

ML Test

THES0 Internal Document

Aug 2023

Choa, Tustin Annika S.
de Veyra, Julianne Amor B.
Escalona, Jose Miguel A.
Fortiz, Patrick Ryan P.

Table of Contents

1. INTRODUCTION	8
2. OBJECTIVES.....	8
3. FRAMEWORK	9
3.1. Reference Framework	9
3.2. Proposed Framework.....	10
4. TOOLS	11
5. DATASET.....	11
5.1. Dataset Exploration	12
5.1.1. MalbehavD-V1	12
5.1.2. Oliveira	12
5.1.3. Catak	13
5.2. File Hashes	14
5.3. Dataset Comparison	15
5.3.1. MalbehavD-V1	15
5.3.2. Oliveira	16
5.3.3. Catak	16
5.4. Dataset Cleaning/Pre-Processing	17
5.4.1. Dataset Pre-Cleaning Procedure	17
5.4.2. Pre-Cleaning Results.....	18
5.4.3. Exploring Pre-Cleaned Data	19
5.5. Cleaned Dataset.....	22
5.5.1. Cleaned Dataset Attributes.....	22
5.5.2. Cleaned Dataset Preview	23
5.6. Dataset Maximizing and Trimming.....	21
5.6.1. Expansion/Maximizing	21
5.6.2. Trimming/Minimizing	21
5.6.3. Recommended Setup (Cross-Dataset Setup Combination)	21
5.7. Label Encoding	21
5.8. Saved Dataset Files	22
5.8.1. Pre-Cleaned (containing string API calls).....	23
5.8.2. Cleaned (containing String API calls).....	23
5.8.3. Cleaned (containing LabelEncoded API calls)	24
5.8.4. Maximized (LabelEncoded Only).....	24
5.8.5. Trimmed (LabelEncoded Only).....	24
5.9. Dataset Pre-processing and Cleaning Time.....	24
6. ML TEST	25
6.1. ML Models.....	25
6.2. Common Implementation.....	26
6.3. Tuning	28
6.4. Saving the Trained Model as a File.....	31
6.5. Performance Metrics	33
6.6. Other Metrics.....	33
6.6.1. Training Time.....	33
6.6.2. Prediction Time.....	33
6.6.3. Tuning Time.....	33

6.7.	Tests and Comparisons	34
6.7.1.	Stratified K-Folds Test	34
6.7.2.	Default and Tuned Model Performance Comparison	34
6.7.3.	Dataset Performance Comparison	34
6.7.4.	Model Robustness Test	35
6.7.5.	Time Test	35
6.8.	Results	36
6.8.1.	Default vs Tuned Model Performance Comparison	36
6.8.2.	Dataset Performance Comparison	38
6.8.3.	Model Robustness Test	39
6.8.4.	Time Test	47
7.	Observations, Interpretations, and Discussion of Data	49
7.1.	Default vs Tuned Model Performance Comparison	49
7.2.	Dataset Performance Comparison	49
7.3.	Model Robustness Test	51
7.4.	Time Test	51
7.5.	Other Observations	53
8.	CONCLUSION	53
9.	RECOMMENDATIONS	54
10.	REFERENCES	55
11.	APPENDIX	56
11.1.	Classification Reports (MalbehavD-V1)	56
11.2.	Classification Reports (Oliveira)	59
11.3.	Confusion Matrices (MalbehavD-V1)	65
11.3.1.	Default	65
11.3.2.	Tuned	66
11.4.	Confusion Matrices (Oliveira)	67
11.4.1.	Default	67
11.4.2.	Tuned	68
11.5.	Decision Trees	69
11.5.1.	MalbehavD-V1 - Default (Left), Tuned (Right)	69
11.5.2.	Oliveira - Default (Left), Tuned (Right)	69
11.6.	Dataset Performance	70
11.6.1.	MalbehavD-V1: Default	70
11.6.2.	MalbehavD-V1: Tuned	70
11.6.3.	MalbehavD-V1: Improvement	71
11.6.4.	Oliveira: Default	71
11.6.5.	Oliveira: Tuned	71
11.6.6.	Oliveira: Improvement	72
11.6.7.	MalbehavD-V1 [-] vs Oliveira [+] – Defaults	72
11.6.8.	MalbehavD-V1 [-] vs Oliveira [+] – Defaults (Interpreted)	72
11.6.9.	MalbehavD-V1 [-] vs Oliveira [+] – Tuned	73
11.6.10.	MalbehavD-V1 [-] vs Oliveira [+] – Tuned (Interpreted)	73
11.7.	Time Data	73
11.8.	Robustness	74
11.9.	Tuning Time (s)	84

11.10. System Specifications	84
------------------------------------	----

List of Tables

Table 1 - Datasets	11
Table 2 - MalbehavD-V1 SHA-256	14
Table 3 - Oliveira SHA-256	14
Table 4 - Catak SHA-256	14
Table 5 - Dataset Shape & Element Size	15
Table 6 - Top 20 API calls in each dataset	19
Table 7 - API Call Coverage	20
Table 8 - Dataset Similarity	21
Table 9 - Dataset Reshaping Configurations	21
Table 10 – Dataset Attributes	22
Table 11 - Improvement Value Definition	34
Table 12 - Changes After Tuning (MalbehavD-V1)	49
Table 13 - Changes in ROC-AUC After Tuning (Oliveira)	49
Table 14 - Dataset Performance Range Values (Default)	50
Table 15 - Dataset Performance in Individual Recall Values (Default)	50
Table 16 - Dataset Performance Range Values (Tuned)	50
Table 17 - Dataset Performance in Individual Recall Values (Tuned)	51
Table 18 - Training Time Ranges (s)	52
Table 19 - Training Time Post-Tuning Change Summary	52
Table 20 - Prediction Time Ranges (ms)	52
Table 21 - Tuning Time Ranges (s)	53

List of Figures

Figure 1 - Framework as shown in [1]	9
Figure 2 - Framework as shown in [2]	9
Figure 3 - Proposed Framework	10
Figure 4 - MalbehavD-V1 GitHub Repository	12
Figure 5 - Oliveira IEEE DataPort Repository	12
Figure 6 - Catak GitHub Repository	13
Figure 7 - Files in Catak Dataset.....	13
Figure 8 - MalbehavD-V1 Dataset Preview.....	15
Figure 9 - NaN values in MalbehavD-V1	15
Figure 10 - Oliveira Dataset Preview.....	16
Figure 11 - Catak Dataset Preview	16
Figure 12 - Pre-Cleaned MalbehavD-V1	18
Figure 13 - Pre-Cleaned Oliveira.....	18
Figure 14 - Pre-Cleaned Catak.....	18
Figure 15 - Labels in Pre-Cleaned/Cleaned Datasets	20
Figure 16 - Before Label Encoding (MalbehavD-V1).....	22
Figure 17 - After Label Encoding (MalbehavD-V1)	22
Figure 18 – Preview of Cleaned MalbehavD-V1 (LabelEncoded).....	23
Figure 19 – Preview of Cleaned Oliveira (LabelEncoded).....	23
Figure 20 – Preview of Cleaned Catak (LabelEncoded)	23
Figure 21 - Data Pre-processing and Cleaning Time	24
Figure 22 - ML Training Pseudocode (without Automated Tuning)	27
Figure 23 - Automated Hyperparameter Tuning Pseudocode.....	30
Figure 24 - Saving Trained Model to File Pseudocode.....	32
Figure 25 - Training Time Pseudocode	33
Figure 26 - Prediction Time Pseudocode	33
Figure 27 - Default Config (Default vs Tuned)	36
Figure 28 - Tuned Config (Default vs Tuned)	36
Figure 29 - Malbehavd-V1 Improvement Post-Tuning	37
Figure 30 - Oliveira Improvement Post-Tuning	37
Figure 31 - MalbehavD-V1 vs Oliveira (Default)	38

Figure 32 - MalbehavD-V1 vs Oliveira (Tuned)	38
Figure 33 - Train: MalbehavD-V1 Test: Oliveira (Trimmed)	39
Figure 34 - Train: MalbehavD-V1 Test: Oliveira (Trimmed) - Improvement	39
Figure 35 - Train: Oliveira Test: MalbehavD-V1 (Trimmed)	40
Figure 36 - Train: Oliveira Test: MalbehavD-V1 (Trimmed) - Improvement	40
Figure 37 - Train: MalbehavD-V1 Test: Oliveira (Maximized)	41
Figure 38 - Train: MalbehavD-V1 Test: Oliveira (Maximized) - Improvement	41
Figure 39 - Train: Oliveira Test: MalbehavD-V1 (Maximized)	42
Figure 40 - Train: Oliveira Test: MalbehavD-V1 (Maximized) - Improvement	42
Figure 41 - Testing on Trimmed Oliveira [+] vs Testing on MalbehavD-V1 [-] (Default)	43
Figure 42 - Testing on Trimmed Oliveira [+] vs Testing on MalbehavD-V1 [-] (Tuned)	43
Figure 43 - Testing on Trimmed MalbehavD-V1 [+] vs Testing on Oliveira [-] (Default)	44
Figure 44 - Testing on Trimmed MalbehavD-V1 [+] vs Testing on Oliveira [-] (Tuned)	44
Figure 45 - Testing on Maximized Oliveira [+] vs Testing on MalbehavD-V1 [-] (Default)	45
Figure 46 - Testing on Maximized Oliveira [+] vs Testing on MalbehavD-V1 [-] (Tuned)	45
Figure 47 - Testing on Maximized MalbehavD-V1 [+] vs Testing on Oliveira [-] (Default)	46
Figure 48 - Testing on Maximized MalbehavD-V1 [+] vs Testing on Oliveira [-] (Tuned)	46
Figure 49 - Training Time (s)	47
Figure 50 - Prediction Time (ms)	47
Figure 51 - Tuning Time (s)	48

1. INTRODUCTION

This document contains an extracurricular testing of various ML models in SKLearn where those are trained on the behavior-based/dynamic malware candidate datasets for the thesis study.

2. OBJECTIVES

The objectives of the study are as follows:

1. To determine the strengths, weaknesses, and feasibility of the candidate datasets in building an ML model.
2. To determine and test primary techniques in dataset preparation by means of data pre-processing and cleaning.
3. To determine the basic processes and factors that may affect building an ML model.
4. To determine which of the candidate datasets perform best regardless of ML model.
5. To determine which of the selected models perform best on each dataset.
6. To determine if ensemble/boosted ML models are more beneficial than traditional ML models.

3. FRAMEWORK

3.1. Reference Framework

Before proceeding with the test, a framework must be devised to establish a clear procedure to follow to proceed with the test. With this, the framework/flowchart presented in [1] and [2] will be used as the reference framework as it offers the most elaborate and relevant procedures to the study.

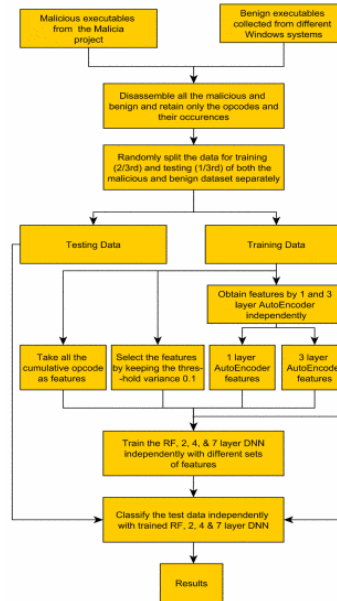


Figure 1 - Framework as shown in [1]

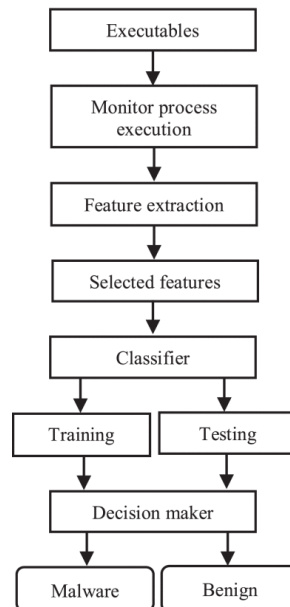


Figure 2 - Framework as shown in [2]

3.2. Proposed Framework

The proposed framework consists of three major segments which are the dataset, ML training, and ML detector.

The dataset component of the framework consists of the first three rows in the framework which involve the candidate datasets, the dataset preparation (i.e., dataset pre-processing & cleaning), and finally, the division of each of the datasets for training and testing, as well as the reshaping of the datasets for purposes of robustness testing. This part will also include the examination whether the dataset is useful for use in building an ML model for malware detection.

The ML training component of the framework consists of the succeeding two rows which include the training and testing of the ML models as well as the hyperparameter tuning should the test results found to be not optimal enough.

Lastly, the ML detector component of the framework consists of the last two rows which show the final product, a malware detector, which when given an input will return a malicious or non-malicious/benign classification.

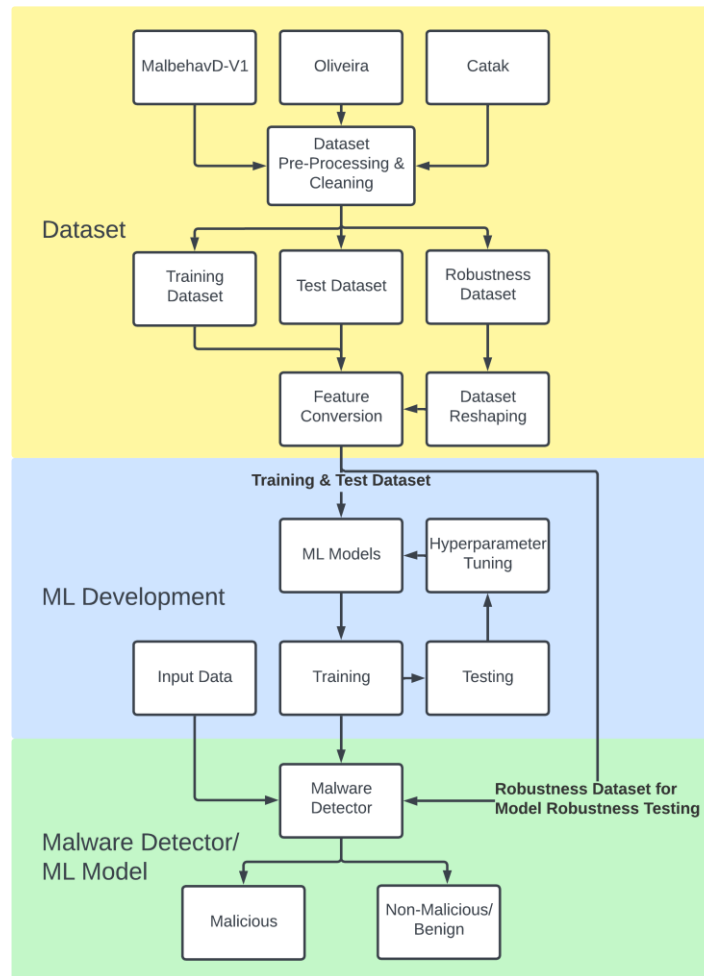


Figure 3 - Proposed Framework

4. TOOLS

The software application and tools that will be used in this test will be the following:

1. Python 3
2. Anaconda
3. Jupyter
4. Python 3 Libraries: Scikit-Learn, Pandas, Numpy

5. DATASET

Before anything can proceed, the dataset (i.e., one of the key ingredients in building ML models) must be pre-processed and cleaned first to ensure its suitability for use in creating an ML model by means of training/fitting.

The test will explore and clean the three candidate datasets selected in the Thesis document, which are the following:

Table 1 - Datasets

Dataset Title	Dataset Alt. Title	Reference
MalbehavD-V1	MalbehavD-V1	[3]
Malware Analysis Datasets: API Call Sequences	Oliveira	[4]
Windows Malware Dataset with PE API Calls	Catak	[5]

5.1. Dataset Exploration

5.1.1. MalbehavD-V1

MalbehavD-V1 is found in this [link](#) which is a repository containing a lone CSV file which is the dataset itself.

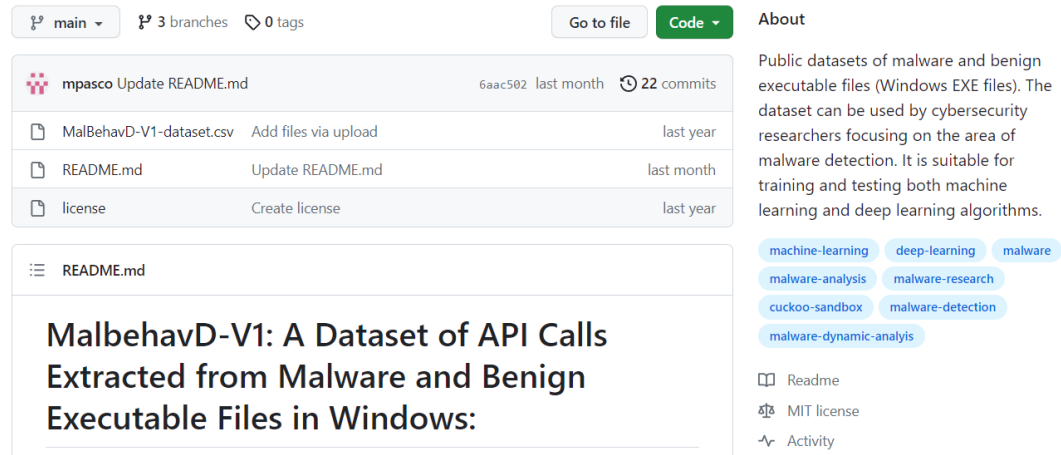


Figure 4 - MalbehavD-V1 GitHub Repository

5.1.2. Oliveira

Oliveira/‘Malware Analysis Datasets: API Call Sequences’ is found in this [link](#) which is a file (possibly downloaded as a ZIP) containing a lone CSV file which is the dataset itself.

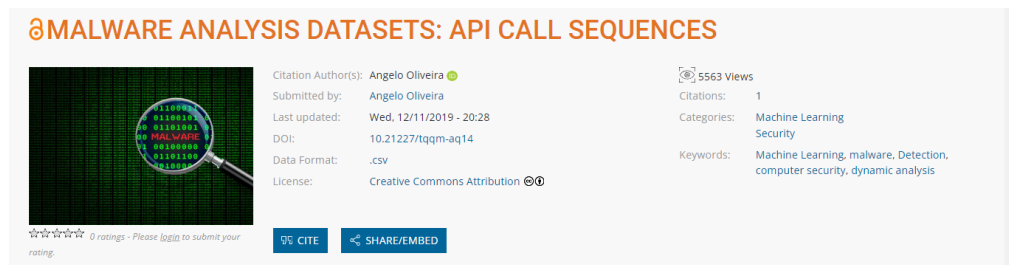
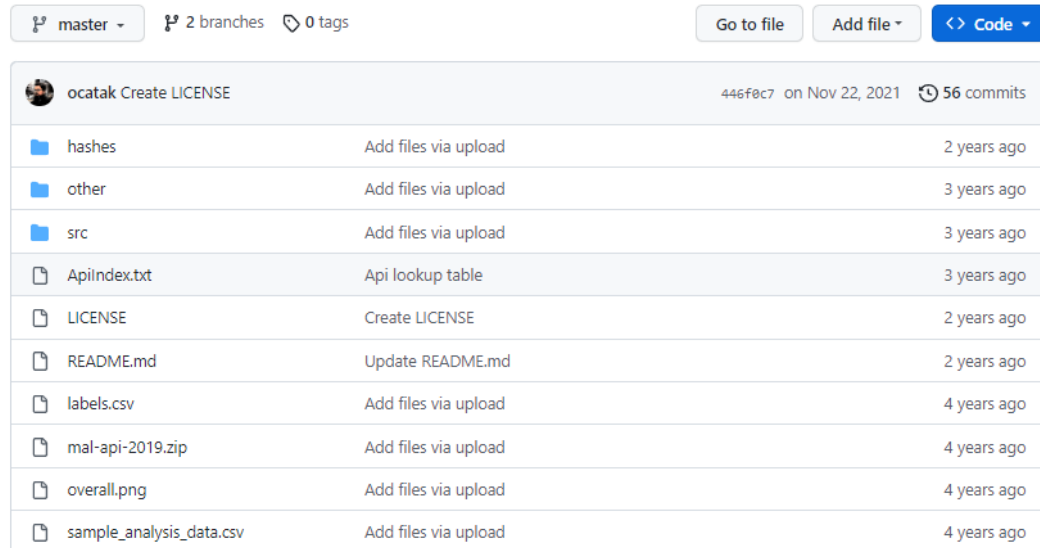


Figure 5 - Oliveira IEEE DataPort Repository

5.1.3. Catak

Catak/'Windows Malware Dataset with PE API Calls' is found in this [link](#) which is a repository containing various files.

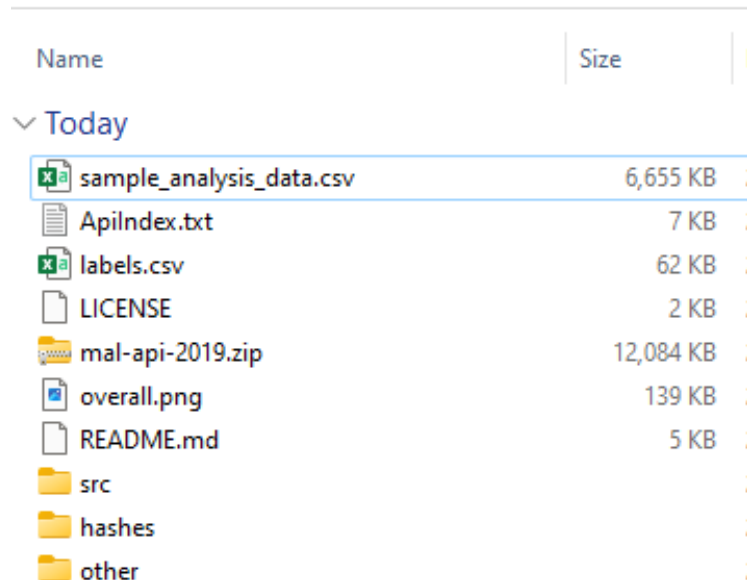


The screenshot shows the GitHub repository page for 'ocatak Create LICENSE'. The repository has 2 branches and 0 tags. The file list includes:

File Name	Description	Commit Date
hashes	Add files via upload	2 years ago
other	Add files via upload	3 years ago
src	Add files via upload	3 years ago
ApiIndex.txt	Api lookup table	3 years ago
LICENSE	Create LICENSE	2 years ago
README.md	Update README.md	2 years ago
labels.csv	Add files via upload	4 years ago
mal-api-2019.zip	Add files via upload	4 years ago
overall.png	Add files via upload	4 years ago
sample_analysis_data.csv	Add files via upload	4 years ago

Figure 6 - Catak GitHub Repository

Among these files, are the zipped file 'mal-api-2019.zip' which when extracted contains a 2.01GB file called 'all_analysis_data.csv' where each row is linked to the 'labels.csv'. All other subdirectories contain auxiliary files which may be useful for other purposes, but not so for the purposes of the study at hand.



The screenshot shows a file explorer view of the Catak dataset. The files and their sizes are:

Name	Size
sample_analysis_data.csv	6,655 KB
ApiIndex.txt	7 KB
labels.csv	62 KB
LICENSE	2 KB
mal-api-2019.zip	12,084 KB
overall.png	139 KB
README.md	5 KB
src	
hashes	
other	

Figure 7 - Files in Catak Dataset

5.2. File Hashes

Note that the hashes shown in the tables below may be outdated at some point as the authors of these datasets may have updated their datasets. Nonetheless, these are the hashes that were determined at the time when the datasets were downloaded and saved on the team's repository.

5.2.1.1. MalbehavD-V1

Table 2 - MalbehavD-V1 SHA-256

File Name	Value
MalBehavD-V1-dataset.csv	1e39c43a014e9b4ee56766ad6bb367ffe8ec4316eb0be75eb182c3b2d15f6364

5.2.1.2. Oliveira

Table 3 - Oliveira SHA-256

File Name	SHA-256
dynamic_api_call_sequence_per_malware_100_0_306.csv	11005ff6f5007bfee7d60bd0dc2e787f4e77b46f3b0a3d09424421c5339a8406

5.2.1.3. Catak

Table 4 - Catak SHA-256

File	SHA-256
all_analysis_data.csv	9962709e453af6344bed2fb514aa183adabc129582b97739b8f849a9f86d445f
labels.csv	908ac54f82f2daae5489603b4f31eb82e8a04424cfe6370d589881956578d739

5.2.1.4. Dataset Shape and Element Size

Table 5 - Dataset Shape & Element Size

Dataset	Shape (Row x Column)	Element Size/Count
MalbehavD-V1	2570 x 177	454,890
Oliveira	43876 x 102	4,475,352
Catak	7106 x 1	7,106

5.3. Dataset Comparison

To easily compare the data, Panda's 'head()' method was called on the DataFrame of each of the dataset.

5.3.1. MalbehavD-V1

Figure 8 - MalbehavD-V1 Dataset Preview

The MalbehavD-V1 dataset contains 177 columns, 175 of which containing API calls and the remaining two being non-API call related (i.e., labels).

7	...	Unnamed: 167	Unnamed: 168	Unnamed: 169	Unnamed: 170	Unnamed: 171	Unnamed: 172	Unnamed: 173	Unnamed: 174	Unnamed: 175	Unnamed: 176
GetSystemWindowsDirectoryW	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CoUninitialize	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GetFileAttributesW	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NtClose	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NtAllocateVirtualMemory	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 9 - NaN values in MalbehavD-V1

Notably, the dataset also contains NaN values which indicate empty cells in the dataset, implying that not all samples in the dataset have the same count of API calls.

The setup/look of MalbehavD-V1 dataset will be the setup for the remaining datasets during the pre-processing stage where it is characterized by the following:

1. The first two columns are set as label related and the remaining columns are feature/API calls related.
2. The naming mechanism for the string form of the API calls is camel cased.
3. The naming mechanism for the feature labels are in consecutive numeric form.

5.3.2. Oliveira

	hash	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	...	t_91	t_92	t_93	t_94	t_95	t_96	t_97	t_98	t_99	malware
0	071e8c3f8922e186e57548cd4c703a5d	112	274	158	215	274	158	215	298	76	...	71	297	135	171	215	35	208	56	71	1
1	33f8e6d08a6ae939f25a8e0d63d523	82	208	187	208	172	117	172	117	172	...	81	240	117	71	297	135	171	215	35	1
2	b68abd064e975e1c6d525e748663076	16	110	240	117	240	117	240	117	240	...	65	112	123	65	112	123	65	113	112	1
3	72049be7bd30ea61297ea624ae198067	82	208	187	208	172	117	172	117	172	...	208	302	208	302	187	208	302	228	302	1
4	c9b3700a77facf29172f32d6b6c77f48	82	240	117	240	117	240	117	240	117	...	209	260	40	209	260	141	260	141	260	1

5 rows × 102 columns

Figure 10 - Oliveira Dataset Preview

The Oliveira dataset contains 102 columns, 100 of which containing API calls and the remaining two being non-API call related (i.e., labels). The labels (hash and malware are) found on both ends of the dataset respectively which must be pre-processed first before proceeding to its use. The hash in the dataset was determined to be MD5 hashes as per the results shown in CyberChef.

Notably, the API calls used in the dataset uses a number system instead of the API call in string form. An advantage of this approach is that the dataset file uses less space and that the file is ready to use as it is already LabelEncoded (i.e., string features are converted into numeric features).

This method can be used later as part of data pre-processing as most ML models require integer-based features as supposed to string, even for categorical data.

The following are the numerous issues that needs to be addressed during pre-processing:

1. Dataset only contains numerical representations of API calls.
2. Headers of the API calls have a 't_x' prefix instead of only consecutive numbers.
3. Position of the 'malware' column is on the last column.

5.3.3. Catak

[illegible]

Figure 11 - Catak Dataset Preview

The Oliveira dataset contains a single 1 column which is due to the wrong delimiting value of a space character (‘ ’) instead of a comma (‘,’) as one would using a CSV file. It was also noticeable that the dataset also contains no headers and that the per-row contents contain repetitive API calls which is the primary reason for the 2.01GB file size of the dataset. Additionally, given its contents, it could be concluded that the dataset is more suited for malware classification rather than detection.

The following are the numerous issues that needs to be addressed during pre-processing:

1. Incorrect delimiter for a CSV file.
2. Lower-cased features/API string.
3. Repeated API calls per row/sample.
4. No label columns of any form; at least ‘malware_types’ from labels.csv in place of sha256/hash in previous datasets.
5. No ‘malware’ column (i.e., 1s).

5.4. Dataset Cleaning/Pre-Processing

5.4.1. Dataset Pre-Cleaning Procedure

The dataset cleaning procedure for each dataset will follow the steps mentioned below. MalbehavD-V1 will be the standard/reference ‘cleaned’ dataset where other datasets must follow suit.

1. Oliveira
 - a. Making a version that contains string API calls.
 - b. Removing ‘t_x’ prefix in API call column headers (i.e., only consecutive numbers).
 - c. Moving position of ‘malware’ column to the second column.
2. Catak
 - a. Replacing spaces to commas as delimiters for each API call.
 - b. Changing features/API string casing from lower-case to camel case.
 - c. Removing repeated API calls per row/sample (i.e., only first instance will be retained).
 - d. Adding a first column for ‘malware_types’ where the per-row contents will be from the contents of ‘labels.csv’
 - e. Adding a second column for ‘malware’ which will contain 1s.
3. All Datasets
 - a. The NaN values (as seen in Pandas) will be replaced/represente with a space character (‘ ’) instead as LabelEncoding does not support NaN values.

5.4.2. Pre-Cleaning Results

All datasets are pre-cleaned using Pandas. The code can be seen in the Jupyter notebook file. The preview is as shown below.

	sha256	malware	0	1	2	3	4	5	6
0	5c18291c481a192ed5033084dab2d8a117f63736359218...	0	LdrUnloadDll	CoUninitialize	NtQueryKey	NtDuplicateObject	GetShortPathNameW	GetSystemInfo	IsDebuggerPresent
1	4683fa3da550fb594c5513c4cbb34f64df85f27fd1c...	0	NtOpenMutant	GetForegroundWindow	NtQueryKey	DrawTextExW	NtSetInformationFile	RegQueryValueExA	LdrGetProcedureAddress
2	9a0aea1c7290031d7c3429d0e9211f07282cc6eab854ee...	0	GetForegroundWindow	DrawTextExW	GetSystemInfo	IsDebuggerPresent	GetSystemWindowsDirectoryW	NtQueryValueKey	RegCloseKey
3	e0f3e4d5f50af09c31e51dd9941c5a52d57c7c524f5d11...	0	NtQueryValueKey	LdrUnloadDll	GlobalMemoryStatus	WriteConsoleA	NtOpenKey	LdrGetProcedureAddress	NtTerminateProcess
4	ec2b6d29992f13e74015f0b129150b4fae15c5934e407...	0	LdrUnloadDll	GetSystemTimeAsFileTime	NtOpenKey	WSAStartup	SetUnhandledExceptionFilter	NtTerminateProcess	NtClose

5 rows × 10 columns

Figure 12 - Pre-Cleaned MalbehavD-V1

	hash	malware	0	1	2	3	4	5	6	7	...
0	071e6c3f8922e186e57548cd4c703a5d	1	HttpSendRequestA	WSAAccept	NtCreateSection	Process32NextW	WSAAccept	NtCreateSection	Process32NextW	recvfrom	Internet
1	33f8e6d0a6aae939f25a8e0a63d3d523	1	GetFileVersionInfoExW	OleInitialize	NtQueryKey	OleInitialize	NtLoadKey	InternetConnectA	NtLoadKey	InternetConnectA	FindR
2	b68abd064e975e1c6d5f25e748663076	1	CreateActCvW	HttpOpenRequestW	RemoveDirectoryW	InternetConnectA	RemoveDirectoryW	InternetConnectA	RemoveDirectoryW	InternetConnectA	InternetGetConnected
3	72049be7bd30ae61297ea632ae198067	1	GetFileVersionInfoExW	OleInitialize	NtQueryKey	OleInitialize	NtLoadKey	InternetConnectA	NtLoadKey	InternetConnectA	Process
4	c9b3700a77fcd2912f252df6bc7748	1	GetFileVersionInfoExW	RemoveDirectoryW	InternetConnectA	RemoveDirectoryW	InternetConnectA	RemoveDirectoryW	InternetConnectA	RemoveDirectoryW	CryptUnprot

5 rows × 12 columns

Figure 13 - Pre-Cleaned Oliveira

	malware_type	malware	0	1	2	3	4	5	6	7	...	156	157	158	15
0	Trojan	1	LdrLoadDll	LdrGetProcedureAddress	RegOpenKeyExA	NtOpenKey	NtOpenKeyEx	NtQueryValueKey	NtClose	NtQueryAttributesFile	...	NaN	NaN	NaN	Na
1	Trojan	1	GetSystemTimeAsFileTime	NtAllocateVirtualMemory	NtFreeVirtualMemory	LdrGetDllHandle	LdrGetProcedureAddress	SetUnhandledExceptionFilter	NtCreateMutant	NtClose	...	NaN	NaN	NaN	Na
2	Backdoor	1	LdrGetDllHandle	LdrGetProcedureAddress	GetSystemDirectoryA	CopyFileA	RegOpenKeyExA	RegSetValueExA	RegCloseKey	RegCreateKeyExA	...	NaN	NaN	NaN	Na
3	Backdoor	1	LdrLoadDll	LdrGetProcedureAddress	RegOpenKeyExA	NtOpenKey	NtOpenKeyEx	NtQueryValueKey	NtClose	NtQueryAttributesFile	...	NaN	NaN	NaN	Na
4	Trojan	1	LdrLoadDll	LdrGetProcedureAddress	WSAStartup	NtCreateMutant	RegOpenKeyExA	RegDeleteKeyA	RegCloseKey	CopyFileA	...	NaN	NaN	NaN	Na

5 rows × 16 columns

Figure 14 - Pre-Cleaned Catak

5.4.3. Exploring Pre-Cleaned Data

5.4.3.1. Features/API Calls

All API calls found in all the three datasets are saved as 'CombinedAPIs.csv' file. In total there are 308 unique API calls found across the three datasets. The quantity of each API call per dataset are found in their own files named as '<Dataset>_Freq.txt'. The first top 20 API calls (except for the NaN fill-in ' ') for each dataset is shown below.

Table 6 - Top 20 API calls in each dataset

Oliveira		MalbehavD-V1		Quantity	
API Call	Size	API Call	Size	API Call	Size
InternetConnectA	732701	NtClose	2524	NtOpenKeyEx	7686
RemoveDirectoryW	412278	NtQueryValueKey	2447	NtClose	7030
NtLoadKey	314298	NtOpenKey	2446	NtOpenKey	6656
SetStdHandle	260429	LdrGetProcedureAddress	2324	LdrGetProcedureAddress	6599
Process32NextW	222786	NtCreateFile	2181	NtQueryValueKey	6520
GetFileType	191735	NtAllocateVirtualMemory	2173	NtAllocateVirtualMemory	6449
GetAdaptersAddresses	188556	LdrUnloadDll	2139	LdrLoadDll	6408
FindResourceW	186112	RegCloseKey	1936	LdrGetDllHandle	6201
LookupAccountSidW	184347	LdrGetDllHandle	1922	NtCreateFile	5969
OleInitialize	176499	LdrLoadDll	1894	NtFreeVirtualMemory	5545
WSAAccept	102230	NtFreeVirtualMemory	1891	RegCloseKey	5515
NtCreateSection	101969	GetSystemTimeAsFileTime	1796	RegOpenKeyExA	5395
CryptHashData	83484	NtReadFile	1596	LdrUnloadDll	5287
GetFileVersionInfoExW	72684	NtTerminateProcess	1544	NtMapViewOfSection	4373
anomaly	53122	NtMapViewOfSection	1539	RegOpenKeyExW	4239
recv	39452	NtCreateSection	1473	NtTerminateProcess	4160
NtQueryKey	39237	NtWriteFile	1455	RegQueryValueExA	4051
NtSetValueKey	37756	RegOpenKeyExW	1433	NtCreateSection	3946
HttpSendRequestA	33689	RegQueryValueExW	1339	RegQueryValueExW	3883
RegEnumValueA	32925	GetFileAttributesW	1338	GetSystemMetrics	3688

5.4.3.2. Labels/ 'Malware or Not'

The datasets were determined to have labels 1 (malicious) and 0 (non-malicious/benign). Notably, and as expected, the Catak dataset will only contain a malware label of 1 as all its samples are malicious.

```
y = malbehavd['malware'].to_numpy()
labels = malbehavd['malware'].unique()
print("MalbehavD - No. of unique labels: ", labels.size)
print(labels)
```

```
MalbehavD - No. of unique labels: 2
[0 1]
```

```
y = oliveira['malware'].to_numpy()
labels = oliveira['malware'].unique()
print("Oliviera - No. of unique labels: ", labels.size)
print(labels)
```

```
Oliviera - No. of unique labels: 2
[1 0]
```

```
y = catak['malware'].to_numpy()
labels = catak['malware'].unique()
print("Catak - No. of unique labels: ", labels.size)
print(labels)
```

```
Catak - No. of unique labels: 1
[1]
```

Figure 15 - Labels in Pre-Cleaned/Cleaned Datasets

5.4.3.3. API Call Matching (API Call Coverage)

API Call matching aims to determine how close each dataset is to one another, and which dataset covers most of the API calls found in all the datasets. The file 'CombinedAPIs.csv' will be used for this as the reference list of API calls for which comparisons are made. The summary is as found in Table 7 - API Call Coverage:

Table 7 - API Call Coverage

Dataset	API Match/Coverage
MalbehavD-V1	94.48%
Oliveira	85.39%
Catak	92.53%

With this data, it can be concluded MalbehavD-V1 has the highest API match rate/coverage with Oliveira being the lowest. While it can be assumed the results show MalbehavD-V1 to have more 'quality', it may not be so necessary when it is used for ML purposes where a dataset with acceptable quality yet substantial quantity (i.e., Oliveira) can outperform a dataset with superior quality yet lower quantity (i.e., MalbehavD-V1).

5.4.3.4. Similarity amongst Datasets

The point of reference for this similarity checking will be from MalbehavD-V1 where the results show that the nearest dataset to MalbehavD-V1 is the Catak dataset instead of Oliveira's. However, this result will be discarded as the Catak dataset is not suitable for the purposes of malware detection.

Table 8 - Dataset Similarity

Dataset Pair	Match/Similarity Rate
MalbehavD-V1 to Oliveira	85.5670%
MalbehavD-V1 to Catak	93.4708%

5.5. Dataset Maximizing and Trimming

During development, an error was encountered where the no. of features used in the fitted dataset must also be the same as the test/input dataset. Hence, there is another dataset pre-processing technique that must be done to allow cross-dataset training and testing. The datasets will then be LabelEncoded once re-shaped.

5.5.1. Expansion/Maximizing

The datasets will be resized to the biggest dataset in terms of column size which in this case will be MalbehavD-V1's 175 columns (including the 2 labels). Extending the datasets will result to adding NaN values, represented by a space character.

5.5.2. Trimming/Minimizing

The datasets will be resized to the smallest dataset in terms of column size which in this case will be Oliveira's 102 columns (including the 2 labels). Trimming the datasets will result to the excess API calls to be trimmed/removed.

5.5.3. Recommended Setup (Cross-Dataset Setup Combination)

Table 9 - Dataset Reshaping Configurations

Reshaping	Training Dataset	Testing Dataset
Maximize	MalbehavD_LabelEncoded.csv	Oliveira_LabelEnc_Max.csv
Maximize	Oliveira_LabelEnc_Max.csv	MalbehavD_LabelEncoded.csv
Minimize	MalbehavD_LabelEnc_Trim.csv	Oliveira_LabelEncoded.csv
Minimize	Oliveira_LabelEncoded.csv	MalbehavD_LabelEnc_Trim.csv

5.6. Label Encoding

During test development, an error was encountered where fitting (and by extension, predicting) on ML models can only be done if the feature data is numeric, hence there must be a way to convert the string API calls to numeric form such that a number equates to a specific API call. The same concept can also be found in [4].

The list of APIs to be used as reference will come from 'CombinedAPIs.csv' where it will be fed to the SciKit Learn's LabelEncoder pre-processing function. An example of the results is as shown below. Notice the API call 'NtQueryKey' being labelled numerically as 65.

	0	1	2	3	4	5
0	LdrUnloadDll	CoUninitialize	NtQueryKey	NtDuplicateObject	GetShortPathNameW	GetSystemInfo
1	NtOpenMutant	GetForegroundWindow	NtQueryKey	DrawTextExW	NtSetInformationFile	RegQueryValueExA
2	GetForegroundWindow	DrawTextExW	GetSystemInfo	IsDebuggerPresent	GetSystemWindowsDirectoryW	NtQueryValueKey
3	NtQueryValueKey	LdrUnloadDll	GlobalMemoryStatus	WriteConsoleA	NtOpenKey	LdrGetProcedureAddress
4	LdrUnloadDll	GetSystemTimeAsFileTime	NtOpenKey	WSAStartup	SetUnhandledExceptionFilter	NtTerminateProcess

Figure 16 - Before Label Encoding (MalbehavD-V1)

	0	1	2	3	4	5
0	26	2	65	52	29	31
1	35	19	65	9	78	92
2	14	7	26	35	34	77
3	39	31	35	95	66	43
4	26	23	60	94	104	83

Figure 17 - After Label Encoding (MalbehavD-V1)

5.7. Cleaned Dataset

The resulting cleaned dataset will be saved as a file with a filename format of '<Dataset>_Str.csv' & '<Dataset>_LabelEncoded.csv' for the string API call and LabelEncoded API call respectively.

5.7.1. Cleaned Dataset Attributes

The resulting cleaned dataset attributes are as follows:

Table 10 – Dataset Attributes

Dataset (Filename)	Shape (Row x Col)	Element Size/Count	File Size – String	File Size - LabelEncoded
MalbehavD-V1	2570 x 177	454,890	2.55 MB	1.13 MB
Oliveira	43876 x 102	4,475,352	65.8 MB	15.4 MB
Catak	7106 x 168	1,193,808	6.73 MB	2.73 MB

5.7.2. Cleaned Dataset Preview

Cleaned datasets as seen in ‘ML Test’ Jupyter Notebook.

2.1.1 Preview MalbehavD

```
malbehavd.head()
```

	sha256	malware	0	1	2	3	4	5	6	7	...	165	166	167	168	169	170	171	172	173	174
0	5c18291c461a192ed5003084dab2d8a117fd3736359218...		0	26	2	66	53	30	32	45	38	...	0	0	0	0	0	0	0	0	0
1	4683faf3da550ffb594cf5513c4cbb34f64df85f27fd1c...		0	35	19	66	10	79	93	47	2	...	0	0	0	0	0	0	0	0	0
2	9a0aea1c7290031d7c3429d0e921f107282cc6eab854ee...		0	14	7	27	36	35	78	95	23	...	0	0	0	0	0	0	0	0	0
3	e0f3e4d5f50afd9c31e51dd9941c5a52d57c7c524f5d11...		0	39	31	36	96	67	44	86	55	...	0	0	0	0	0	0	0	0	0
4	ec2b6d29992f13e74015ff0b129150b4afae15c593e4b7...		0	26	23	61	95	105	84	57	54	...	0	0	0	0	0	0	0	0	0

5 rows × 177 columns

Figure 18 – Preview of Cleaned MalbehavD-V1 (LabelEncoded)

2.1.2. Preview Oliveira

```
oliveira.head()
```

	hash	malware	0	1	2	3	4	5	6	7	...	90	91	92	93	94	95	96	97	98	99	
0	071e8c3f8922e186e57548cd4c703a5d		1	36	74	54	68	108	71	91	123	...	73	41	177	83	103	126	23	128	34	40
1	33f8e6d08a6aae939f25a8e0d63dd523		1	22	54	62	65	71	54	75	52	...	35	49	143	72	42	174	85	110	135	23
2	b68abd064e975e1c6d5f25e748663076		1	3	32	75	43	92	54	97	52	...	77	39	72	77	39	68	79	41	73	69
3	72049be7bd30ea61297ea624ae198067		1	22	54	62	65	71	54	75	52	...	129	122	180	126	182	109	128	185	141	177
4	c9b3700a77facf29172f32df6bc77f48		1	22	63	42	74	51	98	56	96	...	26	123	153	26	125	152	88	159	90	152

5 rows × 102 columns

Figure 19 – Preview of Cleaned Oliveira (LabelEncoded)

2.1.3. Preview Catak

```
catak.head()
```

	malware_type	malware	0	1	2	3	4	5	6	7	...	156	157	158	159	160	161	162	163	164	165
0	Trojan		1	27	32	87	66	71	84	64	75	...	0	0	0	0	0	0	0	0	0
1	Trojan		1	19	39	54	41	45	118	67	54	...	0	0	0	0	0	0	0	0	0
2	Backdoor		1	25	32	22	2	99	109	106	96	...	0	0	0	0	0	0	0	0	0
3	Backdoor		1	27	32	87	66	71	84	64	75	...	0	0	0	0	0	0	0	0	0
4	Trojan		1	27	32	106	55	99	101	106	4	...	0	0	0	0	0	0	0	0	0

5 rows × 168 columns

Figure 20 – Preview of Cleaned Catak (LabelEncoded)

5.8. Saved Dataset Files

5.8.1. Pre-Cleaned (containing string API calls)

The saved pre-cleaned dataset files can be found in the ‘Cache’ subfolder of the ‘Notebook’ folder in the Thesis GitHub repository.

5.8.2. Cleaned (containing String API calls)

The saved cleaned dataset files can be found in the ‘Notebook’ folder in the Thesis GitHub repository with a filename suffix of ‘_Str’.

5.8.3. Cleaned (containing LabelEncoded API calls)

The saved cleaned dataset files can be found in the 'Notebook' folder in the Thesis GitHub repository with a filename suffix of '_LabelEncoded'.

5.8.4. Maximized (LabelEncoded Only)

The saved cleaned dataset files can be found in the 'Notebook' folder in the Thesis GitHub repository with a filename suffix of '_LabelEnc_Max'.

5.8.5. Trimmed (LabelEncoded Only)

The saved cleaned dataset files can be found in the 'Notebook' folder in the Thesis GitHub repository with a filename suffix of '_LabelEnc_Trim'.

5.9. Dataset Pre-processing and Cleaning Time

The time it will take for the dataset pre-processing and cleaning will be dependent on the single core performance of the CPU, RAM read/write speed, and storage read/write speed. The data pre-processing code, as designed, is not multi-threaded which does not benefit on multi-core processing.

7. Time Taken

```
: dur_s = time.time()-start_time
dur_min = dur_s/60
print(f"{dur_s}s")
print(f"{dur_min:.2f}mins")

314.7926678657532s
5.25mins
```

Figure 21 - Data Pre-processing and Cleaning Time

6. ML TEST

6.1. ML Models

The models that were used in this test are as follows:

1. Traditional
 - a. K-Nearest Neighbors (KNN)
 - b. Logistic Regression (LR)
 - c. Decision Tree (DT/DTC)
 - d. Support Vector Machine (SVM)
 - e. Random Forest (RF)
 - f. Gaussian Naive Bayes (GNB)
2. Boosted
 - a. AdaBoost
 - b. Gradient Tree Boosting (GBT)
 - c. Histogram-based Gradient Boosting Classification Tree (HGBT)
3. Neural Network Based
 - a. Multi-layer Perceptron (MLP)

The models were selected as most of these were found in most studies involving behavior-based malware detection. Some of the models such as AdaBoost, MLP, GBT, and HGBT are, as far as the Thesis' RRL is concerned, not yet explored in the studies made for behavior-based approach.

The models such as AdaBoost and HGBT are the most nearest ML models to XGBoost and LightGBM as both are boosted models, with HGBT being designed based on LightGBM itself [6]. Meanwhile, MLP is a neural network implementation of ML [6].

6.2. Common Implementation

The common process of the ML model is as shown in the pseudocode below where it uses RF as an example ML model to train/fit and test/evaluate. The pseudocode assumes that the dataset is already pre-processed and cleaned.

```
#Import Data Processing Libraries
import numpy as np
import pandas as pd

#Import Visualizers
import seaborn as sns
import matplotlib.pyplot as plt

#Import Model Selection (used for dividing dataset to training and test data)
import sklearn.model_selection

#Import Stratified K-Folds for K-Folds Testing
from sklearn.model_selection import StratifiedKFold

#Import Metrics
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.metrics import roc_auc_score, classification_report, ConfusionMatrixDisplay

#Import ML Model
from sklearn.ensemble import RandomForestClassifier as RF

#Import Dataset
df = pd.read_csv('<Dataset>.csv', low_memory=False)

#Collect Feature Label (i.e., Headers)
df_feats = []
for i in range(df.shape[1]-2):
    df.append(str(i))

'''
Usual variable naming convention in ML pseudocode.
X = Features, y = labels
'''

#Parsing features and labels
X, y = df[df_feats], df['malware'].to_numpy()

#Divide Training and Test Data (80:20)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=.20, random_state=1)

#Load, Train/Fit Model, and Test/Evaluate RF Model
rf = RF(random_state=1, n_jobs=-1)
rf.fit(X_train, y_train)
```

```

y_pred = rf.predict(X_test)

#Print Classification Report
rf_cr = classification_report(y_test, y_pred, digits=4)
print(rf_cr)
#Show Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix_df = pd.DataFrame(conf_matrix)
rf_conf_matrix = classification_report(y_test,y_pred, digits=4)
plt.figure(figsize=(4,4))
sns.set(font_scale=1.3)
sns.heatmap(conf_matrix_df, cbar=False, annot=True, fmt=".0f")
plt.title("RF Confusion Matrix")
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.show()

#Stratified K-Folds Test
splits = 10 #Range: 5-10
axis = 0
kf = StratifiedKFold(n_splits=splits, shuffle=True, random_state=1)
for i, (train_index, test_index) in enumerate(kf.split(X,y)):
    #Divide train and test set
    training_set = np.take(X, train_index, axis)
    training_set_labels = np.take(y, train_index, axis)
    test_set = np.take(X, test_index, axis)
    test_set_labels = np.take(y, test_index, axis)
    #Fit/Train
    rf.fit(training_set,training_set_labels)
    #Predict/Evaluate
    y_pred = rf.predict(test_set)
    #Print Individual Metrics
    print("Fold:", i)
    print("Accuracy:", round(accuracy_score(test_set_labels, y_pred),4))
    print("F1-Score Weighted:", round(f1_score(test_set_labels, y_pred, average='weighted'),4))
    print("Precision:", round(precision_score(test_set_labels, y_pred,zero_division=0),4))
    print("Recall:", round(recall_score(test_set_labels, y_pred),4))
    print("ROC_AUC:", round(roc_auc_score(test_set_labels, y_pred),4))
    print("")

```

Figure 22 - ML Training Pseudocode (without Automated Tuning)

Note that the ‘random_state’ hyperparameter is set with a value (1) to make the tests repeatable to be suitable during fine tuning (i.e., when comparing changes for each ML model) and comparing models from one another (i.e., for K-Folds data distribution consistency). This ‘random_state’ variable is like the seed value for number generators found in different programming languages and even in the video game Minecraft which enables repeatable generation.

6.3. Tuning

Tuning is an important process in ML training to ensure that the ML model is as high-performance as possible. This is done by repetitively tweaking and testing the hyperparameters of the ML models that are used. The links to the documentations of each of the models used in this document is listed below.

- | | |
|---|--|
| 1. K-Nearest Neighbors (KNN) | 7. AdaBoost |
| 2. Logistic Regression (LR) | 8. Multi-layer Perceptron (MLP) |
| 3. Decision Tree (DT/DTC) | 9. Gradient Tree Boosting (GBT) |
| 4. Support Vector Machine (SVM) | 10. Histogram-based Gradient Boosting Classification Tree (HGBT) |
| 5. Random Forest (RF) | |
| 6. Gaussian Naive Bayes (GNB) | |

There are ways to conduct [automated and iterative](#) hyper-parameter tuning. Due to limited time and computing resources, the hyperparameter tuning setup will be using ‘RandomizedSearchCV’ instead of the preferable ‘GridSearchCV’ with a ‘cv’ value of only 3 instead of at least 5, and the hyperparameter setup for each model is also not that broad which can affect the quality of the ‘best’ tuning. Nonetheless, it will still provide a near optimal hyperparameter setup for each ML model which can be useful enough for comparison purposes.

The tuning process is done twice, hence the two ‘Tuned’ notebooks, where the dataset used for training are MalbehavD-V1 and Oliveira separately. This will determine which tuning is best for each dataset and if such tuning also applies across datasets. The tuning that will be used for robustness testing (i.e., cross-dataset testing) will follow the origin dataset (e.g., if training dataset is based on A, then the tuning used is from dataset A).

Figure 13 shows the sample pseudocode for automated hyperparameter tuning using ‘RandomSearchCV’.

```

import time, datetime
import numpy as np
import pandas as pd
import seaborn as sns
# Plotting
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import sklearn.model_selection as model_selection
# Performance Metrics
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
from sklearn.metrics import classification_report, ConfusionMatrixDisplay, mean_squared_error as MSE
from sklearn.model_selection import cross_val_score
# ML Models
from sklearn.neighbors import KNeighborsClassifier
# Automated Fine-Tuning
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from scipy.stats import expon

# Import Dataset
malbehavd = pd.read_csv('MalbehavD_LabelEncoded.csv', low_memory=False)

# Collect feature labels
malbehavd_feats = []
for i in range(malbehavd.shape[1]-2):
    malbehavd_feats.append(str(i))

# Set X (features) & y (labels)
X = malbehavd[malbehavd_feats]
y = malbehavd['malware'].to_numpy()

# Split samples to train and test
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=.20, random_state=1)

# KNN Config Parameters (in a dict format):
# More options the better but can be longer to complete
knn_grid = [
    {
        'n_neighbors':[1,2,4,8,10,20],
        'weights':['uniform','distance'],
        #'algorithm':['auto'],
        'leaf_size':[2,4,8,20,40,80,100],
        #'p':[2],
        #'metric':['minkowski'],
        'n_jobs':[-1]
    }
]

```

```

# Run automated tuning
# Note that it uses StratK-Fold of 3 instead of the recommended min. of 5
# Note that n_jobs can be set -1 as long as the system resources (CPU & RAM) can handle it
tuner = RandomizedSearchCV(knn_grid, KNeighborsClassifier(), refit=True, cv=3,
                           verbose=3, n_jobs=-1, error_score=0, random_state=1)
tuner.fit(X_train,y_train)
tuner.predict(X_test)
knn_t = tuner.best_params_(knn_grid, KNeighborsClassifier(), X_train, y_train, X_test)

# Print best tuned parameters determined
print("KNN Auto Tune Best Params:\n", knn_t)
print("")

# Create the model with the best hyperparam config determined
knn = KNeighborsClassifier(**knn_t)

# Train the model on the
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)

# Print classification report
knn_cr = classification_report(y_test, y_pred, digits=4)
print(knn_cr)

# Show Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix_df = pd.DataFrame(conf_matrix)
knn_conf_matrix = classification_report(y_test,y_pred, digits=4)
plt.figure(figsize=(4,4))
sns.set(font_scale=1.3)
sns.heatmap(conf_matrix_df, cbar=False, annot=True, fmt=".0f")
plt.title("KKN Confusion Matrix")
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.show()

```

Figure 23 - Automated Hyperparameter Tuning Pseudocode

6.4. Saving the Trained Model as a File

The trained model can be saved as a file which can be reused later on whenever further testing is needed or whenever the model will be deployed as part of production software. The pseudocode is as shown in . Note that this wasn't implemented on the actual code used in ML test.

```
import time, datetime
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn.model_selection as model_selection

#ML Metrics
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score
from sklearn.metrics import recall_score, roc_auc_score, classification_report, ConfusionMatrixDisplay
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import cross_val_score

#ML Model
from sklearn.ensemble import HistGradientBoostingClassifier as HGBT

#ML Model to File
from joblib import dump, load

#Loading dataset
malbehavd = pd.read_csv('MalbehavD_LabelEncoded.csv', low_memory=False)

#Parsing X (features) and y (labels)
malbehavd_feats = []
for i in range(malbehavd.shape[1]-2):
    malbehavd_feats.append(str(i))
X = malbehavd[malbehavd_feats]
y = malbehavd['malware'].to_numpy()

#Dividing dataset to train and test
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=.20, random_state=1)
```

```

#Model training on manual tuning
hgbt = HGBT(loss='log_loss', learning_rate=0.1, max_iter=1000, max_leaf_nodes=None, max_depth=None,
            min_samples_leaf=20, l2_regularization=0.0, max_bins=255, categorical_features=None,
            monotonic_cst=None, interaction_cst=None, warm_start=False, early_stopping='auto',
            scoring='loss', validation_fraction=0.1, n_iter_no_change=10, tol=1e-07, verbose=False,
            random_state=1, class_weight=None)
hgbt.fit(X_train, y_train)
y_pred = hgbt.predict(X_test)
hgbt_cr = classification_report(y_test, y_pred, digits=4)
print(hgbt_cr)

#Dumping trained/fitted model to file; File extension can technically be arbitrary
dump(hgbt, './Saved Model/saved_model.joblib')

#Loading recently dumped trained/fitted model file; File extension can technically be arbitrary
hgbt_file = load('./Saved Model/saved_model.joblib')

#Using dumped file for prediction
y_pred = hgbt_file.predict(X_test)
hgbt_cr = classification_report(y_test, y_pred, digits=4)
print(hgbt_cr)

```

Figure 24 - Saving Trained Model to File Pseudocode

6.5. Performance Metrics

The performance metrics selected for the ML Test, as defined in [7] are as shown below.

1. **Accuracy** – The ratio of cases the model correctly predicted.
2. **F1-Score** – The harmonic mean of positive predictive value and recall. (Note: F1-Score Weighted was used in the ML Test)
3. **Precision** – The ratio of truly positive cases from all cases the model predicted positive.
4. **Recall** – The ratio of positive cases predicted as positive.
5. **ROC-AUC** – A range value (0-1; worst to perfect) representing the ROC curve, which is graphed as the Recall/TPR on the y-axis and FPR on the x-axis [8].

6.6. Other Metrics

6.6.1. Training Time

Training time, as measured in seconds, is the time taken by each model to train using a given dataset. The pseudocode for its execution is as follows:

```
startTime = time.time()
rf.fit(X_train,y_train)
endTime = time.time() - startTime
```

Figure 25 - Training Time Pseudocode

6.6.2. Prediction Time

Prediction time, as measured in milliseconds, is the time taken by the model to predict a given sample. The pseudocode for its execution is as follows:

```
#For a set of test samples
startTime = time.time()
y_pred = rf.predict(X_test)
endTime = 1000*(time.time() - startTime)/len(X_test)

#For a single sample
startTime = time.time()
y_pred = rf.predict(X_sample)
endTime = 1000*(time.time() - startTime)
```

Figure 26 - Prediction Time Pseudocode

6.6.3. Tuning Time

Tuning time, as measured in seconds, is the time taken by the model be automatically tuned using 'RandomizedSearchCV'. This metric however is not absolute nor definitive as it is affected by many factors such as tuning configuration, tuning parameter size for each specific model, training and prediction time of the model, and dataset size.

6.7. Tests and Comparisons

The tests will be done on both default and tuned configurations. Improvements (also known as changes) during comparison will be measured in % where the ranges are as follows:

Table 11 - Improvement Value Definition

Range	Interpretation
$<\pm 1\%$	Miniscule Change (i.e., insignificant)
$>\pm 1$ to 10%	Minor Change
$>\pm 10$ to 50%	Considerable Change
$>\pm 50\%$	Major Change

6.7.1. Stratified K-Folds Test

On each dataset & configuration combination, Stratified K-Fold will be used as suppose to the normal K-Folds in order "*to ensure that relative class frequencies are effectively sustained in each train and validation fold when using stratified sampling rather than random sampling*" [9]. The number of folds executed for each model is 10 which is well within the recommended range of 5-10. The results of the tests won't necessarily be shown in the main content of the paper; however, the results it produces are used in the succeeding two comparisons/tests.

6.7.2. Default and Tuned Model Performance Comparison

The model's default and tuned config on each model trained on each dataset is compared to one another. The data to be presented in this test will be a range of $-/+$ percentages where the $[-]$ values will indicate that Default Model is a better tuning while a $[+]$ value will indicate that Tuned Model is a better one.

In this test, directionality of the value (either a '+' or '-') matters as it dictates which tuning is better.

The test will utilize the results from the Stratified K-Folds Test to make the comparison.

6.7.3. Dataset Performance Comparison

The dataset's impact in model performance (on both default and tuned) is compared to one another. The data to be presented in this test will be a range of $-/+$ percentages where the $[-]$ values will indicate that MalbehavD-V1 is a better dataset while a $[+]$ value will indicate that Oliveira is a better one.

In this test, directionality of the value (either a '+' or '-') matters as it dictates which dataset is better.

The test will utilize the results from the Stratified K-Folds Test to make the comparison.

6.7.4. Model Robustness Test

Model Robustness will be tested by means of training a model on one dataset and testing on another. This aims to determine how good a model is when encountering a similar dataset but not exactly as trained. Simply put, its either that the model can encounter unknown API calls (relative to what the model is trained from) or not. This test also investigates which dataset reshaping technique (i.e., trimming/maximizing) performs better.

In this test, amplitude of the value matters as it dictates if the trained model on a specific dataset performed best in testing on another dataset (i.e., indicating robustness).

6.7.5. Time Test

Testing training and prediction time for each model and dataset combination. The increase in time is also measured here where [-] means a reduction in time while a [+] means an increase in time.

In this test, directionality of the value (either a '+' or '-') matters as it dictates which model/dataset is better.

The test will utilize the results from the Stratified K-Folds Test to make the comparison.

6.8. Results

6.8.1. Default vs Tuned Model Performance Comparison

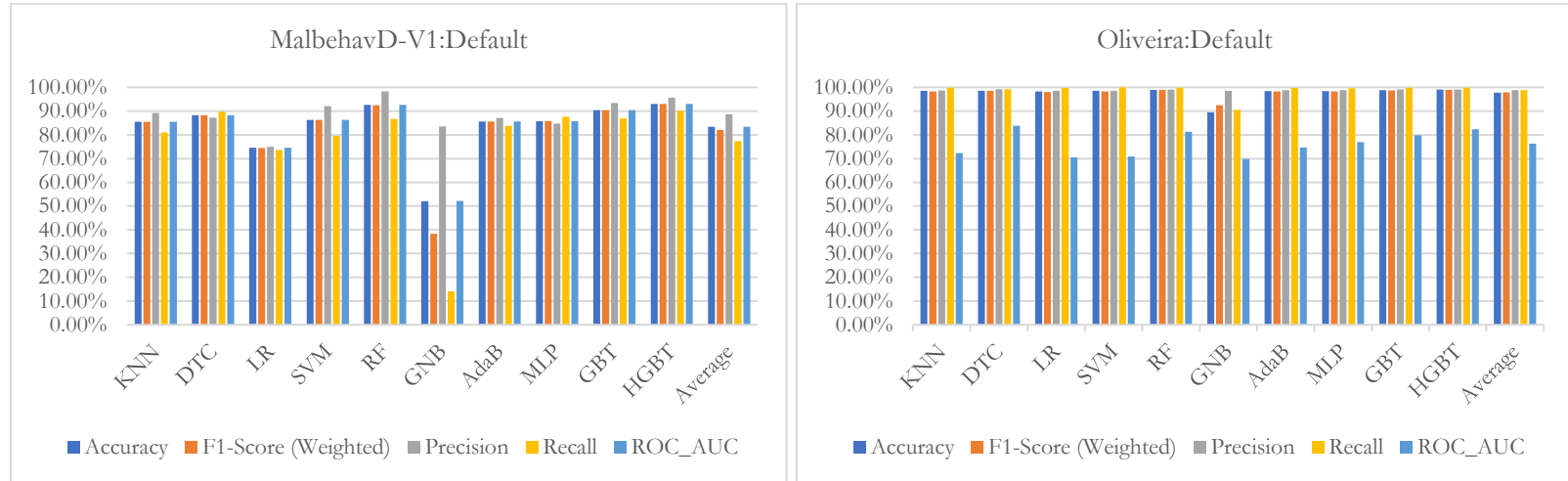


Figure 27 - Default Config (Default vs Tuned)

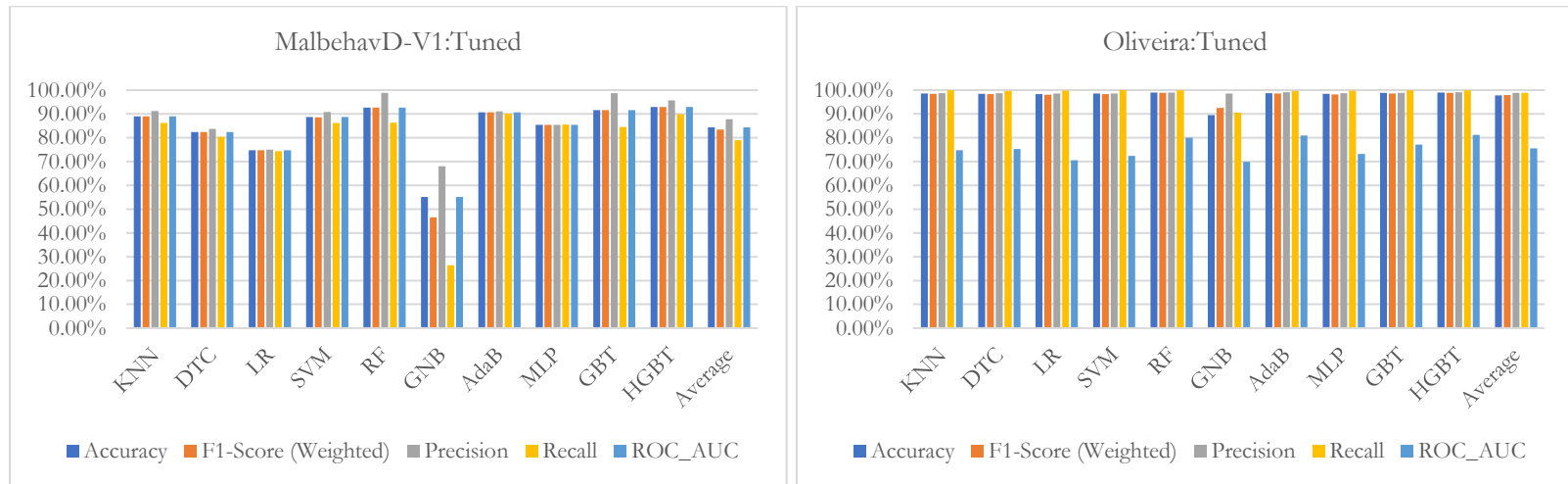


Figure 28 - Tuned Config (Default vs Tuned)

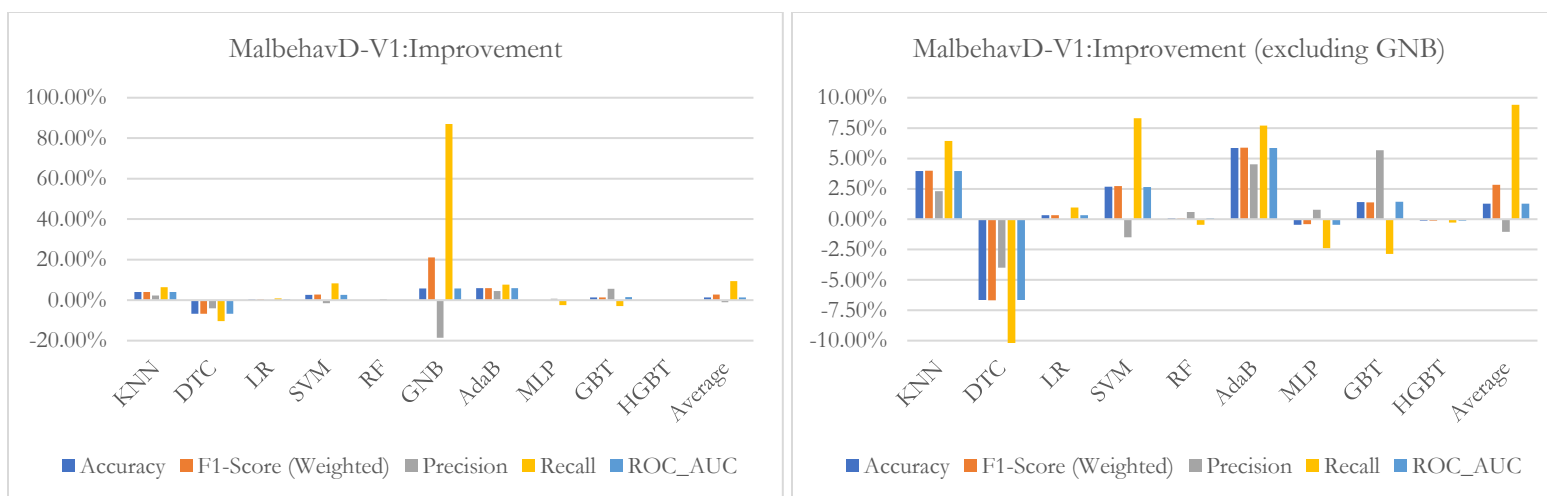


Figure 29 - Malbehavd-V1 Improvement Post-Tuning

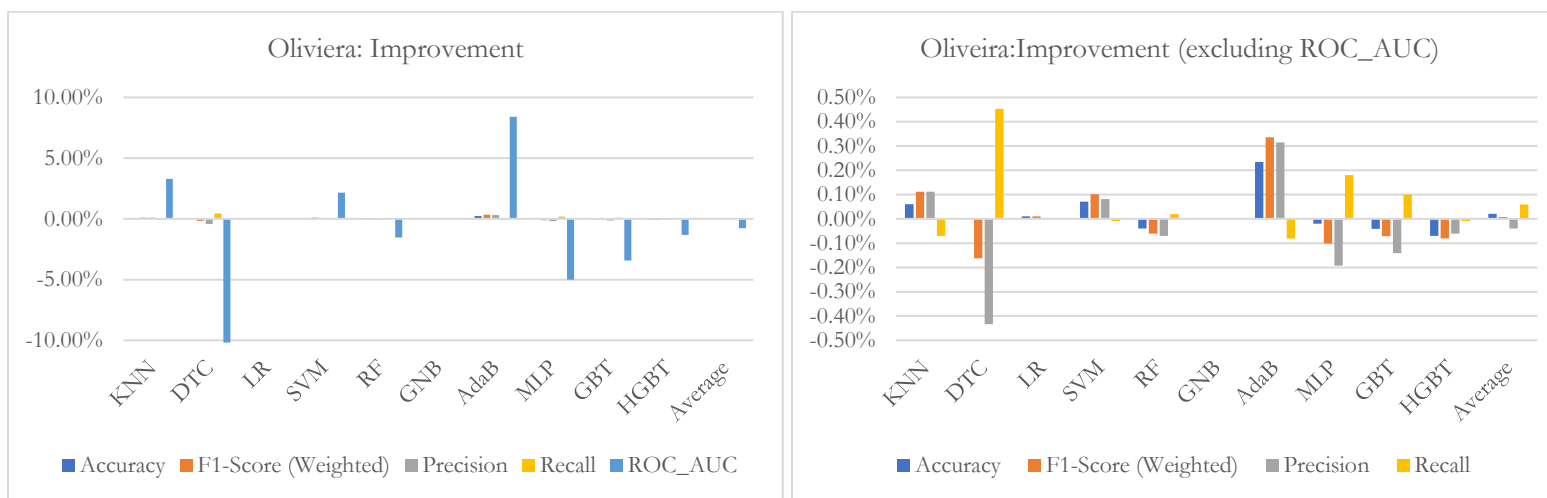


Figure 30 - Oliveira Improvement Post-Tuning

6.8.2. Dataset Performance Comparison

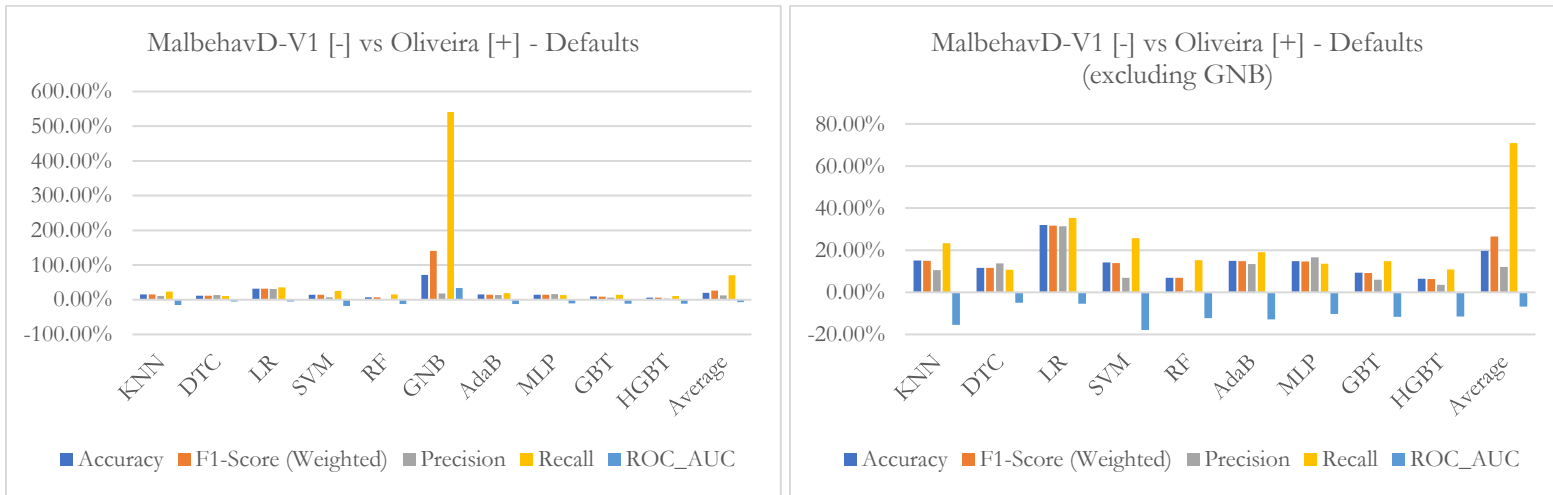


Figure 31 - MalbehavD-V1 vs Oliveira (Default)

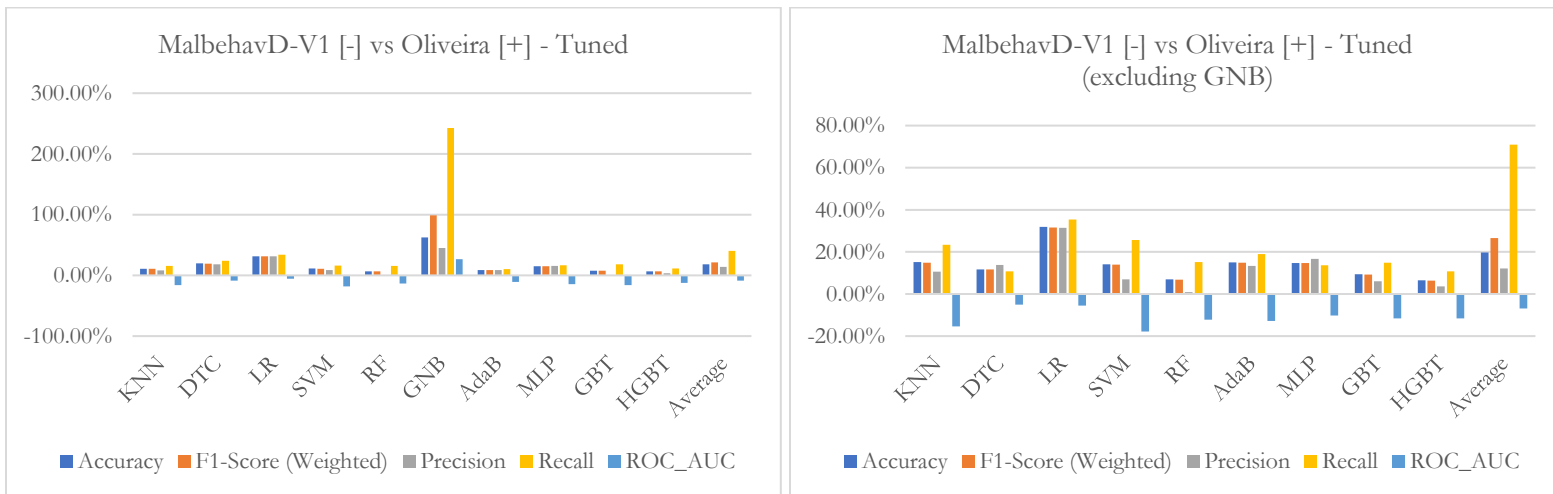


Figure 32 - MalbehavD-V1 vs Oliveira (Tuned)

6.8.3. Model Robustness Test

6.8.3.1. Trimmed

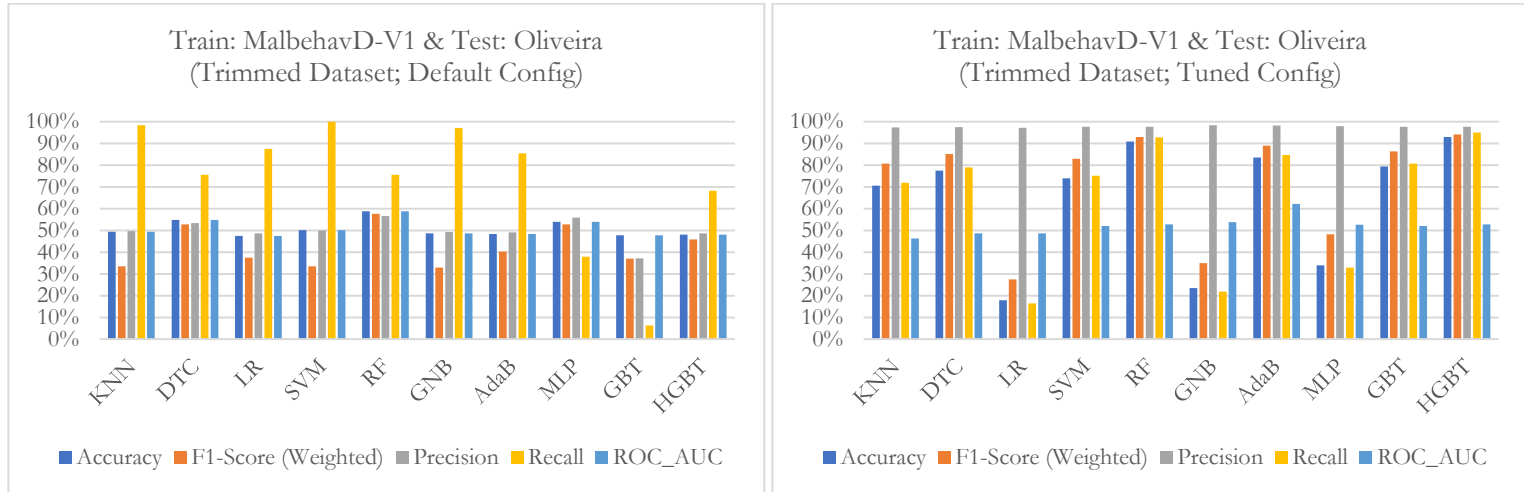


Figure 33 - Train: MalbehavD-V1 Test: Oliveira (Trimmed)

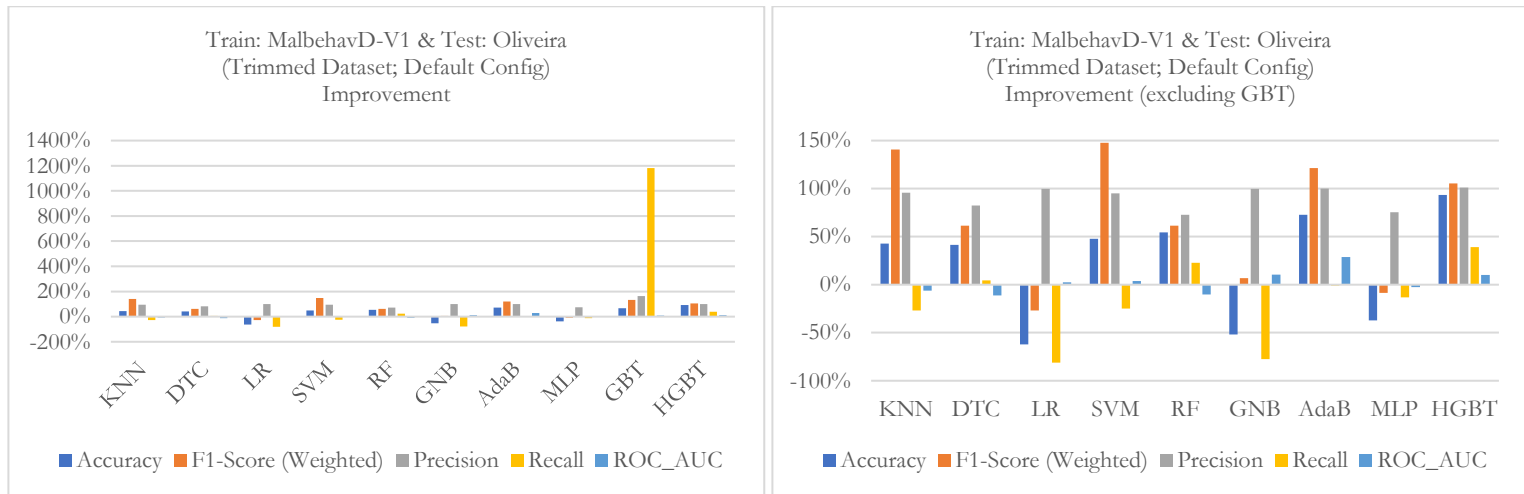


Figure 34 – Train: MalbehavD-V1 Test: Oliveira (Trimmed) - Improvement

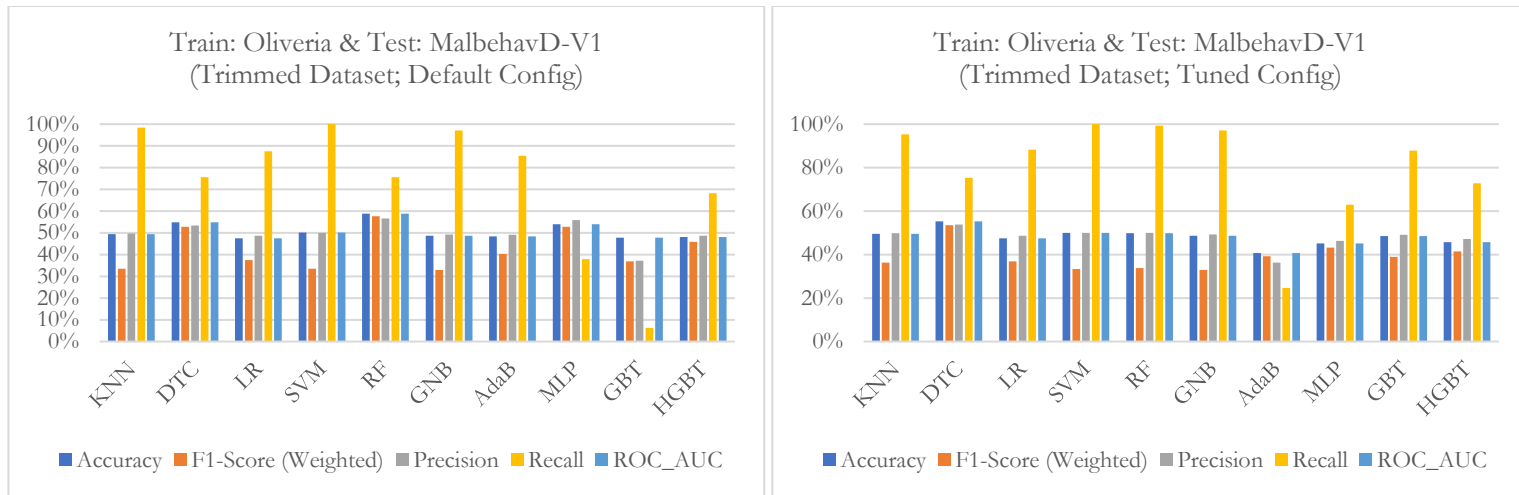


Figure 35 - Train: Oliveira Test: MalbehavD-V1 (Trimmed)

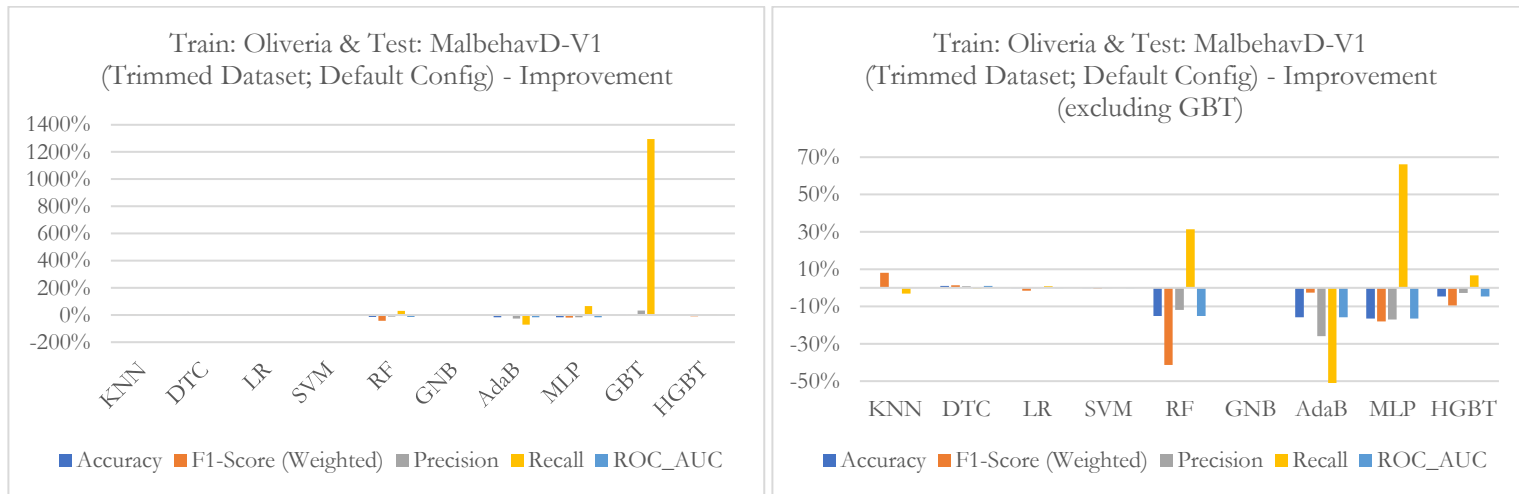


Figure 36 - Train: Oliveira Test: MalbehavD-V1 (Trimmed) - Improvement

6.8.3.2. Maximized

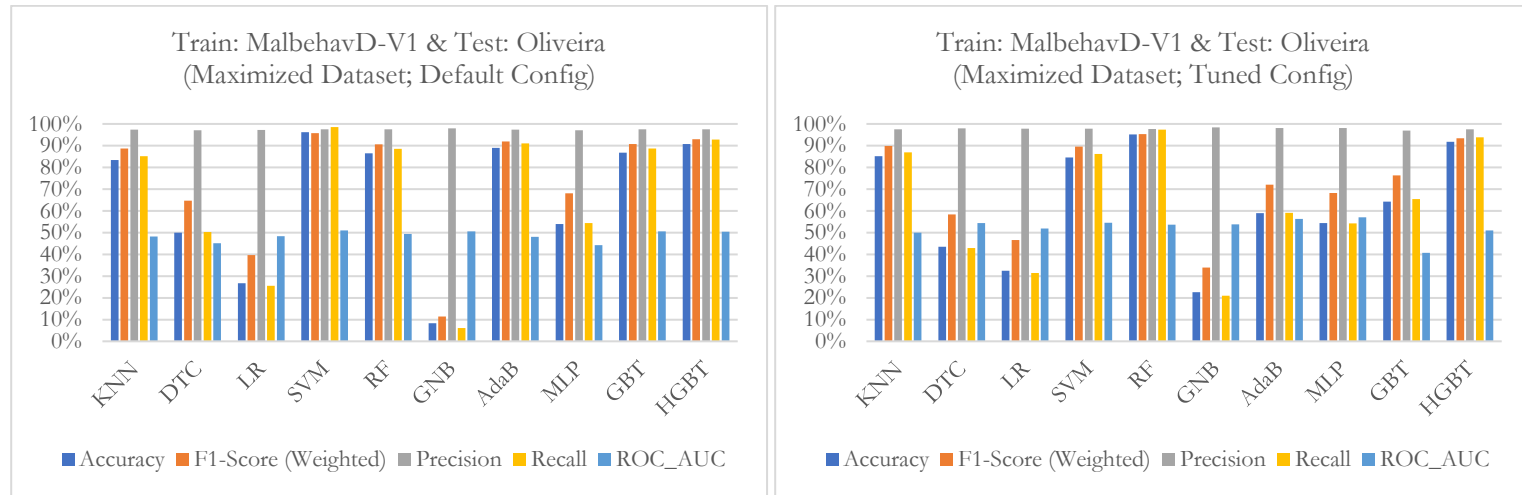


Figure 37 - Train: MalbehavD-V1 Test: Oliveira (Maximized)

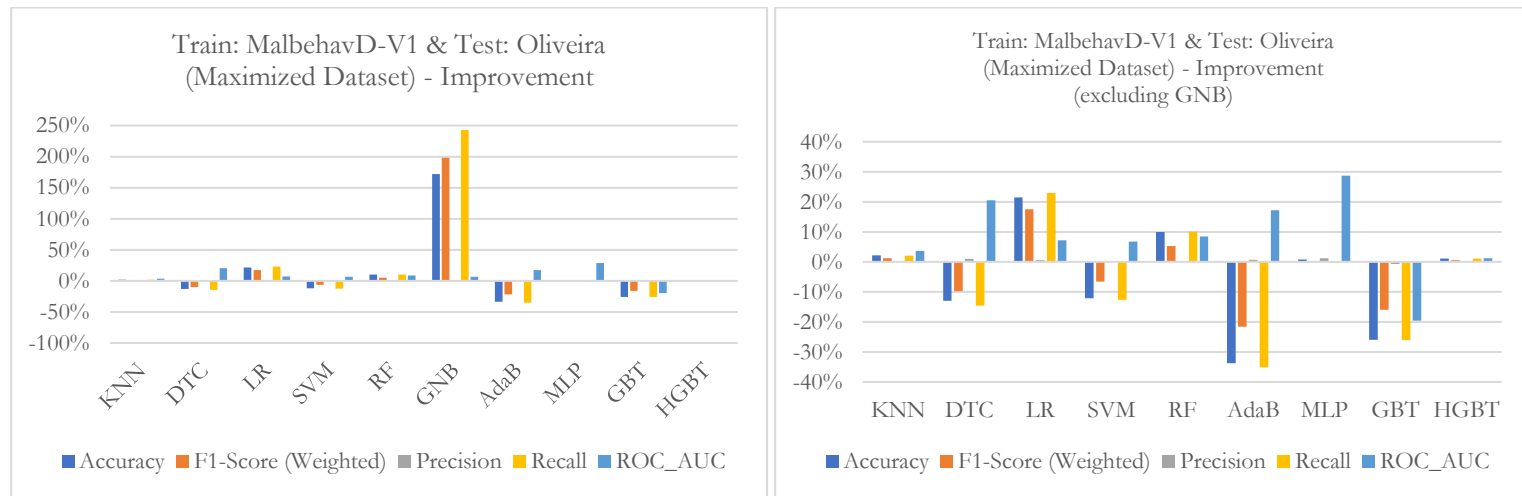


Figure 38 - Train: MalbehavD-V1 Test: Oliveira (Maximized) - Improvement

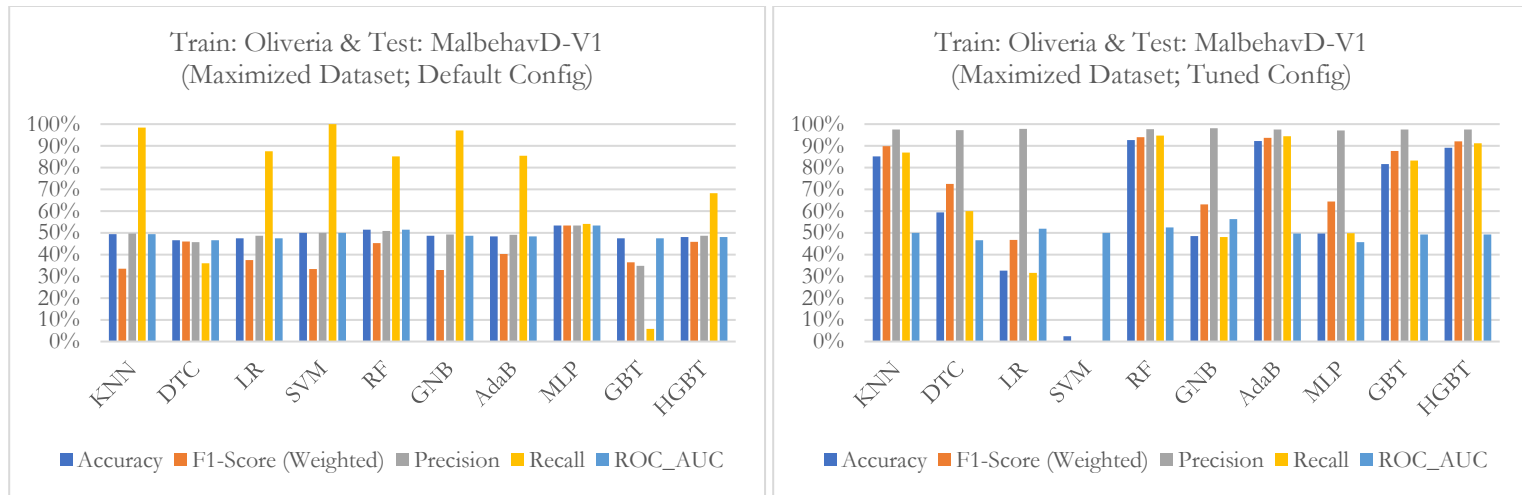


Figure 39 - Train: Oliveira Test: MalbehavD-V1 (Maximized)

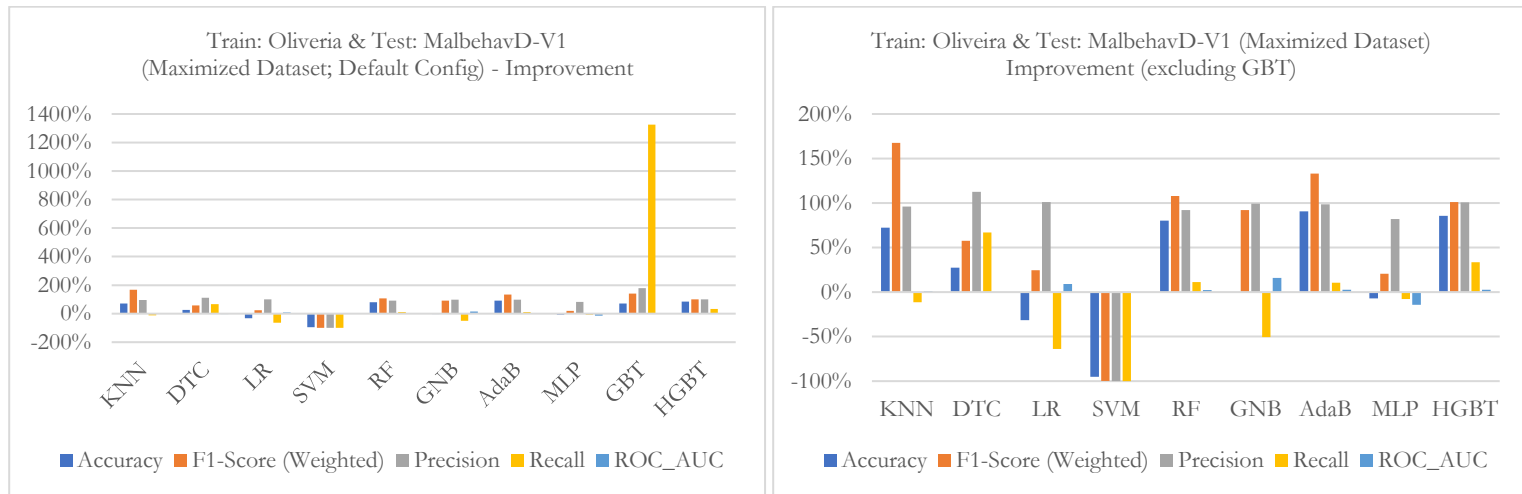


Figure 40 - Train: Oliveira Test: MalbehavD-V1 (Maximized) - Improvement

6.8.3.3. Comparison

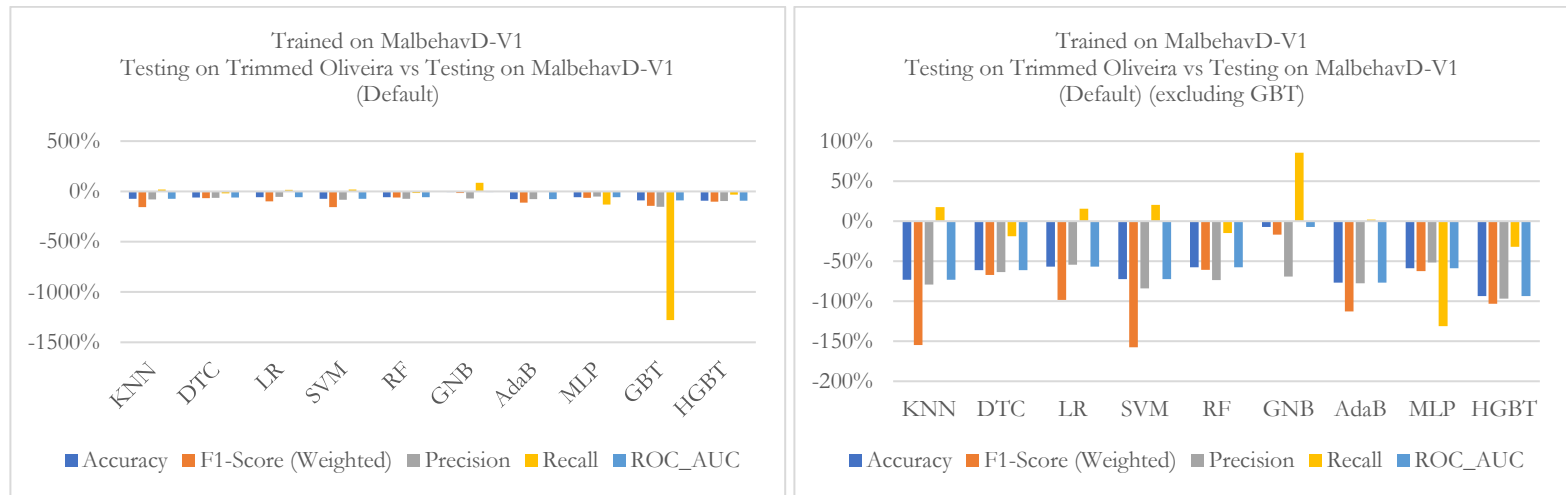


Figure 41 - Testing on Trimmed Oliveira [+] vs Testing on MalbehavD-V1 [-] (Default)

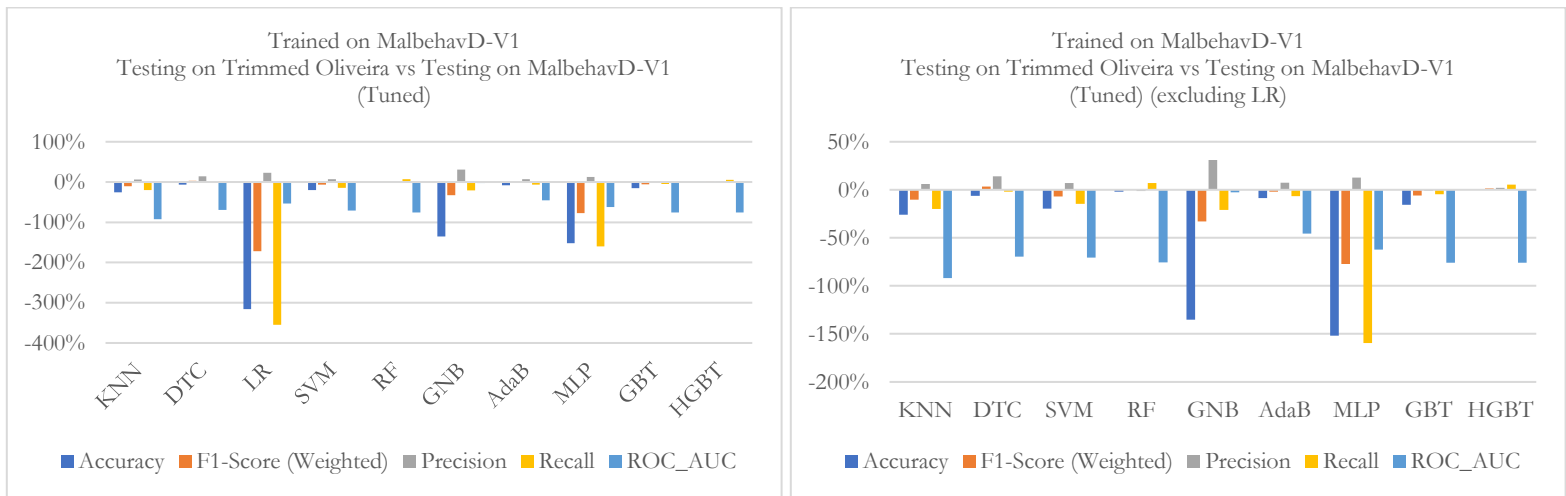


Figure 42 - Testing on Trimmed Oliveira [+] vs Testing on MalbehavD-V1 [-] (Tuned)

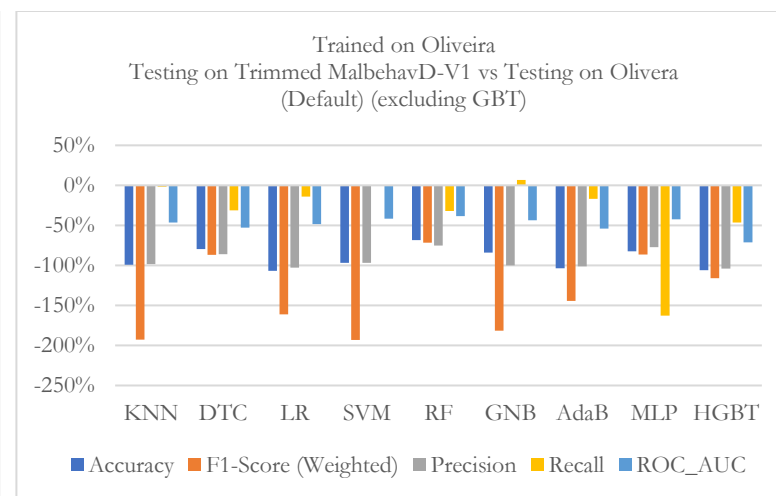
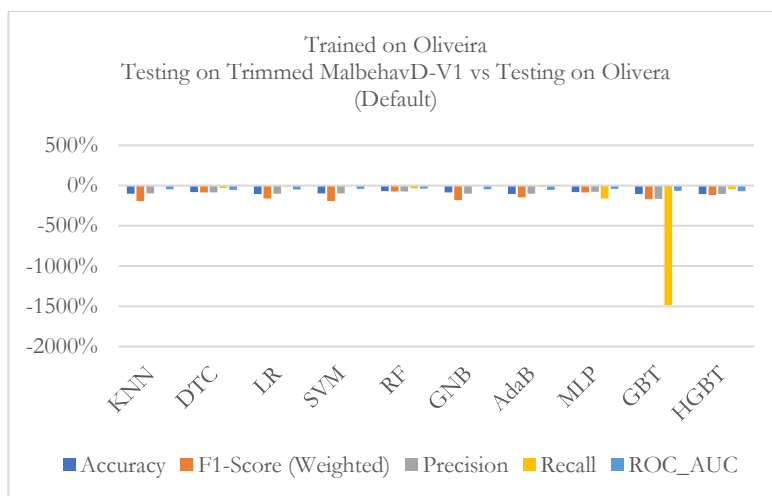


Figure 43 - Testing on Trimmed MalbehavD-V1 [+] vs Testing on Oliveira [-] (Default)

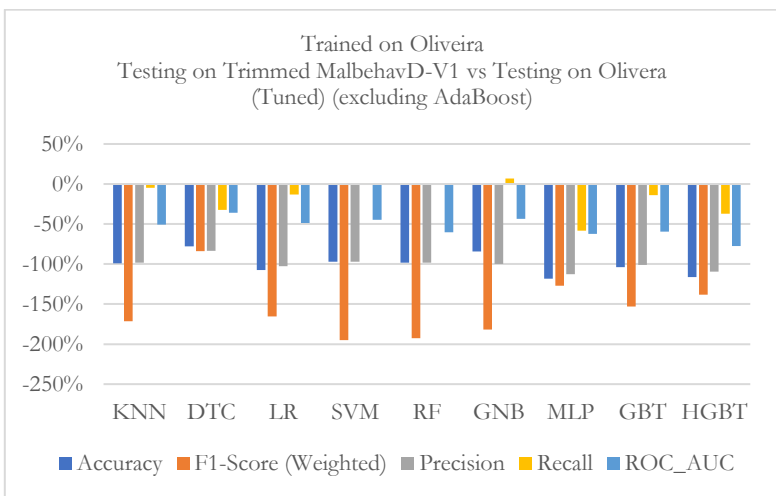
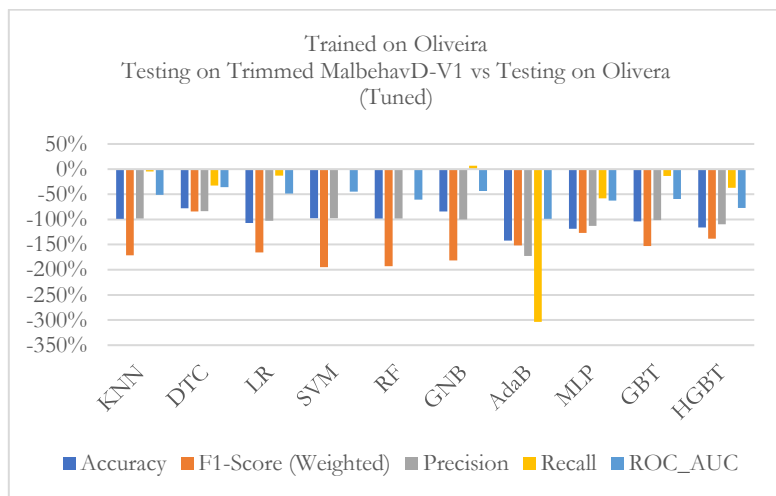


Figure 44 – Testing on Trimmed MalbehavD-V1 [+] vs Testing on Oliveira [-] (Tuned)

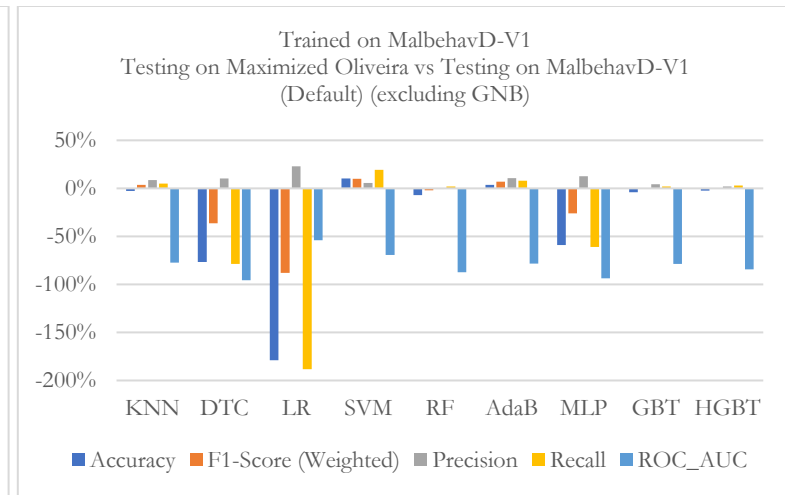
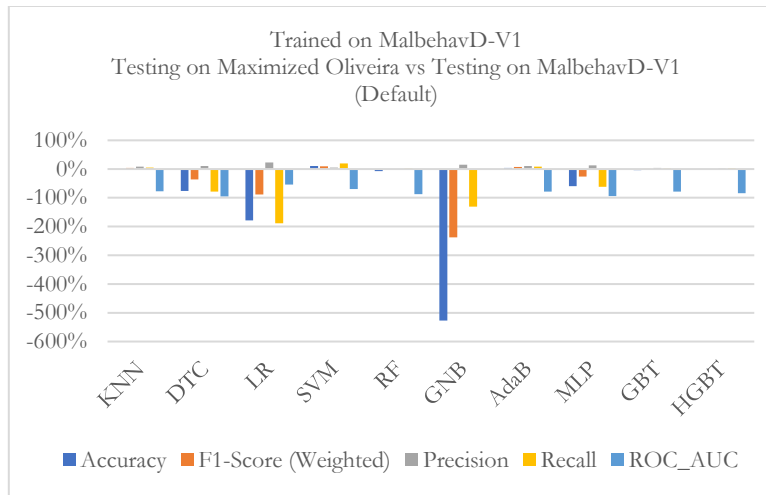


Figure 45 - Testing on Maximized Oliveira [+] vs Testing on MalbehavD-V1 [-] (Default)

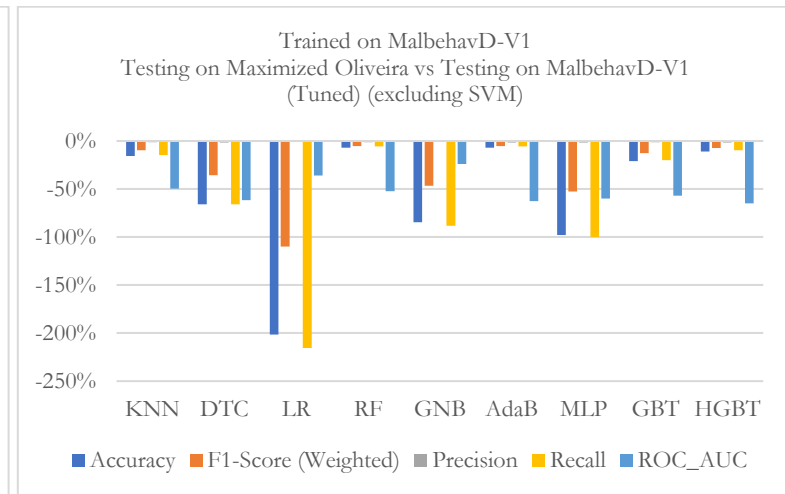
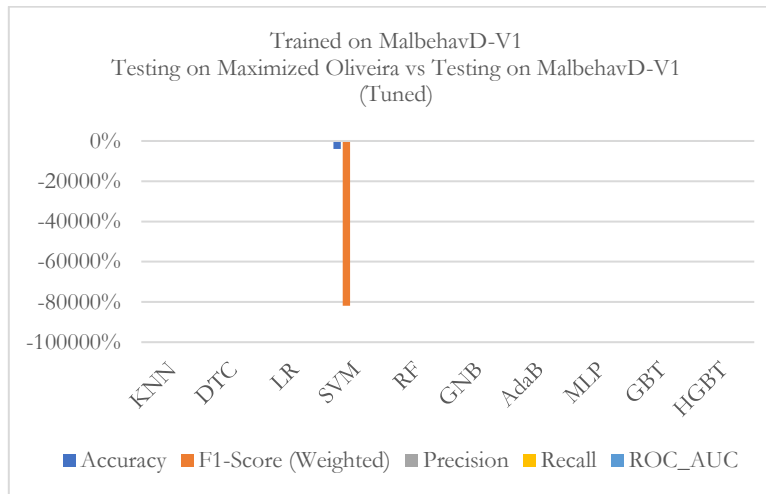


Figure 46 – Testing on Maximized Oliveira [+] vs Testing on MalbehavD-V1 [-] (Tuned)

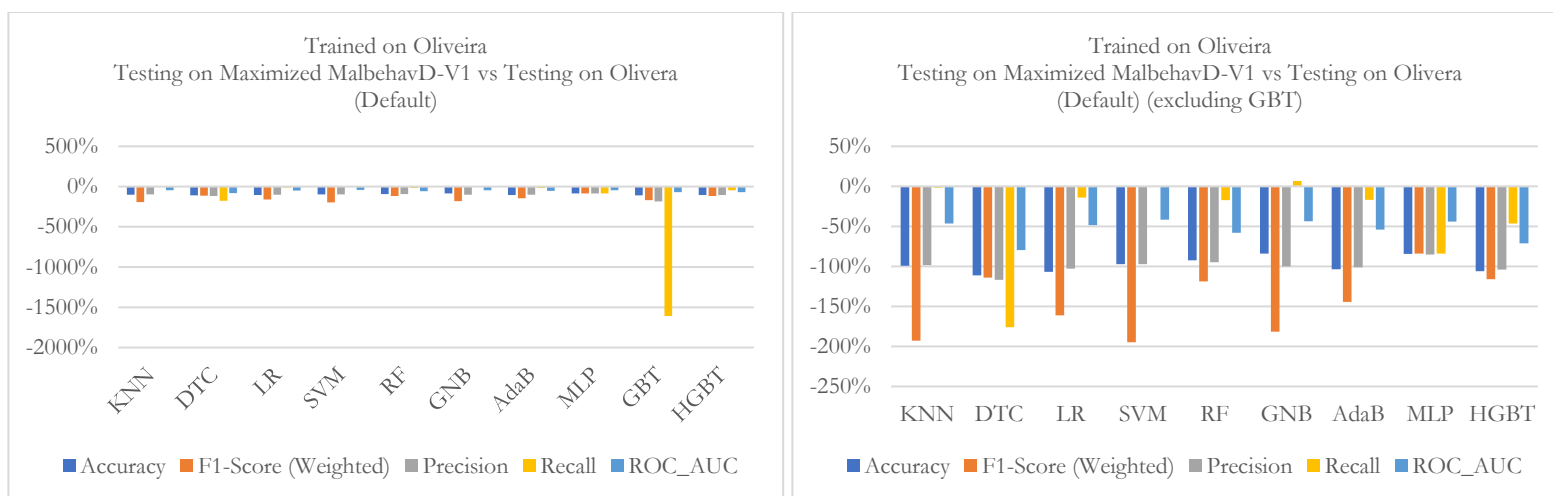


Figure 47 - Testing on Maximized MalbehavD-V1 [+] vs Testing on Oliveira [-] (Default)

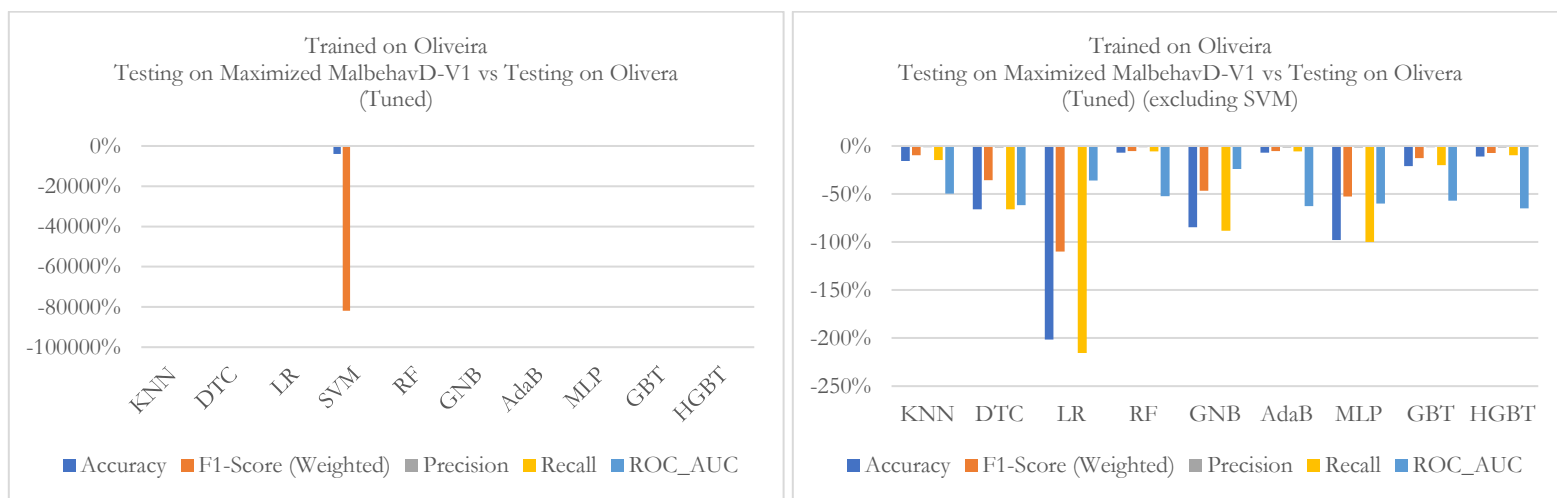


Figure 48 - Testing on Maximized MalbehavD-V1 [+] vs Testing on Oliveira [-] (Tuned)

6.8.4. Time Test

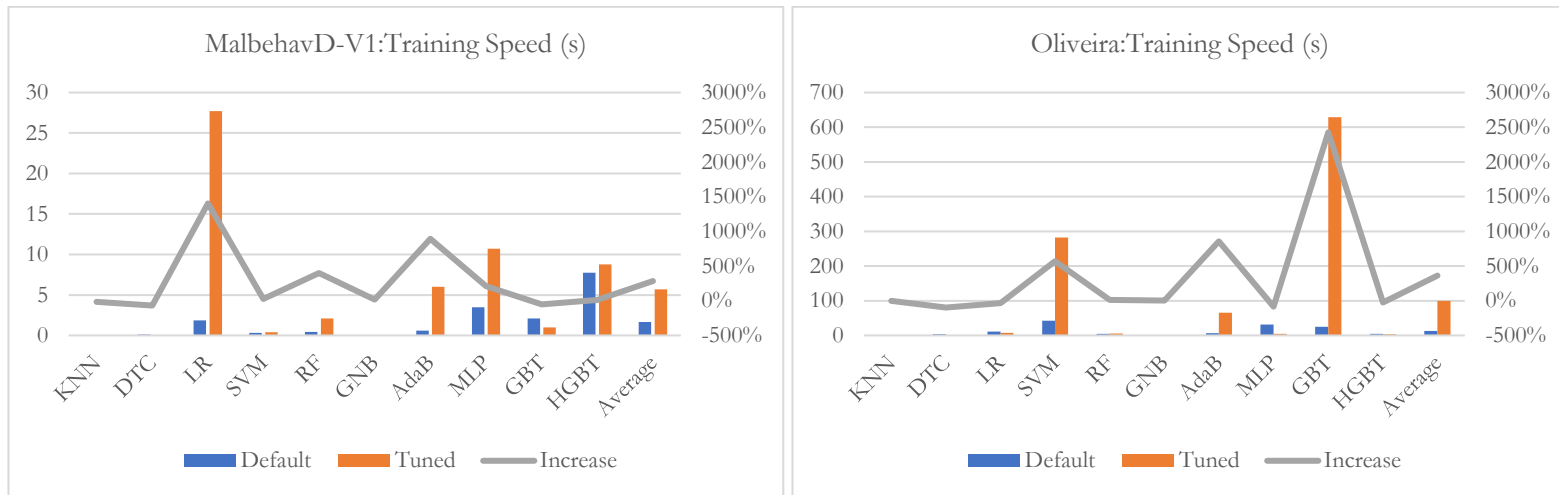


Figure 49 - Training Time (s)

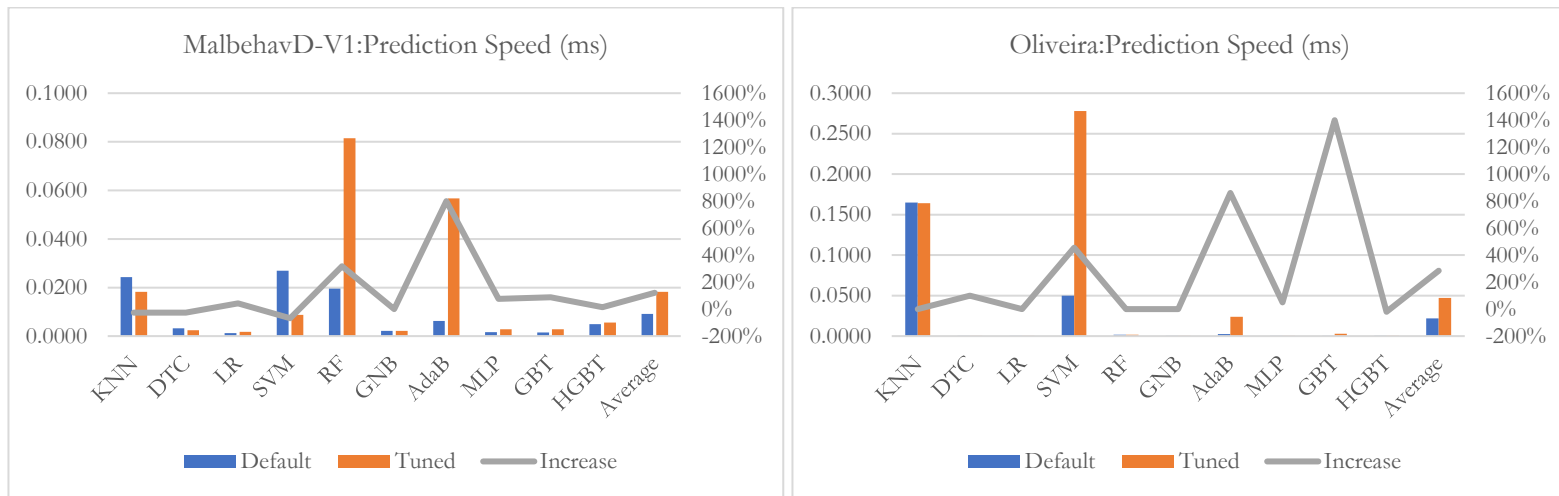


Figure 50 - Prediction Time (ms)

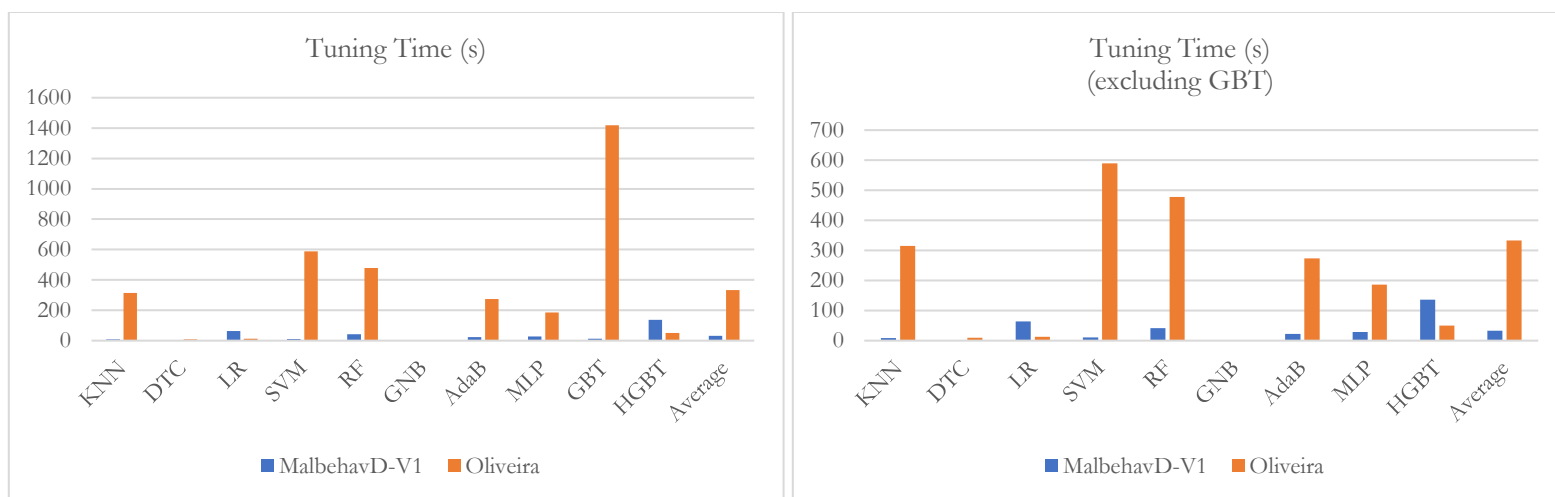


Figure 51 - Tuning Time (s)

7. Observations, Interpretations, and Discussion of Data

7.1. Default vs Tuned Model Performance Comparison

For models trained in MalbehavD-V1, tuning did result to minor improvements on KNN, SVM, LR, and AdaBoost models. The same cannot be said for DTC, MLP, and HGBT models where it experienced minor deterioration in performance in most/all metrics. GBT and GNB did experience a mixed case in changes where it has both varying levels of improvement and deterioration in certain metrics. HGBT and RF meanwhile did not experience any significant changes after tuning, implying that the performance might've been optimal from the beginning. On average (across all ML models), there is a minor increase in recall and F1-score (weighted), while the rest of the metrics have miniscule changes. The summary models that improve and deteriorate after tuning are as follows:

Table 12 - Changes After Tuning (MalbehavD-V1)

Improved	Deteriorated	Mixed	No Changes
KNN, SVM, LR, AdaBoost	DTC	GBT, GNB	HGBT, RF

For models trained in Oliveira, tuning did not improve all models in most metrics except for ROC-AUC where it showed varying changes on most models (DTC, RF, MLP, GBT, & HGBT) experienced minor deterioration while the remaining (KNN, SVM, & AdaBoost) experienced miniscule to minor improvement in this metric.

Table 13 - Changes in ROC-AUC After Tuning (Oliveira)

ROC-AUC: Improved	ROC-AUC: Deteriorated	ROC-AUC: No Change
AdaBoost	DTC	KNN, LR, SVM, RF, MLP, GBT, HGBT

In summary, when done correctly, tuning the hyperparameters of an ML model can improve its performance across a range of metrics. The dataset used in training can also play a role in determining possible improvements, if any, after tuning as some datasets may benefit from tuning while others might not. Ensemble models also had good results, mostly experiencing no changes after tuning implying an already near-optimal performance from its default tuning already.

7.2. Dataset Performance Comparison

On default tuning, MalbehavD-V1 and Oliveira showed the following range values per metric across all ML models as shown in

Table 14. The data shows that the models trained in Oliveira dataset has a higher overall performance and is less spread than MalbehavD-V1 which has a lower overall performance than the prior and is more spread-out as certain models perform very poorly than others.

This test also further proves that, regardless of the dataset used, it was found that GNB mostly performed worst on all metrics while HGBT performed best on most metrics which suggests performance advantages of ensemble ML models, like HGBT, over traditional ones.

Table 14 - Dataset Performance Range Values (Default)

Metric	MalbehavD-V1	Oliveira
Accuracy	0.5210 (GNB) to 0.9300 (HGBT)	0.8948 (GNB) to 0.9901 (HGBT)
F1-Score (Weighted)	0.3839 (GNB) to 0.9299 (HGBT)	0.9252 (GNB) to 0.9892 (HGBT)
Precision	0.7498 (LR) to 0.9826 (HGBT)	0.9854 (GNB) to 0.9920 (HGBT)
Recall	0.1412 (GNB) to 0.9012 (HGBT)	0.9050 (GNB) to 0.9999 (SVM)
ROC-AUC	0.5214 (GNB) to 0.9300 (HGBT)	0.6981 (GNB) to 0.8382 (DTC)

Despite the Oliveira dataset's overall performance, it has certain weaknesses especially in terms of individual recall values for non-malicious samples when compared to the MalbehavD-V1 dataset via the classification report as seen in table of range values in Table 15 which is assumed due to the imbalance of data in Oliveira between the number of malicious and non-malicious samples.

Table 15 - Dataset Performance in Individual Recall Values (Default)

Recall	MalbehavD-V1	Oliveira
0	0.0522 (GNB) to 0.9960 (RF)	0.3467 (LR) to 0.6436 (HGBT)
1	0.7019 (LR) to 0.9698 (RF)	0.9149 (GNB) to 0.9999 (SVM)

After fine tuning, the results did have some minor changes in certain metrics where some models improved while others deteriorated, as shown in Table 16. This is especially for MalbehavD-V1 where its previous lowest values had an increase. Notably, ensemble models, still retained their places as leading models in most metrics after fine tuning, only being beaten by other models by a margin of 10^{-4} .

Table 16 - Dataset Performance Range Values (Tuned)

Metric	MalbehavD-V1	Oliveira
Accuracy	0.5514 (GNB) to 0.9288 (HGBT)	0.8948 (GNB) to 0.9895 (RF)
F1-Score (Weighted)	0.4652 (GNB) to 0.9287 (HGBT)	0.9252 (GNB) to 0.9884 (HGBT)
Precision	0.6796 (GNB) to 0.9885 (RF)	0.9854 (LR) to 0.9906 (HGBT)
Recall	0.2460 (GNB) to 0.9020 (AdaBoost)	0.9050 (GNB) to 0.9999 (SVM)
ROC-AUC	0.5516 (GNB) to 0.9288 (HGBT)	0.6981 (GNB) to 0.8120 (HGBT)

Despite fine tuning, the issue regarding individual recalls persisted where models either remained the same or only had miniscule changes.

Table 17 - Dataset Performance in Individual Recall Values (Tuned)

Recall	MalbehavD-V1	Oliveira
0	0.0522 (GNB) to 0.9960 (RF)	0.3378 (LR) to 0.6356 (AdaBoost)
1	0.7019 (LR) to 0.9698 (RF)	0.9149 (GNB) to 1.0000 (SVM)

Further comparing the two datasets, as shown in Section 4.7.2, it suggests that using Oliveira does inherently offer better performance even regardless if the model is tuned or not.

In summary, there is a notable difference between the datasets of MalbehavD-V1 and Oliveira where the latter performed best from overall performance perspective. However, that is without its flaws as the Oliveira dataset, due to its great quantity imbalance between malicious and non-malicious samples, resulted to issues in the recall metric specifically for non-malicious samples.

7.3. Model Robustness Test

The results of the test show, as seen in Section 4.7.3.3, reveals that all models failed in varying degrees when testing with a dataset different from what the model is trained from. Table 9 shows the different configurations/pairings between reshaped training and testing datasets.

In terms of tuning, it was observed that the performance of some models had increased when after tuning which further suggests the advantages of hyperparameter tuning.

In terms of the reshaping technique, it was observed that there are mixed results depending on which dataset is used in training. There were some models trained on MalbehavD-V1 that had relatively good performance regardless of the reshaping techniques used on both training and testing. In comparison there were also some models trained on Oliveira dataset meanwhile did perform as well as the prior but only if the Oliveira dataset is maximized.

In terms of the dataset used for training, using the MalbehavD-V1 dataset resulted to better model robustness which can be attributed on the number of its features (i.e., dataset breadth) despite its miniscule no. of samples compared to the Oliveira dataset.

In summary, all models trained in a specific dataset does not result in high model robustness. However, interventions such as tuning, input dataset reshaping, and the which dataset used in training can contribute to improving the performance of the model to make it more robust when encountering unforeseen/untrained datasets.

7.4. Time Test

In terms of training times, there is a major increase between the default and tuned models which can be attributed on the complexity of how the model is being trained on a given dataset. The dataset itself, particularly its size, played a role in the increase/decrease of the training time as this is associated to the amount of data that the model must process as part of its training. Additionally, it is apparent that the system's computing and memory capacity also played a role in the training times that resulted in this test.

Among ensemble ML models, AdaBoost did experience an increase in training time by more than 850% on both datasets while the rest of the ensemble models, GBT and HGBT, did experience a mix of increase/decrease in training time depending on the dataset.

Table 18 shows the minimum and maximum values of the training times for each dataset on the model's tuning while Table 19 shows which models on each dataset have sped up or slowed down after tuning.

Table 18 - Training Time Ranges (s)

Dataset	MalbehavD-V1		Oliviera	
Change	Default	Tuned	Default	Tuned
Lowest	0.0113 (KNN)	0.0096 (KNN)	0.0357 (KNN)	0.0351 (KNN)
Highest	7.7572 (HGBT)	27.7076 (LR)	42.0474 (SVM)	629.1891 (GBT)

Table 19 - Training Time Post-Tuning Change Summary

Dataset	Improved (i.e., faster)	Deteriorated (i.e., slower)
MalbehavD-V1	KNN (-15%) DTC (-69%) GBT (-53%)	LR (1402%) SVM (24%) RF (398%) GNB (17%) AdaBoost (890%) MLP (210%) HGBT (13%)
Oliveira	KNN (-2%) DTC (-98%) LR (-34%) MLP (-84%) HGBT (-25%)	SVM (570%) RF (11%) GNB (4%) GBT (2427%)

In terms of prediction times, most models did experience a considerable to major increase in prediction times in both datasets as shown in Table 20. However, as this is measured milliseconds, the values show that it might be imperceivable except during cases where there are bulk samples that will be predicted by the ML model.

Table 20 - Prediction Time Ranges (ms)

Dataset	MalbehavD-V1		Oliviera	
Change	Default	Tuned	Default	Tuned
Lowest	0.0012 (LR)	0.0017 (LR)	0.0001 (LR & DTC)	0.0001 (LR)
Highest	0.0269 (SVM)	0.0815 (RF)	0.1650 (KNN)	.2779 (SVM)

In terms of tuning times, its value is influenced by many factors mentioned in Section 4.5.3, hence it is not conclusive. However, it is still useful as it gives a sense relativity in terms of how difficult each models and datasets is being used in terms of time constraint associated to tuning it. Between the two datasets, there is an increase in tuning time when comparing tuning on MalbehavD-V1 and Oliveira which is attributable to Oliveira dataset's size. In both datasets, GNB had the lowest tuning time of just around one second while GBT and HGBT had the highest tuning time for both MalbehavD-V1 and Oliveira datasets respectively as shown in Table 21.

Table 21 - Tuning Time Ranges (s)

Dataset	Minimum Tuning Time	Maximum Tuning Time
MalbehavD-V1	0.3784 (GNB)	136.7645 (HGBT)
Oliviera	1.2072 (GNB)	1418.0289 (GBT)

In summary, different factors play a role in training, prediction, and tuning times of different models (default and tuned) and datasets. This, however, shows another perspective regarding advantages and disadvantages of these models and datasets as one model may have the advantage of being high performant but at a disadvantage of being difficult/time-consuming to train, while another model may have the advantage of being easy/quick to train but at the expense of its performance.

7.5. Other Observations

During tuning process, it was noticeable that tuning models for Oliveira, especially for the case of KNN, with the tuning parameter 'n_jobs' or workers set to -1 (i.e., using all cores) resulted to a system crashing due to high memory usage associated to multiple instances of the same workload (KNN using ~2GB for each instance) with different model hyperparameter configuration. This suggests that the available memory must be in line with the dataset size.

There are certain models that are single-threaded which suggests that the single-core performance of the CPU might be important as much as multi-core performance is.

High-speed storage may also be helpful as it may contribute to faster data read and write especially during dataset pre-processing and cleaning.

8. CONCLUSION

To conclude, building ML models require pre-processed and cleaned datasets, ML libraries, and powerful computing systems that can handle the demanding task of dataset preparation, model tuning, model training, and model evaluation.

When comparing ML models to one another, the system specifications of the computing environment and datasets used are important constant variables to consider which ensures minimal variability of the tests from another such that only the aspects related to tuning, training, and evaluating are non-constant variables which can still be easily controlled as it is software-based.

Despite the assumptions regarding balanced datasets resulting to better performing models, it is apparent that the current tests show that it does not necessarily result to high individual recall scores. While MalbehavD-V1 had better individual recalls as it is more balanced than its

counterpart, its lack of sample quantity hampers its performance from an overall perspective. The Oliveira dataset, meanwhile, despite its massive sample imbalance problem that results to lower its individual recall score for benign samples, has a better performance when looked at from an overall/macro perspective. In terms of model robustness however, the opposite goes for the two datasets where MalbehavD-V1 outcompetes Oliveira on certain models regardless of the reshaping technique used.

The test also determined that traditional models were mostly found to be relatively light to tune, train, and evaluate albeit at the expense of its performance across a broad range of metrics. Meanwhile, ensemble models were found to be quite heavy to tune, train, and evaluate, however at the advantage of having better performance across a broad range of metrics.

9. RECOMMENDATIONS

The following are recommended to consider when conducting a similar experiment:

1. Consider having a more powerful system with ample amount of computing capacity and memory relative to the datasets to use.
2. Consider using 'GridSearchCV' if possible to allow for a more exhaustive search if deemed viable and more beneficial than 'RandomSearchCV'.
3. Consider increasing the 'cv' value on the tuning parameters to allow the model to reduce instances of overfitting that can affect the model hyperparameter selection on the tuning process.
4. Consider increasing hyperparameter tuning options when tuning to extract the most amount of performance possible from the model.
5. Consider implementing the same test on the libraries of XGBoost and LightGBM to determine procedure design viability.

10. REFERENCES

- [1] M. Sewak, S. K. Sahay, and H. Rathore, “Comparison of Deep Learning and the Classical Machine Learning Algorithm for the Malware Detection,” in *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Busan: IEEE, Jun. 2018, pp. 293–296. doi: 10.1109/SNPD.2018.8441123.
- [2] O. Aslan and R. Samet, “A Comprehensive Review on Malware Detection Approaches,” *IEEE Access*, vol. 8, pp. 6249–6271, 2020, doi: 10.1109/ACCESS.2019.2963724.
- [3] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, “API-MalDetect: Automated malware detection framework for windows based on API calls and deep learning techniques,” *J. Netw. Comput. Appl.*, vol. 218, p. 103704, Sep. 2023, doi: 10.1016/j.jnca.2023.103704.
- [4] A. Oliveira, “Malware Analysis Datasets: API Call Sequences.” IEEE DataPort, Oct. 23, 2019. doi: 10.21227/TQQM-AQ14.
- [5] F. O. Catak and A. F. Yazı, “A Benchmark API Call Dataset for Windows PE Malware Classification,” 2019, doi: 10.48550/ARXIV.1905.01999.
- [6] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, 2011.
- [7] B. J. Erickson and F. Kitamura, “Magician’s Corner: 9. Performance Metrics for Machine Learning Models,” *Radiol. Artif. Intell.*, vol. 3, no. 3, p. e200126, May 2021, doi: 10.1148/ryai.2021200126.
- [8] D. Chicco and G. Jurman, “The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification,” *BioData Min.*, vol. 16, no. 1, p. 4, Feb. 2023, doi: 10.1186/s13040-023-00322-4.
- [9] S. Prusty, S. Patnaik, and S. K. Dash, “SKCV: Stratified K-fold cross-validation on ML classifiers for predicting cervical cancer,” *Front. Nanotechnol.*, vol. 4, 2022, Accessed: Aug. 17, 2023. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fnano.2022.972421>

11. APPENDIX

11.1. Classification Reports (MalbehavD-V1)

Dataset	MalbehavD-V1			
Tuning	Default			
Model	KNN			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8007	0.9357	0.8630	249
1 - Malicious	0.9283	0.7811	0.8484	265
Accuracy			0.8560	514
Macro-Avg	0.8645	0.8584	0.8557	514
Weighted-Avg	0.8665	0.8560	0.8554	514

Dataset	MalbehavD-V1			
Tuning	Default			
Model	DTC			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8730	0.8554	0.8641	249
1 - Malicious	0.8667	0.8830	0.8748	265
Accuracy			0.8696	514
Macro-Avg	0.8698	0.8692	0.8694	514
Weighted-Avg	0.8697	0.8696	0.8696	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	KNN			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8375	0.9518	0.8910	249
1 - Malicious	0.9481	0.8264	0.8831	265
Accuracy			0.8872	514
Macro-Avg	0.8928	0.8891	0.8870	514
Weighted-Avg	0.8945	0.8872	0.8869	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	DTC			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8014	0.8916	0.8441	249
1 - Malicious	0.8861	0.7925	0.8367	265
Accuracy			0.8405	514
Macro-Avg	0.8438	0.8420	0.8404	514
Weighted-Avg	0.8451	0.8405	0.8403	514

Dataset	MalbehavD-V1			
Tuning	Default			
Model	LR			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.7148	0.7952	0.7529	249
1 - Malicious	0.7848	0.7019	0.7410	265
Accuracy			0.7471	514
Macro-Avg	0.7498	0.7485	0.7469	514
Weighted-Avg	0.7509	0.7471	0.7468	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	LR			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.7256	0.8072	0.7643	249
1 - Malicious	0.7975	0.7132	0.7530	265
Accuracy			0.7588	514
Macro-Avg	0.7616	0.7602	0.7586	514
Weighted-Avg	0.7627	0.7588	0.7584	514

Dataset	MalbehavD-V1			
Tuning	Default			
Model	SVM			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.7613	0.9478	0.8444	249
1 - Malicious	0.9363	0.7208	0.8145	265
Accuracy			0.8307	514
Macro-Avg	0.8488	0.8343	0.8294	514
Weighted-Avg	0.8515	0.8307	0.8290	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	SVM			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8453	0.9438	0.8918	249
1 - Malicious	0.9407	0.8377	0.8862	265
Accuracy			0.8891	514
Macro-Avg	0.8930	0.8908	0.8890	514
Weighted-Avg	0.8945	0.8891	0.8889	514

Dataset	MalbehavD-V1			
Tuning	Default			
Model	RF			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8464	0.9960	0.9151	249
1 - Malicious	0.9955	0.8302	0.9053	265
Accuracy			0.9105	514
Macro-Avg	0.9209	0.9131	0.9102	514
Weighted-Avg	0.9233	0.9105	0.9101	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	RF			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8464	0.9960	0.9151	249
1 - Malicious	0.9955	0.8302	0.9053	265
Accuracy			0.9105	514
Macro-Avg	0.9209	0.9131	0.9102	514
Weighted-Avg	0.9233	0.9105	0.9101	514

Dataset	MalbehavD-V1			
Tuning	Default			
Model	GNB			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.6190	0.0522	0.0963	249
1 - Malicious	0.5213	0.9698	0.6781	265
Accuracy			0.5253	514
Macro-Avg	0.5702	0.5110	0.3872	514
Weighted-Avg	0.5687	0.5253	0.3963	514
Dataset	MalbehavD-V1			
Tuning	Default			
Model	AdaBoost			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8333	0.9036	0.8671	249
1 - Malicious	0.9016	0.8302	0.8644	265
Accuracy			0.8658	514
Macro-Avg	0.8675	0.8669	0.8657	514
Weighted-Avg	0.8685	0.8658	0.8657	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	GNB			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.6190	0.0522	0.0963	249
1 - Malicious	0.5213	0.9698	0.6781	265
Accuracy			0.5253	514
Macro-Avg	0.5702	0.5110	0.3872	514
Weighted-Avg	0.5687	0.5253	0.3963	514
Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	AdaBoost			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8958	0.9317	0.9134	249
1 - Malicious	0.9333	0.8981	0.9154	265
Accuracy			0.9144	514
Macro-Avg	0.9145	0.9149	0.9144	514
Weighted-Avg	0.9151	0.9144	0.9144	514

Dataset	MalbehavD-V1			
---------	--------------	--	--	--

Dataset	MalbehavD-V1			
---------	--------------	--	--	--

Tuning	Default			
Model	MLP			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8231	0.9157	0.8669	249
1 - Malicious	0.9114	0.8151	0.8606	265
Accuracy			0.8638	514
Macro-Avg	0.8672	0.8654	0.8637	514
Weighted-Avg	0.8686	0.8638	0.8636	514

Tuning	Tuned			
Model	MLP			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8051	0.8795	0.8407	249
1 - Malicious	0.8760	0.8000	0.8363	265
Accuracy			0.8385	514
Macro-Avg	0.8406	0.8398	0.8385	514
Weighted-Avg	0.8417	0.8385	0.8384	514

Dataset	MalbehavD-V1			
Tuning	Default			
Model	GBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8691	0.9598	0.9122	249
1 - Malicious	0.9582	0.8642	0.9087	265
Accuracy			0.9105	514
Macro-Avg	0.9136	0.9120	0.9105	514
Weighted-Avg	0.9150	0.9105	0.9104	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	GBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.7930	1.0000	0.8845	249
1 - Malicious	1.0000	0.7547	0.8602	265
Accuracy			0.8735	514
Macro-Avg	0.8965	0.8774	0.8724	514
Weighted-Avg	0.8997	0.8735	0.8720	514

Dataset	MalbehavD-V1			
Tuning	Default			
Model	HGBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8817	0.9880	0.9318	249
1 - Malicious	0.9872	0.8755	0.9280	265
Accuracy			0.9300	514
Macro-Avg	0.9345	0.9317	0.9299	514
Weighted-Avg	0.9361	0.9300	0.9298	514

Dataset	MalbehavD-V1			
Tuning	Tuned			
Model	HGBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8773	0.9759	0.9240	249
1 - Malicious	0.9747	0.8717	0.9203	265
Accuracy			0.9222	514
Macro-Avg	0.9260	0.9238	0.9221	514
Weighted-Avg	0.9275	0.9222	0.9221	514

11.2.Classification Reports (Oliveira)

Dataset	Oliveira			
Tuning	Default			
Model	KNN			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8661	0.4311	0.5757	225
1 - Malicious	0.9852	0.9982	0.9917	8551
Accuracy			0.9837	8776
Macro-Avg	0.9256	0.7147	0.7837	8776
Weighted-Avg	0.9822	0.9837	0.9810	8776

Dataset	Oliveira			
Tuning	Tuned			
Model	KNN			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9024	0.4933	0.6379	225
1 - Malicious	0.9868	0.9986	0.9927	8551
Accuracy			0.9856	8776
Macro-Avg	0.9446	0.7460	0.8153	8776
Weighted-Avg	0.9847	0.9856	0.9836	8776

Dataset	Oliveira			
Tuning	Default			
Model	DTC			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.6682	0.6444	0.6561	225
1 - Malicious	0.9907	0.9916	0.9911	8551
Accuracy			0.9827	8776
Macro-Avg	0.8294	0.8180	0.8236	8776
Weighted-Avg	0.9824	0.9827	0.9825	8776

Dataset	Oliveira			
Tuning	Tuned			
Model	DTC			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8516	0.4844	0.6176	225
1 - Malicious	0.9866	0.9978	0.9922	8551
Accuracy			0.9846	8776
Macro-Avg	0.9191	0.7411	0.8049	8776
Weighted-Avg	0.9831	0.9846	0.9825	8776

Dataset	Oliveira			
---------	----------	--	--	--

Dataset	Oliveira			
---------	----------	--	--	--

Tuning	Default			
Model	LR			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8125	0.3467	0.4860	225
1 - Malicious	0.9831	0.9979	0.9904	8551
Accuracy			0.9812	8776
Macro-Avg	0.8978	0.6723	0.7382	8776
Weighted-Avg	0.9787	0.9812	0.9775	8776

Tuning	Tuned			
Model	LR			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8085	0.3378	0.4765	225
1 - Malicious	0.9828	0.9979	0.9903	8551
Accuracy			0.9810	8776
Macro-Avg	0.8957	0.6678	0.7334	8776
Weighted-Avg	0.9784	0.9810	0.9771	8776

Dataset	Oliveira			
Tuning	Default			
Model	SVM			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9888	0.3911	0.5605	225
1 - Malicious	0.9842	0.9999	0.9920	8551
Accuracy			0.9843	8776
Macro-Avg	0.9865	0.6955	0.7763	8776
Weighted-Avg	0.9843	0.9843	0.9809	8776

Dataset	Oliveira			
Tuning	Tuned			
Model	SVM			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	1.0000	0.4400	0.6111	225
1 - Malicious	0.9855	1.0000	0.9927	8551
Accuracy			0.9856	8776
Macro-Avg	0.9927	0.7200	0.8019	8776
Weighted-Avg	0.9859	0.9856	0.9829	8776

Dataset	Oliveira			
Tuning	Default			
Model	RF			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9396	0.6222	0.7487	225
1 - Malicious	0.9901	0.9989	0.9945	8551
Accuracy			0.9893	8776
Macro-Avg	0.9649	0.8106	0.8716	8776
Weighted-Avg	0.9889	0.9893	0.9882	8776

Dataset	Oliveira			
Tuning	Tuned			
Model	RF			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9643	0.6000	0.7397	225
1 - Malicious	0.9896	0.9994	0.9945	8551
Accuracy			0.9892	8776
Macro-Avg	0.9769	0.7997	0.8671	8776
Weighted-Avg	0.9889	0.9892	0.9879	8776

Dataset	Oliveira			
Tuning	Default			
Model	GNB			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.1133	0.4133	0.1778	225
1 - Malicious	0.9834	0.9149	0.9479	8551
Accuracy			0.9020	8776
Macro-Avg	0.5483	0.6641	0.5629	8776
Weighted-Avg	0.9611	0.9020	0.9282	8776

Dataset	Oliveira			
Tuning	Tuned			
Model	GNB			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.1133	0.4133	0.1778	225
1 - Malicious	0.9834	0.9149	0.9479	8551
Accuracy			0.9020	8776
Macro-Avg	0.5483	0.6641	0.5629	8776
Weighted-Avg	0.9611	0.9020	0.9282	8776

Dataset	Oliveira			
Tuning	Default			
Model	AdaBoost			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8110	0.4578	0.5852	225
1 - Malicious	0.9859	0.9972	0.9915	8551
Accuracy			0.9834	8776
Macro-Avg	0.8985	0.7275	0.7884	8776
Weighted-Avg	0.9814	0.9834	0.9811	8776

Dataset	Oliveira			
Tuning	Tuned			
Model	AdaBoost			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.8614	0.6356	0.7315	225
1 - Malicious	0.9905	0.9973	0.9939	8551
Accuracy			0.9880	8776
Macro-Avg	0.9260	0.8164	0.8627	8776
Weighted-Avg	0.9872	0.9880	0.9872	8776

Dataset	Oliveira			
Tuning	Default			
Model	MLP			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.7111	0.5689	0.6321	225
1 - Malicious	0.9887	0.9939	0.9913	8551
Accuracy			0.9830	8776
Macro-Avg	0.8499	0.7814	0.8117	8776
Weighted-Avg	0.9816	0.9830	0.9821	8776

Dataset	Oliveira			
Tuning	Default			
Model	GBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9178	0.5956	0.7224	225
1 - Malicious	0.9895	0.9986	0.9940	8551
Accuracy			0.9883	8776
Macro-Avg	0.9536	0.7971	0.8582	8776
Weighted-Avg	0.9876	0.9883	0.9870	8776

Dataset	Oliveira			
Tuning	Tuned			
Model	MLP			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9300	0.4133	0.5723	225
1 - Malicious	0.9848	0.9992	0.9919	8551
Accuracy			0.9842	8776
Macro-Avg	0.9574	0.7063	0.7821	8776
Weighted-Avg	0.9834	0.9842	0.9812	8776

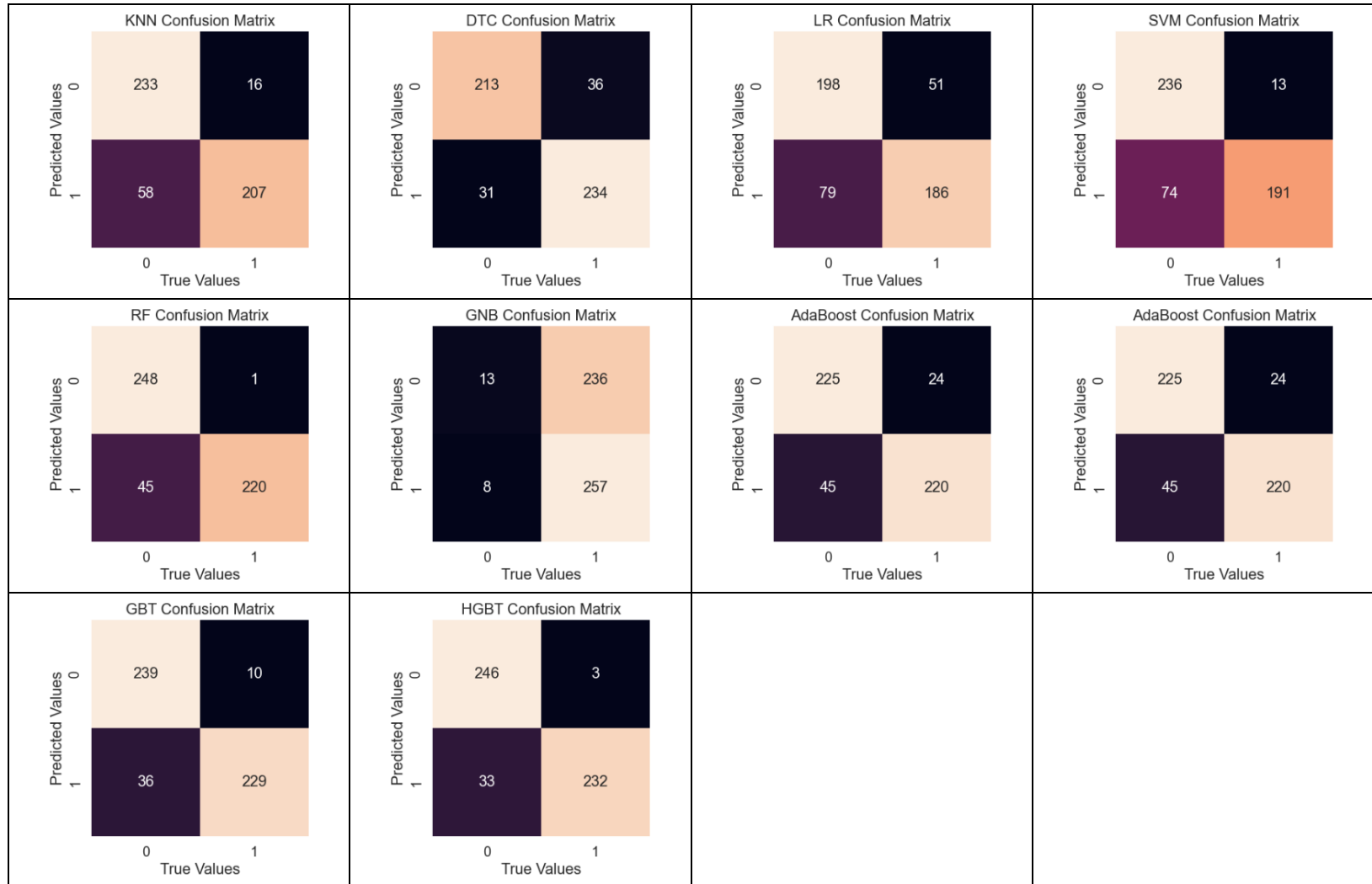
Dataset	Oliveira			
Tuning	Tuned			
Model	GBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9917	0.5289	0.6899	225
1 - Malicious	0.9878	0.9999	0.9938	8551
Accuracy			0.9878	8776
Macro-Avg	0.9897	0.7644	0.8418	8776
Weighted-Avg	0.9879	0.9878	0.9860	8776

Dataset	Oliveira			
Tuning	Default			
Model	HGBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9408	0.6356	0.7586	225
1 - Malicious	0.9905	0.9989	0.9947	8551
Accuracy			0.9896	8776
Macro-Avg	0.9656	0.8173	0.8767	8776
Weighted-Avg	0.9892	0.9896	0.9886	8776

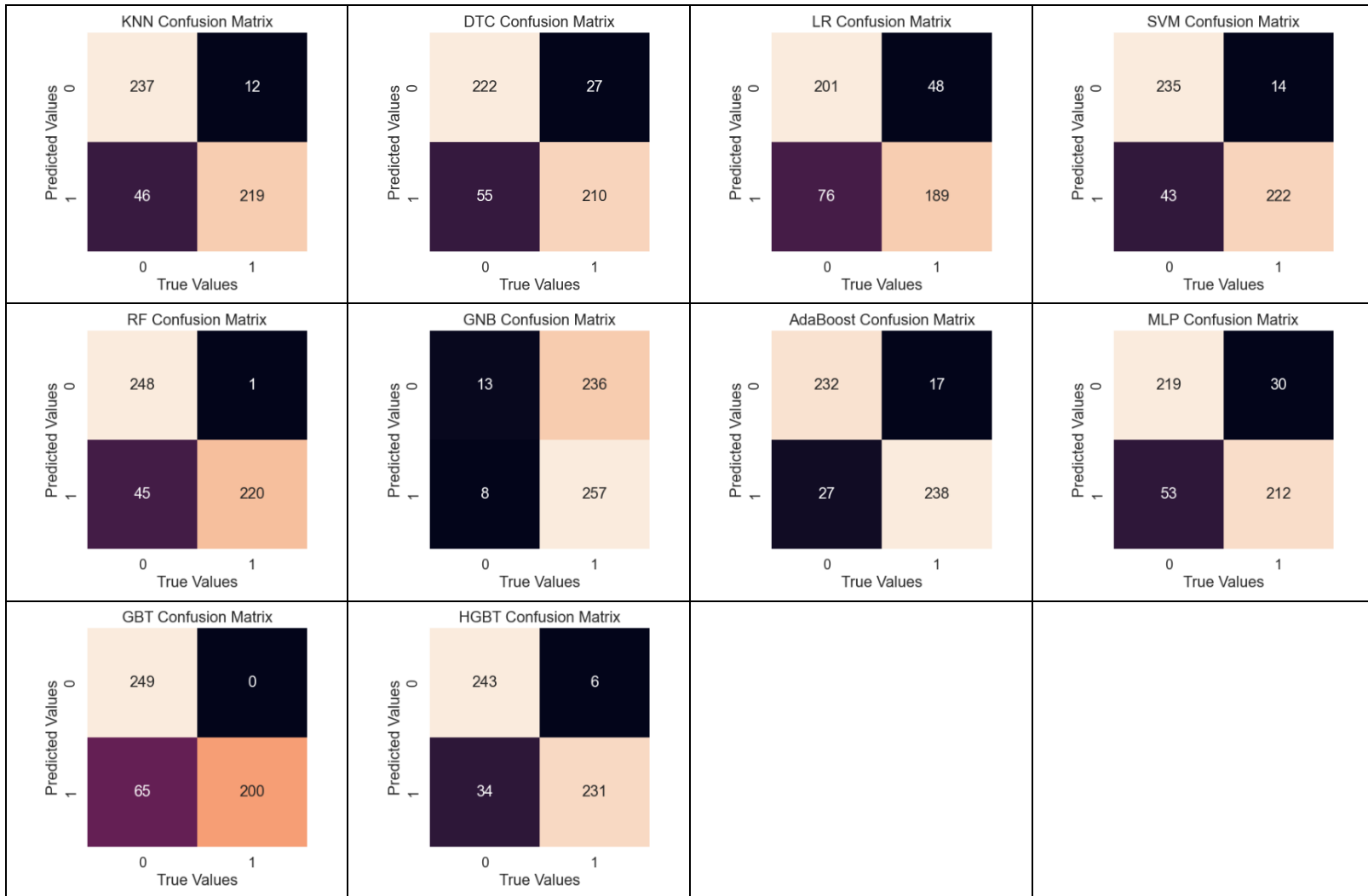
Dataset	Oliveira			
Tuning	Tuned			
Model	HGBT			
Label	Precision	Recall	F1-Score	Support
0 - Non-Malicious	0.9448	0.6089	0.7405	225
1 - Malicious	0.9898	0.9991	0.9944	8551
Accuracy			0.9891	8776
Macro-Avg	0.9673	0.8040	0.8675	8776
Weighted-Avg	0.9887	0.9891	0.9879	8776

11.3.Confusion Matrices (MalbehavD-V1)

11.3.1.Default

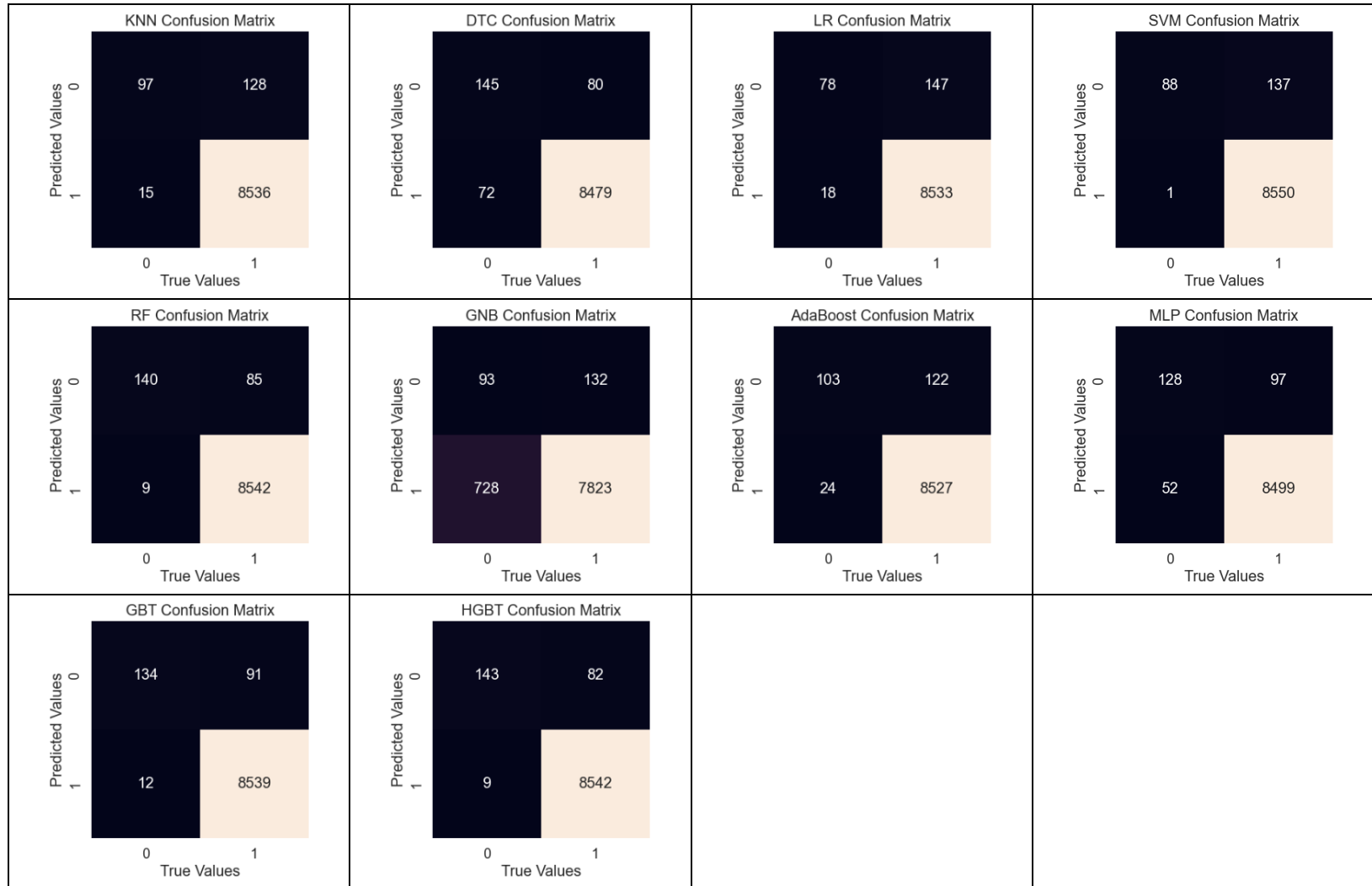


11.3.2. Tuned

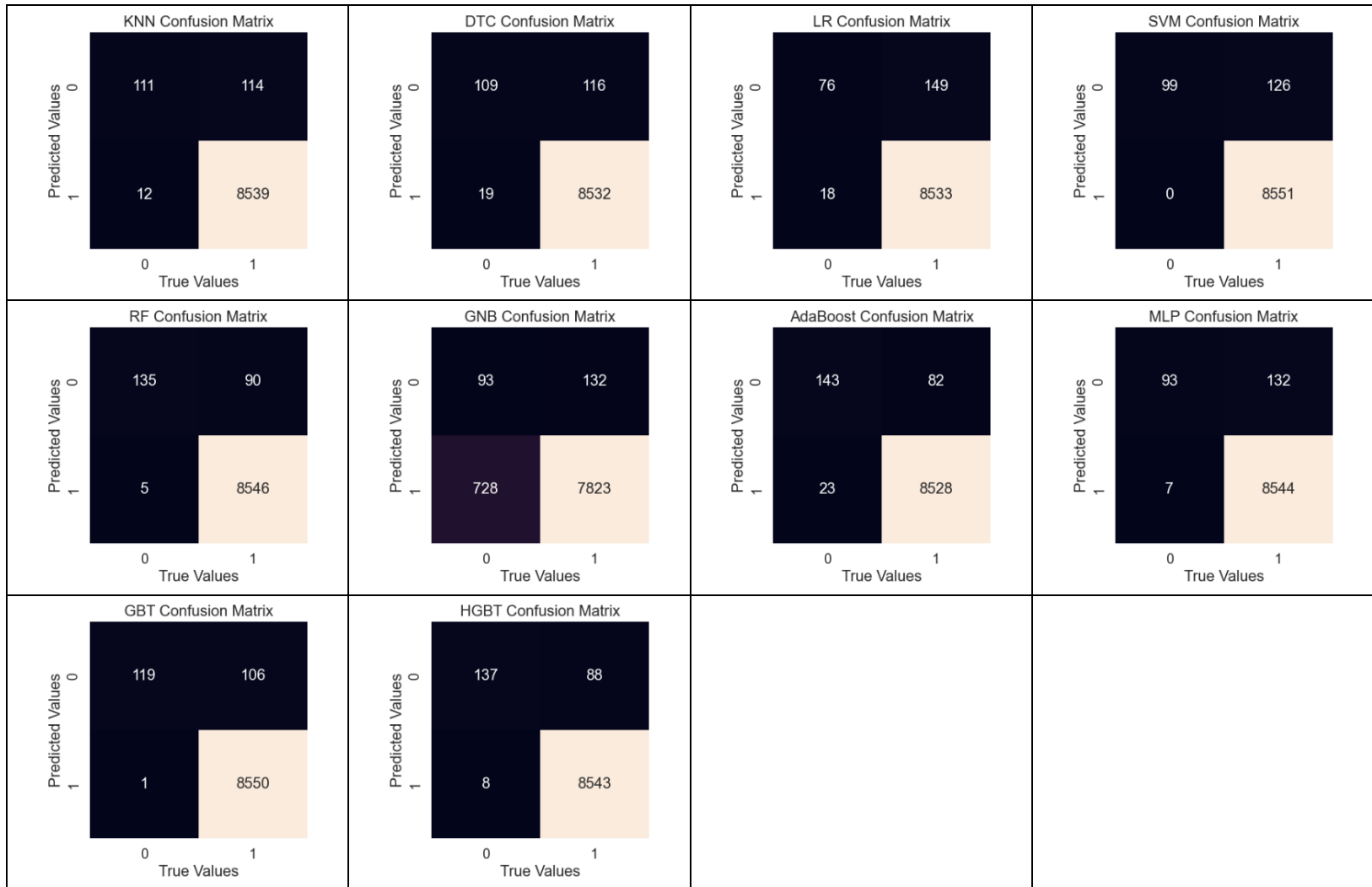


11.4. Confusion Matrices (Oliveira)

11.4.1. Default

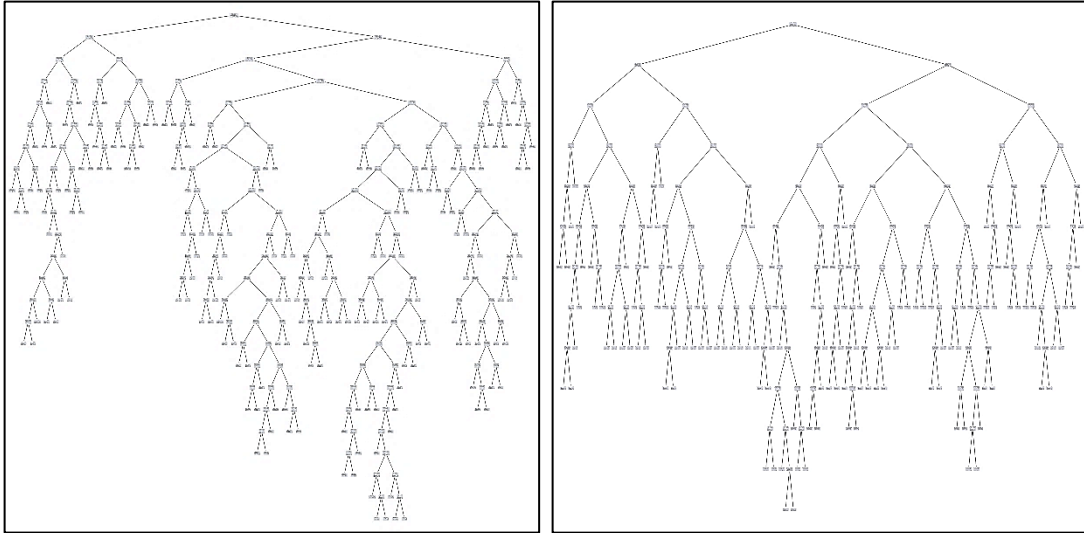


11.4.2. Tuned

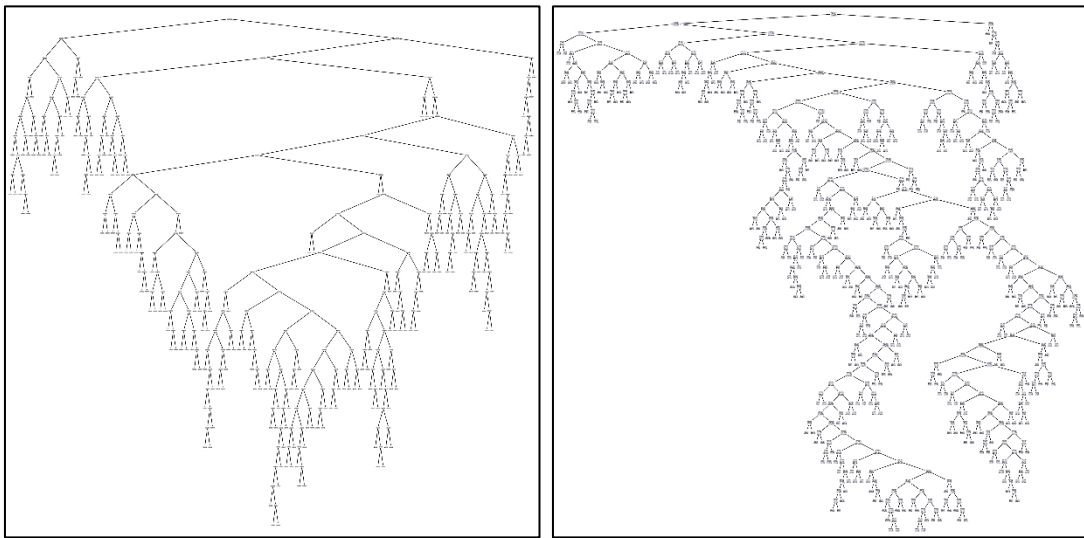


11.5. Decision Trees

11.5.1. MalbehavD-V1 - Default (Left), Tuned (Right)



11.5.2. Oliveira - Default (Left), Tuned (Right)



11.6.Dataset Performance

11.6.1.MalbehavD-V1: Default

MalbehavD-V1: Default					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.8556	0.8553	0.8918	0.8101	0.8556
DTC	0.8821	0.8820	0.8722	0.8965	0.8821
LR	0.7451	0.7449	0.7498	0.7369	0.7451
SVM	0.8634	0.8627	0.9214	0.7954	0.8635
RF	0.9257	0.9254	0.9826	0.8670	0.9257
GNB	0.5210	0.3839	0.8348	0.1412	0.5214
AdaB	0.8560	0.8559	0.8708	0.8374	0.8561
MLP	0.8576	0.8572	0.8476	0.8763	0.8576
GBT	0.9035	0.9034	0.9337	0.8693	0.9035
HGBT	0.9300	0.9299	0.9565	0.9012	0.9300
AVE	0.8340	0.8201	0.8861	0.7731	0.8341

11.6.2.MalbehavD-V1: Tuned

MalbehavD-V1: Tuned					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.8895	0.8894	0.9124	0.8623	0.8895
DTC	0.8233	0.8232	0.8373	0.8040	0.8234
LR	0.7475	0.7474	0.7495	0.7439	0.7475
SVM	0.8864	0.8862	0.9076	0.8615	0.8864
RF	0.9265	0.9261	0.9885	0.8631	0.9265
GNB	0.5514	0.4652	0.6796	0.2640	0.5516
AdaB	0.9062	0.9062	0.9102	0.9020	0.9063
MLP	0.8537	0.8536	0.8541	0.8553	0.8537
GBT	0.9163	0.9159	0.9866	0.8444	0.9164
HGBT	0.9288	0.9287	0.9563	0.8988	0.9288
AVE	0.8430	0.8342	0.8782	0.7899	0.8430

11.6.3.MalbehavD-V1: Improvement

MalbehavD-V1: Improvement					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	3.96%	3.99%	2.31%	6.44%	3.96%
DTC	-6.67%	-6.67%	-4.00%	-10.32%	-6.65%
LR	0.32%	0.34%	-0.04%	0.95%	0.32%
SVM	2.66%	2.72%	-1.50%	8.31%	2.65%
RF	0.09%	0.08%	0.60%	-0.45%	0.09%
GNB	5.83%	21.18%	-18.59%	86.97%	5.79%
AdaB	5.86%	5.88%	4.52%	7.71%	5.86%
MLP	-0.45%	-0.42%	0.77%	-2.40%	-0.45%
GBT	1.42%	1.38%	5.67%	-2.86%	1.43%
HGBT	-0.13%	-0.13%	-0.02%	-0.27%	-0.13%
AVE	1.29%	2.83%	-1.03%	9.41%	1.29%

11.6.4.Oliveira: Default

Oliveira: Default					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	0.9855	0.9830	0.9863	0.9991	0.7238
DTC	0.9849	0.9848	0.9920	0.9924	0.8382
LR	0.9831	0.9804	0.9854	0.9976	0.7045
SVM	0.9856	0.9827	0.9855	0.9999	0.7090
RF	0.9899	0.9889	0.9907	0.9991	0.8128
GNB	0.8948	0.9252	0.9860	0.9050	0.6981
AdaB	0.9845	0.9827	0.9874	0.9968	0.7463
MLP	0.9843	0.9829	0.9886	0.9954	0.7697
GBT	0.9883	0.9871	0.9900	0.9981	0.7989
HGBT	0.9901	0.9892	0.9912	0.9987	0.8228
AVE	0.9771	0.9787	0.9883	0.9882	0.7624

11.6.5.Oliveira: Tuned

Oliveira: Tuned					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	0.9861	0.9841	0.9874	0.9984	0.7476
DTC	0.9849	0.9832	0.9877	0.9969	0.7528
LR	0.9832	0.9805	0.9854	0.9976	0.7046
SVM	0.9863	0.9837	0.9863	0.9998	0.7242
RF	0.9895	0.9883	0.99	0.9993	0.8004
GNB	0.8948	0.9252	0.986	0.905	0.6981
AdaB	0.9868	0.986	0.9905	0.996	0.8089
MLP	0.9841	0.9819	0.9867	0.9972	0.7312
GBT	0.9879	0.9864	0.9886	0.9991	0.7716
HGBT	0.9894	0.9884	0.9906	0.9986	0.812
AVE	0.9773	0.9788	0.9879	0.9888	0.7551

11.6.6.Oliveira: Improvement

Oliveira: Improvement					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	0.06%	0.11%	0.11%	-0.07%	3.29%
DTC	0.00%	-0.16%	-0.43%	0.45%	-10.19%
LR	0.01%	0.01%	0.00%	0.00%	0.01%
SVM	0.07%	0.10%	0.08%	-0.01%	2.14%
RF	-0.04%	-0.06%	-0.07%	0.02%	-1.53%
GNB	0.00%	0.00%	0.00%	0.00%	0.00%
AdaB	0.23%	0.34%	0.31%	-0.08%	8.39%
MLP	-0.02%	-0.10%	-0.19%	0.18%	-5.00%
GBT	-0.04%	-0.07%	-0.14%	0.10%	-3.42%
HGBT	-0.07%	-0.08%	-0.06%	-0.01%	-1.31%
AVE	0.02%	0.01%	-0.04%	0.06%	-0.76%

11.6.7.MalbehavD-V1 [-] vs Oliveira [+] – Defaults

MalbehavD-V1 [-] vs Oliveira [+] - Defaults					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	15%	15%	11%	23%	-15%
DTC	12%	12%	14%	11%	-5%
LR	32%	32%	31%	35%	-5%
SVM	14%	14%	7%	26%	-18%
RF	7%	7%	1%	15%	-12%
GNB	72%	141%	18%	541%	34%
AdaB	15%	15%	13%	19%	-13%
MLP	15%	15%	17%	14%	-10%
GBT	9%	9%	6%	15%	-12%
HGBT	6%	6%	4%	11%	-12%
AVE	19.72%	26.51%	12.13%	70.95%	-6.82%

11.6.8.MalbehavD-V1 [-] vs Oliveira [+] – Defaults (Interpreted)

MalbehavD-V1 [-] vs Oliveira [+] - Defaults					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
DTC	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
LR	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
SVM	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
RF	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
GNB	Oliveira	Oliveira	Oliveira	Oliveira	Oliveira
AdaB	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
MLP	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
GBT	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
HGBT	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
AVE	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1

11.6.9. MalbehavD-V1 [-] vs Oliveira [+] – Tuned

MalbehavD-V1 [-] vs Oliveira [+] - Tuned					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	11%	11%	8%	16%	-16%
DTC	20%	19%	18%	24%	-9%
LR	32%	31%	31%	34%	-6%
SVM	11%	11%	9%	16%	-18%
RF	7%	7%	0%	16%	-14%
GNB	62%	99%	45%	243%	27%
AdaB	9%	9%	9%	10%	-11%
MLP	15%	15%	16%	17%	-14%
GBT	8%	8%	0%	18%	-16%
HGBT	7%	6%	4%	11%	-13%
AVE	18.09%	21.58%	13.97%	40.50%	-8.91%

11.6.10. MalbehavD-V1 [-] vs Oliveira [+] – Tuned (Interpreted)

MalbehavD-V1 [-] vs Oliveira [+] - Tuned					
Metric	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
DTC	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
LR	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
SVM	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
RF	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
GNB	Oliveira	Oliveira	Oliveira	Oliveira	Oliveira
AdaB	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
MLP	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
GBT	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
HGBT	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1
AVE	Oliveira	Oliveira	Oliveira	Oliveira	MalbehavD-V1

11.7. Time Data

MalbehavD-V1: Training Speed (s)			
Model	Default	Tuned	Increase
KNN	0.0113	0.0096	-15%
DTC	0.1260	0.0387	-69%
LR	1.8446	27.7076	1402%
SVM	0.3291	0.4086	24%
RF	0.4185	2.0821	398%
GNB	0.0134	0.0157	17%
AdaB	0.6074	6.0160	890%
MLP	3.4593	10.7093	210%
GBT	2.1067	0.9856	-53%
HGBT	7.7572	8.7912	13%
AVE	1.6674	5.6764	282%

Oliveira: Training Speed (s)			
Model	Default	Tuned	Increase
KNN	0.0357	0.0351	-2%
DTC	3.9522	0.0865	-98%
LR	10.7791	7.1334	-34%
SVM	42.0474	281.7550	570%
RF	5.0848	5.6513	11%
GNB	0.0825	0.0854	4%
AdaB	6.8824	65.8052	856%
MLP	31.2435	4.9668	-84%
GBT	24.9020	629.1891	2427%
HGBT	4.7686	3.5706	-25%
AVE	12.9778	99.8278	363%

MalbehavD-V1: Prediction Speed (ms)				Oliveira: Prediction Speed (ms)			
Model	Default	Tuned	Increase	Model	Default	Tuned	Increase
KNN	0.0243	0.0182	-25%	KNN	0.1650	0.164	-1%
DTC	0.0032	0.0024	-25%	DTC	0.0001	0.0002	100%
LR	0.0012	0.0017	42%	LR	0.0001	0.0001	0%
SVM	0.0269	0.0087	-68%	SVM	0.0500	0.2779	456%
RF	0.0195	0.0815	318%	RF	0.0017	0.0017	0%
GNB	0.0021	0.0021	0%	GNB	0.0003	0.0003	0%
AdaB	0.0063	0.0567	800%	AdaB	0.0025	0.024	860%
MLP	0.0016	0.0028	75%	MLP	0.0002	0.0003	50%
GBT	0.0015	0.0028	87%	GBT	0.0002	0.003	1400%
HGBT	0.0049	0.0056	14%	HGBT	0.0005	0.0004	-20%
AVE	0.0092	0.0183	122%	AVE	0.0221	0.0472	285%

11.8. Robustness

Reshape:	Trim				
Train:	MalbehavD-V1				
Test:	Oliveira				
Tuning:	Default				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.4946	0.3356	0.4972	0.9837	0.4946
DTC	0.5479	0.5275	0.5338	0.7556	0.5479
LR	0.4751	0.3753	0.4862	0.8747	0.4751
SVM	0.5008	0.3351	0.5004	1.0000	0.5008
RF	0.5879	0.5760	0.5659	0.7556	0.5879
GNB	0.4860	0.3284	0.4929	0.9704	0.4860
AdaB	0.4840	0.4020	0.4908	0.8545	0.4840
MLP	0.5397	0.5275	0.5585	0.3790	0.5397
GBT	0.4782	0.3695	0.3716	0.0630	0.4782
HGBT	0.4802	0.4580	0.4859	0.6825	0.4802

Reshape:	Trim				
Train:	MalbehavD-V1				
Test:	Oliveira				
Tuning:	Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.7062	0.8073	0.9730	0.7188	0.4632
DTC	0.7750	0.8521	0.9745	0.7899	0.4858
LR	0.1796	0.2745	0.9716	0.1637	0.4869
SVM	0.7403	0.8299	0.9767	0.7517	0.5199
RF	0.9084	0.9300	0.9768	0.9281	0.5276
GNB	0.2343	0.3502	0.9837	0.2186	0.5375
AdaB	0.8356	0.8897	0.9824	0.8466	0.6226
MLP	0.3391	0.4821	0.9793	0.3294	0.5266
GBT	0.7935	0.8637	0.9766	0.8077	0.5206
HGBT	0.9290	0.9411	0.9768	0.9498	0.5282

Reshape:	Trim				
Train:	MalbehavD-V1				
Test:	Oliveira				
Tuning:	Default/Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	43%	141%	96%	-27%	-6%
DTC	41%	62%	83%	5%	-11%
LR	-62%	-27%	100%	-81%	2%
SVM	48%	148%	95%	-25%	4%
RF	55%	61%	73%	23%	-10%
GNB	-52%	7%	100%	-77%	11%
AdaB	73%	121%	100%	-1%	29%
MLP	-37%	-9%	75%	-13%	-2%
GBT	66%	134%	163%	1182%	9%
HGBT	93%	105%	101%	39%	10%

Reshape:	Trim				
Train:	MalbehavD-V1				
Test:	Oliveira				
	Changes (%) from Testing on MalbehavD-V1				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-73%	-155%	-79%	18%	-73%
DTC	-61%	-67%	-63%	-19%	-61%
LR	-57%	-98%	-54%	16%	-57%
SVM	-72%	-157%	-84%	20%	-72%
RF	-57%	-61%	-74%	-15%	-57%
GNB	-7%	-17%	-69%	85%	-7%
AdaB	-77%	-113%	-77%	2%	-77%
MLP	-59%	-63%	-52%	-131%	-59%
GBT	-89%	-144%	-151%	-1280%	-89%
HGBT	-94%	-103%	-97%	-32%	-94%
Average	-65%	-98%	-80%	-134%	-65%

Reshape:	Trim				
Train:	MalbehavD-V1				
Test:	Oliveira				
	Tuned - Changes (%) from Testing on MalbehavD-V1				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-26%	-10%	6%	-20%	-92%
DTC	-6%	3%	14%	-2%	-69%
LR	-316%	-172%	23%	-354%	-54%
SVM	-20%	-7%	7%	-15%	-70%
RF	-2%	0%	-1%	7%	-76%
GNB	-135%	-33%	31%	-21%	-3%
AdaB	-8%	-2%	7%	-7%	-46%
MLP	-152%	-77%	13%	-160%	-62%
GBT	-15%	-6%	-1%	-5%	-76%
HGBT	0%	1%	2%	5%	-76%
Average	-68%	-30%	10%	-57%	-62%

Reshape:	Maximize				
Train:	MalbehavD-V1				
Test:	Oliveira				
Tuning:	Default				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.8338	0.8876	0.9744	0.8520	0.4825
DTC	0.4996	0.6465	0.9708	0.5021	0.4512
LR	0.2670	0.3961	0.9725	0.2557	0.4842
SVM	0.9626	0.9579	0.9759	0.9861	0.5102
RF	0.8655	0.9058	0.9751	0.8847	0.4943
GNB	0.0831	0.1136	0.9798	0.0612	0.5056
AdaB	0.8897	0.9189	0.9744	0.9109	0.4805
MLP	0.5390	0.6808	0.9704	0.5439	0.4429
GBT	0.8678	0.9072	0.9757	0.8865	0.5058
HGBT	0.9078	0.9292	0.9756	0.9287	0.5046

Reshape:	Maximize				
Train:	MalbehavD-V1				
Test:	Oliveira				
Tuning:	Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.8517	0.8980	0.9754	0.8699	0.5003
DTC	0.4347	0.5836	0.9803	0.4291	0.5436
LR	0.3244	0.4653	0.9783	0.3143	0.5191
SVM	0.8461	0.8952	0.9779	0.8616	0.5448
RF	0.9520	0.9536	0.9772	0.9735	0.5359
GNB	0.2261	0.3388	0.9841	0.2099	0.5378
AdaB	0.5893	0.7206	0.9806	0.5906	0.5632
MLP	0.5432	0.6826	0.9817	0.5418	0.5703
GBT	0.6428	0.7629	0.9686	0.6550	0.4068
HGBT	0.9174	0.9345	0.9759	0.9385	0.5105

Reshape:	Maximize				
Train:	MalbehavD-V1				
Test:	Oliveira				
Tuning:	Default/Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	2%	1%	0%	2%	4%
DTC	-13%	-10%	1%	-15%	20%
LR	21%	17%	1%	23%	7%
SVM	-12%	-7%	0%	-13%	7%
RF	10%	5%	0%	10%	8%
GNB	172%	198%	0%	243%	6%
AdaB	-34%	-22%	1%	-35%	17%
MLP	1%	0%	1%	0%	29%
GBT	-26%	-16%	-1%	-26%	-20%
HGBT	1%	1%	0%	1%	1%

Reshape:	Maximize				
Train:	MalbehavD-V1				
Test:	Oliveira				
	Tuned - Changes (%) from Testing on MalbehavD-V1				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-3%	4%	8%	5%	-77%
DTC	-77%	-36%	10%	-79%	-96%
LR	-179%	-88%	23%	-188%	-54%
SVM	10%	10%	6%	19%	-69%
RF	-7%	-2%	-1%	2%	-87%
GNB	-527%	-238%	15%	-131%	-3%
AdaB	4%	7%	11%	8%	-78%
MLP	-59%	-26%	13%	-61%	-94%
GBT	-4%	0%	4%	2%	-79%
HGBT	-2%	0%	2%	3%	-84%
Average	-84%	-37%	9%	-42%	-72%

Reshape:	Maximize				
Train:	MalbehavD-V1				
Test:	Oliveira				
	Default - Changes (%) from Testing on MalbehavD-V1				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-4%	1%	6%	1%	-78%
DTC	-89%	-41%	15%	-87%	-51%
LR	-130%	-61%	23%	-137%	-44%
SVM	-5%	1%	7%	0%	-63%
RF	3%	3%	-1%	11%	-73%
GNB	-144%	-37%	31%	-26%	-3%
AdaB	-54%	-26%	7%	-53%	-61%
MLP	-57%	-25%	13%	-58%	-50%
GBT	-43%	-20%	-2%	-29%	-125%
HGBT	-1%	1%	2%	4%	-82%
Average	-52%	-20%	10%	-37%	-63%

Reshape:	Trim				
Train:	Oliveira				
Test:	MalbehavD-V1				
Tuning:	Default				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.4946	0.3356	0.4972	0.9837	0.4946
DTC	0.5479	0.5275	0.5338	0.7556	0.5479
LR	0.4751	0.3753	0.4862	0.8747	0.4751
SVM	0.5008	0.3351	0.5004	1.0000	0.5008
RF	0.5879	0.5760	0.5659	0.7556	0.5879
GNB	0.4860	0.3284	0.4929	0.9704	0.4860
AdaB	0.4840	0.4020	0.4908	0.8545	0.4840
MLP	0.5397	0.5275	0.5585	0.3790	0.5397
GBT	0.4782	0.3695	0.3716	0.0630	0.4782
HGBT	0.4802	0.4580	0.4859	0.6825	0.4802

Reshape:	Trim				
Train:	Oliveira				
Test:	MalbehavD-V1				
Tuning:	Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	0.4957	0.3627	0.4978	0.9525	0.4957
DTC	0.5533	0.5347	0.5381	0.7533	0.5533
LR	0.4743	0.3692	0.4859	0.8825	0.4743
SVM	0.5	0.3333	0.5	1	0.5
RF	0.4988	0.3376	0.4994	0.9922	0.4988
GNB	0.486	0.3284	0.4929	0.9704	0.486
AdaB	0.4074	0.3917	0.3635	0.2467	0.4074
MLP	0.4506	0.4324	0.4636	0.6296	0.4506
GBT	0.4844	0.3896	0.4913	0.8786	0.4844
HGBT	0.4576	0.4149	0.4725	0.7276	0.4576

Reshape:	Trim				
Train:	Oliveira				
Test:	MalbehavD-V1				
Tuning:	Default/Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC_AUC
KNN	0%	8%	0%	-3%	0%
DTC	1%	1%	1%	0%	1%
LR	0%	-2%	0%	1%	0%
SVM	0%	-1%	0%	0%	0%
RF	-15%	-41%	-12%	31%	-15%
GNB	0%	0%	0%	0%	0%
AdaB	-16%	-3%	-26%	-71%	-16%
MLP	-17%	-18%	-17%	66%	-17%
GBT	1%	5%	32%	1295%	1%
HGBT	-5%	-9%	-3%	7%	-5%

Reshape:	Trim				
Train:	Oliveira				
Test:	MalbehavD-V1				
	Changes (%) from Testing on Oliveira				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-99%	-193%	-98%	-2%	-46%
DTC	-80%	-87%	-86%	-31%	-53%
LR	-107%	-161%	-103%	-14%	-48%
SVM	-97%	-193%	-97%	0%	-42%
RF	-68%	-72%	-75%	-32%	-38%
GNB	-84%	-182%	-100%	7%	-44%
AdaB	-103%	-144%	-101%	-17%	-54%
MLP	-82%	-86%	-77%	-163%	-43%
GBT	-107%	-167%	-166%	-1484%	-67%
HGBT	-106%	-116%	-104%	-46%	-71%
Average	-93%	-140%	-101%	-178%	-51%

Reshape:	Trim				
Train:	Oliveira				
Test:	MalbehavD-V1				
	Changes (%) from Testing on Oliveira				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-99%	-171%	-98%	-5%	-51%
DTC	-78%	-84%	-84%	-32%	-36%
LR	-107%	-166%	-103%	-13%	-49%
SVM	-97%	-195%	-97%	0%	-45%
RF	-98%	-193%	-98%	-1%	-60%
GNB	-84%	-182%	-100%	7%	-44%
AdaB	-142%	-152%	-172%	-304%	-99%
MLP	-118%	-127%	-113%	-58%	-62%
GBT	-104%	-153%	-101%	-14%	-59%
HGBT	-116%	-138%	-110%	-37%	-77%
Average	-104%	-156%	-108%	-46%	-58%

Reshape:	Maximize				
Train:	Oliveira				
Test:	MalbehavD-V1				
Tuning:	Default				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.4946	0.3356	0.4972	0.9837	0.4946
DTC	0.4665	0.4604	0.4574	0.3595	0.4665
LR	0.4751	0.3753	0.4862	0.8747	0.4751
SVM	0.5	0.3333	0.5	1	0.5
RF	0.5144	0.4522	0.5086	0.8514	0.5144
GNB	0.486	0.3284	0.4929	0.9704	0.486
AdaB	0.484	0.402	0.4908	0.8545	0.484
MLP	0.5342	0.5342	0.5338	0.5409	0.5342
GBT	0.4747	0.3646	0.3488	0.0584	0.4747
HGBT	0.4802	0.458	0.4859	0.6825	0.4802

Reshape:	Maximize				
Train:	Oliveira				
Test:	MalbehavD-V1				
Tuning:	Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	0.8517	0.8980	0.9754	0.8699	0.5003
DTC	0.5939	0.7253	0.9727	0.6006	0.4657
LR	0.3262	0.4675	0.9782	0.3163	0.5182
SVM	0.0246	0.0012	0.0000	0.0000	0.5000
RF	0.9265	0.9397	0.9767	0.9472	0.5255
GNB	0.4845	0.6307	0.9817	0.4805	0.5627
AdaB	0.9220	0.9366	0.9753	0.9440	0.4975
MLP	0.4969	0.6440	0.9713	0.4989	0.4575
GBT	0.8157	0.8769	0.9750	0.8324	0.4922
HGBT	0.8916	0.9202	0.9750	0.9123	0.4923

Reshape:	Maximize				
Train:	Oliveira				
Test:	MalbehavD-V1				
Tuning:	Default/Tuned				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	72%	168%	96%	-12%	1%
DTC	27%	58%	113%	67%	0%
LR	-31%	25%	101%	-64%	9%
SVM	-95%	-100%	-100%	-100%	0%
RF	80%	108%	92%	11%	2%
GNB	0%	92%	99%	-50%	16%
AdaB	90%	133%	99%	10%	3%
MLP	-7%	21%	82%	-8%	-14%
GBT	72%	141%	180%	1325%	4%
HGBT	86%	101%	101%	34%	3%

Reshape:	Maximize				
Train:	Oliveira				
Test:	MalbehavD-V1				
	Changes (%) from Testing on Oliveira				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-99%	-193%	-98%	-2%	-46%
DTC	-111%	-114%	-117%	-176%	-80%
LR	-107%	-161%	-103%	-14%	-48%
SVM	-97%	-195%	-97%	0%	-42%
RF	-92%	-119%	-95%	-17%	-58%
GNB	-84%	-182%	-100%	7%	-44%
AdaB	-103%	-144%	-101%	-17%	-54%
MLP	-84%	-84%	-85%	-84%	-44%
GBT	-108%	-171%	-184%	-1609%	-68%
HGBT	-106%	-116%	-104%	-46%	-71%
Average	-99%	-148%	-108%	-196%	-56%

Reshape:	Maximize				
Train:	Oliveira				
Test:	MalbehavD-V1				
	Tuned - Changes (%) from Testing on Oliveira				
Model	Accuracy	F1-Score (Weighted)	Precision	Recall	ROC AUC
KNN	-16%	-10%	-1%	-15%	-49%
DTC	-66%	-36%	-2%	-66%	-62%
LR	-201%	-110%	-1%	-215%	-36%
SVM	-3909%	-81875%	#DIV/0!	#DIV/0!	-45%
RF	-7%	-5%	-1%	-6%	-52%
GNB	-85%	-47%	0%	-88%	-24%
AdaB	-7%	-5%	-2%	-6%	-63%
MLP	-98%	-52%	-2%	-100%	-60%
GBT	-21%	-12%	-1%	-20%	-57%
HGBT	-11%	-7%	-2%	-9%	-65%
Average	-442%	-8216%	-1%	-52%	-51%

11.9. Tuning Time (s)

Model	MalbehavD-V1	Oliveira
KNN	8.4275	315.1593
DTC	0.4961	9.3904
LR	63.9053	12.7848
SVM	10.9501	589.0862
RF	41.2800	477.9451
GNB	0.3784	1.2072
AdaB	22.8574	273.6070
MLP	28.3705	186.6005
GBT	12.1466	1418.0289
HGBT	136.7645	50.2759

11.10. System Specifications

Component	Specification	Brand/Model
CPU	2.1Ghz (3.7Ghz Boost) 4 Cores 8 Threads	AMD Ryzen 5 3500U
RAM	12GB (9.9GB Usable)	N/A
Storage	1TB 5400RPM	Western Digital