
submission.head()
```

	Id	class_0	class_1
0	00eed32682bb	0.660142	0.339858
1	010ebe33f668	0.660142	0.339858
2	02fa521e1838	0.660142	0.339858
3	040e15f562a2	0.660142	0.339858
4	046e85c7cc7f	0.660142	0.339858

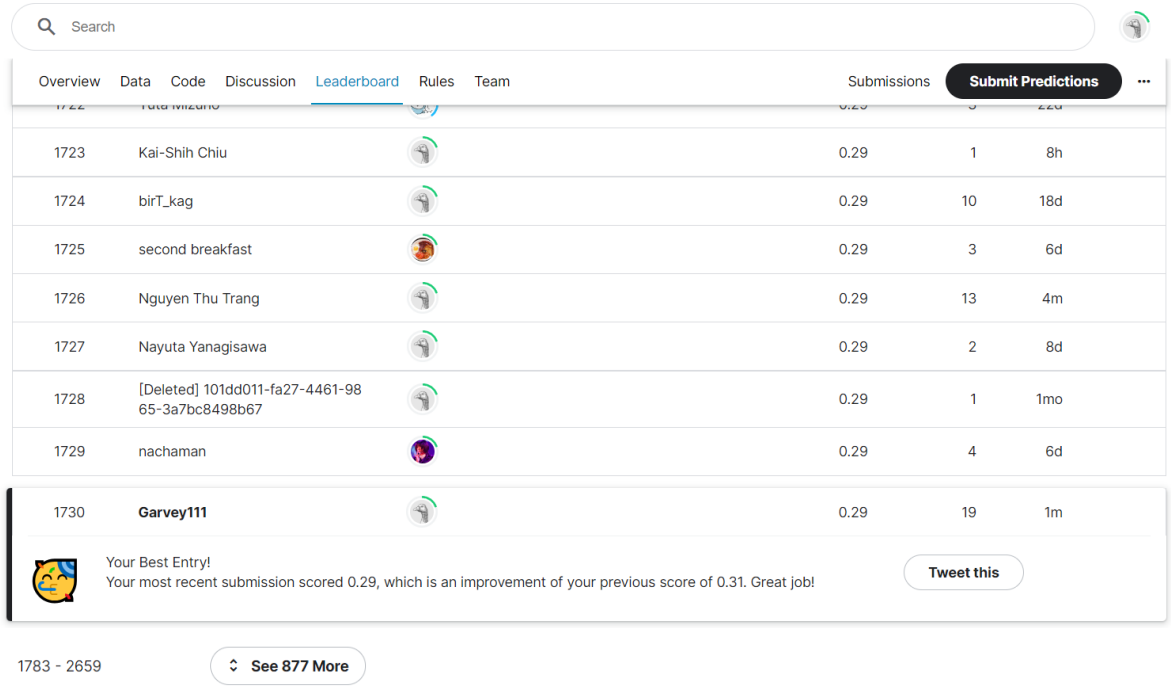
```
submission.to_csv('/kaggle/working/submission.csv',index=False)

#submission.to_csv('submission.csv',index=False)
```

图 6-2 构造提交结果

七、数据分析结论

参与 Kaggle 进行中的竞赛结果：



公榜：1730/2659；排名前 66%

结论：

1.采用基于 XGBClassifier 的 SelectFromModel 进行特征选择，筛选 DU,AB,CR,BQ,FR,DA,GL,CU,DY,EB,AF,FE,EG,BG,FI,DE,EE,DI,FS,DH,BR,AZ,FD,BH,CB,BD,EH,AY,DV 共计 29 个重要变量，这些变量对于预测受试者是否患有 B, D,G 其中一种或者多种疾病发挥着重要的作用，其中所列出的 29 个特征的变量重要性依次递减；

2.同时根据前文提出的诸多问题进行总结：

a.在训练集数据量有限的情况下，应当尽可能地减少暴力删除缺失值，按照赛题给定的实验数据进行有效的填补才是比较好的处理方式；

b.在数据集当中如果数据集的特征变量是脱敏的情况下,我们则不能简单的采取结合主题进行筛选的方式，本文通过基本模型建模以及结合网格搜索的结果进行误差、评估指标的探索，综合评估选取 XGBClassifier 转为最终模型，并且采取基于 XGBClassifier 的 SelectFromModel 特征选择方法,同时对于变量不平衡问题进行探索解决，采用集成学习的阈值调整方法。

c.在进行模型网格调优的时候，如果数据量并不是较大的时候，并不建议采用 10 折交叉验证，通过测试，5 折交叉验证效果更不错，避免模型过拟合；还有对于网格搜索调优的结果应该继续误差验证分析，如果只是简单采用网格搜索的最优参数可能会导致模型过拟合，在后续应用并不会有较好的表现。

附录

数据集情况(部分)

训练集：

	Id	AB	AF	AH	AM	AR	AX	AY	AZ	BC	BD	BN	BP	BQ	BR	BZ	CB	CC	CD	. .	CL	CR	CS	CU	CW	DA	DE
1	000ff2bfdfe9	0.2093...	3109....	85.2...	22....	8.13...	0.699...	0.025...	9.8...	5.5...	4126...	22.5...	17...	152....	823...	25...	47...	0....	23....	. .	1.0...	0....	13.7...	1....	36....	6...	295...
2	007255e47698	0.1452...	978.7...	85.2...	36....	8.13...	3.632...	0.025...	13....	1.2...	5496...	19.4...	15...	14.7...	51....	25...	30...	0....	50....	. .	1.1...	1....	28.3...	1....	37....	7...	178...
3	013f2bd269f5	0.47003	2635....	85.2...	32....	8.13...	6.732...	0.025...	12....	1.2...	5135...	26.4...	12...	219....	482...	25...	32...	0....	85....	. .	1.0...	0....	39.3...	1....	21....	7...	321...
4	043ac50845d5	0.2521...	3819....	120....	77....	8.13...	3.685...	0.025...	11....	1.2...	4169...	23.6...	23...	11.0...	661...	25...	15...	0....	88....	. .	1.4...	0....	41.1...	0....	21....	4...	196...
5	044fb8a146ec	0.3802...	3733....	85.2...	14....	8.13...	3.942...	0.054...	3.3...	102...	5728...	24.0...	32...	149....	607...	25...	82...	0....	72....	. .	1.0...	0....	31.7...	0....	34....	7...	200...
6	04517a3c90bd	0.2093...	2615....	85.2...	8.5...	8.13...	4.013...	0.025...	12....	1.2...	5237...	10.2...	14...	16.5...	642...	25...	18...	0....	80....	. .	1.0...	0....	32.4...	1....	7.0...	5...	135...
7	049232ca8356	0.3482...	1733....	85.2...	8.3...	15.3...	1.913...	0.025...	6.5...	1.2...	5710...	17.6...	14...	344....	719...	25...	38...	0....	78....	. .	1.0...	0....	13.7...	2....	21....	1...	107...
8	057287f2da6d	0.2691...	966.4...	85.2...	21....	8.13...	4.987...	0.025...	9.4...	1.2...	5040...	20.8...	17...	6.19...	701...	25...	12...	0....	71....	. .	1.0...	1....	41.9...	1....	42....	6...	326...
9	0594b00fb30a	0.3461...	3238....	85.2...	28....	8.13...	4.021...	0.025...	8.2...	3.6...	6569...	20.4...	13...		601...	25...	11...	0....	93....	. .	1.0...	1....	29.9...	1....	43....	7...	231...
10	05f2bc0155cd	0.3247...	5188....	85.2...	12....	8.13...	4.593...	0.025...	10....	1.2...	4951...	21.8...	20...	107....	906...	25...	41...	0....	58....	. .	1.0...	0....	36.7...	1....	36....	3...	403...
11	06055f3f6785	0.5042...	6089....	85.2...	189...	17.0...	6.2013	10.31...	14....	212...	4388...	25.4...	19...	344....	51....	59...	47...	0....	132....	. .	1.0...	0....	17.0...	2....	50....	6....	189...
12	06554e7b9979	0.7605...	6957....	200....	23....	8.13...	7.627...	0.025...	16....	1.2...	1208...	26.4...	39...	28.9...	827...	25...	12...	0....	66....	. .	1.0...	0....	44.5...	0....	36....	5...	204...
13	068a4e4bbbab	0.4913...	2627....	138....	48....	8.13...	4.216...	0.025...	12....	1.2...	5688...	23.6...	33...	101....	343...	25...	66...	0....	69....	. .	3.2...	0....	47.2...	0....	36....	6...	353...
14	06c0ddf265c5	0.5341...	4784....	90.7...	8.7...	8.13...	7.003...	0.025...	10....	14....	5535...	24.7...	37...	344....	670...	10...	19...	0....	126....	. .	1.2...	0....	33.2...	1....	31....	3...	211...
15	075bd937ab85	0.5084...	1632....	85.2...	11....	17.6...	7.104...	0.234...	12....	11....	1056...	26.4...	33...	73.2...	276...	13...	12...	0....	23....	. .	1.0...	0....	70.3...	1....	38....	5...	203...

名义测试集：(真正的竞赛评价采用隐藏测试集)

	Id	AB	AF	AH	AM	AR	AX	AY	AZ	BC	BD	BN	BP	BQ	BR	BZ	CB	CC	CD	CF	CH	CL	CR	CS	CU	CW	DA	DE	DF	DH	DI	DL	DN	DU	DV	...
1	00eed32682bb	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	010ebe33f668	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	02fa521e1838	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	040e15f562a2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	046e85c7cc7f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

greeks(训练集的补充元数据集):

	Id	Alpha	Beta	Gamma	Delta	Epsilon
1	000ff2bfdfe9	B	C	G	D	3/19/2019
2	007255e47698	A	C	M	B	Unknown
3	013f2bd269f5	A	C	M	B	Unknown
4	043ac50845d5	A	C	M	B	Unknown
5	044fb8a146ec	D	B	F	B	3/25/2020
6	04517a3c90bd	A	C	M	B	10/1/2019
7	049232ca8356	A	C	M	B	5/29/2019
8	057287f2da6d	A	C	M	B	4/24/2019
9	0594b00fb30a	A	C	M	B	2/18/2019
10	05f2bc0155cd	A	B	M	B	6/19/2020
11	06055f3f6785	D	B	F	B	6/23/2020
12	06554e7b9979	A	C	N	B	12/18/2019
13	068a4e4bbbab	A	C	N	B	11/8/2019
14	06c0ddf265c5	D	B	E	B	9/27/2019
15	075bd937ab85	A	C	M	C	12/13/2018

sample_submission(提交结果的样式):

	Id	class_0	class_1
1	00eed32682bb	0.5	0.5
2	010ebe33f668	0.5	0.5
3	02fa521e1838	0.5	0.5
4	040e15f562a2	0.5	0.5
5	046e85c7cc7f	0.5	0.5