XGBoost_cts

2021 年 4 月 16 日

0.1 导入数据

[1]:
```python
import pandas as pd
df = pd.read_csv('test2.1.csv')
df.set_index('id', inplace=True)
df.head()
```

[1]:
```
              breach  breachNewYear  region  bondBalance     regCap  comScale  \
id
143376.SH          1           2020      18         8.20   3.000000         1
011758127.IB       1           2018      13        30.00  23.162800         2
1380202.IB         1           2018      13         5.50   6.000000         0
150660.SH          1           2019       6        20.00  12.960761         1
150570.SH          1           2019       6        13.79  16.313768         1

              industry  guaCom  typeBond  bondTerm  …  sbreachNewYearGPBR  \
id                                                  …
143376.SH            5       2         2    5.0000  …           7048.5800
011758127.IB         5       2         7    0.7397  …           1466.5189
1380202.IB           4       2         1    7.0000  …           1466.5189
150660.SH            0       3         2    5.0000  …           2106.2400
150570.SH            0       3         2    3.0000  …           2106.2400

              sbreachNewYearGPBE     sSSFR  sbreachNewYearUrbanArea  \
id
143376.SH               10052.99  0.701143                 12421.76
011758127.IB             4637.24  0.316248                  3093.66
1380202.IB               4637.24  0.316248                  3093.66
```

```
150660.SH                3103.20  0.678732                 2585.19
150570.SH                3103.20  0.678732                 2585.19

              sbreachNewYearCityNumber  sbreachNewYearPCDI  \
id
143376.SH                           11            49898.84
011758127.IB                         4            19975.10
1380202.IB                           4            19975.10
150660.SH                            1            39506.15
150570.SH                            1            39506.15

              sbreachNewYearPowerUsage  sbreachNewYearHousePrice  \
id
143376.SH                      4706.22                  15304.00
011758127.IB                   2542.85                   4965.00
1380202.IB                     2542.85                   4965.00
150660.SH                       861.44                  16054.64
150570.SH                       861.44                  16054.64

              sbreachNewYearUnemploymentRate  \
id
143376.SH                               2.52
011758127.IB                            2.58
1380202.IB                              2.58
150660.SH                               3.51
150570.SH                               3.51

              sbreachNewYearMinimumLivingSecurity
id
143376.SH                                 19.4300
011758127.IB                              65.3502
1380202.IB                                65.3502
150660.SH                                  8.0000
150570.SH                                  8.0000

[5 rows x 40 columns]
```

通过如下代码将特征变量和目标变量单独提取出来，代码如下：

[2]:
```
X = df.drop(columns='breach')
y = df['breach']
```

特征变量描述性分析

[3]:
```
X.describe()
```

[3]:

	breachNewYear	region	bondBalance	regCap	comScale	\
count	62647.000000	62647.000000	62647.000000	62647.000000	62647.000000	
mean	2019.998978	11.300797	2230.267537	307.809451	2.145466	
std	0.051466	8.305893	3401.400396	1155.002734	0.793186	
min	2016.000000	0.000000	0.000000	0.080000	0.000000	
25%	2020.000000	3.000000	125.800000	37.759450	2.000000	
50%	2020.000000	11.000000	408.417864	77.504314	2.000000	
75%	2020.000000	18.000000	2177.000000	196.871963	3.000000	
max	2020.000000	30.000000	86951.396040	17395.000000	4.000000	

	industry	guaCom	typeBond	bondTerm	bondIssuePrice	\
count	62647.000000	62647.000000	62647.000000	62647.000000	62647.000000	
mean	8.799958	2.014143	4.996169	0.965660	98.635466	
std	2.574340	0.118080	1.542537	1.551403	1.360883	
min	0.000000	2.000000	0.000000	0.019200	94.786700	
25%	10.000000	2.000000	5.000000	0.251400	97.648300	
50%	10.000000	2.000000	5.000000	0.504100	99.086000	
75%	10.000000	2.000000	5.000000	1.000000	100.000000	
max	11.000000	3.000000	11.000000	30.000000	119.020000	

	…	sbreachNewYearGPBR	sbreachNewYearGPBE	sSSFR	\
count	…	62647.000000	62647.000000	62647.000000	
mean	…	5334.420784	8263.695302	0.624341	
std	…	2934.926543	3395.626078	0.180965	
min	…	272.890000	1647.430000	0.151448	
25%	…	3052.929700	5850.960000	0.449738	
50%	…	5817.100000	7408.190000	0.677853	
75%	…	7048.580000	10052.990000	0.785226	
max	…	12654.530000	17297.850000	0.876006	

检查特征变量是否存在空值

```
       sbreachNewYearUrbanArea  sbreachNewYearCityNumber  sbreachNewYearPCDI  \
count             62647.000000              62647.000000        62647.000000
mean              10316.451414                  8.684933        43892.913490
std                6260.675045                  6.727405        17751.557356
min                 688.150000                  1.000000        18118.090000
25%                5102.940000                  1.000000        27679.710000
50%                8186.150000                 10.000000        39014.280000
75%               16410.000000                 14.000000        67755.910000
max               23206.320000                 21.000000        69441.560000

       sbreachNewYearPowerUsage  sbreachNewYearHousePrice  \
count              62647.000000              62647.000000
mean                2846.948581              17584.913993
std                 1941.524868              11524.857927
min                  354.890000               4965.000000
25%                 1166.400000               8070.000000
50%                 2214.300000              11637.000000
75%                 3856.060000              30677.000000
max                 6695.850000              35905.000000

       sbreachNewYearUnemploymentRate  sbreachNewYearMinimumLivingSecurity
count                    62647.000000                         62647.000000
mean                         2.675670                            18.252859
std                          0.828362                            14.788203
min                          1.300000                             3.680000
25%                          2.250000                             6.540000
50%                          2.730000                            14.850000
75%                          3.500000                            21.570000
max                          4.160000                           134.500000

[8 rows x 39 columns]
```

0.2 数据预处理

检查特征变量是否存在空值

[4]:
```
X.isnull().sum().sort_values(ascending = False)
```

[4]:
```
sbreachNewYearMinimumLivingSecurity    0
bondIssuePrice                         0
breachNewYearGDPgrowthindex            0
intBeginDateGDP                        0
breachNewYearGDP                       0
bondIssuerRating                       0
intBeginDate                           0
bondInterest                           0
bondTotalAmt                           0
bondTerm                               0
breachNewYearFinanceScaleRate          0
typeBond                               0
guaCom                                 0
industry                               0
comScale                               0
regCap                                 0
bondBalance                            0
region                                 0
intBeginDateGDPgrowthindex             0
intBeginDateFinanceScaleRate           0
sbreachNewYearUnemploymentRate         0
sbreachNewYearGPBR                     0
sbreachNewYearHousePrice               0
sbreachNewYearPowerUsage               0
sbreachNewYearPCDI                     0
sbreachNewYearCityNumber               0
sbreachNewYearUrbanArea                0
sSSFR                                  0
sbreachNewYearGPBE                     0
sbreachNewYearPD                       0
breachNewYearConsumptionLevel          0
sbreachNewYearPNGT                     0
sbreachNewYearCPI                      0
sbreachNewYearGDPGrowthIndex           0
sbreachNewYearGDP                      0
```

```
intBeginDateConsumptionLevelIndex      0
breachNewYearConsumptionLevelIndex     0
intBeginDateConsumptionLevel           0
breachNewYear                          0
dtype: int64
```

导入工具库，使特征变量归一化

[5]:
```python
from sklearn.preprocessing import StandardScaler
X_new = StandardScaler().fit_transform(X)
pd.DataFrame(X_new).head()  # 查看归一化后的数据
```

[5]:
```
           0         1         2         3         4         5         6  \
0   0.019850  0.806567 -0.653285 -0.263906 -1.444144 -1.476102 -0.119773
1 -38.840941  0.204580 -0.646876 -0.246449 -0.183396 -1.476102 -0.119773
2 -38.840941  0.204580 -0.654079 -0.261308 -2.704892 -1.864555 -0.119773
3 -19.410545 -0.638202 -0.649816 -0.255282 -1.444144 -3.418364  8.349112
4 -19.410545 -0.638202 -0.651642 -0.252379 -1.444144 -3.418364  8.349112

          7         8         9  …        29        30        31        32  \
0 -1.942381  2.600467  1.002691  …  0.584060  0.526945  0.424402  0.336278
1  1.299060 -0.145650  1.002691  … -1.317898 -1.067987 -1.702511 -1.153685
2 -2.590669  3.889633  1.002691  … -1.317898 -1.067987 -1.702511 -1.153685
3 -1.942381  2.600467  1.002691  … -1.099928 -1.519760  0.300559 -1.234902
4 -1.942381  1.311301  1.002691  … -1.099928 -1.519760  0.300559 -1.234902

         33        34        35        36        37        38
0  0.344128  0.338335  0.957642 -0.197914 -0.187926  0.079601
1 -0.696401 -1.347375 -0.156630 -1.095026 -0.115494  3.184817
2 -0.696401 -1.347375 -0.156630 -1.095026 -0.115494  3.184817
3 -1.142342 -0.247122 -1.022662 -0.132781  1.007214 -0.693319
4 -1.142342 -0.247122 -1.022662 -0.132781  1.007214 -0.693319

[5 rows x 39 columns]
```

提取完特征变量后，通过如下代码将数据拆分为训练集及测试集：

[6]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2)
```

利用主成分分析法进行特征变量降维

[7]:
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=0.9)# 保证降维后的数据保持 90% 的信息
pca.fit(X_train)
```

[7]:
```
PCA(n_components=0.9)
```

对训练集和测试集进行数据降维, 并赋值给 **X_X_train_pca,X_test_pca**

[8]:
```python
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

通过如下代码验证 **PCA** 是否降维:

[9]:
```python
print(X_train_pca.shape)
print(X_test_pca.shape)
```

```
(50117, 14)
(12530, 14)
```

查看降维的系数

[10]:
```python
PCAcom=pca.components_
dfPCA=pd.DataFrame(PCAcom)
dfPCA.to_csv("PCA.csv")
dfPCA
```

[10]:

	0	1	2	3	4	5	6 \
0	0.005414	-0.261641	0.201375	0.143964	-0.179318	-0.021749	0.018476
1	-0.249120	-0.018444	0.028650	0.003210	-0.048541	-0.015921	0.010671
2	-0.324117	0.015681	-0.025253	0.000570	0.027748	-0.006864	0.005290
3	-0.049830	-0.083398	-0.056460	-0.001266	-0.069610	-0.056387	0.014211
4	0.018468	-0.007640	-0.156524	0.255056	0.121072	-0.346529	0.368358
5	0.002931	0.044293	0.149522	0.097863	-0.312515	-0.111501	0.255668
6	-0.005508	0.160262	0.447980	0.466017	0.157928	0.390428	0.203208
7	0.002300	0.181068	-0.028042	0.110240	-0.065850	-0.295894	0.015162
8	-0.006752	-0.144241	0.017144	0.317289	0.099363	-0.007104	0.104263

9	-0.005576	0.209624	0.062662	-0.057162	-0.134852	0.051007	-0.400854
10	0.008150	-0.153308	-0.009274	-0.076492	-0.247106	-0.141835	-0.336211
11	-0.002673	0.056463	-0.131064	-0.140525	0.528024	-0.021661	-0.257077
12	0.000950	0.218238	-0.144117	-0.087316	-0.145301	-0.095361	0.220177
13	-0.000171	0.059409	-0.251713	-0.309031	-0.296348	0.616129	0.341806

	7	8	9	…	29	30	31	32 \
0	0.003633	0.039442	0.069298	…	0.097060	-0.042673	0.299050	0.125234
1	-0.018449	0.022407	-0.013437	…	0.026153	0.015676	0.035021	0.029785
2	-0.014926	0.015086	0.024326	…	-0.085977	-0.083696	-0.051191	-0.066331
3	-0.001093	0.037314	0.052397	…	0.404955	0.421237	0.178104	0.333413
4	-0.300338	0.484879	0.163539	…	-0.099889	-0.042669	-0.135536	0.145139
5	-0.141385	0.276635	0.214202	…	0.102445	0.052532	0.122422	-0.298029
6	0.033459	0.064769	-0.365974	…	-0.040799	-0.037451	-0.046610	0.004403
7	0.399116	-0.108242	0.485995	…	-0.118274	-0.129880	-0.075283	-0.027670
8	0.465517	-0.118482	0.077203	…	0.065384	0.157518	-0.081156	0.027054
9	-0.282054	0.117496	0.035311	…	-0.010403	0.053339	-0.155427	-0.067104
10	0.128525	0.027761	-0.182915	…	-0.097299	-0.050560	-0.075962	0.114789
11	-0.038279	0.121202	0.085534	…	0.087264	0.005579	0.136914	-0.053669
12	0.407298	-0.097710	-0.133195	…	0.010233	-0.037379	0.026648	-0.045056
13	-0.045014	-0.016077	0.230853	…	-0.063529	-0.042628	-0.048482	0.104616

	33	34	35	36	37	38
0	-0.296093	0.349336	-0.168357	0.353580	-0.192167	-0.224409
1	-0.009831	0.030560	0.006173	0.028204	-0.017795	-0.008411
2	-0.038307	-0.028827	-0.070628	-0.018159	0.015751	0.025962
3	0.240146	0.048087	0.373148	0.016256	-0.097193	-0.084258
4	0.018550	-0.050570	-0.047247	0.009333	-0.229272	0.093061
5	0.027682	-0.014587	0.051319	-0.086801	0.413709	-0.001481
6	0.048765	-0.062521	0.044593	-0.059625	-0.037682	-0.147296
7	0.002489	-0.091328	0.008492	-0.070806	-0.098140	-0.225452
8	0.099587	0.000138	-0.054739	0.029214	-0.010529	0.448535
9	0.124183	-0.084917	-0.119359	0.003918	-0.305265	0.082990
10	0.038553	-0.052754	0.012476	-0.051094	0.051530	0.192821
11	-0.095704	0.072650	0.077514	0.027503	0.142345	-0.261820
12	-0.034352	-0.029787	-0.001005	-0.009531	-0.096551	-0.294061

```
13 -0.018623 -0.023835  0.029465 -0.033520 -0.063178 -0.017483

[14 rows x 39 columns]
```

查看此时降维后的 **X_train_pca** 和 **X_test_pca**

[11]:
```python
pd.DataFrame(X_train_pca).head()
pd.DataFrame(X_test_pca).head()
```

[11]:
```
          0         1         2         3         4         5         6  \
0 -0.863884 -2.433233  1.482733  1.858663 -0.180729 -1.222185  0.707273
1 -2.321023 -1.273418  0.941012 -0.559762 -0.161169 -0.617901  0.627936
2  4.235525 -0.978434  0.707951  0.246900 -0.394597 -1.234797 -0.689885
3 -1.123108 -0.838234 -0.200331  5.410859  0.889762  1.849604  0.128327
4 -2.339678  0.939150 -0.216356 -2.224602  2.871920  3.106451 -0.391717

          7         8         9        10        11        12        13
0 -0.649258 -0.490956 -0.727238 -0.310451  0.707288 -0.102438 -0.066412
1 -0.350038 -0.553530 -0.760497 -0.171184  0.546280 -0.248033 -0.023432
2  0.437284 -0.344212  0.415278 -0.458113 -1.132259 -0.338352  0.940863
3 -1.505360 -1.587084  2.203331 -0.887822  0.401666 -1.099852  0.651840
4 -0.898591 -1.866768  1.828021  0.587925  0.438961  0.852280 -0.156096
```

0.3 常见机器学习模型默认超参数的结果

[12]:
```python
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kfold = KFold(n_splits=10)
# xgboost
from xgboost import XGBClassifier
xgbc_model=XGBClassifier(use_label_encoder=False, eval_metric='auc')

# 随机森林
from sklearn.ensemble import RandomForestClassifier
rfc_model=RandomForestClassifier()

# ET
from sklearn.ensemble import ExtraTreesClassifier
```

```python
et_model=ExtraTreesClassifier()

# 朴素贝叶斯
from sklearn.naive_bayes import GaussianNB
gnb_model=GaussianNB()

#K 最近邻
from sklearn.neighbors import KNeighborsClassifier
knn_model=KNeighborsClassifier()

# 逻辑回归
from sklearn.linear_model import LogisticRegression
lr_model=LogisticRegression()

# 决策树
from sklearn.tree import DecisionTreeClassifier
dt_model=DecisionTreeClassifier()

# 支持向量机
from sklearn.svm import SVC
svc_model=SVC()

#result = cross_val_score(model , X_train_pca, y_train , cv=kfold).mean()
print("\n使用 10 折交叉验证方法得到模型的准确率（每次迭代的准确率的均值）：")
print("\tXGBoost 模型：",cross_val_score(xgbc_model , X_train_pca, y_train␣
↪,scoring='roc_auc', cv=kfold).mean())
print("\t随机森林模型：",cross_val_score(rfc_model , X_train_pca, y_train␣
↪,scoring='roc_auc',  cv=kfold).mean())
print("\tET 模型：",cross_val_score(et_model , X_train_pca, y_train ,␣
↪scoring='roc_auc', cv=kfold).mean())
print("\t高斯朴素贝叶斯模型：",cross_val_score(gnb_model , X_train_pca, y_train␣
↪,scoring='roc_auc',  cv=kfold).mean())
print("\tK 最近邻模型：",cross_val_score(knn_model , X_train_pca, y_train␣
↪,scoring='roc_auc',  cv=kfold).mean())
print("\t逻辑回归：",cross_val_score(lr_model , X_train_pca, y_train ,␣
↪scoring='roc_auc', cv=kfold).mean())
```

```
print("\t决策树: ",cross_val_score(dt_model , X_train_pca, y_train␣
↪,scoring='roc_auc',  cv=kfold).mean())
print("\t支持向量机: ",cross_val_score(svc_model , X_train_pca, y_train␣
↪,scoring='roc_auc',  cv=kfold).mean())
```

```
使用 10 折交叉验证方法得到模型的准确率（每次迭代的准确率的均值):
	XGBoost 模型:  0.9965806288340227
	随机森林模型:  0.9749761157269237
	ET 模型:  0.8061721434490107
	高斯朴素贝叶斯模型:  0.9349179297501257
	K 最近邻模型:  0.932024739354347
	逻辑回归:  0.9823637357498212
	决策树:  0.7555230147271651
	支持向量机:  0.9603518034122157
```

0.4 引入 XGBoost 分类器进行模型训练

基于主成分分析法降维后数据，代码如下：

[13]:
```
# 引入 XGBoost 分类器进行模型训练了，基于主成分分析法降维后数据，代码如下：
from xgboost import XGBClassifier
clf_pca = XGBClassifier(max_depth=5,
                        learning_rate=0.1,
                        n_estimators=500,# 迭代次数
                        gamma=0,
                        min_child_weight=1,
                        subsample=0.8,
                        colsample_bytree=0.8,
                        colsample_bylevel=1,
                        reg_alpha=0,
                        reg_lambda=1,
                        use_label_encoder=False,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')
clf_pca.fit(X_train_pca, y_train)
```

[13]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=500, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.8,
              tree_method='exact', use_label_encoder=False,
              validate_parameters=1, verbosity=0)
```

[14]:
```
y_pred_pca = clf_pca.predict(X_test_pca)# 降维后的数据及模型
y_pred_pca
```

[14]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[15]:
```
# 通过和之前章节类似的代码，我们可以将预测值和实际值进行对比：
b = pd.DataFrame()  # 创建一个空 DataFrame
b['预测值'] = list(y_pred_pca)
b['实际值'] = list(y_test)
b
```

[15]:

	预测值	实际值
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
…	…	…
12525	0	0
12526	0	0
12527	0	0
12528	0	0
12529	0	0

[12530 rows x 2 columns]

[16]:
```
# 所有测试集数据的预测准确度，可以使用如下代码：
from sklearn.metrics import accuracy_score
```

```
score = accuracy_score(y_pred_pca, y_test)
score
```

[16]: 0.999122106943336

[17]:
```
# 我们还可以通过 XGBClassifier() 自带的 score() 函数来查看模型预测的准确度评分，代码如下
clf_pca.score(X_test_pca, y_test)
```

[17]: 0.999122106943336

未降维的数据建模

[18]:
```
# 划分为训练集和测试集之后，就可以引入 XGBoost 分类器进行模型训练了，基于未降维的数据，代码如下：
from xgboost import XGBClassifier
clf = XGBClassifier(max_depth=5,
                        learning_rate=0.1,
                        n_estimators=500,# 迭代次数
                        gamma=0,
                        min_child_weight=1,
                        subsample=0.8,
                        colsample_bytree=0.8,
                        colsample_bylevel=1,
                        reg_alpha=0,
                        reg_lambda=1,
                        use_label_encoder=False,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')
clf.fit(X_train, y_train)
```

[18]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=500, n_jobs=16, num_parallel_tree=1, random_state=0,
```

```
                     reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.8,
                     tree_method='exact', use_label_encoder=False,
                     validate_parameters=1, verbosity=0)
```

[19]:
```
# 模型搭建完毕后，通过如下未降维数据，预测测试集数据：
y_pred = clf.predict(X_test)
y_pred  # 打印预测结果
```

[19]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[20]:
```
# 我们可以将预测值和实际值进行对比：
a = pd.DataFrame()  # 创建一个空 DataFrame
a['预测值'] = list(y_pred)
a['实际值'] = list(y_test)
a
```

[20]:

	预测值	实际值
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
…	…	…
12525	0	0
12526	0	0
12527	0	0
12528	0	0
12529	0	0

[12530 rows x 2 columns]

[21]:
```
# 所有测试集数据的预测准确度，可以使用如下代码：
from sklearn.metrics import accuracy_score
score = accuracy_score(y_pred, y_test)
score
```

[21]: 0.9990422984836392

```python
[22]: # 我们还可以通过 XGBClassifier() 自带的 score() 函数来查看模型预测的准确度评分，代码如下
clf.score(X_test, y_test)
```

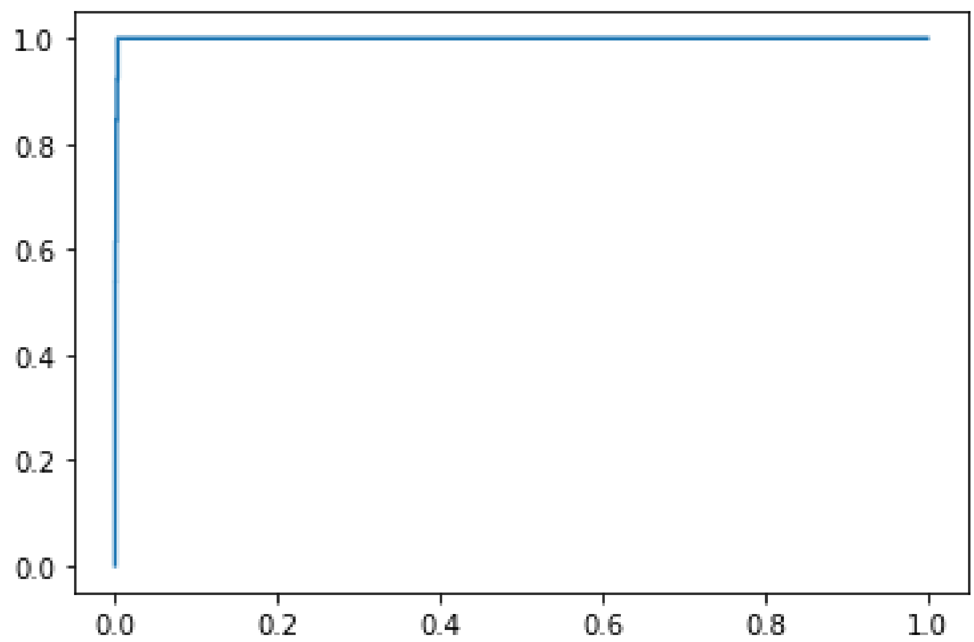
```
[22]: 0.9990422984836392
```

可以看出主成分降维后有一定的提升，对主成分降维后的模型进行分析

```python
[23]: # XGBClassifier 分类器本质预测的并不是准确的 0 或 1 的分类，而是预测其属于某一分类的概率，可以通过 predict_proba() 函数查看预测属于各个分类的概率，代码如下：
y_pred_pca_proba = clf_pca.predict_proba(X_test_pca)
print(y_pred_pca_proba[0:5])  # 查看前 5 个预测的概率
```

```
[[9.9999970e-01 3.0106258e-07]
 [9.9999982e-01 1.7359697e-07]
 [9.9999952e-01 4.9591586e-07]
 [9.9999255e-01 7.4715372e-06]
 [9.9999118e-01 8.8266861e-06]]
```

```python
[24]: # 利用相关代码绘制 ROC 曲线来评估模型预测的效果：
from sklearn.metrics import roc_curve
fpr, tpr, thres = roc_curve(y_test, y_pred_pca_proba[:,1])
import matplotlib.pyplot as plt
plt.plot(fpr, tpr)
plt.show()
```



[25]:
```python
# 通过如下代码求出模型的 AUC 值：
from sklearn.metrics import roc_auc_score
score = roc_auc_score(y_test, y_pred_pca_proba[:,1])

score
```

[25]: 0.9992318139637785

[26]:
```python
# 我们可以通过查看各个特征的特征重要性 (feature importance) 判断最重要的特征变量：
clf_pca.feature_importances_
```

[26]:
```
array([0.05653665, 0.16274147, 0.06217598, 0.04881957, 0.11478559,
       0.05247765, 0.05615563, 0.06749621, 0.06013201, 0.07862388,
       0.06831793, 0.04401254, 0.06697358, 0.06075124], dtype=float32)
```

[31]:
```python
# 进行整理，方便结果呈现，代码如下：
features = X.columns  # 获取特征名称
importances = clf.feature_importances_  # 获取特征重要性

# 通过二维表格形式显示
```

```python
importances_df = pd.DataFrame()
importances_df['特征名称'] = features
importances_df['特征重要性'] = importances
importances_df.sort_values('特征重要性', ascending=False)
```

[31]:

	特征名称	特征重要性
0	breachNewYear	0.277536
14	breachNewYearGDP	0.197524
16	breachNewYearGDPgrowthindex	0.091247
9	bondIssuePrice	0.059798
2	bondBalance	0.032052
3	regCap	0.021941
37	sbreachNewYearUnemploymentRate	0.020941
13	bondIssuerRating	0.019489
25	sbreachNewYearGDPGrowthIndex	0.017744
38	sbreachNewYearMinimumLivingSecurity	0.016971
8	bondTerm	0.016337
5	industry	0.015551
34	sbreachNewYearPCDI	0.015037
4	comScale	0.014552
31	sSSFR	0.014523
30	sbreachNewYearGPBE	0.013652
27	sbreachNewYearPNGT	0.013287
1	region	0.012141
24	sbreachNewYearGDP	0.011487
35	sbreachNewYearPowerUsage	0.010989
36	sbreachNewYearHousePrice	0.010973
29	sbreachNewYearGPBR	0.010941
11	bondInterest	0.010538
7	typeBond	0.010304
28	sbreachNewYearPD	0.007751
10	bondTotalAmt	0.007614
32	sbreachNewYearUrbanArea	0.007261
6	guaCom	0.006888
12	intBeginDate	0.006784
26	sbreachNewYearCPI	0.006311

```
33               sbreachNewYearCityNumber  0.006037
15                         intBeginDateGDP  0.005957
19           intBeginDateFinanceScaleRate  0.005564
17             intBeginDateGDPgrowthindex  0.004279
23    intBeginDateConsumptionLevelIndex  0.000000
22   breachNewYearConsumptionLevelIndex  0.000000
21           intBeginDateConsumptionLevel  0.000000
20          breachNewYearConsumptionLevel  0.000000
18          breachNewYearFinanceScaleRate  0.000000
```

0.5 超参数优化

网格搜索法（**3** 个超参数优化，**10** 折交叉验证）

[34]:
```python
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
parameters = {  'max_depth': [1, 2, 3, 5, 10],
    'learning_rate': [ 0.02, 0.05, 0.1, 0.15],
  'n_estimators': [ 200, 300, 500, 700]}  # 指定模型中参数的范围
```

[35]:
```python
# 下面我们将数据传入网格搜索模型并输出参数最优值:
clf_pca = XGBClassifier(gamma=0,
                        min_child_weight=1,
                        subsample=0.8,
                        colsample_bytree=0.8,
                        colsample_bylevel=1,
                        reg_alpha=0,
                        reg_lambda=1,
                        use_label_encoder=False,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')  # 构建模型
grid_search = GridSearchCV(clf_pca, parameters, scoring='roc_auc', cv=10)
grid_search.fit(X_train_pca, y_train)  # 传入数据
print("参数的最佳取值：:",grid_search.best_params_)
print("最佳模型得分:",grid_search.best_score_)
```

```
参数的最佳取值：: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 300}
最佳模型得分: 0.9954505509139947
```

随机网格搜索法（**11** 个超参数优化，**10** 折交叉验证）

[36]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
parameters = {  'max_depth':[1, 2, 3, 5, 10],
    'learning_rate': [ 0.02, 0.05, 0.1, 0.15],
    'n_estimators': [ 100,200, 300, 500, 700],
    'min_child_weight': [0, 2, 5, 10, 20],
    'max_delta_step': [0, 0.2, 0.6, 1, 2],
    'subsample': [0.6, 0.7, 0.8, 0.85, 0.95],
    'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9],
    'reg_alpha': [0, 0.25, 0.5, 0.75, 1],
    'reg_lambda': [0.2, 0.4, 0.6, 0.8, 1],
    'scale_pos_weight': [0.2, 0.4, 0.6, 0.8, 1],
    'gamma':[0,0.01,0.05,0.1,0.2,0.4,0.6]}  # 指定模型中参数的范围
```

[37]:
```
# 下面我们将数据传入网格搜索模型并输出参数最优值:
clf_pca = XGBClassifier(colsample_bylevel=1,
                        use_label_encoder=False,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')  # 构建模型
Rgrid_search = RandomizedSearchCV(clf_pca, parameters, scoring='roc_auc', cv=10)
Rgrid_search.fit(X_train_pca, y_train)  # 传入数据
print("参数的最佳取值: :",Rgrid_search.best_params_)
print("最佳模型得分:",Rgrid_search.best_score_)
```

```
参数的最佳取值: : {'subsample': 0.7, 'scale_pos_weight': 0.4, 'reg_lambda': 0.4,
'reg_alpha': 0.75, 'n_estimators': 200, 'min_child_weight': 0, 'max_depth': 2,
'max_delta_step': 1, 'learning_rate': 0.1, 'gamma': 0, 'colsample_bytree': 0.5}
最佳模型得分: 0.9887431918195002
```

贝叶斯优化法（**11** 个超参数优化，**10** 折交叉验证）

[78]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from bayes_opt import BayesianOptimization
import numpy as np
rf = XGBClassifier(colsample_bylevel=1,
```

```
                        use_label_encoder=False,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')
def rf_cv(max_depth, learning_rate, n_estimators, min_child_weight,␣
 ↪max_delta_step, subsample,␣
 ↪colsample_bytree,reg_alpha,reg_lambda,scale_pos_weight,gamma):
    val = cross_val_score(
         XGBClassifier(max_depth = int(max_depth),
                       learning_rate = float(learning_rate),
                       use_label_encoder=False,
                       n_estimators = int(n_estimators),
                        min_child_weight = int(min_child_weight),
                        max_delta_step = float(max_delta_step),
                        subsample = float(subsample),
                        colsample_bytree = float(colsample_bytree),
                        reg_alpha = float(reg_alpha),
                        reg_lambda = float(reg_lambda),
                        scale_pos_weight = float(scale_pos_weight),
                        gamma = float(gamma)
        ),
        X_train_pca, y_train , scoring='roc_auc', cv=10
    ).mean()
    return val
rf_bo = BayesianOptimization(
        rf_cv,
                {'max_depth':(1,10),
                'learning_rate': (0.02,0.2),
                'n_estimators': (200,1000),
                'min_child_weight': (0,20),
                'max_delta_step': (0.01,2),
                'subsample':(0.6,1),
                'colsample_bytree': (0.5,1),
                'reg_alpha': (0.01,1),
                'reg_lambda': (0.2,1),
                'scale_pos_weight': (0.2,1),
```

```
                 'gamma':(0,1)}
    )
rf_bo.maximize()
rf_bo.max
```

```
|   iter    |  target   | colsam... |   gamma   | learni... | max_de... |
max_depth | min_ch... | n_esti... | reg_alpha | reg_la... | scale_... |
subsample |
--------------------------------------------------------------------------------
-----------------------------------------------------------------------------
|  1        |  0.8105   |  0.9091   |  0.4019
|  0.1384   |  0.1954   |  2.995    |  13.52
|  457.2    |  0.3792   |  0.3412   |  0.4227
|  0.8768   |
|  2        |  0.9636   |  0.8771   |  0.1358

 |  0.08338  |  1.726    |  8.928    |  10.2

 |  740.2    |  0.07443  |  0.5078   |

0.4884    |  0.792    |
|  3        |  0.9561   |  0.712    |  0.5591
|  0.1118   |  1.943    |  2.639    |  15.56
|  389.5    |  0.2704   |  0.5633   |  0.3893
|  0.7282   |
|  4        |  0.948    |  0.7099   |  0.9354
|  0.187    |  1.729    |  6.012    |  13.16
|  899.1    |  0.5194   |  0.993    |  0.4271
|  0.9334   |
|  5        |  0.952    |  0.7667   |  0.7983
|  0.04016  |  1.436    |  2.713    |  15.45
|  748.9    |  0.5374   |  0.2437   |  0.492
|  0.9096   |
|  6        |  0.9406   |  0.7476   |  0.1786
|  0.05076  |  0.2513   |  8.263    |  10.12
|  739.2    |  0.3089   |  0.7847   |  0.5438
|  0.9896   |
|  7        |  0.961    |  1.0      |  0.0
```

```
|  0.2      |  2.0      |  10.0     |  7.782
|  748.4    |  0.01     |  0.2      |  0.2
|  0.6      |
|  8        |  0.9912   |  0.7867   |  0.1822

 |  0.1394   |  0.5469   |  6.093    |

3.688     |  393.6    |  0.01974  |  0.2285   |
 0.6548   |  0.9101   |
|  9        |  0.9784   |  0.739    |  0.3482
|  0.02     |  2.0      |  6.805    |  7.534
|  402.2    |  0.5512   |  0.2919   |  0.8042
|  0.8226   |
|  10       |  0.9937   |  0.7251   |  0.7631

 |  0.1749   |  0.7092   |  9.413    |

1.495     |  382.5    |  0.3504   |  0.6068   |
 0.8254   |  0.8671   |
|  11       |  0.993    |  0.9754   |  0.4161
|  0.08282  |  1.08     |  2.048    |  0.5338
|  372.1    |  0.791    |  0.6683   |  0.9558
|  0.6522   |
|  12       |  0.5      |  0.5      |  1.0
|  0.2      |  0.01     |  10.0     |  10.12
|  370.9    |  0.01     |  0.2      |  0.2
|  1.0      |
|  13       |  0.9615   |  0.7631   |  0.5666
|  0.06482  |  0.3198   |  1.604    |  0.2663
|  386.4    |  0.6314   |  0.4442   |  0.7655
|  0.6536   |
|  14       |  0.5      |  1.0      |  0.0
|  0.2      |  0.01     |  1.0      |  18.71
|  402.8    |  0.01     |  1.0      |  0.2
|  1.0      |
|  15       |  0.9917   |  0.5      |  1.0
|  0.02     |  2.0      |  10.0     |  0.0
|  402.0    |  1.0      |  0.2      |  1.0
```

| 0.6 |
| 16 | 0.9942 | 0.7423 | 0.4041
| 0.04373 | 0.8069 | 8.847 |
0.6126 | 414.1 | 0.2714 | 0.5765 |
0.3744 | 0.6482 |
| 17 | 0.9919 | 0.5429 | 0.8305
| 0.02731 | 0.9236 | 9.638 | 2.949
| 426.5 | 0.1688 | 0.639 | 0.8351
| 0.9705 |
| 18 | 0.9759 | 0.8397 | 0.04041
| 0.152 | 1.747 | 1.264 | 5.309
| 420.5 | 0.8898 | 0.8954 | 0.6401
| 0.7673 |
| 19 | 0.7106 | 0.6113 | 0.0
| 0.2 | 0.01 | 1.0 | 0.0
| 433.6 | 0.02295 | 1.0 | 0.2
| 0.6 |
| 20 | 0.6021 | 0.5 | 1.0
| 0.02 | 0.05309 | 10.0 | 9.934
| 419.8 | 1.0 | 0.2 | 1.0
| 1.0 |
| 21 | 0.9918 | 0.9512 | 0.6888
| 0.08462 | 0.8539 | 3.999 | 2.522
| 408.4 | 0.08055 | 0.257 | 0.5133
| 0.992 |
| 22 | 0.9939 | 0.8566 | 0.6244
| 0.1438 | 1.224 | 3.238 | 0.1618
| 401.0 | 0.5514 | 0.29 | 0.6891
| 0.6143 |
| 23 | 0.9838 | 0.9121 | 0.6801
| 0.124 | 0.9667 | 5.221 | 8.234
| 385.8 | 0.2203 | 0.6774 | 0.9965
| 0.7873 |
| 24 | 0.9613 | 1.0 | 1.0
| 0.2 | 2.0 | 10.0 | 20.0
| 742.9 | 1.0 | 0.2 | 1.0

```
|  0.6      |
|  25       |  0.9596   |  1.0      |  0.0
|  0.2      |  2.0      |  10.0     |  20.0
|  756.4    |  1.0      |  1.0      |  1.0
|  0.6      |
|  26       |  0.9542   |  0.62     |  0.6324
|  0.08352  |  0.786    |  4.88     |  9.44
|  760.9    |  0.1337   |  0.5582   |  0.3744
|  0.8572   |
|  27       |  0.5      |  0.5767   |  0.252
|  0.125    |  0.01954  |  6.348    |  19.03
|  768.4    |  0.3284   |  0.4905   |  0.2751
|  0.9766   |
|  28       |  0.9862   |  0.6389   |  1.0
|  0.07411  |  2.0      |  1.0      |  2.687
|  754.1    |  1.0      |  1.0      |  1.0
|  0.8474   |
|  29       |  0.9932   |  0.5335   |  0.02304
|  0.09671  |  1.255    |  8.851    |  0.7951
|  759.9    |  0.2595   |  0.8932   |  0.7622
|  0.7737   |
|  30       |  0.9863   |  0.5894   |  0.7115
|  0.0881   |  1.91     |  2.26     |  0.09341
|  766.6    |  0.4705   |  0.6914   |  0.2507
|  0.9624   |
================================================================================
=============================================================================
```

[78]:
```
{'target': 0.9942003438041904,
 'params': {'colsample_bytree': 0.7423149626112193,
  'gamma': 0.4041491563174455,
  'learning_rate': 0.04372842773946696,
  'max_delta_step': 0.8068812912438583,
  'max_depth': 8.847495440604824,
  'min_child_weight': 0.6125910158781278,
  'n_estimators': 414.1133206068373,
  'reg_alpha': 0.27138414911981196,
```

```
 'reg_lambda': 0.5764643153330873,
 'scale_pos_weight': 0.37440413624132207,
 'subsample': 0.6482261815174488}}
```

0.6 根据三个优化结果新的参数建模，重新搭建 XGBoost 分类器，比较结果

网格搜索法

[52]:
```
clf_pca_g = XGBClassifier(max_depth=10,
                        learning_rate=0.1,
                        n_estimators=300,# 迭代次数
                        gamma=0,
                        min_child_weight=1,
                        subsample=0.8,
                        colsample_bytree=0.8,
                        colsample_bylevel=1,
                        reg_alpha=0,
                        reg_lambda=1,
                        use_label_encoder=False,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')
clf_pca_g.fit( X_train_pca, y_train)
```

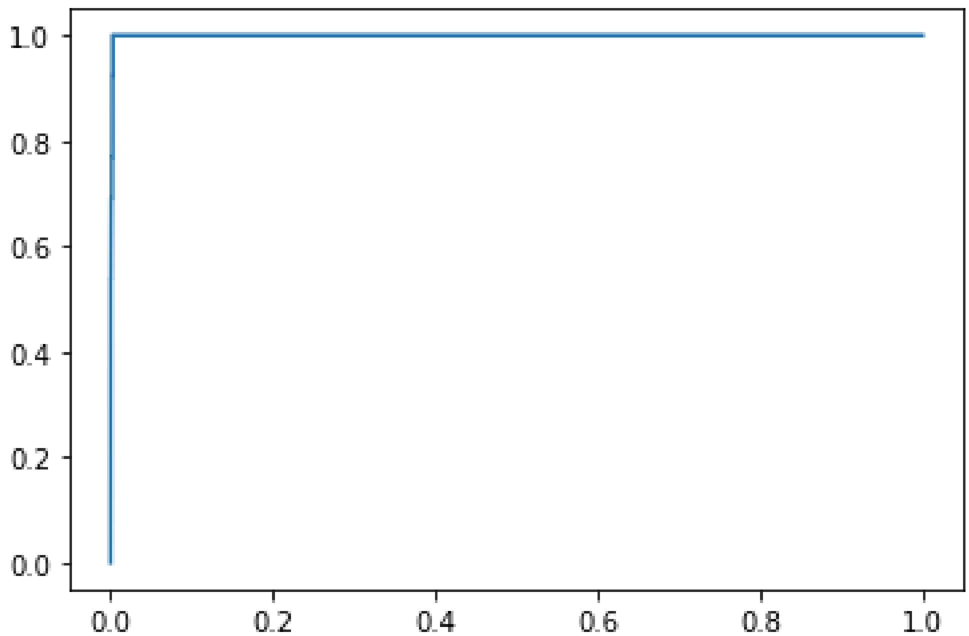
[52]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=300, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.8,
              tree_method='exact', use_label_encoder=False,
              validate_parameters=1, verbosity=0)
```

[53]:
```
# AUC 值:
y_pred_proba = clf_pca_g.predict_proba(X_test_pca)
from sklearn.metrics import roc_auc_score
```

```
score = roc_auc_score(y_test, y_pred_proba[:,1])
print(score)
```

0.9992318139637786

[54]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thres = roc_curve(y_test, y_pred_proba[:,1])
import matplotlib.pyplot as plt
plt.plot(fpr, tpr)
plt.show()
```



[55]:
```
# 通过如下代码求出模型的 AUC 值:
from sklearn.metrics import roc_auc_score
score = roc_auc_score(y_test, y_pred_proba[:,1])


score
```

[55]: 0.9992318139637786

[56]:
```python
# 将预测值和实际值进行对比:
ag = pd.DataFrame()  # 创建一个空 DataFrame
ag['预测值'] = list(y_pred)
ag['实际值'] = list(y_test)
ag.to_csv("ag.csv",encoding="utf_8_sig")
```

随机网格搜索法

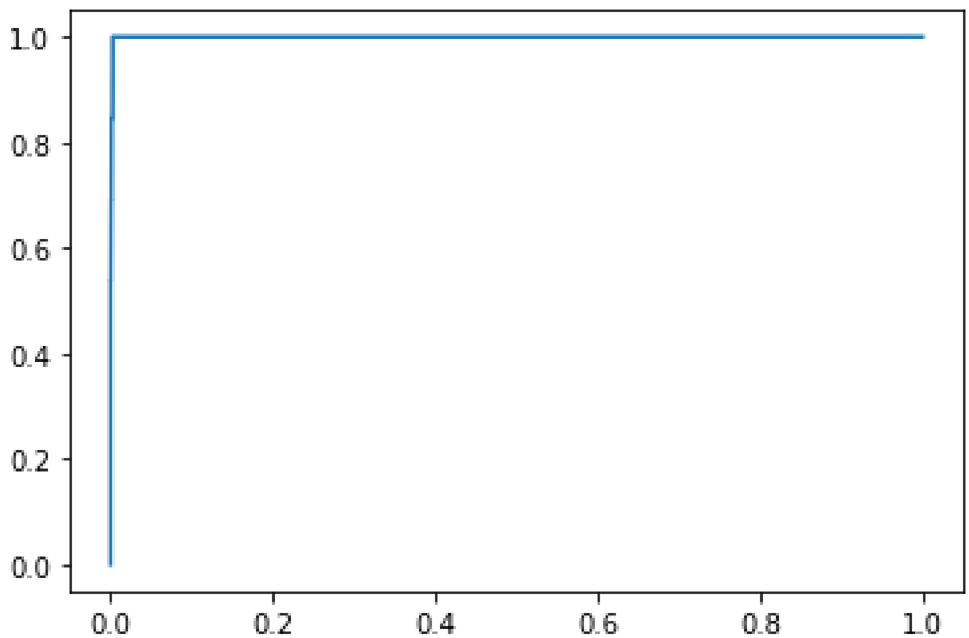
[57]:
```python
clf_pca_rg = XGBClassifier(max_depth=2,
                        learning_rate=0.1,
                        scale_pos_weight=0.4,
                        n_estimators=200,# 迭代次数
                        gamma=0,
                        min_child_weight=0,
                        subsample=0.7,
                        colsample_bytree=0.5,
                        colsample_bylevel=1,
                        reg_alpha=0.75,
                        reg_lambda=0.4,
                        use_label_encoder=False,
                        max_delta_step=1,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')
clf_pca_rg.fit( X_train_pca, y_train)
```

[57]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=1, max_depth=2,
              min_child_weight=0, missing=nan, monotone_constraints='()',
              n_estimators=200, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0.75, reg_lambda=0.4, scale_pos_weight=0.4,
              subsample=0.7, tree_method='exact', use_label_encoder=False,
              validate_parameters=1, verbosity=0)
```

[58]:
```
# AUC 值:
y_pred_proba = clf_pca_rg.predict_proba(X_test_pca)
from sklearn.metrics import roc_auc_score
score = roc_auc_score(y_test, y_pred_proba[:,1])
print(score)
```

```
0.9994223241007614
```

[59]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thres = roc_curve(y_test, y_pred_proba[:,1])
import matplotlib.pyplot as plt
plt.plot(fpr, tpr)
plt.show()
```



[60]:
```
# 通过如下代码求出模型的 AUC 值:
from sklearn.metrics import roc_auc_score
score = roc_auc_score(y_test, y_pred_proba[:,1])


score
```

[60]: 0.9994223241007614

[61]:
```
# 将预测值和实际值进行对比:
arg = pd.DataFrame()  # 创建一个空 DataFrame
arg['预测值'] = list(y_pred)
arg['实际值'] = list(y_test)
arg.to_csv("arg.csv",encoding="utf_8_sig")
```

贝叶斯优化

[90]:
```
clf_pca_b = XGBClassifier(max_depth=8,
                        learning_rate=0.04372842773946696,
                        scale_pos_weight=0.37440413624132207,
                        n_estimators=414,
                        gamma=0.4041491563174455,
                        min_child_weight=0,
                        subsample=0.6482261815174488,
                        colsample_bytree=0.7423149626112193,
                        colsample_bylevel=1,
                        reg_alpha=0.27138414911981196,
                        reg_lambda=0.5764643153330873,
                        use_label_encoder=False,
                        max_delta_step=0.8068812912438583,
                        verbosity=0,
                        objective='binary:logistic',
                        eval_metric='auc')
clf_pca_b.fit( X_train_pca, y_train)
```

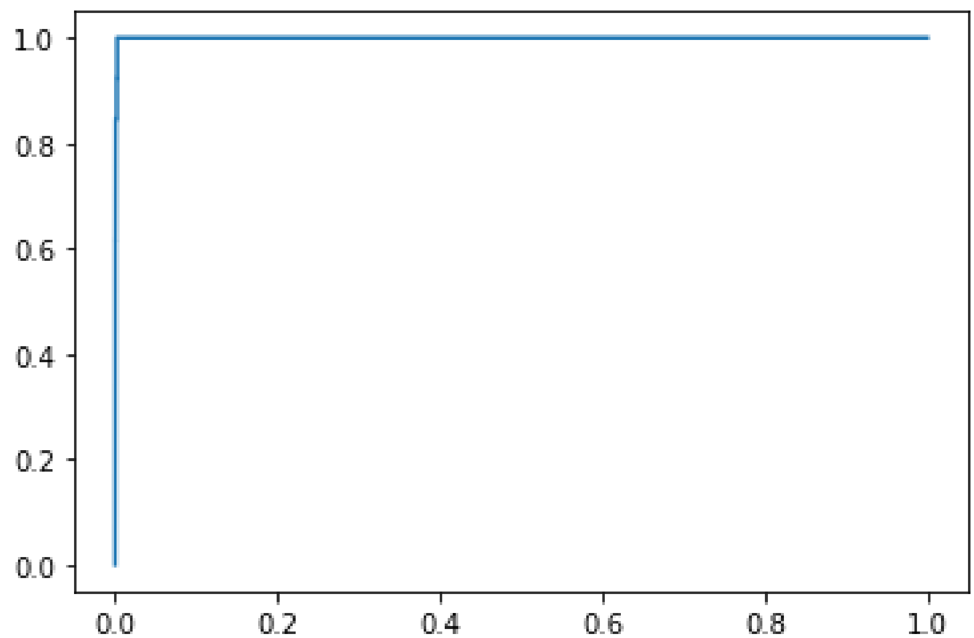
[90]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7423149626112193,
              gamma=0.4041491563174455, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.04372842773946696,
              max_delta_step=0.8068812912438583, max_depth=8,
              min_child_weight=0, missing=nan, monotone_constraints='()',
              n_estimators=414, n_jobs=16, num_parallel_tree=1, random_state=0,
              reg_alpha=0.27138414911981196, reg_lambda=0.5764643153330873,
              scale_pos_weight=0.37440413624132207,
              subsample=0.6482261815174488, tree_method='exact',
```

use_label_encoder=False, validate_parameters=1, verbosity=0)

[91]:
```
# AUC 值:
y_pred_proba = clf_pca_b.predict_proba(X_test_pca)
from sklearn.metrics import roc_auc_score
score = roc_auc_score(y_test, y_pred_proba[:,1])
print(score)
```

0.999514506425108

[92]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thres = roc_curve(y_test, y_pred_proba[:,1])
import matplotlib.pyplot as plt
plt.plot(fpr, tpr)
plt.show()
```



[93]:
```
# 通过如下代码求出模型的 AUC 值:
from sklearn.metrics import roc_auc_score
score = roc_auc_score(y_test, y_pred_proba[:,1])
```

```
score
```

[93]: 0.999514506425108

[94]:
```
# 将预测值和实际值进行对比:
ab = pd.DataFrame()  # 创建一个空 DataFrame
ab['预测值'] = list(y_pred)
ab['实际值'] = list(y_test)
ab.to_csv("ab.csv",encoding="utf_8_sig")
```

0.7 迁移学习城投债数据

[95]:
```
import pandas as pd
df1 = pd.read_csv('ctest2.1.csv')
cid = df1['id']
ctest=df1.drop(columns=['breach','id'])

ctest_new = StandardScaler().fit_transform(ctest)
pd.DataFrame(ctest)
#pd.DataFrame(X_train)
```

[95]:

	breachNewYear	region	bondBalance	regCap	comScale	industry	\
0	2020	27	30.80000	4.000000	2	5	
1	2020	27	30.80000	4.000000	2	5	
2	2020	21	15.80000	32.350400	2	5	
3	2020	21	15.80000	32.350400	2	5	
4	2020	9	2.00000	1.923669	1	5	
…	…	…	…	…	…	…	
13959	2020	5	12.00000	51.750362	3	2	
13960	2020	12	74.90000	40.693061	2	2	
13961	2020	5	12.00000	51.750362	3	2	
13962	2020	9	939.99999	459.000000	2	5	
13963	2020	9	939.99999	459.000000	2	5	

	guaCom	typeBond	bondTerm	bondIssuePrice	…	sbreachNewYearGPBR	\
0	1	1	7.0	100	…	2134.93	
1	1	1	7.0	100	…	2134.93	

2	1	1	7.0	100	…	3007.15
3	1	1	7.0	100	…	3007.15
4	1	1	7.0	100	…	6526.71
…	…	…	…	…	…	…
13959	0	1	10.0	100	…	4070.83
13960	0	1	10.0	100	…	1811.89
13961	0	1	10.0	100	…	4070.83
13962	0	1	10.0	100	…	6526.71
13963	0	1	10.0	100	…	6526.71

	sbreachNewYearGPBE	sSSFR	sbreachNewYearUrbanArea \
0	4847.6800	0.440402	7659.78
1	4847.6800	0.440402	7659.78
2	8034.4200	0.374283	5102.94
3	8034.4200	0.374283	5102.94
4	10739.7600	0.607715	23206.32
…	…	…	…
13959	10348.1712	0.393386	8609.50
13960	5850.9600	0.309674	5814.43
13961	10348.1712	0.393386	8609.50
13962	10739.7600	0.607715	23206.32
13963	10739.7600	0.607715	23206.32

	sbreachNewYearCityNumber	sbreachNewYearPCDI	sbreachNewYearPowerUsage \
0	1	28920.41	1160.27
1	1	28920.41	1160.27
2	13	27679.71	1864.32
3	13	27679.71	1864.32
4	16	31596.98	6218.72
…	…	…	…
13959	18	24703.15	2635.83
13960	14	23328.21	1907.33
13961	18	24703.15	2635.83
13962	16	31596.98	6218.72
13963	16	31596.98	6218.72

```
       sbreachNewYearHousePrice  sbreachNewYearUnemploymentRate  \
0                       8402.00                            2.60
1                       8402.00                            2.60
2                       6127.00                            2.73
3                       6127.00                            2.73
4                       8070.00                            3.29
…                          …                              …
13959                   7448.00                            3.31
13960                   6505.34                            2.60
13961                   7448.00                            3.31
13962                   8070.00                            3.29
13963                   8070.00                            3.29

       sbreachNewYearMinimumLivingSecurity
0                                  28.1000
1                                  28.1000
2                                  50.7000
3                                  50.7000
4                                  13.2837
…                                     …
13959                              76.8400
13960                              30.4900
13961                              76.8400
13962                              13.2837
13963                              13.2837

[13964 rows x 39 columns]
```

[97]:

```
ctest_new_pca = pca.transform(ctest_new)
```

[98]:

```
#import xgboost as xgb
cy_pred = clf_pca_b.predict(ctest_new_pca)
```

[99]:

```
cy_pred
```

[99]:

```
array([0, 0, 0, …, 0, 0, 0])
```

[100]:
```
df1['pred'] = cy_pred
df1
```

[100]:
```
             id  breach  breachNewYear  region  bondBalance      regCap  \
0     127353.SH       0           2020      27     30.80000    4.000000
1    1580326.IB       0           2020      27     30.80000    4.000000
2     139328.SH       0           2020      21     15.80000   32.350400
3    1580313.IB       0           2020      21     15.80000   32.350400
4     127172.SH       0           2020       9      2.00000    1.923669
…           …     …              …       …           …          …
13959  122724.SH       0           2020       5     12.00000   51.750362
13960 1280042.IB       0           2020      12     74.90000   40.693061
13961 1280044.IB       0           2020       5     12.00000   51.750362
13962  122742.SH       0           2020       9    939.99999  459.000000
13963 1280010.IB       0           2020       9    939.99999  459.000000

       comScale  industry  guaCom  typeBond  …  sbreachNewYearGPBE  \
0             2         5       1         1  …           4847.6800
1             2         5       1         1  …           4847.6800
2             2         5       1         1  …           8034.4200
3             2         5       1         1  …           8034.4200
4             1         5       1         1  …          10739.7600
…         …       …       …       …  …                 …
13959         3         2       0         1  …          10348.1712
13960         2         2       0         1  …           5850.9600
13961         3         2       0         1  …          10348.1712
13962         2         5       0         1  …          10739.7600
13963         2         5       0         1  …          10739.7600

          sSSFR  sbreachNewYearUrbanArea  sbreachNewYearCityNumber  \
0      0.440402                  7659.78                         1
1      0.440402                  7659.78                         1
2      0.374283                  5102.94                        13
3      0.374283                  5102.94                        13
4      0.607715                 23206.32                        16
…         …                     …                        …
13959  0.393386                  8609.50                        18
```

13960	0.309674	5814.43	14
13961	0.393386	8609.50	18
13962	0.607715	23206.32	16
13963	0.607715	23206.32	16

	sbreachNewYearPCDI	sbreachNewYearPowerUsage	sbreachNewYearHousePrice \
0	28920.41	1160.27	8402.00
1	28920.41	1160.27	8402.00
2	27679.71	1864.32	6127.00
3	27679.71	1864.32	6127.00
4	31596.98	6218.72	8070.00
…	…	…	…
13959	24703.15	2635.83	7448.00
13960	23328.21	1907.33	6505.34
13961	24703.15	2635.83	7448.00
13962	31596.98	6218.72	8070.00
13963	31596.98	6218.72	8070.00

	sbreachNewYearUnemploymentRate	sbreachNewYearMinimumLivingSecurity \
0	2.60	28.1000
1	2.60	28.1000
2	2.73	50.7000
3	2.73	50.7000
4	3.29	13.2837
…	…	…
13959	3.31	76.8400
13960	2.60	30.4900
13961	3.31	76.8400
13962	3.29	13.2837
13963	3.29	13.2837

	pred
0	0
1	0
2	0
3	0

```
4          0
…      …
13959      0
13960      0
13961      0
13962      0
13963      0

[13964 rows x 42 columns]
```

[101]:
```
df1['pred'] = cy_pred
df2=df1[['id','pred']]
df2.set_index('id', inplace=True)
df2.to_csv('result.csv',encoding="utf_8_sig")
```

[ ]: