> **Essential ML process for Intrusion Detection**

`python 3.7.13 scikit-learn 1.0.2`
`numpy 1.19.5 pandas 1.3.5`

Import the main libraries

In [1]:
```
import numpy
import pandas

import os
data_path = '../datasets'
```

Import the Dataset

In [2]:
```
# Using boosted Train and preprocessed Test

data_file = os.path.join(data_path, 'NSL_ppTrain.csv')
train_df = pandas.read_csv(data_file)
print('Train Dataset: {} rows, {} columns'.format(train_df.shape[0], train_df.shape

data_file = os.path.join(data_path, 'NSL_ppTest.csv')
test_df = pandas.read_csv(data_file)
print('Test Dataset: {} rows, {} columns'.format(test_df.shape[0], test_df.shape[1]
```

```
Train Dataset: 125973 rows, 43 columns
Test Dataset: 22544 rows, 43 columns
```

Data Preparation and EDA (consistency checks)

- *Check column names of numeric attributes*

In [3]:
```
trnn = train_df.select_dtypes(include=['float64','int64']).columns
tstn = test_df.select_dtypes(include=['float64','int64']).columns
trndif = numpy.setdiff1d(trnn, tstn)
tstdif = numpy.setdiff1d(tstn, trnn)

print("Numeric features in the train_set that are not in the test_set: ",end='')
if len(trndif) > 0:
    print('\n',trndif)
else:
    print('None')

print("Numeric features in the test_set that are not in the train_set: ",end='')
if len(tstdif) > 0:
    print('\n',tstdif)
else:
    print('None')

print()
# correct any differences here
```

```
Numeric features in the train_set that are not in the test_set: None
Numeric features in the test_set that are not in the train_set: None
```

- *Check column names of categorical attributes*

In [4]:
```
trnn = train_df.select_dtypes(include=['object']).columns
tstn = test_df.select_dtypes(include=['object']).columns
trndif = numpy.setdiff1d(trnn, tstn)
tstdif = numpy.setdiff1d(tstn, trnn)

print("Categorical features in the train_set that are not in the test_set: ",end=''
```

```
if len(trndif) > 0:
    print('\n',trndif)
else:
    print('None')

print("Categorical features in the test_set that are not in the train_set: ",end=''
if len(tstdif) > 0:
    print('\n\t',tstdif)
else:
    print('None')

print()
# correct any differences here
```

```
Categorical features in the train_set that are not in the test_set: None
Categorical features in the test_set that are not in the train_set: None
```

- *Drop columns with only one value*

In [5]:
```
n_eq_one = []
for col in train_df.columns:
    lctrn = len(train_df[col].unique())
    lctst = len(test_df[col].unique())
    if (lctrn == 1) and (lctrn == lctst):
        n_eq_one.append(train_df[col].name)

if len(n_eq_one) > 0:
    print('Dropping single-valued features')
    print(n_eq_one)
    train_df.drop(n_eq_one, axis=1, inplace=True)
    test_df.drop(n_eq_one, axis=1, inplace=True)
```

```
Dropping single-valued features
['num_outbound_cmds']
```

- _Check categorical feature values:

differences will be resolved by one-hot encoding the combined test and train sets_

In [6]:
```
trnn = train_df.select_dtypes(include=['object']).columns
for col in trnn:
    tr = train_df[col].unique()
    ts = test_df[col].unique()
    trd = numpy.setdiff1d(tr, ts)
    tsd = numpy.setdiff1d(ts, tr)

    print(col,'::> ')
    print("\tUnique text values in the train_set that are not in the test_set: ",en
    if len(trd) > 0:
        print('\n\t',trd)
    else:
        print('None')

    print("\tUnique text values in the test_set that are not in the train_set: ",en
    if len(tsd) > 0:
        print('\n\t',tsd)
    else:
        print('None')
```

```
protocol_type ::>
        Unique text values in the train_set that are not in the test_set: None
        Unique text values in the test_set that are not in the train_set: None
service ::>
        Unique text values in the train_set that are not in the test_set:
         ['aol' 'harvest' 'http_2784' 'http_8001' 'red_i' 'urh_i']
        Unique text values in the test_set that are not in the train_set: None
flag ::>
        Unique text values in the train_set that are not in the test_set: None
        Unique text values in the test_set that are not in the train_set: None
label ::>
        Unique text values in the train_set that are not in the test_set:
         ['spy' 'warezclient']
        Unique text values in the test_set that are not in the train_set:
         ['apache2' 'httptunnel' 'mailbomb' 'mscan' 'named' 'processtable' 'ps'
 'saint' 'sendmail' 'snmpgetattack' 'snmpguess' 'sqlattack' 'udpstorm'
 'worm' 'xlock' 'xsnoop' 'xterm']
atakcat ::>
        Unique text values in the train_set that are not in the test_set: None
        Unique text values in the test_set that are not in the train_set: None
```

- *Combine for processing classification target and text features*

In [7]:
```
combined_df = pandas.concat([train_df, test_df])
print('Combined Dataset: {} rows, {} columns'.format(
    combined_df.shape[0], combined_df.shape[1]))
```

```
Combined Dataset: 148517 rows, 42 columns
```

In [8]:
```
# Classification Target feature:
# two columns of labels are available
#    * Two-class: labels     * Multiclass: atakcat

# Two-class: Reduce the detailed attack labels to 'normal' or 'attack'
labels_df = combined_df['label'].copy()
labels_df[labels_df != 'normal'] = 'attack'

# drop target features
combined_df.drop(['label'], axis=1, inplace=True)
combined_df.drop(['atakcat'], axis=1, inplace=True)
```

QLCFF: Quick Layered Correlation-based Feature Filter

> ***library requirements:***

* Dataframe with only numeric columns
* Numeric class labels in "array-like" with shape (n,1)
* Binary classification (not multiclass or multilabel)

In [9]:
```
# one-Hot encoding the remaining text features
categori = combined_df.select_dtypes(include=['object']).columns
category_cols = categori.tolist()

features_df = pandas.get_dummies(combined_df, columns=category_cols)
features_df.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 148517 entries, 0 to 22543
Columns: 121 entries, duration to flag_SH
dtypes: float64(15), int64(22), uint8(84)
memory usage: 55.0 MB
```

In [10]:
```
# numeric values for the target feature
from sklearn.preprocessing import LabelEncoder
ynum = LabelEncoder().fit_transform(labels_df)
```

---

import the local library

In [11]:
```
# add parent folder path where lib folder is
import sys
if ".." not in sys.path:import sys; sys.path.insert(0, '..')
```

In [12]:
```
## or ## from QLCFF import *
#
from QLCFF import mkbins
from QLCFF import filter_fcy, filter_fdr, filter_fcc
from QLCFF import get_filter, rpt_ycor, rpt_fcor

# mkbins: applies sklearn KBinsDiscretizer(strategy='uniform', encode='ordinal')
#       https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-di

# filter_fcy: naive filter, feature-to-label (f2y) correlations
#       filter all with low correlation to target
# filter_fdr: sklearn univariate chi-square test: FDR or FWE
#       https://scikit-learn.org/stable/modules/feature_selection.html#univariate-f
# filter_fcc: FCBF-style, filter on feature-to-feature (f2f) correlations
#       using Pearson correlation (PC) or symmetric uncertainty (SU)

# get_filter: returns list of features from f2y report
# rpt_ycor: print feature-to-label (f2y) correlations
# rpt_fcor: print feature-to-feature (f2f) correlations
```

---

mkbins(): FDR/FWE and SU requre binning

In [13]:
```
### def mkbins(indf, nb=20, detail=False):
# Requires: features dataframe
#    nb =  maximum number of bins, default 20
#    detail = print report, boolean, default False
# Returns: binned (discretized) dataframe

# applies sklearn KBinsDiscretizer(strategy='uniform', encode='ordinal')
# R doc for miMRMR asserts 10 bins is consistent with the literature
# 20 bins with uniform strategy means each bin is 5% of observed values
#   (ideally - it is really max_bins, uniform means all are the same size)

tmpdf = mkbins(features_df, detail=True)
# tmpdf.info(verbose=False)
```

```
Using only numeric datatypes [ All features are in ]
Unique value count: Original ::> Binned
  Same for all features except:
duration                     3424 ::> 18
src_bytes                    3601 ::> 8
dst_bytes                   10401 ::> 3
hot                            29 ::> 10
num_compromised                96 ::> 5
num_root                       91 ::> 5
num_file_creations             36 ::> 10
count                         512 ::> 20
srv_count                     512 ::> 20
serror_rate                    99 ::> 20
srv_serror_rate                94 ::> 20
rerror_rate                    98 ::> 20
srv_rerror_rate                95 ::> 20
same_srv_rate                 101 ::> 20
diff_srv_rate                 101 ::> 20
srv_diff_host_rate             87 ::> 20
dst_host_count                256 ::> 20
dst_host_srv_count            256 ::> 20
dst_host_same_srv_rate        101 ::> 20
dst_host_diff_srv_rate        101 ::> 20
dst_host_same_src_port_rate   101 ::> 20
dst_host_srv_diff_host_rate    75 ::> 20
dst_host_serror_rate          101 ::> 20
dst_host_srv_serror_rate      101 ::> 20
dst_host_rerror_rate          101 ::> 20
dst_host_srv_rerror_rate      101 ::> 20
```

In [14]:
```
# naive filter and univariate tests both filter features
#    on the basis that low correlation with the target labels
#    means low utility for distinguishing class membership
# chi_sq is a formal test for independence, fwe will select more to drop
#    than fdr, lower threshold will select more than higher
# naive filter will select all from either univariate test, and more
```

filter_fcy(): naive filter

In [15]:
```
### def filter_fcy(indf, ingt, minpc=0.1, minsu=0.01):
# naive filter: keep if f2y_pc >= minpc or f2y_su >= minsu
# Requires: features_df, numeric_labels
#    minpc = threshold (alpha) for pearson correlation
#    minsu = threshold (alpha) for symmetric uncertainty
# Returns: f2y report for features to drop

print('\nNaive filter (low correlation with target)')
nfdrop_yc, nfkeep_yc = filter_fcy(tmpdf, ynum)

if (len(nfdrop_yc) > 1):
    nfdrop = get_filter(nfdrop_yc)
    nfkeep = get_filter(nfkeep_yc)

    print('To Keep:',len(nfkeep),'features')
#    rpt_ycor(nfkeep_yc)
    print('To Drop:',len(nfdrop),'features')
#    rpt_ycor(nfdrop_yc)
else:
    print('No features were selected')
```

```
Naive filter (low correlation with target)
To Keep: 40 features
To Drop: 81 features
```

In [16]:
```
# apply the "keep" filter to the real dataset
# pandas has a lot of rules about returning a 'view' vs. a copy
#        so we force it to create a new dataframe
# python assigns by reference (namespaces) so keeping a reference
#        to the original is minimal overhead

# features_df_original = features_df
filtered_df = features_df[nfkeep].copy()
filtered_df.info(verbose=False)
# features_df = filtered_df
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 148517 entries, 0 to 22543
Columns: 40 entries, wrong_fragment to flag_SF
dtypes: float64(15), int64(6), uint8(19)
memory usage: 27.6 MB
```

filter_fdr(): univariate chi_sq layer

In [17]:
```
### def filter_fdr(dfin, gtin, t=0.01, usefdr=True):
# Requires: features_df, numeric_labels
#    t = threshold (alpha) for chi_sq test, sklearn default is 0.5
#    usefdr = test, boolean, fdr if True, else fwe
# Returns: f2y report for features to drop

# set test:
# tst = 'FDR'
# ufd = True
# = or =
tst = 'FWE'
ufd = False

print('\nUnivariate chi-sq',tst,'test')
uvdrop_yc = filter_fdr(tmpdf, ynum, usefdr=ufd)
```

```
if (len(uvdrop_yc) > 1):
    uvdrop = get_filter(uvdrop_yc)

    print('Progressive_filtering: Dropping',len(uvdrop),'features')
    tmpdf.drop(uvdrop, axis = 1, inplace = True)

    print('\nFiltered:',tst,'Layer')
    rpt_ycor(uvdrop_yc)
else:
    uvdrop = uvdrop_yc
    print('\n',tst,'Layer: No features were selected')
```

```
Univariate chi-sq FWE test
Progressive_filtering: Dropping 14 features

Filtered: FWE Layer
--Feature--              PCy          SUy          MIy
land                    -0.0088       0.0001          0.0
urgent                  -0.0039          0.0          0.0
num_shells               0.0003       0.0001          0.0
is_host_login           -0.0078       0.0001          0.0
is_guest_login          -0.0038          0.0          0.0
service_aol             -0.0038          0.0          0.0
service_harvest         -0.0038          0.0          0.0
service_http_2784       -0.0027          0.0          0.0
service_http_8001       -0.0038          0.0          0.0
service_other            -0.002          0.0          0.0
service_red_i            0.0071       0.0001          0.0
service_tftp_u            0.005       0.0001          0.0
service_tim_i           -0.0018          0.0          0.0
service_urh_i            0.0079       0.0001          0.0
```

In [18]:
```
# FCBF-Pearson and FCBF-SU Layers use the same code:  filter_fcc()
#            metric depends on the boolean argument:  usesu
# just use appropriate names for the return values
#      features to keep are called "predominant features" in the FCBF paper;
#      they act as proxies for the highly correlated features to drop
#      see Lei Yu & Huan Liu, Proc. 20th ICML 2003
```

filter_fcc(): FCBF-SU layer

In [19]:
```
### def filter_fcc(dfin, ingt, t=0.7, usesu=False):
# Requires: features_df, numeric_labels
#    t = threshold (alpha) for "high" f2f correlation
#        standard for detecting multicollinearity is 0.7
#    usesu = metric, boolean, su if True, else pearson
# Returns: f2y report for features to drop
#          f2y report for features to keep
#          f2f above threshold report

sudrop_yc, sukeep_yc, su_hicorr = filter_fcc(tmpdf, ynum, usesu=True)

if (len(sudrop_yc) > 1):
    sudrop = get_filter(sudrop_yc)

    print('Progressive_filtering: Dropping',len(sudrop),'features')
    tmpdf.drop(sudrop, axis = 1, inplace = True)

    print('\nKept: FCBF (SU) Layer')
    rpt_ycor(sukeep_yc)
    print('\nFiltered: FCBF (SU) Layer')
    rpt_ycor(sudrop_yc)
    print('\nHighly correlated features: FCBF (SU) Layer')
    rpt_fcor(su_hicorr)
else:
    sudrop = sudrop_yc
    print('\nFCBF (SU) Layer\nNo features were selected')
```

```
Calculating the SU correlation matrix takes some time ...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%  --Done
Progressive_filtering: Dropping 7 features

Kept: FCBF (SU) Layer
--Feature--                     PCy          SUy          MIy
flag_S0                        -0.5856        0.338       0.2117
dst_host_srv_rerror_rate       -0.2969       0.0861       0.0553
num_root                        0.0087       0.0005       0.0002

Filtered: FCBF (SU) Layer
--Feature--                     PCy          SUy          MIy
serror_rate                    -0.5874       0.3071       0.2222
dst_host_serror_rate           -0.5883        0.301       0.2287
dst_host_srv_serror_rate       -0.5939       0.3236       0.2233
srv_serror_rate                -0.5865       0.3128       0.2158
rerror_rate                    -0.3011       0.0848       0.0524
srv_rerror_rate                -0.2992       0.0828       0.0493
num_compromised                 0.0076       0.0004       0.0001

Highly correlated features: FCBF (SU) Layer
--Feature--                 PCf      SUf      MIf       --Feature--
num_root                     0.989   0.8657   0.0021   num_compromised
srv_serror_rate             0.9914   0.8361   0.6029   serror_rate
srv_rerror_rate             0.9861   0.8002   0.4161   rerror_rate
dst_host_serror_rate        0.9737   0.7388   0.5843   serror_rate
dst_host_serror_rate        0.9699   0.7042   0.5334   srv_serror_rate
dst_host_srv_serror_rate    0.9755   0.7449   0.5372   serror_rate
dst_host_srv_serror_rate    0.9815   0.7976   0.5485   srv_serror_rate
dst_host_srv_serror_rate    0.9823   0.7567   0.5732   dst_host_serror_rate
dst_host_srv_rerror_rate    0.9645    0.708   0.3855   srv_rerror_rate
flag_S0                     0.9727   0.7873   0.5176   serror_rate
flag_S0                     0.9768   0.8389   0.5235   srv_serror_rate
flag_S0                     0.9689   0.7477   0.5188   dst_host_serror_rate
flag_S0                     0.9751   0.8336   0.5203   dst_host_srv_serror_rate
```

filter_fcc(): FCBF-Pearson layer

In [20]:

```
### def filter_fcc(dfin, ingt, t=0.7, usesu=False):
# Requires: features_df, numeric_labels
#    t = threshold (alpha) for "high" f2f correlation
#        standard for detecting multicollinearity is 0.7
#    usesu = metric, boolean, su if True, else pearson
# Returns: f2y report for features to drop
#          f2y report for features to keep
#          f2f above threshold report

pcdrop_yc, pckeep_yc, pc_hicorr = filter_fcc(tmpdf, ynum)

if (len(pcdrop_yc) > 1):
    pcdrop = get_filter(pcdrop_yc)

    print('Progressive_filtering: Dropping',len(pcdrop),'features')
    tmpdf.drop(pcdrop, axis = 1, inplace = True)

    print('\nKept: Pearson Layer')
    rpt_ycor(pckeep_yc)
    print('\nFiltered: Pearson Layer')
    rpt_ycor(pcdrop_yc)
    print('\nHighly correlated features: Pearson Layer')
    rpt_fcor(pc_hicorr)
else:
    pcdrop = pcdrop_yc
    print('\nPearson Layer\nNo features were selected')
```

```
Progressive_filtering: Dropping 10 features

Kept: Pearson Layer
--Feature--                 PCy          SUy          MIy
flag_SF                      0.7277       0.4423       0.3015
dst_host_rerror_rate        -0.3004       0.0694       0.0524
service_domain_u             0.2573       0.0971       0.0455
logged_in                    0.6641       0.3653       0.2496
protocol_type_icmp          -0.1911       0.0428       0.0198

Filtered: Pearson Layer
--Feature--                      PCy          SUy          MIy
dst_host_srv_count                0.6928       0.2119        0.287
dst_host_same_srv_rate            0.6689       0.2092       0.2658
flag_S0                          -0.5856        0.338       0.2117
same_srv_rate                     0.7091       0.3049       0.3056
flag_REJ                         -0.2275        0.054       0.0275
dst_host_srv_rerror_rate         -0.2969       0.0861       0.0553
protocol_type_tcp                 -0.057       0.0028       0.0016
protocol_type_udp                 0.2114       0.0454        0.024
service_http                      0.5676        0.278       0.1839
service_eco_i                    -0.1511       0.0307       0.0128

Highly correlated features: Pearson Layer
--Feature--                 PCf      SUf      MIf       --Feature--
dst_host_srv_count           0.7091   0.2761   0.4595   same_srv_rate
dst_host_same_srv_rate       0.7923   0.3265   0.5159   same_srv_rate
dst_host_same_srv_rate       0.8977   0.5813   1.1233   dst_host_srv_count
dst_host_srv_rerror_rate     0.9195   0.5592   0.3939   dst_host_rerror_rate
protocol_type_udp           -0.7791   0.5898   0.2471   protocol_type_tcp
service_domain_u              0.73    0.5391   0.1643   protocol_type_udp
service_eco_i                0.7093   0.5299   0.1003   protocol_type_icmp
service_http                 0.7084   0.4177   0.2724   logged_in
service_http                 0.7196   0.2467   0.3264   dst_host_srv_count
flag_REJ                     0.8086   0.4046   0.2318   dst_host_rerror_rate
flag_REJ                      0.836    0.529   0.2433   dst_host_srv_rerror_rate
flag_S0                     -0.7211   0.3102   0.2905   same_srv_rate
flag_SF                      0.8159   0.3879   0.3846   same_srv_rate
flag_SF                      0.7319   0.2684   0.3381   dst_host_same_srv_rate
flag_SF                     -0.7108   0.4865   0.2995   flag_S0
```

In [21]:
```
## QLCFF: Quick Layered Correlation-based Feature Filter
## full layered drop filter for real DF
QLCFFilter = []
QLCFFilter.extend(x for x in uvdrop if x not in QLCFFilter)
QLCFFilter.extend(x for x in pcdrop if x not in QLCFFilter)
QLCFFilter.extend(x for x in sudrop if x not in QLCFFilter)
```

In [22]:
```
# apply the "drop" filter to the real dataset

# features_df_original = features_df
filtered_df = features_df.drop(QLCFFilter, axis = 1)
filtered_df.info(verbose=False)
# features_df = filtered_df
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 148517 entries, 0 to 22543
Columns: 90 entries, duration to flag_SH
dtypes: float64(6), int64(15), uint8(69)
memory usage: 34.7 MB
```

In [ ]: