Intrusion Detection System

Approach

Step 1: Data preprocessing:

Data cleaning is done then all features are made numerical using one-Hot-encoding. The features are scaled to avoid features with large values that may weigh too much in the results.

Step 2: Feature Selection:

Eliminate redundant and irrelevant data by selecting a subset of relevant features that fully represents the given problem. Univariate feature selection with ANOVA F-test. This analyzes each feature individually to detemine the strength of the relationship between the feature and labels. Using SecondPercentile method (sklearn.feature_selection) to select features based on percentile of the highest scores. When this subset is found: Recursive Feature Elimination (RFE) is applied.

Step 4: Build the model:

Decision tree supervised learning classifier model is built.

Step 5: Prediction & Evaluation (validation):

Using the test data to make predictions of the model. Multiple scores are considered such as:accuracy,precision, recall, f1-score, confusion matrix. perform a 10-fold cross-validation.

Version Check for libraries

In [1]:
```
import pandas as pd
import numpy as np
import sys
import sklearn
import seaborn as sn
import matplotlib.pyplot as plt
print(pd.__version__)
print(np.__version__)
print(sys.version)
print(sklearn.__version__)
```

```
1.1.3
1.19.2
3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
0.23.2
```

Load the Dataset

In [2]:
```
# 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.csv & KDDTest.csv are the datasets
# these have already been removed.
df = pd.read_csv("KDDTrain.csv", header=None, names = col_names)
df_test = pd.read_csv("KDDTest.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 [3]:
```
# first five rows
df.head(5)
```

Out[3]:

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

5 rows × 42 columns

Summary of dataset

In [4]:
```
df.describe()
```

Out[4]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000	1
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.000111	
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.014366	
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000	

8 rows × 38 columns

Label Distribution of Training and Testing dataset

In [5]:
```
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
rootkit              13
xterm                13
teardrop             12
xlock                 9
land                  7
xsnoop                4
ftp_write             3
phf                   2
sqlattack             2
```

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

Step 1: Data preprocessing:

One-Hot-Encoding (one-of-K) is used to to transform all categorical features into binary features. Requirement for One-Hot-encoding: "The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature. It is assumed that input features take on values in the range [0, n_values)."

Therefore the features first need to be transformed with LabelEncoder, to transform every category to a number.

Identifying categorical features

In [6]:

```
# Columns that are categorical and not numerical yet: protocol_type (column 2), serv
# exploring 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

#Distribution of the feature service is, it is evenly distributed and therefore we n
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 [7]:

```
# 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
```

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

Dummies to be made for all categories as the distribution is

fairly even. In total: 3+70+11=84 dummies.

On Comparing it shows that the Test set has lesser categories (6), these need to be added as empty columns.

LabelEncoder

Inserting categorical features into a 2D numpy array

In [8]:
```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']
# 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[8]:

	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 [9]:
```
# 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)

#same method 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', 'ser
vice_X11', 'service_Z39_50', 'service_aol', 'service_auth', 'service_bgp', 'service_
courier', 'service_csnet_ns', 'service_ctf', 'service_daytime', 'service_discard',
'service_domain', 'service_domain_u', 'service_echo', 'service_eco_i', 'service_ecr_
i', 'service_efs', 'service_exec', 'service_finger', 'service_ftp', 'service_ftp_dat
a', 'service_gopher', 'service_harvest', 'service_hostnames', 'service_http', 'servi
ce_http_2784', 'service_http_443', 'service_http_8001', 'service_imap4', 'service_is
o_tsap', 'service_klogin', 'service_kshell', 'service_ldap', 'service_link', 'servic
e_login', 'service_mtp', 'service_name', 'service_netbios_dgm', 'service_netbios_n
```

```
s', 'service_netbios_ssn', 'service_netstat', 'service_nnsp', 'service_nntp', 'servi
ce_ntp_u', 'service_other', 'service_pm_dump', 'service_pop_2', 'service_pop_3', 'se
rvice_printer', 'service_private', 'service_red_i', 'service_remote_job', 'service_r
je', 'service_shell', 'service_smtp', 'service_sql_net', 'service_ssh', 'service_sun
rpc', 'service_supdup', 'service_systat', 'service_telnet', 'service_tftp_u', 'servi
ce_tim_i', 'service_time', 'service_urh_i', 'service_urp_i', 'service_uucp', 'servic
e_uucp_path', 'service_vmnet', 'service_whois', 'flag_OTH', 'flag_REJ', 'flag_RSTO',
'flag_RSTOS0', 'flag_RSTR', 'flag_S0', 'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF',
'flag_SH']
```

Transform categorical features into numbers using LabelEncoder()

In [10]:

```
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_tra
```

```
   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 [11]:

```
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=te

df_cat_data.head()
```

Out[11]:

	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	service_X11	service_Z39_5
0	0.0	1.0	0.0	0.0	0.0	0.
1	0.0	0.0	1.0	0.0	0.0	0.
2	0.0	1.0	0.0	0.0	0.0	0.
3	0.0	1.0	0.0	0.0	0.0	0.
4	0.0	1.0	0.0	0.0	0.0	0.

5 rows × 84 columns

Add 6 missing categories from train set to test set

In [12]:

```
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[12]:

```
['service_urh_i',
 'service_http_8001',
```

```
 'service_harvest',
 'service_http_2784',
 'service_red_i',
 'service_aol']
```

In [13]:
```
for col in difference:
    testdf_cat_data[col] = 0

testdf_cat_data.shape
```

Out[13]: (22544, 84)

Join encoded categorical dataframe with the non-categorical dataframe

In [14]:
```
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

Renaming every attack as label: normal=0, DoS=1, Probe=2, R2L=3 and U2R=4.

Replacing labels column with new labels column

In [15]:
```
# 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
                           'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'msc
                           ,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3
                           'buffer_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit':
newlabeldf_test=labeldf_test.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land
                           'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'msc
                           ,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3
                           'buffer_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit':
# 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 [16]:

```
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)
```

Step 2: Feature Scaling:

In [17]:

```
# Spliting 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
```

Store all list of feature names for further use (it is the same for every attack category). Column names are dropped at this stage.

In [18]:
```
colNames=list(X_DoS)
colNames_test=list(X_DoS_test)
```

Use StandardScaler() to scale the dataframes

In [19]:
```
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)
```

Check that the Standard Deviation is 1 for veifying proper feature scaling

In [20]:
```
print(X_DoS.std(axis=0))
```

```
[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. 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. 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 [21]:
```
X_Probe.std(axis=0);
X_R2L.std(axis=0);
X_U2R.std(axis=0);
```

Step 3: Feature Selection:

Univariate Feature Selection using ANOVA F-test

In [22]:
```
#univariate feature selection with ANOVA F-test is done first by using second-Percen
#SelectPercentile: removes all but a user-specified highest scoring percentage of fe

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
```

```
C:\Users\Aditya\anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_se
lection.py:114: UserWarning: Features [ 16  44  63  66  68  86 114] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
```

Out[22]: (113270, 13)

Features selected for : DoS

In [23]:
```
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[23]:
```
['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 [24]:
```
X_newProbe = selector.fit_transform(X_Probe,Y_Probe)
X_newProbe.shape
```

```
C:\Users\Aditya\anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_se
lection.py:114: UserWarning: Features [ 4 16] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
```

Out[24]:
```
(78999, 13)
```

Features selected for : Probe

In [25]:
```
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[25]:
```
['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 [26]:
```
X_newR2L = selector.fit_transform(X_R2L,Y_R2L)
X_newR2L.shape
```

```
C:\Users\Aditya\anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_se
lection.py:114: UserWarning: Features [  4  16  43  44  46  47  48  49  50  51  54
57  58  62  63  64  66  67
  68  70  71  72  73  74  76  77  78  79  80  81  82  83  86  87  89  92
  93  96  98  99 100 107 108 109 110 114] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
```

Out[26]:
```
(68338, 13)
```

Features selected for : R2L

In [27]:
```
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[27]:
```
['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 [28]:
```
X_newU2R = selector.fit_transform(X_U2R,Y_U2R)
X_newU2R.shape
```

```
C:\Users\Aditya\anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_se
lection.py:114: UserWarning: Features [  4  16  43  44  46  47  48  49  50  51  54
57  58  62  63  64  66  67
  68  70  71  72  73  74  75  76  77  78  79  80  81  82  83  86  87  89
  92  93  96  98  99 100 107 108 109 110 114] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
```

Out[28]:
```
(67395, 13)
```

Features selected for : U2R

In [29]:
```
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[29]:
```
['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 all features selected by Univariate Feature Selection

In [30]:
```
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_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']

Features selected for Probe: ['logged_in', 'rerror_rate', 'srv_rerror_rate', 'dst_ho
st_srv_count', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_sr
v_diff_host_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'Protocol_typ
e_icmp', 'service_eco_i', 'service_private', 'flag_SF']

Features selected for R2L: ['src_bytes', 'dst_bytes', 'hot', 'num_failed_logins', 'i
s_guest_login', 'dst_host_srv_count', 'dst_host_same_src_port_rate', 'dst_host_srv_d
iff_host_rate', 'service_ftp', 'service_ftp_data', 'service_http', 'service_imap4',
'flag_RSTO']

Features selected for U2R: ['urgent', 'hot', 'root_shell', 'num_file_creations', 'nu
m_shells', 'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_host_s
ame_src_port_rate', 'dst_host_srv_diff_host_rate', 'service_ftp_data', 'service_htt
p', 'service_telnet']
```

To reduce to a subset feature the second option is considered as this uses Recursive feature elimination algorithm. The number of features for every attack category considered to be is 13.

Recursive Feature Elimination for feature ranking (1. feature has highest importance)

In [31]:

```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
# Create a decision tree classifier. By convention, clf means 'classifier'
clf = DecisionTreeClassifier(random_state=0)

#rank all features, i.e continue the elimination until the last one
rfe = RFE(clf, n_features_to_select=1)
rfe.fit(X_newDoS, Y_DoS)
print ("DoS Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_DoS)))
```

```
DoS Features sorted by their rank:
[(1, 'same_srv_rate'), (2, 'count'), (3, 'flag_SF'), (4, 'dst_host_serror_rate'),
(5, 'dst_host_same_srv_rate'), (6, 'dst_host_srv_count'), (7, 'dst_host_count'), (8,
'logged_in'), (9, 'serror_rate'), (10, 'dst_host_srv_serror_rate'), (11, 'srv_serror
_rate'), (12, 'service_http'), (13, 'flag_S0')]
```

In [32]:

```
rfe.fit(X_newProbe, Y_Probe)
print ("Probe Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_Probe)))
```

```
Probe Features sorted by their rank:
[(1, 'dst_host_same_src_port_rate'), (2, 'dst_host_srv_count'), (3, 'dst_host_rerror
_rate'), (4, 'service_private'), (5, 'logged_in'), (6, 'dst_host_diff_srv_rate'),
(7, 'dst_host_srv_diff_host_rate'), (8, 'flag_SF'), (9, 'service_eco_i'), (10, 'rerr
or_rate'), (11, 'Protocol_type_icmp'), (12, 'dst_host_srv_rerror_rate'), (13, 'srv_r
error_rate')]
```

In [33]:

```
rfe.fit(X_newR2L, Y_R2L)

print ("R2L Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_R2L)))
```

```
R2L Features sorted by their rank:
[(1, 'src_bytes'), (2, 'dst_bytes'), (3, 'hot'), (4, 'dst_host_srv_diff_host_rate'),
(5, 'service_ftp_data'), (6, 'dst_host_same_src_port_rate'), (7, 'dst_host_srv_coun
```

```
t'), (8, 'num_failed_logins'), (9, 'service_imap4'), (10, 'is_guest_login'), (11, 's
ervice_ftp'), (12, 'flag_RSTO'), (13, 'service_http')]
```

In [34]:
```
rfe.fit(X_newU2R, Y_U2R)

print ("U2R Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newcolname_U2R)))
```

```
U2R Features sorted by their rank:
[(1, 'hot'), (2, 'dst_host_srv_count'), (3, 'dst_host_count'), (4, 'root_shell'),
(5, 'num_shells'), (6, 'service_ftp_data'), (7, 'dst_host_srv_diff_host_rate'), (8,
'num_file_creations'), (9, 'dst_host_same_src_port_rate'), (10, 'service_telnet'),
(11, 'srv_diff_host_rate'), (12, 'service_http'), (13, 'urgent')]
```

2. Recursive Feature Elimination, select 13 best features using Decision tree classifier

In [35]:
```
from sklearn.feature_selection import RFE
clf = DecisionTreeClassifier(random_state=0)
rfe = RFE(estimator=clf, n_features_to_select=13, step=1)
rfe.fit(X_DoS, Y_DoS)
X_rfeDoS=rfe.transform(X_DoS)
true=rfe.support_
rfecolindex_DoS=[i for i, x in enumerate(true) if x]
rfecolname_DoS=list(colNames[i] for i in rfecolindex_DoS)
```

In [36]:
```
rfe.fit(X_Probe, Y_Probe)
X_rfeProbe=rfe.transform(X_Probe)
true=rfe.support_
rfecolindex_Probe=[i for i, x in enumerate(true) if x]
rfecolname_Probe=list(colNames[i] for i in rfecolindex_Probe)
```

In [37]:
```
rfe.fit(X_R2L, Y_R2L)
X_rfeR2L=rfe.transform(X_R2L)
true=rfe.support_
rfecolindex_R2L=[i for i, x in enumerate(true) if x]
rfecolname_R2L=list(colNames[i] for i in rfecolindex_R2L)
```

In [38]:
```
rfe.fit(X_U2R, Y_U2R)
X_rfeU2R=rfe.transform(X_U2R)
true=rfe.support_
rfecolindex_U2R=[i for i, x in enumerate(true) if x]
rfecolname_U2R=list(colNames[i] for i in rfecolindex_U2R)
```

Summary of features selected by RFE with decision tree classifier

In [39]:
```
print('Features selected for DoS:',rfecolname_DoS)
print()
print('Features selected for Probe:',rfecolname_Probe)
print()
print('Features selected for R2L:',rfecolname_R2L)
print()
print('Features selected for U2R:',rfecolname_U2R)
```

```
Features selected for DoS: ['src_bytes', 'dst_bytes', 'wrong_fragment', 'num_comprom
ised', 'same_srv_rate', 'diff_srv_rate', 'dst_host_count', 'dst_host_same_srv_rate',
'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'service_ecr_i', 'flag_RSTR', 'f
lag_S0']
```

```
Features selected for Probe: ['src_bytes', 'dst_bytes', 'rerror_rate', 'dst_host_sam
e_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_rerr
or_rate', 'service_finger', 'service_ftp_data', 'service_http', 'service_private',
'service_smtp', 'service_telnet']

Features selected for R2L: ['duration', 'src_bytes', 'dst_bytes', 'hot', 'num_failed
_logins', 'num_access_files', 'dst_host_count', 'dst_host_srv_count', 'dst_host_same
_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'service_f
tp_data', 'service_imap4']

Features selected for U2R: ['duration', 'src_bytes', 'dst_bytes', 'hot', 'root_shel
l', 'num_file_creations', 'num_shells', 'srv_count', 'dst_host_count', 'dst_host_sam
e_srv_rate', 'dst_host_srv_diff_host_rate', 'service_ftp_data', 'service_other']
```

In [40]:

```
print(X_rfeDoS.shape)
print(X_rfeProbe.shape)
print(X_rfeR2L.shape)
print(X_rfeU2R.shape)
```

```
(113270, 13)
(78999, 13)
(68338, 13)
(67395, 13)
```

Step 4: Building the classifier model : Decision tree

Decision Tree Classifier is trained for all features and for reduced relevant set of features, for comparison.

The classifier model itself is stored in the clf variable.

In [41]:

```
# 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[41]:

```
DecisionTreeClassifier(random_state=0)
```

In [42]:

```
# selected features
clf_rfeDoS=DecisionTreeClassifier(random_state=0)
clf_rfeProbe=DecisionTreeClassifier(random_state=0)
clf_rfeR2L=DecisionTreeClassifier(random_state=0)
clf_rfeU2R=DecisionTreeClassifier(random_state=0)
clf_rfeDoS.fit(X_rfeDoS, Y_DoS)
clf_rfeProbe.fit(X_rfeProbe, Y_Probe)
clf_rfeR2L.fit(X_rfeR2L, Y_R2L)
clf_rfeU2R.fit(X_rfeU2R, Y_U2R)
```

Out[42]:

```
DecisionTreeClassifier(random_state=0)
```

Step 5: Prediction & Evaluation (validation):

1. Using all features decision tree model

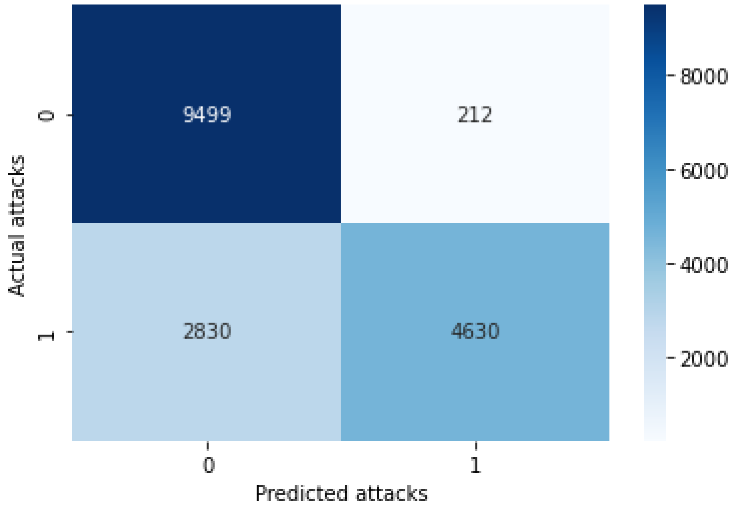
evaluation.

In [43]:
```
# # Apply the classifier we trained to the test data (which it has never seen before
# print(clf_DoS.predict(X_DoS_test))

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

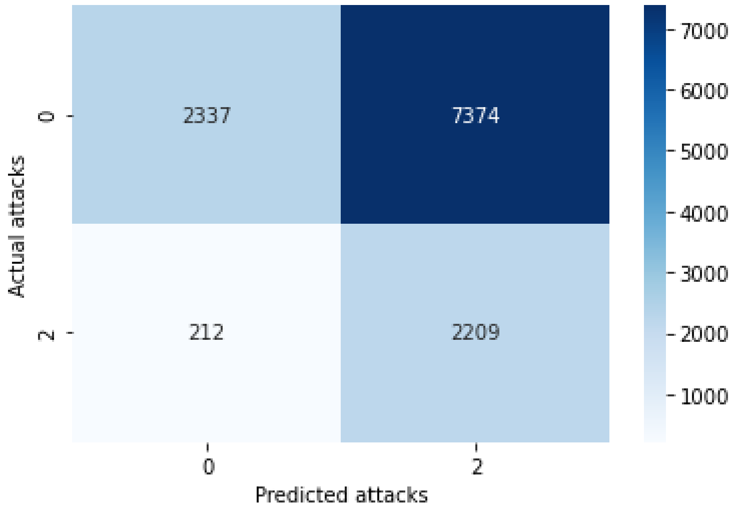
1. a) Confusion matrices

In [44]:
```
# Creating Confusion matrix

Y_DoS_pred=clf_DoS.predict(X_DoS_test)
cm1_DoS=pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['

Y_Probe_pred=clf_Probe.predict(X_Probe_test)
cm1_Probe=pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colna

Y_R2L_pred=clf_R2L.predict(X_R2L_test)
cm1_R2L=pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['

Y_U2R_pred=clf_U2R.predict(X_U2R_test)
cm1_U2R=pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['

print("Plotting Confusion Matrices of Decision tree classifier model on all features
print("")

print("*** For DoS Confusion Matrix ***")
print("")

sn.heatmap(cm1_DoS, annot=True, cmap= 'Blues', fmt='d')
plt.show()

print("")
print("")
print("*** For Probe Confusion Matrix ***")
print("")

sn.heatmap(cm1_Probe, annot=True, cmap= 'Blues', fmt='d')
plt.show()

print("")
print("")
print("*** For R2L Confusion Matrix ***")
print("")

sn.heatmap(cm1_R2L, annot=True, cmap= 'Blues', fmt='d')
plt.show()

print("")
print("")
print("*** For U2R Confusion Matrix ***")
print("")

sn.heatmap(cm1_U2R, annot=True, cmap= 'Blues', fmt='d')
plt.show()
```

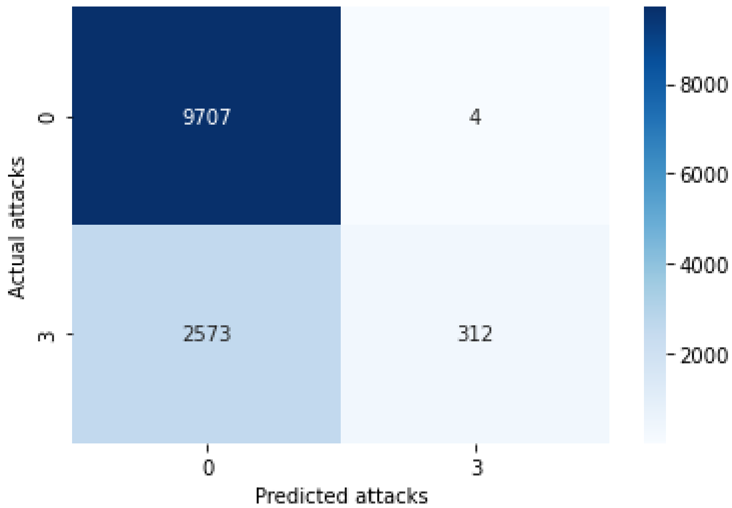
```
Plotting Confusion Matrices of Decision tree classifier model on all features for ea
ch category

*** For DoS Confusion Matrix ***
```



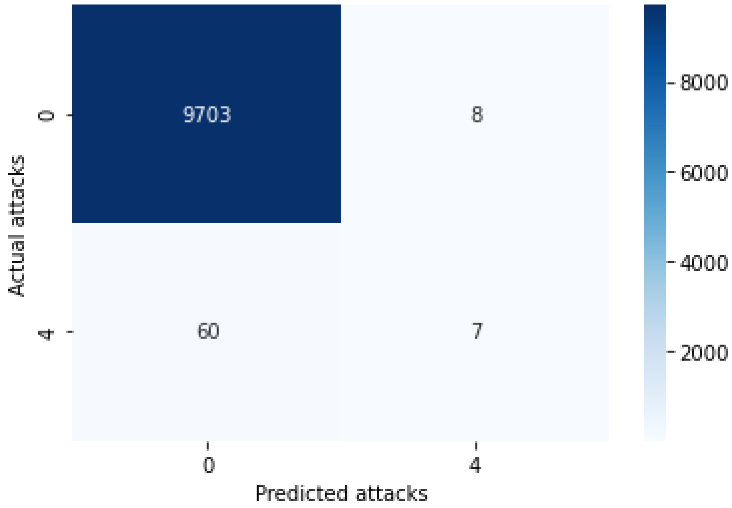

```
*** For Probe Confusion Matrix ***
```




```
*** For R2L Confusion Matrix ***
```




```
*** For U2R Confusion Matrix ***
```



1. b) Performance Metrics: Accuracy, Precision, Recall, F-measure

In [45]:
```
from sklearn.model_selection import cross_val_score
from sklearn import metrics

print("*** For DoS All Performance metrics : Accuracy, Precision, Recall, F-measure
print("")

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='precisi
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))

print("")
print("")
print("*** For Probe All Performance metrics : Accuracy, Precision, Recall, F-measur
print("")

accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='ac
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='p
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='reca
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))

print("")
print("")
print("*** For R2L All Performance metrics : Accuracy, Precision, Recall, F-measure
print("")

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='precisi
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_mac
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))
```

```
print("")
print("")
print("*** For U2R All Performance metrics : Accuracy, Precision, Recall, F-measure
print("")

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='precisi
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_mac
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))
```

```
*** For DoS All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.99639 (+/- 0.00341)
Precision: 0.99505 (+/- 0.00477)
Recall: 0.99665 (+/- 0.00483)
F-measure: 0.99585 (+/- 0.00392)


*** For Probe All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.99571 (+/- 0.00328)
Precision: 0.99392 (+/- 0.00684)
Recall: 0.99267 (+/- 0.00405)
F-measure: 0.99329 (+/- 0.00512)


*** For R2L All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.97920 (+/- 0.01053)
Precision: 0.97151 (+/- 0.01736)
Recall: 0.96958 (+/- 0.01379)
F-measure: 0.97051 (+/- 0.01478)


*** For U2R All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.99652 (+/- 0.00228)
Precision: 0.86295 (+/- 0.08961)
Recall: 0.90958 (+/- 0.09211)
F-measure: 0.88210 (+/- 0.06559)
```

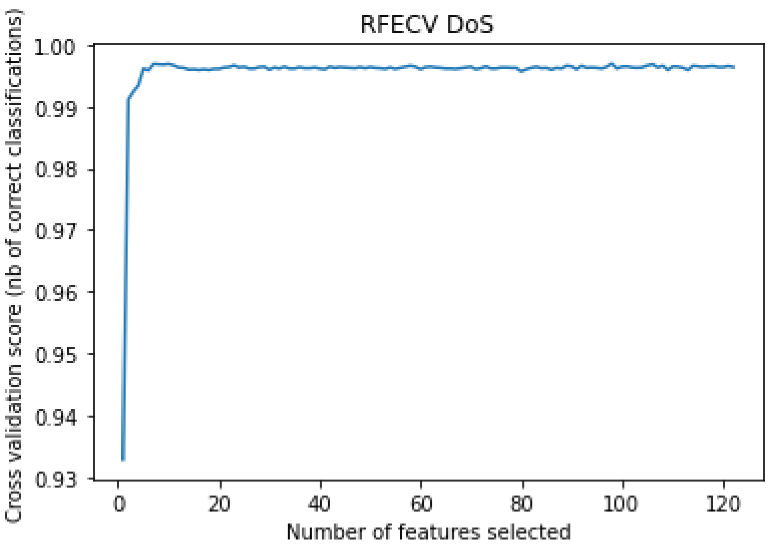
1. c) Recursive feature elimination cross validation for illustration

In [47]:

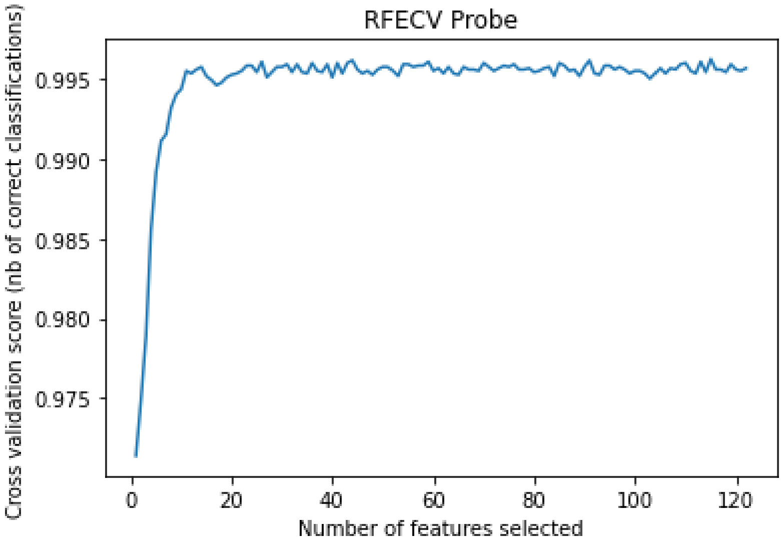
```
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold

# Plot number of features VS. cross-validation scores

print("*** Plotting of RFECV for DOS attack category ***")
print("")

rfecv_DoS = RFECV(estimator=clf_DoS, step=1, cv=10, scoring='accuracy')
rfecv_DoS.fit(X_DoS_test, Y_DoS_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV DoS')
plt.plot(range(1, len(rfecv_DoS.grid_scores_) + 1), rfecv_DoS.grid_scores_)
```

```
plt.show()

print("")
print("*** Plotting of RFECV for Probe attack category ***")
print("")

rfecv_Probe = RFECV(estimator=clf_Probe, step=1, cv=10, scoring='accuracy')
rfecv_Probe.fit(X_Probe_test, Y_Probe_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV Probe')
plt.plot(range(1, len(rfecv_Probe.grid_scores_) + 1), rfecv_Probe.grid_scores_)
plt.show()

print("")
print("*** Plotting of RFECV for R2L attack category ***")
print("")

rfecv_R2L = RFECV(estimator=clf_R2L, step=1, cv=10, scoring='accuracy')
rfecv_R2L.fit(X_R2L_test, Y_R2L_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV R2L')
plt.plot(range(1, len(rfecv_R2L.grid_scores_) + 1), rfecv_R2L.grid_scores_)
plt.show()

print("")
print("*** Plotting of RFECV for U2R attack category ***")
print("")

rfecv_U2R = RFECV(estimator=clf_U2R, step=1, cv=10, scoring='accuracy')
rfecv_U2R.fit(X_U2R_test, Y_U2R_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV U2R')
plt.plot(range(1, len(rfecv_U2R.grid_scores_) + 1), rfecv_U2R.grid_scores_)
plt.show()
```

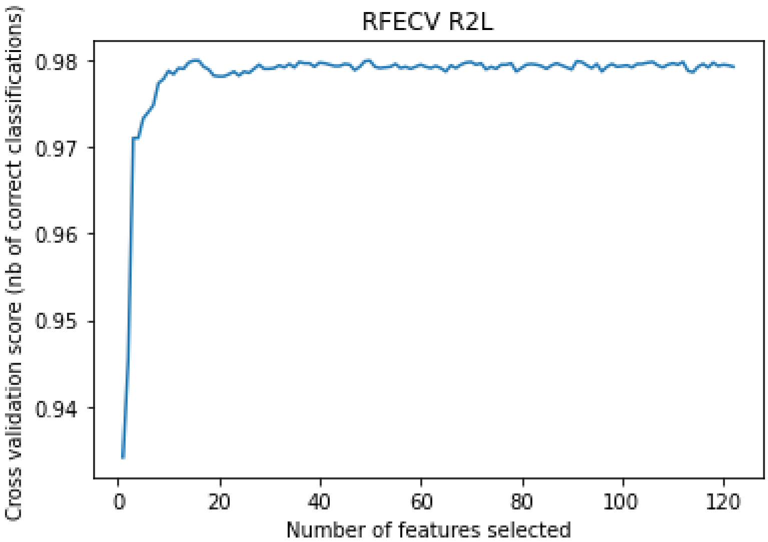
```
*** Plotting of RFECV for DOS attack category ***
```



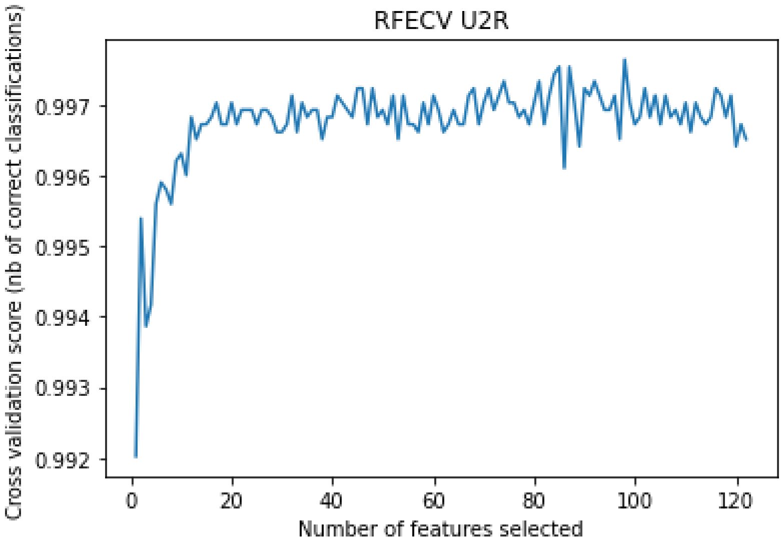

```
*** Plotting of RFECV for Probe attack category ***
```




```
*** Plotting of RFECV for R2L attack category ***
```




```
*** Plotting of RFECV for U2R attack category ***
```



2. Using selected 13 RFE features decision tree model evaluation.

In [48]:

```
# reduce test dataset to 13 features, use only features described in rfecolname_DoS
```

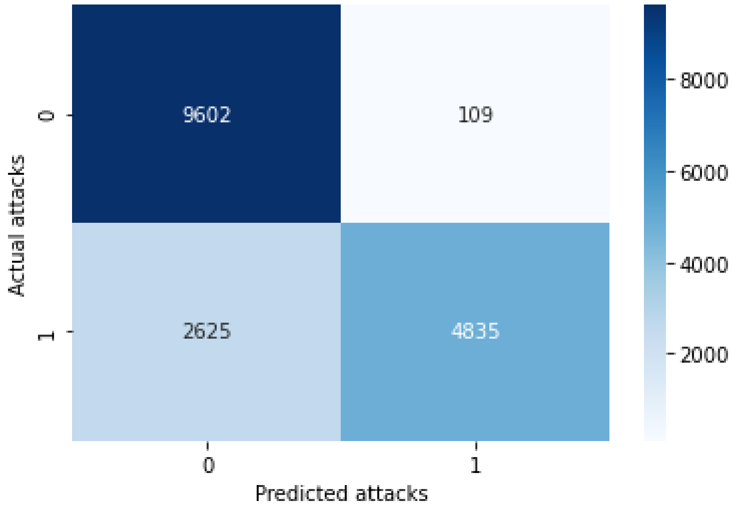
```
X_DoS_test2=X_DoS_test[:,rfecolindex_DoS]
X_Probe_test2=X_Probe_test[:,rfecolindex_Probe]
X_R2L_test2=X_R2L_test[:,rfecolindex_R2L]
X_U2R_test2=X_U2R_test[:,rfecolindex_U2R]
X_U2R_test2.shape
```

Out[48]:
```
(9778, 13)
```

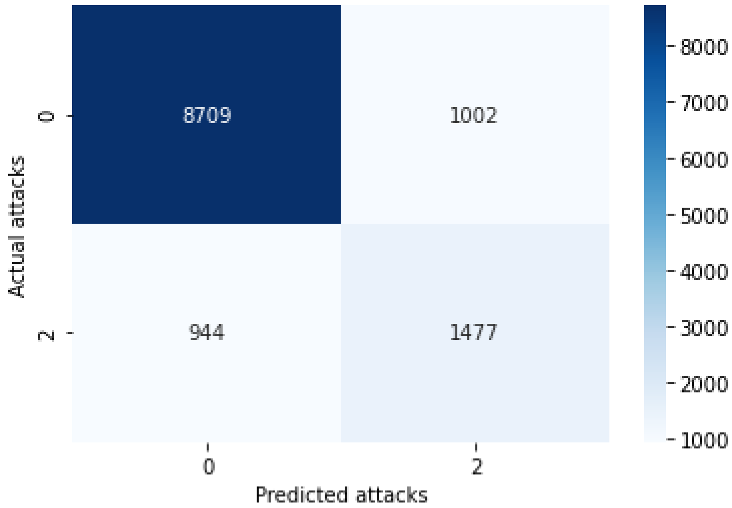
2. a) Confusion matrices

In [51]:
```
# Creating Confusion matrix

Y_DoS_pred2=clf_rfeDoS.predict(X_DoS_test2)
cm2_rfeDoS=pd.crosstab(Y_DoS_test, Y_DoS_pred2, rownames=['Actual attacks'], colname

Y_Probe_pred2=clf_rfeProbe.predict(X_Probe_test2)
cm2_rfeProbe=pd.crosstab(Y_Probe_test, Y_Probe_pred2, rownames=['Actual attacks'], c

Y_R2L_pred2=clf_rfeR2L.predict(X_R2L_test2)
cm2_rfeR2L=pd.crosstab(Y_R2L_test, Y_R2L_pred2, rownames=['Actual attacks'], colname

Y_U2R_pred2=clf_rfeU2R.predict(X_U2R_test2)
cm2_rfeU2R=pd.crosstab(Y_U2R_test, Y_U2R_pred2, rownames=['Actual attacks'], colname

print("Plotting Confusion Matrices of Decision tree classifier model on selected 13
print("")

print("*** For DoS Confusion Matrix ***")
print("")

sn.heatmap(cm2_rfeDoS, annot=True, cmap= 'Blues', fmt='d')
plt.show()

print("")
print("")
print("*** For Probe Confusion Matrix ***")
print("")

sn.heatmap(cm2_rfeProbe, annot=True, cmap= 'Blues', fmt='d')
plt.show()

print("")
print("")
print("*** For R2L Confusion Matrix ***")
print("")

sn.heatmap(cm2_rfeR2L, annot=True, cmap= 'Blues', fmt='d')
plt.show()

print("")
print("")
print("*** For U2R Confusion Matrix ***")
print("")

sn.heatmap(cm2_rfeU2R, annot=True, cmap= 'Blues', fmt='d')
plt.show()
```

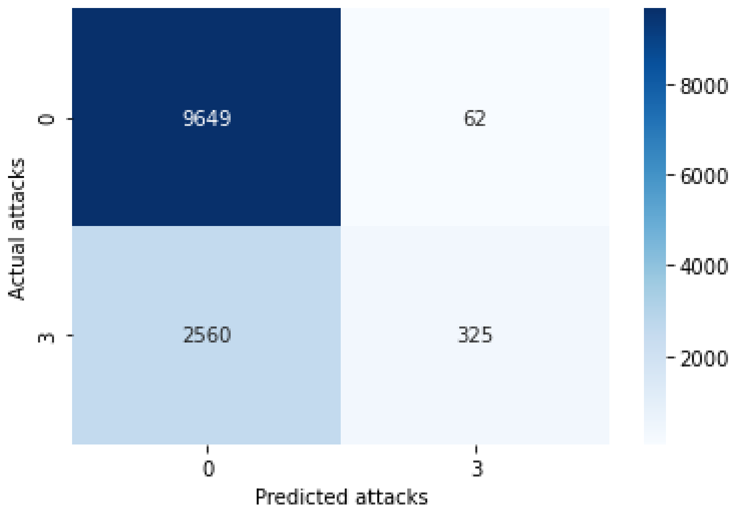
```
Plotting Confusion Matrices of Decision tree classifier model on selected 13 feature
s for each category

*** For DoS Confusion Matrix ***
```



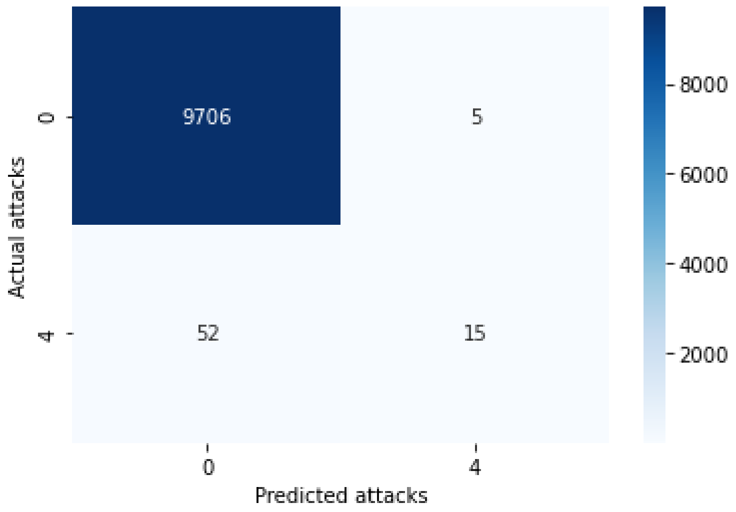

```
*** For Probe Confusion Matrix ***
```




```
*** For R2L Confusion Matrix ***
```




```
*** For U2R Confusion Matrix ***
```



2. b) Performance metrics : Accuracy, Precision, Recall, F-measure

In [52]:

```
print("*** For DoS All Performance metrics : Accuracy, Precision, Recall, F-measure
print("")

accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='accu
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='pre
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='recall
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

print("")
print("")
print("*** For Probe All Performance metrics : Accuracy, Precision, Recall, F-measur
print("")

accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scorin
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='f1_ma
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

print("")
print("")
print("*** For R2L All Performance metrics : Accuracy, Precision, Recall, F-measure
print("")

accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='accu
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='pre
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='recall
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

print("")
print("")
```

```
print("*** For U2R All Performance metrics : Accuracy, Precision, Recall, F-measure
print("")

accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='accu
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='pre
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='recall
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
*** For DoS All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.99738 (+/- 0.00267)
Precision: 0.99692 (+/- 0.00492)
Recall: 0.99705 (+/- 0.00356)
F-measure: 0.99698 (+/- 0.00307)


*** For Probe All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.99085 (+/- 0.00559)
Precision: 0.98674 (+/- 0.01179)
Recall: 0.98467 (+/- 0.01026)
F-measure: 0.98566 (+/- 0.00871)


*** For R2L All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.97459 (+/- 0.00910)
Precision: 0.96689 (+/- 0.01311)
Recall: 0.96086 (+/- 0.01571)
F-measure: 0.96379 (+/- 0.01305)


*** For U2R All Performance metrics : Accuracy, Precision, Recall, F-measure ***

Accuracy: 0.99652 (+/- 0.00278)
Precision: 0.87538 (+/- 0.15433)
Recall: 0.89540 (+/- 0.14777)
F-measure: 0.87731 (+/- 0.09647)
```

2. c) Cross Validation for 2, 5, 10, 30, 50 folds for each category of attacks.

In [53]:

```
print("*** For DoS Accuracy on k-folds cross validation ***")
print("")
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=2, scoring='accur
print("1. Accuracy on 2 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=5, scoring='accur
print("2. Accuracy on 5 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='accu
print("3. Accuracy on 10 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=30, scoring='accu
print("4. Accuracy on 30 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=50, scoring='accu
print("5. Accuracy on 50 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

print("")
print("")
```

```
print("*** For Probe Accuracy on k-folds cross validation ***")
print("")

accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=2, scoring=
print("1. Accuracy on 2 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=5, scoring=
print("2. Accuracy on 5 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring
print("3. Accuracy on 10 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=30, scoring
print("4. Accuracy on 30 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=50, scoring
print("5. Accuracy on 50 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

print("")
print("")
print("*** For R2L Accuracy on k-folds cross validation ***")
print("")

accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=2, scoring='accur
print("1. Accuracy on 2 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=5, scoring='accur
print("2. Accuracy on 5 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='accu
print("3. Accuracy on 10 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=30, scoring='accu
print("4. Accuracy on 30 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=50, scoring='accu
print("5. Accuracy on 50 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

print("")
print("")
print("*** For U2R Accuracy on k-folds cross validation ***")
print("")

accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=2, scoring='accur
print("1. Accuracy on 2 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=5, scoring='accur
print("2. Accuracy on 5 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std

accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='accu
print("3. Accuracy on 10 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=30, scoring='accu
print("4. Accuracy on 30 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st

accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=50, scoring='accu
print("5. Accuracy on 50 fold cv: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.st
```

```
*** For DoS Accuracy on k-folds cross validation ***

1. Accuracy on 2 fold cv: 0.99662 (+/- 0.00116)
2. Accuracy on 5 fold cv: 0.99709 (+/- 0.00064)
3. Accuracy on 10 fold cv: 0.99738 (+/- 0.00267)
4. Accuracy on 30 fold cv: 0.99726 (+/- 0.00430)
5. Accuracy on 50 fold cv: 0.99703 (+/- 0.00622)
```

```
*** For Probe Accuracy on k-folds cross validation ***

1. Accuracy on 2 fold cv: 0.99060 (+/- 0.00165)
2. Accuracy on 5 fold cv: 0.99093 (+/- 0.00233)
3. Accuracy on 10 fold cv: 0.99085 (+/- 0.00559)
4. Accuracy on 30 fold cv: 0.99118 (+/- 0.00742)
5. Accuracy on 50 fold cv: 0.99085 (+/- 0.01122)


*** For R2L Accuracy on k-folds cross validation ***

1. Accuracy on 2 fold cv: 0.97118 (+/- 0.00143)
2. Accuracy on 5 fold cv: 0.97388 (+/- 0.00624)
3. Accuracy on 10 fold cv: 0.97459 (+/- 0.00910)
4. Accuracy on 30 fold cv: 0.97467 (+/- 0.01644)
5. Accuracy on 50 fold cv: 0.97523 (+/- 0.01795)


*** For U2R Accuracy on k-folds cross validation ***

1. Accuracy on 2 fold cv: 0.99519 (+/- 0.00184)
2. Accuracy on 5 fold cv: 0.99714 (+/- 0.00153)
3. Accuracy on 10 fold cv: 0.99652 (+/- 0.00278)
4. Accuracy on 30 fold cv: 0.99693 (+/- 0.00571)
5. Accuracy on 50 fold cv: 0.99662 (+/- 0.00755)
```

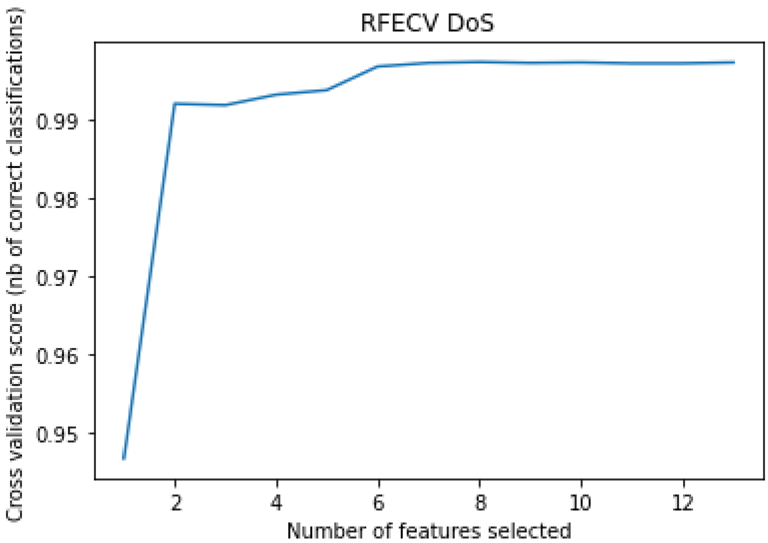
2. d) Recursive feature elimination cross validation for illustration

In [54]:

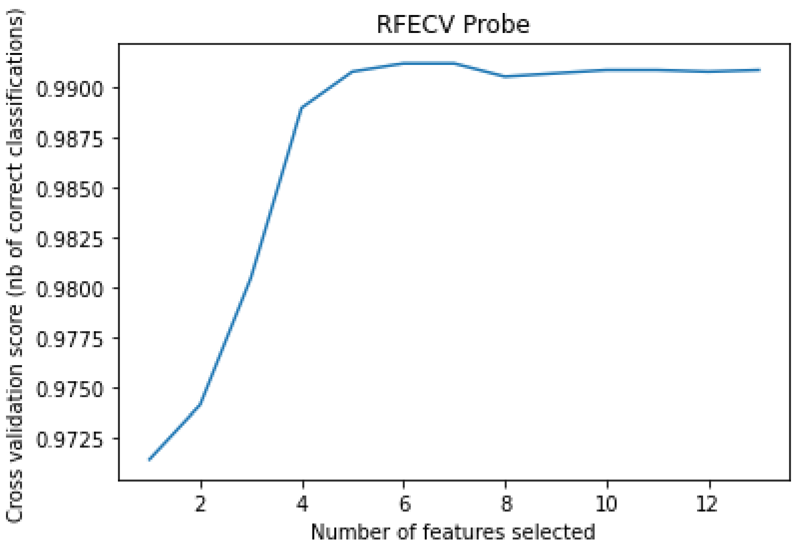
```
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold

# Plot number of features VS. cross-validation scores

print("*** Plotting of RFECV for DOS attack category ***")
print("")

rfecv_rfeDoS = RFECV(estimator=clf_rfeDoS, step=1, cv=10, scoring='accuracy')
rfecv_rfeDoS.fit(X_DoS_test2, Y_DoS_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV DoS')
plt.plot(range(1, len(rfecv_rfeDoS.grid_scores_) + 1), rfecv_rfeDoS.grid_scores_)
plt.show()

print("")
print("*** Plotting of RFECV for Probe attack category ***")
print("")

rfecv_rfeProbe = RFECV(estimator=clf_rfeProbe, step=1, cv=10, scoring='accuracy')
rfecv_rfeProbe.fit(X_Probe_test2, Y_Probe_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV Probe')
plt.plot(range(1, len(rfecv_rfeProbe.grid_scores_) + 1), rfecv_rfeProbe.grid_scores_
plt.show()

print("")
print("*** Plotting of RFECV for R2L attack category ***")
```

```
print("")

rfecv_rfeR2L = RFECV(estimator=clf_rfeR2L, step=1, cv=10, scoring='accuracy')
rfecv_rfeR2L.fit(X_R2L_test2, Y_R2L_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV R2L')
plt.plot(range(1, len(rfecv_rfeR2L.grid_scores_) + 1), rfecv_rfeR2L.grid_scores_)
plt.show()

print("")
print("*** Plotting of RFECV for U2R attack category ***")
print("")

rfecv_rfeU2R = RFECV(estimator=clf_rfeU2R, step=1, cv=10, scoring='accuracy')
rfecv_rfeU2R.fit(X_U2R_test2, Y_U2R_test)

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV U2R')
plt.plot(range(1, len(rfecv_rfeU2R.grid_scores_) + 1), rfecv_rfeU2R.grid_scores_)
plt.show()
```

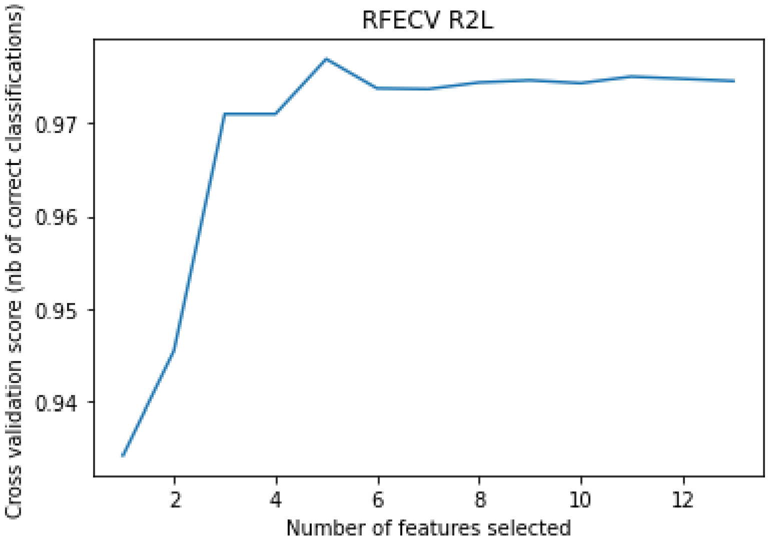
```
*** Plotting of RFECV for DOS attack category ***
```



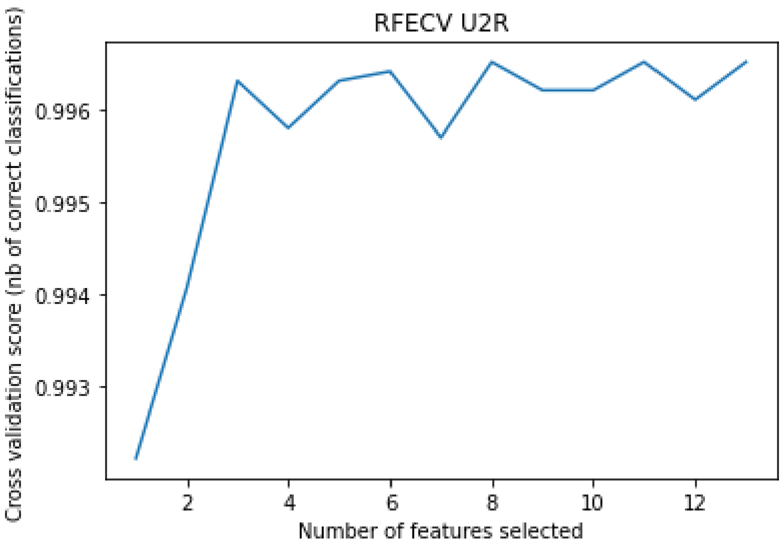

```
*** Plotting of RFECV for Probe attack category ***
```



```
*** Plotting of RFECV for R2L attack category ***
```




```
*** Plotting of RFECV for U2R attack category ***
```




```
In [ ]:
```