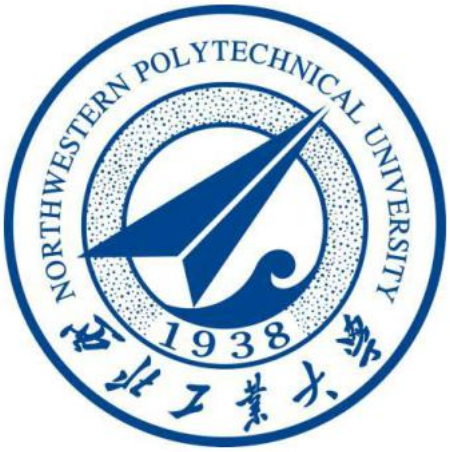
信 息 内 容 安 全 实 验 报 告



实验项目名称：	基于分类方法的入侵检测技术
班级：	SC011701
姓名：	裴嘉琨
学号：	2017302242
指导教师：	杨黎斌
实验时间：	2020.5.2

摘要

随着电子信息行业的兴起，更多的数据被数据化、电子化。未来越多的电子信息被使用于人们的日常交流、工业生产、国家运营等方面。然而庞大的数据，在不同的系统中、在传递的过程中，有可能会受到入侵。所以针对一些数据所传递的信息是否正确我们需要入侵检测来进行检测，以保证我们的生活以及工业生产的正常运行。

在 1999 年的 KDD 中，提供了大量的数据集供大家进行入侵检测试验。在本次实验中，采用了 KNN、SVM、随机森林算法、决策树算法、bagging 算法五种不同的算法，针对已有的数据集进行入侵检测的测试，主要目的是将正常数据与入侵数据可以区分开来。除此之外，还针对三分类情况下，进行了部分算法的测试。并对最终的实验结果进行了分析总结。

关键词：入侵检测　　KNN　　SVM　　CUP99

Summary

With the rise of electronic information industry, more data are digitized and electronic.In the future, more electronic information will be used in People's Daily communication, industrial production, national operation and other aspects.However, large amounts of data may be invaded in different systems and in the process of transmission.Therefore, we need intrusion detection to detect whether the information conveyed by some data is correct, so as to ensure the normal operation of our life and industrial production.

KDD in 1999, a large number of data sets were provided for intrusion detection tests.In this experiment, five different algorithms, KNN, SVM, random forest algorithm, decision tree algorithm and bagging algorithm, were used to carry out intrusion detection tests on the existing data sets. The main purpose is to distinguish the normal data from the intrusion data.In addition, some algorithms are tested in the case of three classifications.Finally, the experimental results are analyzed and summarized.

Key words: intrusion detection KNN SVM CUP99

目录

1. 背景

随着近几年互联网的快速发展，信息安全成为人们所关注的一重要话题。不论是在人们生活中或是在工业生产中，数据的传递是一个很重要的环节。为了防止人们的生活、工业生产、国家运营出现差错，入侵检测技术逐渐兴起。入侵检测技术是防火墙技术的合理补充，可以提供对内部攻击、外部攻击及误操作的实时保护。所谓入侵是指未经授权的系统用户窃取或试图窃取系统的访问权限，以及系统的授权用户超出被授予访问权限利用系统资源的行为，这种行为危害系统的完整性、保密性和可用性。

入侵检测系统（IDS：Intrusion Detection System)可以检测入侵行为能够帮助网络系统快速发现网络攻击的发生，提高了信息安全基础结构的完整性。因此，IDS 自 1980 年被提出以来，越来越多地受到人们的关注，成为信息安全领域的研究重点。数据挖掘技术是帮助用户发现隐藏在大型数据库中的规律和模式，且融合了人工智能、统计、模式识别等多种学科理论与方法技术的大规模数据处理的方法与手段。

本问在基于 KDD CUP99 的数据集使用上，分别使用 5 中不同的机器学习算法对其进行入侵检测的测试。

2. KDD CUP99 数据库介绍

2.1 KDD CUP

国际知识发现和数据挖掘竞赛（KDD-CUP）竞赛是由 ACM 的数据挖掘及知识发现专委会（SIGKDD）主办的数据挖掘研究领域的国际顶级赛事。其中 KDD 的英文全称是 Knowledge Discovery and Data Mining，即知识发现与数据挖掘。

KDD Cup 比赛由 ACM 协会的 SIGKDD 分会举办，从 1997 年开始，每年举办一次，目前是数据挖掘领域最有影响力的赛事。该比赛同时面向企业界和学术界，云集了世界数据挖掘界的顶尖专家、学者、工程师、学生等参加，通过竞赛，为数据挖掘从业者们提供了一个学术交流和研究成果展示的理想场所。

2.2 KDD CUP99 数据集

KDD CUP99 数据集则是在 1999 年，国际知识发现和数据挖掘竞赛所出的题目对应的数据集[1]。该年的比赛主要内容是竞争任务是建立一个网络入侵检测器，这是一种能够区分称为入侵或攻击的“不良”连接和“良好”的正常连接的预测模型。该数据集包含一组要审核的标准数据，其中包括在军事网络环境中模拟的多种入侵。

其中主要的数据文件包含：

[1] 数据集：http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

kddcup.names	功能列表；
kddcup.data.gz	完整数据集；
kddcup.data_10_percent.gz	10％的训练数据集
kddcup.newtestdata_10_percent_unlabeled.gz	10%的测试数据集
kddcup.testdata.unlabeled.gz	完整的测试数据集
kddcup.testdata.unlabeled_10_percent.gz	10%的测试数据集
corrected.gz	正确标签的测试数据
training_attack_types	入侵类型列表
typo-correction.txt	关于数据集中的简要说明

2.3 数据集数据来源

1998 年美国国防部高级规划署（DARPA）在 MIT 林肯实验室进行了一项入侵检测评估项目。林肯实验室建立了模拟美国空军局域网的一个网络环境，收集了 9 周时间的 TCPdump()网络连接和系统审计数据，仿真各种用户类型、各种不同的网络流量和攻击手段，使它就像一个真实的网络环境。这些 TCPdump 采集的原始数据被分为两个部分：7 周时间的训练数据，大概包含 5,000,000 多个网络连接记录，剩下的 2 周时间的测试数据大概包含 2,000,000 个网络连接记录。

一个网络连接定义为在某个时间内从开始到结束的 TCP 数据包序列，并且在这段时间内，数据在预定义的协议下（如 TCP、UDP）从源 IP 地址到目的 IP 地址的传递。每个网络连接被标记为正常（normal）或异常（attack），异常类型被细分为 4 大类共 39 种攻击类型，其中 22 种攻击类型出现在训练集中，另有 17 种未知攻击类型出现在测试集中。

4 种异常类型分别是：

- **DOS（denial-of-service）**拒绝服务攻击，例如 ping-of-death, syn flood, smurf 等。
- **R2L（unauthorized access from a remote machine to a local machine）**来自远程主机的未授权访问，例如 guessing password。
- **U2R（unauthorized access to local superuser privileges by a local unpivileged user）**未授权的本地超级用户特权访问，例如 buffer overflow attacks。
- **PROBING（surveillance and probing）** 端口监视或扫描，例如 port-scan, ping-sweep 等。

2.4 数据详细解释

下面展现了其中 3 条记录，总共有 42 项特征，最后一列是标记特征（Label），其他前 41 项特征共分为四大类。

- TCP 连接基本特征（共 9 种，序号 1～9）
- TCP 连接的内容特征（共 13 种，序号 10～22）
- 基于时间的网络流量统计特征 （共 9 种，序号 23～31）
- 基于主机的网络流量统计特征 （共 10 种，序号 32～41）

2.4.1 TCP 连接基本特征

基本连接特征包含了一些连接的基本属性，如连续时间，协议类型，传送的字节数等。

1) **duration** 连接持续时间，以秒为单位，连续类型。范围是 [0, 58329] 。它的定义是从 TCP 连接以 3 次握手建立算起，到 FIN/ACK 连接结束为止的时间；若为 UDP 协议类型，则将每个 UDP 数据包作为一条连接。数据集中出现大量的 duration = 0 的情况，是因为该条连接的持续时间不足 1 秒。
2) **protocol_type** 协议类型，离散类型，共有 3 种：TCP, UDP, ICMP。
3) **service** 目标主机的网络服务类型，离散类型，共有 70 种。'aol', 'auth', 'bgp', 'courier', 'csnet_ns', 'ctf', 'daytime', 'discard', 'domain', 'domain_u', 'echo', 'eco_i', 'ecr_i', 'efs', 'exec', 'finger', 'ftp', 'ftp_data', 'gopher', 'harvest', 'hostnames', 'http', 'http_2784′, 'http_443′, 'http_8001′, 'imap4′, 'IRC', 'iso_tsap', 'klogin', 'kshell', 'ldap', 'link', 'login', 'mtp', 'name', 'netbios_dgm', 'netbios_ns', 'netbios_ssn', 'netstat', 'nnsp', 'nntp', 'ntp_u', 'other', 'pm_dump', 'pop_2′, 'pop_3′, 'printer', 'private', 'red_i', 'remote_job', 'rje', 'shell', 'smtp', 'sql_net', 'ssh', 'sunrpc', 'supdup', 'systat', 'telnet', 'tftp_u', 'tim_i', 'time', 'urh_i', 'urp_i', 'uucp', 'uucp_path', 'vmnet', 'whois', 'X11′, 'Z39_50′。
4) **flag** 连接正常或错误的状态，离散类型，共 11 种。'OTH', 'REJ', 'RSTO', 'RSTOS0′, 'RSTR', 'S0′, 'S1′, 'S2′, 'S3′, 'SF', 'SH'。它表示该连接是否按照协议要求开始或完成。例如 SF 表示连接正常建立并终止；S0 表示只接到了 SYN 请求数据包，而没有后面的 SYN/ACK。其中 SF 表示正常，其他 10 种都是 error。
5) **src_bytes** 从源主机到目标主机的数据的字节数，连续类型，范围是 [0, 1379963888]。
6) **dst_bytes** 从目标主机到源主机的数据的字节数，连续类型，范围是 [0. 1309937401]。
7) **land** 若连接来自/送达同一个主机/端口则为 1，否则为 0，离散类型，0 或 1。
8) **wrong_fragment** 错误分段的数量，连续类型，范围是 [0, 3]。
9) **urgent** 加急包的个数，连续类型，范围是[0, 14]。

2.4.2 TCP 连接的内容特征

对于 U2R 和 R2L 之类的攻击，由于它们不像 DoS 攻击那样在数据记录中具有频繁序列模式，而一般都是嵌入在数据包的数据负载里面，单一的数据包和正常连接没有什么区别。为了检测这类攻击，Wenke Lee 等从数据内容里面抽取了部分可能反映入侵行为的内容特征，如登录失败的次数等。

10) **hot** 访问系统敏感文件和目录的次数，连续，范围是 [0, 101]。例如访问系统目录，建立或执行程序等。
11) **num_failed_logins** 登录尝试失败的次数。连续，[0, 5]。
12) **logged_in** 成功登录则为 1，否则为 0，离散，0 或 1。
13) **num_compromised** compromised 条件出现的次数，连续，[0, 7479]。
14) **root_shell** 若获得 root shell 则为 1，否则为 0，离散，0 或 1。root_shell 是指获得超级用户权限。
15) **su_attempted** 若出现"su root" 命令则为 1，否则为 0，离散，0 或 1。
16) **num_root** root 用户访问次数，连续，[0, 7468]。

17) **num_file_creations** 文件创建操作的次数，连续，[0, 100]。
18) **num_shells** 使用 shell 命令的次数，连续，[0, 5]。
19) **num_access_files** 访问控制文件的次数，连续，[0, 9]。例如对 /etc/passwd 或 .rhosts 文件的访问。
20) **num_outbound_cmds** 一个 FTP 会话中出站连接的次数，连续，0。数据集中这一特征出现次数为 0。
21) **is_hot_login** 登录是否属于“hot”列表，是为 1，否则为 0，离散，0 或 1。例如超级用户或管理员登录。
22) **is_guest_login** 若是 guest 登录则为 1，否则为 0，离散，0 或 1。

2.4.3 基于时间的网络流量统计特征

由于网络攻击事件在时间上有很强的关联性，因此统计出当前连接记录与之前一段时间内的连接记录之间存在的某些联系，可以更好的反映连接之间的关系。这类特征又分为两种集合：一个是 “same host”特征，只观察在过去两秒内与当前连接有 相同目标主机 的连接，例如相同的连接数，在这些相同连接与当前连接有相同的服务的连接等等；另一个是 “same service”特征，只观察过去两秒内与当前连接有 相同服务 的连接，例如这样的连接有多少个，其中有多少出现 SYN 错误或者 REJ 错误。

23) **count** 过去两秒内，与当前连接具有相同的目标主机的连接数，连续，[0, 511]。
24) **srv_count** 过去两秒内，与当前连接具有相同服务的连接数，连续，[0, 511]。
25) **serror_rate** 过去两秒内，在与当前连接具有相同目标主机的连接中，出现“SYN”错误的连接的百分比，连续，[0.00, 1.00]。
26) **srv_serror_rate** 过去两秒内，在与当前连接具有相同服务的连接中，出现“SYN”错误的连接的百分比，连续，[0.00, 1.00]。
27) **rerror_rate** 过去两秒内，在与当前连接具有相同目标主机的连接中，出现“REJ”错误的连接的百分比，连续，[0.00, 1.00]。
28) **srv_rerror_rate** 过去两秒内，在与当前连接具有相同服务的连接中，出现“REJ”错误的连接的百分比，连续，[0.00, 1.00]。
29) **same_srv_rate** 过去两秒内，在与当前连接具有相同目标主机的连接中，与当前连接具有相同服务的连接的百分比，连续，[0.00, 1.00]。
30) **diff_srv_rate** 过去两秒内，在与当前连接具有相同目标主机的连接中，与当前连接具有不同服务的连接的百分比，连续，[0.00, 1.00]。
31) **srv_diff_host_rate** 过去两秒内，在与当前连接具有相同服务的连接中，与当前连接具有不同目标主机的连接的百分比，连续，[0.00, 1.00]。

注意：这一大类特征中，23、25、27、29、30 这 5 个特征是“same host”特征，前提都是与当前连接具有相同目标主机的连接；24、26、28、31 这 4 个特征是“same service”特征，前提都是与当前连接具有相同服务的连接。

2.4.4 基于主机的网络流量统计特征

基于时间的流量统计只是在过去两秒的范围内统计与当前连接之间的关系，而在实际入侵中，有些 Probing 攻击使用慢速攻击模式来扫描主机或端口，当它们扫描的频率大于 2 秒的时候，基于时间的统计方法就无法从数据中找到关联。

所以 Wenke Lee 等按照目标主机进行分类，使用一个具有 100 个连接的时间窗，统计当前连接之前 100 个连接记录中与当前连接具有 相同目标主机的统计信息。

32) **dst_host_count** 前 100 个连接中，与当前连接具有相同目标主机的连接数，连续，[0, 255]。
33) **dst_host_srv_count** 前 100 个连接中，与当前连接具有相同目标主机相同服务的连接数，连续，[0, 255]。
34) **dst_host_same_srv_rate** 前 100 个连接中，与当前连接具有相同目标主机相同服务的连接所占的百分比，连续，[0.00, 1.00]。
35) **dst_host_diff_srv_rate** 前 100 个连接中，与当前连接具有相同目标主机不同服务的连接所占的百分比，连续，[0.00, 1.00]。
36) **dst_host_same_src_port_rate** 前 100 个连接中，与当前连接具有相同目标主机相同源端口的连接所占的百分比，连续，[0.00, 1.00]。
37) **dst_host_srv_diff_host_rate** 前 100 个连接中，与当前连接具有相同目标主机相同服务的连接中，与当前连接具有不同源主机的连接所占的百分比，连续，[0.00, 1.00]。
38) **dst_host_serror_rate** 前 100 个连接中，与当前连接具有相同目标主机的连接中，出现 SYN 错误的连接所占的百分比，连续，[0.00, 1.00]。
39) **dst_host_srv_serror_rate** 前 100 个连接中，与当前连接具有相同目标主机相同服务的连接中，出现 SYN 错误的连接所占的百分比，连续，[0.00, 1.00]。
40) **dst_host_rerror_rate** 前 100 个连接中，与当前连接具有相同目标主机的连接中，出现 REJ 错误的连接所占的百分比，连续，[0.00, 1.00]。
41) **dst_host_srv_rerror_rate** 前 100 个连接中，与当前连接具有相同目标主机相同服务的连接中，出现 REJ 错误的连接所占的百分比，连续，[0.00, 1.00]。

2.4.5 样本分析

Wende Lee 等人在处理原始连接数据时将部分重复数据去除，例如进行 DoS 攻击时产生大量相同的连接记录，就只取攻击过程中 5 分钟内的连接记录作为该攻击类型的数据集。同时，也会随机抽取正常(normal)数据连接作为正常数据集。KDD99 数据集总共由 500 万条记录构成，它还提供一个 10%的训练子集和测试子集，它的样本类别分布如下：

- **NORMAL**：正常访问，训练集（10%）有 97278 个样本，测试集（Corrected）有 60593 个样本。
- **PROBE**：端口监视或扫描，训练集（10%）有 4107 个样本，测试集（Corrected）有 4166 个样本。攻击包括： ipsweep、 mscan、nmap、portsweep、saint、satan。
- **DOS**：拒绝服务攻击，训练集（10%）有 391458 个样本，测试集（Corrected）有 229853 个样本。攻击包括： apache2、back、land、mailbomb、neptune、pod、processtable、smurf、teardrop、udpstorm。
- **U2R**：未授权的本地超级用户特权访问，训练集（10%）有 52 个样本，测试集（Corrected）有 228 个样本。攻击包括：buffer_overflow、httptunnel、loadmodule、perl、ps、rootkit、sqlattack、 xterm。
- **R2L**：来自远程主机的未授权访问，训练集（10%）有 1126 个样本，测试集（Corrected）有 16189 个样本。攻击包括：ftp_write、guess_passwd、imap、multihop、named、phf、sendmail、snmpgetattack、snmpguess、spy、warezclient、

warezmaster、worm、xlock、xsnoop。

KDD99 将攻击类型分为 4 类，然后又细分为 39 小类，每一类代表一种攻击类型，类型名被标记在训练数据集每一行记录的最后一项。

某些攻击类型只在测试集（或训练集）中出现，而未在训练集（或测试集）中出现。比如 10%的数据集中，训练集中共出现了 22 个攻击类型，而剩下的 17 种只在测试集中出现，这样设计的目的是检验分类器模型的泛化能力，对未知攻击类型的检测能力是评价入侵检测系统好坏的重要指标。

针对本次实验，综合考虑电脑处理速度和实现难度，主要针对 10%的数据库进行测试。根据上述针对数据文件中解释，我们可以针对每一条数据进行人工的解析和解释。例如，我们从文件中选取 4 条数据，可以结合上述的解析进行观察和解释。

```
0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,
9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1
9,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,
29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,
39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.
```

在上述的随机选取的 4 条数据中，我们可以看出，每一个数据会有对应的 42 个数据，针对前 41 个数据，他们分别是前面所介绍的相关的不同情况下截取的报文数据，通过解析可以将每一条报文内的信息所解析出来。针对最后一项的’normal’是对该条报文进行标签，标签其是正常的或是受到某种攻击的情况。这样对后面的数据挖掘训练是有所帮助的。

3. 数据处理

在进行相关的入侵检测分析之前，我们首先需要将数据转变成机器可以识别的情况，并且为了后期更好的计算与提高结果效率，对数据需要进行预处理。本次实验对数据主要进行字符型转换为数值型、标准化、归一化三部分的预处理。接下来针对每一部分进行阐述。

3.1 字符型转换为数值型

在数据挖掘的过程中，数据的预处理一直都是非常重要的一个环节，只有把数据转化为分类器认可的形式才可以对其进行训练。

所以针对这一部分的数据预处理环节，主要进行的是将本身数据中有的非数字化的数据转变成数值，从而使计算机可以更容易地处理他们。通过对每一种字符型对一一对应的数值，将其转变。从而得到最终处理好的数值型数据，并且将该数据存储到 csv 文件当中，以方便后面的使用。

3.1.1 使用函数库

- numpy: NumPy(Numerical Python) 是 Python 语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库[2]。
- scv:CSV (Comma Separated Vaules) 格式是电子表格和数据库中最常见的输入、输出文件格式[3]。
- pandas:pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。pandas 提供了大量能使我们快速便捷地处理数据的函数和方法[4]。

3.1.2 代码实现

- 文件读取、写入：该部分主要是现在的功能是将文件进行读取，并且写入到新的 csv 文件当中。其中需要注意到的是为了避免在文件中出现多次空行的情况,我们需要 newline = "来解决这种情况。

```
source_file = 'kddcup.testdata.unlabeled_10_percent'
handled_file = 'kddcup.testdata.unlabeled_10_percent.csv'
data_file = open(handled_file, 'w',newline = '')
```

- find_index 函数：将相应的非数字类型转换为数字标识即符号型数据转化为数值型数据，方便后续转变成相对应的数字。

```
def find_index(x, y):
    return [i for i in range(len(y)) if y[i] == x]
```

- handleProtocol 函数：定义将源文件行中 3 种协议类型转换成数字标识的函数；handleService 函数：定义将源文件行中 70 种网络服务类型转换成数字标识的函数；除此以外，另外两种部分函数 handleFlag,handleLabel 也与这两部分函数相类似。对已有的数值相对应的一对一的转换。

[2] 菜鸟教程：https://www.runoob.com/numpy/numpy-tutorial.html
[3] https://docs.python.org/zh-cn/3.7/library/csv.html
[4] 百度百科：https://baike.baidu.com/item/pandas/17209606?fr=aladdin

```
def handleProtocol(inputs):
    protocol_list = ['tcp', 'udp', 'icmp']
    if inputs[1] in protocol_list:
        return find_index(inputs[1], protocol_list)[0]
def handleService(inputs):
    service_list = ['aol', 'auth', 'bgp', 'courier', 'csnet_ns', 'ctf', 'daytime', 'discard',
'domain', 'domain_u','echo', 'eco_i', 'ecr_i', 'efs', 'exec', 'finger', 'ftp', 'ftp_data', 'gopher',
'harvest','hostnames','http', 'http_2784', 'http_443', 'http_8001', 'imap4', 'IRC',
'iso_tsap', 'klogin', 'kshell','ldap','link', 'login', 'mtp', 'name', 'netbios_dgm', 'netbios_ns',
'netbios_ssn', 'netstat', 'nnsp','nntp','ntp_u', 'other', 'pm_dump', 'pop_2', 'pop_3',
'printer', 'private', 'red_i', 'remote_job', 'rje','shell','smtp', 'sql_net', 'ssh', 'sunrpc',
'supdup', 'systat', 'telnet', 'tftp_u', 'tim_i', 'time','urh_i', 'urp_i','uucp', 'uucp_path',
'vmnet', 'whois', 'X11', 'Z39_50']
    if inputs[2] in service_list:
        return find_index(inputs[2], service_list)[0]
```

最后针对已有的数据集文件，进行每一行循环读取并写入新的csv文件当中。在进行写入时使用刚才说明的函数，将其编译成电脑可以读取的形式。接下来进行数据预处理的第二部分：数据标准化。

3.1.3 结果展示

为了可以更好的对比出结果，我们选用之前已选出的部分数据及情况来进行对比。

转换前：

```
0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,
9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1
9,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.
```

转换后：

```
0,0,21,9,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,
1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,0
0,0,21,9,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19
,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,0
```

可以看出转换后的数据集中并没有出现英文字符（字符串）的情况，而是全部呈现数值的结果。并且文件变为.csv 为文件并非纯文本文件，这样可以让计算机更好地进行后面的处理。

3.2 标准化

在一般进行数据挖掘的时候,我们对数据的预处理不能仅仅局限于数值上的

转变。数据的标准化主要是应对特征向量中数据很分散的情况，防止一些小数据被大数据吞并的情况。另外数据标准化也有加速训练防止梯度爆炸的作用。通过对数据标准化，我们可以更好地将数据所可能带来的缺陷进行最小化，从而使我们的结果更加的准确。

数据的标准化是将数据按比例缩放，使之落入一个小的特定区间。在某些比较和评价的指标处理中经常会用到，去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行比较和加权。

3.2.1 Z-Score 标准化

Z-score 标准化主要的公式为:$\frac{x-\mu}{\delta}$,其中 μ 表示的是总体数据的均值；δ 表示的是总体数据的标准差；x 表示的是个体的观测值。

通过这样的数据标准化，我们可以将所有数据变成同一个量程上面的数据，以保证数据的更加合理化以及结果的准确性。

3.2.2 使用函数库

在该环节函数库使用情况与上一环节基本相同。出现过的函数库将不再出现阐述。

- Math:该模块提供了对 C 标准定义的数学函数的访问。但这些函数不适用于复数；如果你需要计算复数，请使用 cmath 模块中的同名函数[5]。

3.2.3 代码实现

- Handle_data 函数：实现对文件中的数据标准化。前半部分是对文件的读取并对求和求平均值，求绝对误差，平均绝对误差等操作的定义。后半部分是进行对文件的循环读取后，并且将相对应的数值进行求和，求平均值，求绝对误差，平均绝对误差。之后将其得到的数据存入到预先设定好的空字典当中，最终生成新的文件。

```
source_file = "kddcup.testdata.unlabeled_10_percent.csv"；
handled_file = "kddcup.testdata.unlabeled_10_percent_result.csv"
data_file = open(handled_file,'w',newline='')
with open(source_file,'r') as data_source:
    csv_reader = csv.reader(data_source)
    count = 0；    row_num = ""
    for row in csv_reader:
        count = count+1；row_num = row
    sum = np.zeros(len(row_num))    #和              sum.astype(float)
    avg = np.zeros(len(row_num)) #平均值
    avg.astype(float)
    stadsum = np.zeros(len(row_num)) #绝对误差
    stadsum.astype(float)
    stad = np.zeros(len(row_num)) #平均绝对误差
    stad.astype(float)
```

[5] https://docs.python.org/zh-cn/3.7/library/math.html

```
    for i in range(0,len(row_num)):
        with open(source_file,'r') as data_source:
            csv_reader = csv.reader(data_source)
            for row in csv_reader:
                sum[i] += float(row[i])
        avg[i] = sum[i] / count       #每一列的平均值求得
        with open(source_file,'r') as data_source:
            csv_reader = csv.reader(data_source)
            for row in csv_reader:
                stadsum[i] += math.pow(abs(float(row[i]) - avg[i]), 2)
        stad[i] = stadsum[i] / count #每一列的平均绝对误差求得
        with open(source_file,'r') as data_source:
            csv_reader = csv.reader(data_source)
            list = []
            for row in csv_reader:
                temp_line=np.array(row)
                if avg[i] == 0 or stad[i] == 0:
                    temp_line[i] = 0
                else:
                    temp_line[i] = abs(float(row[i]) - avg[i]) / stad[i]
                list.append(temp_line[i])
            lists.append(list)
    for j in range(0,len(lists)):
        dic[j] = lists[j] #将每一列的元素值存入字典中
```

3.2.4 结果展示

通过以上函数对文件数据中的标准化处理后，我们可以得到的数据如下。

原始数据：

```
0,1,47,9,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,2
55,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,0
0,1,47,9,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,2
55,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,0
```

标准化后数据：

```
0.00,1.89,0.50,0.16,4.04,9.80,0,0,0,0.07,1.00,1.10,0.08,1.00,0,0.05,0.27,0.60,0.45,0,1.0
0,1.00,0.01,0.01,1.25,1.46,1.00,1.00,2.02,1.23,1.63,0.00,0.00,1.51,5.56,7.35,1.28,3.11,
1.30,1.73,1.47,0.47
0.00,1.89,0.50,0.16,4.04,9.80,0,0,0,0.07,1.00,1.10,0.08,1.00,0,0.05,0.27,0.60,0.45,0,1.0
0,1.00,0.01,0.01,1.25,1.46,1.00,1.00,2.02,1.23,1.63,0.00,0.00,1.51,5.56,7.35,1.28,3.11,
1.30,1.73,1.47,0.47
```

通过对比可以看出一些原本数据很大的数据已经被处理成一个小小的数值，说明通过数据标准化后，所有数据的数值大小均被合理的规划在了一定的范围之

内，这样的结处理情况可以减少数值大的数据对数值较小的数据所产生的影响。

文本经过数值标准化后，接下来将进入到数据预处理的第三步：归一化。

3.3 归一化

数据标准化处理是数据挖掘的一项技术工作。不同评价指标往往具有不同的量纲和量纲单位。这样的情况会影响数据分析的结果，为了消除指标之间的量纲我和你分享完了数据标准化处理后，各指标属于同一量级。此时需要对其进行归一化处理，将其数据均规范到[0,1]范围之内，因为最终的目的与数值的大小并无太多关系，所以尽量要减少数值的大小对其结果的影响，因此可以对数据进行归一化处理，提高最后结果的准确性。这也是数据预处理的最后一步。

3.3.1 Min-max 标准化

min-max 标准化也称作归一化，是对原始数据的线性变换，使结果值映射到[0 , 1]之间。转换函数如下：

$$x' = \frac{x - \min}{\max - \min}$$

其中 max 为样本数据的最大值，min 为样本数据的最小值。

3.3.2 代码实现

该部分代码所使用的函数库与上一部分完全相同，在此不再进行过多的阐述。

- Find_Maxmin 函数：该函数实现功能即为将数据归一化。前半部分的代码是在求寻求数据中同一列别里面的数据的最大值与最小值，并将其保存。后半部分代码主要实现对文本中数据具体的归一化操作。使用到的公式即为 3.3.1 的公式。

```
def Find_Maxmin():
    with open(source_file,'r') as data_source:
        csv_reader=csv.reader(data_source)
        with open(source_file,'r') as data_source:
            csv_reader=csv.reader(data_source)
            final_list = list(csv_reader)
            print(final_list)
            jmax = []
            jmin = []
            for k in range(0, len(final_list)):
                jmax.append(max(final_list[k]))
                jmin.append(min(final_list[k]))
            jjmax = float(max(jmax))
            jjmin = float(min(jmin))
```

```
        for i in range(0,len(row_num)):
            lists = []
            with open(source_file,'r') as data_source:
                csv_reader=csv.reader(data_source)
                for row in csv_reader:
                    if (jjmax-jjmin) == 0:
                        x = 0
                    else:
                        x = (float(row[i])-jjmin) / (jjmax-jjmin)
                    lists.append(x)
            listss.append(lists)
        for j in range(0,len(listss)):
            dic[j] = listss[j]
```

3.3.3 结果展示

接下来针对数据归一化的结果进行展示，可看出归一化之后的数据均在零到一的范围之内，因为数据过多，所以目前只展示一条数据最后归一化的结果。

```
0.0,0.019226856561546286,0.00508646998982706,0.0016276703967446594,0.0410986775
1780265,0.09969481180061039,0.0,0.0,0.0,0.0007121057985757885,0.01017293997965412
,0.011190233977619533,0.0008138351983723297,0.01017293997965412,0.0,0.0005086469
989827061,0.0027466937945066125,0.006103763987792472,0.004577822990844354,0.0,0.
01017293997965412,0.01017293997965412,0.00010172939979654121,0.000101729399796
54121,0.01271617497456765,0.01485249237029501 5,0.01017293997965412,0.0101729399
7965412,0.020549338758901324,0.012512716174974568,0.016581892166836216,0.0,0.0,0.
015361139369277722,0.05656154628687691,0.07477110885045778,0.01302136317395727
5,0.031637843336724314,0.013224821973550356,0.017599186164801627,0.014954221770
091557,0.004781281790437436
```

通过字符型转换为数值型、标准化、归一化，这三次对初步数据的预处理后。数据已经被我们处理为电脑可以进行识别，并且有利于后面操作的数据格式。接下来我们针对这些已经被处理好的数据集进行不同方式的入侵检测测试。

4. 基于 KNN 方式的入侵检测

因为本次实验主要针对的是基于数据挖掘方式的入侵检测系统，所以我选取了部分可以进行数据挖掘方式的算法进行了本次实验。首先，我采取的是 KNN 算法。

4.1 KNN 算法

KNN（K-Nearest Neighbor）最邻近分类算法是数据挖掘分类（classification）技术中最简单的算法之一，其指导思想是”近朱者赤，近墨者黑“，即由你的邻居来推断出你的类别。

KNN 最邻近分类算法的实现原理：为了判断未知样本的类别，以所有已知类别的样本作为参照，计算未知样本与所有已知样本的距离，从中选取与未知样本距离最近的 K 个已知样本，根据少数服从多数的投票法则（majority-voting），将未知样本与 K 个最邻近样本中所属类别占比较多的归为一类。

实际 K 这个字母的含义就是要选取的最邻近样本实例的个数，如下图所示，如何判断绿色圆应该属于哪一类，是属于红色三角形还是属于蓝色四方形？如果 K=3，由于红色三角形所占比例为 2/3，绿色圆将被判定为属于红色三角形那个类，如果 K=5，由于蓝色四方形比例为 3/5，因此绿色圆将被判定为属于蓝色四方形类。

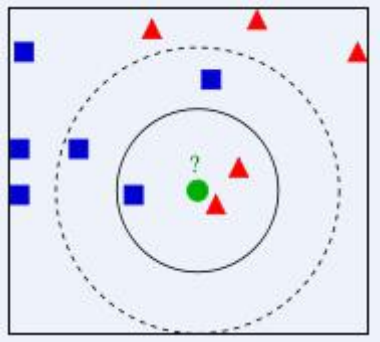


图 4-1：KNN 示例图片

4.2 模型构建

基于 KNN 算法的入侵检测系统主要需要进行的步骤为：加载数据集、划分数据集、KNN 训练、结果展示。

其中加载数据集与划分数据集部分是我们提供给计算机已经构建好的数据集。在进行 KNN 训练时，有一个重要的环节就是计算欧式距离。欧式距离本身指的是两个物体之间的相对距离。但是在本次实验中，主要指的是两个文本之间的差距大小在一种数值上面的反应。欧式距离的计算公式：

$$dist(X,Y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

对于待检测的数据集，首先计算其欧式距离，选取距离较小训练好的数据，将该待检测的数据归类为这些有结果数据中出现情况最多的结果，从而实现利用 KNN 算法进行入侵检测。

在结果展示部分，主要针对该模型的准确率与召回率进行分析。为了可以更直观看到结果的情况，引用 ROC 曲线的绘制。ROC 曲线：接收者操作特征曲线，是反映敏感性和特异性连续变量的综合指标，roc 曲线上每个点反映着对同一信号刺激的感受性。一般来说，ROC 曲线中的横坐标表示：1-Specificity，伪正类率(False positive rate， FPR)，预测为正但实际为负的样本占所有负例样本的比例；纵坐标为：Sensitivity，真正类率(True positive rate， TPR)，预测为正且实际为正的样本占所有正例样本的比例。

ROC 曲线的展现，可以更好的将实验的结果可视化在大家面前。关于 ROC

结果分析将于结果展示部分进行阐述。

4.3 使用函数库

该部分使用了其他函数库已在前面部分阐述。

- Matplotlib：Matplotlib 是 Python 的绘图库。它可与 NumPy 一起使用，提供了一种有效的 MatLab 开源替代方案。它也可以和图形工具包一起使用，如 PyQt 和 wxPython。主要用途是进行相关图形结果的绘制工作[6]。

4.4 代码实现

实现基于 KNN 算法的入侵检测几个主要函数。

- classify 函数：对 KNN 算法进行训练，对待检测的数据集进行欧式距离的计算，并且根据 KNN 算法的要求进行分类。

```
def classify(input_vct, data_set):
    data_set_size = data_set.shape[0]
    diff_mat = np.tile(input_vct, (data_set_size, 1)) - data_set
    sq_diff_mat = diff_mat**2   # 矩阵中每个元素都平方
    distance = sq_diff_mat.sum(axis=1)**0.5   # 每行相加求和并开平方根
    return distance.min(axis=0)   # 返回最小距离
```

- file2mat 函数：将表格存入矩阵，test_filename 为表格路径，para_num 为存入矩阵的列数并返回目标矩阵，和矩阵每一行数据的类别。主要是针对文本文件的转换。

```
def file2mat(test_filename, para_num):
    fr = open(test_filename)
    lines = fr.readlines()
    line_nums = len(lines)
    result_mat = np.zeros((line_nums, para_num))   # 创建 line_nums 行，para_num 列的矩阵
    class_label = []
    for i in range(line_nums):
        line = lines[i].strip()
        item_mat = line.split(',')
        result_mat[i, :] = item_mat[0: para_num]
        class_label.append(item_mat[-1])   # 表格中最后一列正常 1 异常 2 的分类存入 class_label
    fr.close()
    return result_mat, class_label
```

- roc 函数：计算相关 roc 数值，以便于得到最终的 ROC 曲线图。

[6] 菜鸟教程：https://www.runoob.com/numpy/numpy-matplotlib.html

```
def roc(data_set):
    normal = 0
    data_set_size = data_set.shape[1]
    roc_rate = np.zeros((2, data_set_size))
    for i in range(data_set_size):
        if data_set[2][i] == 1:
            normal += 1
    abnormal = data_set_size - normal
    max_dis = data_set[1].max()
    for j in range(1000):
        threshold = max_dis / 1000 * j
        normal1 = 0
        abnormal1 = 0
        for k in range(data_set_size):
            if data_set[1][k] > threshold and data_set[2][k] == 1:
                normal1 += 1
            if data_set[1][k] > threshold and data_set[2][k] == 2:
                abnormal1 += 1
        roc_rate[0][j] = normal1 / normal
        roc_rate[1][j] = abnormal1 / abnormal
    return roc_rate
```

- tes 函数：主函数，并且进行两个图表的绘制。

```
def tes(training_filename, test_filename):
    training_mat, training_label = file2mat(training_filename, 32)
    test_mat, test_label = file2mat(test_filename, 32)
    test_size = test_mat.shape[0]
    result = np.zeros((test_size, 3))
    for i in range(test_size):
        result[i] = i + 1, classify(test_mat[i], training_mat), test_label[i]  # 序号， 最小欧氏距离， 测试集数据类别
    result = np.transpose(result)  # 矩阵转置
    plt.figure(1)
    plt.scatter(result[0], result[1], c=result[2], edgecolors='None', s=1, alpha=1)
    # 图 1 散点图：横轴为序号，纵轴为最小欧氏距离，点中心颜色根据测试集数据类别而定， 点外围无颜色，点大小为最小 1，灰度为最大 1
    roc_rate = roc(result)
    plt.figure(2)#ROC 曲线
    plt.scatter(roc_rate[0], roc_rate[1], edgecolors='None', s=1, alpha=1)
    plt.show()
```

5. 基于 SVM 方式的入侵检测

针对数据挖掘方式，我们第二种算法采用 SVM 算法。

5.1 SVM 算法

SVM 又称为支持向量机，是一种二分类的模型。支持向量机可以分为线性核非线性两大类。其主要思想为找到空间中的一个更够将所有数据样本划开的超平面，并且使得本本集中所有数据到这个超平面的距离最短。

SVM 使用铰链损失函数（hinge loss）计算经验风险（empirical risk）并在求解系统中加入了正则化项以优化结构风险（structural risk），是一个具有稀疏性和稳健性的分类器[7]。SVM 可以通过核方法（kernel method）进行非线性分类，是常见的核学习（kernel learning）方法之一[8]。

5.1.1 线性可分性

在分类问题中给定输入数据和学习目标：$X = \{X_1,X_2,\cdots,X_N\}$，$y = \{y_1,y_2,\cdots,y_N\}$，其中输入数据的每个样本都包含多个特征并由此构成特征空间：$X_i = [x_1,x_2,\cdots,x_n] \in \chi$，而学习目标为二元变量 $y \in \{-1,1\}$，表示负类和正类。

若输入数据所在的特征空间存在作为决策边界的超平面将学习目标按正类和负类分开，并使任意样本的点到平面距离大于等于 1：

$$w^T X + b = 0$$

$$y_i(w^T X_i + b) \geq 1$$

则称该分类问题具有线性可分性，参数 w,b 分别为超平面的法向量和截距。

满足该条件的决策边界实际上构造了 2 个平行的超平面作为间隔边界以判别样本的分类：

$$w^T X_i + b \geq +1 \Rightarrow y_i = +1$$

$$w^T X_i + b \leq -1 \Rightarrow y_i = -1$$

所有在上间隔边界上方的样本属于正类，在下间隔边界下方的样本属于负类。两个间隔边界的距离 $d = \frac{2}{\|w\|}$被定义为边距，位于间隔边界上的正类和负类样本为支持向量。

5.1.2 损失函数

在一个分类问题不具有线性可分性时，使用超平面作为决策边界会带来分类损失，即部分支持向量不再位于间隔边界上，而是进入了间隔边界内部，或落入决策边界的错误一侧。损失函数可以对分类损失进行量化，其按数学意义可以得到的形式是 0-1 损失函数：

$$L(p) = \begin{cases} 0 & p < 0 \\ 1 & p \geq 0 \end{cases}$$

0-1 损失函数不是连续函数，不利于优化问题的求解，因此通常的选择是构造代理损失。可用的选择包括铰链损失函数、logistic 损失函数、和指数损失函数，其中 SVM 使用的是铰链损失函数：

---

[7] 周志华．机器学习．北京：清华大学出版社，2016：pp.121-139, 298-300

[8] 李航．统计学习方法．北京：清华大学出版社，2012：第七章，pp.95-135

$$L(p) = \max(0,1-p)$$

对替代损失的相合性研究表明，当代理损失是连续凸函数，并在任意取值下是 0-1 损失函数的上界，则求解代理损失最小化所得结果也是 0-1 损失最小化的解。铰链损失函数满足上述条件。

5.1.3 经验风险与结构风险

按统计学习理论，分类器在经过学习并应用于新数据时会产生风险，风险的类型可分为经验风险和结构风险：

$$empirical\ risk: \quad \varepsilon = \sum_{i=1}^{N} L(p_i) = \sum_{i=1}^{N} L[f(X_i,w),y_i]$$

$$structural\ risk: \quad \Omega(f) = \|w\|^p$$

式中 f 表示分类器，经验风险由损失函数定义，描述了分类器所给出的分类结果的准确程度；结构风险由分类器参数矩阵的范数定义，描述了分类器自身的复杂程度以及稳定程度，复杂的分类器容易产生过拟合，因此是不稳定的。若一个分类器通过最小化经验风险和结构风险的线性组合以确定其模型参数：

$$L = \|w\|^p + C\sum_{i=1}^{N} L[f(X_i,w),y_i]$$

$$w = argminL$$

则对该分类器的求解是一个正则化问题，常数 C 是正则化系数。当 p=2 时，该式被称为 L_2 正则化或 Tikhonov 正则化。SVM 的结构风险按 p=2 表示，在线性可分问题下，硬边界 SVM 的经验风险可以归 0，因此其是一个完全最小化结构风险的分类器；在线性不可分问题中，软边界 SVM 的经验风险不可归 0，因此其是一个 L_2 正则化分类器，最小化结构风险和经验风险的线性组合。

5.1.4 核方法

由于映射函数具有复杂的形式，难以计算其内积，因此可使用核方法（kernel method），即定义映射函数的内积为核函数（kernel function）。下面列出常用的核函数：

名称	解析式
多项式核（polynomial kernel）	$\kappa(X_1,X_2) = (X_1^T X_2)^n$
径向基函数核（RBF kernel）	$\kappa(X_1,X_2) = \exp\left(-\frac{\|X_1 - X_2\|^2}{2\sigma^2}\right)$
拉普拉斯核（Laplacian kernel）	$\kappa(X_1,X_2) = \exp\left(-\frac{\|X_1 - X_2\|}{\sigma}\right)$
Sigmoid 核（Sigmoid kernel）	$\kappa(X_1,X_2) = \tanh\left[a\left(X_1^T X_2\right) - b\right],\ a,b > 0$

5.2 模型构建

该做模式的情况下，根据可使用的相关函数库，采用对数据集进行二元分类方法。主要采用 svm.svc 函数库：

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision function shape=None,random state=None)
```

上面的代码是对函数库使用的一种方式。在使用该函数库时，可以改变多种参数。在本次实验中主要使用到的一些参数为：

- kernel='rbf' 核函数，默认是 rbf
- C=0.1 SVC 的惩罚参数，默认值是 1.0。C 越大，相当于惩罚松弛变量，希望松弛变量接近 0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C 值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。
- verbose= True 是否允许冗余输出
- decision_function_shape='ovr'
  decision_function_shape 可选择范围：‘ovo’, ‘ovr’ or None
- probability=True probability ：是否采用概率估计

5.3 使用函数库

针对 SVM 算法的使用，主要利用 Python 中相关函数库自带的函数库进行算法的实现。这样可以大大减少相关的代码量。

- Sklearn:sklearn 库共分为 6 大部分，分别用于完成分类任务、回归任务、聚类任务、降维任务、模型选择以及数据的预处理。在该部分主要针对 sklearn 中 svm.svc 模块进行使用,通过这种方式，直接进行相关的数据挖掘以及机器学习等相关工作[9]。
  其余相关函数库已经在上文部分进行过相关的阐述。

5.4 代码实现

- svm_classifier 主函数：使用 svm.svc 函数库进行主要的算法检测部分，并在该部分设定好相关参数。

```
    def svm_classifier(self):
        # Create SVM classification object
        model = svm.SVC(kernel='rbf', C=0.1, verbose= True, decision_function_shape='ovr', probability=True)
        model.fit(self.X_train[:50000], self.y_train[:50000])
        self.predicting(model, 'SVM')
```

- predicting 函数：该部分函数主要是针对预测后结果进行分析总结。通过计算，可以得到在这种方法下所检测到的准确率、召回率、容错率当不同衡量标准的数值结果。可以通过这些数值在后期进行更多的结果分析与总结。

[9] https://www.cnblogs.com/herryzz/p/10243533.html

```
    def predicting(self, model, model_name):
        # Predict
        predicts = model.predict(self.X_test)
        print("Classifier:")
        accuracy = accuracy_score(self.y_test, predicts)
        print("Accuracy: ", accuracy)
        con_matrix = confusion_matrix(self.y_test, predicts, labels=[0, 1])
        # con_matrix = confusion_matrix(y_test, predicts, labels=["normal.",
"abnormal."])
        print("confusion matrix:")
        print(con_matrix)
        precision = con_matrix[0][0] / (con_matrix[0][0] + con_matrix[1][0])
        recall = con_matrix[0][0] / (con_matrix[0][0] + con_matrix[0][1])
        tpr = recall
        fpr = con_matrix[1][0] / (con_matrix[1][0] + con_matrix[1][1])
        print("Precision:", precision)
        print("Recall:", recall)
        print("TPR:", tpr)
        print("FPR:", fpr)
```

6. 基于随机森林算法的入侵检测

6.1 随机森林算法(RandomForest)

随机森林，指的是利用多棵树对样本进行训练并预测的一种分类器。简单来说，随机森林就是由多棵 CART（Classification And Regression Tree）构成的。对于每棵树，它们使用的训练集是从总的训练集中有放回采样出来的，这意味着，总的训练集中的有些样本可能多次出现在一棵树的训练集中，也可能从未出现在一棵树的训练集中。在训练每棵树的节点时，使用的特征是从所有特征中按照一定比例随机地无放回的抽取的。

因此，随机森林的训练过程可以总结如下：

(1)给定训练集 S，测试集 T，特征维数 F。确定参数：使用到的 CART 的数量 t，每棵树的深度 d，每个节点使用到的特征数量 f，终止条件：节点上最少样本数 s，节点上最少的信息增益 m；

◆ **对于第 1-t 棵树，i=1-t：**

(2)从 S 中有放回的抽取大小和 S 一样的训练集 S(i)，作为根节点的样本，从根节点开始训练；

(3)如果当前节点上达到终止条件，则设置当前节点为叶子节点，如果是分类问题，该叶子节点的预测输出为当前节点样本集合中数量最多的那一类 c(j)，概率 p 为 c(j)占当前样本集的比例；如果是回归问题，预测输出为当前节点样本集各个样本值的平均值。然后继续训练其他节点。如果当前节点没有达到终止条件，则从 F 维特征中无放回的随机选取 f 维特征。利用这 f 维特征，寻找分类效

果最好的一维特征 k 及其阈值 th，当前节点上样本第 k 维特征小于 th 的样本被划分到左节点，其余的被划分到右节点。继续训练其他节点。有关分类效果的评判标准在后面会讲。

(4)重复(2)(3)直到所有节点都训练过了或者被标记为叶子节点。

(5)重复(2),(3),(4)直到所有的树都被训练过。

6.2 模型构建

利用随机森林的预测过程主要是：

◆ **对于第 1-t 棵树，i=1-t：**

(1)从当前树的根节点开始，根据当前节点的阈值 th，判断是进入左节点(<th)还是进入右节点(>=th)，直到到达，某个叶子节点，并输出预测值；

(2)重复执行(1)直到所有 t 棵树都输出了预测值。如果是分类问题，则输出为所有树中预测概率总和最大的那一个类，即对每个 c(j)的 p 进行累计；如果是回归问题，则输出为所有树的输出的平均值。

6.3 使用函数库

- sklearn.ensemble.RandomForestClassifier[10]：针对随机森林相关的函数库。

  其余相关函数库已在前面提到。

```
sklearn.ensemble.RandomForestClassifier(n_estimator=10,criterion='gini',max_depth=Non
e,bootstrap=True,random_state=None,min_sample_lit=2)
```

该函数库的相关参数如上：max_features="auto"：默认为“auto”，每个决策树的最大特征数量，即为 m 值得选取方法。if"auto",求平方根；if"sqrt"，求平方根；if"log2"，求 log2（）；if None,使用 M 值。在本次实验中，并没有明确针对某种参数进行改变，均使用其默认情况。

6.4 代码实现

- random_forest_classifier 主函数：将已有的数据系带入到已知的函数库当中，进行随机森林算法的检测。（这里类别主要分两类）

```
def random_forest_classifier(self):
    model = ensemble.RandomForestClassifier()
    model.fit(self.X_train, self.y_train)
    self.predicting(model, "RFC")
```

- predicting 函数：该部分函数于上一环节函数一致，均是针对采用这种方法进行入侵检测后，为了可以更好的分析结果，所计算出的准确率、召回率等参数值。再次并不做过多的阐述。

[10] 官网介绍：
https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

7. 基于其他方法的入侵检测

通过对以上三种不同方法的入侵检测。我们可以看到，在 Python 的环境下，有很多相关的数据挖掘函数库可以供人们使用。因此我还采用了别的函数库进行入侵检测。将结果与其他三种方式进行比较，可以更好的看出哪一种方法最终得到的结果最好。

我还尝试的方法还有：决策树算法和 bagging 算法。

7.1 决策树算法

一棵决策树的生成过程主要分为以下 3 个部分:

① 特征选择：特征选择是指从训练数据中众多的特征中选择一个特征作为当前节点的分裂标准，如何选择特征有着很多不同量化评估标准标准，从而衍生出不同的决策树算法。

② 决策树生成： 根据选择的特征评估标准，从上至下递归地生成子节点，直到数据集不可分则停止决策树停止生长。 树结构来说，递归结构是最容易理解的方式。

③ 剪枝：决策树容易过拟合，一般来需要剪枝，缩小树结构规模、缓解过拟合。剪枝技术有预剪枝和后剪枝两种。

7.1.1 特征挑选方法（信息增益法）

选择具有最高信息增益的特征作为测试特征，利用该特征对节点样本进行划分子集，会使得各子集中不同类别样本的混合程度最低，在各子集中对样本划分所需的信息（熵）最少。

7.1.2 决策树的生成

① 从根节点出发，根节点包括所有的训练样本。

② 一个节点（包括根节点），若节点内所有样本均属于同一类别，那么将该节点就成为叶节点，并将该节点标记为样本个数最多的类别。

③ 否则利用采用信息增益法来选择用于对样本进行划分的特征，该特征即为测试特征，特征的每一个值都对应着从该节点产生的一个分支及被划分的一个子集。在决策树中，所有的特征均为符号值，即离散值。如果某个特征的值为连续值，那么需要先将其离散化。

④ 递归上述划分子集及产生叶节点的过程，这样每一个子集都会产生一个决策（子）树，直到所有节点变成叶节点。

◆ 递归操作的停止条件是：

（1）一个节点中所有的样本均为同一类别，那么产生叶节点

（2）没有特征可以用来对该节点样本进行划分，这里用 attribute_list=null 为表示。此时也强制产生叶节点，该节点的类别为样本个数最多的类别

（3）没有样本能满足剩余特征的取值，即 test_attribute= 对应的样本为空。此时也强制产生叶节点，该节点的类别为样本个数最多的类别。

7.1.3 决策树剪枝

由于噪声等因素的影响，会使得样本某些特征的取值与样本自身的类别不相匹配的情况，基于这些数据生成的决策树的某些枝叶会产生一些错误；尤其是在决策树靠近枝叶的末端，由于样本变少，这种无关因素的干扰就会突显出来；由此产生的决策树可能存在过拟合的现象。树枝修剪就是通过统计学的方法删除不可靠的分支，使得整个决策树的分类速度和分类精度得到提高。

7.2 bagging 算法

Bagging 属于一种集成算法，与随机森林算法相类似。集成算法就是通过构建多个学习器来完成学习任务，是由多个基学习器或者是个体学习器来完成的。它可以是由决策树，神经网络等多种基学习算法组成。就像是投票表决答案一样，多数人的参与总会比一个人的观点更加准确。集成学习通过多个学习器进行结合，可以获得比单一学习器显著优越的泛化性能。而且集成学习一般都是以弱学习器集成来得到一个强的学习器获得更好地性能。

对数据进行自助采样法，对结果进行简单投票法。对于给定的包含 m 个样本的数据集，我们随机选择一个样本放入采样集中，再把该样本放回初始数据集，使得下次采样仍有可能被选中。我们这样选择的样本有的在采样集里面重复出现，有的则从未出现。我们分类任务使用简单投票法；对分类任务使用简单平均法；若分类投票出现相同的票数情况，则随机选择一个。

Bagging 算法是一种很高效的一种算法，但是也具有一定的局限性，他不能经修改的适用于多分类和回归等任务。

7.3 使用函数库

在使用这两种算法时候，我们主要采用了是调用 Python 自带的函数库。相关函数库的具体细节以及参数如下：

- ensemble.BaggingClassifier[11]：在 scikit-learn 中，bagging 方法使用统一的 BaggingClassifier 元估计器。可以实现 bagging 算法，在该次实验中均使用了默认的设置情况。

```
sklearn.ensemble.BaggingClassifier(base_estimator=None,n_estimators=10,
max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False,
oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0)
```

- base_estimator：对象或无，可选（默认=无）。基本估计量适合数据集的随机子集。如果为 None，则基本估计量为决策树；
- n_estimators : int，可选（默认值为 10）。集合中的基本估计量；
- max_samples : int 或 float，可选（默认值= 1.0）。从 X 抽取以训练每个基本估计量的样本数。如果为 int，则抽取样本 max_samples；如果 float，则抽取本 max_samples * X.shape[0]；

[11] 官网介绍：
https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.BaggingClassifier.html#sklearn.ensemble.BaggingClassifier

- max_features : int 或 float，可选（默认值= 1.0）。从 X 绘制以训练每个基本估计量的要素数量。如果为 int，则绘制特征 max_features。如果是浮动的，则绘制特征 max_features * X.shape[1]；
- bootstrap : 布尔值，可选（默认= True）。是否抽取样本进行替换。如果为 False，则执行不替换的采样；
- oob_score : 布尔变量，可选（默认为 False）。是否使用现成的样本来估计泛化误差；
- warm_start : 布尔变量，可选（默认= False）。设置为 True 时，请重用上一个调用的解决方案以适合并为集合添加更多估计量，否则，仅适合一个全新的集合；
- n_jobs : int 或 None（可选）（默认为 None）。fit 和 并行运行的作业数 predict。None 除非 joblib.parallel_backend 上下文中，否则表示 1 。-1 表示使用所有处理器；
- random_state : 整型，RandomState 实例或无，可选（默认值：无）。如果为 int，则 random_state 是随机数生成器使用的种子；否则为 false；

- tree.DecisionTreeClassifier[12]：实现决策树算法的相关函数，其相关参数在下方已显示，针对本次实验，主要选择 criterion="entropy"，其余参数选项于本身默认参数设置一致。

```
sklearn.tree.DecisionTreeClassifier(criterion='gini',splitter='best',max_depth=None,
in_samples_split=2,min_samples_leaf =1, min_weight_fraction_leaf=0.0, max_features=None,
random_state=None, max_leaf_nodes=None,class_weight=None, presort=False)
```

- criterion:string 类型：可选（默认为"gini"）：衡量分类的质量。支持的标准有"gini"代表的是 Gini impurity(不纯度)与"entropy"代表的是 information gain（信息增益）；
- splitter:string 类型，可选（默认为"best"）：一种用来在节点中选择分类的策略。支持的策略有"best"，选择最好的分类，"random"选择最好的随机分类；
- max_features:int,float,string or None 可选（默认为 None）：在进行分类时需要考虑的特征数；
- max_depth:int or None,可选（默认为"None"）：表示树的最大深度。如果是"None",则节点会一直扩展直到所有的叶子都是纯的或者所有的叶子节点都包含少于 min_samples_split 个样本点。忽视 max_leaf_nodes 是不是为 None；
- min_samples_split:int,float,可选（默认为 2）：区分一个内部节点需要的最少的样本数；
- min_samples_leaf:int,float,可选（默认为 1）：一个叶节点所需要的最小样本数；
- min_weight_fraction_leaf:float,可选（默认为 0）：一个叶节点的输入样本所需要的最小的加权分数；
- max_leaf_nodes:int,None 可选（默认为 None）：在最优方法中使用 max_leaf_nodes 构建一个树。最好的节点是在杂质相对减少。如果是 None 则对叶节点的数目没有限制。如果不是 None 则不考虑 max_depth；

[12] 官网介绍：
https://scikit-learn.org/dev/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

- class_weight:dict,list of dicts,"Banlanced" or None,可选（默认为 None）：表示在表{class_label:weight}中的类的关联权值。如果没有指定，所有类的权值都为 1。对于多输出问题，一列字典的顺序可以与一列 y 的次序相同；
- random_state:int,RandomState instance or None：如果是 int,random_state 是随机数字发生器的种子；如果是 RandomState，random_state 是随机数字发生器，如果是 None，随机数字发生器是 np.random 使用的 RandomState instance；
- persort:bool,可选（默认为 False）：是否预分类数据以加速训练时最好分类的查找。在有大数据集的决策树中，如果设为 true 可能会减慢训练的过程。当使用一个小数据集或者一个深度受限的决策树中，可以减速训练的过程。

8. 结果展示

我们分析针对检测结果进行分析时，主要采用准确率、精确率、召回率三个衡量准则进行分析。接下来针对结果分析时相关衡量标准解释说明。

在实验过程中我们主要对数据集进行分类，也就是说我们会针对该行数据是否为正常数据进行分析，得到正常与异常两种不同的结果。针对这两种不同的结果，我们需要一些衡量的标准，可以这怒地我们的检测结果进行分析。

8.1 度量指标

在进行检测过程时，针对原本的数据情况以及我们检测出来的数据情况，均有两种不同的情况产生。也就是正常的与不正常的情况，针对这两种情况以及检测出来的结果，他们有着不同的定义。主要会形成四种情况，如下所示：

1. 真阳性(True Positive，TP)：检测非正常数据，且实际非正常；正确肯定的匹配数目；
2. 假阳性(False Positive，FP)：检测非正常数据，但实际正常；误报，给出的匹配是不正确的；
3. 真阴性(True Negative，TN)：检测正常数据，且实际正常；正确拒绝的非匹配数目；
4. 假阴性(False Negative，FN)：检测正常数据，但实际数据非正常；漏报，没有正确找到的匹配的数目。

在已有的四种基础上我们可以得到：准确率、召回率、误判率等其余衡量标准的结果。其中几个比较重要的参数指标为：准确率、召回率、真阴性率、假阴性率、假阳性率，这几个参数指标计算方法如下：

- $正确率 = \frac{预测正确的个数}{总体样本数量}$
- $准确率 = \frac{TP}{TP+FP} = \frac{判断为入侵检测且正确的个数}{判断为断为入侵检测的}$

- 召回率$\text{TPR} = \frac{\text{TP}}{\text{TP}+\text{FN}} = \frac{\text{判断为断为入侵且正确的个数}}{\text{实际入侵检侵检测的}}$
- 真阴性率$TNR = \frac{TN}{FP+TN}$
- 假阴性率$FNR = 1\text{-}TPR$
- 假阳性率$FPR = 1-TNR$

为了可以更加清楚地表示这几种参数指标的形成，我们绘制表格将其表示出来：

	实际为不正常	实际为正常	合计	
预测为不正常	TP	FP	TP+FP	**正确率** =TP/TP+FP
预测为正常	FN	TN	FN+TN	
合计	TP+FN	FP+TN	N=TP+FN+FP+TN	
	召回率 =TP/TP+FN	**假阳性率** =FP/FP+TN		
	假阴性率 =FN/TP+FN	**真阴性率** =TN/FP+TN		

表 8-1：结果分类解释

8.2 ROC 曲线

在上方已有一些基础的度量标准的情况下，我们可以使用 ROC 曲线，更加的直观看出最后检测的结果是否良好。

- 横坐标：1-Specificity，伪正类率(False positive rate，FPR)，预测为正但实际为负的样本占所有负例样本的比例；
- 纵坐标：Sensitivity，真正类率(True positive rate，TPR)，预测为正且实际为正的样本占所有正例样本的比例。

8.2.1 具体分析

一般的 ROC 曲线的横纵坐标定义如上，在一个二分类模型中，假设采用逻辑回归分类器，其给出针对每个实例为正类的概率，那么通过设定一个阈值如 0.6，概率大于等于 0.6 的为正类，小于 0.6 的为负类。对应的就可以算出一组(FPR,TPR)，在平面中得到对应坐标点。随着阈值的逐渐减小，越来越多的实例被划分为正类，但是这些正类中同样也掺杂着真正的负实例，即 TPR 和 FPR 会同时增大。阈值最大时，对应坐标点为(0,0)，阈值最小时，对应坐标点(1,1)。

理想情况下，TPR 应该接近 1，FPR 应该接近 0。ROC 曲线上的每一个点对应于一个 threshold，对于一个分类器，每个 threshold 下会有一个 TPR 和 FPR。比如 Threshold 最大时，TP=FP=0，对应于原点；Threshold 最小时，TN=FN=1，对应于右上角的点(1,1)。

P 和 N 得分不作为特征间距离 d 的一个函数，随着阈值 theta 增加，TP 和 FP 都增加。所以针对 ROC 曲线的定义来看，相对于的纵轴横轴实际的表示不下。

如下，所以我们也可以得知相关的理想目标为：曲线的凸点靠近（0,1）点。

横轴 FPR：1-TNR，1-Specificity，FPR 越大，预测正类中实际负类越多。

纵轴 TPR：Sensitivity(正类覆盖率)，TPR 越大，预测正类中实际正类越多。

理想目标：TPR=1，FPR=0，即图中(0,1)点，故 ROC 曲线越靠拢(0,1)点，越偏离 45 度对角线越好，Sensitivity、Specificity 越大效果越好。

8.3 结果展示与分析

本次实验主要使用了五种不同的方法针对已有的数据集进行入侵检测。接下来针对每一种方法进行结果的展示。其中 KNN 算法结果使用了 ROC 曲线来进行展示，将结果可视化可以更加的看出其算法的检测结果。其余算法均是计算了相关的准确率与召回率。并且记录每种算法在运行时的大致时间，因为该数据集比较庞大，所以不论使用某一种算法来进行入侵检测时，相对应的检测时间均是可以测量的，所以运行的时间也可以当做一种衡量标准对该算法进行分析。

不论是用五种算法的某一种算法，均是已经使用了数据预处理后的数据进行检测。

接下来是五种算法详细的结果展示。

8.3.1 基于 KNN 算法的入侵检测结果展示与分析

针种方法，我们主要绘制了两个图像。其中一个是欧式距离的结果呈现，另一个则是 ROC 曲线图。在欧式距离的呈现情况上看，颜色的不同主要表示了正常数据以及入侵数据两种数据，根据欧式距离的大小，它们被分为不同的类别。

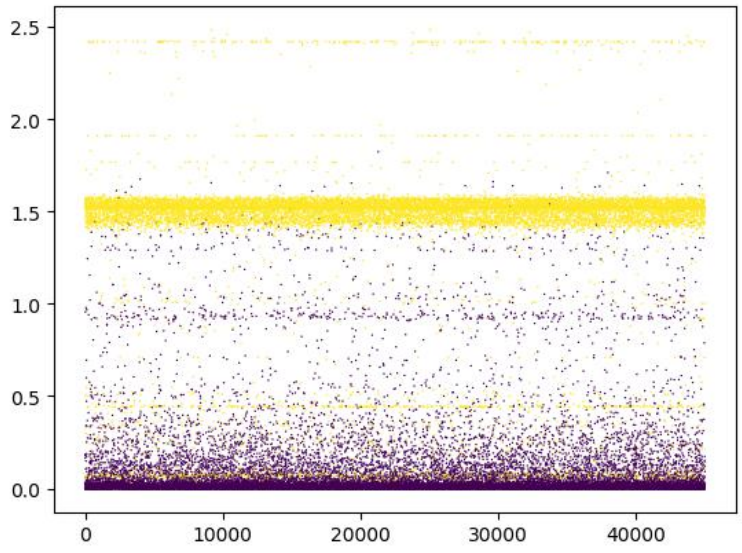


图 8-1：KNN 算法下的欧式距离呈现

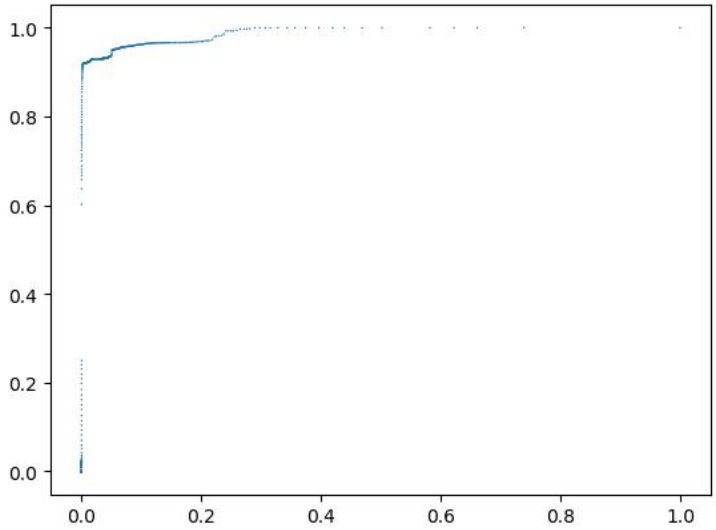


图 8-2：KNN 算法下的 ROC 曲线呈现

观察 KNN 算法下 ROC 曲线,可以发它的曲线凸点更加趋近于(0,1)。根据上述对 ROC 曲线的分析，我们可以得知，该结果可以表明 KNN 算法对入侵检测的结果是很好的。但是同时，根据多次实验的结果来看，在使用该算法情况下，其运行入侵检测的时间是大约半个小时左右。在整体这几种算法中算是比较长的时间。

8.3.2 基于 SVM 算法的入侵检测结果展示及分析

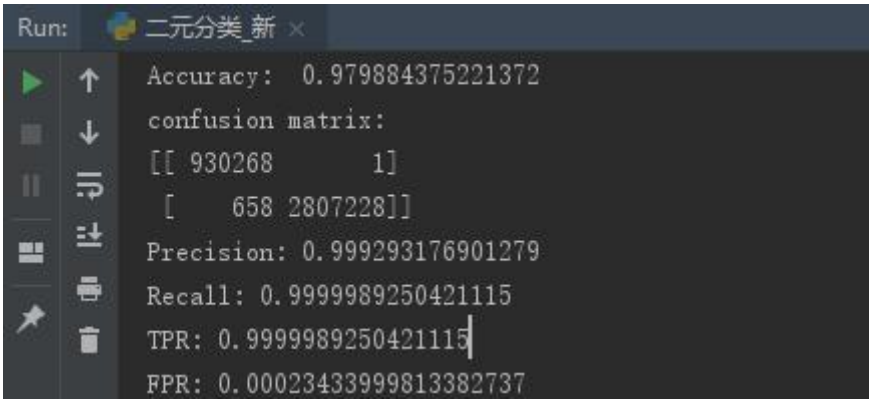


图 8-3：基于 SVM 算法下的检测结果

可以看出该算法情况下入侵检测，其准确率并未达到 99%的情况，但是准确率，召回率均已达到 99%。说明该种算法仍然是一种比较好的算法对于入侵检测而言。该种算法在时间上的消耗与 KNN 算法时间上消耗大致相同。

除此之外，我们针对这种算法进行了三元分类的测试，增加了 smurf 类，其结果如下：

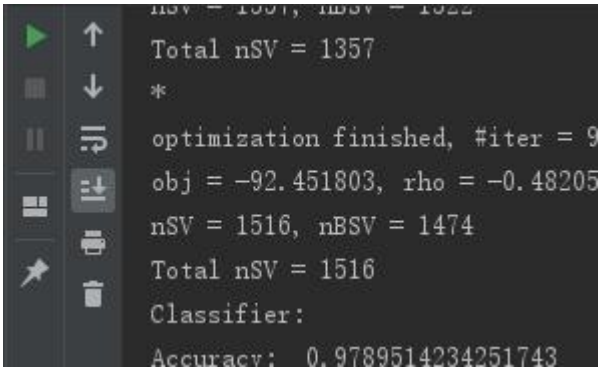


图 8-4：基于 SVM 算法下三元分类的检测结果

8.3.3 基于随机森林算法的入侵检测结果展示与分析

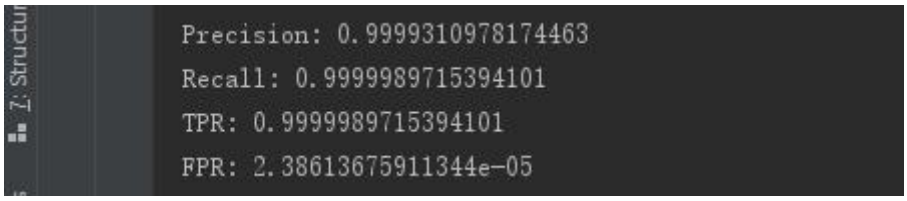


图 8-5：基于随机森林算法的入侵检测

该种算法的入侵检测情况，整体来看是很好的。他在，精确率以及召回率上均有着很高的结果。而且在运行时间上，也远远短于刚才使用过的两种算法。整体来看，目前随机森林算法是一种比较良好的入侵检测算法。

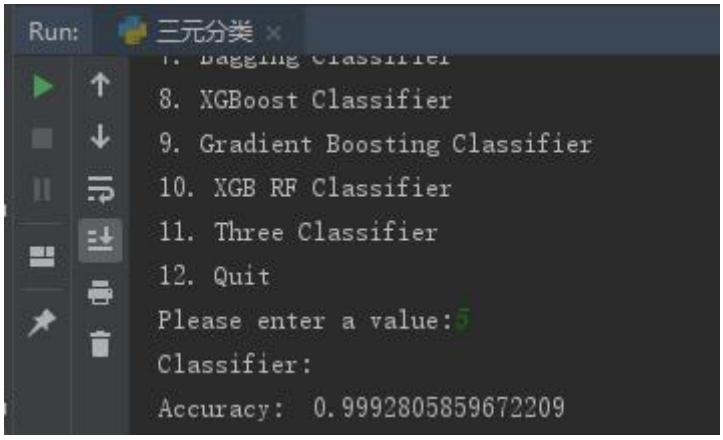


图 8-6：基于随机森林算法的入侵检测 三元分类结果展示

8.3.4 基于 bagging 算法的入侵结果展示与分析

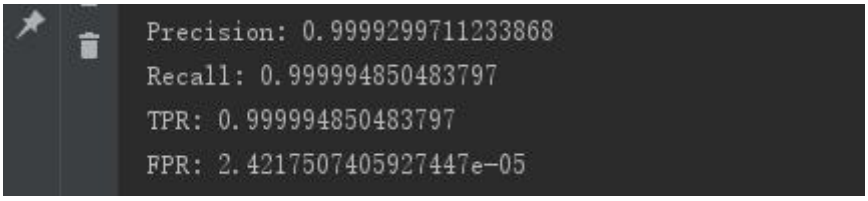


图 8-7：基于 bagging 算法的入侵检测

该种算法的结果情况也算比较好的，其准确率精确率以及召回率均在 99%

以上。并且通过多次实验可以得知其计算时的时间消耗也并不是特别的长,且在时间消耗方面主要是在计算其精确率、召回率的时候时间较长。

8.3.5 基于决策树算法的入侵结果展示与分析

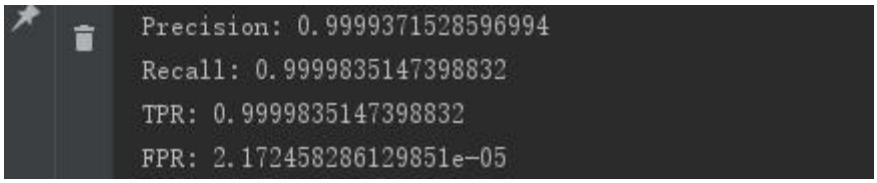


图 8-8：基于决策树算法的入侵检测

基于决策树算法的入侵检测结果，整体来看其准确率以及召回率均在 99%以上。且运行的时也并没有 KNN 以及 SVM 算法的时间长，且在时间消耗方面主要是在计算其精确率、召回率的时候时间较长。

8.4 结果分析

通过以上五种不同的算法对数据集的入侵检测，我们可以进行结果分析。因为我们主要的目的是区分正常数据以及入侵数据这两种大的部分。分类并没有很仔细。目前来看最终的效果也都大都是很好的。所有的算法在二分的情况下均可以达到 95%以上的准确率。但是召回率的大小与我们所测试的数据集很有关系，如果使用以往已经标注好标签的数据集进行检测分析，所得到的结果的召回率是很高的，如果使用未标明标签，并且是用来测试的数据集进行入侵检测。那么此时的召回率会大打折扣，其相对应的准确率也会下降。我认为这可能在使用算法进行入侵检测时，具体的使用情况所受的影响。

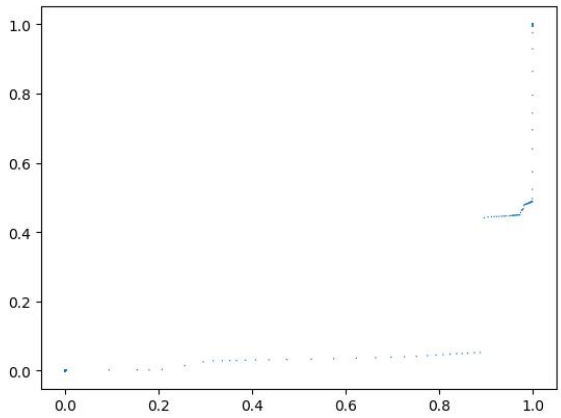


图 8-9：基于 KNN 算法的入侵检测 失败案例结果呈现

例如上图就是其中一次使用测集并未非常正确的情况下得到的 ROC 曲线图。可以看出其结果是非常不理想的。除此之外，我认为对数据集的预处理也十分重要。一开始我进行了对数据预处理的第一步之后即立刻使用了 KNN 算法进行入侵检测。但是其结果并不理想：

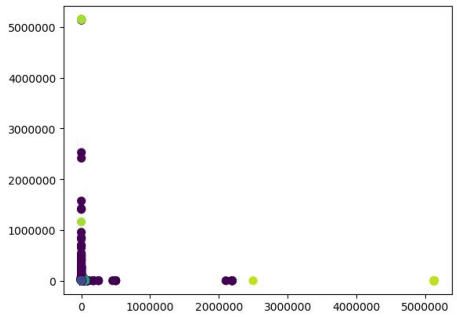


图 8-10：使用 KNN 算法的失败情况

因为并没有对数据集进行归一化以及标准化处理，导致其相对应的结果受到了数据的大小的影响。然而实际情况下，数据本身的大小并没有对最终的结果会有相关的影响

除此之外，不同的算法所得到入侵检测的结果所用的时间也不尽相同。如果使用 KNN 或者 SVM 算法，他们在得到最后的准确率以及召回率等数据消耗的时间较长。相反，如果使用决策树算法或是随机森林算法做得到相对应的结果时间会较短。

在本次的实验过程中，我发现因为数据集的过于庞大，会导致不论算法去进行入侵检测，他都会去消耗很多的时间先进行学习，而后进行相对应的对测试集的数据进行检测。因为电脑的性能，本次实验主要选择 10%的数据集进行实验。所以可以大，如果使用原本的数据集还得数据可庞大到 490 万条数据左右。如此庞大的数据集，在一方面可以增加整体的入侵检测准确率，但是同时在电脑性能方面也有更高在一开始进行预处理的时候，使用电脑时也会经常出现暂时性的内存过量情况，并且在处理已有的数据时所消耗的时间很长的，并不能在很短的时间内所得到结果。这可能与所使用算法时的具体情况有关系，使得最终的结果并不特别理想。

最后除了本次实验所要求的两分类，我也尝试了对数据进行三分类，并且进行入侵检测。在使用同样的训练集以及测试集的情况下，三分类后的结果，相对应的正确率是有所下降的，而且因为分类的增多，使得所消耗的时间也有增加。

9. 总结与展望

9.1 总结

本次入侵检测实验，采用了 KDD CUP99 的数据集进行学习与入侵检测。实验的要求主要是能辨别正常行为以及入侵行为。通过五种不同的机器学习方法，对已有的数据集进行学习和测试。我们得到了一些数据，可以看出这五种方法是可以将大多数的入侵行为所检测出来，并且相对应的准确率以及召回率较高。

但是本次实验仍然存在一定的不足。因为电脑的性能问题，我们只选取了 10%的数据进行学习与测试。这可能导致最终结果准确率会偏高。除此之外，可以发现针对以前学训练集中未出现的入侵情况，在后期的测试中比较难去辨别出来。除 KNN 算法以外的其余算法，因为对函数库使用掌握并不是很熟练，最终

并没有画出所对应的 ROC 曲线。

本次实验，除了二分类情况，我也尝试了进行了三分类的测试。可以粗略的看出当分类增加时，会使用更多的时间进行学习，并且去入侵检测，而最后的结果也会随着分类增加有所降低。

9.2 展望

通过本次实验，我认为最大的时间上的不足，以及在针对不同的训练集进行训练后所得到的有些结果不是很理想。首先，随着分类的越来越详细，也应当提高所相对应检测的准确率。除此之外，为了尽快减少时间上的消耗，可以将多种方法进行融合，取长补短。从而提高入侵检测的效率。