**信 息 内 容 安 全 实 验 报 告**

|  |  |
| --- | --- |
| 实验项目名称： | 入侵检测 |
| 班级： | 08021602 |
| 学号： | 2016301782 |
| 姓名： | 闫增辉 |
| 指导教师： | 杨黎斌 |
| 实验时间： | 2020年4月14日 |

**入侵检测**

**一、实验要求**

给定的cup99数据集,要求实现至少应用2-3种机器学习算法进行甄别正常行为和入侵行为。请参考给出的文献,进行数据预处理，检测模型构建，入侵检测等操作，并设定合理的性能指标验证实验结果，对所采用的检测算法进行较为全面的性能分析。

本实验为课程设计，要求每个同学从阅读文献，编码实现，实验设计，撰写文档均独立完成。

**二、实验原理**

基于数据挖掘技术的入侵检测系统的原理一般是：通过分析历史数据等可以提取出用户的行为特征、总结入侵行为的规律，从而建立起比较完备的规则库来进行入侵检测。

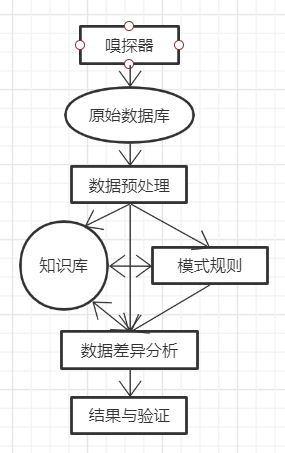
该过程主要分为以下几步：

（1）数据收集。

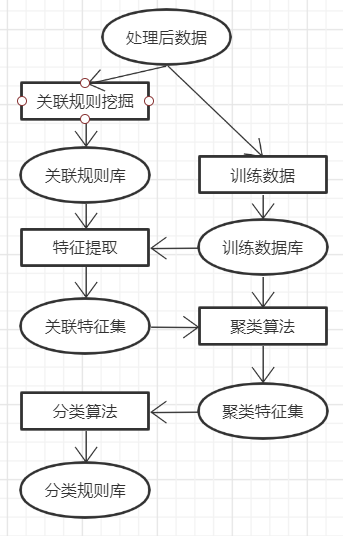
（2）数据的预处理：在数据挖掘中训练数据的好坏直接影响到提取的用户特征和推导出的规则的准确性。用于训练的数据必须不包含任何入侵，否则对此入侵不会有任何反应而漏报，且要格式化成数据挖掘算法可以处理的形式。

（3）数据挖掘：从预处理过的数据中提取行为特征或规则等，再对所得的规则进行归总更新，建立起规则知识库。

（4）入侵检测：依据规则知识库的规则对当前用户的行为进行检测，报告结果并验证，从而采取相应的对付手段。



图一 一个基于数据挖掘的入侵检测模型



图二 模式规则形成流程图

**三、背景阐述**

将数据挖掘技术引入到入侵检测的关键在于，根据从审计数据中观察到的用户行为中提取感兴趣、隐含、潜在的知识，从而判断异常检测行为。目前已有许多研究者将KDD(Knowledge Discovery in Database )算法引入IDS的设计中，主要涉及关联规则，神经网络，聚类分析、遗传算法、免疫算法在入侵检测技术中的应用。

**四、数据集介绍**

KDD Cup 1999数据集是与KDD-99第五届知识发现和数据挖掘国际会议同时举行的第三届国际知识发现和数据挖掘工具竞赛使用的数据集。

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。

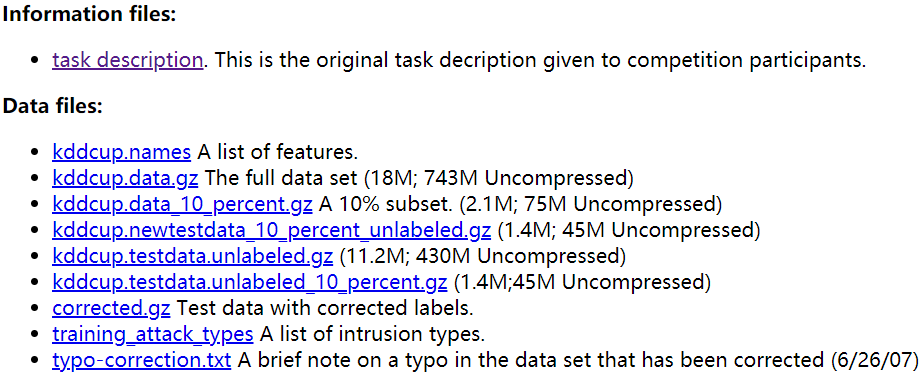
该数据集来源于MIT林肯实验室调查和评估入侵检测方面的研究。1998年美国国防部高级规划署（DARPA）在MIT林肯实验室进行了一项入侵检测评估项目。林肯实验室建立了模拟美国空军局域网的一个网络环境，收集了9周时间的TCP网络连接和系统审计数据，仿真各种用户类型、各种不同的网络流量和攻击手段，使它就像一个真实的网络环境。这些采集的原始数据被分为两个部分：7周时间的训练数据，大概包含5,000,000多个网络连接记录，剩下的2周时间的测试数据大概包含2,000,000个网络连接记录。

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。

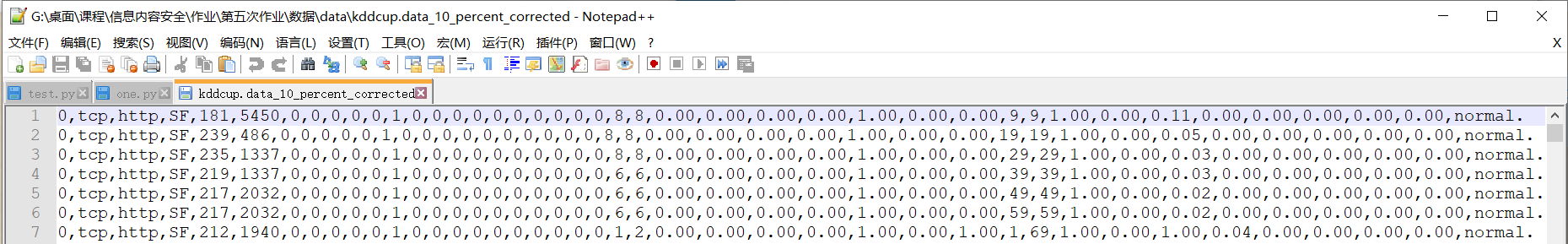
|  |  |  |
| --- | --- | --- |
| 分类 | 全称 | 包括（仅训练集） |
| DOS | 拒绝服务攻击  （denial-of-service） | apache2、back、land、mailbomb、neptune、pod、processtable、smurf、teardrop、udpstorm |
| R2L | 来自远程主机的未授权访问  （unauthorized access from a remote machine to a local machine） | ftp\_write、guess\_passwd、imap、multihop、named、phf、sendmail、snmpgetattack、snmpguess、spy、warezclient、warezmaster、worm、xlock、xsnoop |
| U2R | 未授权的本地超级用户特权访问  （unauthorized access to local superuser privileges by a local unpivileged user） | buffer\_overflow、httptunnel、loadmodule、perl、ps、rootkit、sqlattack、 xterm |
| probing | 端口监视或扫描（surveillance and probing） | ipsweep、 mscan、nmap、portsweep、saint、satan |

表一 异常类型

数据集中包括多个数据文件，为简单起见，本文选择了kddcup.data\_10\_percent.gz，即只有10%数据的数据集。



图三 数据集



图四 数据文件（部分）

上图展现了数据文件中部分样本，每条样本总共有42项特征，最后一列是标记特征，其他前41项特征共分为四大类。

|  |  |  |
| --- | --- | --- |
| 序号 | 特征 | 说明 |
| 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]。 |

表二 TCP连接基本特征

|  |  |  |
| --- | --- | --- |
| 序号 | 特征 | 说明 |
| 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。 |
| 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]。 |
| 20 | num\_outbound\_cmds | 一个FTP会话中出站连接的次数，连续，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。 |
| 22 | is\_guest\_login | 若是guest 登录则为1，否则为0，离散，0或1。 |

表三 TCP连接的内容特征（共13种，序号10～22）

|  |  |  |
| --- | --- | --- |
| 序号 | 特征 | 说明 |
| 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]。 |

表四 基于时间的网络流量统计特征

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。

|  |  |  |
| --- | --- | --- |
| 序号 | 特征 | 说明 |
| 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]。 |

表五 基于主机的网络流量统计特征

**五、系统设计思路之KNN**

1. 库

numpy, pandas, sklearn, matplotlib

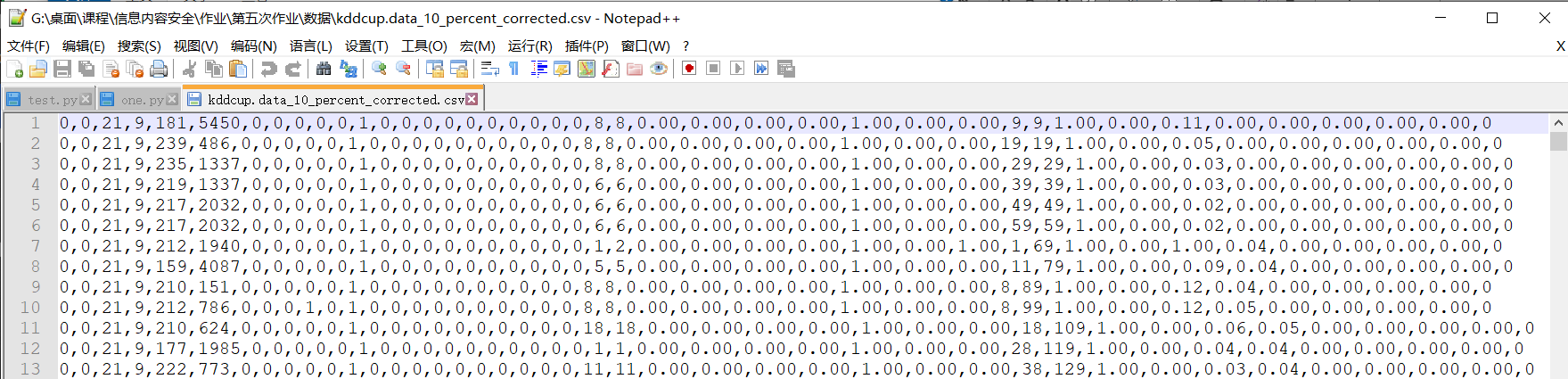
2.数据预处理

由图四可以看到，第2，3，4，42列数据为字符型数据，所以需要先将字符型数据转换为数值型数据。

（1）字符型转换为数值型

转换方法：将同一列中所有可能出现的字符保存到同一数组中，将它们的索引值作为转换后的数值。

|  |
| --- |
| **# -\*- coding: utf-8 -\*-**  **import numpy as np**  **import pandas as pd**  **import csv**  **global label\_list**  **# 符号型数据转化为数值型数据**  **def find\_index(x,y):**  **return [i for i in range(len(y)) if y[i]==x]**  **# 协议类型转换成数字**  **def handleProtocol(inputs):**  **protocol\_list=['tcp','udp','icmp']**  **if inputs[1] in protocol\_list:**  **return find\_index(inputs[1], protocol\_list)[0]**  **# 网络服务类型转换成数字**  **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]**  **# 网络连接状态转换成数字**  **def handleFlag(inputs):**  **flag\_list=['OTH','REJ','RSTO','RSTOS0','RSTR','S0','S1','S2','S3','SF','SH']**  **if inputs[3] in flag\_list:**  **return find\_index(inputs[3],flag\_list)[0]**  **# 攻击类型转换成数字**  **def handleLabel(inputs):**  **label\_list=['normal.', 'buffer\_overflow.', 'loadmodule.', 'perl.', 'neptune.', 'smurf.', 'guess\_passwd.', 'pod.', 'teardrop.', 'portsweep.', 'ipsweep.', 'land.', 'ftp\_write.', 'back.', 'imap.', 'satan.', 'phf.', 'nmap.', 'multihop.', 'warezmaster.', 'warezclient.','spy.', 'rootkit.']**  **if inputs[41] in label\_list:**  **return find\_index(inputs[41],label\_list)[0]**  **else:**  **label\_list.append(inputs[41])**  **return find\_index(inputs[41],label\_list)[0]**  **if \_\_name\_\_=='\_\_main\_\_':**  **# 文件名**  **source\_file='data/kddcup.data\_10\_percent\_corrected'**  **handled\_file='kddcup.data\_10\_percent\_corrected.csv'**    **# 文件写入**  **with open(source\_file,'r') as data\_source:**  **csv\_reader = csv.reader(data\_source)**  **data\_file = open(handled\_file,'w',newline='')**  **csv\_writer = csv.writer(data\_file)**  **count = 0 #行数**  **for row in csv\_reader:**  **temp\_line=np.array(row)**  **temp\_line[1] = handleProtocol(row)**  **temp\_line[2] = handleService(row)**  **temp\_line[3] = handleFlag(row)**  **temp\_line[41] = handleLabel(row)**  **csv\_writer.writerow(temp\_line)**  **count += 1**  **data\_file.close()** |



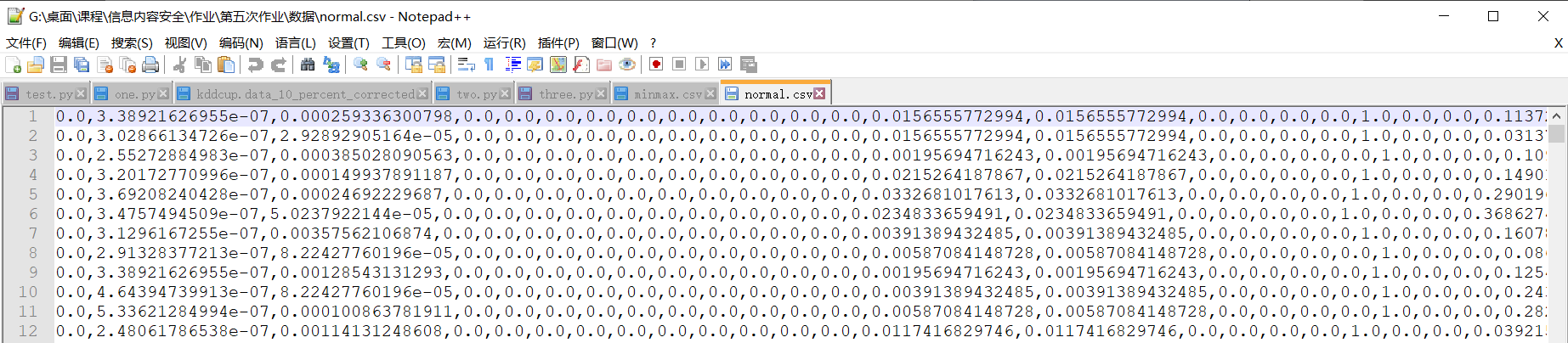
图五 转换结果

由图五可以看到，数据中不同列相差很大，直接采用KNN会发生大数据吞并小数据，所以我们需要对连续值做标准化处理核对数据做归一化处理。

（2）数据标准化

方法：Z-score标准化

|  |
| --- |
| **# -\*- coding: utf-8 -\*-**  **import numpy as np**  **import pandas as pd**  **import csv**  **global x\_mat**  **# 数据标准化**  **def ZscoreNormalization(x, n):**  **print(len(x))**  **i = 0**  **while i<len(x):**  **x\_mat[i][n] = (x[i] - np.mean(x)) / np.std(x)**  **if x\_mat[i][n]>0:**  **print(x\_mat[i][n])**  **i = i + 1**  **print("The ", n , "feature is normal.")**  **# 创建矩阵存储数据集**  **x\_mat = np.zeros((line\_nums, 42))**  **with open("kddcup.data\_10\_percent\_corrected.csv") as fr:**  **lines = fr.readlines()**  **line\_nums = len(lines)**  **for i in range(line\_nums):**  **line = lines[i].strip()**  **item\_mat = line.split(',')**  **x\_mat[i, :] = item\_mat[0:42]**  **# 标准化处理**  **ZscoreNormalization(x\_mat[:, 0], 0) #duration**  **ZscoreNormalization(x\_mat[:, 0], 4) #src\_bytes**  **ZscoreNormalization(x\_mat[:, 0], 5) #dst\_bytes**  **ZscoreNormalization(x\_mat[:, 0], 7) #wrong\_fragment**  **ZscoreNormalization(x\_mat[:, 0], 8) #urgent**  **ZscoreNormalization(x\_mat[:, 0], 9) #hot**  **ZscoreNormalization(x\_mat[:, 0], 10) #num\_failed\_logins**  **ZscoreNormalization(x\_mat[:, 0], 12) #num\_compromised**  **ZscoreNormalization(x\_mat[:, 0], 14) #su\_attempte**  **ZscoreNormalization(x\_mat[:, 0], 15) #num\_root**  **ZscoreNormalization(x\_mat[:, 0], 16) #num\_file\_creations**  **ZscoreNormalization(x\_mat[:, 0], 17) #num\_shells**  **ZscoreNormalization(x\_mat[:, 0], 18) #num\_access\_files**  **ZscoreNormalization(x\_mat[:, 0], 19) #num\_outbound\_cmds**  **ZscoreNormalization(x\_mat[:, 0], 22) #count**  **ZscoreNormalization(x\_mat[:, 0], 23) #srv\_count**  **ZscoreNormalization(x\_mat[:, 0], 24) #serror\_rate**  **ZscoreNormalization(x\_mat[:, 0], 25) #srv\_serror\_rate**  **ZscoreNormalization(x\_mat[:, 0], 26) #rerror\_rate**  **ZscoreNormalization(x\_mat[:, 0], 27) #srv\_rerror\_rate**  **ZscoreNormalization(x\_mat[:, 0], 28) #same\_srv\_rate**  **ZscoreNormalization(x\_mat[:, 0], 29) #diff\_srv\_rate**  **ZscoreNormalization(x\_mat[:, 0], 30) #srv\_diff\_host\_rate**  **ZscoreNormalization(x\_mat[:, 0], 31) #dst\_host\_count**  **ZscoreNormalization(x\_mat[:, 0], 32) #dst\_host\_srv\_count**  **ZscoreNormalization(x\_mat[:, 0], 33) #dst\_host\_same\_srv\_rate**  **ZscoreNormalization(x\_mat[:, 0], 34) #dst\_host\_diff\_srv\_rate**  **ZscoreNormalization(x\_mat[:, 0], 35) #dst\_host\_same\_src\_port\_rate**  **ZscoreNormalization(x\_mat[:, 0], 36) #dst\_host\_srv\_diff\_host\_rate**  **ZscoreNormalization(x\_mat[:, 0], 37) #dst\_host\_serror\_rate**  **ZscoreNormalization(x\_mat[:, 0], 38) #dst\_host\_srv\_serror\_rate**  **ZscoreNormalization(x\_mat[:, 0], 39) #dst\_host\_rerror\_rate**  **ZscoreNormalization(x\_mat[:, 0], 40) #dst\_host\_srv\_rerror\_rate**  **# 写入**  **with open("test-normal-result.csv",'wb+') as data\_file:**  **csv\_writer = csv.writer(data\_file)**  **i = 0**  **while i<len(x\_mat[:, 0]):**  **csv\_writer.writerow(x\_mat[i, :])**  **i += 1** |

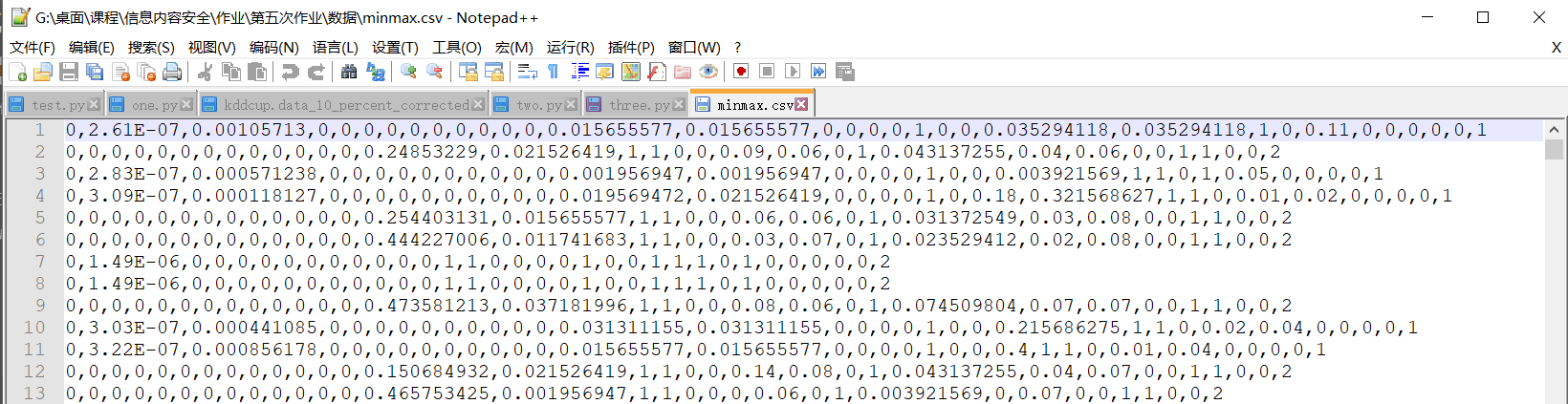


图六 标准化结果

（3）数据归一化

方法：min-max标准化

|  |
| --- |
| **# -\*- coding: utf-8 -\*-**  **import numpy as np**  **import pandas as pd**  **import csv**  **global x\_mat**  **# 数据归一化**  **def MinmaxNormalization(x, n):**  **print(len(x))**  **i = 0**  **while i<len(x):**  **x\_mat[i][n] = (x[i] - np.min(x)) / (np.max(x)-np.min(x))**  **i = i + 1**  **print("The ", n , "feature is normal.")**  **# 创建矩阵存储数据集**  **x\_mat = np.zeros((line\_nums, 42))**  **with open("normal.csv") as fr:**  **lines = fr.readlines()**  **line\_nums = len(lines)**  **for i in range(line\_nums):**  **line = lines[i].strip()**  **item\_mat = line.split(',')**  **x\_mat[i, :] = item\_mat[0:42]**  **# 归一化处理**  **MinmaxNormalization(x\_mat[:, 0], 0) #duration**  **MinmaxNormalization(x\_mat[:, 0], 4) #src\_bytes**  **MinmaxNormalization(x\_mat[:, 0], 5) #dst\_bytes**  **MinmaxNormalization(x\_mat[:, 0], 7) #wrong\_fragment**  **MinmaxNormalization(x\_mat[:, 0], 8) #urgent**  **MinmaxNormalization(x\_mat[:, 0], 9) #hot**  **MinmaxNormalization(x\_mat[:, 0], 10) #num\_failed\_logins**  **MinmaxNormalization(x\_mat[:, 0], 12) #num\_compromised**  **MinmaxNormalization(x\_mat[:, 0], 14) #su\_attempte**  **MinmaxNormalization(x\_mat[:, 0], 15) #num\_root**  **MinmaxNormalization(x\_mat[:, 0], 16) #num\_file\_creations**  **MinmaxNormalization(x\_mat[:, 0], 17) #num\_shells**  **MinmaxNormalization(x\_mat[:, 0], 18) #num\_access\_files**  **MinmaxNormalization(x\_mat[:, 0], 19) #num\_outbound\_cmds**  **MinmaxNormalization(x\_mat[:, 0], 22) #count**  **MinmaxNormalization(x\_mat[:, 0], 23) #srv\_count**  **MinmaxNormalization(x\_mat[:, 0], 24) #serror\_rate**  **MinmaxNormalization(x\_mat[:, 0], 25) #srv\_serror\_rate**  **MinmaxNormalization(x\_mat[:, 0], 26) #rerror\_rate**  **MinmaxNormalization(x\_mat[:, 0], 27) #srv\_rerror\_rate**  **MinmaxNormalization(x\_mat[:, 0], 28) #same\_srv\_rate**  **MinmaxNormalization(x\_mat[:, 0], 29) #diff\_srv\_rate**  **MinmaxNormalization(x\_mat[:, 0], 30) #srv\_diff\_host\_rate**  **MinmaxNormalization(x\_mat[:, 0], 31) #dst\_host\_count**  **MinmaxNormalization(x\_mat[:, 0], 32) #dst\_host\_srv\_count**  **MinmaxNormalization(x\_mat[:, 0], 33) #dst\_host\_same\_srv\_rate**  **MinmaxNormalization(x\_mat[:, 0], 34) #dst\_host\_diff\_srv\_rate**  **MinmaxNormalization(x\_mat[:, 0], 35) #dst\_host\_same\_src\_port\_rate**  **MinmaxNormalization(x\_mat[:, 0], 36) #dst\_host\_srv\_diff\_host\_rate**  **MinmaxNormalization(x\_mat[:, 0], 37) #dst\_host\_serror\_rate**  **MinmaxNormalization(x\_mat[:, 0], 38) #dst\_host\_srv\_serror\_rate**  **MinmaxNormalization(x\_mat[:, 0], 39) #dst\_host\_rerror\_rate**  **MinmaxNormalization(x\_mat[:, 0], 40) #dst\_host\_srv\_rerror\_rate**  **# 写入**  **with open("minmax.csv",'wb+') as data\_file:**  **csv\_writer = csv.writer(data\_file)**  **i = 0**  **while i<len(x\_mat[:, 0]):**  **csv\_writer.writerow(x\_mat[i, :])**  **i = i + 1** |



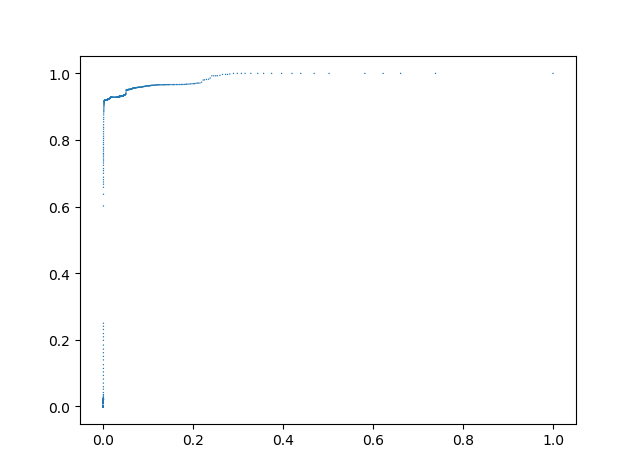
图七 归一化结果

3. KNN

|  |
| --- |
| **# -\*- coding: utf-8 -\*-**  **import os**  **import csv**  **import numpy as np**  **from sklearn.svm import SVC**  **from sklearn import metrics**  **import matplotlib.pyplot as plt**  **from matplotlib.colors import ListedColormap**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn.decomposition import PCA**  **from sklearn import neighbors**  **# 创建矩阵存储数据集**  **x\_mat = np.zeros((line\_nums, 31))**  **y\_label = []**  **with open("minmax.csv") as fr:**  **lines = fr.readlines()**  **line\_nums = len(lines)**  **for i in range(line\_nums):**  **line = lines[i].strip()**  **item\_mat = line.split(',')**  **x\_mat[i, :] = item\_mat[0:31]**  **y\_label.append(item\_mat[-1])**  **# 划分数据集(训练集80%，测试集20%)**  **y = []**  **for n in y\_label:**  **y.append(int(float(n)))**  **y = np.array(y, dtype = int)**  **train\_data, test\_data, train\_target, test\_target = train\_test\_split(x\_mat, y, test\_size=0.2, random\_state=42)**  **print(train\_data.shape, train\_target.shape)**  **print(test\_data.shape, test\_target.shape)**  **# KNN**  **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**  **distance = sq\_diff\_mat.sum(axis=1)\*\*0.5**  **return distance.min(axis=0)**  **test\_size = len(test\_target)**  **result = np.zeros((test\_size, 3))**  **for i in range(test\_size):**  **result[i] = i + 1, classify(test\_data[i], train\_data), test\_target[i]**  **result = np.transpose(result)** |

4.评价

|  |
| --- |
| **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] != 1:**  **abnormal1 += 1**  **roc\_rate[0][j] = normal1 / normal**  **roc\_rate[1][j] = abnormal1 / abnormal**  **return roc\_rate**  **roc\_rate = roc(result)**  **plt.scatter(roc\_rate[0], roc\_rate[1], edgecolors='None', s=1, alpha=1)**  **plt.show()** |



图八 ROC曲线

其中，横轴为误报率，纵轴为检测率。误报率=阈值以上正常点/全体正常的点，检测率-阈值以上异常点/全体异常点

5.参考

[1] KDD CUP 99数据集介绍[EB/OJ]. https://blog.csdn.net/com\_stu\_zhang/article/details/6987632.

[2] KDD CUP99数据集预处理(Python实现)[EB/OJ]. https://blog.csdn.net/asialee\_bird/article/details/80491256.

[3] 使用多种机器学习算法挖掘KDD99记录中的攻击记录[EB/OJ]. https://blog.csdn.net/qq\_37865996/article/details/88111980.

6.感受

结果不太理想，个人感觉可能KNN对这个问题并不是很有效，我需要看一下KNN的具体实现，但这就需要接下来对这一问题进行深入研究了。

**六、系统设计之CNN**

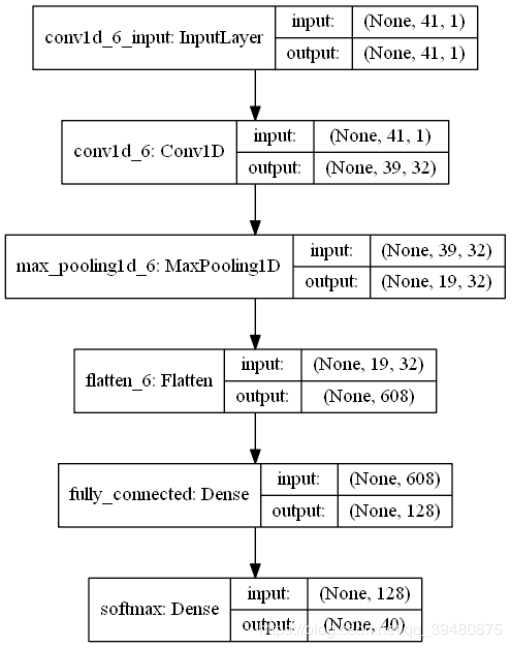
1.库

pandas，keras，sklearn，matplotlib

2.数据预处理

与上文相同，不再赘述。

3.CNN



图九 神经网络

|  |
| --- |
| **import pandas as pd**  **import keras**  **from keras.models import Sequential # 序贯模型**  **from keras.layers import Dense # 全连接层**  **from keras.layers import Dropout # 随机失活层**  **from keras.layers import Flatten # 展平层，从卷积层到全连接层必须展平**  **from keras.layers import Conv1D # 卷积层**  **from keras.layers import MaxPooling1D # 最大值池化**  **from keras import backend as k**  **from sklearn.cross\_validation import train\_test\_split # 随机划分为训练子集和测试子集**  **from keras.utils.vis\_utils import plot\_model**  **from keras.optimizers import SGD**  **import matplotlib.pyplot as plt**  **batch\_size = 128 #一批训练样本128张图片**  **num\_classes = 40 # 有40个类别**  **epochs = 12 # 一共迭代12轮**  **x\_train = pd.read\_csv('train\_x.csv',header=None).values**  **y\_train = pd.read\_csv('train\_y.csv',header=None).values**  **x\_test = pd.read\_csv('test\_x.csv',header=None).values**  **y\_test = pd.read\_csv('test\_y.csv',header=None).values**  **if k.image\_data\_format() == 'channels\_first':**  **x\_train = x\_train.reshape(x\_train.shape[0], 1, 41)**  **x\_test = x\_test.reshape(x\_test.shape[0], 1, 41)**  **input\_shape = (1, 41)**  **else:**  **x\_train = x\_train.reshape(x\_train.shape[0], 41, 1)**  **x\_test = x\_test.reshape(x\_test.shape[0], 41, 1)**  **input\_shape = (41, 1)**  **model = Sequential()**  **model.add(Conv1D(32, 3, activation='relu',input\_shape=input\_shape))**  **model.add(MaxPooling1D(pool\_size=(2)))**  **model.add(Flatten())**  **model.add(Dense(128, activation='relu',name='fully\_connected'))**  **model.add(Dense(num\_classes, activation='softmax',name='softmax'))**  **# 损失函数:多类对数损失，用于多分类问题**  **# 优化函数：adadelta**  **# 评估：准确率**  **model.compile(loss='categorical\_crossentropy', optimizer='adadelta', metrics=['accuracy'])**  **# 训练**  **model.fit(x\_train, y\_train, batch\_size= batch\_size, epochs=epochs, verbose=1)**  **# 测试**  **score = model.evaluate(x\_test, y\_test, verbose=0,batch\_size= batch\_size)**  **print('Test loss:', score[0])**  **print('Test accuracy: %.2f%%' % (score[1] \* 100))** |



图十 测试结果

4.参考

[1] KDDCUP99数据集处理(Keras) [EB/OJ]. https://blog.csdn.net/qq\_39480875/article/details/103782178.

5.感受

这一方法主要参考了其他人设计好的网络，但与别人的结果相差很大，起初推测可能是划分数据集时的随机性引起的，但测试几次后仍相差很大，一时间找不到原因，最后才发现是预处理这一步骤引起了结果的差异，想通过采用更复杂的网络来提升结果准确率，但因为电脑性能原因只好作罢。