MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles

This is the code for the paper entitled "**MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles**" accepted in IEEE Internet of Things Journal.
Authors: Li Yang (liyanghart@gmail.com), Abdallah Moubayed, and Abdallah Shami
Organization: The Optimized Computing and Communications (OC2) Lab, ECE Department, Western University

If you find this repository useful in your research, please cite:
L. Yang, A. Moubayed, and A. Shami, "MTH-IDS: A Multi-Tiered Hybrid Intrusion Detection System for Internet of Vehicles," IEEE Internet of Things Journal, vol. 9, no. 1, pp. 616-632, Jan.1, 2022.

Import libraries

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,precision_recall
_fscore_support
from sklearn.metrics import f1_score,roc_auc_score
from sklearn.ensemble import
RandomForestClassifier,ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
from xgboost import plot_importance
```

Read the sampled CICIDS2017 dataset

The CICIDS2017 dataset is publicly available at: https://www.unb.ca/cic/datasets/ids-2017.html
Due to the large size of this dataset, the sampled subsets of CICIDS2017 is used. The subsets are in the "data" folder.
If you want to use this code on other datasets (e.g., CAN-intrusion dataset), just change the dataset name and follow the same steps. The models in this code are generic models that can be used in any intrusion detection/network traffic datasets.

```
#Read dataset
df = pd.read_csv('./data/CICIDS2017.csv')
# The results in this code is based on the original CICIDS2017
dataset. Please go to cell [21] if you work on the sampled dataset.

df
```

```
         Flow Duration  Total Fwd Packets  Total Backward Packets  \
0                    3                  2                       0
1                  109                  1                       1
2                   52                  1                       1
3                   34                  1                       1
4                    3                  2                       0
...                ...                ...                     ...
2830738          32215                  4                       2
2830739            324                  2                       2
2830740             82                  2                       1
2830741        1048635                  6                       2
2830742          94939                  4                       2

         Total Length of Fwd Packets  Total Length of Bwd Packets  \
0                                 12                            0
1                                  6                            6
2                                  6                            6
3                                  6                            6
4                                 12                            0
...                              ...                          ...
2830738                          112                          152
2830739                           84                          362
2830740                           31                            6
2830741                          192                          256
2830742                          188                          226

         Fwd Packet Length Max  Fwd Packet Length Min  Fwd Packet
Length Mean  \
0                            6                      6
6.0
1                            6                      6
6.0
2                            6                      6
6.0
3                            6                      6
6.0
4                            6                      6
6.0
...                        ...                    ...
...
2830738                     28                     28
28.0
2830739                     42                     42
```

```
42.0
2830740                       31                         0
15.5
2830741                       32                        32
32.0
2830742                       47                        47
47.0

         Fwd Packet Length Std  Bwd Packet Length Max  ...  \
0                      0.00000                      0  ...
1                      0.00000                      6  ...
2                      0.00000                      6  ...
3                      0.00000                      6  ...
4                      0.00000                      0  ...
...                        ...                    ...  ...
2830738                0.00000                     76  ...
2830739                0.00000                    181  ...
2830740               21.92031                      6  ...
2830741                0.00000                    128  ...
2830742                0.00000                    113  ...

         min_seg_size_forward  Active Mean  Active Std  Active Max  \
0                          20          0.0         0.0           0
1                          20          0.0         0.0           0
2                          20          0.0         0.0           0
3                          20          0.0         0.0           0
4                          20          0.0         0.0           0
...                       ...          ...         ...         ...
2830738                    20          0.0         0.0           0
2830739                    20          0.0         0.0           0
2830740                    32          0.0         0.0           0
2830741                    20          0.0         0.0           0
2830742                    20          0.0         0.0           0

         Active Min  Idle Mean  Idle Std  Idle Max  Idle Min   Label
0                 0        0.0       0.0         0         0  BENIGN
1                 0        0.0       0.0         0         0  BENIGN
2                 0        0.0       0.0         0         0  BENIGN
3                 0        0.0       0.0         0         0  BENIGN
4                 0        0.0       0.0         0         0  BENIGN
...             ...        ...       ...       ...       ...     ...
2830738           0        0.0       0.0         0         0  BENIGN
2830739           0        0.0       0.0         0         0  BENIGN
2830740           0        0.0       0.0         0         0  BENIGN
2830741           0        0.0       0.0         0         0  BENIGN
2830742           0        0.0       0.0         0         0  BENIGN

[2830743 rows x 78 columns]
```

```
df.Label.value_counts()
```

```
BENIGN          2273097
DoS              380699
PortScan         158930
BruteForce        13835
WebAttack          2180
Bot                1966
Infiltration         36
Name: Label, dtype: int64
```

Preprocessing (normalization and padding values)

```
# Z-score normalization
features = df.dtypes[df.dtypes != 'object'].index
df[features] = df[features].apply(
    lambda x: (x - x.mean()) / (x.std()))
# Fill empty values by 0
df = df.fillna(0)
```

```
C:\Users\41364\AppData\Roaming\Python\Python35\site-packages\pandas\
compat\_optional.py:106: UserWarning: Pandas requires version '2.6.2'
or newer of 'numexpr' (version '2.6.1' currently installed).
  warnings.warn(msg, UserWarning)
```

Data sampling

Due to the space limit of GitHub files and the large size of network traffic data, we sample a small-sized subset for model learning using **k-means cluster sampling**

```
labelencoder = LabelEncoder()
df.iloc[:, -1] = labelencoder.fit_transform(df.iloc[:, -1])

df.Label.value_counts()
```

```
0    2273097
3     380699
5     158930
2      13835
6       2180
1       1966
4         36
Name: Label, dtype: int64
```

```
# retain the minority class instances and sample the majority class
instances
df_minor = df[(df['Label']==6)|(df['Label']==1)|(df['Label']==4)]
df_major = df.drop(df_minor.index)

X = df_major.drop(['Label'],axis=1)
y = df_major.iloc[:, -1].values.reshape(-1,1)
y=np.ravel(y)
```

```python
# use k-means to cluster the data samples and select a proportion of
data from each cluster
from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(n_clusters=1000, random_state=0).fit(X)

klabel=kmeans.labels_
df_major['klabel']=klabel

df_major['klabel'].value_counts()
```

```
318    22146
2      20340
258    20225
308    18461
432    18154
       ...
366       70
92        21
596       14
756       10
295        3
Name: klabel, Length: 997, dtype: int64
```

```python
cols = list(df_major)
cols.insert(78, cols.pop(cols.index('Label')))
df_major = df_major.loc[:, cols]

df_major
```

```
         Flow Duration  Total Fwd Packets  Total Backward Packets  \
0            -0.439347          -0.009819               -0.010421
1            -0.439344          -0.011153               -0.009418
2            -0.439345          -0.011153               -0.009418
3            -0.439346          -0.011153               -0.009418
4            -0.439347          -0.009819               -0.010421
...                ...                ...                     ...
2830738      -0.438390          -0.007151               -0.008416
2830739      -0.439337          -0.009819               -0.008416
2830740      -0.439344          -0.009819               -0.009418
2830741      -0.408187          -0.004484               -0.008416
2830742      -0.436526          -0.007151               -0.008416

         Total Length of Fwd Packets  Total Length of Bwd Packets  \
0                          -0.053765                    -0.007142
1                          -0.054365                    -0.007139
2                          -0.054365                    -0.007139
3                          -0.054365                    -0.007139
4                          -0.053765                    -0.007142
...                              ...                          ...
2830738                    -0.043758                    -0.007075
2830739                    -0.046560                    -0.006982
```

```
2830740                       -0.051863                       -0.007139
2830741                       -0.035753                       -0.007029
2830742                       -0.036153                       -0.007042

         Fwd Packet Length Max  Fwd Packet Length Min  Fwd Packet
Length Mean  \
0                    -0.281099              -0.210703                -
0.280518
1                    -0.281099              -0.210703                -
0.280518
2                    -0.281099              -0.210703                -
0.280518
3                    -0.281099              -0.210703                -
0.280518
4                    -0.281099              -0.210703                -
0.280518
...                        ...                    ...
...
2830738              -0.250424               0.153902                -
0.162296
2830739              -0.230903               0.385923                -
0.087065
2830740              -0.246240              -0.310140                -
0.229468
2830741              -0.244846               0.220194                -
0.140802
2830742              -0.223931               0.468788                -
0.060196

         Fwd Packet Length Std  Bwd Packet Length Max  ...  Active
Mean  \
0                    -0.245069              -0.447423  ...    -
0.125734
1                    -0.245069              -0.444340  ...    -
0.125734
2                    -0.245069              -0.444340  ...    -
0.125734
3                    -0.245069              -0.444340  ...    -
0.125734
4                    -0.245069              -0.447423  ...    -
0.125734
...                        ...                    ...  ...          ..
.
2830738              -0.245069              -0.408376  ...    -
0.125734
2830739              -0.245069              -0.354429  ...    -
0.125734
2830740              -0.167112              -0.444340  ...    -
0.125734
```

```
2830741                   -0.245069                 -0.381659  ...    -
0.125734
2830742                   -0.245069                 -0.389366  ...    -
0.125734

         Active Std  Active Max  Active Min  Idle Mean  Idle Std  Idle
Max  \
0          -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
1          -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
2          -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
3          -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
4          -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
...              ...         ...         ...        ...       ...
...
2830738    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
2830739    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
2830740    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
2830741    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
2830742    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868

         Idle Min  klabel  Label
0       -0.338993     391      0
1       -0.338993     498      0
2       -0.338993     499      0
3       -0.338993     787      0
4       -0.338993     391      0
...           ...     ...    ...
2830738 -0.338993     813      0
2830739 -0.338993     916      0
2830740 -0.338993     267      0
2830741 -0.338993     634      0
2830742 -0.338993     978      0

[2826561 rows x 79 columns]

def typicalSampling(group):
    name = group.name
    frac = 0.008
    return group.sample(frac=frac)
```

```
result = df_major.groupby(
    'klabel', group_keys=False
).apply(typicalSampling)
```

```
result['Label'].value_counts()
```

```
0    18185
3     3029
5     1280
2      118
Name: Label, dtype: int64
```

```
result
```

```
         Flow Duration  Total Fwd Packets  Total Backward Packets  \
6980         -0.437857          -0.011153               -0.009418
1506627      -0.438252          -0.011153               -0.009418
1377524      -0.438860          -0.011153               -0.009418
2056871      -0.435684          -0.011153               -0.009418
2005567      -0.437738          -0.011153               -0.009418
...                ...                ...                     ...
1031173      -0.438439          -0.011153               -0.009418
1608048      -0.438422          -0.009819               -0.008416
817023       -0.437935          -0.011153               -0.009418
559006       -0.437946          -0.011153               -0.009418
985052       -0.437554          -0.011153               -0.009418

         Total Length of Fwd Packets  Total Length of Bwd Packets  \
6980                       -0.054965                    -0.007142
1506627                    -0.054965                    -0.007142
1377524                    -0.054965                    -0.007142
2056871                    -0.054965                    -0.007142
2005567                    -0.054965                    -0.007142
...                              ...                          ...
1031173                    -0.050963                    -0.007110
1608048                    -0.047160                    -0.007083
817023                     -0.050863                    -0.007108
559006                     -0.050763                    -0.007111
985052                     -0.050963                    -0.007110

         Fwd Packet Length Max  Fwd Packet Length Min  Fwd Packet
Length Mean  \
6980                 -0.289465              -0.310140             -
0.312760
1506627              -0.289465              -0.310140             -
0.312760
1377524              -0.289465              -0.310140             -
0.312760
2056871              -0.289465              -0.310140             -
0.312760
```

```
2005567                   -0.289465                 -0.310140                -
0.312760
...                             ...                       ...
...
1031173                   -0.233691                  0.352777                -
0.097812
1608048                   -0.235086                  0.336204                -
0.103186
817023                    -0.232297                  0.369350                -
0.092438
559006                    -0.230903                  0.385923                -
0.087065
985052                    -0.233691                  0.352777                -
0.097812

         Fwd Packet Length Std  Bwd Packet Length Max  ...  Active
Mean  \
6980                      -0.245069                 -0.447423  ...    -
0.125734
1506627                   -0.245069                 -0.447423  ...    -
0.125734
1377524                   -0.245069                 -0.447423  ...    -
0.125734
2056871                   -0.245069                 -0.447423  ...    -
0.125734
2005567                   -0.245069                 -0.447423  ...    -
0.125734
...                             ...                       ...  ...          ..
.
1031173                   -0.245069                 -0.410431  ...    -
0.125734
1608048                   -0.245069                 -0.413000  ...    -
0.125734
817023                    -0.245069                 -0.408376  ...    -
0.125734
559006                    -0.245069                 -0.411459  ...    -
0.125734
985052                    -0.245069                 -0.410431  ...    -
0.125734

         Active Std  Active Max  Active Min  Idle Mean  Idle Std  Idle
Max  \
6980      -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
1506627   -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
1377524   -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
2056871   -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
```

```
0.356868
2005567   -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
...              ...         ...         ...        ...        ...
...
1031173   -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
1608048   -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
817023    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
559006    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868
985052    -0.104565   -0.149326   -0.101016  -0.351926  -0.10946 -
0.356868

         Idle Min  klabel  Label
6980    -0.338993       0      0
1506627 -0.338993       0      0
1377524 -0.338993       0      0
2056871 -0.338993       0      0
2005567 -0.338993       0      0
...           ...     ...    ...
1031173 -0.338993     999      0
1608048 -0.338993     999      0
817023  -0.338993     999      0
559006  -0.338993     999      0
985052  -0.338993     999      0

[22612 rows x 79 columns]

result = result.drop(['klabel'],axis=1)
result = result.append(df_minor)

result.to_csv('./data/CICIDS2017_sample_km.csv',index=0)
```

split train set and test set

```
# Read the sampled dataset
df=pd.read_csv('./data/CICIDS2017_sample_km.csv')

X = df.drop(['Label'],axis=1).values
y = df.iloc[:, -1].values.reshape(-1,1)
y=np.ravel(y)

X_train, X_test, y_train, y_test = train_test_split(X,y, train_size =
0.8, test_size = 0.2, random_state = 0,stratify = y)
```

Feature engineering

Feature selection by information gain

```
from sklearn.feature_selection import mutual_info_classif
importances = mutual_info_classif(X_train, y_train)

# calculate the sum of importance scores
f_list = sorted(zip(map(lambda x: round(x, 4), importances),
features), reverse=True)
Sum = 0
fs = []
for i in range(0, len(f_list)):
    Sum = Sum + f_list[i][0]
    fs.append(f_list[i][1])

# select the important features from top to bottom until the
accumulated importance reaches 90%
f_list2 = sorted(zip(map(lambda x: round(x, 4), importances/Sum),
features), reverse=True)
Sum2 = 0
fs = []
for i in range(0, len(f_list2)):
    Sum2 = Sum2 + f_list2[i][0]
    fs.append(f_list2[i][1])
    if Sum2>=0.9:
        break

X_fs = df[fs].values

X_fs.shape

(26794, 44)
```

Feature selection by Fast Correlation Based Filter (FCBF)

The module is imported from the GitHub repo: https://github.com/SantiagoEG/FCBF_module

```
from FCBF_module import FCBF, FCBFK, FCBFiP, get_i
fcbf = FCBFK(k = 20)
#fcbf.fit(X_fs, y)

X_fss = fcbf.fit_transform(X_fs,y)

X_fss.shape

(26794, 20)
```

Re-split train & test sets after feature selection

```
X_train, X_test, y_train, y_test = train_test_split(X_fss,y,
train_size = 0.8, test_size = 0.2, random_state = 0,stratify = y)

X_train.shape

(21435, 20)

pd.Series(y_train).value_counts()

0    14548
3     2423
6     1744
1     1573
5     1024
2       94
4       29
dtype: int64
```

SMOTE to solve class-imbalance

```
from imblearn.over_sampling import SMOTE
smote=SMOTE(n_jobs=-1,sampling_strategy={2:1000,4:1000})

C:\Users\41364\AppData\Roaming\Python\Python35\site-packages\sklearn\
externals\six.py:31: DeprecationWarning: The module is deprecated in
version 0.21 and will be removed in version 0.23 since we've dropped
support for Python 2.7. Please rely on the official version of six
(https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)

X_train, y_train = smote.fit_resample(X_train, y_train)

pd.Series(y_train).value_counts()

0    14548
3     2423
6     1744
1     1573
5     1024
4     1000
2     1000
dtype: int64
```

Machine learning model training

Training four base learners: decision tree, random forest, extra trees, XGBoost

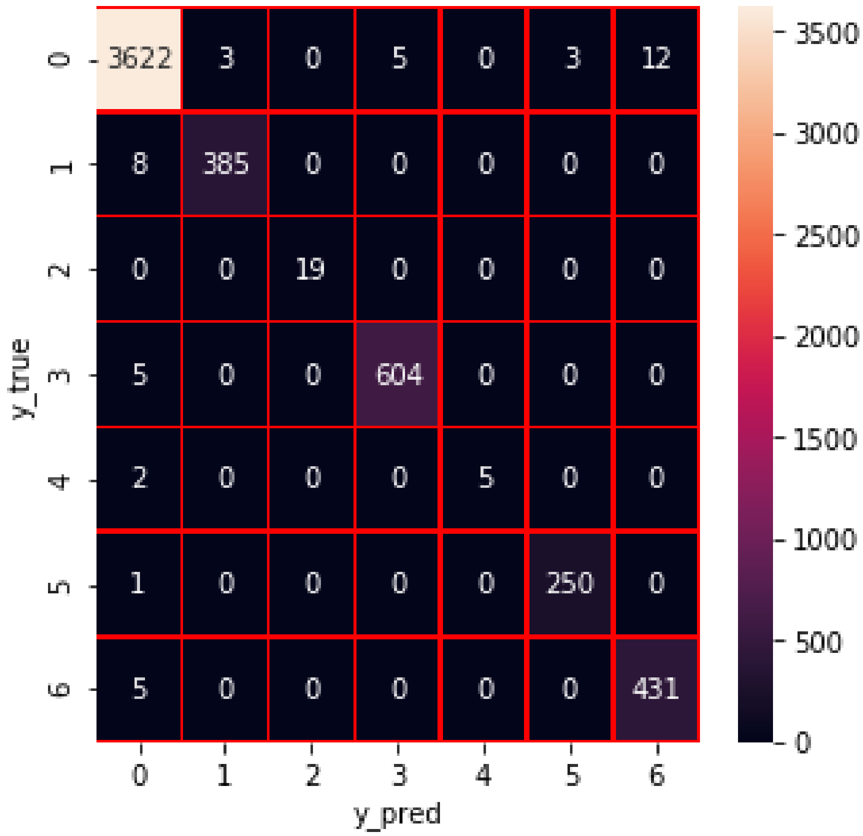
Apply XGBoost

```
xg = xgb.XGBClassifier(n_estimators = 10)
xg.fit(X_train,y_train)
xg_score=xg.score(X_test,y_test)
y_predict=xg.predict(X_test)
y_true=y_test
print('Accuracy of XGBoost: '+ str(xg_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of XGBoost: '+(str(precision)))
print('Recall of XGBoost: '+(str(recall)))
print('F1-score of XGBoost: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of XGBoost: 0.9917910447761195
Precision of XGBoost: 0.991821481887011
Recall of XGBoost: 0.9917910447761195
F1-score of XGBoost: 0.9917663641435618
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3645
           1       0.99      0.98      0.99       393
           2       1.00      1.00      1.00        19
           3       0.99      0.99      0.99       609
           4       1.00      0.71      0.83         7
           5       0.99      1.00      0.99       251
           6       0.97      0.99      0.98       436

    accuracy                           0.99      5360
   macro avg       0.99      0.95      0.97      5360
weighted avg       0.99      0.99      0.99      5360
```



Hyperparameter optimization (HPO) of XGBoost using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

Based on the GitHub repo for HPO: https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

```
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from sklearn.model_selection import cross_val_score, StratifiedKFold
def objective(params):
    params = {
        'n_estimators': int(params['n_estimators']),
        'max_depth': int(params['max_depth']),
        'learning_rate':  abs(float(params['learning_rate'])),

    }
    clf = xgb.XGBClassifier( **params)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    score = accuracy_score(y_test, y_pred)

    return {'loss':-score, 'status': STATUS_OK }

space = {
    'n_estimators': hp.quniform('n_estimators', 10, 100, 5),
```

```
    'max_depth': hp.quniform('max_depth', 4, 100, 1),
    'learning_rate': hp.normal('learning_rate', 0.01, 0.9),
}

best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=20)
print("XGBoost: Hyperopt estimated optimum {}".format(best))
```

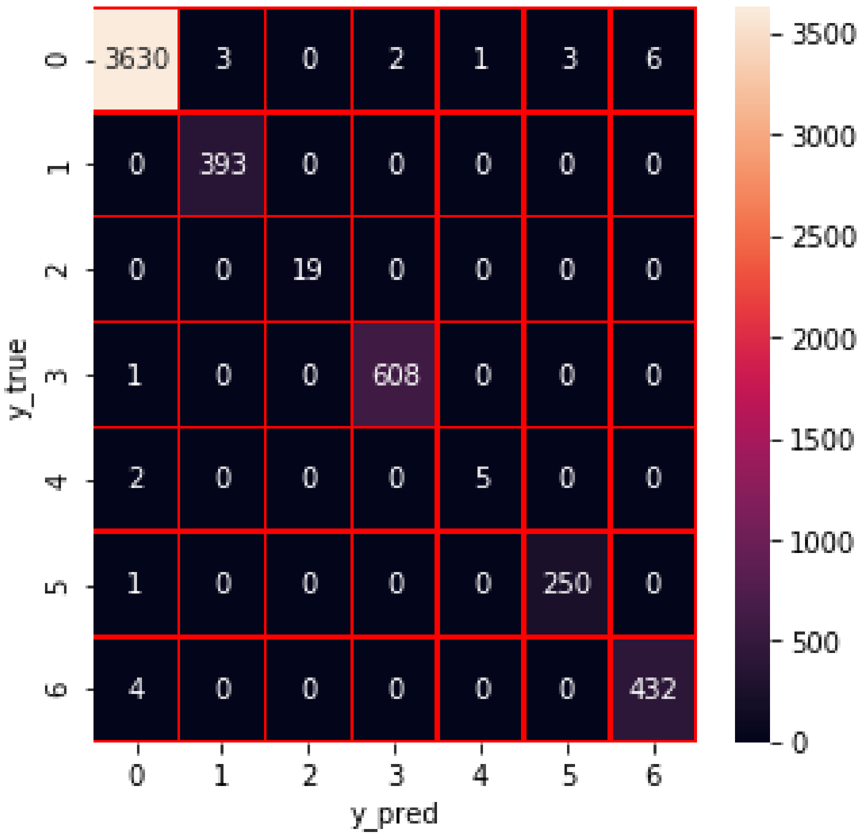
```
100%|██████████████████████████████████████████████| 20/20
[00:30<00:00,  1.53s/trial, best loss: -0.9957089552238806]
XGBoost: Hyperopt estimated optimum {'learning_rate':
0.7340229699980686, 'n_estimators': 70.0, 'max_depth': 14.0}
```

```
xg = xgb.XGBClassifier(learning_rate= 0.7340229699980686, n_estimators
= 70, max_depth = 14)
xg.fit(X_train,y_train)
xg_score=xg.score(X_test,y_test)
y_predict=xg.predict(X_test)
y_true=y_test
print('Accuracy of XGBoost: '+ str(xg_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of XGBoost: '+(str(precision)))
print('Recall of XGBoost: '+(str(recall)))
print('F1-score of XGBoost: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of XGBoost: 0.9957089552238806
Precision of XGBoost: 0.9956893766598436
Recall of XGBoost: 0.9957089552238806
F1-score of XGBoost: 0.9956902750637269
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3645
           1       0.99      1.00      1.00       393
           2       1.00      1.00      1.00        19
           3       1.00      1.00      1.00       609
           4       0.83      0.71      0.77         7
           5       0.99      1.00      0.99       251
           6       0.99      0.99      0.99       436
```

```
    accuracy                           1.00      5360
   macro avg       0.97      0.96      0.96      5360
weighted avg       1.00      1.00      1.00      5360
```




```
xg_train=xg.predict(X_train)
xg_test=xg.predict(X_test)
```

Apply RF

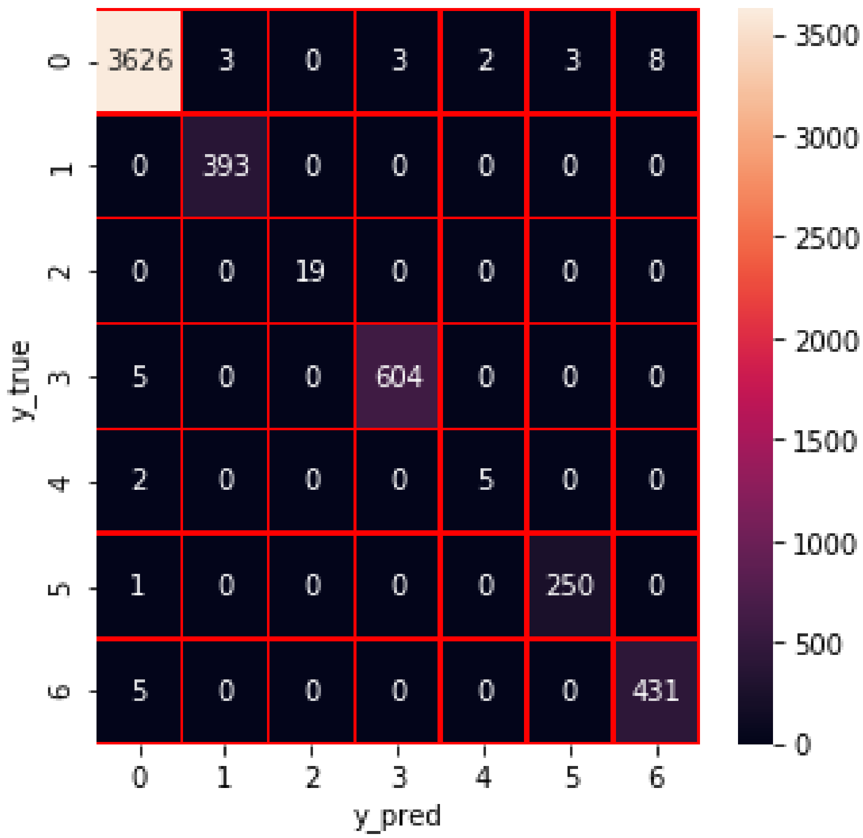
```
rf = RandomForestClassifier(random_state = 0)
rf.fit(X_train,y_train)
rf_score=rf.score(X_test,y_test)
y_predict=rf.predict(X_test)
y_true=y_test
print('Accuracy of RF: '+ str(rf_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of RF: '+(str(precision)))
print('Recall of RF: '+(str(recall)))
print('F1-score of RF: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
```

```
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of RF: 0.9940298507462687
Precision of RF: 0.9940428718755328
Recall of RF: 0.9940298507462687
F1-score of RF: 0.9940328670009781
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      3645
           1       0.99      1.00      1.00       393
           2       1.00      1.00      1.00        19
           3       1.00      0.99      0.99       609
           4       0.71      0.71      0.71         7
           5       0.99      1.00      0.99       251
           6       0.98      0.99      0.99       436

    accuracy                           0.99      5360
   macro avg       0.95      0.96      0.95      5360
weighted avg       0.99      0.99      0.99      5360
```



Hyperparameter optimization (HPO) of random forest using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

Based on the GitHub repo for HPO: https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

```
# Hyperparameter optimization of random forest
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from sklearn.model_selection import cross_val_score, StratifiedKFold
# Define the objective function
def objective(params):
    params = {
        'n_estimators': int(params['n_estimators']),
        'max_depth': int(params['max_depth']),
        'max_features': int(params['max_features']),
        "min_samples_split":int(params['min_samples_split']),
        "min_samples_leaf":int(params['min_samples_leaf']),
        "criterion":str(params['criterion'])
    }
    clf = RandomForestClassifier( **params)
    clf.fit(X_train,y_train)
    score=clf.score(X_test,y_test)
```

```
    return {'loss':-score, 'status': STATUS_OK }
# Define the hyperparameter configuration space
space = {
    'n_estimators': hp.quniform('n_estimators', 10, 200, 1),
    'max_depth': hp.quniform('max_depth', 5, 50, 1),
    "max_features":hp.quniform('max_features', 1, 20, 1),
    "min_samples_split":hp.quniform('min_samples_split',2,11,1),
    "min_samples_leaf":hp.quniform('min_samples_leaf',1,11,1),
    "criterion":hp.choice('criterion',['gini','entropy'])
}

best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=20)
print("Random Forest: Hyperopt estimated optimum {}".format(best))
```

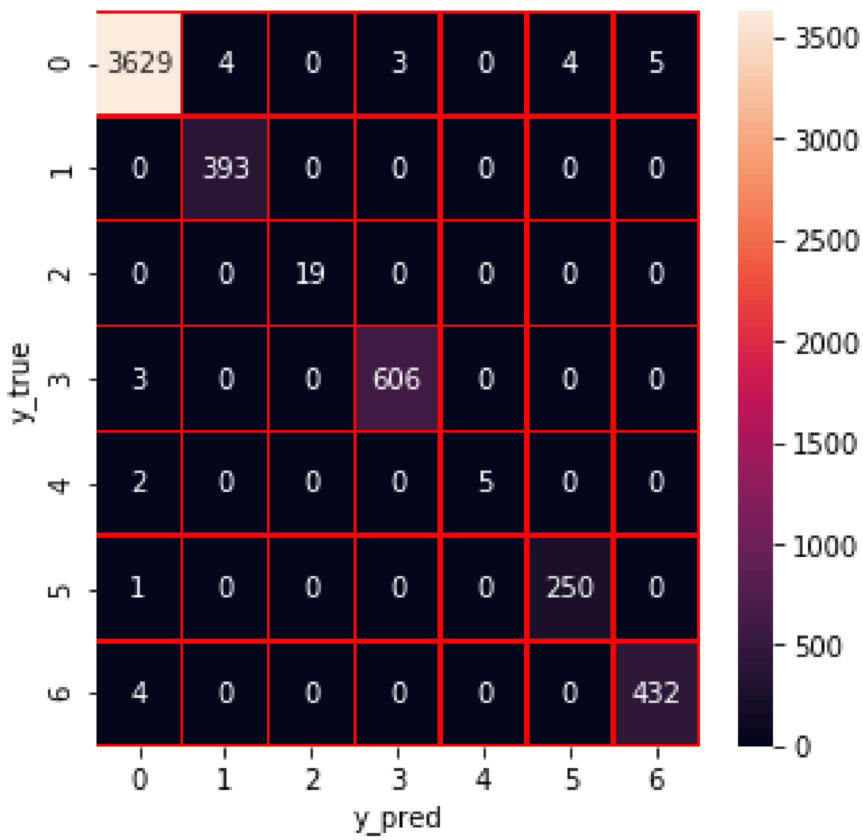
```
100%|██████████████████████████████████████████████| 20/20
[01:34<00:00,  4.72s/trial, best loss: -0.9955223880597015]
Random Forest: Hyperopt estimated optimum {'n_estimators': 71.0,
'min_samples_leaf': 1.0, 'max_depth': 46.0, 'min_samples_split': 9.0,
'max_features': 20.0, 'criterion': 1}
```

```
rf_hpo = RandomForestClassifier(n_estimators = 71, min_samples_leaf =
1, max_depth = 46, min_samples_split = 9, max_features = 20, criterion
= 'entropy')
rf_hpo.fit(X_train,y_train)
rf_score=rf_hpo.score(X_test,y_test)
y_predict=rf_hpo.predict(X_test)
y_true=y_test
print('Accuracy of RF: '+ str(rf_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of RF: '+(str(precision)))
print('Recall of RF: '+(str(recall)))
print('F1-score of RF: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of RF: 0.9951492537313433
Precision of RF: 0.9951646455154706
Recall of RF: 0.9951492537313433
F1-score of RF: 0.9951217831414103
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00      3645
           1       0.99      1.00      0.99       393
           2       1.00      1.00      1.00        19
           3       1.00      1.00      1.00       609
           4       1.00      0.71      0.83         7
           5       0.98      1.00      0.99       251
           6       0.99      0.99      0.99       436

    accuracy                           1.00      5360
   macro avg       0.99      0.96      0.97      5360
weighted avg       1.00      1.00      1.00      5360
```




```
rf_train=rf_hpo.predict(X_train)
rf_test=rf_hpo.predict(X_test)
```

Apply DT

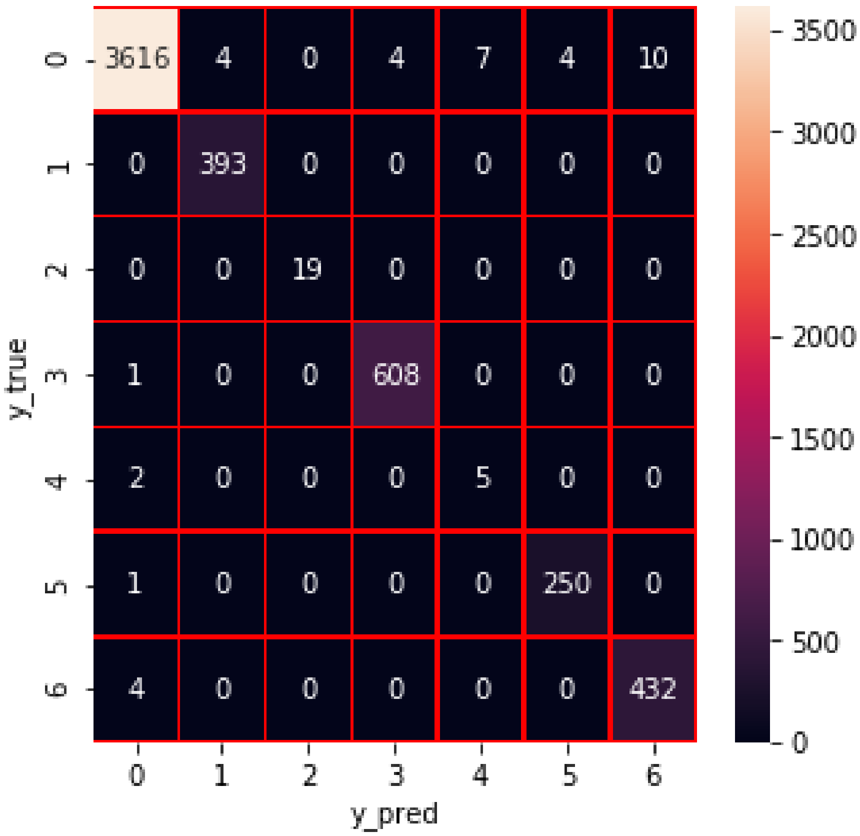
```
dt = DecisionTreeClassifier(random_state = 0)
dt.fit(X_train,y_train)
dt_score=dt.score(X_test,y_test)
y_predict=dt.predict(X_test)
```

```python
y_true=y_test
print('Accuracy of DT: '+ str(dt_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true, y_predict, average='weighted')
print('Precision of DT: '+(str(precision)))
print('Recall of DT: '+(str(recall)))
print('F1-score of DT: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of DT: 0.9930970149253732
Precision of DT: 0.9936778376607716
Recall of DT: 0.9930970149253732
F1-score of DT: 0.9933227091323931
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      3645
           1       0.99      1.00      0.99       393
           2       1.00      1.00      1.00        19
           3       0.99      1.00      1.00       609
           4       0.42      0.71      0.53         7
           5       0.98      1.00      0.99       251
           6       0.98      0.99      0.98       436

    accuracy                           0.99      5360
   macro avg       0.91      0.96      0.93      5360
weighted avg       0.99      0.99      0.99      5360
```



Hyperparameter optimization (HPO) of decision tree using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

Based on the GitHub repo for HPO: https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

```
# Hyperparameter optimization of decision tree
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from sklearn.model_selection import cross_val_score, StratifiedKFold
# Define the objective function
def objective(params):
    params = {
        'max_depth': int(params['max_depth']),
        'max_features': int(params['max_features']),
        "min_samples_split":int(params['min_samples_split']),
        "min_samples_leaf":int(params['min_samples_leaf']),
        "criterion":str(params['criterion'])
    }
    clf = DecisionTreeClassifier( **params)
    clf.fit(X_train,y_train)
    score=clf.score(X_test,y_test)

    return {'loss':-score, 'status': STATUS_OK }
```

```python
# Define the hyperparameter configuration space
space = {
    'max_depth': hp.quniform('max_depth', 5, 50, 1),
    "max_features":hp.quniform('max_features', 1, 20, 1),
    "min_samples_split":hp.quniform('min_samples_split',2,11,1),
    "min_samples_leaf":hp.quniform('min_samples_leaf',1,11,1),
    "criterion":hp.choice('criterion',['gini','entropy'])
}

best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=50)
print("Decision tree: Hyperopt estimated optimum {}".format(best))
```

```
100%|██████████████████████████████████████████████| 50/50
[00:04<00:00, 11.13trial/s, best loss: -0.9936567164179104]
Decision tree: Hyperopt estimated optimum {'min_samples_leaf': 2.0,
'max_depth': 47.0, 'min_samples_split': 3.0, 'max_features': 19.0,
'criterion': 0}
```

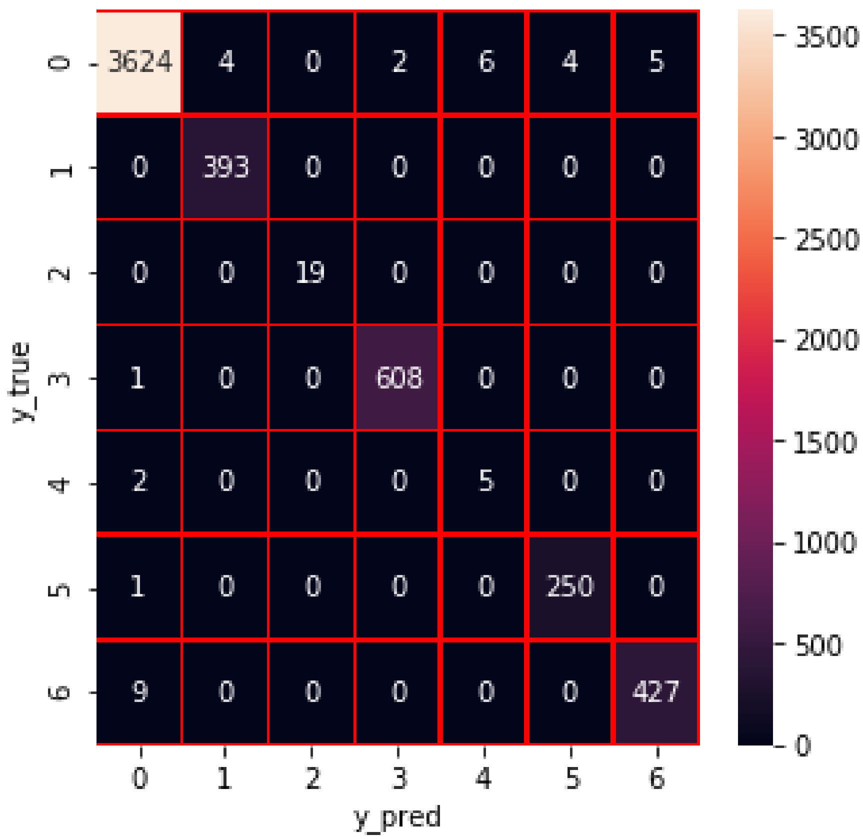
```python
dt_hpo = DecisionTreeClassifier(min_samples_leaf = 2, max_depth = 47,
min_samples_split = 3, max_features = 19, criterion = 'gini')
dt_hpo.fit(X_train,y_train)
dt_score=dt_hpo.score(X_test,y_test)
y_predict=dt_hpo.predict(X_test)
y_true=y_test
print('Accuracy of DT: '+ str(dt_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of DT: '+(str(precision)))
print('Recall of DT: '+(str(recall)))
print('F1-score of DT: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of DT: 0.9936567164179104
Precision of DT: 0.9940667447648622
Recall of DT: 0.9936567164179104
F1-score of DT: 0.9938179408993949
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      3645
           1       0.99      1.00      0.99       393
```

```
           2       1.00      1.00      1.00        19
           3       1.00      1.00      1.00       609
           4       0.45      0.71      0.56         7
           5       0.98      1.00      0.99       251
           6       0.99      0.98      0.98       436

    accuracy                           0.99      5360
   macro avg       0.92      0.95      0.93      5360
weighted avg       0.99      0.99      0.99      5360
```




```
dt_train=dt_hpo.predict(X_train)
dt_test=dt_hpo.predict(X_test)
```

Apply ET

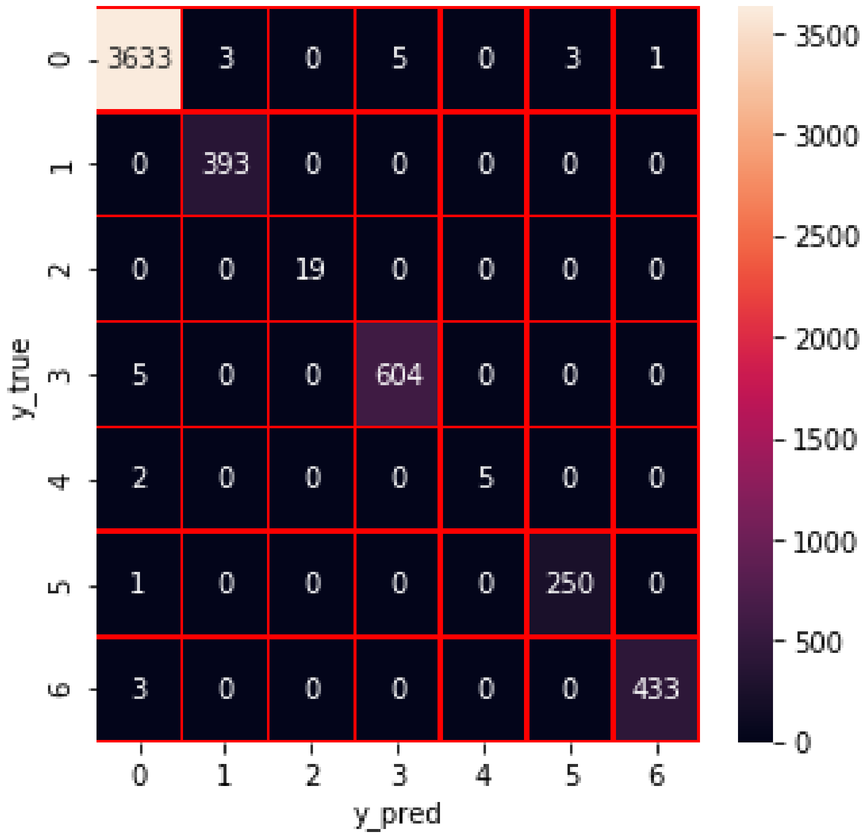
```
et = ExtraTreesClassifier(random_state = 0)
et.fit(X_train,y_train)
et_score=et.score(X_test,y_test)
y_predict=et.predict(X_test)
y_true=y_test
print('Accuracy of ET: '+ str(et_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
```

```
y_predict, average='weighted')
print('Precision of ET: '+(str(precision)))
print('Recall of ET: '+(str(recall)))
print('F1-score of ET: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of ET: 0.9957089552238806
Precision of ET: 0.9957161969649261
Recall of ET: 0.9957089552238806
F1-score of ET: 0.9956792533287913
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3645
           1       0.99      1.00      1.00       393
           2       1.00      1.00      1.00        19
           3       0.99      0.99      0.99       609
           4       1.00      0.71      0.83         7
           5       0.99      1.00      0.99       251
           6       1.00      0.99      1.00       436

    accuracy                           1.00      5360
   macro avg       1.00      0.96      0.97      5360
weighted avg       1.00      1.00      1.00      5360
```



Hyperparameter optimization (HPO) of extra trees using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

Based on the GitHub repo for HPO: https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

```
# Hyperparameter optimization of extra trees
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from sklearn.model_selection import cross_val_score, StratifiedKFold
# Define the objective function
def objective(params):
    params = {
        'n_estimators': int(params['n_estimators']),
        'max_depth': int(params['max_depth']),
        'max_features': int(params['max_features']),
        "min_samples_split":int(params['min_samples_split']),
        "min_samples_leaf":int(params['min_samples_leaf']),
        "criterion":str(params['criterion'])
    }
    clf = ExtraTreesClassifier( **params)
    clf.fit(X_train,y_train)
    score=clf.score(X_test,y_test)
```

```
    return {'loss':-score, 'status': STATUS_OK }
# Define the hyperparameter configuration space
space = {
    'n_estimators': hp.quniform('n_estimators', 10, 200, 1),
    'max_depth': hp.quniform('max_depth', 5, 50, 1),
    "max_features":hp.quniform('max_features', 1, 20, 1),
    "min_samples_split":hp.quniform('min_samples_split',2,11,1),
    "min_samples_leaf":hp.quniform('min_samples_leaf',1,11,1),
    "criterion":hp.choice('criterion',['gini','entropy'])
}

best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=20)
print("Random Forest: Hyperopt estimated optimum {}".format(best))
```

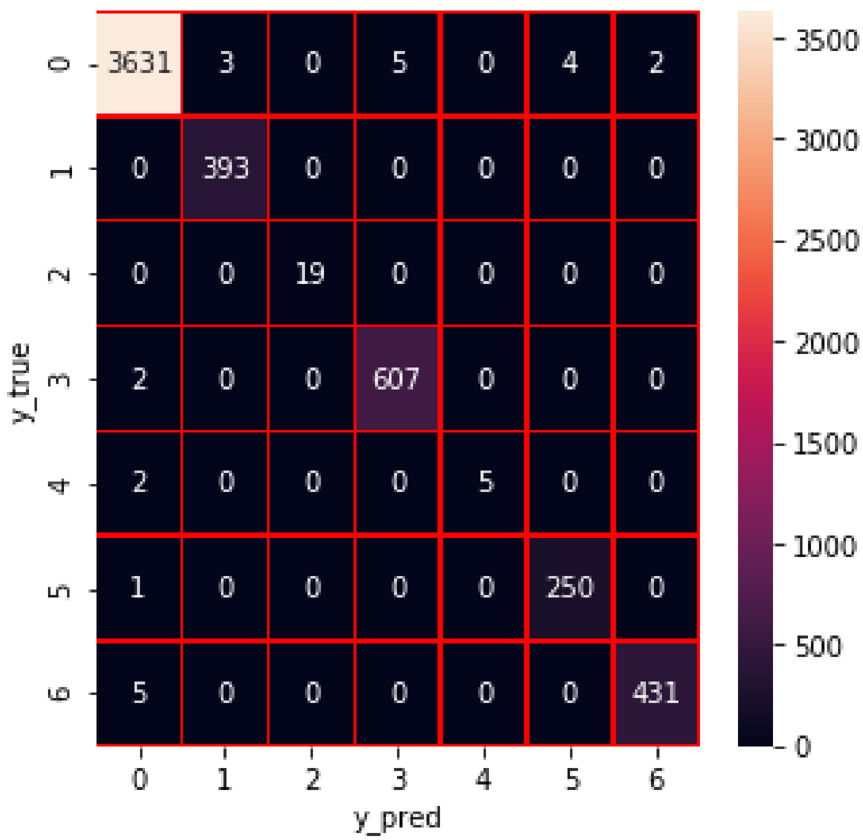
```
100%|████████████████████████████████████████████████| 20/20
[00:25<00:00,  1.28s/trial, best loss: -0.9955223880597015]
Random Forest: Hyperopt estimated optimum {'n_estimators': 53.0,
'min_samples_leaf': 1.0, 'max_depth': 31.0, 'min_samples_split': 5.0,
'max_features': 20.0, 'criterion': 1}
```

```
et_hpo = ExtraTreesClassifier(n_estimators = 53, min_samples_leaf = 1,
max_depth = 31, min_samples_split = 5, max_features = 20, criterion =
'entropy')
et_hpo.fit(X_train,y_train)
et_score=et_hpo.score(X_test,y_test)
y_predict=et_hpo.predict(X_test)
y_true=y_test
print('Accuracy of ET: '+ str(et_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of ET: '+(str(precision)))
print('Recall of ET: '+(str(recall)))
print('F1-score of ET: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of ET: 0.9955223880597015
Precision of ET: 0.9955353802920419
Recall of ET: 0.9955223880597015
F1-score of ET: 0.9954932494250629
              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00      3645
              1       0.99      1.00      1.00       393
              2       1.00      1.00      1.00        19
              3       0.99      1.00      0.99       609
              4       1.00      0.71      0.83         7
              5       0.98      1.00      0.99       251
              6       1.00      0.99      0.99       436

       accuracy                           1.00      5360
      macro avg       0.99      0.96      0.97      5360
   weighted avg       1.00      1.00      1.00      5360
```




```
et_train=et_hpo.predict(X_train)
et_test=et_hpo.predict(X_test)
```

Apply Stacking

The ensemble model that combines the four ML models (DT, RF, ET, XGBoost)

```
base_predictions_train = pd.DataFrame( {
    'DecisionTree': dt_train.ravel(),
        'RandomForest': rf_train.ravel(),
     'ExtraTrees': et_train.ravel(),
     'XgBoost': xg_train.ravel(),
    })
base_predictions_train.head(5)
```

	DecisionTree	ExtraTrees	RandomForest	XgBoost
0	0	0	0	0
1	0	0	0	0
2	1	1	1	1
3	0	0	0	0
4	3	3	3	3

```
dt_train=dt_train.reshape(-1, 1)
et_train=et_train.reshape(-1, 1)
rf_train=rf_train.reshape(-1, 1)
xg_train=xg_train.reshape(-1, 1)
dt_test=dt_test.reshape(-1, 1)
et_test=et_test.reshape(-1, 1)
rf_test=rf_test.reshape(-1, 1)
xg_test=xg_test.reshape(-1, 1)
```

```
dt_train.shape
```

```
(23334, 1)
```

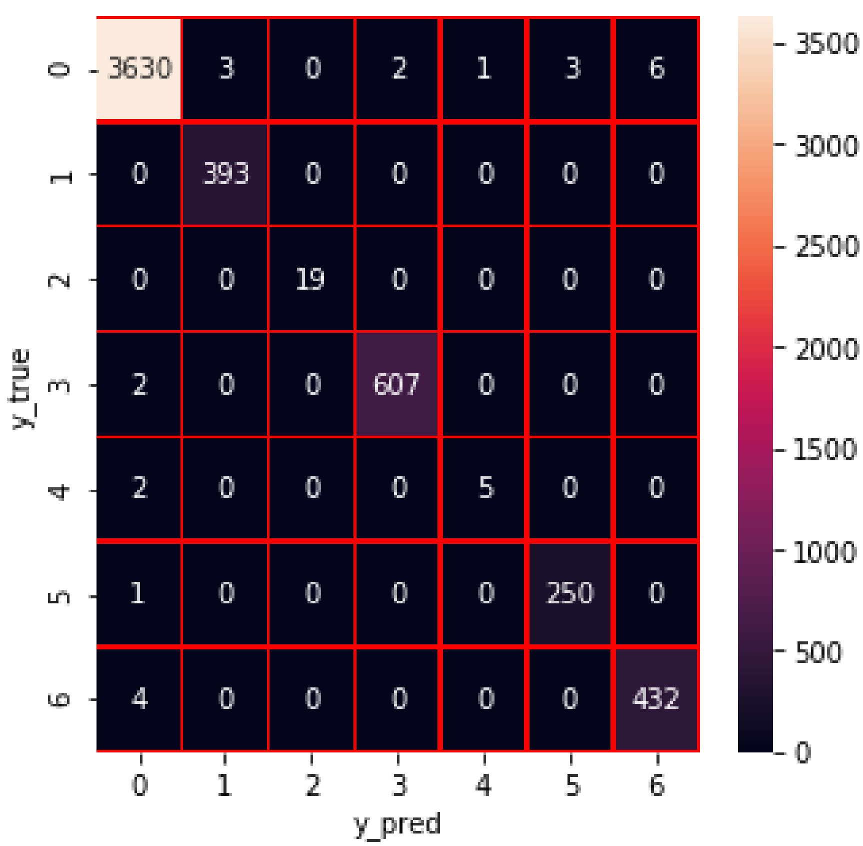
```
x_train = np.concatenate(( dt_train, et_train, rf_train, xg_train),
axis=1)
x_test = np.concatenate(( dt_test, et_test, rf_test, xg_test), axis=1)
```

```
stk = xgb.XGBClassifier().fit(x_train, y_train)
y_predict=stk.predict(x_test)
y_true=y_test
stk_score=accuracy_score(y_true,y_predict)
print('Accuracy of Stacking: '+ str(stk_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of Stacking: '+(str(precision)))
print('Recall of Stacking: '+(str(recall)))
print('F1-score of Stacking: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of Stacking: 0.9955223880597015
Precision of Stacking: 0.9955023011268291
Recall of Stacking: 0.9955223880597015
F1-score of Stacking: 0.99550369632154
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3645
           1       0.99      1.00      1.00       393
           2       1.00      1.00      1.00        19
           3       1.00      1.00      1.00       609
           4       0.83      0.71      0.77         7
           5       0.99      1.00      0.99       251
           6       0.99      0.99      0.99       436

    accuracy                           1.00      5360
   macro avg       0.97      0.96      0.96      5360
weighted avg       1.00      1.00      1.00      5360
```



Hyperparameter optimization (HPO) of the stacking ensemble model (XGBoost) using Bayesian optimization with tree-based Parzen estimator (BO-TPE)

Based on the GitHub repo for HPO: https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

```
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from sklearn.model_selection import cross_val_score, StratifiedKFold
def objective(params):
    params = {
        'n_estimators': int(params['n_estimators']),
        'max_depth': int(params['max_depth']),
        'learning_rate':  abs(float(params['learning_rate'])),

    }
    clf = xgb.XGBClassifier( **params)
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    score = accuracy_score(y_test, y_pred)

    return {'loss':-score, 'status': STATUS_OK }

space = {
    'n_estimators': hp.quniform('n_estimators', 10, 100, 5),
    'max_depth': hp.quniform('max_depth', 4, 100, 1),
    'learning_rate': hp.normal('learning_rate', 0.01, 0.9),
}

best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=20)
print("XGBoost: Hyperopt estimated optimum {}".format(best))
```

```
100%|██████████████████████████████████████████████| 20/20
[00:08<00:00,  2.34trial/s, best loss: -0.9957089552238806]
XGBoost: Hyperopt estimated optimum {'learning_rate': -
0.19229249758051492, 'n_estimators': 30.0, 'max_depth': 36.0}
```

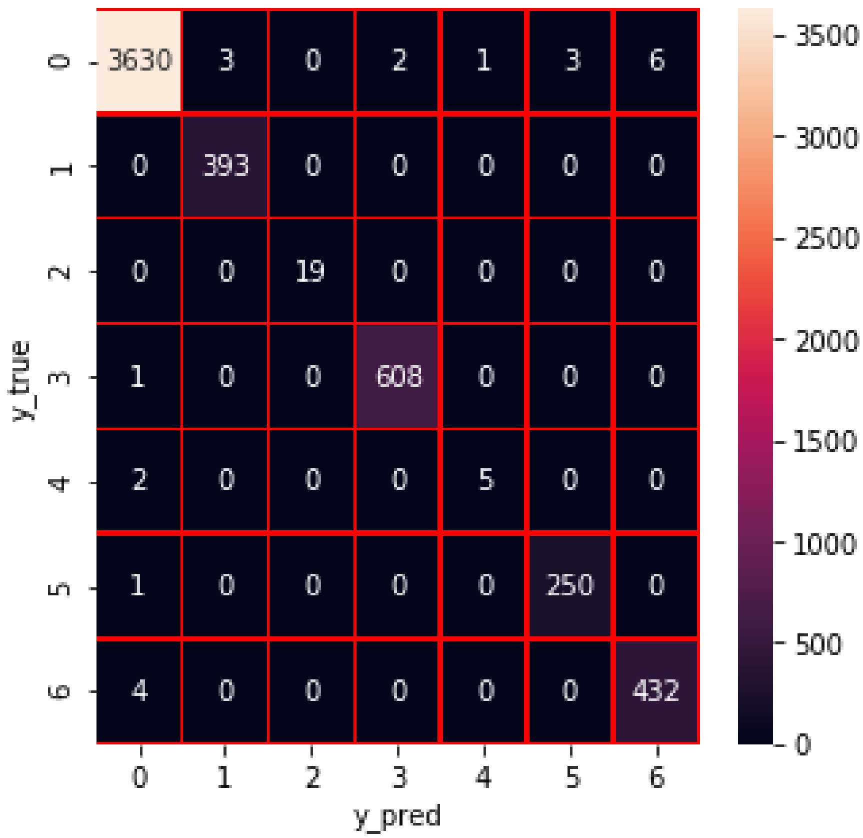
```
xg = xgb.XGBClassifier(learning_rate= 0.19229249758051492,
n_estimators = 30, max_depth = 36)
xg.fit(x_train,y_train)
xg_score=xg.score(x_test,y_test)
y_predict=xg.predict(x_test)
y_true=y_test
print('Accuracy of XGBoost: '+ str(xg_score))
precision,recall,fscore,none= precision_recall_fscore_support(y_true,
y_predict, average='weighted')
print('Precision of XGBoost: '+(str(precision)))
print('Recall of XGBoost: '+(str(recall)))
```

```python
print('F1-score of XGBoost: '+(str(fscore)))
print(classification_report(y_true,y_predict))
cm=confusion_matrix(y_true,y_predict)
f,ax=plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot=True,linewidth=0.5,linecolor="red",fmt=".0f",ax=a
x)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

```
Accuracy of XGBoost: 0.9957089552238806
Precision of XGBoost: 0.9956893766598436
Recall of XGBoost: 0.9957089552238806
F1-score of XGBoost: 0.9956902750637269
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3645
           1       0.99      1.00      1.00       393
           2       1.00      1.00      1.00        19
           3       1.00      1.00      1.00       609
           4       0.83      0.71      0.77         7
           5       0.99      1.00      0.99       251
           6       0.99      0.99      0.99       436

    accuracy                           1.00      5360
   macro avg       0.97      0.96      0.96      5360
weighted avg       1.00      1.00      1.00      5360
```



Anomaly-based IDS

Generate the port-scan datasets for unknown attack detection

```
df=pd.read_csv('./data/CICIDS2017_sample_km.csv')

df.Label.value_counts()

0    18225
3     3042
6     2180
1     1966
5     1255
2       96
4       36
Name: Label, dtype: int64

df1 = df[df['Label'] != 5]
df1['Label'][df1['Label'] > 0] = 1
df1.to_csv('./data/CICIDS2017_sample_km_without_portscan.csv',index=0)

df2 = df[df['Label'] == 5]
df2['Label'][df2['Label'] == 5] = 1
df2.to_csv('./data/CICIDS2017_sample_km_portscan.csv',index=0)
```

Read the generated datasets for unknown attack detection

```
df1 = pd.read_csv('./data/CICIDS2017_sample_km_without_portscan.csv')
df2 = pd.read_csv('./data/CICIDS2017_sample_km_portscan.csv')

features = df1.drop(['Label'],axis=1).dtypes[df1.dtypes !=
'object'].index
df1[features] = df1[features].apply(
    lambda x: (x - x.mean()) / (x.std()))
df2[features] = df2[features].apply(
    lambda x: (x - x.mean()) / (x.std()))
df1 = df1.fillna(0)
df2 = df2.fillna(0)

df1.Label.value_counts()

0    18225
1     7320
Name: Label, dtype: int64

df2.Label.value_counts()

1    1255
Name: Label, dtype: int64

df2p=df1[df1['Label']==0]
df2pp=df2p.sample(n=None, frac=1255/18225, replace=False,
weights=None, random_state=None, axis=0)
df2=pd.concat([df2, df2pp])

df2.Label.value_counts()

1    1255
0    1255
Name: Label, dtype: int64

df = df1.append(df2)

X = df.drop(['Label'],axis=1) .values
y = df.iloc[:, -1].values.reshape(-1,1)
y=np.ravel(y)
pd.Series(y).value_counts()

0    19480
1     8575
dtype: int64
```

Feature engineering (IG, FCBF, and KPCA)

Feature selection by information gain (IG)

```
from sklearn.feature_selection import mutual_info_classif
importances = mutual_info_classif(X, y)

# calculate the sum of importance scores
f_list = sorted(zip(map(lambda x: round(x, 4), importances),
features), reverse=True)
Sum = 0
fs = []
for i in range(0, len(f_list)):
    Sum = Sum + f_list[i][0]
    fs.append(f_list[i][1])

# select the important features from top to bottom until the
accumulated importance reaches 90%
f_list2 = sorted(zip(map(lambda x: round(x, 4), importances/Sum),
features), reverse=True)
Sum2 = 0
fs = []
for i in range(0, len(f_list2)):
    Sum2 = Sum2 + f_list2[i][0]
    fs.append(f_list2[i][1])
    if Sum2>=0.9:
        break

X_fs = df[fs].values

X_fs.shape
```

```
(28055, 50)
```

```
X_fs
```

```
array([[-0.34612159, -0.51326791, -0.44364535, ..., -0.11333586,
        -0.13353417, -0.05349902],
       [-0.3443274 , -0.51326791, -0.44364535, ..., -0.11333586,
        -0.13353417, -0.05349902],
       [-0.3443274 , -0.51326791, -0.44364535, ..., -0.11333586,
        -0.13353417, -0.05349902],
       ...,
       [-0.36859622, -0.20454057, -0.32295149, ..., -0.11333586,
        -0.13353417, -0.05349902],
       [-0.3561313 ,  0.63721854,  0.36583358, ..., -0.11333586,
        -0.13353417,  0.00459227],
       [ 2.7318634 , -0.53347551, -0.44364535, ..., -0.11333586,
        -0.13353417, -0.05349902]])
```

Feature selection by Fast Correlation Based Filter (FCBF)

The module is imported from the GitHub repo: https://github.com/SantiagoEG/FCBF_module

```
from FCBF_module import FCBF, FCBFK, FCBFiP, get_i
fcbf = FCBFK(k = 20)
#fcbf.fit(X_fs, y)

X_fss = fcbf.fit_transform(X_fs,y)

X_fss.shape

(28055, 20)

X_fss

array([[-0.34612159, -0.53319222, -0.34935843, ..., -0.42229765,
        -0.2803002 , -0.41947688],
       [-0.3443274 , -0.54906516, -0.34935843, ..., -0.42229765,
        -0.2803002 , -0.41947688],
       [-0.3443274 , -0.55544206, -0.34935843, ..., -0.42229765,
        -0.2803002 , -0.41947688],
       ...,
       [-0.36859622, -0.56375976, -0.34935843, ..., -0.42229765,
        -0.2803002 , -0.32403604],
       [-0.3561313 ,  0.00413109, -0.33807808, ..., -0.41021078,
        -0.27174505,  0.36453998],
       [ 2.7318634 , -0.53929186, -0.34935843, ..., -0.42229765,
        -0.2803002 , -0.42271216]])
```

kernel principal component analysis (KPCA)

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 10, kernel = 'rbf')
kpca.fit(X_fss, y)
X_kpca = kpca.transform(X_fss)

# from sklearn.decomposition import PCA
# kpca = PCA(n_components = 10)
# kpca.fit(X_fss, y)
# X_kpca = kpca.transform(X_fss)
```

Train-test split after feature selection

```
X_train = X_kpca[:len(df1)]
y_train = y[:len(df1)]
X_test = X_kpca[len(df1):]
y_test = y[len(df1):]
```

Solve class-imbalance by SMOTE

```
pd.Series(y_train).value_counts()
```

```
0    18225
1     7320
dtype: int64
```

```
from imblearn.over_sampling import SMOTE
smote=SMOTE(n_jobs=-1,sampling_strategy={1:18225})
X_train, y_train = smote.fit_resample(X_train, y_train)
```

```
pd.Series(y_train).value_counts()
```

```
1    18225
0    18225
dtype: int64
```

```
pd.Series(y_test).value_counts()
```

```
1    1255
0    1255
dtype: int64
```

Apply the cluster labeling (CL) k-means method

```
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN,MeanShift
from sklearn.cluster import
SpectralClustering,AgglomerativeClustering,AffinityPropagation,Birch,M
iniBatchKMeans,MeanShift
from sklearn.mixture import GaussianMixture, BayesianGaussianMixture
from sklearn.metrics import classification_report
from sklearn import metrics

def CL_kmeans(X_train, X_test, y_train, y_test,n,b=100):
    km_cluster = MiniBatchKMeans(n_clusters=n,batch_size=b)
    result = km_cluster.fit_predict(X_train)
    result2 = km_cluster.predict(X_test)

    count=0
    a=np.zeros(n)
    b=np.zeros(n)
    for v in range(0,n):
        for i in range(0,len(y_train)):
            if result[i]==v:
                if y_train[i]==1:
                    a[v]=a[v]+1
                else:
                    b[v]=b[v]+1
    list1=[]
    list2=[]
```

```
    for v in range(0,n):
        if a[v]<=b[v]:
            list1.append(v)
        else:
            list2.append(v)
    for v in range(0,len(y_test)):
        if result2[v] in list1:
            result2[v]=0
        elif result2[v] in list2:
            result2[v]=1
        else:
            print("-1")
    print(classification_report(y_test, result2))
    cm=confusion_matrix(y_test,result2)
    acc=metrics.accuracy_score(y_test,result2)
    print(str(acc))
    print(cm)

CL_kmeans(X_train, X_test, y_train, y_test, 8)
```

```
              precision    recall  f1-score   support

           0       0.58      0.69      0.63      1255
           1       0.62      0.51      0.56      1255

    accuracy                           0.60      2510
   macro avg       0.60      0.60      0.60      2510
weighted avg       0.60      0.60      0.60      2510

0.5984063745019921
[[864 391]
 [617 638]]
```

Hyperparameter optimization of CL-k-means

Tune "k"

```
#Hyperparameter optimization by BO-GP
from skopt.space import Real, Integer
from skopt.utils import use_named_args
from sklearn import metrics

space  = [Integer(2, 50, name='n_clusters')]
@use_named_args(space)
def objective(**params):
    km_cluster = MiniBatchKMeans(batch_size=100, **params)
    n=params['n_clusters']

    result = km_cluster.fit_predict(X_train)
    result2 = km_cluster.predict(X_test)
```

```python
    count=0
    a=np.zeros(n)
    b=np.zeros(n)
    for v in range(0,n):
        for i in range(0,len(y_train)):
            if result[i]==v:
                if y_train[i]==1:
                    a[v]=a[v]+1
                else:
                    b[v]=b[v]+1
    list1=[]
    list2=[]
    for v in range(0,n):
        if a[v]<=b[v]:
            list1.append(v)
        else:
            list2.append(v)
    for v in range(0,len(y_test)):
        if result2[v] in list1:
            result2[v]=0
        elif result2[v] in list2:
            result2[v]=1
        else:
            print("-1")
    cm=metrics.accuracy_score(y_test,result2)
    print(str(n)+" "+str(cm))
    return (1-cm)
from skopt import gp_minimize
import time
t1=time.time()
res_gp = gp_minimize(objective, space, n_calls=20, random_state=0)
t2=time.time()
print(t2-t1)
print("Best score=%.4f" % (1-res_gp.fun))
print("""Best parameters: n_clusters=%d""" % (res_gp.x[0]))
```

```
30 0.6972111553784861
43 0.7127490039840637
43 0.399203187250996
43 0.47051792828685257
32 0.653784860557769
20 0.34860557768924305
16 0.9195219123505977
5 0.4370517928286853
15 0.6729083665338645
25 0.7063745019920319
2 0.47808764940239046
50 0.4199203187250996
```

```
C:\Program Files\Anaconda3\lib\site-packages\skopt\optimizer\
optimizer.py:409: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "

2 0.47768924302788845

C:\Program Files\Anaconda3\lib\site-packages\skopt\optimizer\
optimizer.py:409: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "

50 0.39282868525896414
17 0.42828685258964144

C:\Program Files\Anaconda3\lib\site-packages\skopt\optimizer\
optimizer.py:409: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "

2 0.47768924302788845

C:\Program Files\Anaconda3\lib\site-packages\skopt\optimizer\
optimizer.py:409: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "

2 0.47768924302788845

C:\Program Files\Anaconda3\lib\site-packages\skopt\optimizer\
optimizer.py:409: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "

16 0.6992031872509961

C:\Program Files\Anaconda3\lib\site-packages\skopt\optimizer\
optimizer.py:409: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "

16 0.3737051792828685

C:\Program Files\Anaconda3\lib\site-packages\skopt\optimizer\
optimizer.py:409: UserWarning: The objective has been evaluated at
this point before.
  warnings.warn("The objective has been evaluated "

50 0.6250996015936255
9.127083539962769
Best score=0.9195
Best parameters: n_clusters=16
```

```python
#Hyperparameter optimization by BO-TPE
from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.cluster import MiniBatchKMeans
from sklearn import metrics

def objective(params):
    params = {
        'n_clusters': int(params['n_clusters']),
    }
    km_cluster = MiniBatchKMeans(batch_size=100, **params)
    n=params['n_clusters']

    result = km_cluster.fit_predict(X_train)
    result2 = km_cluster.predict(X_test)

    count=0
    a=np.zeros(n)
    b=np.zeros(n)
    for v in range(0,n):
        for i in range(0,len(y_train)):
            if result[i]==v:
                if y_train[i]==1:
                    a[v]=a[v]+1
                else:
                    b[v]=b[v]+1
    list1=[]
    list2=[]
    for v in range(0,n):
        if a[v]<=b[v]:
            list1.append(v)
        else:
            list2.append(v)
    for v in range(0,len(y_test)):
        if result2[v] in list1:
            result2[v]=0
        elif result2[v] in list2:
            result2[v]=1
        else:
            print("-1")
    score=metrics.accuracy_score(y_test,result2)
    print(str(params['n_clusters'])+" "+str(score))
    return {'loss':1-score, 'status': STATUS_OK }
space = {
    'n_clusters': hp.quniform('n_clusters', 2, 50, 1),
}

best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
```

```
             max_evals=20)
print("Random Forest: Hyperopt estimated optimum {}".format(best))
```

```
23 0.34422310756972113
15 0.6685258964143427
46 0.450199203187251
15 0.4896414342629482
29 0.6824701195219124
36 0.3888446215139442
22 0.35776892430278884
25 0.34860557768924305
42 0.41832669322709165
27 0.47051792828685257
26 0.39402390438247015
25 0.6824701195219124
33 0.3848605577689243
19 0.7191235059760956
6 0.5824701195219123
21 0.6697211155378486
24 0.451394422310757
37 0.4681274900398406
14 0.47250996015936253
21 0.8434262948207172
100%|██████████████████████████████████████████████████| 20/20
[00:06<00:00,  2.87trial/s, best loss: 0.15657370517928282]
Random Forest: Hyperopt estimated optimum {'n_clusters': 21.0}
```

```
CL_kmeans(X_train, X_test, y_train, y_test, 16)
```

```
              precision    recall  f1-score   support

           0       0.99      0.90      0.94      1255
           1       0.91      0.99      0.95      1255

    accuracy                           0.95      2510
   macro avg       0.95      0.95      0.94      2510
weighted avg       0.95      0.95      0.94      2510

0.9450199203187251
[[1127  128]
 [  10 1245]]
```

Apply the CL-k-means model with biased classifiers

```
# Only a sample code to show the logic. It needs to work on the entire
dataset to generate sufficient training samples for biased classifiers
def Anomaly_IDS(X_train, X_test, y_train, y_test,n,b=100):
    # CL-kmeans
    km_cluster = MiniBatchKMeans(n_clusters=n,batch_size=b)
    result = km_cluster.fit_predict(X_train)
```

```
result2 = km_cluster.predict(X_test)

count=0
a=np.zeros(n)
b=np.zeros(n)
for v in range(0,n):
    for i in range(0,len(y_train)):
        if result[i]==v:
            if y_train[i]==1:
                a[v]=a[v]+1
            else:
                b[v]=b[v]+1
list1=[]
list2=[]
for v in range(0,n):
    if a[v]<=b[v]:
        list1.append(v)
    else:
        list2.append(v)
for v in range(0,len(y_test)):
    if result2[v] in list1:
        result2[v]=0
    elif result2[v] in list2:
        result2[v]=1
    else:
        print("-1")
print(classification_report(y_test, result2))
cm=confusion_matrix(y_test,result2)
acc=metrics.accuracy_score(y2,result2)
print(str(acc))
print(cm)

#Biased classifier construction
count=0
print(len(y))
a=np.zeros(n)
b=np.zeros(n)
FNL=[]
FPL=[]
for v in range(0,n):
    al=[]
    bl=[]
    for i in range(0,len(y)):
        if result[i]==v:
            if y[i]==1:        #label 1
                a[v]=a[v]+1
                al.append(i)
            else:             #label 0
                b[v]=b[v]+1
```

```
                    bl.append(i)
        if a[v]<=b[v]:
            FNL.extend(al)
        else:
            FPL.extend(bl)
        #print(str(v)+"="+str(a[v]/(a[v]+b[v])))

    dffp=df.iloc[FPL, :]
    dffn=df.iloc[FNL, :]
    dfva0=df[df['Label']==0]
    dfva1=df[df['Label']==1]

    dffpp=dfva1.sample(n=None, frac=len(FPL)/dfva1.shape[0],
replace=False, weights=None, random_state=None, axis=0)
    dffnp=dfva0.sample(n=None, frac=len(FNL)/dfva0.shape[0],
replace=False, weights=None, random_state=None, axis=0)

    dffp_f=pd.concat([dffp, dffpp])
    dffn_f=pd.concat([dffn, dffnp])

    Xp = dffp_f.drop(['Label'],axis=1)
    yp = dffp_f.iloc[:, -1].values.reshape(-1,1)
    yp=np.ravel(yp)

    Xn = dffn_f.drop(['Label'],axis=1)
    yn = dffn_f.iloc[:, -1].values.reshape(-1,1)
    yn=np.ravel(yn)

    rfp = RandomForestClassifier(random_state = 0)
    rfp.fit(Xp,yp)
    rfn = RandomForestClassifier(random_state = 0)
    rfn.fit(Xn,yn)

    dffnn_f=pd.concat([dffn, dffnp])

    Xnn = dffn_f.drop(['Label'],axis=1)
    ynn = dffn_f.iloc[:, -1].values.reshape(-1,1)
    ynn=np.ravel(ynn)

    rfnn = RandomForestClassifier(random_state = 0)
    rfnn.fit(Xnn,ynn)

    X2p = df2.drop(['Label'],axis=1)
    y2p = df2.iloc[:, -1].values.reshape(-1,1)
    y2p=np.ravel(y2p)

    result2 = km_cluster.predict(X2p)

    count=0
    a=np.zeros(n)
```

```
    b=np.zeros(n)
    for v in range(0,n):
        for i in range(0,len(y)):
            if result[i]==v:
                if y[i]==1:
                    a[v]=a[v]+1
                else:
                    b[v]=b[v]+1
    list1=[]
    list2=[]
    l1=[]
    l0=[]
    for v in range(0,n):
        if a[v]<=b[v]:
            list1.append(v)
        else:
            list2.append(v)
    for v in range(0,len(y2p)):
        if result2[v] in list1:
            result2[v]=0
            l0.append(v)
        elif result2[v] in list2:
            result2[v]=1
            l1.append(v)
        else:
            print("-1")
    print(classification_report(y2p, result2))
    cm=confusion_matrix(y2p,result2)
    print(cm)
```

95% of the code has been shared, and the remaining 5% is retained for future extension.
Thank you for your interest and more details are in the paper.