**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)

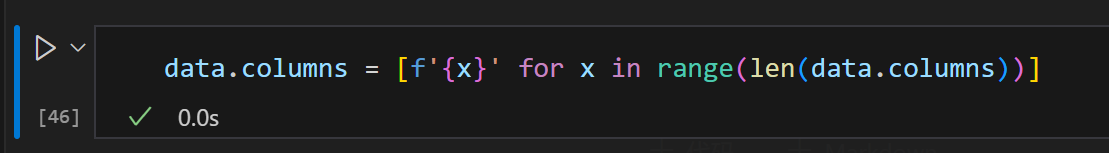
**3．实验要求和步骤**

（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。

a. 使用pandas读取数据：



b. 赋值列名

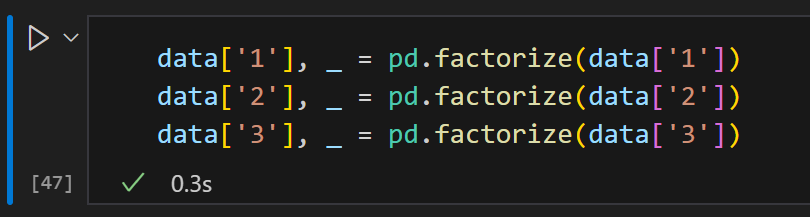


这样方便后面的索引

c. 非数值型转成数值型

对于data的‘1’，‘2’，‘3’列，进行数值编码：



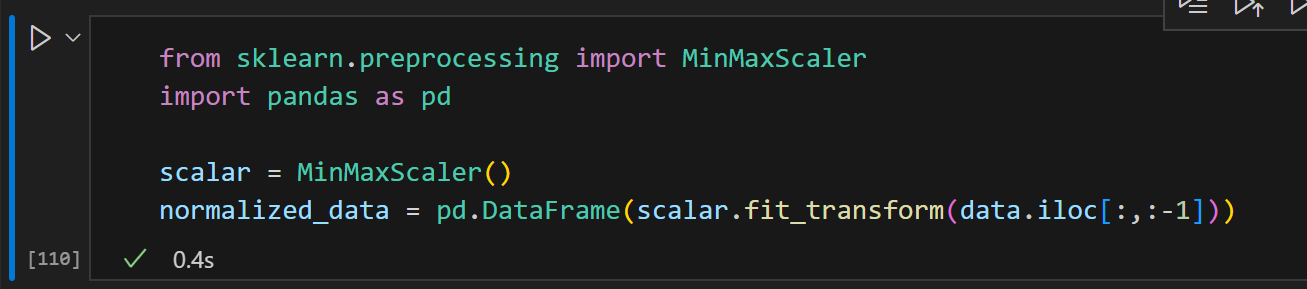


d. 异常值模拟

观察类别数量分布：

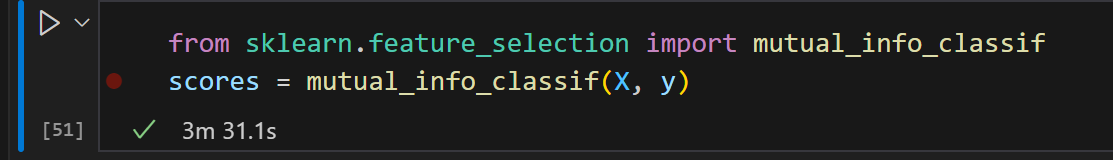
|  |  |
| --- | --- |
|  |  |

e. 进行数据归一化



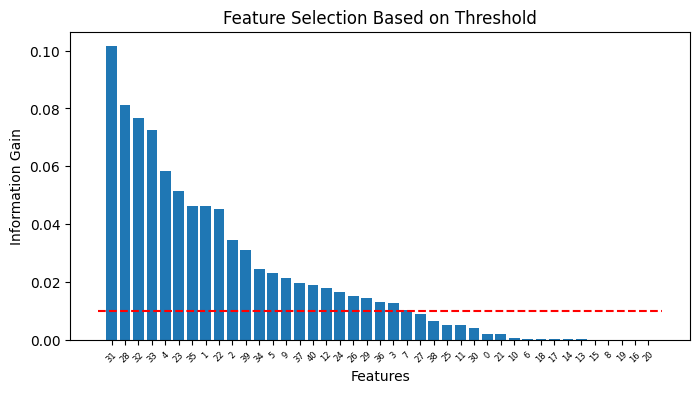
使用最小最大归一化。

f. 特征降维



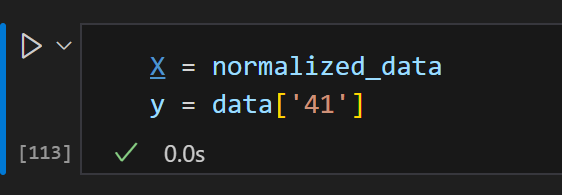
计算信息增益

g. 绘制信息增益图



根据阈值筛选出特征（阈值为0.01）。

（2）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。

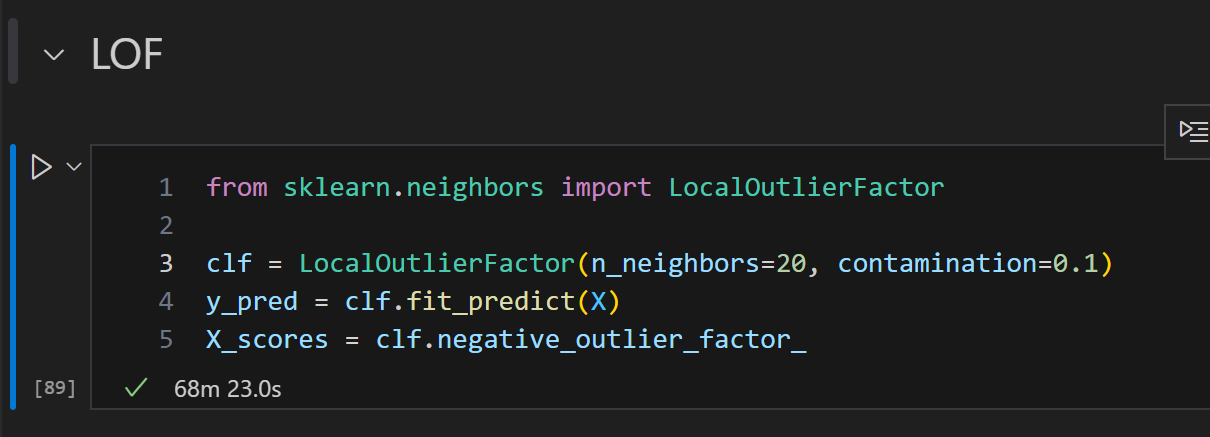


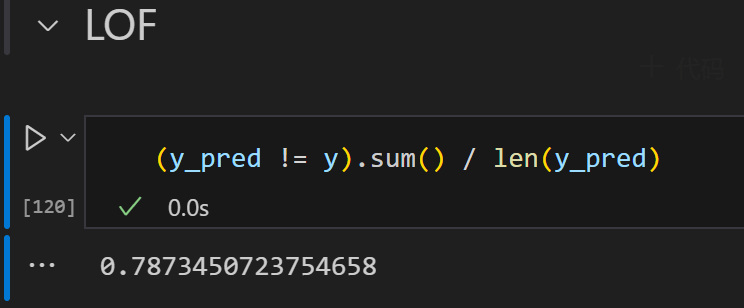
normalized\_data是经过最小最大归一化，以及信息增益降维的归一化数据的非标签部分。

y则是数据标签。

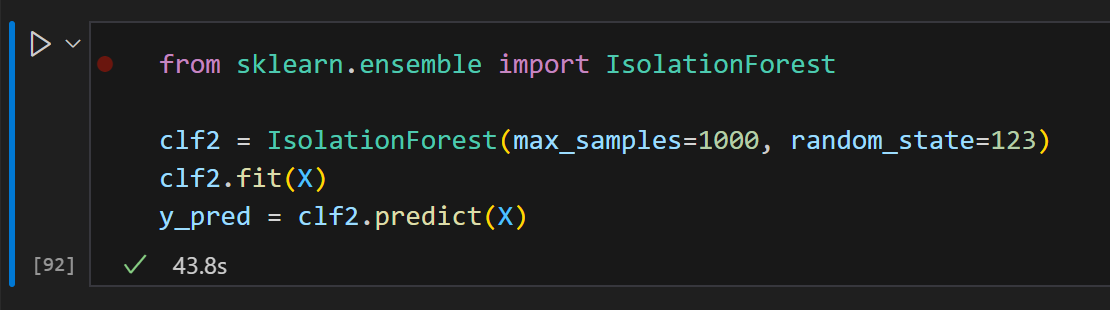
（3）分别使用Isolation Forest算法和Local Outlier Factor算法，训练模型，并输出预测结果y\_pred，根据已知的类标标签，评估模型的性能。

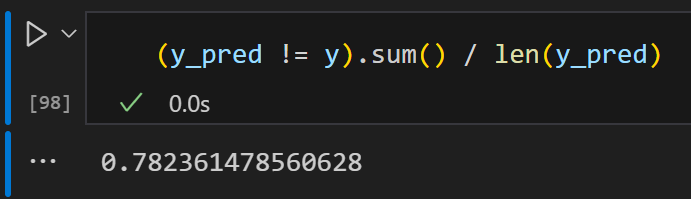
a. 使用局部异常因子进行异常值检测





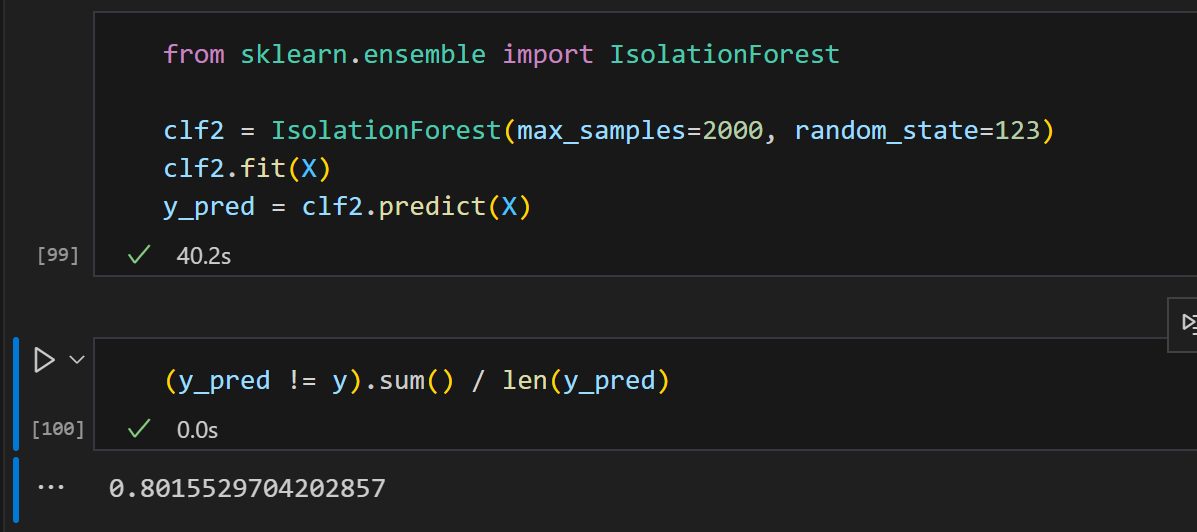
b. 使用孤立森林进行异常值检测



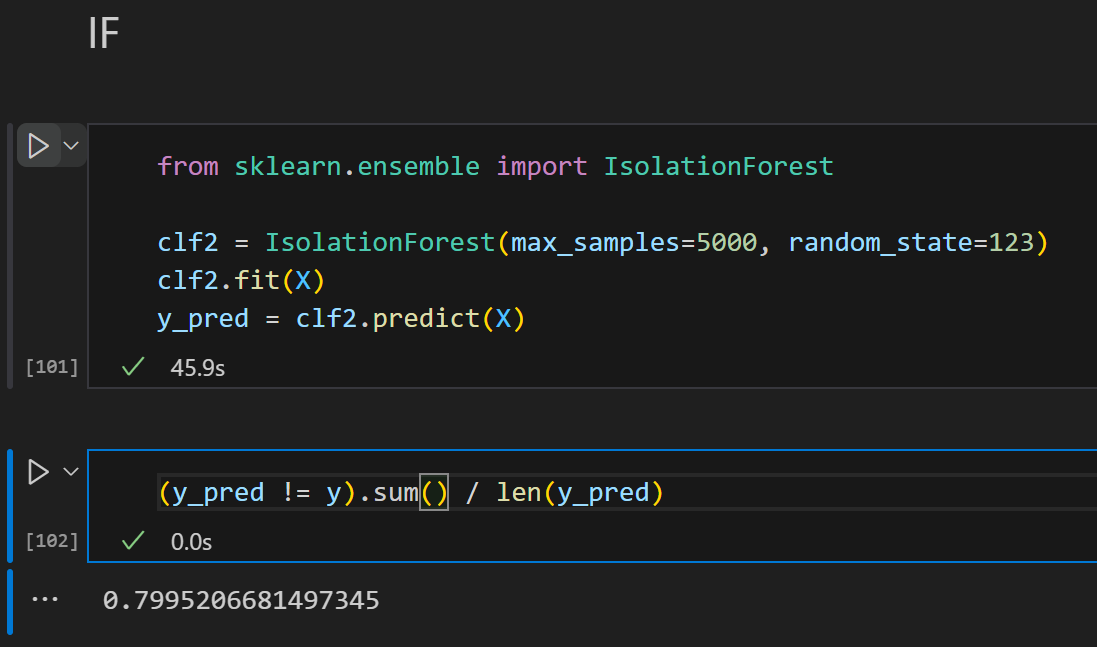


（4）优化模型参数，观察预测准确率是否提高。

调整max\_samples参数为2000，可以观察到准确率的提升：



再次增加max\_samples，观察到准确率降低：



**4．分析与讨论**

讨论还有哪些算法，适合分析预测网络入侵数据。

网络入侵检测是一种重要的安全任务，旨在检测和识别可能的网络攻击和入侵行为。以下是一些适合分析和预测网络入侵数据的常见算法：

1. K近邻（K-Nearest Neighbor, KNN）：KNN是一种基于实例的分类算法，可用于网络入侵检测中识别正常流量和异常流量。该算法通过计算每个样本与其最近邻的距离和标签，将数据分为多个类别，并对每个类别进行分类。KNN算法可以自适应地调整分类器的决策边界，适应不同的数据分布，具有较高的分类精度和鲁棒性。
2. 聚类算法（Clustering）：聚类算法可以将数据集中的样本分为多个类别，可用于网络入侵检测中识别异常流量和攻击模式。该算法通过计算样本之间的相似度或距离，将数据划分为多个簇，并对每个簇进行分类和预测。常见的聚类算法包括K-Means、DBSCAN和层次聚类等。
3. 神经进化算法（Neuro Evolution Algorithm, NEA）：NEA是一种基于进化算法和神经网络的智能优化算法，可用于网络入侵检测中识别异常流量和攻击模式。该算法通过不断优化神经网络的结构和参数，从而提高分类准确性和泛化能力。NEA算法可以自适应地调整神经网络的拓扑结构和权重，适应不同的数据分布和噪声干扰，具有较高的分类精度和鲁棒性。
4. 概率图模型（Probabilistic Graphical Model, PGM）：PGM是一种基于概率论和图论的模型，可用于网络入侵检测中识别异常流量和攻击模式。该算法通过建立概率图模型，描述流量数据的联合概率分布，并利用贝叶斯推断的方法进行分类和预测。常见的PGM包括朴素贝叶斯、高斯混合模型和隐马尔可夫模型等。
5. 集成学习算法（Ensemble Learning）：集成学习是一种基于多个分类器的组合方法，可用于网络入侵检测中识别异常流量和攻击模式。该算法通过组合多个分类器的预测结果，提高分类准确性和泛化能力。常见的集成学习算法包括Bagging、Boosting和Stacking等。

**5．附录**

import pandas as pd

data = pd.DataFrame(pd.read\_csv('kddcup.data\_10\_percent\_corrected', index\_col=None, header=None))

data.columns = [f'{x}' for x in range(len(data.columns))]

data['1'], \_ = pd.factorize(data['1'])

data['2'], \_ = pd.factorize(data['2'])

data['3'], \_ = pd.factorize(data['3'])

data['41'].value\_counts()

# data['41'][data['41'] == 'smurf.'] = 0

# data['41'][data['41'] == 'neptune.'] = 0

data['41'][data['41'] == 'normal.'] = 1

#  再将其他类别全部标记成异常

data['41'][data['41'] != 1] = -1

data['41'].value\_counts()

from sklearn.preprocessing import MinMaxScaler

import pandas as pd

scalar = MinMaxScaler()

normalized\_data = pd.DataFrame(scalar.fit\_transform(data.iloc[:,:-1]))

normalized\_data.describe()

data['41'] = data['41'].astype(int)

X = normalized\_data

y = data['41']

from sklearn.feature\_selection import mutual\_info\_classif

scores = mutual\_info\_classif(X, y)

col = data.columns[:-1]

inf\_gain = {}

for i in range(len(col)):

    inf\_gain[col[i]] = scores[i]

ifg\_rank = sorted(inf\_gain.values(), reverse=True)

ifg\_label = sorted(inf\_gain.keys(), key = lambda k: inf\_gain[k], reverse=True)

import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))

plt.bar(ifg\_label, ifg\_rank)

plt.xticks(rotation=45, fontsize=6)

plt.xlabel('Features')

plt.ylabel('Information Gain')

plt.title('Feature Selection Based on Threshold')

plt.plot([-1,41], [0.01, 0.01],color='red', linestyle='dashed')

pass

import numpy as np

ifg\_rank = np.array(ifg\_rank)

ifg\_label = np.array(ifg\_label)

features = ifg\_label[ifg\_rank >= 0.01]

features

import pickle

with open('data.pickle','wb') as f:

    '''

    分别存储归一化的数据部分（特征）、标签以及筛选出的特征列

    '''

    pickle.dump((normalized\_data, y, features), f)

from sklearn.neighbors import LocalOutlierFactor

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

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

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

(y\_pred != y).sum() / len(y\_pred)

from sklearn.ensemble import IsolationForest

clf2 = IsolationForest(max\_samples=500, random\_state=123)

clf2.fit(X)

y\_pred = clf2.predict(X)

(y\_pred != y).sum() / len(y\_pred)

from sklearn.ensemble import IsolationForest

clf2 = IsolationForest(max\_samples=2000, random\_state=123)

clf2.fit(X)

y\_pred = clf2.predict(X)

(y\_pred != y).sum() / len(y\_pred)

from sklearn.ensemble import IsolationForest

clf2 = IsolationForest(max\_samples=5000, random\_state=123)

clf2.fit(X)

y\_pred = clf2.predict(X)

(y\_pred != y).sum() / len(y\_pred)