VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

INFORMATION SECURITY ANALYSIS AND AUDIT

CSE-3501

J COMPONENT PROJECT REVIEW

REVIEW 3

By

HRITISHA- 18BIT0226

FACULTY- Prof. Sumaiya Thaseen

NETWORK INTRUSION DETECTION USING MACHINE LEARNING TECHNIQUES

ALGORITHMS- **XGBOOST, ADABOOST and LIGHTGBM**

TEAM MEMBER- NITIN 18BIT0227

REVIEW 3- INTERCHANGING OF THE MODELS

ADABOOST MODEL ON 3 DATASETS

WINDOWS-7 DATASET

In [30]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(Xro,yro, test_size=0.3, random_state=42, stratify=yro)
```

In [31]:
```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=42)
import time
time_start=time.clock()
model.fit(X_train, y_train)
print(model.score(X_test,y_test))
time_elapsed=(time.clock()-time_start)
time_elapsed
```

```
C:\Users\hriti\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: DeprecationWarning: time.clock has been deprecated in Pytho
n 3.3 and will be removed from Python 3.8: use time.perf_counter or time.process_time instead
  after removing the cwd from sys.path.
```

```
0.5
```

```
C:\Users\hriti\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: DeprecationWarning: time.clock has been deprecated in Pytho
n 3.3 and will be removed from Python 3.8: use time.perf_counter or time.process_time instead
  import sys
```

Out[31]:
```
25.328206199999983
```

KFold

:
```
model = AdaBoostClassifier(random_state=42)
def evaluate_model(model):
    cv = StratifiedKFold(n_splits=10)
    scores = cross_val_score(model, X_train, y_train, scoring='accuracy', cv=cv)
    return scores

scores = evaluate_model(model)
```

:
```
from numpy import mean
print('Accuracy after k folds= ',mean(scores)*100,'%')
```

```
Accuracy after k folds=  49.99638336347198 %
```

HYPER PARAMETER TUNING

```
params={'n_estimators': [200,300,400],
          'learning_rate': [0.01,0.1,0.15]}
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(AdaBoostClassifier(),params,refit=True,verbose=3)
grid.fit(X_train,y_train)
grid.best_params_
grid.best_estimator_
grid_predictions = grid.predict(X_test)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] learning_rate=0.01, n_estimators=200 ............................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  learning_rate=0.01, n_estimators=200, score=1.000, total=  44.5s
[CV] learning_rate=0.01, n_estimators=200 ............................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   44.4s remaining:    0.0s
[CV]  learning_rate=0.01, n_estimators=200, score=1.000, total=  44.0s
[CV] learning_rate=0.01, n_estimators=200 ............................
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  1.5min remaining:    0.0s
[CV]  learning_rate=0.01, n_estimators=200, score=1.000, total=  44.9s
[CV] learning_rate=0.01, n_estimators=200 ............................
[CV]  learning_rate=0.01, n_estimators=200, score=1.000, total=  44.6s
```

```
[CV] learning_rate=0.15, n_estimators=400 ............................
[CV]  learning_rate=0.15, n_estimators=400, score=1.000, total= 1.9min
[CV] learning_rate=0.15, n_estimators=400 ............................
[CV]  learning_rate=0.15, n_estimators=400, score=1.000, total= 1.8min
[CV] learning_rate=0.15, n_estimators=400 ............................
[CV]  learning_rate=0.15, n_estimators=400, score=1.000, total= 1.7min
[CV] learning_rate=0.15, n_estimators=400 ............................
[CV]  learning_rate=0.15, n_estimators=400, score=1.000, total= 1.6min
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed: 52.5min finished
```

```
In [47]: grid.best_params_
Out[47]: {'learning_rate': 0.01, 'n_estimators': 200}

In [48]: grid.best_estimator_
Out[48]: AdaBoostClassifier(learning_rate=0.01, n_estimators=200)

In [49]: grid_predictions = grid.predict(X_test)

In [50]: print(confusion_matrix(y_test,grid_predictions))
[[2962    0    0    0    0    0    0    0]
 [   0 2962    0    0    0    0    0    0]
 [   0    0 2962    0    0    0    0    0]
 [   0    0    0 2963    0    0    0    0]
 [   0    0    0    0 2962    0    0    0]
 [   0    0    0    0    0 2962    0    0]
 [   0    0    0    0    0    0 2963    0]
```

Accuracy increases from 50% to 100% after Hyper Parametric Tuning

LINUX MEMORY DATASET

TRAIN TEST SPLIT

In [29]:
```
X_train, X_test, y_train, y_test = model_selection.tr
```

MODEL -ADABOOST

In [30]:
```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=42)
```

In [32]:
```
import time
time_start=time.clock()
model.fit(X_train, y_train)
print(model.score(X_test,y_test))

time_elapsed=(time.clock()-time_start)
time_elapsed
```

```
0.5
```

Out[32]:
```
2.561669199996686
```

K FOLD CROSS VALIDATION

- **KFold**
- **Stratified KFold**
- **Repeated Stratified Kfold**

```
model= AdaBoostClassifier(random_state=42)

def evaluate_model(model):

    KF=KFold(n_splits=10)
    score1 = cross_val_score(model, X_train, y_train, scoring='accuracy', cv=KF)

    SKF= StratifiedKFold(n_splits=10)
    score2 = cross_val_score(model, X_train, y_train, scoring='accuracy', cv=SKF)

    RSKF= RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=None)
    score3 = cross_val_score(model, X_train, y_train, scoring='accuracy', cv=RSKF)

    list_scores=[mean(score1),mean(score2),mean(score3)]

    return list_scores

scores = evaluate_model(model)
```

```
names=["KFold","Stratified KFold","Repeated Stratified KFold"]
print("KFOLD CROSS VALIDATION SCORES")
print("------------------------------------")

for (i,j) in zip(scores,names):
    print(j,"-",round(i*100,2),'%')
```

```
KFOLD CROSS VALIDATION SCORES
------------------------------------
KFold - 49.4 %
Stratified KFold - 50.0 %
Repeated Stratified KFold - 50.0 %
```

HYPER PARAMETRIC TUNING

```
params={'n_estimators': [200,300],
           'learning_rate': [0.01,0.015]}
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(AdaBoostClassifier(),params,refit=True,verbose=3)
```

```
# X_train.iloc[np.random.randint(low=0, high=450000, size=int(0.15 * 450000))]
# X_train1=X_train.sample(frac=0.35)
# y_train1=y_train.sample(frac=0.35)
```

```
X_train.shape
y_train.shape
```

```
(45000,)
```

```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV] learning_rate=0.01, n_estimators=200 ............................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  learning_rate=0.01, n_estimators=200, score=0.786, total=   6.0s
[CV] learning_rate=0.01, n_estimators=200 ............................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    5.9s remaining:    0.0s
[CV]  learning_rate=0.01, n_estimators=200, score=0.751, total=   6.1s
[CV] learning_rate=0.01, n_estimators=200 ............................
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   11.9s remaining:    0.0s
[CV]  learning_rate=0.01, n_estimators=200, score=0.795, total=   5.9s
[CV] learning_rate=0.01, n_estimators=200 ............................
```

```
In [73]: grid.best_params_
```

```
Out[73]: {'learning_rate': 0.01, 'n_estimators': 200}
```

```
In [74]: grid.best_estimator_
```

```
Out[74]: AdaBoostClassifier(learning_rate=0.01, n_estimators=200)
```

```
In [75]: grid_predictions = grid.predict(X_test)
```

```
In [76]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [77]: print(confusion_matrix(y_test,grid_predictions))
```

```
[[1301 1131   68    0    0    0]
 [   8 2381  111    0    0    0]
 [   0 1730  770    0    0    0]
 [   0    0    0 2500    0    0]
 [   0    0    0    0 2500    0]
 [ 157    0    0    0    0 2343]]
```

```
In [78]: print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.89      0.52      0.66      2500
           1       0.45      0.95      0.62      2500
           2       0.81      0.31      0.45      2500
           3       1.00      1.00      1.00      2500
           4       1.00      1.00      1.00      2500
           5       1.00      0.94      0.97      2500

    accuracy                           0.79     15000
   macro avg       0.86      0.79      0.78     15000
weighted avg       0.86      0.79      0.78     15000
```

Accuracy increases from 50% to 86% after Hyper Parametric Tuning

LINUX PROCESS DATASET

MODEL -ADABOOST

```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=42)
```

```
import time
time_start=time.clock()
model.fit(X_train, y_train)
print(model.score(X_test,y_test))

time_elapsed=(time.clock()-time_start)
print(time_elapsed)
```

```
C:\Users\hriti\Anaconda3\lib\site-packages\ipykernel_l
n 3.3 and will be removed from Python 3.8: use time.pe
```

```
0.375
56.777735199999825
```

- **KFold**

In [31]:
```
model = AdaBoostClassifier(random_state=42)
def evaluate_model(model):

    KF=KFold(n_splits=10)
    score1 = cross_val_score(model, X_train, y_train, scoring='accuracy', cv=KF)

    list_scores=[mean(score1)]

    return list_scores

scores = evaluate_model(model)
```

In [32]:
```
names=["KFold"]
print("KFOLD CROSS VALIDATION SCORE")
print("------------------------------------")

for (i,j) in zip(scores,names):
    print(j,"-",round(i*100,2),'%')
```

```
KFOLD CROSS VALIDATION SCORE
------------------------------------
KFold - 37.28 %
```

HYPER PARAMETRIC TUNING

```
params={'n_estimators': [200,300],
          'learning_rate': [0.01,0.015]}
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(AdaBoostClassifier(),params,refit=True,verbose=3)
```

```
# X_train.iloc[np.random.randint(low=0, high=450000, size=int(0.15 * 450000))]
# X_train1=X_train.sample(frac=0.35)
# y_train1=y_train.sample(frac=0.35)
```

```
X_train.shape
y_train.shape
```

```
(600000,)
```

```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] learning_rate=0.01, n_estimators=200 ............................
[CV]  learning_rate=0.01, n_estimators=200, score=0.626, total= 2.4min
[CV] learning_rate=0.01, n_estimators=200 ............................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.4min remaining:    0.0s
[CV]  learning_rate=0.01, n_estimators=200, score=0.626, total= 2.4min
[CV] learning_rate=0.01, n_estimators=200 ............................
```

```
grid.best_estimator_
```

```
AdaBoostClassifier(learning_rate=0.01, n_estimators=300)
```

```
grid_predictions = grid.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,grid_predictions))
```

```
[[10948 13898     0     0     0    20   128     6]
 [    0 24062   436     0     0     0   502     0]
 [   49 17268  7560     0     0     0   123     0]
 [    0     0     0 20666     0   891     0  3443]
 [    0     0     0     0 25000     0     0     0]
 [    0     0     0     0     0 23410     0  1590]
 [  216     0     0     0     0     0 24784     0]
 [    0     0     0     0     0 20336     0  4664]]
```

```
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.98      0.44      0.60     25000
           1       0.44      0.96      0.60     25000
           2       0.95      0.30      0.46     25000
           3       1.00      0.83      0.91     25000
           4       1.00      1.00      1.00     25000
           5       0.52      0.94      0.67     25000
           6       0.97      0.99      0.98     25000
           7       0.48      0.19      0.27     25000

    accuracy                           0.71    200000
   macro avg       0.79      0.71      0.69    200000
```

Accuracy increases from 37.5% to 79% after Hyper Parametric Tuning

LINKS OF THE REVIEW:

VIDEO LINK

https://drive.google.com/file/d/1aXrFiZzfPHQNlLGTJRMooka6JAGYS7bs/view?usp=sharing

FOLDER LINK

https://drive.google.com/drive/folders/16z7YcVvAMViLpxB55V8oUcTFHijDzi2j?usp=sharing

PERFORMANCE ANALYSIS OF THE DIFFERENT MODELS

BASE MODEL ACCURACY (in %)

	LINUX-DISC	LINUX-MEMORY	LINUX-PROCESS	WINDOWS-7	WINDOWS-10
XGBOOST	91.8	97.2	96.4	100	100
ADABOOST	43.0	50.0	37.5	50	37.5
LIGHTGBM	98.8	-	95.2	-	100

ACCURACY after KFold (in %)

	LINUX-DISC	LINUX-MEMORY	LINUX-PROCESS	WINDOWS-7	WINDOWS-10
XGBOOST	92.5	96.9	96.37	100	100
ADABOOST	42.9	49.4	37.28	49.9	42.4
LIGHTGBM	98.8	-	94.8	-	100

ACCURACY after HYPERPARAMETRIC TUNING (in %)

	LINUX-DISC	LINUX-MEMORY	LINUX-PROCESS	WINDOWS-7	WINDOWS-10
XGBOOST	87	97	96.4	100	100
ADABOOST	43.3	86	79	100	37.5
LIGHTGBM	99.2	-	95.2	-	100

CLASSIFICATION REPORT AND CONFUSION MATRIX

LINUX MEMORY

	precision	recall	f1-score	support
0	0.99	1.00	1.00	2500
1	0.88	0.97	0.92	2500
2	0.97	0.86	0.91	2500
3	1.00	1.00	1.00	2500
4	1.00	1.00	1.00	2500
5	1.00	1.00	1.00	2500
accuracy			0.97	15000
macro avg	0.97	0.97	0.97	15000
weighted avg	0.97	0.97	0.97	15000

XGBOOST

```
print(confusion_matrix(y_test,grid_predictions))
```

```
[[2496    1    3    0    0    0]
 [   3 2428   69    0    0    0]
 [  10  341 2149    0    0    0]
 [   0    0    0 2500    0    0]
 [   0    0    0    0 2500    0]
 [   3    0    0    0    0 2497]]
```

	precision	recall	f1-score	support
0	0.89	0.52	0.66	2500
1	0.45	0.95	0.62	2500
2	0.81	0.31	0.45	2500
3	1.00	1.00	1.00	2500
4	1.00	1.00	1.00	2500
5	1.00	0.94	0.97	2500
accuracy			0.79	15000
macro avg	0.86	0.79	0.78	15000
weighted avg	0.86	0.79	0.78	15000

ADABOOST

```
print(confusion_matrix(y_test,grid_predictions))
```

```
[[1301 1131   68    0    0    0]
 [   8 2381  111    0    0    0]
 [   0 1730  770    0    0    0]
 [   0    0    0 2500    0    0]
 [   0    0    0    0 2500    0]
 [ 157    0    0    0    0 2343]]
```

LINUX DISC

	precision	recall	f1-score	support
4	1.000000	1.000000	1.000000	30000.000000
0	0.999867	0.999933	0.999900	30000.000000
3	0.992392	1.000000	0.996181	30000.000000
5	0.994253	0.986133	0.990177	30000.000000
weighted avg	0.989354	0.989308	0.989315	240000.000000
macro avg	0.989354	0.989308	0.989315	240000.000000
accuracy	0.989308	0.989308	0.989308	0.989308
6	0.987116	0.985800	0.986458	30000.000000
1	0.989589	0.979067	0.984300	30000.000000
7	0.984363	0.979933	0.982143	30000.000000
2	0.967253	0.983600	0.975358	30000.000000

LIGHTGBM

```
confmetric
```

```
array([[29995,     0,     0,     0,     0,     0,     5,     0],
       [    0, 29413,   454,    10,     0,    16,    49,    58],
       [    0,   118, 29551,   151,     0,    29,    23,   128],
       [    0,   295,     0, 29705,     0,     0,     0,     0],
       [    0,     0,     0,     0, 30000,     0,     0,     0],
       [    0,    30,    37,    10,     0, 29523,   336,    64],
       [    0,    99,    66,    22,     0,    48, 29578,   187],
       [    0,    19,   437,     7,     0,    31,    74, 29432]],
      dtype=int64)
```

	precision	recall	f1-score	support
4	1.000000	1.000000	1.000000	30000.000000
0	0.753815	0.973200	0.849573	30000.000000
accuracy	0.431333	0.431333	0.431333	0.431333
macro avg	0.485243	0.431333	0.377873	240000.000000
weighted avg	0.485243	0.431333	0.377873	240000.000000
1	0.845513	0.209800	0.336182	30000.000000
2	0.193050	0.864367	0.315610	30000.000000
6	0.339663	0.276667	0.304945	30000.000000
3	0.749901	0.126633	0.216677	30000.000000
5	0.000000	0.000000	0.000000	30000.000000
7	0.000000	0.000000	0.000000	30000.000000

ADABOOST

```
confmetric
```

```
array([[29196,     0,     0,   733,     0,     0,    71,     0],
       [    0,  6294, 22821,     0,     0,     0,   885,     0],
       [    0,   172, 25931,     9,     0,     0,  3888,     0],
       [ 2997,     0, 23204,  3799,     0,     0,     0,     0],
       [    0,     0,     0,     0, 30000,     0,     0,     0],
       [    0,    16, 25860,     9,     0,     0,  4115,     0],
       [ 6538,   639, 14023,   500,     0,     0,  8300,     0],
       [    0,   323, 22484,    16,     0,     0,  7177,     0]],
      dtype=int64)
```

	precision	recall	f1-score	support
4	1.000000	1.000000	1.000000	30000.000000
0	0.999467	0.999467	0.999467	30000.000000
accuracy	0.926758	0.926758	0.926758	0.926758
macro avg	0.928758	0.926758	0.926724	240000.000000
weighted avg	0.928758	0.926758	0.926724	240000.000000
6	0.952455	0.900800	0.925908	30000.000000
5	0.891621	0.947467	0.918696	30000.000000
1	0.922137	0.883100	0.902196	30000.000000
3	0.846331	0.954267	0.897064	30000.000000
7	0.940842	0.837600	0.886224	30000.000000
2	0.877214	0.891367	0.884234	30000.000000

XGBOOST

```
array([[29984,     0,     0,     0,     0,     0,    16,     0],
       [    0, 26493,  1812,  1178,     0,   123,   121,   273],
       [    0,   492, 26741,  1279,     0,   701,    61,   726],
       [    0,   277,   263, 28628,     0,     0,   547,   285],
       [    0,     0,     0,     0, 30000,     0,     0,     0],
       [    0,   390,   320,   555,     0, 28424,   259,    52],
       [   16,   438,   468,   737,     0,  1073, 27024,   244],
       [    0,   640,   880,  1449,     0,  1558,   345, 25128]],
      dtype=int64)
```

LINUX PROCESS

	precision	recall	f1-score	support
0	1.00	1.00	1.00	25000
1	0.76	0.92	0.83	25000
2	0.90	0.70	0.79	25000
3	1.00	1.00	1.00	25000
4	1.00	1.00	1.00	25000
5	1.00	1.00	1.00	25000
6	1.00	1.00	1.00	25000
7	1.00	1.00	1.00	25000
accuracy			0.95	200000
macro avg	0.96	0.95	0.95	200000
weighted avg	0.96	0.95	0.95	200000

XGBOOST

```
print(confusion_matrix(y_test,grid_predictions))
```

```
[[24990    10     0     0     0     0     0     0]
 [   19 23061  1920     0     0     0     0     0]
 [   64  7442 17494     0     0     0     0     0]
 [    0     0     0 25000     0     0     0     0]
 [    0     0     0     0 25000     0     0     0]
 [    0     0     0     0     0 25000     0     0]
 [    0     0     0     0     0     0 25000     0]
 [    0     0     0     0     0     0     0 25000]]
```

	precision	recall	f1-score	support
0	0.98	0.44	0.60	25000
1	0.44	0.96	0.60	25000
2	0.95	0.30	0.46	25000
3	1.00	0.83	0.91	25000
4	1.00	1.00	1.00	25000
5	0.52	0.94	0.67	25000
6	0.97	0.99	0.98	25000
7	0.48	0.19	0.27	25000
accuracy			0.71	200000
macro avg	0.79	0.71	0.69	200000
weighted avg	0.79	0.71	0.69	200000

ADABOOST

```
print(confusion_matrix(y_test,grid_predictions))
```

```
[[10948 13898     0     0     0    20   128     6]
 [    0 24062   436     0     0     0   502     0]
 [   49 17268  7560     0     0     0   123     0]
 [    0     0     0 20666     0   891     0  3443]
 [    0     0     0     0 25000     0     0     0]
 [    0     0     0     0     0 23410     0  1590]
 [  216     0     0     0     0     0 24784     0]
 [    0     0     0     0     0 20336     0  4664]]
```

WINDOWS-7

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3949
1	1.00	1.00	1.00	3949
2	1.00	1.00	1.00	3950
3	1.00	1.00	1.00	3950
4	1.00	1.00	1.00	3950
5	1.00	1.00	1.00	3949
6	1.00	1.00	1.00	3950
7	1.00	1.00	1.00	3950
accuracy			1.00	31597
macro avg	1.00	1.00	1.00	31597
weighted avg	1.00	1.00	1.00	31597

XGBOOST ⟷

```
print(confusion_matrix(y_test,predictions))
```

```
[[3949    0    0    0    0    0    0    0]
 [   0 3949    0    0    0    0    0    0]
 [   0    0 3950    0    0    0    0    0]
 [   0    0    0 3950    0    0    0    0]
 [   0    0    0    0 3950    0    0    0]
 [   0    0    0    0    0 3949    0    0]
 [   0    0    0    0    0    0 3950    0]
 [   0    0    0    0    0    0    0 3950]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2962
1	1.00	1.00	1.00	2962
2	1.00	1.00	1.00	2962
3	1.00	1.00	1.00	2963
4	1.00	1.00	1.00	2962
5	1.00	1.00	1.00	2962
6	1.00	1.00	1.00	2963
7	1.00	1.00	1.00	2962
accuracy			1.00	23698
macro avg	1.00	1.00	1.00	23698
weighted avg	1.00	1.00	1.00	23698

ADABOOST ⟷

```
print(confusion_matrix(y_test,grid_predictions))
```

```
[[2962    0    0    0    0    0    0    0]
 [   0 2962    0    0    0    0    0    0]
 [   0    0 2962    0    0    0    0    0]
 [   0    0    0 2963    0    0    0    0]
 [   0    0    0    0 2962    0    0    0]
 [   0    0    0    0    0 2962    0    0]
 [   0    0    0    0    0    0 2963    0]
 [   0    0    0    0    0    0    0 2962]]
```

WINDOWS-10

	precision	recall	f1-score	support
1	1.000000	1.000	1.000000	3896.000
4	1.000000	1.000	1.000000	3896.000
accuracy	0.375000	0.375	0.375000	0.375
0	0.166667	1.000	0.285714	3896.000
macro avg	0.270833	0.375	0.285714	31168.000
weighted avg	0.270833	0.375	0.285714	31168.000
2	0.000000	0.000	0.000000	3896.000
3	0.000000	0.000	0.000000	3896.000
5	0.000000	0.000	0.000000	3896.000
6	0.000000	0.000	0.000000	3896.000
7	0.000000	0.000	0.000000	3896.000

ADABOOST

```
confmetric = confusion_matrix(y_test,y_pred)
confmetric
```

```
array([[3896,    0,    0,    0,    0,    0,    0,    0],
       [   0, 3896,    0,    0,    0,    0,    0,    0],
       [3896,    0,    0,    0,    0,    0,    0,    0],
       [3896,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0, 3896,    0,    0,    0],
       [3896,    0,    0,    0,    0,    0,    0,    0],
       [3896,    0,    0,    0,    0,    0,    0,    0],
       [3896,    0,    0,    0,    0,    0,    0,    0]], dtype=int64)
```

	precision	recall	f1-score	support
0	1.0	1.0	1.0	3896.0
1	1.0	1.0	1.0	3896.0
2	1.0	1.0	1.0	3896.0
3	1.0	1.0	1.0	3896.0
4	1.0	1.0	1.0	3896.0
5	1.0	1.0	1.0	3896.0
6	1.0	1.0	1.0	3896.0
7	1.0	1.0	1.0	3896.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	31168.0
weighted avg	1.0	1.0	1.0	31168.0

XGBOOST

```
confmetric = confusion_matrix(y_test,y_pred)
confmetric
```

```
array([[3896,    0,    0,    0,    0,    0,    0,    0],
       [   0, 3896,    0,    0,    0,    0,    0,    0],
       [   0,    0, 3896,    0,    0,    0,    0,    0],
       [   0,    0,    0, 3896,    0,    0,    0,    0],
       [   0,    0,    0,    0, 3896,    0,    0,    0],
       [   0,    0,    0,    0,    0, 3896,    0,    0],
       [   0,    0,    0,    0,    0,    0, 3896,    0],
       [   0,    0,    0,    0,    0,    0,    0, 3896]], dtype=int64)
```

	precision	recall	f1-score	support
0	1.0	1.0	1.0	3896.0
1	1.0	1.0	1.0	3896.0
2	1.0	1.0	1.0	3896.0
3	1.0	1.0	1.0	3896.0
4	1.0	1.0	1.0	3896.0
5	1.0	1.0	1.0	3896.0
6	1.0	1.0	1.0	3896.0
7	1.0	1.0	1.0	3896.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	31168.0
weighted avg	1.0	1.0	1.0	31168.0

LIGHTGBM

```
confmetric = confusion_matrix(y_test,y_pred)
confmetric
```

```
array([[3896,    0,    0,    0,    0,    0,    0,    0],
       [   0, 3896,    0,    0,    0,    0,    0,    0],
       [   0,    0, 3896,    0,    0,    0,    0,    0],
       [   0,    0,    0, 3896,    0,    0,    0,    0],
       [   0,    0,    0,    0, 3896,    0,    0,    0],
       [   0,    0,    0,    0,    0, 3896,    0,    0],
       [   0,    0,    0,    0,    0,    0, 3896,    0],
       [   0,    0,    0,    0,    0,    0,    0, 3896]], dtype=int64)
```

COMPUTATION TIME ANALYSIS

XGBOOST

	LINUX-DISC	LINUX-MEMORY	LINUX-PROCESS	WINDOWS-7	WINDOWS-10
NORMAL FIT	3 min	6.87 sec	4.2min	22.7 sec	120sec
HYPER PARAMETRIC TUNING	66.3min	70.5 min	1869.4 min(135 fits)	-(already 100%)	-(already 100%)

ADABOOST

	LINUX-DISC	LINUX-MEMORY	LINUX-PROCESS	WINDOWS-7	WINDOWS-10
NORMAL FIT	102.9min	2.56 sec	56.7sec	25.3sec	41.66sec
HYPER PARAMETRIC TUNING	32.4min(36 fits)	2.5min(20 fits)	61.1 min (20 fits)	52.5 min	27.4min(192 fits)

LIGHTGBM

	LINUX-DISC	LINUX-MEMORY	LINUX-PROCESS	WINDOWS-7	WINDOWS-10
NORMAL FIT	26sec	-	40sec	-	18.36sec
HYPER PARAMETRIC TUNING	1591.8 min(192 fits)	-	-	-	39.5min(192 fits)

END

THANK YOU

.....

Hritisha

Digitally signed by Hritisha
Date: 2020.11.06 23:42:54 +05'30'