1．实验目的

（1）理解Isolation Forest算法原理

（2）理解Local Outlier Factor算法原理

（3）掌握离群点检测的无监督方法。

（4）能够正确评价算法结果

2. 算法说明

（1）LOF算法（Local Outlier Factor），又叫局部异常因子，是一种基于距离的、无监督的离群点检测算法。

class sklearn.neighbors.**LocalOutlierFactor**(n\_neighbors=20, \*, algorithm='auto', leaf\_size=30, metric='minkowski', p=2, metric\_params=None, contamination='auto', novelty=False, n\_jobs=None)

参数contamination，相当于离群点在数据集中的占比，用作阈值

* if ‘auto’, the threshold is determined as in the original paper,
* if a float, the contamination should be in the range [0, 0.5].

代码示例：

# fit the model for outlier detection (default)

clf = LocalOutlierFactor(n\_neighbors=20, contamination=0.1)

y\_pred = clf.fit\_predict(X) #LOF没有predict方法

X\_scores = clf.negative\_outlier\_factor\_

（2）孤立森林算法（Isolation Forest ），是一种无监督的离群点检测算法，可发现全局离群点。

class sklearn.ensemble.**IsolationForest**(\*, n\_estimators=100, max\_samples='auto', contamination='auto', max\_features=1.0, bootstrap=False, n\_jobs=None, behaviour='deprecated', random\_state=None, verbose=0, warm\_start=False)

代码示例

# fit the model

clf = IsolationForest(max\_samples=100, random\_state=rng)

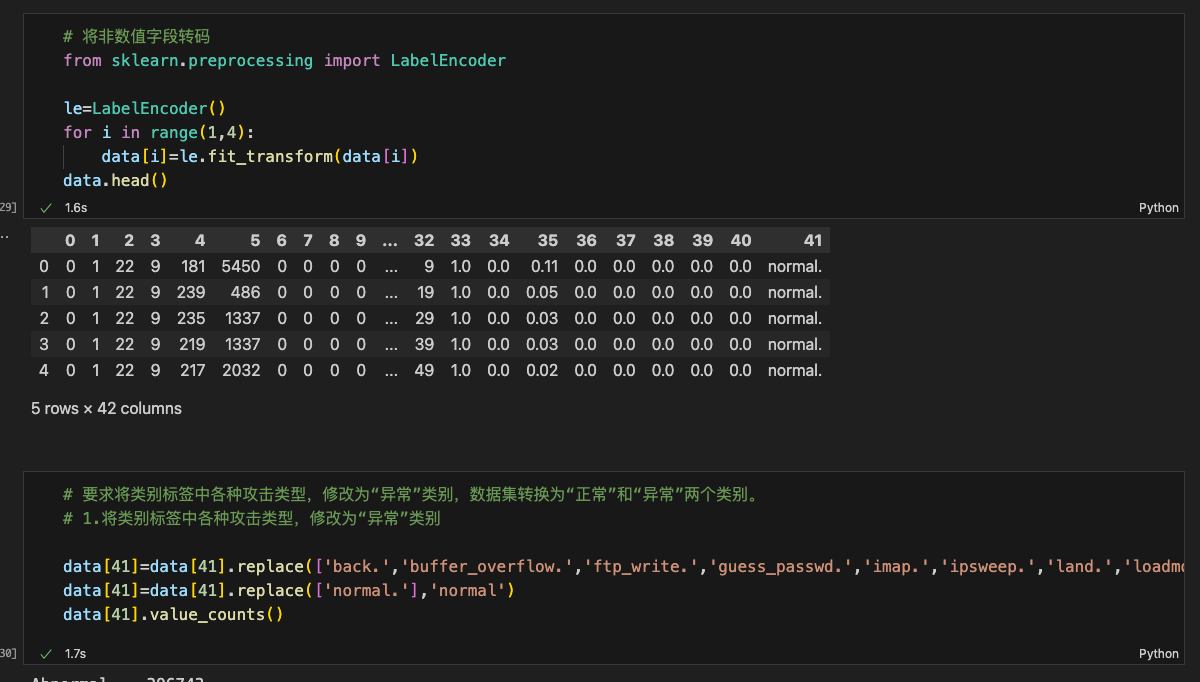
clf.fit(X)

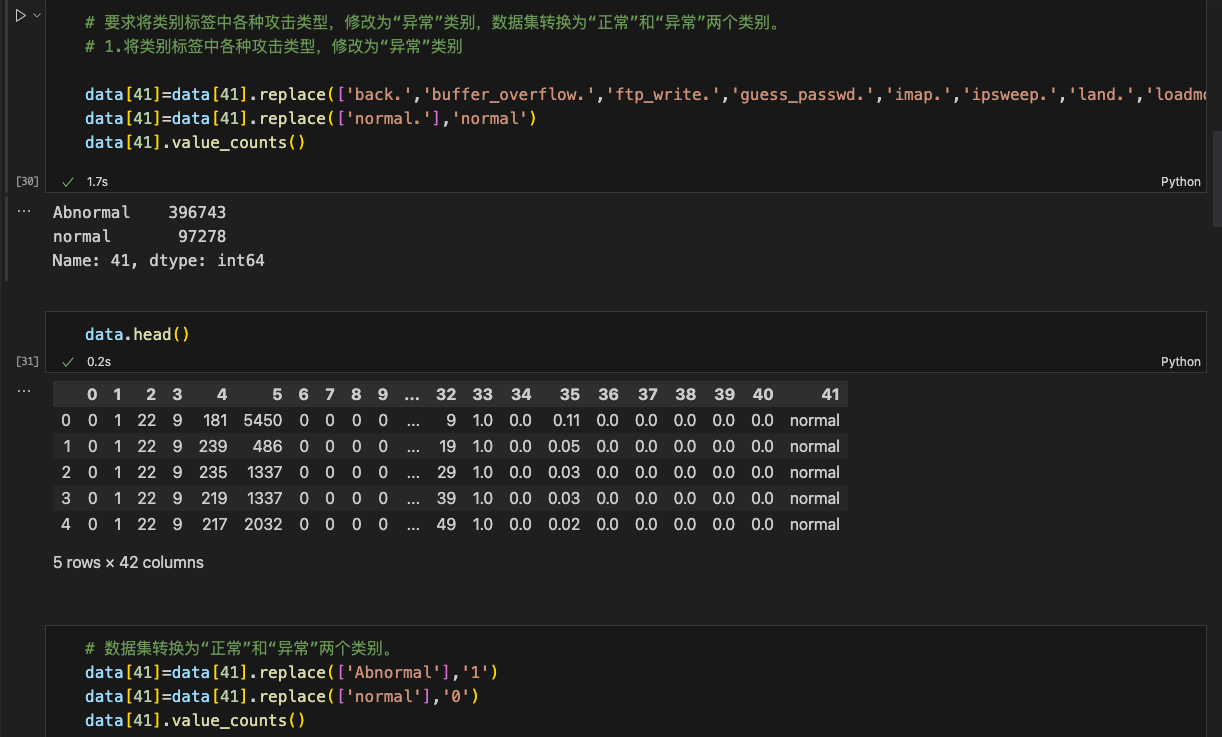
y\_pred = clf.predict(X)

3．实验要求和步骤

（1）读取KDD Cup网络入侵数据kddcup.data\_10\_percent，将非数值字段转码，并完成数据预处理。要求将类别标签中各种攻击类型，修改为“异常”类别，数据集转换为“正常”和“异常”两个类别。

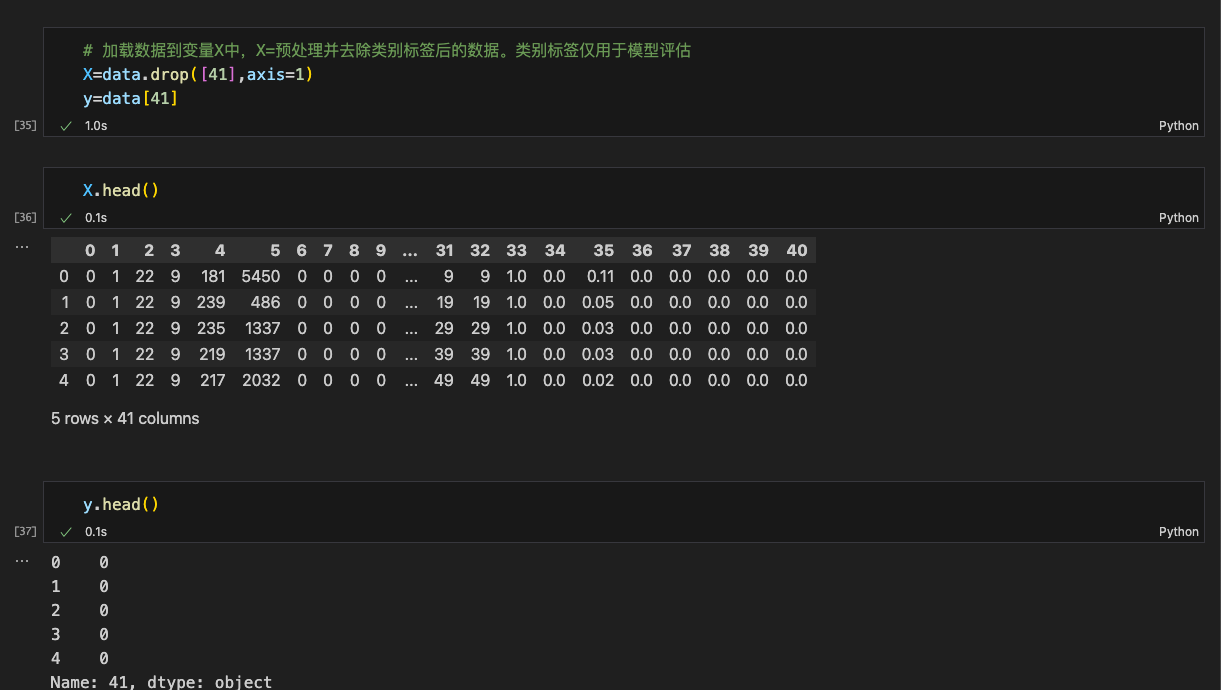






1. 加载数据到变量X中，X=预处理并去除类别标签后的数据。类别标签仅用于模型评估。

设置X为训练机y为测试集

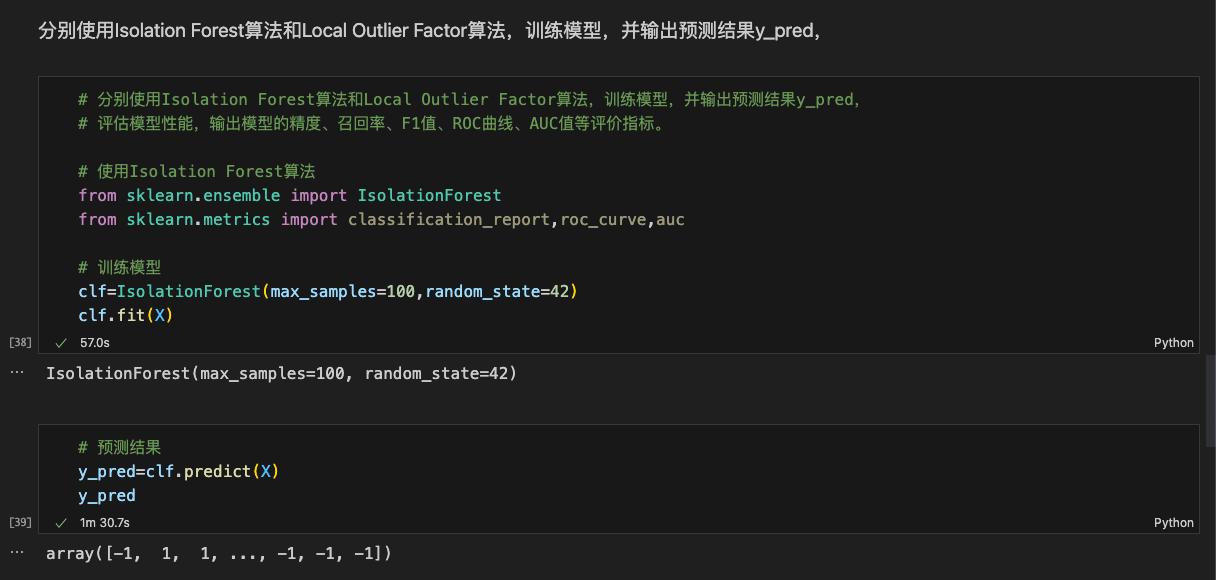


（3）分别使用Isolation Forest算法和Local Outlier Factor算法，训练模型，并输出预测结果y\_pred，根据已知的类标标签，评估模型的性能。

计算错误率的Python示例代码为：

n\_errors = (y\_pred != true\_label).sum()

采用isolationForest训练模型：



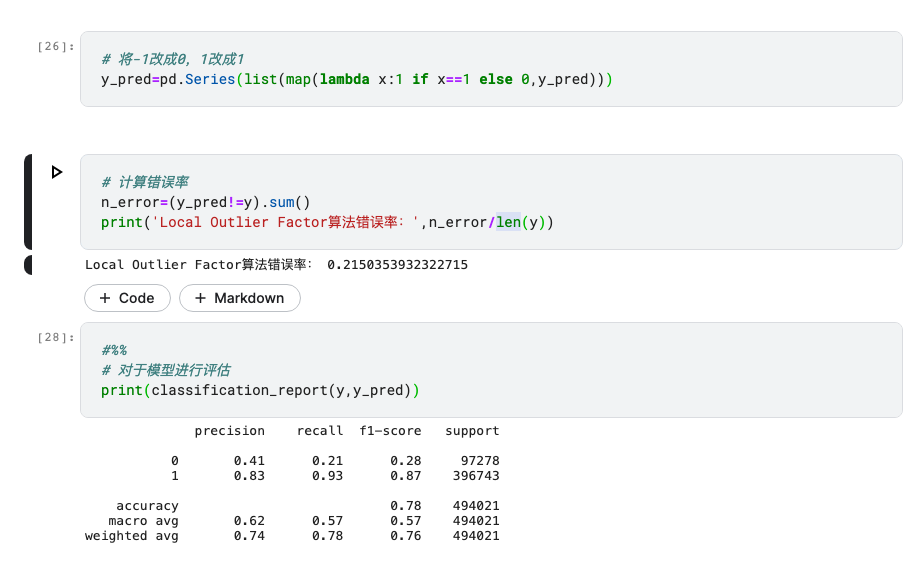
对于模型进行评估：



可以看到这种方法的正确率在81%

采用Local Outlier Factor 算法；

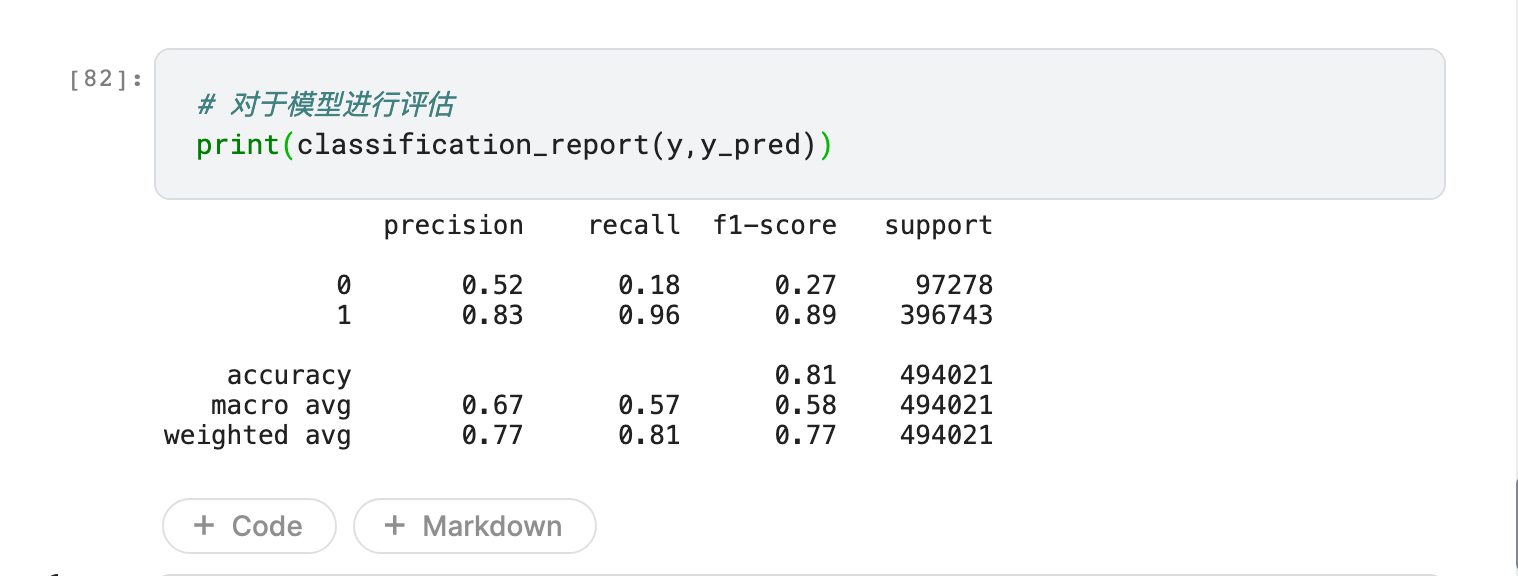




正确率和上一种方法相比略有降低为79%

1. 优化模型参数，观察预测准确率是否提高。

对于IsoForest进行参数调整将MaxSample修改为1000得到下面的情况：



可以看到对于模型检测的正确率略有上升

4．分析与讨论

（1）讨论还有哪些算法，适合分析预测网络入侵数据。

1. 支持向量机 (Support Vector Machines, SVM)：SVM是一种监督学习算法，可用于二分类和多分类问题。它在处理具有高维特征空间的数据时表现良好，可以用于检测异常和分类网络入侵数据。
2. 随机森林 (Random Forest)：随机森林是一种集成学习算法，由多个决策树组成。它可以用于特征选择和分类，适用于处理大规模的网络入侵数据集。
3. 朴素贝叶斯 (Naive Bayes)：朴素贝叶斯是一种基于贝叶斯定理的分类算法，常用于文本分类和垃圾邮件过滤。在网络入侵数据分析中，朴素贝叶斯算法可以用于检测异常和分类。
4. 逻辑回归 (Logistic Regression)：逻辑回归是一种广义线性模型，用于二分类问题。它可以用于网络入侵数据的分类和预测。
5. 神经网络 (Neural Networks)：神经网络是一种模拟人脑神经元连接的机器学习模型。对于复杂的网络入侵数据分析，深度学习的神经网络模型，如卷积神经网络，可能会有更好的表现。

5．附录

import pandas as pd

#%%

data=pd.read\_csv('/kaggle/input/kddcupdata/kddcup.data\_10\_percent.csv',header=None)

#%%

data.head()

#%%

# 将非数值字段转码

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for i in range(1,4):

data[i]=le.fit\_transform(data[i])

data.head()

#%%

# 要求将类别标签中各种攻击类型，修改为“异常”类别，数据集转换为“正常”和“异常”两个类别。

# 1.将类别标签中各种攻击类型，修改为“异常”类别

data[41]=data[41].replace(['back.','buffer\_overflow.','ftp\_write.','guess\_passwd.','imap.','ipsweep.','land.','loadmodule.','multihop.','neptune.','nmap.','perl.','phf.','pod.','portsweep.','rootkit.','satan.','smurf.','spy.','teardrop.','warezclient.','warezmaster.'],'Abnormal')

data[41]=data[41].replace(['normal.'],'normal')

data[41].value\_counts()

#%%

data.head()

#%%

# 数据集转换为“正常”和“异常”两个类别。

data[41]=data[41].replace(['Abnormal'],'1')

data[41]=data[41].replace(['normal'],'0')

data[41].value\_counts()

#%%

data.head()

#%% md

加载数据到变量X中，X=预处理并去除类别标签后的数据。类别标签仅用于模型评估

#%%

# 加载数据到变量X中，X=预处理并去除类别标签后的数据。类别标签仅用于模型评估

X=data.drop([41],axis=1)

y=data[41]

#%%

X.head()

#%%

y.head()

#%% md

分别使用Isolation Forest算法和Local Outlier Factor算法，训练模型，并输出预测结果y\_pred，

#%%

# 分别使用Isolation Forest算法和Local Outlier Factor算法，训练模型，并输出预测结果y\_pred，

# 评估模型性能，输出模型的精度、召回率、F1值、ROC曲线、AUC值等评价指标。

# 使用Isolation Forest算法

from sklearn.ensemble import IsolationForest

from sklearn.metrics import classification\_report,roc\_curve,auc

# 训练模型

clf=IsolationForest(max\_samples=100,random\_state=42)

clf.fit(X)

#%%

# 预测结果

y\_pred=clf.predict(X)

y\_pred

#%%

# 看y\_pred的数据分布

pd.Series(y\_pred).value\_counts()

#%%

y.value\_counts()

#%%

#把-1改成0 1改成1

y\_pred=pd.Series(list(map(lambda x:1 if x==0 else 1,y\_pred)))

#%%

# 解决y\_pred和y的索引不一致的问题

y\_pred.index=range(y\_pred.shape[0])

#%%

# 解决下面报错：Labels in y\_true and y\_pred should be of the same type. Got y\_true=['0' '1'] and y\_pred=[0 1]. Make sure that the predictions provided by the classifier coincides with the true labels.

y=y.astype('int')

#%%

# 计算错误率

import numpy as np

n\_error=(y\_pred!=y).sum()

print('Isolation Forest算法错误率：',n\_error/len(y))

#%%

# 评估模型性能

print(classification\_report(y,y\_pred))

#%%

# 使用 Local Outlier Factor算法

from sklearn.neighbors import LocalOutlierFactor

# 训练模型

clf=LocalOutlierFactor(n\_neighbors=20,contamination=0.1)

clf.fit(X)

#%%

# 预测结果

y\_pred=clf.fit\_predict(X)

y\_pred

#%%

# 看y\_pred的数据分布

pd.Series(y\_pred).value\_counts()

#%%

# 将-1改成0，1改成1

y\_pred=pd.Series(list(map(lambda x:1 if x==1 else 0,y\_pred)))

#%%

# 计算错误率

n\_error=(y\_pred!=y).sum()

print('Local Outlier Factor算法错误率：',n\_error/len(y))

#%%

#%%

# 对于模型进行评估

print(classification\_report(y,y\_pred))

#%% md

对于模型进行改进

#%%

#画出关系相关性热力图

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(20,20))

sns.heatmap(data.corr(),annot=True,fmt='.2f')

plt.show()

#%%

# Explained variance (how important each additional principal component is)

from sklearn.decomposition import PCA

pca=PCA().fit(X)

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

plt.xlabel('number of components')

plt.ylabel('cumulative explained variance')

plt.show()

#%%

# 主成分分析

# PCA

from sklearn.decomposition import PCA

pca=PCA(n\_components=2)

pca.fit(X)

X\_pca=pca.transform(X)

#%%

# 主成分分析结果可视化

plt.figure(figsize=(10,10))

plt.scatter(X\_pca[:,0],X\_pca[:,1],c=y)

plt.show()

#%%

print(X)

#%%

X.head()

#%%

# 对于IsoForest算法进行调参

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import IsolationForest

from sklearn.metrics import make\_scorer

from sklearn.metrics import roc\_auc\_score

# 创建自定义的评分函数

scorer = make\_scorer(roc\_auc\_score)

param\_grid = {'max\_samples': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]}

grid\_search = GridSearchCV(IsolationForest(random\_state=42), param\_grid, cv=5, scoring=scorer)

grid\_search.fit(X, y)

grid\_search.best\_params\_

#%%

# max\_samples=100时，错误率最低

# 训练模型

clf=IsolationForest(max\_samples=2000,random\_state=42)

clf.fit(X)

#%%

# 预测结果

y\_pred=clf.predict(X)

y\_pred

#%%

# 看y\_pred的数据分布

pd.Series(y\_pred).value\_counts()

#%%

# 将-1改成0，0改成-1

y\_pred=pd.Series(list(map(lambda x:1 if x==0 else 0,y\_pred)))

#%%

# 计算错误率

n\_error=(y\_pred!=y).sum()

print('Isolation Forest算法错误率：',n\_error/len(y))

#%%

# 对于模型进行评估

print(classification\_report(y,y\_pred))

#%% md

Iso Forest出了点问题，重新试验一下

#%%

clf = IsolationForest(max\_samples=100, random\_state=42)

clf.fit(X)

y\_pred = clf.predict(X)

#%%

# 将-1改成0，1改成1

y\_pred=pd.Series(list(map(lambda x:1 if x==1 else 0,y\_pred)))

#%%

# 计算错误率

n\_error=(y\_pred!=y).sum()

print('Isolation Forest算法错误率：',n\_error/len(y))

#%%

# 对于模型进行评估

print(classification\_report(y,y\_pred))

#%% md

调整参数

#%%

clf = IsolationForest(max\_samples=1000, random\_state=42)

clf.fit(X)

y\_pred = clf.predict(X)

#%%

# 将-1改成0，1改成1

y\_pred=pd.Series(list(map(lambda x:1 if x==1 else 0,y\_pred)))

#%%

# 计算错误率

n\_error=(y\_pred!=y).sum()

print('Isolation Forest算法错误率：',n\_error/len(y))

#%%

# 对于模型进行评估

print(classification\_report(y,y\_pred))

#%%

# 对于IsoForest算法进行调参

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import IsolationForest

from sklearn.metrics import make\_scorer

from sklearn.metrics import roc\_auc\_score

# 创建自定义的评分函数

scorer = make\_scorer(roc\_auc\_score)

param\_grid = {'max\_samples': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]}

grid\_search = GridSearchCV(IsolationForest(random\_state=42), param\_grid, cv=5, scoring=scorer)

grid\_search.fit(X, y)

grid\_search.best\_params\_