INTRUSION DETECTION SYSTEMS USING MACHINE LEARNING

An internship report

Sajal Saxena

INDEX

1.) INTRODUCTION

As technology advances, our computers have become increasingly more complex. They are capable to perform a lot more high-performance computational tasks than they could do a few years ago. With advancement and wider reach of internet services and with the help of Industry 4.0, technology has become much more accessible to people and has become easier to use. Manual tasks have made way for automation in many organizations, even those which traditionally relied on manual work.

As technology improves, it brings along itself various threats which are needed to be dealt with. These threats can be in the form of cyber-attacks and intrusion. Intrusion refers to an unauthorized access to a network or system by an intruder (A person(s)/software that has malicious intentions) which aims to access sensitive information and/or destabilize the network/system. Intrusion Detection Systems (IDS) are used for detection of intrusion in a system or a network. It scans the system/network and looks for malicious activities (activities that are not like what the system considers "Normal") and reports them to the administrator.

With the speed at which technology is advancing, it is becoming increasingly difficult to keep up with Next Generation Attacks. Zero-day attacks and stealth attacks are becoming the main type of attack strategies used by hackers and are very difficult to detect for the existing systems. Another part of the problem with intrusion detection systems is the lack of publicly available benchmark datasets as obtaining them is difficult due to privacy concerns and artificially constructing them may not capture true essence of NGAs. This makes training advanced IDS models difficult as there is a potential for missing out on data with anomalous properties.

Machine Learning and Artificial Intelligence are being used to create more efficient IDS.

As part of the internship, implementation of Intrusion Detection Systems (IDS) was performed. Intrusion Detection Systems can be classified into two types based on the techniques used to identify intrusion:

A.) Signature based IDS
B.) Anomaly based IDS

Signature based IDS detects intrusion by detecting an attack pattern in the system or network and comparing it with the existing dictionary ('signature') of attack patterns which have been encountered earlier. This makes detection quick and accurate if the attack pattern has occurred before, as the system remembers it and knows how to identify it. But this type of IDS finds it difficult to identify new types of attack, hence it is not very robust.

Anomaly based IDS detects intrusion by learning the characteristics of a 'Normal' pattern and any new pattern which has characteristics different from what the model considers normal, is considered an 'Anomaly'. This type of IDS is more robust in detecting Next Generation of Attacks (NGA) and hence, more suitable for modern systems.

Intrusion Detection Systems can also be classified into:

A.) Network-based IDS (NIDS)

B.) Host-based IDS (HIDS)

Network-based IDS are used to detect intrusion within an entire network. This includes scanning the entire network, usually by processing the information obtained from packet metadata and contents. They have visibility of the traffic flowing through the entire network and can make decisions based on the data from the whole network.

Host-based IDS are used to detect intrusion within a system(host). This includes scanning through the network traffic flowing through and from the machine and inspecting system's processes like System Calls or System Logs. NIDS have their visibility limited only to the machine which may reduce its context for decision-making, but they have deeper insights into the machine's internals.

The work performed during the internship included implementation of a NIDS and an HIDS using machine learning techniques. The implementation of NIDS is based on [1] and the implementation of HIDS is based on [2].

The datasets used are NSL-KDD (for NIDS) and ADFA-LD (for HIDS).

2.) LITERATURE REVIEW

[1] suggested that data needs to be filtered using a Vote algorithm with Information Gain that combines the probability distributions of these base learners to select the important features that positively affect the accuracy of the proposed model. They achieved DR of 99.81% and FAR of 0.25%.

System calls were used for intrusion detection in Integer Data Zero-Watermark Assisted System Calls Abstraction and Normalization for Host Based Anomaly Detection Systems by Haider, et al. [2]. They used system call data to create 6 feature vectors and combining them to capture hidden representations and applied normalizations to those representations. Some of the normalized vectors with neural networks gave a DR of 95% and FAR of 8%.

In [3], a similar integer system calls intrusion detection system was used with 4 feature vectors and combining them to capture hidden representations. It achieved DR of 78% and FAR of 21%.

[4] suggested use of LSTM-based model for system call language modelling. It generated probability distribution of system calls laying down what is to be considered "normal" system call sequence and if a given sequence had system calls with low probability as compared to what had been established from the distribution, it was classified as "attack".

[5] focused on reducing the problem of false alarm rate, using semantic based system call patterns by creating data dictionary every possible combination of sequence of system call names of phrase length. It achieved a DR of 86% and FAR of 2.2%.

[6] created ADFA dataset for Windows OS and suggested using a frequency distribution method and machine learning algorithms integrated with a novel DDLLC based feature construction methodology for intrusion detection. It achieved 72% DR and 12% FAR.

3.) DATASETS

3.1.) NSL-KDD:

The dataset was provided by the Canadian Institute of Cybersecurity based at University of New Brunswick at Fredericton. The NSL-KDD dataset was an improvement on the benchmark KDD'98 dataset. It had removed redundant datapoints and duplicates from KDD'98.

The dataset has a column 'difficulty_level' which is an integer between 1-21 stating how difficult it is to classify the observation. This feature was not considered in the training and evaluation of the model.

The dataset consists of a train dataset and a test dataset. But various subsets of the dataset are available on the UNB website, such as follows:

	DATASET	OBSERVATIONS	FEATURES
1	KDDTrain+	125973	43
2	KDDTest+	22544	43
3	KDDTrain_20Percent	25192	43
4	KDDTrain-21	11850	43

The distribution of features in the dataset is as follows:

Nominal	3
Binary	6
Numeric	32

Binary features have values in {0,1}. Nominal features have values as provided in tables in section (4.1.1).

The KDDTrain_20Percent dataset is a subset of KDDTrain+ file and has 25192 observations. It has 43 features including binary labels ('Normal' and 'Anomaly'). This dataset was used during training due to its low training time.

The following table shows distribution of normal and anomalous packets in different subsets of NSL-KDD:

	KDDTrain+_20Percent	KDDTrain+	KDDTest+
NORMAL	13449	67343	9711
ANOMALOUS	11743	58630	12829

To give an overview of the dataset, the following images (3.1.1 and 3.1.2) show top 5 and bottom 5 rows of KDDTrain+_20Percent dataset after removing the 'difficulty_level' column:

duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate
0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	0.03
0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	0.60
0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	0.05
0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	0.00
0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	tcp	exec	RSTO	0	0	0	0	0	0	...	7	0.03	0.06
0	tcp	ftp_data	SF	334	0	0	0	0	0	...	39	1.00	0.00
0	tcp	private	REJ	0	0	0	0	0	0	...	13	0.05	0.07
0	tcp	nnsp	S0	0	0	0	0	0	0	...	20	0.08	0.06
0	tcp	finger	S0	0	0	0	0	0	0	...	49	0.19	0.03

Image 3.1.1

dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	class
0.17	0.00	0.00	0.00	0.05	0.00	normal
0.88	0.00	0.00	0.00	0.00	0.00	normal
0.00	0.00	1.00	1.00	0.00	0.00	anomaly
0.03	0.04	0.03	0.01	0.00	0.01	normal
0.00	0.00	0.00	0.00	0.00	0.00	normal
...	...	...	...	...	...	...
0.00	0.00	0.00	0.00	1.00	1.00	anomaly
1.00	0.18	0.00	0.00	0.00	0.00	anomaly
0.00	0.00	0.00	0.00	1.00	1.00	anomaly
0.00	0.00	1.00	1.00	0.00	0.00	anomaly
0.01	0.00	1.00	1.00	0.00	0.00	anomaly

Image 3.1.2

The features capture multiple properties of each packet passing through the network when the dataset was created. It has properties like duration of packet flow through network, destination network service used, number of data bytes transferred from source to destination in single connection, etc.

The following image shows the dataset with 8 features selected in section (4.1.3) along with target variable 'class':

	service	flag	src_bytes	dst_bytes	same_srv_rate	diff_srv_rate	dst_host_srv_count	dst_host_same_srv_rate	class
0	ftp_data	SF	491	0	1.00	0.00	25	0.17	normal
1	other	SF	146	0	0.08	0.15	1	0.00	normal
2	private	S0	0	0	0.05	0.07	26	0.10	anomaly
3	http	SF	232	8153	1.00	0.00	255	1.00	normal
4	http	SF	199	420	1.00	0.00	255	1.00	normal
...	...	...	...	...	...	...	...	...	...
25187	exec	RSTO	0	0	0.07	0.07	7	0.03	anomaly
25188	ftp_data	SF	334	0	1.00	0.00	39	1.00	anomaly
25189	private	REJ	0	0	0.07	0.07	13	0.05	anomaly
25190	nnsp	S0	0	0	0.14	0.06	20	0.08	anomaly
25191	finger	S0	0	0	0.24	0.11	49	0.19	anomaly

3.2.) ADFA-LD:

The Australian Defense Force Academy-Linux Dataset consists of System Call traces which are labelled as 'Attack', 'Test' and 'Validation'. The system calls are obtained from Linux system. The attack traces consist of 6 attack classes:

1. Adduser
2. Hydra_FTP
3. Hydra_SSH
4. Java_Meterpreter
5. Meterpreter
6. Webshell

The distribution of the traces is given as follows:

NORMAL TRAINING	ATTACK	NORMAL VALIDATION
833	746	4372

Each trace consists of integer system calls ranging from 3 to 340. The following table shows some system calls with their names(source:[9]):

SYSTEM CALL	NAME
3	sys_read
4	sys_write
5	sys_open
88	sys_reboot
108	sys_fstat
150	sys_mlock
200	sys_getgid
320	sys_utimensat
340	sys_prlimit64

The following images show traces from ADFA-LD dataset:

```
54 175 120 175 175 3 175 175 120 175 120 175 120 175 175 120 175 3 3 3 175 120 175 175 175 7 3 3 175
120 175 7 175 7 119 174 54 3 3 175 175 3 120 175 175 120 175 120 120 175 175 54 140 3 175 120 175 175
175 175 175 174 7 175 7 119 3 3 175 3 175 175 120 175 7 175 3 175 120 175 175 54 7 174 3 175 120 7
175 175 120 175 175 3 175 120 175 3 3 120 175 120 175 175 7 54 175 120 175 7 175 7 119 174 54 3 120
175 175 120 54 3 120 175 175 54 140 175 175 174 54 175 120 175 175 54 140
```

Image 3.2.1

```
85 309 195 10 85 10 102 102 195 196 5 27 27 5 27 102 5 309 102 27 85 195 195 196 309 102 85 265 309
27 309 102 309 102 102 309 102 196 102 102 102 102 309 265 85 6 102 85 85 195 10 6 5 309 102 102 309
27 102 5 102 27 85 102 27 309 102 309 102 102 309 102 102 102 27 102 309 309 5 102 102 309 85 195 102
196 102 85 265 102 27 85 6 195 196 309 195 196 102 196 309 102 102 102 6 102 102 309 102 5 5 102 5
309 27 27 5 102 27 5 5 102 27 102 265 309 6 102 102 6 27 5 85 195 85 195 85 195 85 85 195 85 195 309
102 27 85 195 85 195 85 195 85 85 195 85 195 196 196 265 10 102 102 6 27 5 102 102 27 102 195 102 102
10 102 309 102 265 85 102 309 102 85 85 195 85 196 195 27 85 309 195 309 102 5 309 102 27 102 27 309
102 27 85 195 85 195 85 195 85 85 195 85 195 196 196 265 10 102 102 6 27 5 102 102 309 85 196 102 102
102 102 102 5 309 265 85 102 309 102 102 309 102 102 5 102 6 10 309 102 6 102 10 196 10 5 265 309 5
102 102 309 5 309 102 27 309 5 309 102 27 195 196 196 265 10 102 102 6 27 5 102 102 102 195 5 102 6
309 102 102 102 5 195 195 10 102 102 102 309
```

Image 3.2.2

Image 3.2.1 is a ‘Normal’ trace and Image 3.2.2 is an ‘Anomalous’ trace.

Some of the key differences between ‘Normal’ traces and ‘Anomalous’ traces are as follows:

	Normal Trace	**Anomalous Trace**
Number of traces	833	746
No of System calls in longest trace	2949	2713
Avg length of trace	369	425

The feature matrix FM10 obtained in section (4.2.2) is a 1579x11 matrix which includes class labels. The following image shows top 5 and bottom 5 rows of FM10:

	FV1	FV2	FV3	FV4	FV5	FV6	FV7	FV8	FV9	FV10	Label
468	3.0	311.0	174.0	11.0	62	90	125.236842	81.214681	152	21	1
625	3.0	243.0	221.0	243.0	68	29	152.989691	83.846648	97	16	0
391	4.0	265.0	240.0	265.0	458	697	115.270996	110.844420	1155	5	1
706	3.0	340.0	265.0	195.0	587	193	141.139744	107.556199	780	6	1
436	3.0	265.0	142.0	54.0	208	186	136.314721	83.094492	394	9	1
...	...	...	...	...	...	...	...	...	...	...	...
738	3.0	197.0	3.0	63.0	153	53	68.597087	82.922931	206	16	0
341	3.0	243.0	197.0	9.0	66	81	106.775510	82.732015	147	29	0
556	3.0	311.0	221.0	311.0	226	242	141.880342	85.887673	468	26	0
208	3.0	340.0	265.0	340.0	415	203	151.635922	104.237208	618	6	1
461	3.0	197.0	5.0	141.0	182	36	34.816514	56.963593	218	9	1

Class labels are binary labels 0 and 1 where 0 denotes 'Normal' trace and 1 denotes 'Anomalous' trace.

4.)METHODOLOGY

During training of both NIDS and HIDS, the following classifiers were found to provide most accurate results:

1. **LightGBM Classifier**: Light gradient boosting machine is a free and open source distributed gradient boosting framework for machine learning. It uses a leaf-wise tree growth strategy, where it grows the tree by expanding the leaf with the highest loss reduction. This can lead to faster convergence and improved performance.
2. **XGBoost Classifier**: Extreme gradient boosting is a gradient boosting algorithm and is an ensemble learning method that combines the predictions of multiple weak learners (usually decision trees) to create a strong predictive model. It is similar to LightGBM but differs in the way it expands the tree and also how it uses histogram based approach.
3. **Bagging Classifier**: A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. [7]
4. **Extra Trees Classifier**: It implements a meta estimator that fits several randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.[8]
5. **Random Forest Classifier**: It is a supervised machine learning algorithm that creates several decision trees and use voting mechanism to classify data provided to it.

The detailed methodology used for NIDS and HIDS has been discussed in sections 4.1 and 4.2 respectively.

4.1.) NIDS

4.1.1.) **Data preprocessing:**

- The subset of dataset used did not contain headers, so headers were added to the dataset.
- Nominal features 'Service', 'Flag' and 'Protocol type' and target feature 'Label' were converted to numeric labels. The following were the changes made:

FLAG	NUMERIC LABEL
SF	1
S0	2
REJ	3
RSTR	4
SH	5
RSTO	6
S1	7
RSTOS0	8
S3	9
S2	10
OTH	11

FLAG

PROTOCOL TYPE	NUMERIC LABEL
TCP	1
UDP	2
ICMP	3

PROTOCOL TYPE

LABEL	NUMERIC LABEL
Normal	0
Intrusion	1

CLASS

4.1.2.) **Data Normalization:**

- Data was normalized to bring the values in the range [0,1]. This was done using 'min_max_scaler' from scikitlearn's preprocessing library.

4.1.3) **Feature Selection:**

- Based on feature selection method described in [1], 8 features out of 42 were finally selected. The following table consists of information of those features:

FEATURE NUMBER	FEATURE NAME	FEATURE DESCRIPTION
3	service	Destination network service used
4	Flag	Status of the connection − Normal or Error
5	src bytes	Number of data bytes transferred from source to destination in single connection
6	dst bytes	Number of data bytes transferred from destination to source in single connection
29	same srv rate	The % of connections that were to the same service, among the connections
30	diff srv rate	The % of connections that were to different services, among the connections aggregated in count
33	dst host srv count	The% of connections that were to the same service, among the connections aggregated in dst host count

34	dst host same srv rate	The % of connections that were to different services, among the connections aggregated in dst host count

Table sourced from [1]

4.1.4.) **Models:**

- LazyPredict classifier was run with the normalized data. The top 5 most accurate models on the list were considered for further evaluation.
- The classifiers with highest accuracy were:
    1. LightGBM Classifier
    2. Random Forest Classifier
    3. XGBoost Classifier
    4. Bagging Classifier
    5. Extra Trees Classifier:
- As LightGBM was found to give results with maximum accuracy, this classifier was used for intrusion detection.

4.2.) HIDS

There are three parts in an HIDS:

1.) Data Source (DS)
2.) Feature Retrieval (FR)
3.) Decision Engine (DE)

4.2.1.) Data Source: DS for the implementation was ADFA-LD.

4.2.2.) Feature Retrieval: For FR, the traces were converted into dataframes for training, attack and validation traces. Then we extracted the Feature Vectors from the traces. These feature vectors were aimed to summarize the trace and capture the hidden representation of each trace. This was done by defining the following:

1.) A single process P and processes P_N
2.) System calls S
3.) T as a trace of Nth P
4.) S_i =i^{th} System call in T
5.) FV^1 = min S_i in trace T
6.) FV^2 = max S_i in trace T
7.) FV^3 = most frequent S_i in trace T
8.) FV^4 = least frequent S_i in trace T
9.) FV^5 = number of odd S_i in trace T
10.) FV^6 = number of even S_i in trace T
11.) FV^7 = average of S_i in trace T
12.) FV^8 = standard deviation of S_i in trace T
13.) FV^9 = number of S_i in trace T
14.) FV^{10} = unique S_i in trace T
11.) FM10 = FV1∩FV2∩FV3∩FV4∩FV5∩FV6∩FV7∩FV8∩FV9∩FV10

The feature vectors defined above were used for training by the Decision Engine.

Data was normalized to bring the values in the range [0,1]. This was done using 'min_max_scaler' from scikitlearn's preprocessing library.

4.2.3.) Decision Engine: In Decision Engine (DE), supervised learning algorithms were applied with the feature vector and their Detection Rate (DR) (Rate of detecting anomalous occurrences in the dataset) and False Alarm Rate (FAR) (Rate of falsely classifying a normal trace as anomalous) were observed. The following metrics were calculated using the given confusion matrix:

		Actual	Actual
		0	1
Predicted	0	TN	FP
Predicted	1	FN	TP

- Detection Rate: TP/(TN+FP+FN+TP)
- False Alarm Rate: (FPR+FNR)/2
- False Positive Rate (FPR): FP/(FP+TN)
- False Negative Rate (FNR): FN/(FN+TP)
- Accuracy: (TP+TN)/(TN+FP+FN+TP)

- Training was done using sklearn's train_test_split. The dataset was split into 30% testing data and 70% training data with a random state of 42.
- LazyPredict classifier was used to obtain a list of top-performing (based on accuracy, DR and FAR) ML models with FM10. Hyperparameter tuning was applied on the top performing model and different combinations of parameters were tried.
- The classifiers with highest accuracy were:
    1. LightGBM Classifier
    2. Extra Trees Classifier
    3. Random Forest Classifier
    4. XGBoost Classifier
    5. Bagging Classifier
- As LightGBM was found to give results with maximum accuracy, this classifier was used for intrusion detection.

The model obtained after hyperparameter tuning was the model chosen by the DE for making predictions on the trace.

5.) RESULTS

LazyPredict is a python library that is used to run various machine learning models with their default parameters to provide a quick overview and evaluate the performances. It runs various ML models (20-25 models) and provides the following metrics in decreasing order of accuracy:

- Accuracy Score
- Balanced Accuracy
- ROC AUC
- F1 Score
- Time Taken (in sec)

LazyPredict was used for both NIDS and HIDS to get an overview of how the feature vectors perform against various classifiers and high performing classifiers were chosen for parameter tuning to obtain higher accuracy.

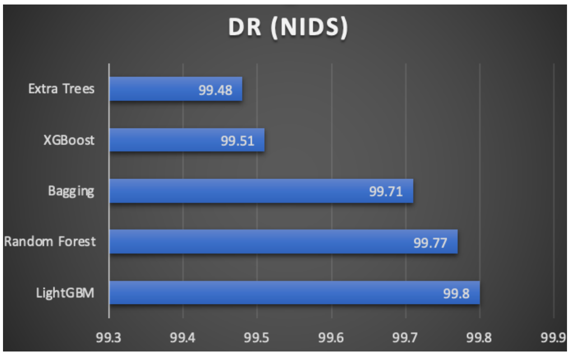
5.1.) Network IDS

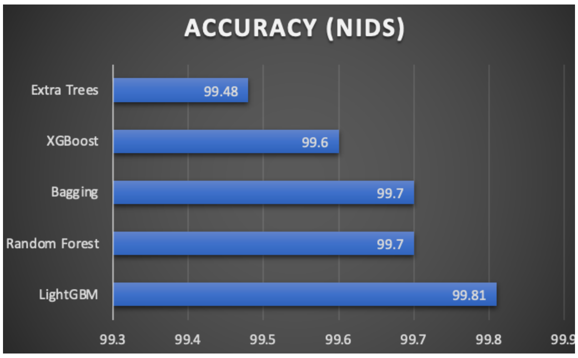
The dataset with 8 selected features was fed into LazyPredict classifier. The following were the results obtained:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
RandomForestClassifier	1.00	1.00	1.00	1.00	1.01
LGBMClassifier	1.00	1.00	1.00	1.00	0.30
XGBClassifier	1.00	1.00	1.00	1.00	1.07
BaggingClassifier	1.00	1.00	1.00	1.00	1.41
ExtraTreesClassifier	0.99	0.99	0.99	0.99	2.55
DecisionTreeClassifier	0.99	0.99	0.99	0.99	0.38
ExtraTreeClassifier	0.99	0.99	0.99	0.99	0.10
KNeighborsClassifier	0.98	0.98	0.98	0.98	1.84
AdaBoostClassifier	0.98	0.98	0.98	0.98	1.12
LabelPropagation	0.97	0.96	0.96	0.97	26.71
LabelSpreading	0.96	0.96	0.96	0.96	25.46
SVC	0.94	0.94	0.94	0.94	7.90

Image A: Top 12 most accurate classifiers for NSL-KDD

As observed from Image A, LightGBM classifier produced most accurate results with selected features of NSL-KDD.





Comparison of Detection Rates and Accuracy Rates

Detection Rates (DR) and False Alarm Rates (FAR) are shown in the following table (in %):

	CLASSIFIER	DR	FAR	ACCURACY
1	LightGBM	99.80	0.18	99.81
2	Random Forest	99.77	0.28	99.70
3	Bagging	99.71	0.29	99.70
4	XGBoost	99.51	0.40	99.60
5	Extra Trees	99.48	0.51	99.48

Some other evaluation metrics are shown in the following table (in %):

	CLASSIFIER	F1 SCORE	RECALL	PRECISION	ROC AUC
1	LightGBM	99.8009	99.8009	99.8009	99.9868
2	Random Forest	99.6874	99.7724	99.6024	99.7130
3	Bagging	99.6872	99.7155	99.6588	99.6588
4	XGBoost	99.5731	99.5164	99.6298	99.5974
5	Extra Trees	99.4456	99.4880	99.4032	99.4842

As LightGBM produced maximum accuracy, it has been used for making predictions. But other classifiers like Random Forest, Bagging, XGBoost and Extra Trees can also be used because they were also found to produce similarly highly accurate results.

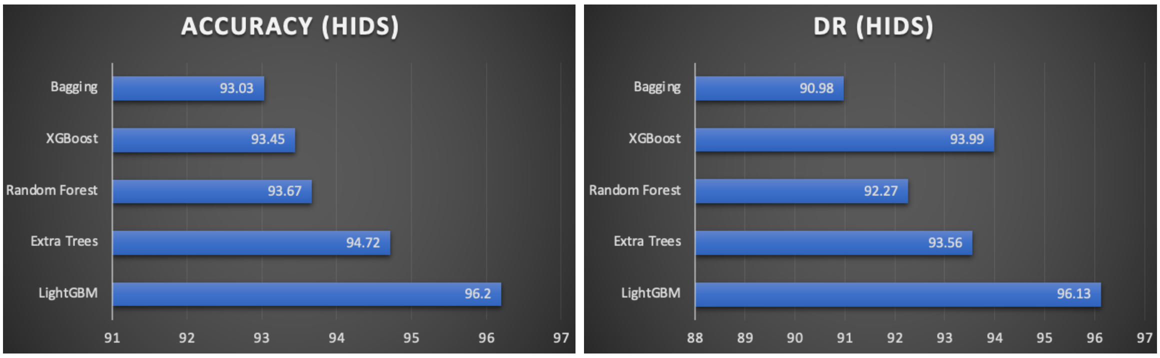
5.2.) Host IDS

FM10 was fed into LazyPredict classifier. The following were the results obtained:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	0.95	0.95	0.95	0.95	0.19
ExtraTreesClassifier	0.95	0.95	0.95	0.95	0.21
RandomForestClassifier	0.94	0.94	0.94	0.94	0.31
XGBClassifier	0.93	0.93	0.93	0.93	0.21
BaggingClassifier	0.92	0.92	0.92	0.92	0.08
DecisionTreeClassifier	0.90	0.90	0.90	0.90	0.03
LabelSpreading	0.89	0.89	0.89	0.89	0.14
LabelPropagation	0.89	0.89	0.89	0.89	0.10
KNeighborsClassifier	0.87	0.87	0.87	0.87	0.05
ExtraTreeClassifier	0.86	0.86	0.86	0.86	0.03
SVC	0.85	0.85	0.85	0.85	0.09
AdaBoostClassifier	0.85	0.85	0.85	0.85	0.17

Image B: Top 12 most accurate classifiers for ADFA-LD

It was observed that the combination of FM10 and LightGBM produced maximum DR and minimum FAR among the combinations applied.



Comparison of Detection Rates and Accuracy Rates

The following were some other results obtained by combining FM10 with other classifiers:

	CLASSIFIER	DR	FAR	ACCURACY
1	LightGBM	96.13	3.79	96.20
2	Extra Trees	93.56	5.29	94.72
3	Random Forest	92.27	6.35	93.67
4	XGBoost	93.99	6.53	93.45
5	Bagging	90.98	6.99	93.03

LightGBM classifier detected maximum anomalous traces correctly with an accuracy of 96.2%, DR of 96.13% and FAR of 3.79%.

Some other evaluation metrics are shown in the following table (in %):

	CLASSIFIER	F1 SCORE	RECALL	PRECISION	ROC AUC
1	LightGBM	95.5032	95.7081	95.2991	95.5719
2	Extra Trees	94.5770	93.5622	95.6140	94.7064
3	Random Forest	93.4782	92.2746	94.7136	93.6477
4	XGBoost	93.3901	93.9914	92.7966	93.4687
5	Bagging	92.7789	90.9871	94.6428	93.0039

Intensive parameter tuning was performed on the most accurate classifier, LightGBM. The best results were obtained with the following parameters:

METRIC	BOOSTING TYPE	DATA SAMPLING STRATEGY	LEARNING RATE	NUMBER OF ROUNDS	ACCURACY
Binary_error	Gbdt	Goss	0.05	300	96.20

Other highly accurate results were obtained using the following combinations of parameters:

	METRIC	BOOSTING TYPE	DATA SAMPLING STRATEGY	LEARNING RATE	NUMBER OF ROUNDS	ACCURACY
1	Binary_error	gbdt	goss	0.05	350	95.99
2	auc	gbdt	Bagging	0.05	300	95.78

3	Binary_error	gbdt	Bagging	0.05	300	95.56
4	Binary_logloss	dart	goss	0.05	400	95.56
5	Binary_logloss	gbdt	goss	0.05	300	95.35

The following parameters were tested:

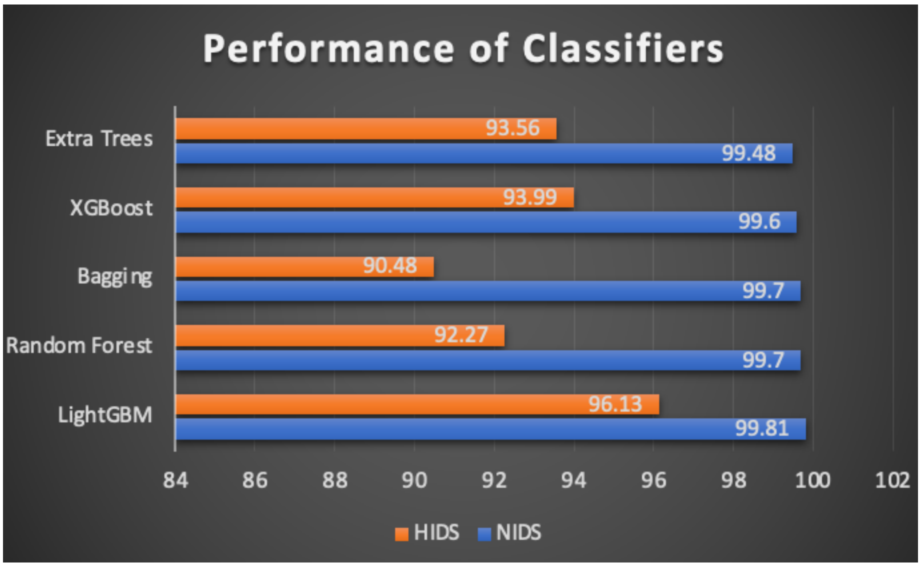
- Metric: Binary_error; Binary_logloss; AUC
- Boosting Type: GBDT; RF (random forest); Dart
- Data Sampling Strategy: Bagging; Goss
- Number of Rounds: 200; 250; 300; 350; 400
- Learning Rate: 0.01; 0.05; 0.1

$$\text{Binary Error} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(\hat{y}_i \neq y_i)$$

$$\text{Logloss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$$

- **GBDT:** Gradient Boosting Decision Tree is an ensemble learning machine learning algorithm that combines predictive power of decision trees with boosting algorithm.
- **GOSS:** Gradient One Side Sampling is a data sampling strategy that prioritizes instances with high gradient magnitudes during training to improve model efficiency while maintaining predictive accuracy.

Performance of different classifiers:



6.) REFERENCES

[1] Aljawarneh, S., Aldwairi, M. and Yassein, M.B., 2018. Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. *Journal of Computational Science*, *25*, pp.152-160.

[2] Haider, W., Hu, J., Yu, X. and Xie, Y., 2015, November. Integer data zero-watermark assisted system calls abstraction and normalization for host based anomaly detection systems. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 349-355). IEEE.

[3] Haider, W., Hu, J. and Xie, M., 2015, June. Towards reliable data feature retrieval and decision engine in host-based anomaly detection systems. In *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)* (pp. 513-517). IEEE.

[4] Kim, G., Yi, H., Lee, J., Paek, Y. and Yoon, S., 2016. LSTM-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems. *arXiv preprint arXiv:1611.01726*.

[5] Anandapriya, M.B.L.M. and Lakshmanan, B., 2015, January. Anomaly based host intrusion detection system using semantic based system call patterns. In *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)* (pp. 1-4). IEEE.

[6] Haider, W., Creech, G., Xie, Y. and Hu, J., 2016. Windows based data sets for evaluation of robustness of host based intrusion detection systems (IDS) to zero-day and stealth attacks. *Future Internet*, *8*(3), p.29.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier

[8] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier

[9] https://github.com/torvalds/linux/blob/v3.13/arch/x86/syscalls/syscall_32.tbl