基于特征化网络流量和机器学习的电信网络威胁检测

张志心，魏子舒

摘要

在本次实验中，我们对基于两份电信网络流量数据集，设计了基于特征化网络流量和机器学习方法的电信网络威胁检测系统（**IDS**）。我们对实验的数据集进行了数据预处理，对特征进行了进一步的筛选和精细处理，使用于训练的特征数据能够更好的拟合出结果；在此基础上，我们采用了多种不同的机器学习方法对特征数据集进行训练和测试，并对不同的模型性能进行分析。

1 网络威胁检测的背景和挑战

1.1 网络威胁检测概述

网络威胁（网络入侵）是一系列旨在破坏计算平台上资源的完整性、机密性或可用性的行为。网络威胁检测系统（**IDS**）用于监控网络流量以寻找可疑流量并在发现此类流量的时候发出警报，并在检测到异常的时候采取行动，比如阻止来自可疑地 **IP** 地址的流量等等。

IDS 有以下四种类型：

1. 网络入侵防范系统（**NIPS**）：通过分析协议活动监视整个网络以寻找可疑流量；
2. 无线入侵防范系统：通过分析无线网络协议件来监视整个网络以寻找可疑流量；
3. 网络行为分析：检查网络以识别诸如分布式拒绝服务之类的威胁；
4. 基于主机的入侵防范系统：通过分析主机来监测主机的可疑活动。

网络威胁检测通常有两种检测技术：

1. 异常检测：基于与正常行为偏离的偏差识别恶意活动，并将这些偏差视为攻击；
2. 特征检测（又称滥用检测）：检测假设入侵者活动可以用一种模式来表示，系统的目标是检测主体活动是否符合这些模式。 它可以将已有的入侵方法检查出来，但对新的入侵方法无能为力。此外，此类检测方法的误报率很高。

网络威胁检测系统（**IDS**）的设计通常包括以下四个步骤：

1. 收集数据：为了进行**IDS**，我们需要收集有关网络流量的信息，如流量类型、主机和协议详情。更详细的信息比如 **IP** 地址，设备，请求内容，时间等等。
2. 特征选择：从数据特征中提取有用的特征；
3. 分析数据：对所选特征数据进行分析，以确定数据是否异常；
4. 执行操作：当发生攻击时，**IDS**通过系统管理员生成警报或警报，并告知攻击类型。**IDS**还通过关闭网络端口和终止进程参与控制攻击。

1.2 网络威胁检测面临的挑战

网络威胁检测系统面临的挑战包括但不只如下几种：

1. 攻击流量的种类越来越多样化；
2. 攻击流量常常具有伪装性和隐藏性，传统检测手段束手无策；
3. IDS需要处理庞大的数据流，因此需要具备高吞吐量和低延迟；
4. 需要降低威胁检测的误报率以提升效率；

2 实验原理

本实验为利用特征数据处理的二分类问题。

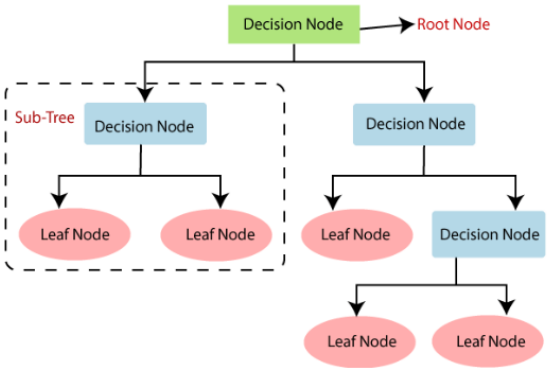
问题定义：假设给定训练数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中，$x_i=(x_i^{(1)},x_i^{(2)},\cdots,x_i^{(n)})^T$ 为输入实例（特征向量），n 为特征个数，$y_i\in\{0,1\}$ 为类标记，$i=1,2,\cdots,N$，N 为样本容量。目标是给一个具体的模型，能够尽可能准确地给实例进行分类。

2.1 机器学习的基本概念方法

2.1.1 决策树模型

分类决策树模型是一种描述对实例进行分类的树形结构。由结点（node）和有向边（derected edge）组成。结点有两种类型：内部结点（internal node）和叶节点（leaf node）。内部结点表示一个特征或属性，叶结点表示一个类。

- 从根结点开始，对实例的某一特征进行测试，
  根据测试结果，将实例分配到其子结点，每个子结点对应该特征的一个取值。
  如此递归地对实例进行测试并分配，直到达到叶结点。最后将实例分到叶结点的类中。
- 我们可以认为决策树就是一个分支语句，不断执行 `if-then` 的操作。
  直到达到叶子结点，然后返回分类的结果。



2.1.1.1 条件熵

条件熵用于在已知随机变量 X 的条件下随机变量 Y 的不确定性，记为 $H(Y|X)$，定义为 X 给定条件下 Y 的条件概率分布的熵对 X 的数学期望：

$$H(Y|X)=\sum_{i=1}^{n}p_iH(Y|X=x_i),p_i=P(X=x_i),i=1,2,\cdots,n.$$

特别的，如果有 0 概率，令 $0\log 0=0$。然而在实际情况下，熵和条件熵的具体值往往无法计算，在具体学习中，我们使用对它们极大似然估计，称为经验熵和经验条件熵。

2.1.1.2 信息增益→信息增益比

信息增益**(infomation gain)**表示得知特征 X 的信息后使得类 Y 的信息的不确定性减少的程度。特征 A 对训练数据集 D 的信息增益 $g(D,A)$，定义为集合 D 的经验熵与特征 A 在个顶条件下 D 的经验条件熵 $H(D|A)$ 之差。

$$g(D,A)=H(D)-H(D|A).$$

选择信息增益最大的特征首先用于决策。

信息增益总是偏向于取值较多的特征，为了解决这一问题，使用信息增益比**(infomation gain ratio)**来进行矫正。特征 A 对训练数据集 D 的信息增益比 $g_R(D,A)$，定义为其信息增益 $g(D,A)$ 与训练数据集关于特征 A 的值的经验熵 $H_A(D)$ 之比。假设特征 A 有 n_A 种取值，

$$g_R(D,A)=\frac{g(D,A)}{H_A(D)},H_A(D)=-\sum_{i=1}^{n_A}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}.$$

2.1.1.3 基尼系数

实际问题中，由于信息增益（比）过于难算，还可以使用如基尼指数$Gini(p)$代替进行特征选择。

分类问题中，假设有 K 个类，样本点属于第 k 类的概率为 p_k，则概率分布的基尼指数定义为：

$$\text{Gini}(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2.$$

对于给定样本集合 D，其基尼指数为：

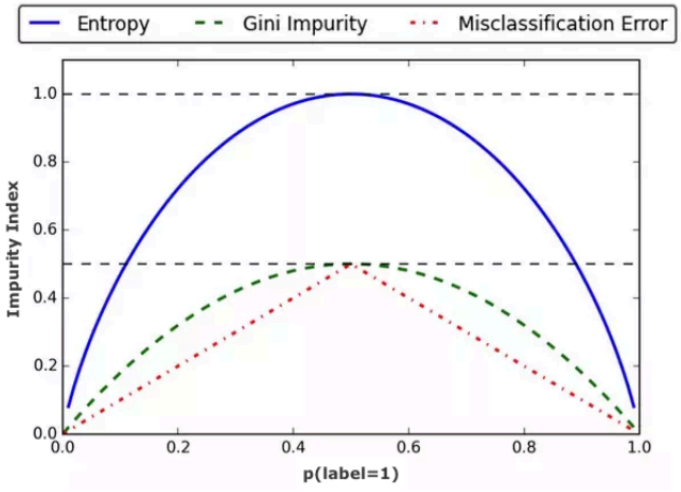
$$\text{Gini}(D)=1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2$$

这里，C_k 是 D 中属于第 k 类的样本子集，K 是类的个数。

如果样本集合 D 根据特征 A 的取值被分为 $D_1,\cdots,D_{n_A}$ 个部分，那么在特征 A 的条件下，集合 D 的基尼指数定义为：

$$\text{Gini}(D,A)=\sum_{i=1}^{n_A}\frac{|D_i|}{|D|}\text{Gini}(D_i).$$

对于二分类问题，基尼指数$\text{Gini}(p)$，熵**(bit)**之半 $H(p)/2$ 和分类误差率的关系如下所示：



2.1.1.4 决策树剪枝

设树 T 的叶结点个数为 $|T|$，N_t 表示在 t 号叶结点上的样本点。$H_t(T)$ 为叶结点 t 上的经验熵。决策树算法的损失函数可以认为是：

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T).$$

如果决策树的深度过高，结点数过多，不仅意味着构造的模型更加复杂，而且会出现过拟合的现象，因此我们可以修正损失函数为

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha|T|.$$

其中 $\alpha \geq 0$ 为一个超参。

2.1.2 随机森林

随机森林由很多决策树构成，不同决策树之间相关性很低；

做法是从样本集中随机抽取少量样本，再从特征集中随机抽取 m 个属性，满足 $m << n$，n 为总特征数，建立决策树，按照此方法建立一系列的决策树。

所有决策树参与投票，决定分类。

算法优点：

- 可以处理高维数据，且不用降维；
- 不容易过拟合；
- 实现简单，而且容易设计并行计算，效率高。

2.1.3 K临近算法（KNN）

K 临近算法目的是找到特征空间中与测试点最接近的 **K** 个点，并用它们的特征进行投票，得到预测结果。常常使用 **k-d** 树（**k** 维特征空间的划分树，每一层对一个维度进行划分，叶子结点代表两两不相交的子空间）进行维护：

求解过程如下：

输入：已构造的 **k-d** 树，目标点 **x**。

输出：**x** 的 **k** 近邻点集 **S**。

算法：一开始，$S=\emptyset$。S 在实现的过程中，可以使用线性表（当 **k** 比较时），或者采用搜索树/优先队列。集合的自维护体制为，若执行加点操作 **insert(S, y)**：将 **y** 加入 **S** 中，若此时 **S** 中的元素个数超过 **k**，则将 **S** 中与 **x** 结点距离最远的元素删除（因此可以考虑 **S** 中保存 **{v, dist(x, v)}** 的二元组，以方便查询。对集合的另一个查询操作：查询集合中离 **x** 距离最远的点与 **x** 的距离，为了方便，将其记为 **d**，并且设定当 S 的元素个数不足 **k** 时，$d=+\infty$。

1. 深度优先搜索：在 **k-d** 树中找出代表区域包含了目标点 **x** 的叶结点：从根节点出发，递归地向下访问 **k-d** 树。若目标点 **x** 当前维坐标小于切分点的坐标，则移动到左子结点，否则移动到右子结点。直到子结点为叶结点为止。设当前叶子结点为 **ul**，执行一次 **insert(S, ul)**，即尝试将 **ul** 加入集合中。
2. 回溯过程：从当前叶子结点向上回溯。对于当前节点 **u**，首先执行 **insert(S, u)**，即尝试将 **u** 加入集合中。**u** 的子结点中至少有一个子结点没有被访问过，对于一个没有访问过的子结点，检查它的所代表的区域是否与以目标点为圆心，以 **d**（集合中离**x**）为球心的超球体相交，（或者检查目标点与该区域的估算距离是否小于 **d**）。如果是，则递归搜索该子结点的子树，否则不进入搜索。

当实例点集合中的点数大于等于 **k** 时，返回一个 **x** 的 **k** 近邻点集合 **S**，表示实例点集中与 **x** 距离最近的 **k** 个点。

2.1.4 朴素贝叶斯算法

利用先验分布计算后验分布，从而得到测试点最有可能的标签。

令 $P(Y=c_k)=\theta$，设其先验分布为 $\theta\sim Beta(a,b)$，令 $n=\sum_{i=1}^{N}I(y_i=c_k)$。

可以计算后验分布为：

$$
\begin{aligned}
P(\theta|y_1,y_2,\dots y_n)&=\frac{P(\theta)P(y_1,y_2,\dots,y_n|\theta)}{\int_0^1 P(y_1,y_2,\dots,y_n|\theta)P(\theta)\mathrm{d}\theta} &(1)\\
&=\frac{\theta^n(1-\theta)^{N-n}\theta^{a-1}(1-\theta)^{b-1}}{\int_0^1\theta^n(1-\theta)^{N-n}\theta^{a-1}(1-\theta)^{b-1}\mathrm{d}\theta} &(2)\\
&=\frac{\theta^{n+a-1}(1-\theta)^{N-n+b-1}}{B(n+a,N-n+b)} &(3)
\end{aligned}
$$

接着计算后验分布的极值点：

$$
\frac{\partial\ln P(\theta|y_1,y_2\dots,y_n)}{\partial\theta}=\frac{1}{B(n+a,N-n+b)}\left[\frac{n+a-1}{\theta}-\frac{N-n+b-1}{1-\theta}\right]
$$

$$
\left.\frac{\partial\ln P(\theta|y_1,y_2\dots,y_n)}{\partial\theta}\right|_{\theta=\hat{\theta}}=0
$$

所以，

$$\hat{\theta}=\frac{n+a-1}{N+a+b-2}$$

取 $a=\lambda+1, b=(K-1)\lambda-3$，则

$$\hat{P}_\lambda(Y=c_k)=\frac{n+\lambda}{N+K\lambda}=\frac{\sum_{i=1}^N I(y_i=c_k)+\lambda}{N+K\lambda}\quad k=1,2,\dots,K$$

2.1.5 支持向量机算法

2.1.5.1 线性支持向量机

支持向量机（**SVM**）算法是解决分类问题的重要工具。

假设 d 维空间中有 N 个样本点 $x_1,\dots,x_N\in\mathbb{R}^d$，每个样本点有一个标签 $y_1,\dots,y_n\in\{0,1\}$。

我们要构造一个超平面 $w\cdot x=0$，使得正样本点和负样本点在曲面的两侧，且与样本点的间隔尽可能大。即最大化

$$\gamma=\min_{i=1}^N y_i(\frac{w}{\|w\|}+b).$$

这等价于最优化问题

$$\min_{w,b}\frac{1}{2}\|w\|^2$$
$$\text{s.t.}y_i(w\cdot x_i+b)\geq 1,\forall i=1,\dots,n.$$

然而，数据并不是完全理想化的，可能完全分隔所有正例和负例的超平面不存在（如“异或”运算），但我们可以尽量满足多数点的条件。即加入惩罚项，每有一个样本点进入错误的区域一个单位长度就进行 C 的惩罚。将最优化问题改写为

$$\min_{w,b}\frac{1}{2}\|w\|^2+C\sum_{i=1}^N\xi_i$$
$$\text{s.t.}y_i(w\cdot x_i+b)\geq 1-\xi_i,\forall i=1,\dots,n.$$

直接求解这个二次规划问题很困难，考虑利用拉格朗日乘子法转化为对偶问题（过程省略）：

$$\min_{\alpha}\frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N\alpha_i\alpha_j y_i y_j(x_i\cdot x_j)-\sum_{i=1}^N\alpha_i$$
$$\text{s.t.}\sum_{i=1}^N\alpha_i y_i=0.$$

这是一个凸二次规划问题，有经典的优化算法求解它。求出最优解 α^* 之后，再由

$$w^*=\sum_{i=1}^N\alpha_i^* y_i x_i b^*=y_j-\sum_{i=1}^N y_i\alpha_i^*(x_i\cdot x_j)$$

即可。最终得到的分类函数决策式为

$$\hat{Y}=\text{sign}(w\cdot X+b)$$

2.1.5.2 非线形支持向量机和核函数

有时用超平面分隔两个区域并不是最合理的方式，应该用超曲面。但任意超曲面这个范围太大了，所以我们考虑限制一个曲面族。这就需要引入核函数来限制。

将原规划问题中的内积 $x_i \cdot x_j$ 修改为正定核函数 $K(x_i, x_j)$，求解新的规划问题。

相应的，

$$\hat{Y} = \text{sign}(\sum_{i=1}^{N} a_i^* y_i K(x_i, x) + b^*)$$

核函数可选

- 线性核函数 $K(x, y) = x \cdot y$；此时即线性支持向量机。
- 多项式核函数 $K(x, y) = (x \cdot y + 1)^p$，参数 p 可调；
- 高斯核函数 $K(x, y) = \exp\{-\gamma \|x - y\|^2\}$。

2.1.6 eXtreme Gradient Boosting

一将弱分类算法提升为一个强分类算法的算法，主要是用决策树模型进行集成，通过反复迭代训练弱学习器来提高模型的性能。每一轮迭代都根据前一轮的结果来调整模型，以减小预测误差。

XGBoost 在目标函数中引入了正则化项，帮助控制模型的复杂性，防止过拟合。正则化项包括 **L1** 正则化（**Lasso**）和 **L2** 正则化（**Ridge**），可以根据需要进行调整。

2.1.7 神经网络算法

本次实验中采用的神经网络如下：

2.1.7.1 模型架构

输入层：输入数据。

隐藏层**1**：全连接层（**Linear**）具有**256**个神经元。

批标准化层**1**：应用批标准化以加速收敛和提高模型稳定性。

ReLU激活层**1**：使用**ReLU**（**Rectified Linear Unit**）激活函数。

隐藏层**2**：全连接层具有**128**个神经元。

批标准化层**2**：第二个批标准化层。

ReLU激活层**2**：第二个**ReLU**激活层。

输出层：全连接层，输出**1**个值。

Sigmoid激活层：将输出映射到范围**[0, 1]**，用于二分类问题。

2.1.7.2 正则化

使用了丢弃（**Dropout**）层，防止过拟合，参数设置为**0.5**。

2.1.7.3 损失函数和优化器

使用二元交叉熵损失函数（`nn.BCELoss()`）。优化器选择了**Adam**优化器，学习率设置为**0.001**。

2.1.7.4 学习率调度

使用学习率调度器（`lr_scheduler.StepLR`），在每**5**个**epoch**时将学习率缩小为原来的**0.1**。

2.1.7.5 训练过程

模型在训练集上进行多个**epoch**的训练，每个**epoch**中进行前向传播、反向传播和优化器更新。每个**epoch**结束后，在测试集上评估模型性能并输出测试准确度。

在所有**epoch**完成后，模型再次在测试集上进行最终评估，输出最终测试准确度。

3 实验过程

3.1 数据集介绍

本实验使用了两份电信网络威胁数据集，其中包含的特征如下图所示：

生成时间	结束时间	事件名称	事件类型	攻击结果	威胁等级	源IP	源端口	目的IP	目的端口	设备来源	发生次数	规则ID	响应码	设备动作	请求体	响应体	研判
2023/5/9 0:02	2023/5/9 15:0	挖矿程序查询D	挖矿样本	成功	较大	199.19.54.1 美国	53	112.17.46.91(1 中国,浙江,杭州	46217	石桥DPI-uts2(	24	49014	0	允许	--	--	非攻击
2023/5/9 5:54	2023/5/9 15:0	Web服务登录证	通用网络攻击	企图	轻微	39.170.123.25 中国,浙江,杭州	1984	211.140.25.24(中国,浙江,杭州	6080	石桥UTS2(188	2	50603	200	允许	POST /login H Host: 211.140. Content-Type: User-Agent: M Connection: K Content-Leng wlanAcName=	HTTP/1.1 200 Date: Tue, 09 Content-Type: Content-Leng Cache-Contro Expires: Tue, 1 Keep-Alive: tir Connection: K {"returnCode"	非攻击
2023/5/9 8:23	2023/5/9 15:0	Todesk远程控[	通用网络攻击	企图	一般	10.72.133.164 --	63449	211.140.13.188 中国,浙江,杭州	53	萧山UTS5(188	6	50621	0	允许	--	--	非攻击
2023/5/9 3:38	2023/5/9 15:0	挖矿蠕虫Wann	蠕虫病毒	成功	较大	192.52.178.30 美国,弗吉尼亚(	53	112.17.46.24(1 中国,浙江,杭州	59278	石桥DPI-uts2(	3	49019	0	允许	--	--	非攻击
2023/5/9 10:0	2023/5/9 15:0	Realtek Jungle	注入型攻击	企图	较大	176.97.210.100 --	55323	218.205.60.54 中国,浙江,杭州	9034	石桥DPI-uts2(	6	25702	0	允许	--	--	非攻击
2023/5/9 10:0	2023/5/9 15:0	Realtek Jungle	注入型攻击	企图	较大	176.97.210.100 --	53885	112.15.232.51(中国,浙江,杭州	9034	石桥DPI-uts2(	5	25702	0	允许	--	--	非攻击
2023/5/9 1:26	2023/5/9 15:0	Cobalt Strike对	通用网络攻击	成功	较大	117.180.184.54 中国	46435	211.140.12.65(中国,浙江,杭州	53	石桥DPI-uts1('	6	41763	0	允许	--	--	非攻击
2023/5/9 15:0	2023/5/9 15:0	Realtek Jungle	注入型攻击	企图	较大	176.97.210.100 --	57227	211.140.13.19(中国,浙江,杭州	9034	石桥DPI-uts2(	1	25702	0	允许	--	--	非攻击
2023/5/9 11:0	2023/5/9 15:0	Realtek Jungle	注入型攻击	企图	较大	176.97.210.100 --	33190	120.199.235.2 中国,浙江	9034	石桥DPI-uts2(	2	25702	0	允许	--	--	非攻击
2023/5/9 7:29	2023/5/9 15:0	挖矿程序查询D	挖矿样本	成功	较大	205.251.197.19 美国	53	112.17.46.88(1 中国,浙江,杭州	57970	石桥DPI-uts1('	10	49014	0	允许	--	--	非攻击
2023/5/9 15:0	2023/5/9 15:0	驱动人生下载器	通用网络攻击	成功	较大	117.180.184.21 中国	36700	211.140.12.65(中国,浙江,杭州	53	石桥DPI-uts1('	1	49040	0	允许	--	--	非攻击
2023/5/9 15:0	2023/5/9 15:0	Realtek Jungle	注入型攻击	企图	较大	45.61.185.107 美国,纽约州,纽	34869	112.15.229.53(中国,浙江,杭州	9034	石桥DPI-uts1('	1	25702	0	允许	--	--	非攻击

其中，‘研判’为数据标签列，包含的标签有：攻击、非攻击、无法判定。

3.2 实验环境

Python 3.11.6：安装了 **sklearn**、**pytorch**、**numpy**、**pandas** 等环境。

使用 **jupyter notebook** 进行开发。

3.3 数据预处理

查看每一列的取值个数：

```
data.nunique()
```

```
生成时间      2863
结束时间      2871
事件名称       830
事件类型        27
攻击结果         3
```

```
威胁等级         4
源IP       15958
源端口       36515
目的IP        913
目的端口       2029
设备来源         45
发生次数        680
规则ID        834
响应码          34
设备动作          3
请求体       29680
响应体       12164
研判            3
dtype: int64
```

查看部分列具体有哪些种类：

```
event_type_v = data['事件类型'].unique()
attack_res_v = data['攻击结果'].unique()
threaten_lev_v = data['威胁等级'].unique()
response_code_v = data['响应码'].unique()
device_act_v = data['设备动作'].unique()
label_v = data['研判'].unique()

print("事件类型", event_type_v)
print("攻击结果", attack_res_v)
print("威胁等级", threaten_lev_v)
print("响应码", response_code_v)
print("设备动作", device_act_v)
print("研判", label_v)
```

```
事件类型 ['注入型攻击' '挖矿样本' '扫描探测' 'web漏洞攻击' '通用网络攻击' '目录遍历攻击' '远
程登录' '蠕虫病毒' '拒绝服务攻击'
 '木马' '恶意后门' '账号弱口令' '主机漏洞攻击' '文件上传攻击' 'webshell攻击' '远程文件包含攻
击' '勒索软件'
 '数据泄露' '远程代码命令执行' '缓冲区溢出攻击' '身份认证绕过' '反序列化攻击' '权限提升' '跨
站脚本攻击' '未授权访问攻击'
 '账号暴力破解' '僵尸网络']
攻击结果 ['企图' '成功' '失败']
威胁等级 ['较大' '一般' '轻微' '重大']
响应码 ['0' '--' '200' '404' '403' '302' '400' '405' '100' '406' '511' '401'
 '501' '502' '301' '500' '206' '505' 0 200 404 403 100 401 302 400 501 406
 500 502 511 405 505 '417']
设备动作 ['允许' '--' '阻断']
研判 ['非攻击' '无法研判' '攻击']
```

这里发现响应码的类型不一致，考虑将其统一修改成字符串类型：

```
data['响应码'] = data['响应码'].astype(str)
response_code_v = data['响应码'].unique()
print("响应码", response_code_v)
```

```
响应码 ['0' '--' '200' '404' '403' '302' '400' '405' '100' '406' '511' '401'
 '501' '502' '301' '500' '206' '505' '417']
```

通过观察数据发现，“无法研判”的行中的请求体往往是一些乱码，而且该类标签对于做网络威胁检测的意义不大，为了方便进行分类问题，首先删除所有研判结果为“无法研判”的行：

20811		一般	120.216.65.197 中国,河南,信阳	30201	117.148.137.43 (117.148.137.43)ip:117.148.137.43 中国,浙江,宁波	9091	石桥DPI-ut	1	10520	0	允许	N fl 6&4 O ek / r`~ kt q ? v I _ L)(6 > ,0 / $ (k # ' g 9 3 = < 5 / u 3t http/1.1 1 0. - 3 & $ m .A $q F & M ?$ 5 aR W	--	无法研判
20812		一般	223.73.138.29 中国,广东,	26142	117.148.137.43 (117.148.13	9091	石桥DPI-ut	2	10520	0	允许	X w T kl w 7 0 , ($ kjih9876 2.*& = 5 / + '# g@?>3210 EDCB1-)% < /	--	无法研判
20813		一般	223.104.162.213 中国,浙江,温州	30908	211.140.25.163(211.140.25.163)ip:211.140.25.163 中国,浙江,杭州	10070	石桥UTS1(:	1	60678	0	允许	t J 3+ {9GV , UV 5 " f v 8 Fi m ocL H >:- 8 X'm w >/$ G/ " Q f V Au XX j L;U X- h ~ z m ^ J { ~ & 1 M N b A_o - R= 9 r n ob } Li s b S r })C z g?] d ; > u >l < c qC G A +i 1 t,p y dZ538z A5 +0x J q C l_) 8 ;L C1 i q E %)g# Z _ 2@ ; BC Tn i 8OQ0 & d<' c n Z TH 0wrEdm ~ } %0J J % lH w d A S 9 9-h 9P < 7c J 4Z / E kL d c rN O,9 9 Y V wV$h 2x o) xr1 5L@ mT s }dU g 5! u bu D i| b m < F x @ N4v _ Q _ i ((jA D D X bf , I tLA Eq <]& } r ' kV	--	无法研判

```
data = data[data['研判'] != '无法研判']
label_v = data['研判'].unique()
print("研判", label_v)
```

```
研判 ['非攻击' '攻击']
```

根据数据特征的类型：

- 事件名称，事件类型，攻击结果，响应码，设备来源，规则ID，响应码，设备动作，研判为离散格式，考虑先编码为整数格式(后采用独热编码)。
- 威胁等级根据轻微，一般，较大，重大，为连续整数，考虑分别赋值为 **0, 1, 2, 3**；

```
def setup_map (column_name):
    tmp_map = dict()
    tmp = tuple(data[column_name].unique())
    for i in range(len(tmp)):
        tmp_map[tmp[i]] = i
    data[column_name] = data[column_name].map(tmp_map)

setup_map('事件名称')
setup_map('事件类型')
setup_map('攻击结果')
setup_map('设备来源')
setup_map('规则ID')
setup_map('响应码')
setup_map('设备动作')
setup_map('研判')
```

```
label_v = data['研判'].unique()
print("研判", label_v)

device_act_v = data['设备动作'].unique()
print("设备动作", device_act_v)
```

```
研判 [0 1]
设备动作 [0 1 2]
```

```
threat_level_mapping = {'轻微': 0, '一般': 1, '较大': 2, '重大': 3}
data['威胁等级'] = data['威胁等级'].map(threat_level_mapping)

print('威胁等级', data['威胁等级'].sample(5))
```

```
威胁等级 7662     1
46813    2
41105    2
4379     1
7949     2
Name: 威胁等级, dtype: int64
```

此时一共有 **57939** 行数据，每个数据有 **18** 个特征。

去掉生成时间和结束时间这两个对研判结果相关性不大的列，再考虑删除重复的行（数据集中的很多攻击都不止进行了一次）。

```
data = data.drop(['生成时间', '结束时间'], axis = 1)
data.drop_duplicates(inplace=True)
data.shape
```

```
(59892, 16)
```

此时数据库的响应体还有一些为 **null**，考虑将其替换为空串：

```
data['响应体'] = data['响应体'].fillna('')
print(data['响应体'].isnull().sum())
```

```
0
```

接下来考虑如何处理请求体和响应体这两个部分的内容：

先查看一下研判结果为攻击的行：

```
                                                  源IP       源端口
31530                          115.194.118.178\n中国,浙江,杭州   34030\t   \
22665                                36.138.140.208\n中国   46182\t
31742  112.17.44.75(112.17.44.75)ip:112.17.44.75\n中国,...   37268\t
31627                          122.224.146.227\n中国,浙江,杭州   35788\t
24230                                36.138.140.208\n中国   51732\t

                                                 目的IP    目的端口
31530  112.15.232.48(112.15.232.48)ip:112.15.232.48\n...   30230   \
22665  218.205.57.148(218.205.57.148)ip:218.205.57.14...    9001
31742  39.173.77.32(39.173.77.32)ip:39.173.77.32\n中国,...    1162
```

```
31627  218.205.111.73(218.205.111.73)ip:218.205.111.7...   9000
24230  112.13.167.228(112.13.167.228)ip:112.13.167.22...   8060

                                                    请求体
31530  POST /cgi-bin/cgitest?LD_PRELOAD=/proc/self/fd...  \
22665  PUT ISOP#@!START/_users/org.couchdb.user:dvtlh...
31742  POST /console/j_security_check HTTP/1.1 \nHost...
31627  GET /joomla2/index.php HTTP/1.1 \nAccept-Encod...
24230  GET /${(#a=@org.apache.commons.io.IOUtils@toSt...

                                                    响应体
31530                                                --
22665                                                --
31742                                                --
31627  HTTP/1.1 404 Not Found \nServer: nginx \nDate:...
24230  HTTP/1.1 404 Not Found \nServer: allmedia/1.18...
```

在查看部分研判结果为非攻击的行:

```
                                                    源IP      源端口
38511                          39.185.215.115\n中国,浙江,宁波     9018  \
11718                       193.118.53.142\n荷兰,北荷兰省,阿姆斯特丹  42617\t
33906                          117.147.40.254\n中国,浙江,杭州     6005
1193   112.17.44.76(112.17.44.76)ip:112.17.44.76\n中国,...  33098\t
7197                                 117.180.185.118\n中国  52397\t

                                                   目的IP   目的端口
38511  112.13.167.152(112.13.167.152)ip:112.13.167.15...  19081  \
11718  112.17.44.215(112.17.44.215)ip:112.17.44.215\n...   6969
33906  112.13.167.152(112.13.167.152)ip:112.13.167.15...  19081
1193    111.1.7.246(111.1.7.246)ip:111.1.7.246\n中国,浙江,台州   1161
7197   211.140.12.65(211.140.12.65)ip:211.140.12.65\n...     53

                                                    请求体
38511  POSTISOP#@!START ISOP#@!END/rest/cmcc_softprob...  \
11718                                                --
33906  POST /rest/cmcc_softprobe/home_gateway/realtim...
1193
7197                                                 --

                                                    响应体
38511  HTTP/1.1ISOP#@!START ISOP#@!END200ISOP#@!START...
11718                                                --
33906                                                --
1193                                                 --
7197                                                 --
```

可以发现攻击数据和非攻击数据在请求体和响应体上的差异很大，考虑解析这两个部分:

```
rst line =  GET
/${Class.forName("com.opensymphony.webwork.ServletActionContext").getMethod("getResponse",
null).invoke(null,null).setHeader("X-Cmd-
Response",Class.forName("javax.script.ScriptEngineManager").newInstance().getEngineByName(
"nashorn").eval("var d='';var i =
java.lang.Runtime.getRuntime().exec('id').getInputStream();
while(i.available())d+=String.fromCharCode(i.read());d"))}/ HTTP/1.1
headers =  Host: 112.13.167.100:8090
User-Agent: Mozilla/5.0 zgrab/0.x
Accept: */*
Accept-Encoding: gzip

body =

rst line =  GET /member/ajax_membergroup.php?action=post&membergroup=@``/*!50000Union */
/*!50000select */ md5(910332797) -- @`` HTTP/1.1
headers =  Host: 218.205.111.5:8098
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.98 Safari/537.36
Accept: */*
Accept-Encoding: gzip

body =

rst line =  POST /cgi-bin/twiki/bin/view HTTP/1.0
headers =  User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
Host: 223.92.5.30
Content-Length: 63
Content-Type: application/x-www-form-urlencoded
Accept: */*
Connection: close
body =  rev=1|ISOP#@!START|echo ISOP#@!ENDxHELLOxWORLDx||
```

可以发现请求体主要包含：请求行(**rstline**)，请求头(**headers**)，请求体(**body**)。

以上述三个为例：

- 请求行可能出现注入攻击：此时请求行的长度比往常的长，可以考虑统计长度和字符出现的频率/方差进行判断。
- 请求体中也可能注入恶意数据，还可能伪装用户代理信息等，可以考虑统计字符出现的频率/方差进行判断。此外还应该重点关注 **Host** 和 **User-Agent** 两行。
  在本次作业中，决定取 **User-Agent** 行的长度，字符频率方差，**Host** 长度，字符频率方差，**Headers** 字符频率方差共**5**个特征。
- 请求体可能出现远程命令执行攻击：此时请求体相比于别的较为特殊，可以考虑统计字符出现的频率/方差进行判断。

经过替换后的特征如下：

```
Index(['事件名称', '事件类型', '攻击结果', '威胁等级', '源IP', '源端口', '目的IP', '目的端
口', '设备来源',
       '发生次数', '规则ID', '响应码', '设备动作', '研判', 'rstline_len', 'useragent_len',
       'hostlen', 'rstlinevar', 'bodyvar', 'headervar', 'useragentvar',
       'hostvar', 'resvar'],
      dtype='object')
(57892, 23)
```

考虑用源端口的数量和目的端口的数量替换这两个特征：

```
def GetNUM(x):
    ret = 1
    for c in x:
        if c == ',':
            ret += 1
    return ret

data['目的端口'] = data['目的端口'].apply(lambda x: GetNUM(str(x)))
data['源端口'] = data['源端口'].apply(lambda x : GetNUM(str(x)))
print(max(data['目的端口']), max(data['源端口']))
```

```
100 100
```

接下来查看 **IP** 地址的组成：

```
9199        223.92.10.46\n中国,浙江,温州
16718          147.185.239.218\n美国
28606           36.138.140.208\n中国
30341           36.138.140.208\n中国
6925      183.136.225.31\n中国,浙江,嘉兴
47369           36.138.140.208\n中国
45798              1.117.84.70\n中国
7102               10.3.33.132\n--
26163           36.138.140.208\n中国
13294    123.152.193.186\n中国,浙江,宁波
Name: 源IP, dtype: object
```

对于上述结构，我们考虑将 **IP** 地址转化为整数表示，比如 `151.80.13.43` 转化为 **2538605867**，然后将城市部分提取为字符串格式，用这两个新特征替换原先的特征。

最终的特征如下：

```
Index(['事件名称', '事件类型', '攻击结果', '威胁等级', '源端口', '目的端口', '设备来源', '发
生次数', '规则ID',
       '响应码', '设备动作', '研判', 'rstline_len', 'useragent_len', 'hostlen',
       'rstlinevar', 'bodyvar', 'headervar', 'useragentvar', 'hostvar',
       'resvar', 'srcIPcity', 'desIPcity', 'srcIP', 'desIP'],
      dtype='object')
```

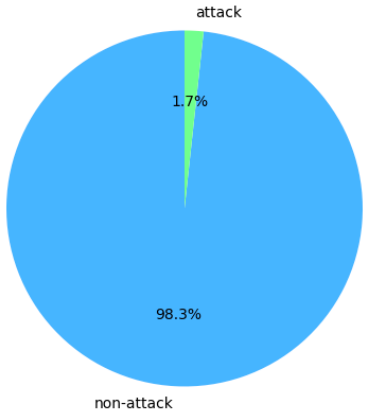
查看部分数据：

	事件名称	事件类型	攻击结果	威胁等级	源端口	目的端口	设备来源	发生次数	规则ID	响应码	...	rstlinevar	bodyvar	headervar	useragentvar	hostvar	resvar	srcIPcity	desIPcity	srcIP	desIP
2909	1	1	1	2	1	1	1	1	1	0	...	0.000000	0.000000	0.00000	0	0.000000	0.0	美国	中国,浙江,杭州	3355015169	1880174167
28482	126	5	0	1	1	1	13	9	132	6	...	22.552457	0.000000	5.26392	79	1.204082	0.0	中国	中国,浙江,杭州	613059792	1879942991
8779	1	1	1	2	1	1	6	1	1	0	...	0.000000	0.000000	0.00000	0	0.000000	0.0	美国,弗吉尼亚州,雷斯顿	中国,浙江,杭州	3223532318	1880174168
4928	781	4	0	1	1	1	1	2	786	0	...	0.146684	0.711538	0.00000	0	0.000000	0.0	中国,广东,佛山	中国,浙江,宁波	2028677243	1972668715
801	6	4	1	2	1	1	6	5	6	0	...	0.000000	0.000000	0.00000	0	0.000000	0.0	美国	中国,浙江,杭州	2728336367	1880174170

5 rows × 25 columns

至此，数据预处理已经全部完成。

下为数据集中两类标签的占比：



我们认为这个数据集中攻击流量太少，容易导致训练效果不佳，尤其是会降低召回率（因为模型会认为绝大部分的数据都是非攻击的）。

3.4 模型测试

分割数据集：

- 在分割数据集之前，对之前提到的几个特征进行独热编码。

```
need_one_hot_columns = ['srcIPcity', 'desIPcity', '事件名称','事件类型','攻击结果','响应码','设备来源','规则ID','响应码','设备动作']
data_encoded = pd.get_dummies(data, columns=need_one_hot_columns)
```

- 分割训练集和测试集

```
x = data_encoded.drop('研判', axis=1)
y = data_encoded['研判']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

定义模型评估函数：输出准确率，精确率，召回率。

这三个评价指标的定义如下：（正例为攻击）

1. 准确率(Accuracy)：

   定义：模型正确预测的样本数量与总样本数量之比。

   公式：$\dfrac{TP+TN}{TP+FP+TN+FN}$

2. 精确率(Precision)：

   定义：所有被模型预测为正类别的样本中，实际为正类别的样本所占的比例。

公式：$\dfrac{TP}{TP+FP}$

3. 召回率(**Recall**)：

   定义：所有实际正类别的样本中，模型成功预测为正类别的样本所占的比例。

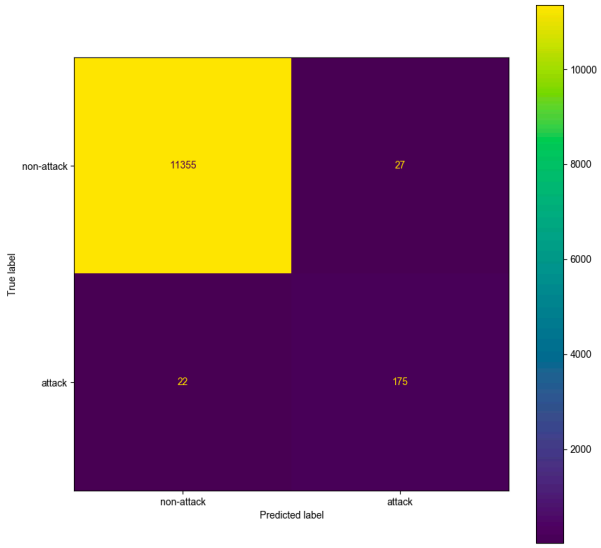
   公式：$\dfrac{TP}{TP+FN}$

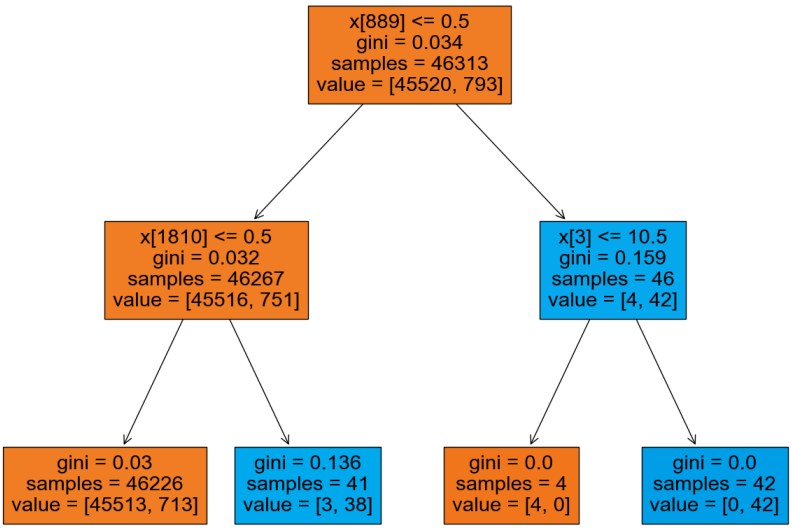
精确率可以反应模型的是否不容易误判，而召回率可以反应模型对威胁检测的全面性。

3.4.1 决策树算法

3.4.1.1 设置最大深度为 3

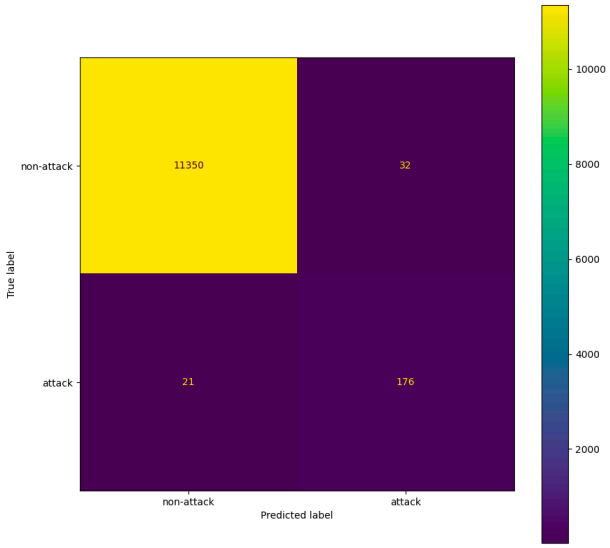
```
Training Accuracy DecisionTreeClassifier 99.99784077904692
Test Accuracy DecisionTreeClassifier 99.57682010536317
Training Precesion DecisionTreeClassifier 100.0
Test Precesion DecisionTreeClassifier 86.63366336633663
Training Recall DecisionTreeClassifier 99.87389659520807
Test Recall DecisionTreeClassifier 88.83248730964468
```

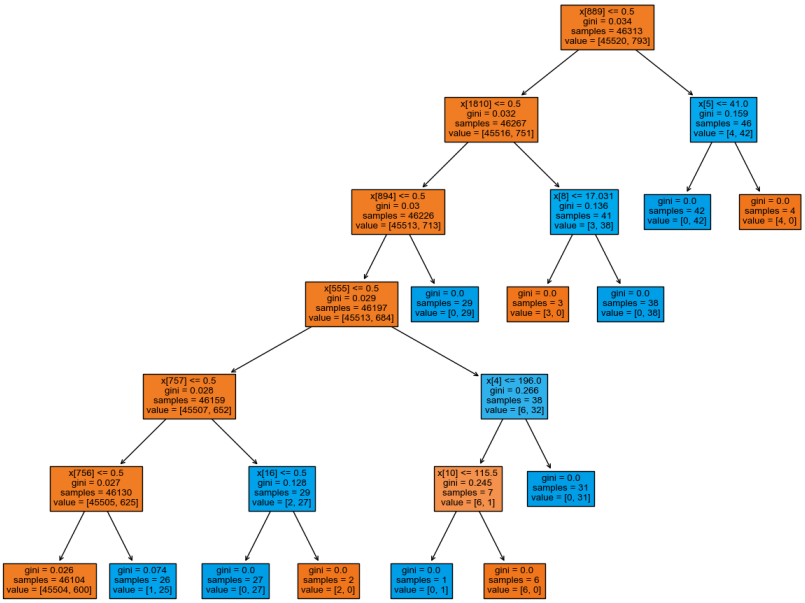




3.4.1.2 设置最大深度为 6

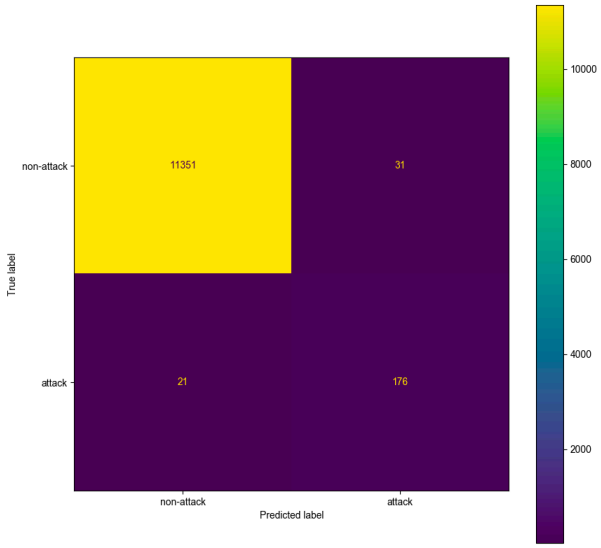
```
Training Accuracy DecisionTreeClassifier 99.99784077904692
Test Accuracy DecisionTreeClassifier 99.54227480784178
Training Precesion DecisionTreeClassifier 100.0
Test Precesion DecisionTreeClassifier 84.61538461538461
Training Recall DecisionTreeClassifier 99.87389659520807
Test Recall DecisionTreeClassifier 89.34010152284264
```

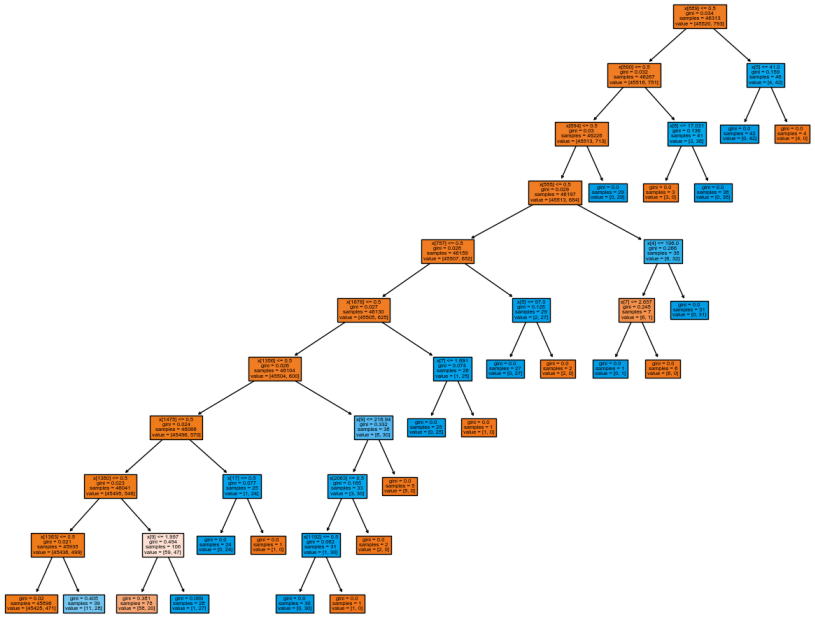




3.4.1.3 设置最大深度为 10

```
Training Accuracy DecisionTreeClassifier 99.99784077904692
Test Accuracy DecisionTreeClassifier 99.55091113222213
Training Precesion DecisionTreeClassifier 100.0
Test Precesion DecisionTreeClassifier 85.02415458937197
Training Recall DecisionTreeClassifier 99.87389659520807
Test Recall DecisionTreeClassifier 89.34010152284264
```

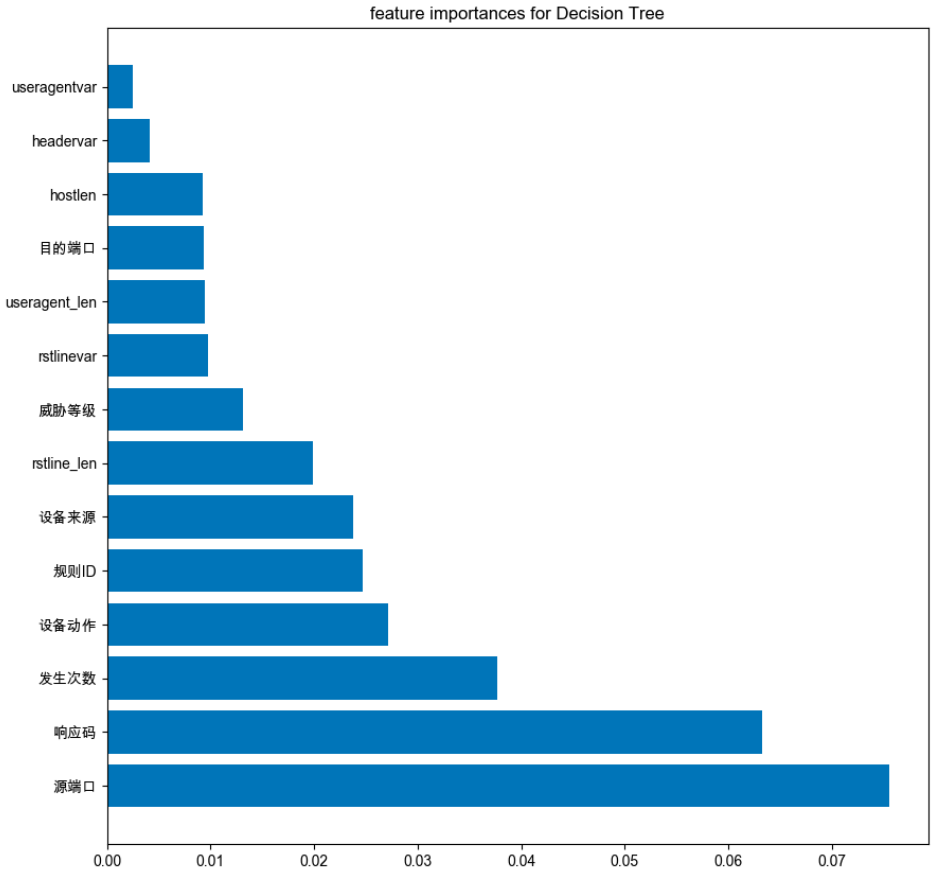




3.4.1.4 分析

采用三种最大深度跑出来结果并没有明显的差异，但深度越深，召回率有微小提升。但是模型明显复杂了很多。

以下为决策树分析的对于分类问题最重要最大的几个特征。



可以发现源端口的重要性最大，我认为这个可能原因是该特征的取值最多，所以自然带来的信息增益比较大。而响应码的取值虽然不多，但是依然带来很大的信息增益。

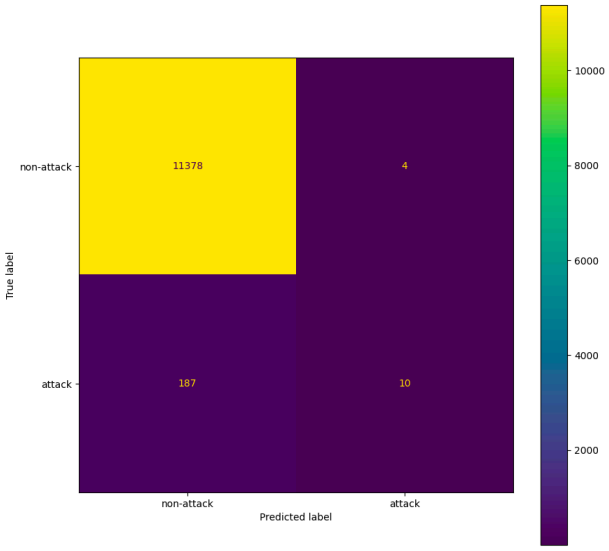
此外发生次数，设备动作，规则ID，设备来源和请求行长度也起一定作用，其中 请求行长度 (rstline_len) 为新分离出的特征，说明其对于分类问题是有价值的。

3.4.2 K邻近（KNN）

```
Training Accuracy KNeighborsClassifier 98.32660376136289
Test Accuracy KNeighborsClassifier 98.35046204335434
Training Precesion KNeighborsClassifier 73.68421052631578
Test Precesion KNeighborsClassifier 71.42857142857143
Training Recall KNeighborsClassifier 3.5308953341740232
Test Recall KNeighborsClassifier 5.0761421319796955
```

KNN 算法的召回率很低，不适合处理该问题。

原因是训练集中的正例点太少了，在进行对 KNN 中的投票结果很不利。

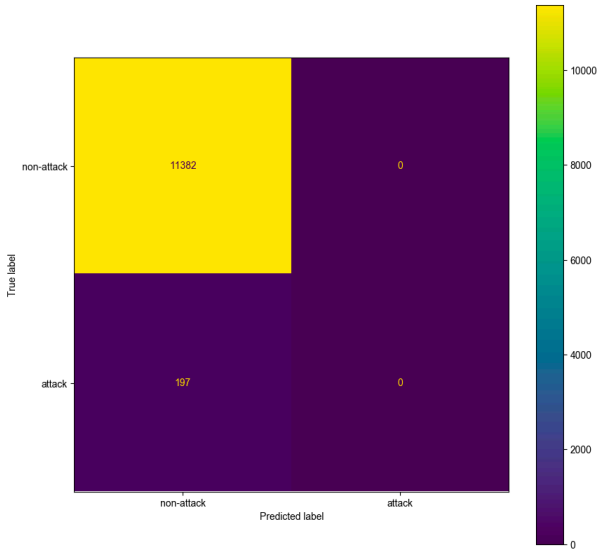


3.4.3 朴素贝叶斯法、SVM（线性核与 RBF 核）

这两个模型的召回率为0。不适合该分类问题。

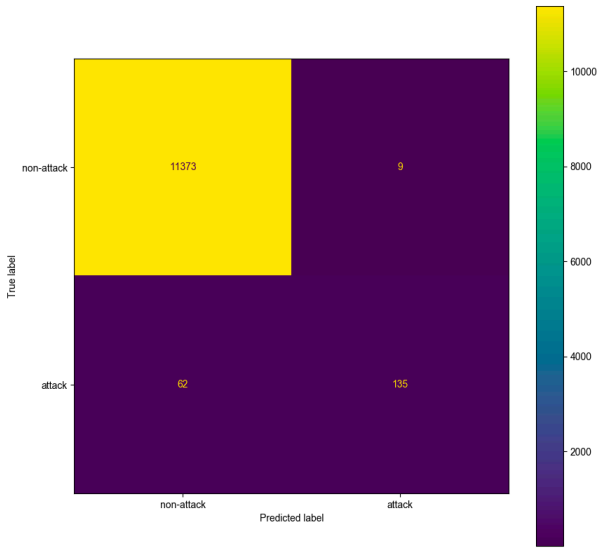
其中 Baiyes 法失败的原因可能是先验分布不够合理，并且极大似然估计的时候正例点不够，导致估计出来的概率几乎为 0。

而 SVM 则可能是因为特征空间中无关维度太多，如果经过一定的降维并且增加训练集中的正例点可能可以提升模型能力。



3.4.4 随机森林

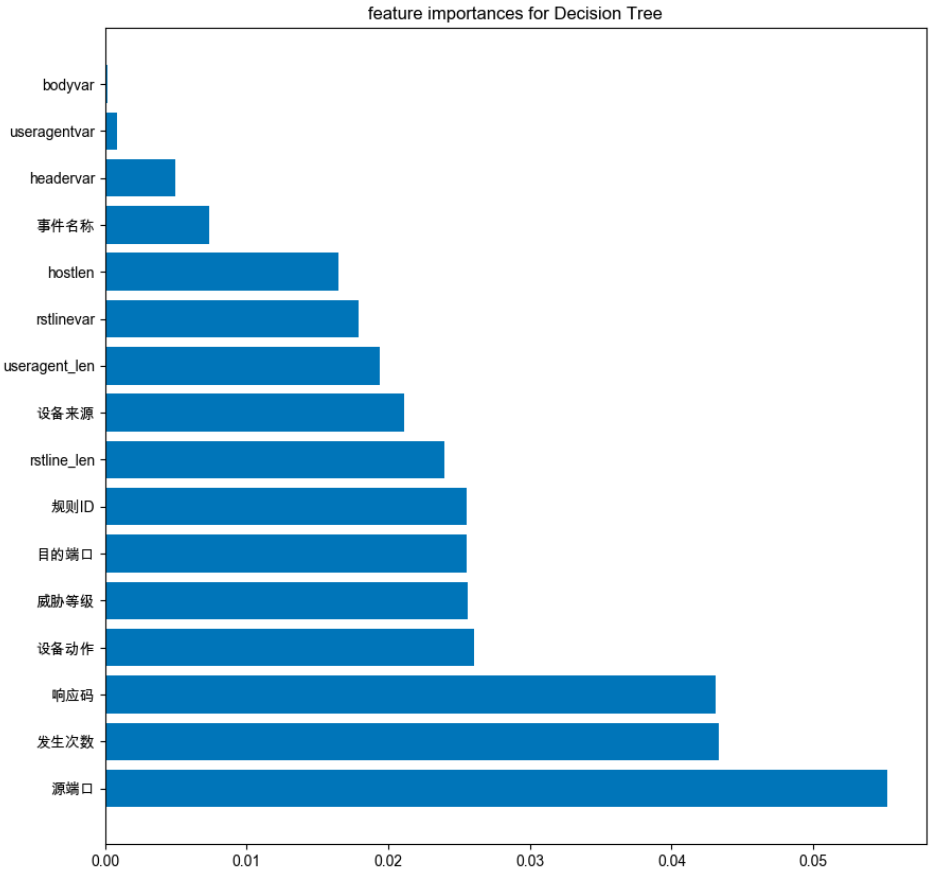
```
Training Accuracy RandomForestClassifier 99.99784077904692
Test Accuracy RandomForestClassifier 99.3868209689956
Training Precesion RandomForestClassifier 100.0
Test Precesion RandomForestClassifier 93.75
Training Recall RandomForestClassifier 99.87389659520807
Test Recall RandomForestClassifier 68.52791878172589
```



相比决策树算法，随机森林的精确度高了不少，但是召回率却降低地很明显。

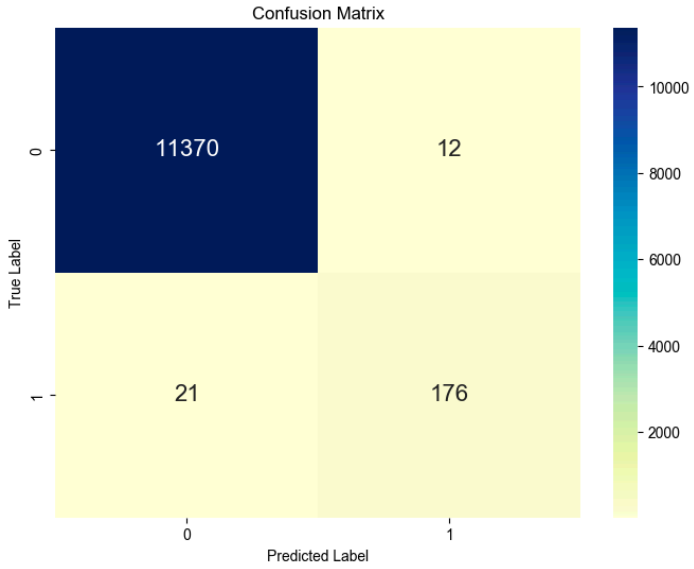
对于特征较少的小规模数据集，集成算法反而不如单模型来的优秀。

但是由于建立了更多的决策树，对结果产生影响的特征变得更多了：



3.4.5 eXtreme Gradient Boosting

```
Training Error XGBOOST 0.005215406377251053
Test error XGBOOST 0.054076832985980317

Accuracy: 0.9971500130
Precision: 0.9361702128
Recall: 0.8934010152
```



eGboost 算法的准确率、精确度、召回率的综合在上述几个模型中是最好的。

3.4.6 FNN

```
class ImprovedNN(nn.Module):
    def __init__(self):
        super(ImprovedNN, self).__init__()
        self.fc1 = nn.Linear(x_train_.shape[1], 256)
        self.bn1 = nn.BatchNorm1d(256)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(256, 128)
        self.bn2 = nn.BatchNorm1d(128)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(128, 1)
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.fc1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.bn2(x)
        x = self.relu2(x)
        x = self.dropout(x)
        x = self.fc3(x)
        x = self.sigmoid(x)
        return x


model = ImprovedNN()
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

epochs = 20
```

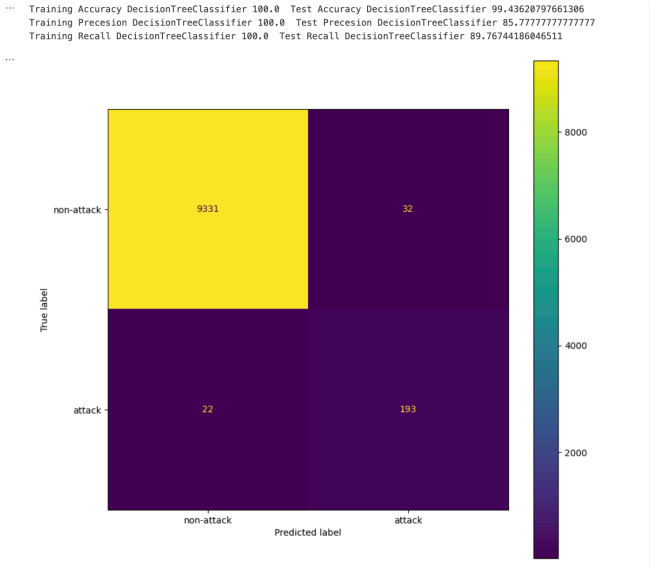
```python
for epoch in range(epochs):
    optimizer.zero_grad()
    outputs = model(x_train_)
    loss = criterion(outputs.squeeze(), y_train_)
    loss.backward()
    optimizer.step()
    scheduler.step()

    with torch.no_grad():
        model.eval()
        outputs_test = model(x_test_)
        predicted_test = (outputs_test.squeeze() > 0.5).float()
        accuracy_test = (predicted_test == y_test_).sum().item() / len(y_test_)

    print(f'Epoch {epoch + 1}/{epochs}, Training Loss: {loss.item()}, Test Accuracy:
{accuracy_test}')

with torch.no_grad():
    model.eval()
    outputs_final = model(x_test_)
    predicted_final = (outputs_final.squeeze() > 0.5).float()
    accuracy_final = (predicted_final == y_test_).sum().item() / len(y_test_)

print(f'Final Test Accuracy: {accuracy_final}')
```

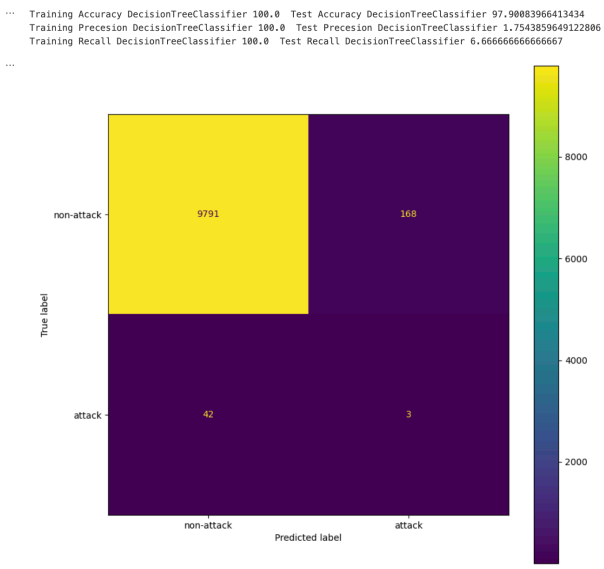
```
Epoch 1/20, Training Loss: 0.8375369906425476, Test Accuracy: 0.9829864409707229
Epoch 2/20, Training Loss: 0.4039837419986725, Test Accuracy: 0.9829864409707229
Epoch 3/20, Training Loss: 0.1287190467119217, Test Accuracy: 0.9829864409707229
Epoch 4/20, Training Loss: 0.08990456163883209, Test Accuracy: 0.9829864409707229
Epoch 5/20, Training Loss: 0.09505240619182587, Test Accuracy: 0.9829864409707229
Epoch 6/20, Training Loss: 0.10558175295591354, Test Accuracy: 0.9829864409707229
Epoch 7/20, Training Loss: 0.10645735263824463, Test Accuracy: 0.9829864409707229
Epoch 8/20, Training Loss: 0.10709362477064133, Test Accuracy: 0.9829864409707229
Epoch 9/20, Training Loss: 0.10753706842660904, Test Accuracy: 0.9829864409707229
Epoch 10/20, Training Loss: 0.10781864076852798, Test Accuracy: 0.9829864409707229
Epoch 11/20, Training Loss: 0.10796122997999191, Test Accuracy: 0.9829864409707229
Epoch 12/20, Training Loss: 0.10796357691287994, Test Accuracy: 0.9829864409707229
Epoch 13/20, Training Loss: 0.10795553028583527, Test Accuracy: 0.9829864409707229
Epoch 14/20, Training Loss: 0.10793884098529816, Test Accuracy: 0.9829864409707229
Epoch 15/20, Training Loss: 0.10791455954313278, Test Accuracy: 0.9829864409707229
Epoch 16/20, Training Loss: 0.10788331180810928, Test Accuracy: 0.9829864409707229
Epoch 17/20, Training Loss: 0.10787952691316605, Test Accuracy: 0.9829864409707229
Epoch 18/20, Training Loss: 0.10787514597177505, Test Accuracy: 0.9829864409707229
Epoch 19/20, Training Loss: 0.1078701838850975, Test Accuracy: 0.9829864409707229
Epoch 20/20, Training Loss: 0.10786474496126175, Test Accuracy: 0.9829864409707229
Final Test Accuracy: 0.9829864409707229
```

神经网络的准确率虽然也有 **98%**，但是召回率仍然很低。

3.4.7 模型泛化能力

我们在 0406 数据集上进行了决策树算法的测试，又把 0406 的数据集作为训练集把 0509 的数据集作为测试集进行了决策树的测试。两次测试结果如下：





可以发现，在 0406 数据集上的测试结果在合并后数据集上好，但是用于测试 0509 的效果却很不好，说明模型的泛化能力并不够好。

要想要得到泛化能力更好的模型，应该更多的数据，尤其需要不同时间的数据进行训练。而单日的数据集中的网络攻击数量少，种类少，特征单一，不具有很好的可总结性，因此分类效果并不好。

4 总结

本次实验我们系统的总结了各类经典分类算法在 IDS 中的应用。对于特征化流量网络，我们认为决策树算法已经梯度提升树算法的拟合效果是最好的。决策树类算法能够很好的计算出重要的特征用于分类，对于特征网络的适应力极强。而诸如 SVM，KNN，贝叶斯估计，神经网络等算法，都需要对特征空间进行很好的设计，在算力和数据集规模不够，正例点数量显著不足，特征多且不易处理的情况下，这些算法在设计上要困难许多，得到的结果也不尽如人意。在 IDS 的设计中，好的特征是能够更好的做分类问题的至关重要的因素，应当充分利用现有的流量特征提取关键信息。为了应对网络攻击的多样化，我们应当基于更庞大的数据集进行训练才可以得到效果最好的模型。