In [2]:
```python
import pandas as pd
import numpy as np
from sklearn import preprocessing
from IPython.display import display, HTML
```

In [3]:
```python
import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```python
# getting the dataset
df = pd.read_csv('C:/Users/nayak/Downloads/malware_BinaryImbalanced.csv')
```

In [5]:
```python
# selecting columns specified in readme file
df = df.loc[:, ['classification', 'os', 'usage_counter', 'prio', 'static_prio', 'normal_prio', 'vm_pgoff', 'vm_trunc
ate_count', 'task_size', 'map_count', 'hiwater_rss', 'total_vm', 'shared_vm', 'exec_vm', 'reserved_vm', 'nr_ptes',
'nvcsw', 'nivcsw', 'signal_nvcsw']]
df.head()
```

Out[5]:

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiwater_rss	total_vm	sha
0	benign	Ubuntu	0	3069403136	16447	0	0	14739	0	7903	0	88	
1	benign	CentOS	0	3069403136	16447	0	0	14739	0	7903	0	88	
2	benign	Ubuntu	0	3069403136	16447	0	0	14739	0	7903	0	88	
3	benign	CentOS	0	3069403136	16447	0	0	14739	0	7903	0	88	
4	benign	Mac	0	3069403136	16447	0	0	14739	0	7903	0	88	

In [6]:
```python
# checking if the data has null values
df.isna().sum()
```

Out[6]:
```
classification       0
os                   0
usage_counter        0
prio                 0
static_prio          0
normal_prio          0
vm_pgoff             0
vm_truncate_count    0
task_size            0
map_count            0
hiwater_rss          0
total_vm             0
shared_vm            0
exec_vm              0
reserved_vm          0
nr_ptes              0
nvcsw                0
nivcsw               0
signal_nvcsw         0
dtype: int64
```

In [7]:
```python
y = df['classification'] # define label as nominal values
le = preprocessing.LabelEncoder()
le.fit(y)
y_encoded = le.transform(y) # encode nominal labels to integers
df['classification'] = y_encoded
```

In [8]:
```python
df_num=df.copy(deep=True)
# create new binary columns
df_dummies=pd.get_dummies(df_num['os'])
df_dummies = df_dummies.astype(int)
# add them to dataframe
df_num=df_num.join(df_dummies)
# drop original columns
df_num.drop('os', axis=1, inplace=True)
```

In [9]:
```python
df_num.head()
```

Out[9]:

	classification	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiwater_rss	...	reserved_vm	nr_pt
0	0	0	3069403136	16447	0	0	14739	0	7903	0	...	80	
1	0	0	3069403136	16447	0	0	14739	0	7903	0	...	80	
2	0	0	3069403136	16447	0	0	14739	0	7903	0	...	80	
3	0	0	3069403136	16447	0	0	14739	0	7903	0	...	80	
4	0	0	3069403136	16447	0	0	14739	0	7903	0	...	80	

5 rows × 23 columns

In [10]:
```python
# Converting n dummy variables to n-1 dummy variables
df_num.drop('CentOS', axis= 1, inplace= True)
```

In [11]:
```python
df.value_counts('classification')
```

Out[11]:
```
classification
0    89899
1    10101
Name: count, dtype: int64
```

In [47]:
```python
# Logistic Regression - Considering the data.
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import make_scorer, precision_score, accuracy_score, roc_auc_score, recall_score, f1_score, auc
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix as cm

x = df_num.drop('classification', axis = 1)
y = df['classification']

# Logistic Regression by Hold - Out Evaluation

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25) # splitting with 75% train and 25% test

clf=LogisticRegression(penalty='l2',solver='newton-cg', max_iter=150, multi_class='ovr')
clf=clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)

conf_Matrix = cm(y_test, y_pred)
print('Confusion Matrix\n', conf_Matrix)
accuracy=accuracy_score(y_test, y_pred)
precision=precision_score(y_test, y_pred, average='macro')
recall=recall_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Logistic Regression By hold-out evaluation: accuracy = ',accuracy, ', precison = ', precision, ', recall = ',
recall)
print('Area under Curve = ',auc_value, ', f1 score = ', f1)
```

```
Confusion Matrix
 [[21846   568]
 [ 1960   626]]
Logistic Regression By hold-out evaluation: accuracy =  0.89888 , precison =  0.7209779610196378 , recall =  0.608365
6973037305
Area under Curve =  0.9017054685810872 , f1 score =  0.6382609969801661
```

In [48]:
```python
# Logistic Regression by N-Fold Cross Validation
clf=LogisticRegression(penalty='l2',solver='lbfgs')
precision = make_scorer(precision_score, average='macro')
accuracy = cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
precision = cross_val_score(clf, x, y, cv=5, scoring=precision).mean()
print('By N-fold Cross Validation: accuracy = ',accuracy, ', precision = ', precision)
```

```
By N-fold Cross Validation: accuracy =  0.8495099999999999 , precision =  0.7557763606778903
```

In [17]:
```python
# Support Vector Machine - Hold Out Evaluation
from sklearn.svm import SVC

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
clf = SVC(kernel='poly', C=0.1, max_iter=-1, probability=True)
clf = clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print('Support Vector Machine accuracy by hold-out evaluation: ',accuracy)
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Logistic Regression By hold-out evaluation: accuracy = ',accuracy, ', precison = ', precision, ', recall = ',
recall)
print('Area under Curve = ',auc_value, ', f1 score = ', f1)
```

```
Support Vector Machine accuracy by hold-out evaluation:  0.8992
Logistic Regression By hold-out evaluation: accuracy =  0.8992 , precison =  0.8495099999999999 , recall =  0.89828
Area under Curve =  0.6796002619612496 , f1 score =  0.8992
```

In [46]:
```python
# Bagging - Random Forest - Hold Out Evaluation
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

x = df_num.drop('classification', axis = 1)
y = df['classification']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
tree = DecisionTreeClassifier(max_depth=3)
bag = BaggingClassifier(tree, n_estimators=10, max_samples=0.750, random_state=1, bootstrap=False)
clf = bag.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Bagging Random Forest by hold-out evaluation: accuracy = ',accuracy, ', precision = ', precision, ', area und
er the curve = ', auc_value, ', f1 score', f1)
```

```
Bagging Random Forest by hold-out evaluation: accuracy =  0.96528 , precision =  0.9676789680665413 , area under the
curve =  0.986084613266666 , f1 score 0.889246274658092
```

In [45]:
```python
# Bagging - Random Forest - Cross Validation
tree = DecisionTreeClassifier(max_depth=3)
bag = BaggingClassifier(tree, n_estimators=10, max_samples=0.75, random_state=1)
precision = make_scorer(precision_score, average='macro')
accuracy = cross_val_score(bag, x, y, cv=5, scoring='accuracy').mean()
precision = cross_val_score(bag, x, y, cv=5, scoring=precision).mean()
print("RandomForest Accuracy by N-fold Cross Validation: accuracy = ",accuracy, "precision = ", precision)
```

```
RandomForest Accuracy by N-fold Cross Validation: accuracy =  0.8449199999999999 precision =  0.5519127196131544
```

In [44]:
```python
# Bagging using Gaussian Naive Bayes - Hold Out Evaluation
from sklearn.naive_bayes import GaussianNB

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
clf = GaussianNB()
bag = BaggingClassifier(tree, n_estimators=10, max_samples=0.750, random_state=1, bootstrap=False)
clf = bag.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Bagging Naive Bayes by hold-out evaluation: accuracy = ',accuracy, ', precision = ', precision, ', area under
the curve = ', auc_value, ', f1 score', f1)
```

```
Bagging Naive Bayes by hold-out evaluation: accuracy =  0.9522 , precision =  0.883021005178763 , area under the curv
e =  0.9737689699204959 , f1 score 0.8570316894966999
```

In [50]:
```python
clf = GaussianNB()
bag = BaggingClassifier(clf, n_estimators=10, max_samples=0.75, random_state=1)
precision = make_scorer(precision_score, average='macro')
accuracy = cross_val_score(bag, x, y, cv=5, scoring='accuracy').mean()
precision = cross_val_score(bag, x, y, cv=5, scoring=precision).mean()
print("Bagging using GaussianNB Accuracy by N-fold Cross Validation: accuracy = ",accuracy, "precision = ", precisio
n)
```

```
Bagging using GaussianNB Accuracy by N-fold Cross Validation: accuracy =  0.7659100000000001 precision =  0.652326322
875741
```

In [72]:
```python
# Boosting Algorithms - AdaBoost
from sklearn.ensemble import AdaBoostClassifier

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
tree = DecisionTreeClassifier(max_depth=5)
clf1 = AdaBoostClassifier(tree, n_estimators=5, random_state=0)
clf1.fit(x_train, y_train)
y_pred = clf1.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
y_pred_proba = clf1.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Ada Boost Classifier hold-out evaluation: accuracy = ',accuracy, ', precision = ', precision, ', area under t
he curve = ', auc_value, ', f1 score', f1)
```

```
Ada Boost Classifier hold-out evaluation: accuracy =  0.99696 , precision =  0.9972378203626321 , area under the curv
e =  0.9990133145722211 , f1 score 0.9916032853180314
```

In [75]:
```python
# Ada Boosting using Cross Validation
tree = DecisionTreeClassifier()
clf = AdaBoostClassifier(tree, n_estimators=100, random_state=0)
precision = make_scorer(precision_score, average='macro')
accuracy = cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
precision = cross_val_score(clf, x, y, cv=5, scoring=precision).mean()
print("AdaBoosting Accuracy by N-fold Cross Validation: acc = ",accuracy, "precision = ", precision)
```

```
AdaBoosting Accuracy by N-fold Cross Validation: acc =  0.8740599999999998 precision =  0.785697537153039
```

In [74]:
```python
# Boosting Algorithms - Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
clf = GradientBoostingClassifier(n_estimators=100, random_state=0, learning_rate=0.001, criterion='squared_error')
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Gradient boosting by hold-out evaluation: accuracy = ',accuracy, ', precision = ', precision, ', area under t
he curve = ', auc_value, ', f1 score', f1)
```

```
Gradient boosting by hold-out evaluation: accuracy =  0.89804 , precision =  0.44902 , area under the curve =  0.9844
048935199955 , f1 score 0.4731407135782175
```

In [76]:
```python
# Gradient Boosting using Cross Validation
clf = GradientBoostingClassifier(n_estimators=100, random_state=0, learning_rate=0.001, criterion='squared_error')
precision = make_scorer(precision_score, average='macro')
accuracy=cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
precision=cross_val_score(clf, x, y, cv=5, scoring=precision).mean()
print("AdaBoosting Accuracy by N-fold Cross Validation: acc = ",accuracy, "precision = ", precision)
```

```
AdaBoosting Accuracy by N-fold Cross Validation: acc =  0.8989900000000001 precision =  0.44949500000000003
```

In [79]:
```python
# XG Boost with Hold-Out Evaluation
from xgboost import XGBClassifier

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
clf = XGBClassifier()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('XG Boost by hold-out evaluation: accuracy = ',accuracy, ', precision = ', precision, ', area under the curve
= ', auc_value, ', f1 score', f1)
```

```
XG Boost by hold-out evaluation: accuracy =  0.99996 , precision =  0.9998012718600954 , area under the curve =  0.99
99999911582318 , f1 score 0.9998894974089773
```

In [81]:
```python
# XG Boost with Cross Validation cv = 5
clf = XGBClassifier()
precision = make_scorer(precision_score, average='macro')
accuracy = cross_val_score(clf, x, y, cv=5, scoring='accuracy').mean()
precision = cross_val_score(clf, x, y, cv=5, scoring=precision).mean()
print("XG Boosting Accuracy by N-fold Cross Validation: accuracy = ",accuracy, "precision = ", precision)
```

```
XG Boosting Accuracy by N-fold Cross Validation: accuracy =  0.86911 precision =  0.7415457497980626
```

In [82]:
```python
# First, check the distribution of labels
print(df_num.groupby(['classification']).size())# - data is imbalanced as it has more rows with classsification 0 th
an 1
```

```
classification
0    89899
1    10101
dtype: int64
```

In [86]:
```python
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

x = df_num.drop('classification',axis=1)
y = df_num['classification']

ros = RandomOverSampler(random_state=10)
ros.fit(x, y)
print('\nOriginal dataset shape {}'.format(Counter(y)))
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After oversampling dataset shape {}'.format(Counter(y_resampled)))

print('\nOriginal dataset shape {}'.format(Counter(y)))
ros = RandomUnderSampler(random_state=30)
ros.fit(x, y)
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After undersampling dataset shape {}'.format(Counter(y_resampled)))

ros = SMOTE(k_neighbors=2)
ros.fit(x, y)
print('\nOriginal dataset shape {}'.format(Counter(y)))
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After oversampling by SMOTE dataset shape {}'.format(Counter(y_resampled)))
```

```
Original dataset shape Counter({0: 89899, 1: 10101})
After oversampling dataset shape Counter({0: 89899, 1: 89899})

Original dataset shape Counter({0: 89899, 1: 10101})
After undersampling dataset shape Counter({0: 10101, 1: 10101})

Original dataset shape Counter({0: 89899, 1: 10101})
After oversampling by SMOTE dataset shape Counter({0: 89899, 1: 89899})
```

In [94]:
```python
# checking values with zero for all rows
df.describe()
```

Out[94]:

	classification	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiwater_rss	total
count	100000.000000	100000.0	1.000000e+05	100000.000000	100000.0	100000.0	100000.000000	100000.0	100000.00000	100000.0	100000.00
mean	0.101010	0.0	3.069706e+09	18183.900070	0.0	0.0	15312.739510	0.0	8771.13948	0.0	266.49
std	0.301343	0.0	2.963061e+05	4609.792765	0.0	0.0	3256.475008	0.0	3785.30516	0.0	311.99
min	0.000000	0.0	3.069190e+09	13988.000000	0.0	0.0	9695.000000	0.0	2588.00000	0.0	4.00
25%	0.000000	0.0	3.069446e+09	14352.000000	0.0	0.0	12648.000000	0.0	6428.00000	0.0	99.00
50%	0.000000	0.0	3.069698e+09	16159.000000	0.0	0.0	15245.000000	0.0	7865.00000	0.0	177.00
75%	0.000000	0.0	3.069957e+09	22182.000000	0.0	0.0	17663.000000	0.0	10684.00000	0.0	327.00
max	1.000000	0.0	3.070222e+09	31855.000000	0.0	0.0	27157.000000	0.0	28184.00000	0.0	2810.00

In [96]:
```python
# dropping rows with all zero values
df_num.drop(['usage_counter', 'normal_prio', 'vm_pgoff', 'task_size', 'hiwater_rss', 'nr_ptes', 'signal_nvcsw'], axi
s = 1, inplace=True)
```

In [99]:
```python
from sklearn.model_selection import KFold
from sklearn import neighbors
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.impute import SimpleImputer

df_knn = df_num.copy(deep=True)
# normalize featureS
for col in df_knn.columns:
    if col != 'classification':# removing classification as it is label
        df_knn[col]=(df_knn[col]-df_knn[col].min())/(df_knn[col].max()-df_knn[col].min())

X = df_knn.loc[:, df_knn.columns!='classification']
y = df_knn.loc[:,'classification']

kf = KFold(n_splits=5, shuffle=True)
data_5folds = []
for train_index, test_index in kf.split(X,y):
    print("\nTRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    fold = [x_train, x_test, y_train, y_test]
    data_5folds.append(fold)

for k in range(5, 6, 2):
    acc_5folds = []
    for x_train, x_test, y_train, y_test in data_5folds:
        print('\nOriginal dataset shape {}'.format(Counter(y_train)))
        ros = RandomOverSampler(random_state=10)
        ros.fit(x_train, y_train)
        x_resampled, y_resampled = ros.fit_resample(x_train, y_train)
        print('After oversampling dataset shape {}'.format(Counter(y_resampled)))
        clf=neighbors.KNeighborsClassifier(k, weights='uniform')
        clf.fit(x_resampled, y_resampled)
        y_pred = clf.predict(x_test)
        acc = accuracy_score(y_test, y_pred)
        acc_5folds.append(acc)
    print('k = ',k,'Accuracy on 5-folds: ', np.mean(acc_5folds))
```

```
TRAIN: [    0     2     3 ... 99997 99998 99999] TEST: [    1    13    23 ... 99972 99985 99994]

TRAIN: [    0     1     2 ... 99996 99997 99999] TEST: [   12    14    22 ... 99986 99995 99998]

TRAIN: [    0     1     2 ... 99997 99998 99999] TEST: [    7     8    16 ... 99987 99991 99992]

TRAIN: [    0     1     5 ... 99995 99996 99998] TEST: [    2     3     4 ... 99989 99997 99999]

TRAIN: [    1     2     3 ... 99997 99998 99999] TEST: [    0     5     9 ... 99990 99993 99996]

Original dataset shape Counter({0: 71970, 1: 8030})
After oversampling dataset shape Counter({0: 71970, 1: 71970})

Original dataset shape Counter({0: 71925, 1: 8075})
After oversampling dataset shape Counter({0: 71925, 1: 71925})

Original dataset shape Counter({0: 71887, 1: 8113})
After oversampling dataset shape Counter({0: 71887, 1: 71887})

Original dataset shape Counter({0: 71896, 1: 8104})
After oversampling dataset shape Counter({0: 71896, 1: 71896})

Original dataset shape Counter({0: 71918, 1: 8082})
After oversampling dataset shape Counter({0: 71918, 1: 71918})
k =  5 Accuracy on 5-folds:  0.9999400000000002
```

In [102]:
```python
# Decision tree bagging after oversampling.

# Over sampling the train set.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
ros = RandomOverSampler(random_state=10)
ros.fit(x_train, y_train)
print('\nOriginal dataset shape {}'.format(Counter(y_train)))
x_resampled, y_resampled = ros.fit_resample(x_train, y_train)
print('After oversampling dataset shape {}'.format(Counter(y_resampled)))

tree = DecisionTreeClassifier(max_depth=5) # max depth increased to 5 from 3
bag = BaggingClassifier(tree, n_estimators=10, max_samples=0.750, random_state=1, bootstrap=False)
clf = bag.fit(x_resampled, y_resampled) #using the resampled data to train
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Bagging Random Forest by hold-out evaluation: accuracy = ',accuracy, ', precision = ', precision, ', area und
er the curve = ', auc_value, ', f1 score', f1)
```

```
Original dataset shape Counter({0: 67394, 1: 7606})
After oversampling dataset shape Counter({0: 67394, 1: 67394})
Bagging Random Forest by hold-out evaluation: accuracy =  0.97 , precision =  0.884738305157203 , area under the curv
e =  0.9956672821314703 , f1 score 0.9260842261373272
```

In [103]:
```python
# using XG Boost with resampled data.
clf = XGBClassifier()
clf.fit(x_resampled, y_resampled)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('XG Boost by hold-out evaluation: accuracy = ',accuracy, ', precision = ', precision, ', area under the curve
= ', auc_value, ', f1 score', f1)
```

```
XG Boost by hold-out evaluation: accuracy =  0.99996 , precision =  0.9999777837021239 , area under the curve =  0.99
99997506677429 , f1 score 0.9998886711192074
```

In [104]:
```python
# logistic regression with resampled train data

clf=LogisticRegression(penalty='l2',solver='newton-cg', max_iter=150, multi_class='ovr')
clf=clf.fit(x_resampled, y_resampled)
y_pred=clf.predict(x_test)

conf_Matrix = cm(y_test, y_pred)
print('Confusion Matrix\n', conf_Matrix)
accuracy=accuracy_score(y_test, y_pred)
precision=precision_score(y_test, y_pred, average='macro')
recall=recall_score(y_test, y_pred, average='macro')
y_pred_proba = clf.predict_proba(x_test)
auc_value = roc_auc_score(y_test, y_pred_proba[:,1])
f1 = f1_score(y_test, y_pred, average='macro')
print('Logistic Regression By hold-out evaluation: accuracy = ',accuracy, ', precison = ', precision, ', recall = ',
recall)
print('Area under Curve = ',auc_value, ', f1 score = ', f1)
```

```
Confusion Matrix
 [[17411  5094]
 [   90  2405]]
Logistic Regression By hold-out evaluation: accuracy =  0.79264 , precison =  0.6577834323208125 , recall =  0.868789
0778223855
Area under Curve =  0.8971396158235155 , f1 score =  0.6758541051741979
```