1．实验目的

1. 理解离群点检测技术
2. 理解Isolation Forest算法原理

（3）理解Local Outlier Factor算法原理

（4）掌握离群点检测的无监督方法。

（5）能够正确评价算法结果

2．任务描述

2.1 离群点检测技术

离群点检测是一种重要的数据挖掘技术，其原理是寻找数据集中与大多数数据点显著不同或异常的数据点。这些异常数据可能代表着不寻常的事件、错误数据或者潜在的异常行为。离群点检测在金融、工业、网络安全等领域都有广泛应用。例如，在金融领域，离群点检测可用于发现欺诈交易；在工业制造中，可以检测设备故障；而在网络安全中，可用于发现网络攻击。

本实验任务是对KDD Cup数据集进行离群点检测，该数据集包含了网络通信数据，其中包括正常通信和多种类型的入侵行为。实验目标是利用机器学习和离群点检测算法，准确地识别和分类入侵行为，从而保护网络安全。

2.2 数据集

在网络安全中，KDD Cup数据集是一个广泛使用的网络入侵检测领域中的数据集。该数据集由加拿大麦吉尔大学研究人员开发，用于评估网络入侵检测系统的性能和准确性。数据集包含来自1998年美国空军研究实验室网络的网络流量信息，其中包括正常流量和攻击流量。攻击流量分为四类: DoS(拒绝服务攻击)、Probing (探测攻击)、U2R(用户到根)、R2L(远程到本地)攻击。该数据集包含4900000个连接记录，对于入侵检测研究来说是一个较大的数据集。这个数据集为入侵检测研究领域提供了基础，并且在不同的领域中被广泛使用，如机器学习、数据挖掘和大数据等。因此在本实验中，我们将采用KDD Cup作为基准数据集。

KDDCup99入侵检测实验数据的标识类型：

表1 标识类型

|  |  |  |
| --- | --- | --- |
| **标识类型** | **含义** | **具体分类标识** |
| Normal | 正常记录 | normal |
| DOS | 拒绝服务攻击 | back, land, neptune, pod, smurf, teardrop |
| Probing | 监视和其他探测活动 | ipsweep, nmap, portsweep,satan |
| R2L | 来自远程机器的非法访问 | ftp\_write, guess\_passwd, imap, multihop, phf, spy, warezclient, warezmaster |
| U2R | 普通用户对本地超级用户特权的非法访问 | buffer\_overflow, loadmodule, perl, rootkit |

单个TCP连接的基本特征：

表2 单个TCP连接的基本特征

|  |  |  |
| --- | --- | --- |
| **特征名** | **描述** | **类型** |
| Duration | 连接时间长度（单位：秒） | 连续型 |
| Protocol\_type | 协议类型，如tcp,udp | 离散型 |
| Service | 在目标机的网络服务，如http,telnet等 | 离散型 |
| src\_bytes | 源地址到目标地址的数据流量 | 连续型 |
| dst\_bytes | 目标地址到源地址的数据流量 | 连续型 |
| flag | 连接状态（正常或错误） | 离散型 |
| land | 1表示数据连接源地址和目标地址为同一主机或端口；0表示其他 | 离散型 |
| wrong\_fragment | 错误碎片的数目 | 连续型 |
| urgent | 紧迫数据包的个数 | 连续型 |

单次连接中包含的内容特征：

表3 单次连接的内容特征

|  |  |  |
| --- | --- | --- |
| **特征名** | **描述** | **类型** |
| hot | 访问系统敏感文件和目录的次数 | 连续型 |
| mum\_failed\_logins | 尝试登录失败的次数 | 连续型 |
| loggged\_in | 1表示成功登录，0表示其他 | 离散型 |
| num\_compromised | 受到威胁状态的次数 | 连续型 |
| root\_shell | 1表示超级用户的shell外壳，0表示其他 | 离散型 |
| su\_attempted | 1表示命令执行尝试，0表示其他 | 离散型 |
| num\_root | root权限访问的次数 | 连续型 |
| num\_file\_creations | 文件创作的操作次数 | 连续型 |
| num\_shells | shell提示符合的个数 | 连续型 |
| num\_access\_files | 访问控制文件的次数 | 连续型 |
| num\_outbound\_cmds | 一次ftp会话中传递命令的次数 | 连续型 |
| is\_hot\_login | 1表示属于热点清单的登录，0表示其他 | 离散型 |
| is\_guest\_login | 1表示guest用户登录，0表示其他用户名登录 | 离散型 |

3．相关知识

LOF算法（Local Outlier Factor），又叫局部异常因子，是一种用于离群点检测的算法，它通过度量每个数据点与其邻域数据点之间的密度来识别潜在的离群点。LOF算法的核心思想是：离群点的局部密度比周围的数据点要低。

具体来说，LOF算法通过以下步骤进行离群点检测：

（1）计算每个数据点与其最近的K个邻居之间的距离，可以选择不同的距离度量方法，如欧式距离、曼哈顿距离等。

（2）计算每个数据点的局部可达密度（Local Reachability Density，LRD），即该数据点与其邻居之间的平均距离的倒数。LRD反映了该数据点的局部密度。

（3）计算每个数据点的局部离群因子（Local Outlier Factor，LOF），即该数据点的局部密度与其邻居的局部密度的比值。LOF越大，说明该数据点的局部密度比周围的邻居要低，更有可能是离群点。

‘sklearn.neighbors.LocalOutlierFactor’是‘Scikit-learn’中的局部离群点因子（LOF）算法的实现类。在该算法中，参数‘contamination’用于设置离群点在数据集中的占比作为阈值来判断数据点是否为离群点。

具体地说，‘contamination’参数有以下两种设置方式：

1. 如果‘contamination’参数设置为字符串值‘auto’，则阈值会根据原始论文中的方法来确定。在原始论文中，LOF算法会根据给定的K值（即‘n\_neighbors’参数）来计算离群点得分，然后将得分按照从大到小进行排序，通过选择前‘n\_samples \* contamination’个得分作为阈值来判断离群点。

2. 如果‘contamination’参数设置为一个浮点数，则需要指定一个具体的数值，取值范围在[0, 0.5]之间。这个值代表了数据集中预期的离群点的比例。例如，如果设置‘contamination=0.1’，则表示预计数据集中有10%的数据点是离群点。

总的来说，‘contamination’参数用于设定一个阈值，超过这个阈值的数据点会被判断为离群点。用户可以根据自己的数据集和应用需求来选择合适的阈值，以达到准确和有效的离群点检测效果。

代码示例：

from sklearn.neighbors import LocalOutlierFactor

# 创建LocalOutlierFactor实例

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

# 拟合模型并预测离群点

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

# 获取每个数据点的离群点得分

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

4．实验要求与步骤

（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。

首先，读取数据文件，该数据为kddcup.data\_10\_percent的一个子集，如图1所示。

data = pd.read\_csv('文件夹路径\kddcup.data\_10\_percent\_corrected', header=None)

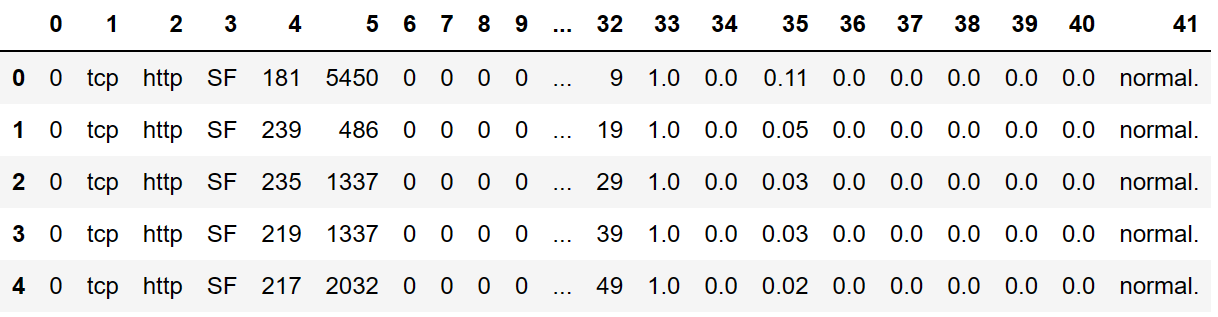


图1 数据示意

接着，将非数值字段转码，初步对第2、3、4列的数据逐行转化。初步转化结果如图2所示

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

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

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

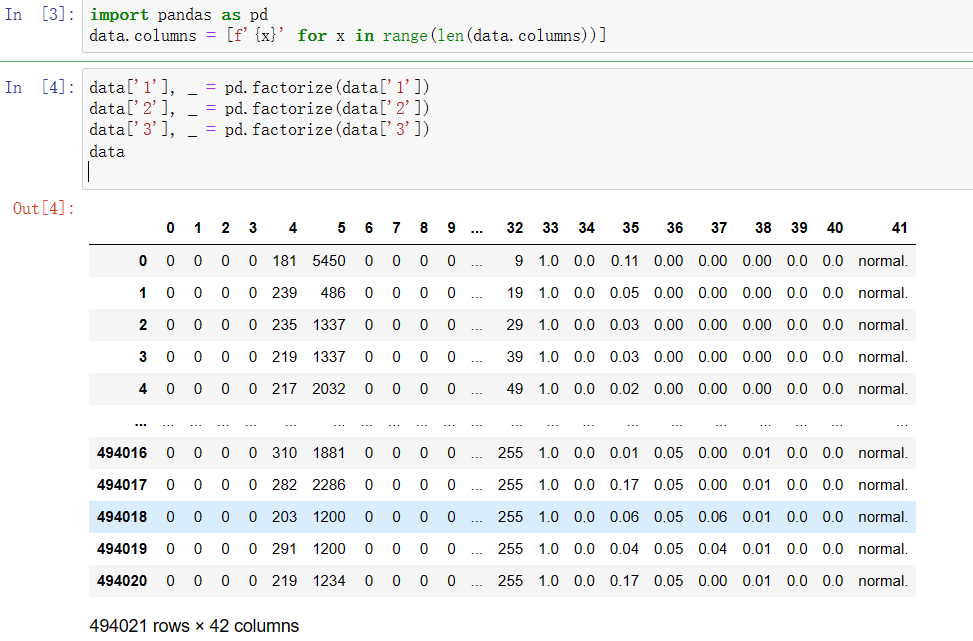


图2 初步转化结果

接下来，根据数据集的介绍中的标签分类，我们将42列中的值为normal的数据划分为正常类别，并对其进行非数值字段转码，将‘normal’转为数值1，并将其他值转为数值-1。完整的转化结果如图3所示。

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

data.loc[data[41] != 1, 41] = -1

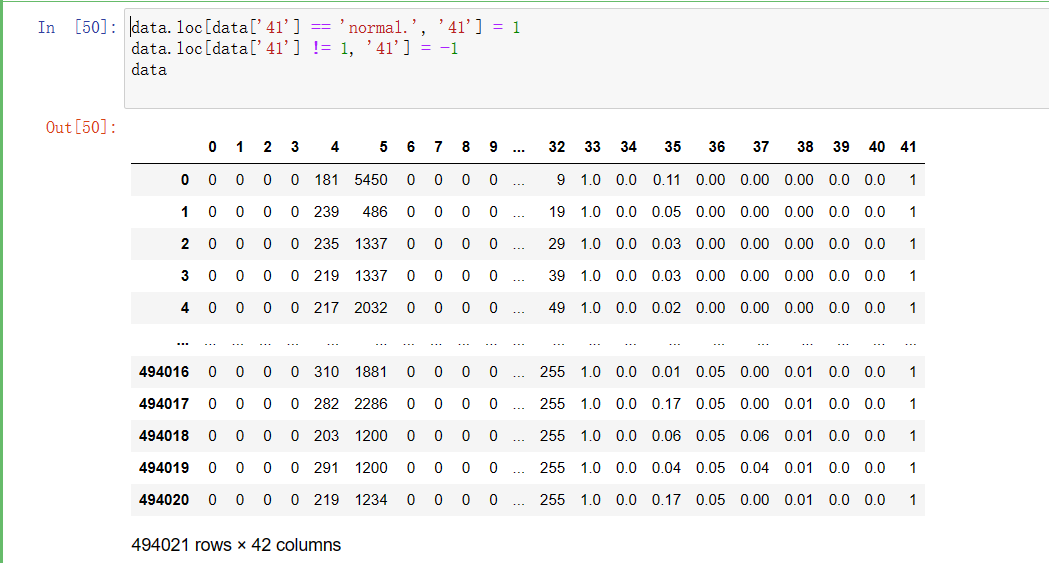


图3 完整的转化结果

（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。

首先，使用drop()方法将训练数据和标签划分开。

ylabel = data[41]

xdata = data.drop(41,axis=1)

接下来，我们进行了数据标准化，这里使用最大最小归一化。

from sklearn.preprocessing import MinMaxScaler

# 创建标准化对象

scaler = MinMaxScaler()

# 对训练数据集进行标准化

xdata = pd.DataFrame(scaler.fit\_transform(xdata))

xdata.describe()

除了数据标准化以外，我们还可以进行特征筛选，它指的是从原始特征集合中选择最具有代表性和重要性的特征子集，以提高模型性能、降低计算复杂度、减少过拟合风险和增加模型可解释性。  
 我们使用基于随机森林模型的特征选择，并使用条形图展示特征的重要性排序。然后，根据设定的特征数量选用特征。  
 import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

# 创建一个RandomForestClassifier模型

model = RandomForestClassifier(n\_estimators=100, max\_depth=10, random\_state=2021, n\_jobs=10)

# 在数据上进行训练

model.fit(xdata, ylabel)

# 获取特征重要性分数

feature\_importance = model.feature\_importances\_

# 按降序对特征进行重要性排序

sorted\_indices = np.argsort(feature\_importance)[::-1]

# 获取前20个重要特征

top\_20\_features = sorted\_indices[:20]

features = xdata.columns[top\_20\_features]

# 绘制特征重要性条形图

plt.figure(figsize=(10, 6))

plt.barh(range(20), feature\_importance[top\_20\_features], align='center')

plt.yticks(range(20), [str(i) for i in range(20)]) # 使用数字标签

# plt.yticks(range(20), features)

plt.xlabel('Feature importance')

plt.ylabel('Feature')

plt.title('The most important feature')

plt.show()

（3）使用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。

首先，通过创建Local Outlier Factor对象，并设置参数n\_neighbors和contamination（异常值比例）。然后，通过fit\_predict方法对特征数据xdata进行离群点检测，得到预测结果y\_pred，其中1表示正常样本，-1表示离群点。接着，获取离群点因子值X\_scores，用于衡量样本的离群程度。最后，计算离群点检测的准确率，通过统计预测结果中与实际标签不相符的样本数量，并除以总样本数得到。

# 使用加载的数据进行LOF算法

print("xdata shape:", xdata.shape)

print("xdata columns:", xdata.columns)

print("features:", features)

features = features.astype(int)

# 使用筛选后的特征列

xdata = xdata[features]

clf = LocalOutlierFactor(n\_neighbors=100, contamination=0.1)

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

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

# 计算错误分类率

error\_rate = (y\_pred != ylabel).sum() / len(y\_pred)

print("分类错误率：", error\_rate)

（4）优化模型参数，观察预测准确率是否提高。

.优化Local Outlier Factor算法中的参数

clf = LocalOutlierFactor(n\_neighbors=100, contamination=0.1)

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

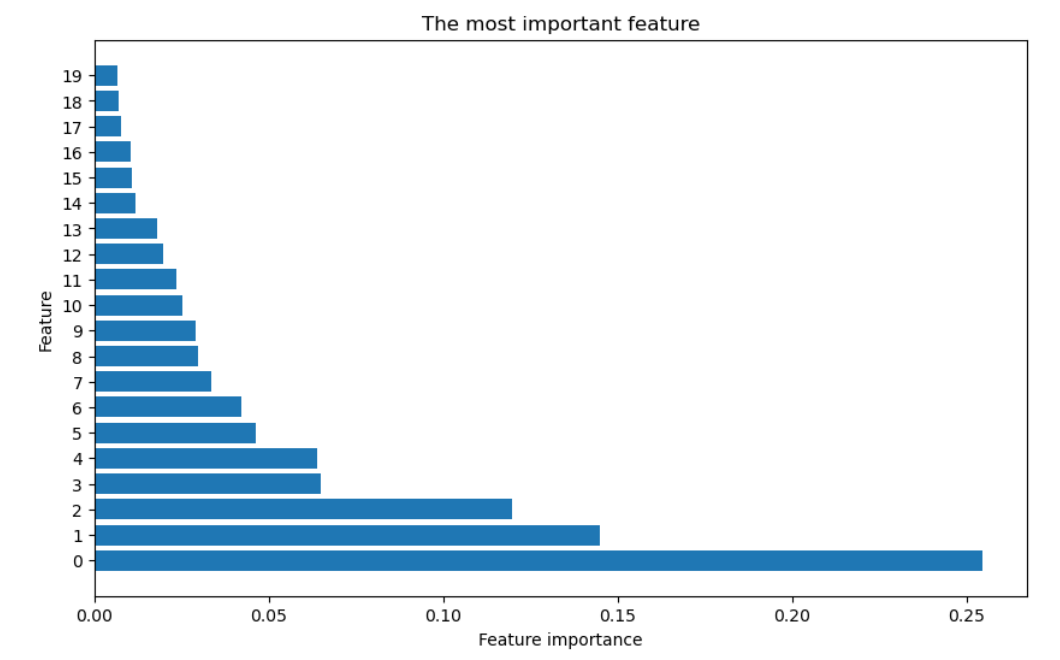
X\_scores = clf.negative\_outlier\_factor\_

(y\_pred != ylabel).sum() / len(y\_pred)  
 例如修改n\_neighbors。请尝试调试Local Outlier Factor算法的参数，以便降低错误率。

5．测试说明

1. 数据标准化

网络协议做了编码，正常流量和异常流量分为两类，因为异常流量可以归为一类

1. 绘制特征的重要性排序图
2. 预测结果

clf = LocalOutlierFactor(n\_neighbors=100, contamination=0.1)

分类错误率： 0.8009396361693126

clf = LocalOutlierFactor(n\_neighbors=200, contamination=0.1)

分类错误率： 0.8061276747344749

clf = LocalOutlierFactor(n\_neighbors=100, contamination=0.2)

分类错误率： 0.7838553421818101

clf = LocalOutlierFactor(n\_neighbors=50, contamination=0.3)

分类错误率： 0.7735237975713583

clf = LocalOutlierFactor(n\_neighbors=100, contamination=0.25)

分类错误率： 0.7751613797794021

6．分析与讨论

1. 离群点检测在实际应用中有哪些重要的应用场景？请举例说明在网络安全、金融风控等领域中离群点检测的应用。

入侵检测系统（IDS）

恶意行为检测

1. 离群点检测算法在面对不平衡数据集时可能会遇到什么问题？你可以提出一些解决不平衡数据集问题的方法，并讨论它们的效果。

偏斜的类别分布： 不平衡数据集通常指其中某个类别的样本数量远远超过另一个类别。这样的分布可能导致模型更容易学习主导类别而忽略罕见类别，从而降低对离群点的检测性能。

模型评估的问题： 由于罕见类别的样本较少，简单地使用准确度作为评估指标可能会导致误导。在不平衡数据集中，模型可能表现出较高的准确度，但在罕见类别上的性能较差。

1. 实验中我们使用了孤立森林和局部离群因子 (Local Outlier Factor, LOF) 算法。这两种方法的原理分别是什么？请比较它们在不同数据集上的表现，并讨论孤立森林在高维数据和LOF在大规模数据集上的优势。

对数据分布的适应性：

孤立森林： 适用于各种类型的数据分布，尤其在高维空间中表现较好。

LOF： 更适用于局部密度发生变化的数据集，对于全局密度差异较大的情况可能表现得相对较差。

计算复杂度：

孤立森林： 具有较低的计算复杂度，适用于大规模数据集。

LOF： 在大规模数据集上可能存在计算复杂度较高的问题，尤其是对于每个样本都需要计算邻域的情况。

对离群点的敏感性：

孤立森林： 对于离群点的检测较为稳健，因为离群点在树中的高度较低。

LOF： 对于局部密度较低的样本更容易被标记为离群点，对于全局密度较低的样本可能表现得较差。

1. 在实际应用中，我们可能会面对异常样本的漂移问题。你认为如何应对数据漂移对离群点检测算法的影响？

采用在线学习的方法，允许模型随着时间的推移进行动态更新。当有新的样本进入系统时，可以使用这些样本来更新模型，以适应数据分布的变化。