|  |
| --- |
|  |
| Intrusion Detection |
| Scikit决策树分类算法 |
|  |
| 彭兆卿  戴君义  倪令斌  Dastan |
|  |

|  |
| --- |
|  |

2016年12月1日

**目录**

1. 网络入侵检测概述
2. [Kdd99数据集](http://blog.csdn.net/ppp8300885/article/details/53406511#t1)
3. [数据预处理](http://blog.csdn.net/ppp8300885/article/details/53406511#t2)
   1. [原始文件预处理](http://blog.csdn.net/ppp8300885/article/details/53406511#t3)
   2. [数据储存](http://blog.csdn.net/ppp8300885/article/details/53406511#t4)
4. [Scikit决策树分类](http://blog.csdn.net/ppp8300885/article/details/53406511#t5)
   1. [加载数据](http://blog.csdn.net/ppp8300885/article/details/53406511#t6)
   2. [类别变量重编码](http://blog.csdn.net/ppp8300885/article/details/53406511#t7)
   3. [特征选取feature selection](http://blog.csdn.net/ppp8300885/article/details/53406511#t8)
   4. [交叉验证（一](http://blog.csdn.net/ppp8300885/article/details/53406511#t9)）
   5. [训练决策树模型](http://blog.csdn.net/ppp8300885/article/details/53406511#t10)
   6. [交叉验证（二](http://blog.csdn.net/ppp8300885/article/details/53406511#t11)）
   7. [测试](http://blog.csdn.net/ppp8300885/article/details/53406511#t12)
5. [总结](http://blog.csdn.net/ppp8300885/article/details/53406511#t13)

1. 网络入侵检测概述

网络入侵检测的目的是防止计算机网络免受非授权用户的侵害。运用机器学习的方法，网络入侵检测器通过学习，可以构建出预测模型，它能够分辨出有害连接（入侵或者攻击）和正常连接。

一般来说，入侵被划分为四种主要类型：

1. DOS（denial-of-service）: 请求拒绝，例如泛洪攻击。
2. R2L:远程机器的未授权接入，例如猜测密码 。
3. U2R:未授权接入本地超级用户特权，例如多种缓冲区的溢出攻击。
4. probing:监察或者其他监测，例如端口扫描 。

某些特征可以被用来分辨普通连接和入侵，比如：

same host：检查之前两秒内的连接和当前连接具有相同的目的主机，并且计算和协议相关的行为、服务等等。

same service：仅仅去检查之前两秒内的连接和当前连接具有同样的服务。

Same host和same service同时被称为 time-based traffic features of the connection records。

但是，某些probing攻击使用多于两秒的时间间隔扫描主机或端口，例如每分钟一次。因此，连接记录也由目的主机进行排序，并且feature被100个连接到同一主机的窗口构造，而不是使用实时窗口。这样就产生了被称为host-based traffic features的数据集。

不同于大多数DOS和probing入侵，在R2L和 U2R 入侵中似乎并没有频繁出现的序列模式。这是因为DOS和 probing入侵包含在很短的时间段内和主机的许多联系，但是R2L和 U2R 入侵是嵌入在数据端口和数据包中的，并且一般它们只包含一个连接。

2. [Kdd99数据集](http://blog.csdn.net/ppp8300885/article/details/53406511#t1)

KDD Cup是由SIGKDD赞助的数据挖掘分析的比赛，每年举行一次，[Kdd99](https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html)是1999年比赛的关于入侵检测的数据集，不过现在学术界已经抛弃了这个数据集，若要[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "_blank)自己算法的有效性建议使用其他数据集，我们只为初窥两种分类算法。

任务目的是检测网络连接，区分正常和非正常的连接。非正常的连接大概有四大类: DOS、R2L、 U2R和probing，每一类下还有若干小类别。

值得注意的是，训练集和测试集中的数据概率分布不同，测试集比训练集多出了14种攻击小类别，官方解释是：这能够很好的模拟真实情况，而且新的攻击类别一般为已知攻击类别的变形。因此算法模型需要能很好的抓住每种攻击类别的feature，数据集可以在官网上获得。

3. 数据预处理

3.1 原始文件预处理

原始训练集和测试集的分类结果都是攻击的特定类，共有24+14=38种 ，在这里，我们不先考虑那么细的分类，只把所有的连接分到四个攻击的大类即可，因此我们根据官网提供的映射关系，将type属性降为5类，用数字0，1，2，3，4替换，分别代表四种攻击的大类和正常情况。

执行以下代码能将位于raw目录下的测试集和训练集预处理到data目录下：

* from io import open
* category = {}
* category["back."] = 2
* category["buffer\_overflow."] = 3
* category["ftp\_write."] = 4
* category["guess\_passwd."] = 4
* category["imap."] = 4
* category["ipsweep."] = 1
* category["land."] = 2
* category["loadmodule."] = 3
* category["multihop."] = 4
* category["neptune."] = 2
* category["nmap."] = 1
* category["perl."] = 3
* category["phf."] = 4
* category["pod."] = 2
* category["portsweep."] = 1
* category["rootkit."] = 3
* category["satan."] = 1
* category["smurf."] = 2
* category["spy."] = 4
* category["teardrop."] = 2
* category["warezclient."] = 4
* category["warezmaster."] = 4
* category["apache2."] = 2
* category["back."] = 2
* category["buffer\_overflow."] = 3
* category["ftp\_write."] = 4
* category["guess\_passwd."] = 4
* category["httptunnel."] = 4
* category["httptunnel."] = 3
* category["imap."] = 4
* category["ipsweep."] = 1
* category["land."] = 2
* category["loadmodule."] = 3
* category["mailbomb."] = 2
* category["mscan."] = 1
* category["multihop."] = 4
* category["named."] = 4
* category["neptune."] = 2
* category["nmap."] = 1
* category["perl."] = 3
* category["phf."] = 4
* category["pod."] = 2
* category["portsweep."] = 1
* category["processtable."] = 2
* category["ps."] = 3
* category["rootkit."] = 3
* category["saint."] = 1
* category["satan."] = 1
* category["sendmail."] = 4
* category["smurf."] = 2
* category["snmpgetattack."] = 4
* category["snmpguess."] = 4
* category["sqlattack."] = 3
* category["teardrop."] = 2
* category["udpstorm."] = 2
* category["warezmaster."] = 2
* category["worm."] = 4
* category["xlock."] = 4
* category["xsnoop."] = 4
* category["xterm."] = 3
* category["normal."] = 0
* attr\_list =['duration', 'protocol\_type','service', 'flag', 'src\_bytes', 'dst\_bytes', 'land','wrong\_fragment', 'urgent','hot', 'num\_failed\_logins', 'logged\_in', 'num\_compromised', 'root\_shell', 'su\_attempted', 'num\_root', 'num\_file\_creations', 'num\_shells', 'num\_access\_files', 'num\_outbound\_cmds', 'is\_host\_login', 'is\_guest\_login', 'count', 'srv\_count', 'serror\_rate', 'srv\_serror\_rate', 'rerror\_rate', 'srv\_rerror\_rate', 'same\_srv\_rate', 'diff\_srv\_rate', 'srv\_diff\_host\_rate', 'dst\_host\_count', 'dst\_host\_srv\_count', 'dst\_host\_same\_srv\_rate', 'dst\_host\_diff\_srv\_rate', 'dst\_host\_same\_src\_port\_rate', 'dst\_host\_srv\_diff\_host\_rate', 'dst\_host\_serror\_rate', 'dst\_host\_srv\_serror\_rate', 'dst\_host\_rerror\_rate', 'dst\_host\_srv\_rerror\_rate', 'type']
* def transform\_type(input\_file, output\_file):
* with open(output\_file, "w") as text\_file:
* with open(input\_file) as f:
* lines = f.readlines()
* for line in lines:
* columns = line.split(',')
* for raw\_type in category:
* flag = False
* if raw\_type == columns[-1].replace("\n", ""):
* str = ','.join(columns[0:attr\_list.index('type')])
* text\_file.write("%s,%d\n" % (str, category[raw\_type]))
* flag = True
* break
* if not flag:
* text\_file.write(line)
* print(line)
* transform\_type("raw/kddcup.data\_10\_percent.txt", "data/kddcup.data\_10\_percent.txt")
* transform\_type("raw/corrected.txt", "data/corrected.txt")

3.2 数据储存

为了后面方便读取、排序等操作，降低运行时间，我们把data目录下的文件数据读取并写入[MongoDB](http://lib.csdn.net/base/mongodb" \o "MongoDB知识库" \t "_blank)[数据库](http://lib.csdn.net/base/mysql)。

读写的过程中，注意将有些变量转化int和float后再存储，同时为training\_set表的type属性增加索引。

* # -------------import data------------------
* from pymongo import MongoClient
* client = MongoClient('localhost', 27017)
* db = client.test
* def isfloat(value):
* try:
* float(value)
* return True
* except ValueError:
* return False
* db.training\_data.delete\_many({})
* db.training\_data.create\_index("training\_set.type")
* with open("data/kddcup.data\_10\_percent.txt") as f:
* lines = f.readlines()
* for line in lines:
* columns = line.split(',')
* dic = {}
* for attr in attr\_list:
* element = columns[attr\_list.index(attr)]
* if element.isdigit():
* element = int(element)
* elif isfloat(element):
* element = float(element)
* dic[attr] = element
* db.training\_data.insert\_one({"training\_set": dic})
* db.test\_data.delete\_many({})
* with open("data/corrected.txt") as f:
* lines = f.readlines()
* for line in lines:
* columns = line.split(',')
* dic = {}
* for attr in attr\_list:
* element = columns[attr\_list.index(attr)]
* if element.isdigit():
* element = int(element)
* elif isfloat(element):
* element = float(element)
* dic[attr] = element
* db.test\_data.insert\_one({"test\_set": dic})

4. Scikit决策树分类

4.1加载数据

我们从MongoDb中取出所有的训练集和测试集，由于决策树对于有序的输入能够加速训练，因此我们使用sort方法对训练集中的type属性进行排序，然后训练集和测试集所有的输入储存在dataset里，将所有的输出目标储存在datatarget，而且为区分训练集和测试集，我们用T\_len记录训练集的大小。

注意：从MongoDb数据库读出的字符串为unicode，因此需将其重新编码为ascii：

* # -------------Fetching data------------------
* from pymongo import MongoClient
* import pymongo
* import numpy as np
* client = MongoClient('localhost', 27017)
* db = client.test
* training\_cursor = db.training\_data.find({"training\_set.src\_bytes": {"$gt": 1000000}})
* test\_cursor = db.test\_data.find({"test\_set.src\_bytes": {"$gt": 1000000}})
* cursor = training\_cursor.sort('training\_set.type', pymongo.ASCENDING)
* dataset = []
* datatarget = []
* for document in cursor:
* tmp = []
* for attr in attr\_list:
* if attr is not 'type':
* try:
* tmp.append(document['training\_set'][attr].encode('ascii'))
* except:
* tmp.append(document['training\_set'][attr])
* dataset.append(tmp)
* dataTarget.append(int(document['training\_set']['type']))
* training\_len = len(dataset)
* for document in test\_cursor:
* tmp = []
* for attr in attr\_list:
* if attr is not 'type':
* try:
* tmp.append(document['test\_set'][attr].encode('ascii'))
* except:
* tmp.append(document['test\_set'][attr])
* dataset.append(tmp)
* dataTarget.append(int(document['test\_set']['type']))
* dataset = np.array(dataset)
* datatarget = np.array(datatarget)
* T\_len = training\_len

4.2类别变量重编码

对于类别变量(Categorical Variable)，例如：男和女、high和low等，这种字符串变量一般是不能直接输入到算法模型中的，需要重编码为数字1,2,3,4等或者是二进制bitmap。

**但是对于1,2,3,4这样的有顺序大小的标量，有些算法理解会出2大于1，即男大于女，而事实上男和女只代表是男或女，并没有先后和大小关系，所以一般使用二进制编码：男用0 1表示 、女用1 0表示。而这样就增加了变量的个数，从一维变量sex增加到了两维变量sex=male和sex=female，因此，若类别变量的种类很多，最后编码后的变量维度会极大的增加，不利于计算，被称为维度灾难。**

而对于决策树算法来说，利用的是信息纯度来分类，因此编码为1,2,3不会产生影响。所以我们将dataset中的三个类别变量（protocol\_type, service, flag）重新编码为数字集合。

# -------------Categorical variable encoding------------------

from sklearn import preprocessing

le\_1 = preprocessing.LabelEncoder()

le\_2 = preprocessing.LabelEncoder()

le\_3 = preprocessing.LabelEncoder()

le\_1.fit(np.unique(dataset[:, 1]))

le\_2.fit(np.unique(dataset[:, 2]))

le\_3.fit(np.unique(dataset[:, 3]))

dataset[:, 1] = le\_1.transform(dataset[:, 1])

dataset[:, 2] = le\_2.transform(dataset[:, 2])

dataset[:, 3] = le\_3.transform(dataset[:, 3])

4.3特征选取（feature selection）

一般的数据集的变量之间会出现一下几种情况：

1. 变量之间相关性太强，互相依赖严重，会导致有冗余变量
2. 变量中的数据极其稀疏，很大部分都是0或者无意义
3. 变量维度太高，经过重编码后变量维度达到了几百个

以上三种情况都很不利对模型的训练，情况2种能通过设置阈值来过滤一些稀疏的变量；情况1和3能通过主成分分析（PCA）提取主要因子进行降维，除此之外，还有属性子集选择方法：通过删除不相关或者冗余的属性减少数据量，有逐步向前选择、逐步向后选择和决策树归纳三种。

经过前面类别变量的重编码，dataset变量一共有42个维度，高维度的数据集在决策树中的训练很容易出现过拟合的情况，因此我们需要对变量进行筛选，也就是feature selection，一组好的feature能直接影响算法的精度，而Scikit已经提供了基于树的属性子集选择方法，我们在这里直接把数据输入ExtraTreesClassifier训练，然后用SelectFromModel提取训练的模型，最后使用transform方法筛选出重要的特征。

由于测试集的变量也需要筛选，但是筛选原则应该与训练集一致，不然训练集和测试集的input会不一样，因此我们将这组feature的索引存在fea\_index数组中。

* # -------------feature selection------------------
* from sklearn.ensemble import ExtraTreesClassifier
* from sklearn.feature\_selection import SelectFromModel
* data\_set = dataset[0:(T\_len - 1)]
* data\_target = datatarget[0:(T\_len - 1)]
* clf = ExtraTreesClassifier()
* clf = clf.fit(data\_set, data\_target)
* print clf.feature\_importances\_
* model = SelectFromModel(clf, prefit=True)
* feature\_set = model.transform(data\_set)
* fea\_index = []
* for A\_col in np.arange(data\_set.shape[1]):
* for B\_col in np.arange(feature\_set.shape[1]):
* if (data\_set[:, A\_col] == feature\_set[:, B\_col]).all():
* fea\_index.append(A\_col)

**Output:**

{'count': '8', 'srv\_serror\_rate': '0.0', 'srv\_count': '8', 'same\_srv\_rate': '1.0', 'dst\_host\_same\_src\_port\_rate': '0.11', 'dst\_host\_srv\_rerror\_rate': '0.0', 'dst\_host\_srv\_count': '9', 'dst\_host\_count': '9', 'logged\_in': '1', 'protocol\_type': '1'}

我们将fea\_index对应的变量名和其中的某行数据打印出来，可以看出最后筛选出的feature：一共有11个，分别为count，srv\_serror\_rate，srv\_count，same\_srv\_rate，dst\_host\_same\_src\_port\_rate，dst\_host\_srv\_rerror\_rate，dst\_host\_srv\_rerror\_rate，dst\_host\_srv\_count，dst\_host\_count，logged\_in和protocol\_type。

4.4交叉验证（一）

在训练模型前，我们将训练集的十分之一用来做交叉验证，剩下数据用来做训练，因此将训练集分为了两部分：

1. training\_set、training\_target：用来训练模型
2. test\_set、test\_target：用来交叉验证

# -------------Cross Validation Split----------------

from sklearn import tree

import random

import pydotplus

from sklearn.externals import joblib

test\_index = random.sample(range(0, len(feature\_set) - 1), int(len(data\_target) \* 0.1))

training\_index = list(set(range(0, len(feature\_set) - 1)) - set(test\_index))

training\_set = feature\_set[training\_index]

training\_target = data\_target[training\_index]

test\_set = feature\_set[test\_index]

test\_target = data\_target[test\_index]

**Output:**

training\_set: (444617, 11)

training\_target: (444617,)

test\_set: (49401, 11)

test\_target: (49401,)

这里能可以看出训练集有444617行，用于交叉验证的训练集集有49401行，两者的变量都为11维度

4.5训练决策树模型

决策树是一种非参数的监督学习算法，常用的有ID3、C4.5和CART，ID3和C4.5基于信息熵entropy，CART基于gini不纯度，而Scikit提供了一种优化版本的CART算法，封装在[DecisionTree Classifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html" \l "sklearn.tree.DecisionTreeClassifier)里，有一些主要的参数：

* criterion：节点分裂的准则，gini or entropy
* max\_depth：树的最大深度，这个值过大会导致算法对训练集的过拟合，而过小的值会妨碍算法对数据的学习，初始值推荐为5，然后慢慢增大，观察树的形状
* min\_samples\_split：需要被分裂成子节点的最小样本数，当样本数小于这个值时，就直接标记为叶节点而不用继续生成子节点，值越大，树的枝越少，达到一定的剪枝效果
* class\_weight：当输入的样本集类别数量相差很大时，树最后的形状会倾向朝大数据样本的方向生成，导致树的不平衡，因此可以设置各个类别的权重，数据量少的类权重大，数据量大的类权重小，能够有效地将树平衡

通过反复尝试，我们找到一组比较好的训练参数：用信息entropy来分裂各个节点；min\_samples\_split＝30；让所有训练样本充分平衡，即输入的各类元组数相等。

# -------------Training Tree------------------

clf = tree.DecisionTreeClassifier(criterion="entropy", min\_samples\_split=30, class\_weight="balanced")

clf = clf.fit(training\_set, training\_target)

class\_names = np.unique([str(i) for i in training\_target])

feature\_names = [attr\_list[i] for i in fea\_index]

dot\_data = tree.export\_graphviz(clf, out\_file=None,

feature\_names=feature\_names,

class\_names=class\_names,

filled=True, rounded=True,

special\_characters=True)

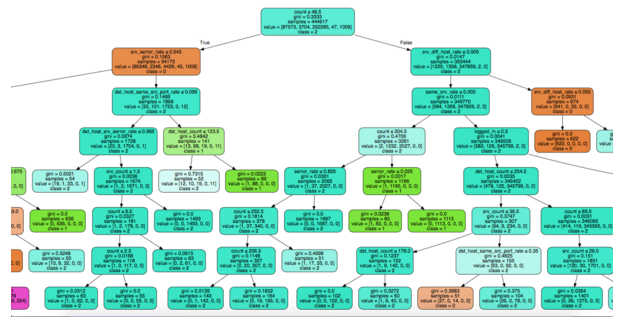
graph = pydotplus.graph\_from\_dot\_data(dot\_data)

graph.write\_pdf("output/tree-vis.pdf")

joblib.dump(clf, 'output/CART.pkl')

训练完模型后，我们可以把模型进行持久化在本地文件里，下次就可以直接从文件读取模型进行预测。

我们还可以对训练的树模型进行可视化，导出为pdf文件，如：



4.6交叉验证（二）

我们对训练集的十分之一（test\_set）进行预测，获得的预测值trained\_target与实际值test\_target进行对比，输出混淆矩阵（confusion matrix）和分类的结果统计，用来验证训练出的模型对于有相同的概率分布的数据集的有效性。

# -------------Prediction of cross validation--------------

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

clf = joblib.load('output/CART.pkl')

trained\_target = clf.predict(test\_set)

print confusion\_matrix(test\_target, trained\_target, labels=[0, 1, 2, 3, 4])

print classification\_report(test\_target, trained\_target)

**Output：**

1.混淆矩阵：

[[ 9558 26 11 17 50]

[ 5 418 2 0 0]

[ 176 19 38992 3 4]

[ 0 0 0 7 1]

[ 4 0 0 6 102]]

混淆矩阵（confusion matrix）给出了分类器分类的最终结果，行表示实际的类，而列表示预测的类。矩阵对角线上的数字表明预测出来的与实际结果一样的元组数，而对角线以外数字表示，预测的与实际结果不一致的元祖个数，如：176表示：有176个元组实际上属于类别2（第2行），而却被预测到了类别0（第0列）。

2.模型精度：

precision recall f1-score support

0 0.98 0.99 0.99 9662

1 0.90 0.98 0.94 425

2 1.00 0.99 1.00 39194

3 0.21 0.88 0.34 8

4 0.65 0.91 0.76 112

avg / total 0.99 0.99 0.99 49401

精度（precision）可以看作是精确性的度量（即标记为正类的元祖实际为正类所占的百分比），而召回率（recall）是完全性的度量（即正元组标记为正的百分比），而f1-score表示精度和召回率的调和均值，它能够度量精度和召回率的组合（以相同的权重），support是这类元组的个数。

可以看出各个类的数量比例差距很大，此交叉验证输入的属于类别3的元组数只有3个。

4.7 测试

最后我们来测试训练的模型是否能够很好的预测测试数据，我们利用T\_len从dataset中取出测试输入test\_data\_set 和test\_data\_target，然后使用fea\_index同样将测试输入提取子属性（特征），最终比较预测值test\_trained\_target和实际值test\_data\_target。

# -------------Prediction of test dataset------------------

test\_data\_set = dataset[T\_len:len(dataset)]

test\_data\_target = datatarget[T\_len:len(dataset)]

test\_feature\_set = test\_data\_set[:, fea\_index]

clf = joblib.load('output/CART.pkl')

test\_trained\_target = clf.predict(test\_feature\_set)

print confusion\_matrix(test\_data\_target, test\_trained\_target, labels=[0, 1, 2, 3, 4])

print classification\_report(test\_data\_target, test\_trained\_target)

**Output：**

1.混淆矩阵：

[[ 6294 38 15 10 11]

[ 5 800 4 0 0]

[ 191 20 41508 1 0]

[ 0 0 0 3 0]

[ 1076 5 0 16 3]]

从混淆矩阵我们可以看出，对于类0、类1和类2，分类器还是能正常识别。对于类别3，只有3个元组被正常归类；而对于类别4，也只有3个元组被正常归类，但是有1076个元组实际上属于类别4却被分到了类别0

2.模型精度：

precision recall f1-score support

0 0.83 0.99 0.90 6368

1 0.93 0.99 0.96 809

2 1.00 0.99 1.00 41720

3 0.10 1.00 0.18 3

4 0.21 0.00 0.01 1100

avg / total 0.96 0.97 0.96 50000

经过混淆矩阵的粗略分析，我们算出各个类的精度、召回率、F度量来描述分类器对每个类的分类情况，可以看出来，训练出的决策树模型对前三类有较好的效果

而对于类别3：

precision=0.1表示： 所有预测出来为类3的元组中，预测正确的占0.1，也就是真实也为类3的占0.1

recall=1表示：所有的真实为类3的元组全部被正确分类到类3

5.总结

对比交叉验证的模型精度和最后测试的的模型精度，我们可以看出训练出来的决策树模型能够很好的预测具有相同概率分布的数据集（十分之一的训练集），而对于测试集，由于其概率分布不同，对于类3和类4的分类效果并不是很好。

总结一下本次项目收获：

1. 输入数据格式要正确转换，对于类别变量要用数字编码或者是二进制编码（one-hot-encoding），取决于算法的要求。
2. 原始数据集需要选取合适的特征，尽量选取主要的、非冗余的属性，遇到高维的数据要降维处理，最好用各种特征选取的方法来尝试，选一个效果最佳。
3. 决策树训练时深度从小到大，观察树的形状，必要的时候需要设置树的最大深度以防止出现模型对训练数据的过拟合；对于样本类别数量差异较大的要先设置各类的权重，不然小类别的数据很容易在树的生成的过程中就被大类别的数据湮没；训练完的树最好能进行后剪枝，将重复的枝剪掉，Scikit集成的决策树还没有支持这种剪枝功能，但是可以通过设置min\_samples\_split在生成树的过程中进行一定的剪枝。