# 利用孤立森林实现异常流量检测

## 关于孤立森林

### 孤立森林的基本思想

孤立森林采用传统的二分法思想，具体来说，**在孤立森林中，采用二叉树对数据进行分裂**，并且样本选取、特征选取、分裂点选取都使用随机化的方式。

从字面简单理解，**孤立森林对异常点的判定遵循孤立原则，即异常点不属于某些样本点的簇，或者距离聚集的样本点的距离较大。**

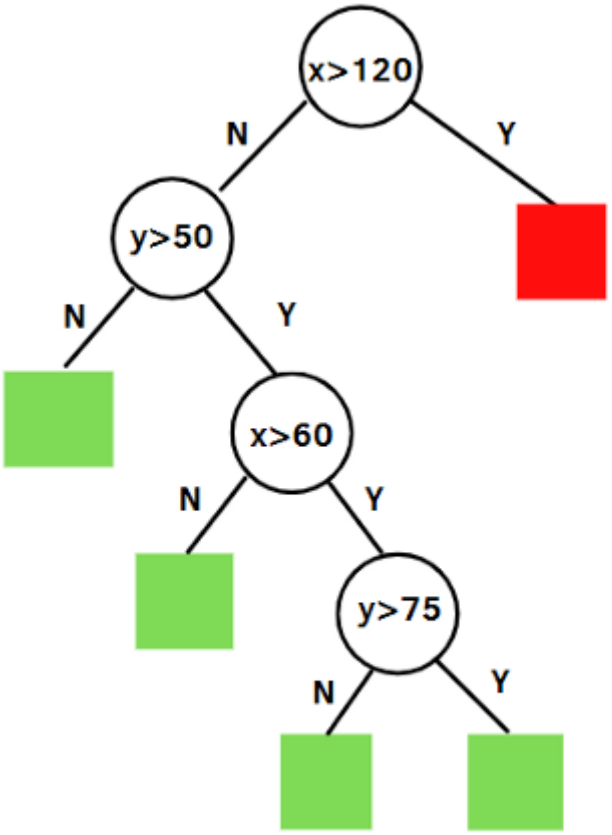
在对样本点的判定的过程中，有以下步骤：

1. 选择样本最大值和最小值作为边界
2. 随机选择一个X值，按照>X 和 <=X 将数据分组
3. 接着在分组之后的数据再选择一个值进行分组
4. 重复以上步骤，直到数据不可再分。

通过上述的步骤，由于比较聚集的点是也就是正常的样本点的密度较大，因此需要分组的次数较多，而孤立的点，也就是异常的样本点则需要较少的分割次数。

这样放在一个二叉树上的表现就是，正常点由于分组的次数较多，对应的子树也就较多，其叶子节点距离根节点的距离较远。
而异常节点由于比较孤立，则划分的子树更少，距离根节点更远。

如下图所示：




标红的样本点被视为异常的样本点，其对应的子树较少（图中只有一个），且叶子节点距离根节点更近。

如果**从统计意义**上来说，**相对聚集的点需要分割的次数较多，比较孤立的点需要的分割次数少，孤立森林就是利用分割的次数来度量一个点是聚集的（正常）还是孤立的（异常）**


# 孤立森林的详解

## 两个假设

首先，满足孤立森林的样本需要满足两个假设：

1. **异常样本不能占比太多**
2. **异常样本与正常样本的差异较大**

在算法上，异常样本会更快的落入叶子节点（也就是划分的次数更少），或者距离根节点更近。

举个例子，对应月平均工资分布在(5000,10000)的样本有100个，而在(10000,15000)有4个，仅针对这些数据进行划分，则(5000,10000)的样本点落入叶子节点的速度远远会小于(10000,15000)的样本，并且这些较大的样本与较小的样本有足够大的差异被认为是异常点。

## 算法详解

孤立森林算法生成树的过程是**从原始数据中，有放回或者不放回的抽取部分样本，选取部分特征，构建一颗二叉树(iTree即Isolation Tree)，利用集成学习的思想，多次抽取样本和特征，完成多棵iTree的构建。**

在paper中提到的伪代码如下：

**Algorithm 1** : $iForest(X, t, \psi)$

**Inputs**: $X$ - input data, $t$ - number of trees, $\psi$ - sub-sampling size

**Output**: a set of $t$ *iTrees*

1: **Initialize** $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: **for** $i = 1$ to $t$ **do**
4: $X' \leftarrow sample(X, \psi)$
5: $Forest \leftarrow Forest \cup iTree(X', 0, l)$
6: **end for**
7: **return** $Forest$

**Algorithm 2** : $iTree(X, e, l)$

**Inputs**: $X$ - input data, $e$ - current tree height, $l$ - height limit

**Output**: an iTree

**if** $e \geq l$ or $|X| \leq 1$ **then**
  return $exNode\{Size \leftarrow |X|\}$
**else**
  let $Q$ be a list of attributes in $X$
  randomly select an attribute $q \in Q$
  randomly select a split point $p$ from $max$ and $min$ values of attribute $q$ in $X$
  $X_l \leftarrow filter(X, q < p)$
  $X_r \leftarrow filter(X, q \geq p)$
  return $inNode\{Left \leftarrow iTree(X_l, e+1, l),$
    $Right \leftarrow iTree(X_r, e+1, l),$
    $SplitAtt \leftarrow q,$
    $SplitValue \leftarrow p\}$
**end if**

其中第一张图是生成孤立树的过程，第二张图是生成一颗子树的过程。

### 生成孤立森林

首先解释一下生成孤立树的方法，伪代码如下：

```
Initialize Forest
set height limit l = ceiling(log2 ψ)
for i = 1 to t do
    X' ← sample(X, ψ)
    Forest ← Forest ∪ iTree(X', 0, l)
end for
return Forest
```

从图中也可以看出，生成孤立树的输入有三个：

- X 输入的数据集
- t - 生成子树数量
- ψ - 子采样大小

其中子采样大小即每次分割抽取多少个子数据。

算法的输出是一系列树的集合，即一组t棵孤立树。

上述伪代码的具体过程：

1. 初始化孤立树
2. 设定孤立树最大高度
3. 从1到t的范围循环
4. 每次循环进行ψ此采样，并且将采样范围和最大高度输入给生成子孤立树的函数
5. 循环结束输出一组孤立树

### 生成孤立树

上述图二的伪代码是生成一颗孤立树的算法， 将会被生成孤立森林的算法调用。

伪代码如下：

```
1: if e ≥ l or |X| ≤ 1 then
2:     return exNode{Size ← |X|}
3: else
4:     let Q be a list of attributes in X
5:     randomly select an attribute q ∈ Q
6:     randomly select a split point p from max and min values of attribute
q in X
7:     X_l ← filter(X, q < p)
8:     X_r ← filter(X, q ≥ p)
9:     return inNode{Left ← iTree(X_l, e + 1, l),
10:                  Right ← iTree(X_r, e + 1, l),
11:                  SplitAtt ← q,
12:                  SplitValue ← p}
13: end if
```

接受输入：

- x - 采样数据
- e - 当前高度
- l - 高度限制

算法流程：

1. 如果 `e` 大于或等于 `l` 或 `|X|` 小于等于 `1`，则返回叶节点，包含当前数据的大小。
2. 否则进行构建子树：
    1. 令Q为X的属性列表
    2. 从Q中随机选择一个属性q
    3. 从q属性的最大值和最小值之间随机选择一个分割点p

4. 依据点p将数据X分割为X_l即小于p 的部分和X_r即大于p的部分
5. 将X_l和X_r作为样本数据递归调用iTree生成子树
6. 返回一个内部节点，包含左右子树、分割属性和分割值。

其中比较重要的几个参数：

- 树的最大深度=ceiling(log(subsimpleSize))，paper里说自动指定，sklearn也是在代码中写死:max_depth = int(np.ceil(np.log2(max(max_samples, 2))))这个值接近树的平均深度，我们只关注那些小于平均深度的异常值，所以无需让树完全生长
- Sub-sampling size，每次抽取的样本数，建议256即可，大于256，性能上不会有大的提升
- Number of tree,建议100，如果特征和样本规模比较大，可以适当增加

### 孤立树检测

上述两个算法用于生成孤立树，在孤立树生成之后，就可以通过样本点在树中的路径检测进行异常检测。

**Algorithm 3** : $PathLength(x, T, e)$

**Inputs** : $x$ - an instance, $T$ - an iTree, $e$ - current path length; to be initialized to zero when first called

**Output**: path length of $x$

1: **if** $T$ is an external node **then**
2: return $e + c(T.size)$ {$c(.)$ is defined in Equation 1}
3: **end if**
4: $a \leftarrow T.splitAtt$
5: **if** $x_a < T.splitValue$ **then**
6: return $PathLength(x, T.left, e + 1)$
7: **else** $\{x_a \geq T.splitValue\}$
8: return $PathLength(x, T.right, e + 1)$
9: **end if**

```
1: if T is an external node then
2:     return e + c(T.size) {c(.) is defined in Equation 1}
3: end if
4: a ← T.splitAtt
5: if x_a < T.splitValue then
6:     return PathLength(x, T.left, e + 1)
7: else {x_a ≥ T.splitValue}
```

```
8:     return PathLength(x, T.right, e + 1)
9: end if
```

在孤立树检测算法中接受三个值：

1. x 一个样本实例
2. T - 一个孤立树
3. e - 当前路径长度，初始值为0

孤立树检测的算法步骤：

1. 如果T是一个叶节点，则返回当前路径长度e与c(T.size)的和。其中c(T.size)是一个修正值
2. 如果T不是一个叶子节点，则：
    1. 令a是树T的一个分割属性
    2. 如果x在属性a上小于T的分割值，则递归调用PathLength()函数在左子树检测
    3. 如果x在属性a上大于T的分割值，则递归调用PathLength()函数在右子树检测
    4. 每次递归检测使路径值+1
3. 回到步骤1

路径长度h(x)的计算公式如下：

$$h(x) = e + c(T.size)$$

其中c(T.size)是依据树的样本数决定修正值，T.size表示和样本x在同一个叶子节点样本的个数。

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$
$$= 2[ln(n-1) + 0.5772156649] - \frac{2(n-1)}{n}$$

其中，0.5772156649为欧拉常数
H(i) 是调和数，可以近似为 l n ( 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。

**为何加入这一修正值?**

由于树的深度设置为ceiling(log(subsimpleSize)) ,所以可能大部分样本的PathLength都比较接近，而如果叶子结点的样本数越多，该样本是异常值的概率也较低(基于fewAndDifferent的假设)。另外c(n)是单调递增的，总之，**加了修正，使得异常和正常样本的PathLength差异更大，可以简单的理解，如果样本快速落入叶子结点，并且该叶子结点的样本数较少，那么其为异常样本的可能性较大。**

### 异常分数

score的范围为[0, 1]。

异常分数的计算公式：

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

其中$h(x)$ 为路径长度，$E(h(x))$为样本在t个itree中的平均路径长度，**c(n)：** 为n个样本构建一个BST二叉树的平均路径长度。
**为什么要算这个二叉排序树?**
因为iTree和BST的结构的等价性， 标准化借鉴BST（Binary Search Tree）去估计路径的平均长度c(n)。

上述公式中，指数部分的值域为$(-\infty, 0)$ 因此，异常分数$s(x,n)$的值域是(0,1)。

观察参数关系，可以得出：
平均路径越小，则异常分数越接近1，说明数据容易被孤立，则可以判定为异常点。
平均路径越大，则异常分数越接近0，数据不易被判定为异常点。

# Sk-learn实现

在了解孤立森林算法的思想之后，使用sklearn就可以很轻松的实现孤立森林进行异常值检测。

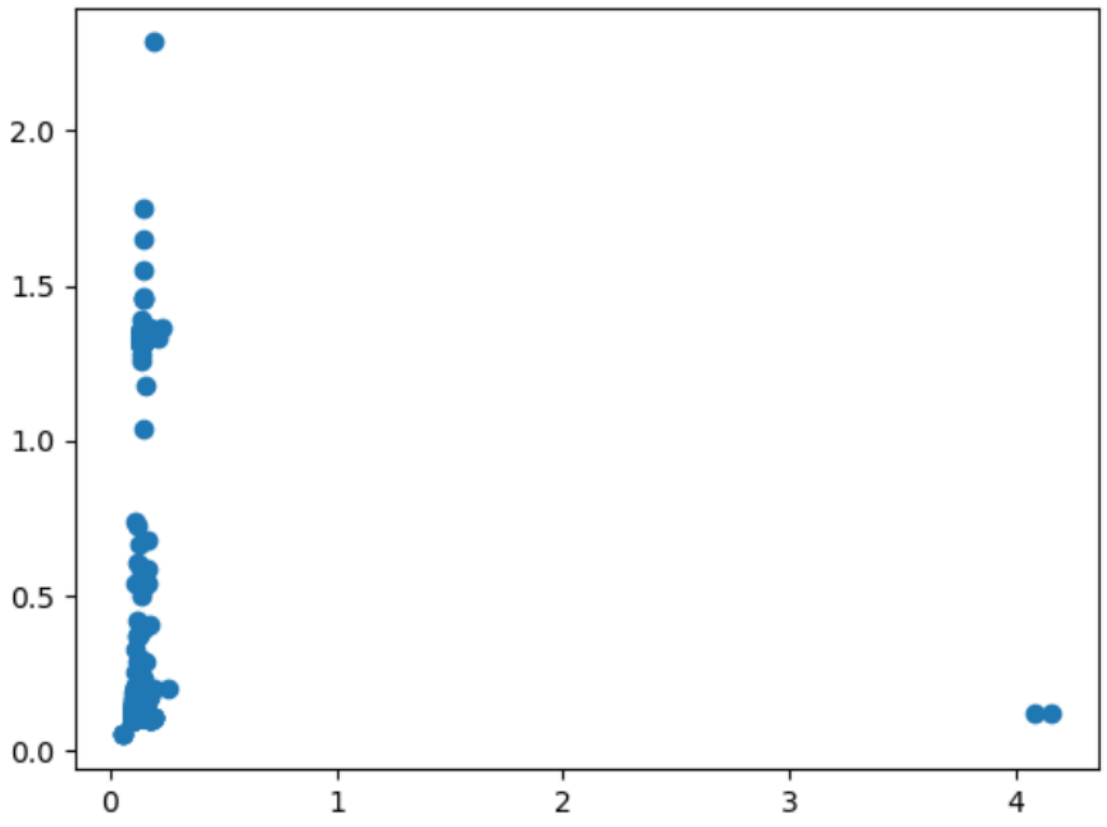
## 数据准备

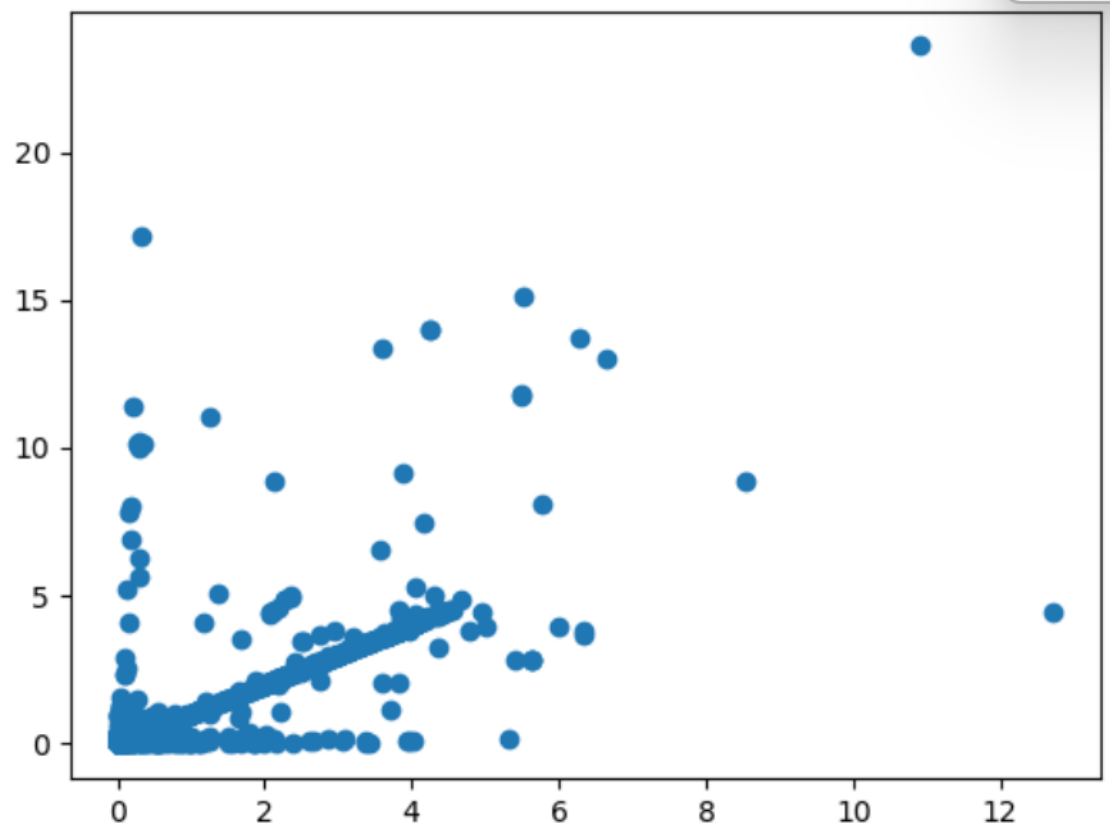
由于我正在开发一个入侵检测系统，正好需要使用孤立森林来实现异常流量的检测，我搜集了几个服务器一段时间的带宽数据，以此作为数据集进行训练，数据集的内容是活跃网卡在一秒内的流量收发速率，以及各个协议的流量速率。

由于最后所有的流量速率都会被归纳带总流量中，因此在训练的时候就只采用出入方向的总流量作为训练数据，并且在验证是否存在异常的时候也只使用总出入方向的流量。

流量速率的单位是：MB/S

这里我准备了一台不怎么活跃的服务器，以及一台已经上线了业务的服务器作为数据来源，使用matplotlib.pyplot绘制散点图：


2.0
1.5
1.0
0.5
0.0
0
1
2
3
4


20
15
10
5
0
0
2
4
6
8
10
12

图一是没有业务运行的服务器，图二是有业务运行的服务，横轴是入网流量情况，纵轴是出网流量情况。

这些数据在我的数据库中以如下形式储存：

| | id | ip | total_sent | total_receiv… | protocol_siz… |
|---|---|---|---|---|---|
| 1 | 1 | 49.232.245.103 | 0.0528 | 0.0524 | {"TCP": 0.002, "UD… |
| 2 | 2 | 49.232.245.103 | 0.1061 | 0.1057 | {"TCP": 0.0024, "U… |
| 3 | 3 | 49.232.245.103 | 0.1066 | 0.1046 | {"TCP": 0.0019, "U… |
| 4 | 4 | 49.232.245.103 | 0.1076 | 0.1067 | {"TCP": 0.003, "UD… |
| 5 | 5 | 49.232.245.103 | 0.106 | 0.1045 | {"TCP": 0.0003, "U… |
| 6 | 6 | 49.232.245.103 | 0.1065 | 0.1043 | {"TCP": 0.0017, "U… |
| 7 | 7 | 49.232.245.103 | 0.1319 | 0.1314 | {"TCP": 0.0011, "U… |
| 8 | 8 | 49.232.245.103 | 0.1059 | 0.1053 | {"TCP": 0.0031, "U… |
| 9 | 9 | 49.232.245.103 | 0.1081 | 0.107 | {"TCP": 0.0022, "U… |
| 10 | 10 | 49.232.245.103 | 0.1073 | 0.1052 | {"TCP": 0.0008, "U… |
| 11 | 11 | 49.232.245.103 | 0.1056 | 0.1044 | {"TCP": 0.0041, "U… |
| 12 | 12 | 49.232.245.103 | 0.1061 | 0.1043 | {"TCP": 0.0003, "U… |

## 模型训练

首先导入必要模块：

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sklearn.ensemble import IsolationForest
import joblib
import sys
import os
import matplotlib.pyplot as plt
```

由于数据储存在mysql数据库中，接着导入数据库模型以及数据库连接配置文件和创建数据库引擎以及数据库会话：

```
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from app.models import traffic_data
from app.config import mysql_username, mysql_password, mysql_host,
mysql_port, mysql_db
# 创建数据库连接
DATABASE_URL = f"mysql+pymysql://{mysql_username}:
{mysql_password}@{mysql_host}:{mysql_port}/{mysql_db}"
# 创建数据库引擎和会话
```

```
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

定义加载数据的方法，返回一个dataframe对象：

```
def load_data():
    # 创建数据库会话
    session = SessionLocal()
    try:
        # 查询所有流量数据
        query = session.query(traffic_data.TrafficData).all()

        # 解析数据
        data = []
        for row in query:
            protocol_sizes = row.protocol_sizes
            data.append({
                'ip': row.ip,
                'total_sent': row.total_sent,
                'total_received': row.total_received,
                'timestamp': row.timestamp
            })
    finally:
        # 关闭数据库会话
        session.close()

    # 返回数据的 DataFrame
    return pd.DataFrame(data)
```

加载数据、提取唯一IP地址、创建目录：

```
# 加载数据
df = load_data()

# 获取所有唯一的IP地址
ips = df['ip'].unique()

# 创建 models 目录
os.makedirs('models', exist_ok=True)
```

遍历IP列表，针对每个IP的出入网流量通过sklearn.ensemble的IsolutionForeast类训练模型。训练完成之后将模型保存为joblib文件。

```
for ip in ips:
    ip_data_sent = df[df['ip'] == ip][['total_sent']]
    ip_data_received = df[df['ip'] == ip][['total_received']]
    # print(ip_data_received['total_received'])
```

```
    # plt.scatter(ip_data_received['total_received'],
ip_data_sent['total_sent'])    # plt.show()
    model_sent = IsolationForest(contamination='auto', random_state=42)
    model_sent.fit(ip_data_sent)
    joblib.dump(model_sent,
f"models/{ip}_sent_isolation_forest_model.joblib")

    model_received = IsolationForest(contamination=0.01, random_state=42)
    model_received.fit(ip_data_received)
    joblib.dump(model_received,
f"models/{ip}_received_isolation_forest_model.joblib")

print("Models trained and saved for each IP.")
```

其中IsolutionForest类的实例化参数：

```
sklearn.ensemble.IsolationForest(
              *,
              n_estimators=100,
              max_samples='auto',
              contamination='auto',
              max_features=1.0,
              bootstrap=False,
              n_jobs=None,
              random_state=None,
              verbose=0,
              warm_start=False)
```

1. **n_estimators : int, optional (default=100)**

iTree的个数，指定该森林中生成的随机树数量，默认为100个

2. **max_samples : int or float, optional (default="auto")**

构建子树的样本数，整数为个数，小数为占全集的比例，用来训练随机数的样本数量，即子采样的大小。

如果设置的是一个int常数，那么就会从总样本X拉取max_samples个样本来生成一棵树iTree

如果设置的是一个float浮点数，那么就会从总样本X拉取max_samples * X.shape[0]个样本,X.shape[0]表示总样本个数

如果设置的是"auto"，则max_samples=min(256, n_samples)，n_samples即总样本的数量

如果max_samples值比提供的总样本数量还大的话，所有的样本都会用来构造数，意思就是没有采样了，构造的n_estimators棵iTree使用的样本都是一样的，即所有的样本

3. **contamination : float in (0., 0.5), optional (default=0.1)**

取值范围为(0., 0.5),表示异常数据占给定的数据集的比例，数据集中污染的数量，其实就是训练数据中异常数据的数量，比如数据集异常数据的比例。定义该参数值的作用是在决策函数中定义阈值。如果设置为'auto'，则决策函数的阈值就和论文中定义的一样

4. **max_features : int or float, optional (default=1.0)**

构建每个子树的特征数，整数位个数，小数为占全特征的比例，指定从总样本X中抽取来训练每棵树iTree的属性的数量，默认只使用一个属性

如果设置为int整数，则抽取max_features个属性

如果是float浮点数，则抽取max_features * X.shape[1]个属性

5. **bootstrap : boolean, optional (default=False)**
   采样是有放回还是无放回，如果为True，则各个树可放回地对训练数据进行采样。如果为False，则执行不放回的采样。
6. **n_jobs : int or None, optional (default=None)**
   在运行fit()和predict()函数时并行运行的作业数量。除了在joblib.parallel_backend上下文的情况下，None表示为1。设置为-1则表示使用所有可用的处理器
7. **random_state : int, RandomState instance or None, optional (default=None)**

每次训练的随机性

如果设置为int常数，则该random_state参数值是用于随机数生成器的种子

如果设置为RandomState实例，则该random_state就是一个随机数生成器

如果设置为None，该随机数生成器就是使用在np.random中的RandomState实例

8. **verbose : int, optional (default=0)**

训练中打印日志的详细程度，数值越大越详细

9. **warm_start : bool, optional (default=False)**
   当设置为True时，重用上一次调用的结果去fit,添加更多的树到上一次的森林1集合中；否则就fit一整个新的森林

## 检验测试

在预测程序中，给定的数据格式是：

```
# 示例数据
sample_data = {
    "ip": "121.43.132.126",
    "data": "{\"total_sent\": 3.3563, \"total_received\": 3.3874,
```

```
\"protocol_sizes\": {\"TCP\": 0.0513, \"UDP\": 0.0019, \"HTTP\": 0.0,
\"HTTPS\": 0.0471, \"SSH\": 0.0006, \"DNS\": 0.0019, \"ICMP\": 0.0}}",
    "type": "traffic_stats",
    "time": "2024-08-02 21:00:02"
}
```

其中包含IP和流量数据，type和time字段在这里没有意义。

定义一个预测方法：

```
def detect_anomaly(data):
    ip = data["ip"]
    traffic_data = json.loads(data["data"])
    total_sent = traffic_data["total_sent"]
    total_received = traffic_data["total_received"]

    model_sent_file = f"models/{ip}_sent_isolation_forest_model.joblib"
    model_received_file =
f"models/{ip}_received_isolation_forest_model.joblib"

    try:
        model_sent = joblib.load(model_sent_file)
        model_received = joblib.load(model_received_file)
    except FileNotFoundError:
        return f"No model found for IP: {ip}", {}, 0

    df_sent = pd.DataFrame([{'total_sent': total_sent}])
    df_received = pd.DataFrame([{'total_received': total_received}])

    # 预测是否为异常点
    is_anomaly_sent = model_sent.predict(df_sent)[0] == -1
    is_anomaly_received = model_received.predict(df_received)[0] == -1

    # 计算综合异常概率
    if is_anomaly_sent and is_anomaly_received:
        probability = 1.0  # 两个字段都异常，风险最高
        result = "Anomaly detected: High risk"
    elif is_anomaly_sent or is_anomaly_received:
        probability = 0.5  # 只有一个字段异常，风险中等
        result = "Anomaly detected: Moderate risk"
    else:
        probability = 0.0  # 两个字段都不异常，风险最低
        result = "Normal traffic"

    return result, {'total_sent': is_anomaly_sent, 'total_received':
is_anomaly_received}, probability
```

提取出输入数据中的关键信息，定义格式化字符串，根据IP来加载出网流量如入网流量的模型。

通过 `joblib.load()` 方法加载joblib模型，如果没有对应的模型则返回未找到此IP的模型。

将total_sent即出网流量和total_received即入网流量转为dafaframe格式。
通过joblib模型实例的predict方法进行预测，当异常时返回-1，否则返回1。

由于这里的异常流量检测需要综合出入网流量的综合判断，因此随后简单对其进行逻辑与的判断进行加权。

孤立森林模型的 `joblib.predict` 方法只能只能返回1和-1。实际上，查看一下他的源代码：

```
check_is_fitted(self)
decision_func = self.decision_function(X)
is_inlier = np.ones_like(decision_func, dtype=int)
is_inlier[decision_func < 0] = -1
return is_inlier
```

他也是通过 `decision_fucntion` 计算异常分数之后，根据异常分数是否小于0来决定是否为异常点的。

## 为什么和公式不一样?

理论上，由异常分数的计算公式，孤立森林的异常得分 $s(x,n)$是一个介于0和1之间的值。

但是实际上在sklearn中，为了简化异常点的判断， `decision_function()` 方法对异常分数进行了归一化处理，并通过以下公式计算：

$$decision_score(x) = E(h(x)) - c(n)$$

也就是通过平均的路径长度减去修正值，也就是平均路径长度的期望值，这样做的好处是可以直接利用零作为分界点，简化了异常点的判断。

由上述公式，当实际平均路径值较小，而期望值较大，也就是异常得分为负值 时，说明样本点很快落入了叶子结点，将被判定为异常点。
同理，如果实际平均值大于期望值，也就是异常得分大于0，说明样本距离根节点较远，落入叶子节点更慢，此时样本点为正常样本点。