Configuration Manual

MSc Research Project
Programme Name

Samita Ramesh Babu
Student ID: 22132201

School of Computing
National College of Ireland

Supervisor: Evgeniia Jayasekera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Samita Ramesh Babu
Student ID:	22132201
Programme:	MSc Cybersecurity **Year:** 2023-2024
Module:	MSc Research Project
Lecturer:	Evgeniia Jayasekera
Submission Due Date:	31 Jan 2024
Project Title:	Unmasking Memory Malware: A Comparative Analysis of Machine Learning and Deep Learning using Ensemble Approaches"
Word Count:	…………780……………… **Page Count:** 13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Samita

Date: 31 Jan 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Samita Ramesh Babu
Student ID: 22132201

1 System Requirements

The project needs following software and hardware requirements to ensure the implementation of project goes smoothly.

1.1 Hardware Requirements

1. Processor 12th Gen Intel(R) Core (TM) i5-12500H 2.50 GHz
2. Installed RAM 16.0 GB (15.7 GB usable)
3. System type 64-bit operating system, x64-based processor

These are hardware requirements needed for the execute the following. The specification is recommended but not the maximum.

1.2 Software Requirements

1. Windows OS 10
2. Jupyter Notebook 7

2 Data Preprocessing

2.1 Importing Libraries

```
import pandas as psAnaly_MMRY
from sklearn import preprocessing as sscAnaly_MMRY
from imblearn.over_sampling import SMOTE as omtAnaly_MMRY
```

```
import warnings as ngsAnaly_MMRY
ngsAnaly_MMRY.filterwarnings("ignore")
import time as tngsAnaly_MMRY
from sklearn.metrics import classification_report as crngsAnaly_MMRY
from sklearn.metrics import confusion_matrix as congsAnaly_MMRY
from sklearn.metrics import ConfusionMatrixDisplay as cdngsAnaly_MMRY
from sklearn.model_selection import GridSearchCV as gdngsAnaly_MMRY
```

```
from sklearn.neural_network import MLPClassifier as gdngsAnaly_MMRY_M
from sklearn.ensemble import AdaBoostClassifier as gdngsAnaly_MMRY_A
from sklearn.naive_bayes import GaussianNB as gdngsAnaly_MMRY_G
from sklearn.ensemble import BaggingClassifier as gdngsAnaly_MMRY_B
from sklearn.linear_model import SGDClassifier as gdngsAnaly_MMRY_S
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

All the libraries like pandas, sklearn, seaborn many more as mentioned in above figure are necessary for data preprocessing.

2.2 Loading the data

```
In [1]: import pandas as psAnaly_MMRY

        Analy_MMRY = psAnaly_MMRY.read_csv('Obfuscated-MalMem2022.csv')
        Analy_MMRY.shape

Out[1]: (58596, 57)

In [2]: Analy_MMRY

Out[2]:
```

Category	pslist.nproc	pslist.nppid	pslist.avg_threads	pslist.nprocs64bit	pslist.avg_handlers	dlllist.ndlls	dlllist.avg_dlls_per_proc	handles.nhar
Benign	45	17	10.555556	0	202.844444	1694	38.500000	
Benign	47	19	11.531915	0	242.234043	2074	44.127660	
Benign	40	14	14.725000	0	288.225000	1932	48.300000	
Benign	32	13	13.500000	0	264.281250	1445	45.156250	
Benign	42	16	11.452381	0	281.333333	2067	49.214286	
...	...	...	...	...	...	...	...	
Ransomware-Shade-ɔe3078d1b9840f06745f160eb...	37	15	10.108108	0	215.486487	1453	39.270270	
Ransomware-Shade-37137caf9a67678cde91e4614...	37	14	9.945946	0	190.216216	1347	36.405405	
Ransomware-Shade-eea111a25da4d0888f3044ae9...	38	15	9.842105	0	210.026316	1448	38.105263	
Ransomware-Shade-c086af2e1d8ebaa6f2c863157...	37	15	10.243243	0	215.513513	1452	39.243243	
Ransomware-Shade-af38346c1755527bd196668e...	38	15	9.868421	0	213.026316	1487	39.131579	

: 57 columns

Dataset Obfuscated-MalMem2022 is the dataset which is named as Analy_MMRY is loaded using pandas. The dataset consists of total records of 58596 and 57 columns. Each columns are of various attributes related to malware memory.

2.3 Creating an Ouput column for Category column

```
for con in list(range(Analy_MMRY.shape[0])):
  name = Analy_MMRY['Category'][con].split('-')
  if name[0]== 'Benign':
    Analy_MMRY['Category'][con]= 'ben'
  elif name[0]== 'Ransomware':
    Analy_MMRY['Category'][con]= 'ransom'
  elif name[0]== 'Trojan':
    Analy_MMRY['Category'][con]= 'trojan'
  elif name[0]== 'Spyware':
    Analy_MMRY['Category'][con]= 'spy'

Analy_MMRY['Category'].value_counts()
```

```
Category
ben        29298
spy        10020
ransom      9791
trojan      9487
Name: count, dtype: int64
```

Creating a new column for Category column mapping with Benign, Ransomware, Trojan and Spyware to shorter labels as ben, ransom, trojan and spy.

```
In [5]: Analy_MMRY.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58596 entries, 0 to 58595
Data columns (total 57 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   Category                               58596 non-null  object
 1   pslist.nproc                           58596 non-null  int64
 2   pslist.nppid                           58596 non-null  int64
 3   pslist.avg_threads                     58596 non-null  float64
 4   pslist.nprocs64bit                     58596 non-null  int64
 5   pslist.avg_handlers                    58596 non-null  float64
 6   dlllist.ndlls                          58596 non-null  int64
 7   dlllist.avg_dlls_per_proc              58596 non-null  float64
 8   handles.nhandles                       58596 non-null  int64
 9   handles.avg_handles_per_proc           58596 non-null  float64
 10  handles.nport                          58596 non-null  int64
 11  handles.nfile                          58596 non-null  int64
 12  handles.nevent                         58596 non-null  int64
 13  handles.ndesktop                       58596 non-null  int64
 14  handles.nkey                           58596 non-null  int64
 15  handles.nthread                        58596 non-null  int64
 16  handles.ndirectory                     58596 non-null  int64
 17  handles.nsemaphore                     58596 non-null  int64
 18  handles.ntimer                         58596 non-null  int64
 19  handles.nsection                       58596 non-null  int64
 20  handles.nmutant                        58596 non-null  int64
 21  ldrmodules.not_in_load                 58596 non-null  int64
 22  ldrmodules.not_in_init                 58596 non-null  int64
 23  ldrmodules.not_in_mem                  58596 non-null  int64
 24  ldrmodules.not_in_load_avg             58596 non-null  float64
 25  ldrmodules.not_in_init_avg             58596 non-null  float64
 26  ldrmodules.not_in_mem_avg              58596 non-null  float64
 27  malfind.ninjections                    58596 non-null  int64
```

The info() command shows each columns in the dataset along with Null and data type.

Removing Unique elements, null values and dupicate data.

```
### deleting unique element columns
for con in list(Analy_MMRY.columns):
  if Analy_MMRY[con].nunique()==1:
    del Analy_MMRY[con]
Analy_MMRY.shape
```

```
(58596, 54)
```

```
display("Sum of Nan data", Analy_MMRY.isnull().values.sum())
```

```
'Sum of Nan data'

0
```

```
display("Sum of Duplicate data", (Analy_MMRY[Analy_MMRY.duplicated()]).shape)
```

```
'Sum of Duplicate data'

(559, 54)
```

After removing the duplicates and unnecessary columns now there are 54 columns and 58037 values.

2.4 Label Encoding

Label encoding***

```
from sklearn import preprocessing as sscAnaly_MMRY

sscAnaly_MMRY_V = sscAnaly_MMRY.LabelEncoder()

Analy_MMRY['Category']= sscAnaly_MMRY_V.fit_transform(Analy_MMRY['Category'])
Analy_MMRY['Class']= sscAnaly_MMRY_V.fit_transform(Analy_MMRY['Class'])
Analy_MMRY
```

	Category	pslist.nproc	pslist.nppid	pslist.avg_threads	pslist.avg_handlers	dlllist.ndlls	dlllist.avg_dlls_per_proc	handles.nhandles	handles.avg_handles_p
0	0	45	17	10.555556	202.844444	1694	38.500000	9129	212
1	0	47	19	11.531915	242.234043	2074	44.127660	11385	242
2	0	40	14	14.725000	288.225000	1932	48.300000	11529	288
3	0	32	13	13.500000	264.281250	1445	45.156250	8457	264
4	0	42	16	11.452381	281.333333	2067	49.214286	11816	281
...	...	...	...	...	...	...	...	...	
58591	1	37	15	10.108108	215.486487	1453	39.270270	7973	215
58592	1	37	14	9.945946	190.216216	1347	36.405405	7038	190
58593	1	38	15	9.842105	210.026316	1448	38.105263	7982	215
58594	1	37	15	10.243243	215.513513	1452	39.243243	7974	215
58595	1	38	15	9.868421	213.026316	1487	39.131579	8095	213

58037 rows × 54 columns

The above snippet converts category and class columns into numeric form to make into machine readable format. Now the category column is converted into 0, 1, 2, 3 and class is converted into 0 and 1 for benign and malware.

```
Analy_MMRY.to_csv('Analy_Memory_Malware.csv', index=False)
```

After all the preprocessing steps, new dataset is created 'Analy_Memory_Malware.csv'. Using this dataset model building for all the models is done.

3 Model Building

3.1 Oversampling

```
from imblearn.over_sampling import SMOTE as omtAnaly_MMRY

omtAnaly_MMRY_O = omtAnaly_MMRY()
x, y = omtAnaly_MMRY_O.fit_resample(x, y)

from collections import Counter as coAnaly_MMRY
print('Smote sampling results %s' % coAnaly_MMRY(y))
```

```
Smote sampling results Counter({0: 29231, 1: 29231, 2: 29231, 3: 29231})
```

SMOTE Technique was used for address the imbalance in the dataset. Fit_resample methods is used to oversample the minority classes that is malware in the dataset using SMOTE.

3.2 Training and Testing Split

```
from sklearn.model_selection import train_test_split as SomtAnaly_MMRY

qq =0.4
dd=0.5
Rs=91
X_Analy_MMRY_NN, X_Analy_MMRY_S, Y_Analy_MMRY_NN, Y_Analy_MMRY_S = SomtAnaly_MMRY(x, y, test_size=qq, random_state=Rs)
X_Analy_MMRY_LL, X_Analy_MMRY_S, Y_Analy_MMRY_LL, Y_Analy_MMRY_S = SomtAnaly_MMRY(X_Analy_MMRY_S, Y_Analy_MMRY_S, test_size=dd,
```

Importing train and test split using sklearn. The test is split into two initial and second split. X_Analy_MMRY_NN and Y_ Analy_MMRY_NN are training files, X_Analy_MMRY_LL and Y_Analy_MMRY_LL are validation files, X_Analy_MMRY_S and Y_Analy_MMRY_S are testing files. X contains feature values of data and Y contains corresponding category labels.

3.3 Model Training

Hyperparameters are set for each model building and GridSearchCV function is used for all the algorithms. Each Model is trained individually and in combination. Different variable names are given.

3.3.1 MLP Classifier

```
from sklearn.neural_network import MLPClassifier as gdngsAnaly_MMRY_M

Individual_vrbbl = {'solver': ['lbfgs', 'sgd', 'adam'],
               'batch_size': [50, 100, 200],
               'alpha':[0.1, 0.01, 0.001]
               }
Individual_vrbbl_MD = gdngsAnaly_MMRY_M(random_state=Rs)
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 27 candidates, totalling 54 fits
{'alpha': 0.001, 'batch_size': 200, 'solver': 'adam'}
```

Classification Report

	precision	recall	f1-score	support
0	1.00	0.98	0.99	5849
1	0.52	0.18	0.27	5839
2	0.32	0.85	0.47	5762
3	0.84	0.07	0.13	5935
accuracy			0.52	23385
macro avg	0.67	0.52	0.46	23385
weighted avg	0.67	0.52	0.46	23385

```
Time Gap taken for Testing 0.25720906257629395
```

3.3.2 AdaBoost Classifier

```
from sklearn.ensemble import AdaBoostClassifier as gdngsAnaly_MMRY_A

Individual_vrbbl = {'n_estimators': [50, 20, 10],
               'algorithm': ['SAMME', 'SAMME.R'],
               'learning_rate':[0.1, 0.01, 0.001]
               }
Individual_vrbbl_MD = gdngsAnaly_MMRY_A(random_state=Rs)
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 18 candidates, totalling 36 fits

{'algorithm': 'SAMME.R', 'learning_rate': 0.1, 'n_estimators': 50}
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	5849
1	0.48	0.39	0.43	5839
2	0.60	0.41	0.49	5762
3	0.49	0.73	0.59	5935
accuracy			0.63	23385
macro avg	0.64	0.63	0.63	23385
weighted avg	0.64	0.63	0.63	23385

```
Time Gap taken for Testing 0.3459053039550781
```

3.3.3 Gaussian Naïve Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB as gdngsAnaly_MMRY_G

Individual_vrbbl = {'var_smoothing': [1e-03, 1e-05, 0.1, 1e-09]
                 }
Individual_vrbbl_MD = gdngsAnaly_MMRY_G()
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits

{'var_smoothing': 1e-09}
```

Classification Report

	precision	recall	f1-score	support
0	0.99	0.99	0.99	5849
1	0.51	0.04	0.08	5839
2	0.49	0.19	0.27	5762
3	0.37	0.93	0.53	5935
accuracy			0.54	23385
macro avg	0.59	0.54	0.47	23385
weighted avg	0.59	0.54	0.47	23385

```
Time Gap taken for Testing 0.12633013725280762
```

3.3.4 Bagging Classifier

```
from sklearn.ensemble import BaggingClassifier as gdngsAnaly_MMRY_B

Individual_vrbbl = {'n_estimators': [50, 20, 10],
                    'max_features': [30, 40, 50]
                   }
Individual_vrbbl_MD = gdngsAnaly_MMRY_B(random_state=Rs)
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits

{'max_features': 50, 'n_estimators': 50}
```

Classification Report

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5849
           1       0.87      0.88      0.88      5839
           2       0.89      0.90      0.90      5762
           3       0.89      0.87      0.88      5935

    accuracy                           0.91     23385
   macro avg       0.91      0.91      0.91     23385
weighted avg       0.91      0.91      0.91     23385

 Time Gap taken for Testing 0.5735526084899902
```

3.3.5 SGD Classifier

```
from sklearn.linear_model import SGDClassifier as gdngsAnaly_MMRY_S

Individual_vrbbl = {'penalty': ['l2', 'l1', 'elasticnet', None],
                    'alpha':[0.1, 0.01, 0.001]
                   }
Individual_vrbbl_MD = gdngsAnaly_MMRY_S(random_state=Rs)
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 12 candidates, totalling 24 fits

{'alpha': 0.001, 'penalty': None}
```

Classification Report

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      5861
           1       0.43      0.14      0.21      5850
           2       0.37      0.63      0.47      5775
           3       0.40      0.40      0.40      5899

    accuracy                           0.54     23385
   macro avg       0.55      0.54      0.52     23385
weighted avg       0.55      0.54      0.52     23385

 Time Gap taken for Validation 0.094573974609375
```

3.3.6 MLP & AdaBoost

```
Individual_vrbbl = {'voting': ['hard', 'soft']}
|
Individual_vrbbl_MD_1 = gdngsAnaly_MMRY_M(alpha= 0.001, batch_size= 50, solver= 'adam')
Individual_vrbbl_MD_2 = gdngsAnaly_MMRY_A(algorithm= 'SAMME.R', learning_rate= 0.1, n_estimators= 50)

Individual_vrbbl_MD= gdngsAnaly_MMRY_V(estimators=[('MLP', Individual_vrbbl_MD_1), ('AB', Individual_vrbbl_MD_2)])
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
{'voting': 'hard'}
```

Classification Report

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5849
           1       0.48      0.39      0.43      5839
           2       0.60      0.41      0.49      5762
           3       0.50      0.74      0.59      5935

    accuracy                           0.64     23385
   macro avg       0.64      0.63      0.63     23385
weighted avg       0.64      0.64      0.63     23385

 Time Gap taken for Testing 0.6603026390075684
```

3.3.7 AdaBoost & Gaussian NB

```
Individual_vrbbl_MD_1 = gdngsAnaly_MMRY_A(algorithm= 'SAMME.R', learning_rate= 0.1, n_estimators= 50)
Individual_vrbbl_MD_2 = gdngsAnaly_MMRY_G(var_smoothing= 1e-09)

Individual_vrbbl_MD= gdngsAnaly_MMRY_V(estimators=[('AB', Individual_vrbbl_MD_1), ('GNB', Individual_vrbbl_MD_2)])
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
{'voting': 'hard'}
```

Classification Report

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5849
           1       0.48      0.42      0.45      5839
           2       0.64      0.39      0.49      5762
           3       0.49      0.73      0.59      5935

    accuracy                           0.64     23385
   macro avg       0.65      0.64      0.63     23385
weighted avg       0.65      0.64      0.63     23385

 Time Gap taken for Testing 0.4819650650024414
```

3.3.8 Gaussian NB & Bagging

```
Individual_vrbbl_MD_1 = gdngsAnaly_MMRY_G(var_smoothing= 1e-09)
Individual_vrbbl_MD_2 = gdngsAnaly_MMRY_B(max_features= 40, n_estimators= 50)

Individual_vrbbl_MD= gdngsAnaly_MMRY_V(estimators=[('GNB', Individual_vrbbl_MD_1), ('BAG', Individual_vrbbl_MD_2)])
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
{'voting': 'hard'}
```

Classification Report

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5849
           1       0.84      0.88      0.86      5839
           2       0.85      0.88      0.87      5762
           3       0.89      0.81      0.85      5935

    accuracy                           0.89     23385
   macro avg       0.89      0.89      0.89     23385
weighted avg       0.89      0.89      0.89     23385

 Time Gap taken for Testing 0.934422492980957
```

3.3.9 Bagging & SGD

```
Individual_vrbbl_MD_1 = gdngsAnaly_MMRY_B(max_features= 40, n_estimators= 50)
Individual_vrbbl_MD_2 = gdngsAnaly_MMRY_S(alpha= 0.1, penalty= 'elasticnet')

Individual_vrbbl_MD= gdngsAnaly_MMRY_V(estimators=[('BAG', Individual_vrbbl_MD_1), ('SGD', Individual_vrbbl_MD_2)])
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
{'voting': 'hard'}
```

Classification Report

	precision	recall	f1-score	support
0	0.99	1.00	1.00	5849
1	0.67	0.90	0.76	5839
2	0.84	0.71	0.77	5762
3	0.88	0.70	0.78	5935
accuracy			0.83	23385
macro avg	0.85	0.83	0.83	23385
weighted avg	0.85	0.83	0.83	23385

Time Gap taken for Testing 0.8384368419647217

3.3.10 SGD & MLP

```
Individual_vrbbl_MD_1 = gdngsAnaly_MMRY_S(alpha= 0.1, penalty= 'elasticnet')
Individual_vrbbl_MD_2 = gdngsAnaly_MMRY_M(alpha= 0.001, batch_size= 50, solver= 'adam')

Individual_vrbbl_MD= gdngsAnaly_MMRY_V(estimators=[('SGD', Individual_vrbbl_MD_1), ('MLP', Individual_vrbbl_MD_2)])
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
{'voting': 'hard'}
```

Classification Report

	precision	recall	f1-score	support
0	0.99	0.99	0.99	5849
1	0.34	1.00	0.50	5839
2	0.77	0.02	0.03	5762
3	0.96	0.00	0.01	5935
accuracy			0.50	23385
macro avg	0.76	0.50	0.38	23385
weighted avg	0.76	0.50	0.38	23385

Time Gap taken for Testing 0.46872973442077637

3.3.11 MLP, AdaBoost & Gaussian NB

```
Individual_vrbbl_MD_1 = gdngsAnaly_MMRY_M(alpha= 0.001, batch_size= 50, solver= 'adam')
Individual_vrbbl_MD_2 = gdngsAnaly_MMRY_A(algorithm= 'SAMME.R', learning_rate= 0.1, n_estimators= 50)
Individual_vrbbl_MD_3 = gdngsAnaly_MMRY_G(var_smoothing= 1e-09)

Individual_vrbbl_MD= gdngsAnaly_MMRY_V(estimators=[('MLP', Individual_vrbbl_MD_1), ('AB', Individual_vrbbl_MD_2), ('GNB', Indivi
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
{'voting': 'hard'}
```

Classification Report

	precision	recall	f1-score	support
0	0.99	1.00	1.00	5849
1	0.39	0.22	0.28	5839
2	0.54	0.26	0.35	5762
3	0.45	0.86	0.59	5935
accuracy			0.59	23385
macro avg	0.59	0.58	0.55	23385
weighted avg	0.59	0.59	0.55	23385

Time Gap taken for Testing 0.5585181713104248

3.3.12 Gaussian NB, Bagging and SGB

```
Individual_vrbbl_MD_1 = gdngsAnaly_MMRY_G(var_smoothing= 1e-09)
Individual_vrbbl_MD_2 = gdngsAnaly_MMRY_B(max_features= 40, n_estimators= 50)
Individual_vrbbl_MD_3 = gdngsAnaly_MMRY_S(alpha= 0.1, penalty= 'elasticnet')

Individual_vrbbl_MD= gdngsAnaly_MMRY_V(estimators=[('GNB', Individual_vrbbl_MD_1), ('BAG', Individual_vrbbl_MD_2), ('SGD', Indiv
Individual_vrbbl_MD = gdngsAnaly_MMRY(Individual_vrbbl_MD, Individual_vrbbl, cv=2, verbose=1)
Individual_vrbbl_MD.fit(X_Analy_MMRY_NN.sample(7000,random_state=Rs), Y_Analy_MMRY_NN.sample(7000,random_state=Rs))

Individual_vrbbl_MD.best_params_
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
{'voting': 'hard'}
```

Classification Report

	precision	recall	f1-score	support
0	0.99	1.00	0.99	5849
1	0.64	0.21	0.31	5839
2	0.81	0.20	0.33	5762
3	0.40	0.95	0.56	5935
accuracy			0.59	23385
macro avg	0.71	0.59	0.55	23385
weighted avg	0.71	0.59	0.55	23385

Time Gap taken for Testing 0.5011446475982666

4. Dataset

In this study, the CIC MalMem_2022 dataset used is from Canadian Institute for Cybersecurity. This dataset is publicly available with synthetic collection of Memory malware (CIC_MalMem_2022, 2022).

References

Jupyter Notebook 7 Download available online ,https://jupyter.org/
CIC_MalMem_2022 Canadian Institute for Cybersecurity, 2022, https://www.unb.ca/cic/datasets/malmem-2022.html