Modelling IDS: Multiclass classification

In [1]:

```
import pandas as pd
import numpy as np
import sys
import sklearn
print(pd.__version__)
print(np.__version__)
print(sys.version)
print(sklearn.__version__)

# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
# ignore all user warnings
simplefilter(action='ignore', category=UserWarning)
```

```
0.23.4
1.15.4
3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]
0.20.1
```

Load the Dataset

In [6]:

```
# attach the column names to the dataset
col_names = ["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","label"]

# KDDTrain+_2.csv & KDDTest+_2.csv are the datafiles without the last column about the
 difficulty score
# these have already been removed.
df = pd.read_csv("../Data/KDDTrain+_2.csv", header=None, names = col_names)
df_test = pd.read_csv("../Data/KDDTest+_2.csv", header=None, names = col_names)

# shape, this gives the dimensions of the dataset
print('Dimensions of the Training set:',df.shape)
print('Dimensions of the Test set:',df_test.shape)
```

```
Dimensions of the Training set: (125973, 42)
Dimensions of the Test set: (22544, 42)
```

Sample view of the training dataset

In [7]:

```
# first five rows
df.head(5)
```

Out[7]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgen
0	0	tcp	ftp_data	SF	491	0	0	0	
1	0	udp	other	SF	146	0	0	0	
2	0	tcp	private	S0	0	0	0	0	
3	0	tcp	http	SF	232	8153	0	0	
4	0	tcp	http	SF	199	420	0	0	

5 rows × 42 columns

Statistical Summary

In [8]:

```
df.describe()
```

Out[8]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	ur
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.00
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.00
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.01
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.00
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.00
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.00

8 rows × 38 columns

Label Distribution of Training and Test set

In [9]:

```
print('Label distribution Training set:')
print(df['label'].value_counts())
print()
print('Label distribution Test set:')
print(df_test['label'].value_counts())
```

```
Label distribution Training set:
normal             67343
neptune            41214
satan               3633
ipsweep             3599
portsweep           2931
smurf               2646
nmap                1493
back                 956
teardrop             892
warezclient          890
pod                  201
guess_passwd          53
buffer_overflow       30
warezmaster           20
land                  18
imap                  11
rootkit               10
loadmodule             9
ftp_write              8
multihop               7
phf                    4
perl                   3
spy                    2
Name: label, dtype: int64

Label distribution Test set:
normal             9711
neptune            4657
guess_passwd       1231
mscan               996
warezmaster         944
apache2             737
satan               735
processtable        685
smurf               665
back                359
snmpguess           331
saint               319
mailbomb            293
snmpgetattack       178
portsweep           157
ipsweep             141
httptunnel          133
nmap                 73
pod                  41
buffer_overflow      20
multihop             18
named                17
ps                   15
sendmail             14
xterm                13
rootkit              13
teardrop             12
xlock                 9
land                  7
xsnoop                4
ftp_write             3
udpstorm              2
loadmodule            2
sqlattack             2
```

```
worm                 2
perl                 2
phf                  2
imap                 1
Name: label, dtype: int64
```

Data preprocessing:

Identify categorical features

In [10]:

```
# colums that are categorical and not binary yet: protocol_type (column 2), service (co
lumn 3), flag (column 4).
# explore categorical features
print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_na
me, unique_cat=unique_cat))

#see how distributed the feature service is, it is evenly distributed and therefore we
 need to make dummies for all.
print()
print('Distribution of categories in service:')
print(df['service'].value_counts().sort_values(ascending=False).head())
```

```
Training set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 70 categories
Feature 'flag' has 11 categories
Feature 'label' has 23 categories

Distribution of categories in service:
http        40338
private     21853
domain_u     9043
smtp         7313
ftp_data     6860
Name: service, dtype: int64
```

In [11]:

```
# Test set
print('Test set:')
for col_name in df_test.columns:
    if df_test[col_name].dtypes == 'object' :
        unique_cat = len(df_test[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_na
me, unique_cat=unique_cat))
```

```
Test set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 64 categories
Feature 'flag' has 11 categories
Feature 'label' has 38 categories
```

LabelEncoder

Insert categorical features into a 2D numpy array

In [12]:

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']
# insert code to get a list of categorical columns into a variable, categorical_columns
categorical_columns=['protocol_type', 'service', 'flag']
 # Get the categorical values into a 2D numpy array
df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]
df_categorical_values.head()
```

Out[12]:

	protocol_type	service	flag
0	tcp	ftp_data	SF
1	udp	other	SF
2	tcp	private	S0
3	tcp	http	SF
4	tcp	http	SF

Make column names for dummies

In [13]:

```
# protocol type
unique_protocol=sorted(df.protocol_type.unique())
string1 = 'Protocol_type_'
unique_protocol2=[string1 + x for x in unique_protocol]
# service
unique_service=sorted(df.service.unique())
string2 = 'service_'
unique_service2=[string2 + x for x in unique_service]
# flag
unique_flag=sorted(df.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]
# put together
dumcols=unique_protocol2 + unique_service2 + unique_flag2
print(dumcols)

#do same for test set
unique_service_test=sorted(df_test.service.unique())
unique_service2_test=[string2 + x for x in unique_service_test]
testdumcols=unique_protocol2 + unique_service2_test + unique_flag2
```

```
['Protocol_type_icmp', 'Protocol_type_tcp', 'Protocol_type_udp', 'service_
IRC', 'service_X11', 'service_Z39_50', 'service_aol', 'service_auth', 'ser
vice_bgp', 'service_courier', 'service_csnet_ns', 'service_ctf', 'service_
daytime', 'service_discard', 'service_domain', 'service_domain_u', 'servic
e_echo', 'service_eco_i', 'service_ecr_i', 'service_efs', 'service_exec',
'service_finger', 'service_ftp', 'service_ftp_data', 'service_gopher', 'se
rvice_harvest', 'service_hostnames', 'service_http', 'service_http_2784',
'service_http_443', 'service_http_8001', 'service_imap4', 'service_iso_tsa
p', 'service_klogin', 'service_kshell', 'service_ldap', 'service_link', 's
ervice_login', 'service_mtp', 'service_name', 'service_netbios_dgm', 'serv
ice_netbios_ns', 'service_netbios_ssn', 'service_netstat', 'service_nnsp',
'service_nntp', 'service_ntp_u', 'service_other', 'service_pm_dump', 'serv
ice_pop_2', 'service_pop_3', 'service_printer', 'service_private', 'servic
e_red_i', 'service_remote_job', 'service_rje', 'service_shell', 'service_s
mtp', 'service_sql_net', 'service_ssh', 'service_sunrpc', 'service_supdu
p', 'service_systat', 'service_telnet', 'service_tftp_u', 'service_tim_i',
'service_time', 'service_urh_i', 'service_urp_i', 'service_uucp', 'service
_uucp_path', 'service_vmnet', 'service_whois', 'flag_OTH', 'flag_REJ', 'fl
ag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0', 'flag_S1', 'flag_S2', 'fl
ag_S3', 'flag_SF', 'flag_SH']
```

Transform categorical features into numbers using LabelEncoder()

In [14]:

```
df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)
print(df_categorical_values_enc.head())
# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder().fit_transf
orm)
```

```
   protocol_type  service  flag
0              1       20     9
1              2       44     9
2              1       49     5
3              1       24     9
4              1       24     9
```

One-Hot-Encoding

In [15]:

```
enc = OneHotEncoder()
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)
# test set
testdf_categorical_values_encenc = enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data = pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testd
umcols)

df_cat_data.head()
```

Out[15]:

	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	service_X11	servi
0	0.0	1.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	0.0	
2	0.0	1.0	0.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	0.0	
4	0.0	1.0	0.0	0.0	0.0	

5 rows × 84 columns

Add 6 missing categories from train set to test set

In [16]:

```
trainservice=df['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]
difference
```

Out[16]:

```
['service_http_8001',
 'service_harvest',
 'service_aol',
 'service_urh_i',
 'service_http_2784',
 'service_red_i']
```

In [17]:

```
for col in difference:
    testdf_cat_data[col] = 0

testdf_cat_data.shape
```

Out[17]:

```
(22544, 84)
```

Join encoded categorical dataframe with the non-categorical dataframe

In [18]:

```
newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)
# test data
newdf_test=df_test.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)
print(newdf.shape)
print(newdf_test.shape)
```

```
(125973, 123)
(22544, 123)
```

Split Dataset into 4 datasets for every attack category

Rename every attack label: 0=normal, 1=DoS, 2=Probe, 3=R2L and 4=U2R. Replace labels column with new labels column Make new datasets

In [19]:

```
# take label column
labeldf=newdf['label']
labeldf_test=newdf_test['label']
# change the label column
newlabeldf=labeldf.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod':
1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm'
: 1, 'worm': 1,
                            'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan'
: 2,'saint' : 2
                            ,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3,'p
hf': 3,'spy': 3,'warezclient': 3,'warezmaster': 3,'sendmail': 3,'named': 3,'snmpgetatta
ck': 3,'snmpguess': 3,'xlock': 3,'xsnoop': 3,'httptunnel': 3,
                            'buffer_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit': 4,
'ps': 4,'sqlattack': 4,'xterm': 4})
newlabeldf_test=labeldf_test.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land':
1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1, 'apache2': 1, 'processtable': 1,
'udpstorm': 1, 'worm': 1,
                            'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan'
: 2,'saint' : 2
                            ,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3,'p
hf': 3,'spy': 3,'warezclient': 3,'warezmaster': 3,'sendmail': 3,'named': 3,'snmpgetatta
ck': 3,'snmpguess': 3,'xlock': 3,'xsnoop': 3,'httptunnel': 3,
                            'buffer_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit': 4,
'ps': 4,'sqlattack': 4,'xterm': 4})
# put the new label column back
newdf['label'] = newlabeldf
newdf_test['label'] = newlabeldf_test
print(newdf['label'].head())
```

```
0    0
1    0
2    1
3    0
4    0
Name: label, dtype: int64
```

In [20]:

```
to_drop_DoS = [2,3,4]
to_drop_Probe = [1,3,4]
to_drop_R2L = [1,2,4]
to_drop_U2R = [1,2,3]
DoS_df=newdf[~newdf['label'].isin(to_drop_DoS)];
Probe_df=newdf[~newdf['label'].isin(to_drop_Probe)];
R2L_df=newdf[~newdf['label'].isin(to_drop_R2L)];
U2R_df=newdf[~newdf['label'].isin(to_drop_U2R)];

#test
DoS_df_test=newdf_test[~newdf_test['label'].isin(to_drop_DoS)];
Probe_df_test=newdf_test[~newdf_test['label'].isin(to_drop_Probe)];
R2L_df_test=newdf_test[~newdf_test['label'].isin(to_drop_R2L)];
U2R_df_test=newdf_test[~newdf_test['label'].isin(to_drop_U2R)];
print('Train:')
print('Dimensions of DoS:' ,DoS_df.shape)
print('Dimensions of Probe:' ,Probe_df.shape)
print('Dimensions of R2L:' ,R2L_df.shape)
print('Dimensions of U2R:' ,U2R_df.shape)
print('Test:')
print('Dimensions of DoS:' ,DoS_df_test.shape)
print('Dimensions of Probe:' ,Probe_df_test.shape)
print('Dimensions of R2L:' ,R2L_df_test.shape)
print('Dimensions of U2R:' ,U2R_df_test.shape)
```

```
Train:
Dimensions of DoS: (113270, 123)
Dimensions of Probe: (78999, 123)
Dimensions of R2L: (68338, 123)
Dimensions of U2R: (67395, 123)
Test:
Dimensions of DoS: (17171, 123)
Dimensions of Probe: (12132, 123)
Dimensions of R2L: (12596, 123)
Dimensions of U2R: (9778, 123)
```

Feature Scaling:

In [21]:

```
# Split dataframes into X & Y
# assign X as a dataframe of feautures and Y as a series of outcome variables
X_DoS = DoS_df.drop('label',1)
Y_DoS = DoS_df.label
X_Probe = Probe_df.drop('label',1)
Y_Probe = Probe_df.label
X_R2L = R2L_df.drop('label',1)
Y_R2L = R2L_df.label
X_U2R = U2R_df.drop('label',1)
Y_U2R = U2R_df.label
# test set
X_DoS_test = DoS_df_test.drop('label',1)
Y_DoS_test = DoS_df_test.label
X_Probe_test = Probe_df_test.drop('label',1)
Y_Probe_test = Probe_df_test.label
X_R2L_test = R2L_df_test.drop('label',1)
Y_R2L_test = R2L_df_test.label
X_U2R_test = U2R_df_test.drop('label',1)
Y_U2R_test = U2R_df_test.label
```

In [22]:

```
colNames=list(X_DoS)
colNames_test=list(X_DoS_test)
```

Use StandardScaler() to scale the dataframes

In [24]:

```
from sklearn import preprocessing
scaler1 = preprocessing.StandardScaler().fit(X_DoS)
X_DoS=scaler1.transform(X_DoS)
scaler2 = preprocessing.StandardScaler().fit(X_Probe)
X_Probe=scaler2.transform(X_Probe)
scaler3 = preprocessing.StandardScaler().fit(X_R2L)
X_R2L=scaler3.transform(X_R2L)
scaler4 = preprocessing.StandardScaler().fit(X_U2R)
X_U2R=scaler4.transform(X_U2R)
# test data
#scaler5 = preprocessing.StandardScaler().fit(X_DoS_test)
#X_DoS_test=scaler5.transform(X_DoS_test)
#scaler6 = preprocessing.StandardScaler().fit(X_Probe_test)
#X_Probe_test=scaler6.transform(X_Probe_test)
#scaler7 = preprocessing.StandardScaler().fit(X_R2L_test)
#X_R2L_test=scaler7.transform(X_R2L_test)
#scaler8 = preprocessing.StandardScaler().fit(X_U2R_test)
#X_U2R_test=scaler8.transform(X_U2R_test)


X_DoS_test=scaler1.transform(X_DoS_test)
X_Probe_test=scaler2.transform(X_Probe_test)
X_R2L_test=scaler3.transform(X_R2L_test)
X_U2R_test=scaler4.transform(X_U2R_test)
```

Check that the Standard Deviation is 1

In [25]:

```
print(X_DoS.std(axis=0))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 1.]
```

In [26]:

```
X_Probe.std(axis=0);
X_R2L.std(axis=0);
X_U2R.std(axis=0);
```

Feature Selection:

In [27]:

```
#univariate feature selection with ANOVA F-test. using secondPercentile method, then RF
E
from sklearn.feature_selection import SelectPercentile, f_classif
np.seterr(divide='ignore', invalid='ignore');
selector=SelectPercentile(f_classif, percentile=10)
X_newDoS = selector.fit_transform(X_DoS,Y_DoS)
X_newDoS.shape
```

Out[27]:

```
(113270, 13)
```

Get the features that were selected: DoS

In [28]:

```
true=selector.get_support()
newcolindex_DoS=[i for i, x in enumerate(true) if x]
newcolname_DoS=list( colNames[i] for i in newcolindex_DoS )
newcolname_DoS
```

Out[28]:

```
['logged_in',
 'count',
 'serror_rate',
 'srv_serror_rate',
 'same_srv_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_serror_rate',
 'dst_host_srv_serror_rate',
 'service_http',
 'flag_S0',
 'flag_SF']
```

In [29]:

```
X_newProbe = selector.fit_transform(X_Probe,Y_Probe)
X_newProbe.shape
```

Out[29]:

```
(78999, 13)
```

Get the features that were selected: Probe

In [30]:

```
true=selector.get_support()
newcolindex_Probe=[i for i, x in enumerate(true) if x]
newcolname_Probe=list( colNames[i] for i in newcolindex_Probe )
newcolname_Probe
```

Out[30]:

```
['logged_in',
 'rerror_rate',
 'srv_rerror_rate',
 'dst_host_srv_count',
 'dst_host_diff_srv_rate',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'dst_host_rerror_rate',
 'dst_host_srv_rerror_rate',
 'Protocol_type_icmp',
 'service_eco_i',
 'service_private',
 'flag_SF']
```

In [31]:

```
X_newR2L = selector.fit_transform(X_R2L,Y_R2L)
X_newR2L.shape
```

Out[31]:

```
(68338, 13)
```

Get the features that were selected: R2L

In [32]:

```
true=selector.get_support()
newcolindex_R2L=[i for i, x in enumerate(true) if x]
newcolname_R2L=list( colNames[i] for i in newcolindex_R2L)
newcolname_R2L
```

Out[32]:

```
['src_bytes',
 'dst_bytes',
 'hot',
 'num_failed_logins',
 'is_guest_login',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp',
 'service_ftp_data',
 'service_http',
 'service_imap4',
 'flag_RSTO']
```

In [33]:

```
X_newU2R = selector.fit_transform(X_U2R,Y_U2R)
X_newU2R.shape
```

Out[33]:

```
(67395, 13)
```

Get the features that were selected: U2R

In [34]:

```
true=selector.get_support()
newcolindex_U2R=[i for i, x in enumerate(true) if x]
newcolname_U2R=list( colNames[i] for i in newcolindex_U2R)
newcolname_U2R
```

Out[34]:

```
['urgent',
 'hot',
 'root_shell',
 'num_file_creations',
 'num_shells',
 'srv_diff_host_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp_data',
 'service_http',
 'service_telnet']
```

Summary of features selected by Univariate Feature Selection

In [35]:

```
print('Features selected for DoS:',newcolname_DoS)
print()
print('Features selected for Probe:',newcolname_Probe)
print()
print('Features selected for R2L:',newcolname_R2L)
print()
print('Features selected for U2R:',newcolname_U2R)
```

```
Features selected for DoS: ['logged_in', 'count', 'serror_rate', 'srv_serr
or_rate', 'same_srv_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_ho
st_same_srv_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'se
rvice_http', 'flag_S0', 'flag_SF']

Features selected for Probe: ['logged_in', 'rerror_rate', 'srv_rerror_rat
e', 'dst_host_srv_count', 'dst_host_diff_srv_rate', 'dst_host_same_src_por
t_rate', 'dst_host_srv_diff_host_rate', 'dst_host_rerror_rate', 'dst_host_
srv_rerror_rate', 'Protocol_type_icmp', 'service_eco_i', 'service_privat
e', 'flag_SF']

Features selected for R2L: ['src_bytes', 'dst_bytes', 'hot', 'num_failed_l
ogins', 'is_guest_login', 'dst_host_srv_count', 'dst_host_same_src_port_ra
te', 'dst_host_srv_diff_host_rate', 'service_ftp', 'service_ftp_data', 'se
rvice_http', 'service_imap4', 'flag_RSTO']

Features selected for U2R: ['urgent', 'hot', 'root_shell', 'num_file_creat
ions', 'num_shells', 'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv
_count', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'se
rvice_ftp_data', 'service_http', 'service_telnet']
```

Build the model:

In [36]:

```
from sklearn.tree import DecisionTreeClassifier

# all features
clf_DoS=DecisionTreeClassifier(random_state=0)
clf_Probe=DecisionTreeClassifier(random_state=0)
clf_R2L=DecisionTreeClassifier(random_state=0)
clf_U2R=DecisionTreeClassifier(random_state=0)
clf_DoS.fit(X_DoS, Y_DoS)
clf_Probe.fit(X_Probe, Y_Probe)
clf_R2L.fit(X_R2L, Y_R2L)
clf_U2R.fit(X_U2R, Y_U2R)
```

Out[36]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

Prediction & Evaluation (validation):

Using all Features for each category

Confusion Matrices

DoS

In [37]:

```
# Apply the classifier we trained to the test data (which it has never seen before)
clf_DoS.predict(X_DoS_test)
```

Out[37]:

```
array([1, 1, 0, ..., 0, 1, 0], dtype=int64)
```

In [38]:

```
# View the predicted probabilities of the first 10 observations
clf_DoS.predict_proba(X_DoS_test)[0:10]
```

Out[38]:

```
array([[0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.]])
```

In [39]:

```
Y_DoS_pred=clf_DoS.predict(X_DoS_test)
# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[39]:

Predicted attacks	0	1
Actual attacks		
0	9494	217
1	2543	4917

Probe

In [40]:

```
Y_Probe_pred=clf_Probe.predict(X_Probe_test)
# Create confusion matrix
pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[40]:

Predicted attacks	0	2
Actual attacks		
0	8278	1433
2	428	1993

R2L

In [41]:

```
Y_R2L_pred=clf_R2L.predict(X_R2L_test)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[41]:

Predicted attacks	0	3
Actual attacks		
0	9710	1
3	2750	135

U2R

In [42]:

```
Y_U2R_pred=clf_U2R.predict(X_U2R_test)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[42]:

Predicted attacks	0	4
Actual attacks		
0	9707	4
4	54	13

Cross Validation: Accuracy, Precision, Recall, F-measure

DoS

In [43]:

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision'
)
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99645 (+/- 0.00343)
Precision: 0.99519 (+/- 0.00509)
Recall: 0.99665 (+/- 0.00483)
F-measure: 0.99591 (+/- 0.00395)
```

Probe

In [44]:

```
accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accur
acy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='prec
ision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_
macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99580 (+/- 0.00317)
Precision: 0.99397 (+/- 0.00678)
Recall: 0.99288 (+/- 0.00372)
F-measure: 0.99342 (+/- 0.00494)
```

R2L

In [45]:

```
accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_
macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro'
)
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.97960 (+/- 0.01008)
Precision: 0.97221 (+/- 0.01617)
Recall: 0.96995 (+/- 0.01386)
F-measure: 0.97105 (+/- 0.01422)
```

U2R

In [46]:

```
accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_
macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro'
)
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99663 (+/- 0.00259)
Precision: 0.86481 (+/- 0.08952)
Recall: 0.91672 (+/- 0.10661)
F-measure: 0.88628 (+/- 0.07462)
```