异常值检测anomaly detection

10215501412 彭一珅

数据

UNSW-NB15数据集：

- 攻击类型（7类）：Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance
- 数据维度：196
- 数据划分：训练集train_data.csv，测试集test_data.csv
- 列名：label表示是否为异常值（0或1），attack_cat表示攻击类型（字符串形式）

异常检测

任务：在UNSW-NB15数据集上进行异常检测，使用经典的无监督异常检测模型iForest、LOF、DeepSVDD。

要求：

- 使用AUROC和AUPRC指标评估；
- 可以调包；
- 可以直接使用DeepOD仓库：https://github.com/xuhongzuo/DeepOD
- 在实验报告中列出表格对比这几类异常检测方法的性能和时间。

实验过程

数据预处理

首先读取训练集和测试集数据：

```
train_data = pd.read_csv('NB15/train_data.csv')
test_data = pd.read_csv('NB15/test_data.csv')
```

由于本次实验使用的是无监督异常检测模型，因此去掉标签列和attack_cat：

```
X_train = train_data.drop(columns=['label'])
X_train = X_train.drop(columns=['attack_cat'])
y_train = train_data['label']
X_test = test_data.drop(columns=['label'])
X_test = X_test.drop(columns=['attack_cat'])
y_test = test_data['label']
```

IForest

隔离森林（Isolation Forest，简称 iForest）是一种无监督学习算法，用于识别异常值。

其基本原理可以概括为一句话：异常数据由于**数量较少**且与正常数据**差异较大**，因此在被隔离时需要较少的步骤。

有两个假设：

1. 异常的值是非常少的（如果异常值很多，可能被识别为正常的）
2. 异常值与其他值的差异较大（这点也可以引出主要是全局上都为异常的异常，局部小异常可能发现不了，因为差异并不大）

训练过程：

1. **子采样**：首先从整个数据集中随机抽取一定数量的样本来为构建树做准备。这些抽样的子集大小通常远小于原始数据集的大小，这样可以限制树的大小，并且减少计算复杂度。
2. **构建孤立树** (iTrees)： 对于每个子采样集，算法构建一棵孤立树。构建孤立树的过程是递归的。在每个节点，算法随机选择一个特征，并在该特征的最大值和最小值之间随机选择一个分割值。然后，数据根据这个分割值将样本分到左子树或右子树（这里其实就是简单的将样本中特征小于这个分割点的样本分到左边，其次分到右边）。这个过程的结束条件：树达到限定的高度， 节点中的样本数量到一定的数目，或者所有样本的所选特征值都是同一个值。
3. **森林构建**：重复1-2构建完特定数量的孤立树，集合为孤立森林。

检测过程：

1. 路径长度（h(x)）: 指样本通过该孤立树构建阶段的特征选择方式，从树的根节点到达该样本被孤立的节点（被孤立就是意味着这个样本最终到达的树的叶子节点）所需要的边数。
2. 平均路径长度E(h(x)): 该样本在森林中所有树的路径长度的平均值。
3. 树的平均路径长度：

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

   H(i)是调和数，可以近似为ln(i) + 0.5772156649，其中n为样本个数，对于给定的数据集大小n，平均路径长度的期望是一个常数，该公式提供了一个标准化的基准，用于将路径长度标准化
4. 异常分数：对于每个样本x，其异常分数s(x, n)的计算公式如下:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

   其中，E(h(x))是数据点x在所有树中路径长度的平均值，n 是训练数据的样本量，c(n)是树的平均路径长度，用于标准化。
   - E(h(x)) 约等于 c(n)， 样本点的路径长度和平均路径长度没啥差别，看不出是否有异常
   - E(h(x)) 越靠近0， score越接近1， 说明样本很容易被孤立，更可能是异常的
   - E(h(x)) 的极端情况就是为样本的n-1， 这时候score很小，不容易被孤立，也看不出是否为异常

路径长度计算: 对于新的测试样本计算它们在每棵孤立树中的路径长度, 并算平均路径长度。

计算异常分数: 利用上述计算，孤立森林会计算每个数据点的异常分数。

判定异常: 根据计算出的异常分数，可以设置一个阈值来判定哪些数据点是异常的。

相关代码通过调用pyod包中的IForest模型实现：

```
# iForest
iForest.fit(X_train)
y_pred = iForest.predict(X_test)
auc_roc = roc_auc_score(y_test, y_pred)
auc_pr = average_precision_score(y_test, y_pred)
results['iForest'] = {'AUROC': auc_roc, 'AUPRC': auc_pr}
```

LOF

LOF (Local Outliers Factor，局部异常因子) 算法 是一种非监督异常检测算法，它是通过计算给定数据点相对于其邻域的局部密度偏差而实现异常检测。

核心思路： LOF算法是通过比较每个点p和邻域点的密度来判断该点是否为异常：点p的密度越低，越有可能是异常点。而点的密度是通过点之间的距离来计算的，点之间距离越远，密度越低；距离越近，密度越高。也就是说，LOF算法中点的密度是通过点的k邻域计算得到的，而不是通过全局计算得到，这里的"k邻域"也就是该算法中"局部"的概念。

相关代码仍然是通过pyod中自带的模型实现：

```
lof.fit(X_train)
y_pred = lof.predict(X_test)
auc_roc = roc_auc_score(y_test, y_pred)
auc_pr = average_precision_score(y_test, y_pred)
results['lof'] = {'AUROC': auc_roc, 'AUPRC': auc_pr}
```

DeepSVDD

Deep SVDD 的核心思想是将数据映射到低维空间，然后在这个空间中找到一个最佳的超球体（Hyper-sphere），使所有正常数据点尽可能地集中在超球体内。任何远离这个球心的数据点则被视为异常。这种方法在处理大规模高维度数据时特别有效，因为它能够自动学习数据的主要特征，并只对这些特征进行异常检测。

模型架构：Deep SVDD 使用神经网络作为非线性映射器，将原始数据转化为低维表示。这允许模型学习数据的复杂结构，并且可以适应各种类型的数据。

损失函数：为了找到最优的超球体，Deep SVDD 使用了特殊的设计的损失函数，即"SVDD Loss"，它最小化所有数据点到球心的距离之和。此外，该算法还包括一个正则化项，以防止过拟合。

训练过程：训练过程中，大部分数据被视为正常，小部分被标记为异常。网络的目标是在保证正常数据点位于超球体内的同时，尽量减小异常点到球心的距离。

deepsvdd的代码是依赖于torch的，所以需要先安装torch环境。此外，需要将数据集处理为torch.tensor格式传入模型。

```
# DeepSVDD
Deepsvdd.fit(X_train_tensor, y=None)
y_pred = Deepsvdd.decision_function(X_test_tensor)

# print(y_pred,y_test)
# print(len(y_pred),len(y_test))
auc_roc = roc_auc_score(y_test, y_pred)
auc_pr = average_precision_score(y_test, y_pred)
results['DeepSVDD'] = {'AUROC': auc_roc, 'AUPRC': auc_pr}
```

结果分析

AUROC和AUPRC指标都是用于评估**二分类**模型性能的指标。

AUC-ROC曲线下面积（Area Under the Receiver Operating Characteristic Curve）（AUC-ROC）（AUROC）： 适用于二分类问题，ROC曲线是以真正例率（True Positive Rate，召回率）为纵轴Y、假正例率（False Positive Rate）为横轴X的曲线，AUC-ROC是ROC曲线下的面积。AUC-ROC通常用于评估模型在不同阈值下的性能。

AUC-PR（Area Under the Precision versus Recall Curve）（AUPR）：PR 曲线则反映了精确率 Precision（预测为真阳样本占所有预测为阳性样本的比例）和召回率Recall（预测为真阳样本占所有实际真样本的比例）的关系，其曲线下面积被认为相比于 AUROC 更能反映一个模型对真样本的富集能力。

	iForest	**lof**	**DeepSVDD**
AUROC	0.562695	0.508903	0.774321
AUPRC	0.211161	0.179957	0.392863
运行时间	21.7s	131.8s	192.4s

1. **iForest**：
    - AUROC和AUPRC指标相对较低，可能是由于数据特征维度很高，虽然iforest不会因为特征的多少而降低算法的效率，但也正因为每次只随机用其中一个特征作为分割的特征，如果特征维度很高，就会有很多特征没有用到。
    - 孤立森林假设数据集中的异常值“少而不同”，因此适用于数据集中异常值比例较低的情况，本次实验使用的数据集异常值比例为5%，相对较适合。
    - 运行时间较短。孤立森林算法特别适合处理大数据集，它具有线性的时间复杂度，并且由于使用了子采样，使得在计算上更加高效。
2. **LOF**：
    - AUROC和AUPRC指标都相对较低，这是因为lof模型最明显的缺点就是检测的数据必须有明显的密度差异，而该数据集可能没有体现密度区别。
    - 运行时间较长，因为lof模型计算比较复杂。
3. **DeepSVDD**：
    - AUROC和AUPRC指标都相对较高，表明模型在不同阈值下的性能较好，能够更好地区分正常样本和异常样本，以及富集真正例。
    - 运行时间较长，但相比于其性能表现更好。