Topic- Investigating the difference between XGBoost and Random Forest algorithms in Network Intrusion Detection and their implications in real-life applications.

Research Question

To what extent is the XGBoost algorithm a better choice in terms of precision, recall, speed than Random Forest in detecting DoS and Probe Network Intrusion attacks?

Word Count: 3998

Table of Contents

1. Introduction 3

2. Machine Learning Algorithms 7

3. Random Forest 9

4. XGBoost 10

5. Experimentation 13

6. Conclusion……..31
..... 29

7. Bibliography 31

8. Appendix 33

9. Glossary 37

1. Introduction

Network and system security play an essential role in the current data communication environment. Hackers and intruders create many successful attempts to cause the crash of networks and web services by unauthorized intrusion.

A Network Intrusion Attack is an attack that attempts to gain unauthorized access, steal data or perform other malicious activity on a network[1]. These attacks are categorized based on their functions, such as Denial-of-Service Attacks (DoS), Probe Attacks, User to Root Attacks (U2R). A DoS attack attempts to make a network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet by flooding the network with incoming traffic that originate from different sources[2]. A Probe is an attack during which the attacker gathers information about the Web application structure, network and scans them for vulnerabilities[3]. Associated solutions to prevent these threats are emerging together with the secured system evolution. Network Intrusion Detection Systems (NIDS) is one of these solutions.

---

[1] "Network Attacks and Network Security Threats." *Cynet*, www.cynet.com/networkattacks/network-attacks-and-network-security-threats/

[2] "Denial-of-Service Attack." *Wikipedia*, Wikimedia Foundation, 20 Feb. 2021, https://en.wikipedia.org/wiki/Denial-of-service_attack

[3] "Site Scanning/Probing: Imperva." *Learning Center*, Imperva, 9 Dec. 2020, www.imperva.com/learn/application-security/site-scanning-probing/

NIDS are devices that are distributed within networks that passively inspect the traffic of the traversing networked devices. A NIDS reads a network firewall's internal interface in a readonly mode and sends alerts to a NIDS Management server through a different network interface[4].

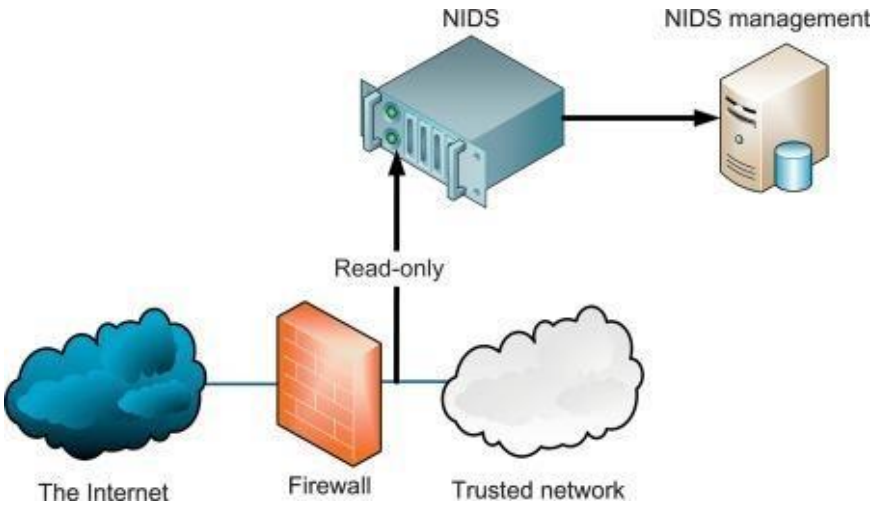

Figure 1: NIDS Architecture[5]

NIDS are categorized based on the method of detecting a Network Attack. They can be broadly divided into:

1. Signature-based Network Intrusion Detection System
2. Anomaly-based Network Intrusion Detection System

[4] *Cisco Security Professional's Guide to Secure Intrusion Detection Systems*. Syngress, 2003.

[5] Conrad, Eric, et al. *Eleventh Hour CISSP: Study Guide*. Syngress, an Imprint of Elsevier, 2017.

However, this essay would focus on the evaluation of new detection methods that a Signaturebased NIDS can utilize to detect known attacks like DoS and Probe attacks through their signatures (network traffic, network protocol, and other attributes). In this essay, DoS and Probe Intrusion attacks have primarily been taken as there has been an exponential growth of these attacks on numerous companies since 2017, with a global estimate of 14.5 million potential DoS and Probe attacks predicted between 2019 and 2022[6]. To detect these attacks, Signature-based NIDS are being combined with Machine Learning (ML) Algorithms to better predict and detect attacks based on their signatures. However, as stated by the "No Free Lunch Theorem,"[7] since no particular ML Algorithm can be considered best for a particular task, numerous ML Algorithms are compared and based on specific evaluation metrics such as Precision and Recall, speed, a particular ML Algorithm is deemed to perform the task better when compared to the other algorithms. Based on these evaluation metrics, research studies and papers on ML Integrated NIDS concluded that the supervised ML algorithm Random Forest outperformed other ML Algorithms to classify and detect these attacks[8]. Technological evolution has resulted in new ML algorithms that can be evaluated with the prior ones. One

---

[6] Freeze, Di. "The 15 Top DDoS Statistics You Should Know In 2020." *Cybercrime Magazine*, 19 Mar. 2020, www.cybersecurityventures.com/the-15-top-ddos-statistics-you-shouldknow-in-2020/

[7] Mavuduru, Amol. "What 'No Free Lunch' Really Means in Machine Learning." *Medium*, Towards Data Science, 12 Nov. 2020, www.towardsdatascience.com/what-no-freelunch-really-means-in-machine-learning-85493215625d

[8] Almseidin, Mohammad, et al. "Evaluation of Machine Learning Algorithms for Intrusion Detection System." *ArXiv.org*, 8 Jan. 2018, www.arxiv.org/abs/1801.02330

such algorithm is the XGBoost algorithm, which has been recently dominating in Applied Machine Learning due to its efficiency of computational time and memory resources utilized[9].

Hence, this essay will be focused on investigating the performance between XGBoost and Random Forest algorithms in detecting and classifying DoS and Probe Intrusion Attacks based on numerous evaluation metrics such as **precision, recall, speed**.

[9] Brownlee, Jason. "A Gentle Introduction to XGBoost for Applied Machine Learning." *Machine Learning Mastery*, 16 Feb. 2021, www.machinelearningmastery.com/gentleintroduction-xgboost-applied-machine-learning/

2. Machine Learning Algorithms

Artificial Intelligence (AI) is a broad science that deals with the simulation of human intelligence in machines[10]. Machine Learning (ML) is a subset of AI focused on imparting cognitive abilities to a machine or making a machine learn. Numerous Data Scientists, Statisticians, and mathematicians have developed several ML Algorithms and methodologies for creating and training these ML-based models. ML models receive input parameters and produce one or more output parameters based on the type of task or problem being solved.

2.1 Types of ML Model

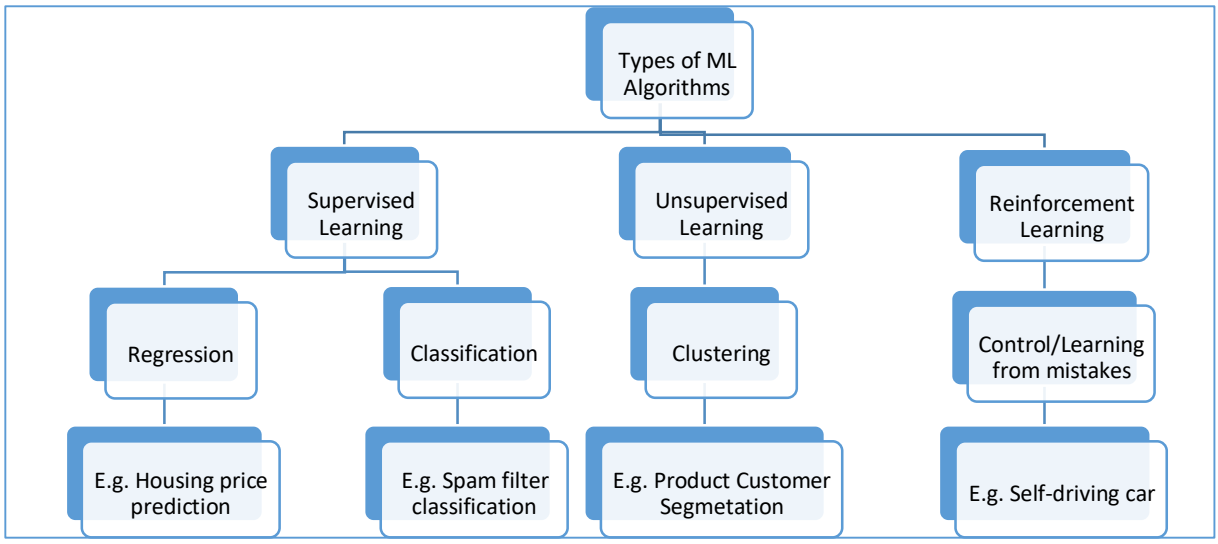

Figure 2: Types of ML Algorithms

For the problem domain that is discussed in this paper, a supervised learning model has to be implemented since a historical labelled data that consists of the attack signatures as the

[10] "Artificial Intelligence Algorithm: Categories & Classification of AI Algorithm." *EDUCBA*, 29 Dec. 2020, www.educba.com/artificial-intelligence-algorithm/

“features” and the type of attack, i.e. the “output label” are required to train and further predict future value (the type of Network Intrusion Attack).

Within the subfield of supervised learning, the task to be performed can be further narrowed down to that of a multi-class classification problem since the output is to be either ‘normal,’ i.e., normal traffic data or the type of attack, i.e., ‘DoS’ or ‘Probe.

3. Random Forest Algorithm

Random Forest is an ensemble learning method for classification, regression, and other tasks that operates by creating multiple decision trees at training time and outputting the class that is the mode of the class (classification) or mean prediction (regression) of the individual decision trees[10].

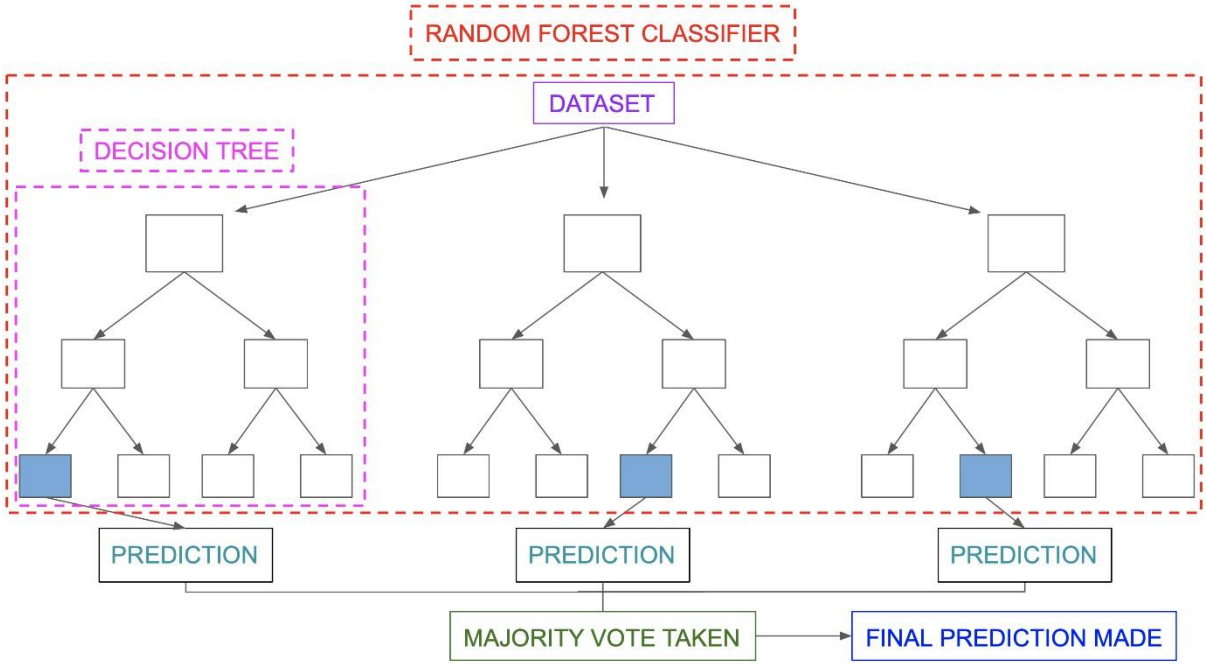

Figure 3: Working of a Random Forest Classifier[11]

A decision tree is built through recursive partitioning wherein data is split in the form of nodes based on distinguishable characteristics in values of the features corresponding to the labels that give the most optimal separation of data. Since a decision tree is a low bias, high variance

[10] "Random Forest." *Wikipedia*, Wikimedia Foundation, 16 Feb. 2021, https://en.wikipedia.org/wiki/Random_forest

[11] Kashyap, Karan. "Machine Learning- Decision Trees and Random Forest Classifiers." *Medium*, Analytics Vidhya, 13 Nov. 2020, www.medium.com/analytics-vidhya/machine-learning-decision-trees-and-random-forest-classifiers-81422887a544

model, it is prone to overfitting a training dataset due to its structure, causing a low recall and precision while performing the task in a real-life scenario.

A Random Forest hence solves this problem using multitudes of decision trees as depicted in 'Figure 3'. The training dataset is divided into multiple random samples, called Feature Bagging. Thereon, each sample is then trained by a different decision tree. The classification of testing data is then done by finding the class with the most votes, or the one wherein a majority of the decision trees classify the data point of the testing data, called aggregation. Since Random Forests aim to tackle the limitation of a single decision tree, a Random Forest is a low variance model, simplifying the data to perform a task in a real-life scenario better.

4. XGBoost Algorithm

XGBoost stands for **Extreme Gradient Boosting** and is a decision-tree-based ensemble ML algorithm that is a specific implementation of the Gradient Boosting Architecture[12]. The ensemble of decision trees utilized by XGBoost are the poor models, i.e., 'weak learners,' which are further grouped together to increase the performance.

[12] "XGBoost Algorithm: XGBoost In Machine Learning." *Analytics Vidhya*, 23 Dec. 2020, www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-mathbehind-xgboost/

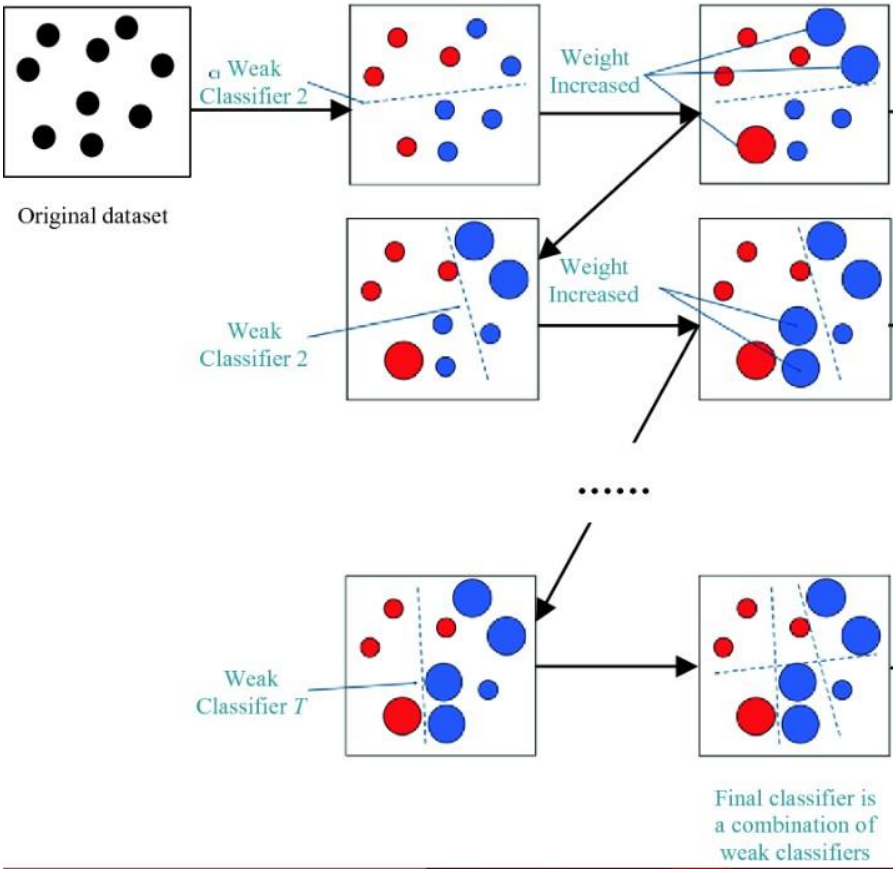

Figure 4: Working of Gradient Boosting[13]

Boosting builds models in an iterative way. In boosting, the individual models are not built on completely random subsets of data and features (as it is done in Random Forest Classifier), but sequentially by putting more weight on instances with wrong predictions and high errors. This is depicted in 'Figure 4'. The general idea behind this is that data points for which it is difficult to predict the label or value correctly for will be focused on during learning so that the model learns from past mistakes.

[13] Team, Data Science. "Gradient Boosting – What You Need to Know - Machine Learning." *DATA SCIENCE*, 15 Dec. 2020, www.datascience.eu/machine-learning/gradientboosting-what-you-need-to-know/

Furthermore, a Regular Gradient Boosting uses the loss function (i.e., the cost function) of a base model (e.g., decision tree) as a method for minimizing the error of the overall model by calculating the 1st Order Gradients This is an optimization process referred to as Gradient Descent.

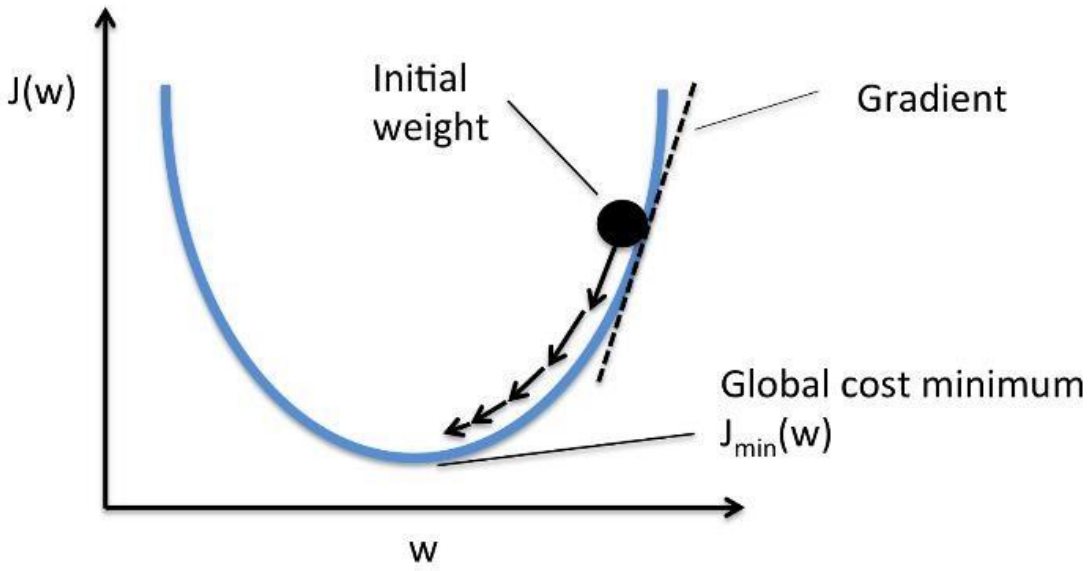

Figure 5: Gradient Descent Optimization[14]

Coming to XGBoost, however, XGBoost utilizes 2nd Order Gradients which provides more information about the direction of gradients to get the minimum of the loss function as an approximation. This is why the XGBoost utilizes a shorter computational time and computer resources to perform a task in real-life scenario that may involve regression, or classification with structured and tabular data, compared to other ML Algorithms like Artificial Neural Networks, K-Nearest Neighbors.

[14] Raschka, Sebastian. "Gradient Descent and Stochastic Gradient Descent." www.rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/

5. Experimentation

5.1 Methodology

Depicted below is a typical methodology followed for an ML-based experiment.

Figure 6: ML Experiment Methodology

To carry out a successful ML-based experiment, an elaborate planning stage is required during which the task to be performed, the data requirements, algorithms and modelling strategies have to be recognized and decided. Upon defining the task to be carried out, data acquisition must be carried out. Once the data is acquired, it is transformed into the required dataset consisting of the features and labels. During this stage, data manipulation, calculations, and other transformations such as Feature-Scaling might also be taken up. Furthermore, an ML model needs to be planned and developed based on the ML algorithm utilized. This is a critical step where Feature Selection, Feature Reduction, and finetuning of the hyper-parameters might also be taken up to minimize the resources utilized to perform the task. Upon completing this, the model has to be developed by training the model. The trained model is then deployed with the test data for which the evaluation metrics precision, recall, and prediction speed are calculated and compared with the previous results on the training data.

5.2 Planning the Experiment

As discussed in the section "Types of ML Model", the problem at hand is that of a multi-class classification - detect and classify DoS, Probe Network Attack, or a normal Network Traffic- which will be carried out using two algorithms, i.e., Random Forest and XGBoost Algorithm.

After conducting an initial study of a host of available programming languages to carry out my experiment, I decided to utilize Python due to its extensive set of libraries for ML Programming, relative high processing speed when compared to other programming languages such as R, Julia, and my prior experience in the programming language to design other projects. Thus, the experiment will be done on the PyCharm IDE utilizing Python 3.7 on a PC with Windows 10 Pro OS, 8GB RAM, Intel Core i5-10210U (4 cores) CPU with a clock speed 2.11GHz. No other user applications will be running when the two algorithms are executed and running. All 4 cores will be utilized for the experiment (**n_jobs = -1**).

Since several different aspects of performances affect the overall strength of an algorithm, in my experimentation, I will be assessing the performance of the algorithms based on precision, recall, and time taken for predicting the classes of the train data and test data by each of the ML Algorithms.

5.3 Data Acquisition

While researching an apt dataset for the experiment, I found several publicly available data sources for Network connections data and the associated attack type. One such dataset was the KDD'99 Dataset available at the UCI (University of California, Irvine) Machine Learning Repository. Since 1999, the KDD'99 has been the most widely used data set for the evaluation of ML Signature-based Network Intrusion Detection methods in numerous research studies, consisting of about 5 million connection records – 4,900,000 single network connection vectors in the training dataset with each vector containing 41 features and is labelled either as normal or an attack, with exactly one attack type.

However, there are numerous inherent problems with the KDD'99 Dataset[15]. Hence, for conducting my ML experiment, I chose the NSL-KDD Dataset containing 'KDDTrain+' dataset and the 'KDDTest+', suggested by the Canadian Institute of Cybersecurity[16].

[15] "NSL-KDD Dataset." *University of New Brunswick Est.1785*, www.unb.ca/cic/datasets/nsl.html

[16] "NSL-KDD Dataset." *University of New Brunswick Est.1785*, www.unb.ca/cic/datasets/nsl.html

5.4 Experiment Steps

5.4.1 Data Initialization and Transformation

Since the experiment focuses only on classifying and predicting Normal connections, DoS, and Probe Network Attacks, other attacks such as U2R and R2L are dropped from the training datasets. Furthermore, the specific attack labels are converted to the general categories of attack based on the table below.

Attack Types	Categories of Attack
Back, Land, Neptune, Pod, Smurf, Teardrop, Mailbomb, Processtable, Udpstorm, Apache2, Worm	DoS Attack
Satan, IPsweep, Nmap, Portsweep, Mscan, Saint	Probe Attack
Normal	Normal (Not an Attack)

Table 1: Transformation of Dataset

Upon dropping the other attack types, we get the distribution of the Normal connections, DoS, and Probe attacks in the datasets as follows (obtained upon running the Python Code):

Figure 7: Distribution of Training Dataset

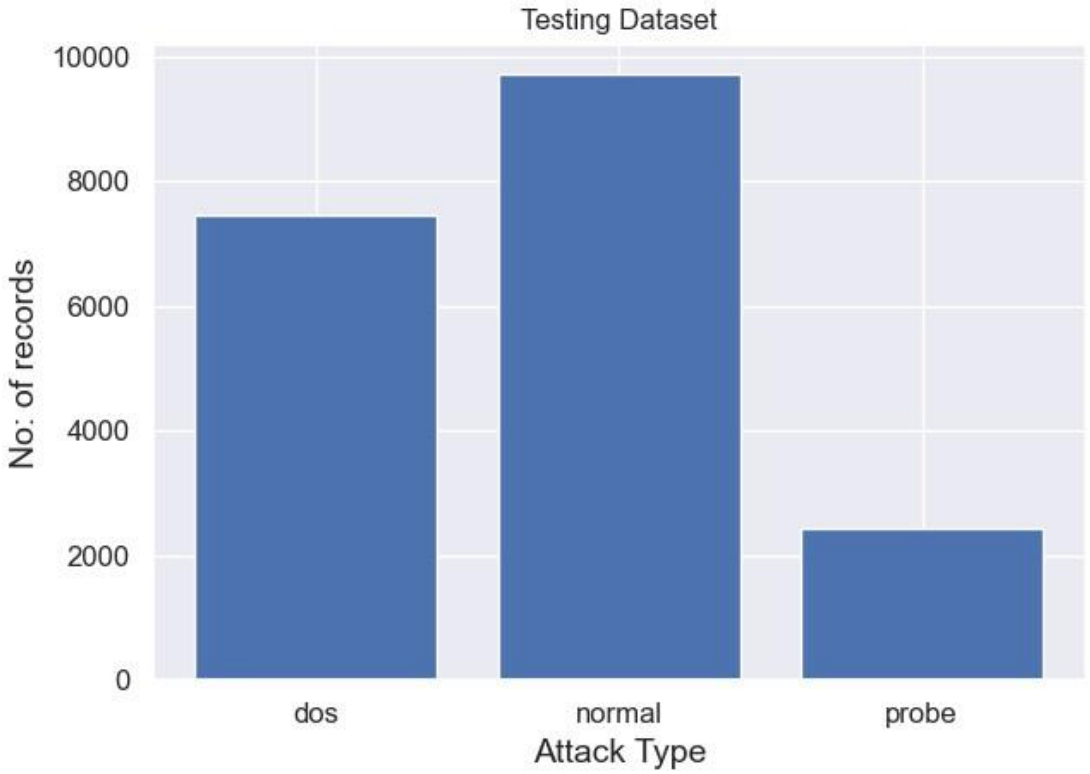

Figure 8: Distribution of Testing Dataset

Selecting Features: Proceeding with the transformation process, we see that in the training dataset, there is a redundant feature, i.e., **'num_outbound_cmds'** as all the records in the dataset contain the same value (minimum and maximum value are same = 0) and would therefore not contribute as a factor in the prediction process of the attack type, and can thus be dropped from the datasets.

	…	num_outbound_cmds	is_host_login	is_guest_login	…
count	…	124926.0	124926.0	124926.0	…
mean	…	0.0	0.000008	0.006996	…
std	…	0.0	0.002829	0.083350	…
min	…	0.0	0.000000	0.000000	…
25%	…	0.0	0.000000	0.000000	…
50%	…	0.0	0.000000	0.000000	…
75%	…	0.0	0.000000	0.000000	…
max	…	0.0	1.000000	1.000000	…

Table 2: Descriptive Statistics of Features

Encoding Categorical Features and the Labels: Furthermore, since some of the features and the labels in the training and testing dataset are alphabetical and categorical, they are mapped to numerical values so that they can be utilized for the training and prediction process.

5.4.2 Feature Reduction

Since the datasets contained over 40 features (reduced from **41** as we dropped '**num_outbound_cmds**'), features needed to be narrowed down and scaled in order to prevent the ML algorithms from overfitting the training data as overfitting would ultimately lead to a poor model that would not be able to generalize well for the testing data and when deployed for real-life utilization as well. Hence, to reduce the number of features in the datasets, numerous dimensionality-reduction algorithms can be implemented, such as **Backward Feature Elimination (BFE) and Principal Component Analysis (PCA).** In **BFE**, all the features are taken into evaluation and the least significant feature is removed at each iteration, improving the performance of the model[17]. **PCA** is a technique that utilizes simple matrix operations from linear algebra and statistics to reduce the dimensionality of datasets, increasing interpretability while minimizing information loss [18] . **BFE** is time-consuming and computationally expensive. I initially designed the ML Experiment using both the algorithms. However, **BFE** took over a day to narrow down the features to be utilized and caused the PC to crash because of overclocking. **PCA**, on the other hand, took minimal time, was less computationally dependant on the PC's resources, and reduced the number of features by replacing them with its nine calculated components in such a way that 95% of the variance between the components is retained after scaling.

---

[17] "Feature Selection Techniques in Machine Learning." *GeeksforGeeks*, 22 Jan. 2021, www.geeksforgeeks.org/feature-selection-techniques-in-machine-learning/

[18] Bonnin, Rodolfo, and Claudio Delrieux. *Machine Learning for Developers: Uplift Your Regular Applications with the Power of Statistics, Analytics, and Machine Learning*. Packt, 2017.

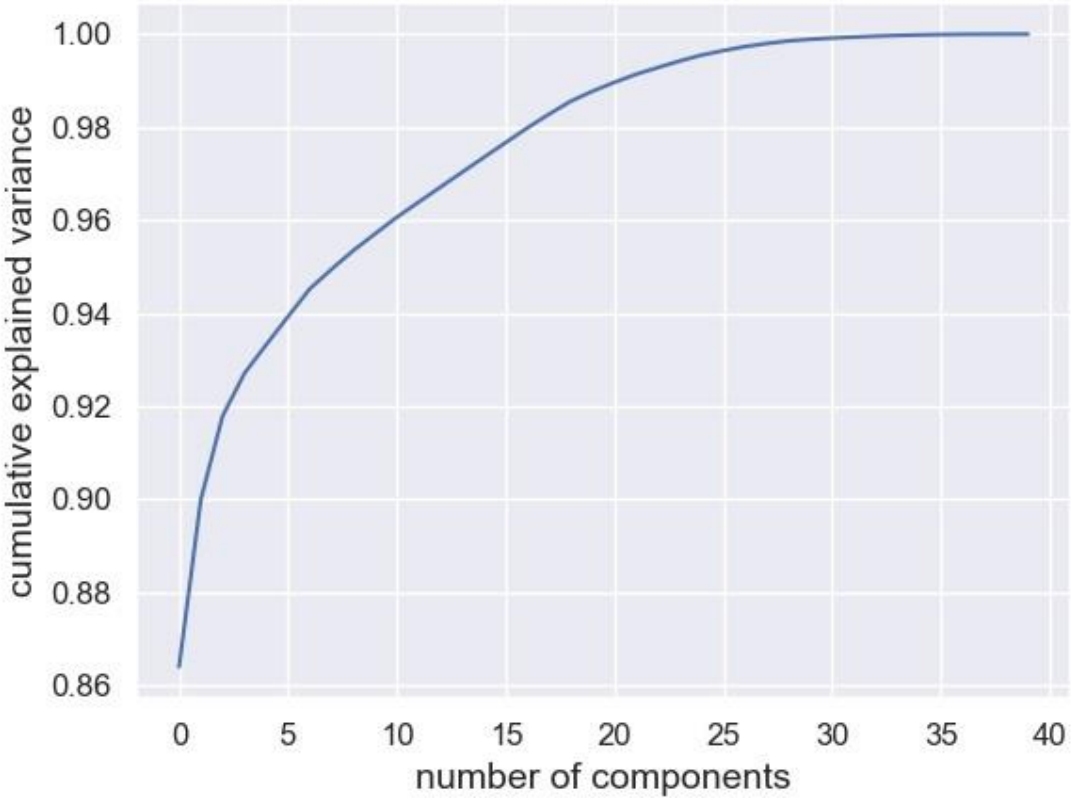

Figure 8: Variance vs No: of PCA Components

Thus, the ML Models will be evaluated in this paper with the utilization of **PCA** for Feature Reduction.

5.4.3 Developing the Model

The two selected algorithms, i.e., Random Forest and XGBoost algorithm were used to initialize the ML Models in the IDE. For both the models created, their hyperparameters have to be finetuned as they control the models' overall behaviour, which will ultimately enable the models to solve the task at hand better.

To finetune the hyperparameters of both the ML models created, I utilized the **GridSearchCV** algorithm, which searches for the best set of hyperparameters from a grid of hyperparameters values. While other algorithms such as **RandomSearchCV** could have been utilized, upon comparing the two algorithms, **GridSearchCV** was chosen over **RandomSearchCV** as in **RandomSearchCV,** random combinations of the hyperparameters are used to find the best solution for the model, leading to omissions of other possible, more optimal combinations.

Random Forest Classifier

Once initialized, the multi-class Random Forest's hyperparameters such **as max_depth, max_features, min_samples_leaf, min_samples_split, n_estimators** have to be finetuned.

Thus, I implemented the **GridSearchCV** algorithm which then calculated the best value to be utilized for each of the hyperparameters mentioned above to solve the task at hand better.

max_depth	max_features	min_samples_leaf	min_samples_split	n_estimators
5	2	1	2	50
10	3	2	5	100
15	4	5	10	150
25	5	10	15	200
30	6	-	20	250

Table 3A: Range of Values Provided for Random Forest Hyperparameters

max_depth	max_features	min_samples_leaf	min_samples_split	n_estimators
15	5	1	2	250

Table 3B: Best Values Selected for the Hyperparameters by **GridSearchCV**

XGBoost Classifier

Once initialized, the multi-class XGBoost's hyperparameters such as **learning_rate, reg_lambda, min_child_weight, n_estimators** have to be finetuned.

Thus, I implemented the **GridSearchCV** algorithm which then calculated the best value to be utilized for each of the hyperparameters mentioned above to solve the task at hand better.

learning_rate	reg_lambda	min_child_weight	n_estimators
0.0001	1	1	50
0.001	2	5	100
0.01	3	10	150
0.1	4	15	200
0.2	4	20	250

Table 4A: Range of Values Provided for XGBoost Hyperparameters

learning_rate	reg_lambda	min_child_weight	n_estimators
0.1	3	5	250

Table 4B: Best Values Selected for the Hyperparameters by **GridSearchCV**

5.5 Evaluation and Results

For comparing the models of the two algorithms for their performance in a real-life scenario, the evaluation metrics calculated for the testing dataset will matter significantly as both the models have not encountered such data beforehand, as opposed to the training dataset for which both the models will be highly biased.

5.5.1 Evaluation Metrics

In order to calculate values for certain evaluation metrics, a few terms must be understood such as-

1. **True Positives (TP) –** When the value of actual class is yes and the value of predicted class is also yes.

2. **True Negatives (TN) –** When the value of actual class is no and value of predicted class is also no.

3. **False Positives (FP) –** When the value of actual class is no and value of predicted class is yes.

4. **False Negatives (FN) –** When the value of actual class is yes and value of predicted class is no.

Precision

Precision quantifies the number of positive class predictions that belong to the positive class and is calculated for each class (in our case, the type of attack). It is calculated using the formula[19]-

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Recall

Recall is the measure of a model correctly identifying True Positives and depicts how accurately a model can identify the relevant data. It is calculated using the formula[20]-

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

[19] Bonnin, Rodolfo, and Claudio Delrieux. *Machine Learning for Developers: Uplift Your Regular Applications with the Power of Statistics, Analytics, and Machine Learning*. Packt, 2017.

[20] Bonnin, Rodolfo, and Claudio Delrieux. *Machine Learning for Developers: Uplift Your Regular Applications with the Power of Statistics, Analytics, and Machine Learning*. Packt, 2017.

Prediction Speed

The speed of the ML Model, i.e., the time taken by the ML Model the predict the class, i.e., the type of network attack.

For the experimentation performed, I used the time () module of Python to record the time taken to detect and classify the classes of the training dataset and testing dataset for the ML model of the two algorithms as precisely as possible.

5.5.2 Results

I conducted the experiment for the two models, multi-class Random Forest and multi-class XGBoost, with the utilization of PCA for Feature Reduction, on the training dataset and testing dataset.

The results obtained from the experiment for both the models are depicted in the tables below:

Model	Types of attack	True Positives	True Negatives	False Positives	False Negatives	Precision	Recall	Avg Precision	Avg Recall	Time taken to predict (entire dataset)
Random Forest	DoS	45922	78996	3	5	0.9999	0.9998	0.9998	0.9994	0.9056s
	Not an attack	67336	57562	21	7	0.9996	0.9998			
	Probe	11640	113266	4	16	0.9996	0.9986			
XGBoost	DoS	45919	78994	5	8	0.9998	0.9998	0.9994	0.9991	1.0621s
	Not an attack	67329	57554	29	14	0.9995	0.9997			
	Probe	11633	113259	11	23	0.9990	0.9980			

Table 5: Results for Training Dataset

Model	Types of attack	True Positives	True Negatives	False Positives	False Negatives	Precision	Recall	Avg Precision	Avg Recall	Time taken to predict (entire dataset)
Random Forest	DoS	1892	11913	220	5568	0.8958	0.2536	0.7244	0.5730	0.2190s
	Not an attack	9566	3704	6178	145	0.6075	0.9850			
	Probe	1164	16598	573	1258	0.6701	0.4805			
XGBoost	DoS	4940	11866	267	2520	0.9487	0.6621	0.7978	0.7104	0.1555s
	Not an attack	9498	6705	3177	213	0.7493	0.9780			
	Probe	1190	16650	521	1232	0.6954	0.4913			

Table 6: Results for Testing Dataset

5.5.3 Analysis of Results

Analysing Precision

From the results of the training dataset depicted in 'Table 5', it can be inferred that the ML Model built utilizing the Random Forest algorithm outperformed the ML Model built utilizing the XGBoost Algorithm in terms of precision of detecting and classifying each type of attack (by 1%), hence leading to a higher value for the average precision of the Random Forest-based ML Model. Furthermore, it can be recognized that for the Random Forest-based ML Model, the precision of detecting DoS, Probe attacks and the average precision of the model is almost equivalent to 1. This means that whenever the ML model predicts the type of network attack, it is correct around 100% of the time. This, however, is a drawback as this shows that the Random Forest-based ML Model has over-fitted the training dataset due to which it will not be

able to generalize/ be precise in its prediction for the test data or in real-life applications for detecting and classifying DoS and Probe attacks. This is supported as we can see that in the results obtained for the testing dataset in 'Table 6', the precision of the Random Forest-based ML Model had vastly dropped for DoS attacks (by approximately 10%), Probe attacks (by approximately 40%), and even for cases when it was not an attack (by approximately 49%). This type of performance would be detrimental for a business organization if deployed as it would predict a network attack around 40% of the time e, even though it is not an attack, which would thus lead to the wastage of the company's investment in the security of its network.

However, in the case of the XGBoost-based ML Model, we see that even though the model performed slightly lower in terms of precision (0.04% lower precision for each attack type) when compared to the Random Forest-based ML Model for the training dataset, the XGBoost-based ML Model did not overfit the training dataset and was able to generalize well for the testing dataset, leading to higher precision for detecting DoS and Probe attacks, and an overall higher average precision (9% higher average precision), hence enabling it to be more precise and reducing its chances of predicting a connection as a DoS or a Probe attack when it is not a network attack in real-life scenarios.

Analysing Recall and Speed

From ‘Table 5’ and ‘Table 6’, we observe a similar pattern for recall scores for the two models, i.e., Random Forest-based ML Model and XGBoost-based ML Model. While the Random Forest-based ML Model slightly outperformed the XGBoost-based ML model in terms of the recall scores for each attack type for the training dataset (by approximately 0.01%), it can be observed from the results calculated for the testing dataset that the Random Forest-based ML Model overfitted the training dataset due to which it had a higher recall score for each attack type as well as a higher average recall score for the training dataset, but a very poor score for the testing dataset. This is supported by the fact that the recall score for the Random Forestbased ML Model tremendously decreased from 0.9998 in the training dataset to 0.2536 in the testing dataset (74.6% decrease) for DoS attacks and from 0.9986 for in the training dataset to 0.4805 in the testing dataset (51.8% decrease) for Probe attacks. In comparison, the XGBoost algorithm only witnessed a decrease of 30% in its recall score for detecting DoS attacks, resulting in a significantly higher recall score for each attack type and a higher average recall score for the testing dataset.

Thus by comparing the recall scores for the two models performance for the training dataset and testing dataset, it can be inferred that the XGBoost-based ML Model would better perform in term of its ability to predict the correct attack in real-life scenarios, i.e., DoS, Probe, or not an attack if it were to be deployed and utilized by a business organization as part of its NIDS for network security.

A similar inference can be drawn from the results obtained for the two ML models for the training and testing dataset for the models' speed in predicting the attack type. While the Random Forest-based ML Model took a comparatively lower time to predict the attack type for the training dataset (14.7% lesser) than the XGBoost-based ML Model (resulting in a higher

prediction speed for the training dataset), the XGBoost-based ML Model, however, outperformed the Random Forest-based ML Model as it took a significantly lesser time (30% lesser) to predict for the testing dataset, hence having a higher prediction speed for the testing dataset.

6. Conclusion

Going by the evaluations and analysis of the results of the conducted experiment on the training and testing dataset, it can be concluded that **the XGBoost algorithm is a better choice in terms of precision, recall, speed than Random Forest in detecting DoS and Probe Network Intrusion attacks in real-life scenarios such as being deployed as part of a Signature MLbased NIDS when compared to the Random Forest algorithm.**

For the models of both the algorithms, however, the precision of detecting Probe attacks was significantly low for the testing dataset. Even though the XGBoost- based ML Model outperformed the Random Forest-based ML Model for detecting Probe attacks in the testing dataset (by approximately 2%), both the models would only be 65-70% correct in their prediction of a Probe attack. Hence, if either of the models were to be adopted to predict Probe intrusion attacks, there is a high possibility for the model to predict it as a wrong attack or as not an attack, which would lead the organization prone to data loss data leaks. Thus to ensure maximum network security, based on the 'No Free-Lunch Theorem,' a hybrid ML-model for a Signature-based NIDS can be proposed to utilize XGBoost and other algorithms such as MLP, Support Vector Machines for the intrusion attacks based on the evaluation metrics utilized for this study.

Limitations

Provided that the scope of this essay goes only until a certain point because of the lack of resources to do further research, there are many limitations in my experimentation such as:

1) Due to the constraints of my experimenting environment, there might be minor discrepancies in the scores calculated for the evaluation metrics when performed on a different environment. However, that would not affect the results of the experiment and its analysis.

2) The limited amount of information provided due to a word limit constraint.

3) The programming language and software used (here, Python in PyCharm).

7. Bibliography

1. “Network Attacks and Network Security Threats.” *Cynet*, www.cynet.com/network-attacks/network-attacks-and-network-security-threats/
2. “Denial-of-Service Attack.” *Wikipedia*, Wikimedia Foundation, 20 Feb. 2021, https://en.wikipedia.org/wiki/Denial-of-service_attack
3. “Site Scanning/Probing: Imperva.” *Learning Center*, Imperva, 9 Dec. 2020, www.imperva.com/learn/application-security/site-scanning-probing/
4. *Cisco Security Professional's Guide to Secure Intrusion Detection Systems*. Syngress, 2003.
5. Conrad, Eric, et al. *Eleventh Hour CISSP: Study Guide*. Syngress, an Imprint of Elsevier, 2017.
6. Freeze, Di. “The 15 Top DDoS Statistics You Should Know In 2020.” *Cybercrime Magazine*, 19 Mar. 2020, www.cybersecurityventures.com/the-15-top-ddos-statisticsyou-should-know-in-2020/
7. Mavuduru, Amol. “What ‘No Free Lunch’ Really Means in Machine Learning.” *Medium*, Towards Data Science, 12 Nov. 2020, www.towardsdatascience.com/whatno-free-lunch-really-means-in-machine-learning-85493215625d
8. Almseidin, Mohammad, et al. “Evaluation of Machine Learning Algorithms for Intrusion Detection System.” *ArXiv.org*, 8 Jan. 2018, www.arxiv.org/abs/1801.02330
9. Brownlee, Jason. “A Gentle Introduction to XGBoost for Applied Machine Learning.” *Machine Learning Mastery*, 16 Feb. 2021, www.machinelearningmastery.com/gentleintroduction-xgboost-applied-machine-learning/
10. “Artificial Intelligence Algorithm: Categories & Classification of AI Algorithm.”

EDUCBA, 29 Dec. 2020, www.educba.com/artificial-intelligence-algorithm/

11. "Random Forest." *Wikipedia*, Wikimedia Foundation, 16 Feb. 2021, https://en.wikipedia.org/wiki/Random_forest
12. Kashyap, Karan. "Machine Learning- Decision Trees and Random Forest Classifiers." *Medium*, Analytics Vidhya, 13 Nov. 2020, www.medium.com/analytics-vidhya/machine-learning-decision-trees-and-random-forest-classifiers-81422887a544
13. "XGBoost Algorithm: XGBoost In Machine Learning." *Analytics Vidhya*, 23 Dec. 2020, www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understandthe-math-behind-xgboost/
14. Raschka, Sebastian. "Gradient Descent and Stochastic Gradient Descent." www.rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/.
15. Team, Data Science. "Gradient Boosting – What You Need to Know - Machine Learning." *DATA SCIENCE*, 15 Dec. 2020, www.datascience.eu/machine-learning/gradient-boosting-what-you-need-to-know/
16. "NSL-KDD Dataset." *University of New Brunswick Est.1785*, www.unb.ca/cic/datasets/nsl.html
17. "Feature Selection Techniques in Machine Learning." *GeeksforGeeks*, 22 Jan. 2021, www.geeksforgeeks.org/feature-selection-techniques-in-machine-learning/
18. Bonnin, Rodolfo, and Claudio Delrieux. *Machine Learning for Developers: Uplift Your Regular Applications with the Power of Statistics, Analytics, and Machine Learning*. Packt, 2017.
19. "Stanford Engineering Everywhere." *Stanford Engineering Everywhere | CS229 - Machine Learning*, www.see.stanford.edu/Course/CS229

8. Appendix

Source Code

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.tree import export_graphviz from
collections import Counter
from sklearn.metrics import plot_confusion_matrix from
sklearn.model_selection import GridSearchCV import
time
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split import
warnings
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA from
sklearn.ensemble import RandomForestClassifier from
sklearn import metrics from xgboost import
XGBClassifier
from sklearn.preprocessing import LabelEncoder



X = np.loadtxt('KDDTrain+.csv', delimiter=',', dtype='object')
print(np.shape(X))
 input()
warnings.filterwarnings('ignore')
 #
Settings
pd.set_option('display.max_columns', None)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12 z =
X[np.any(X == 'normal', axis=1)] train =
pd.read_csv("KDDTrain+.csv") test =
pd.read_csv("KDDTest+.csv")
print(train.describe())

print("Training data has {} rows & {} columns".format(train.shape[0],
train.shape[1])) input()
# print(train.describe())
train.drop(['num_outbound_cmds'], axis=1, inplace=True)
test.drop(['num_outbound_cmds'], axis=1, inplace=True)
# print("Training data has {} rows & {} columns".format(train.shape[0],
train.shape[1])) scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit
variance
cols = train.select_dtypes(include=['float64', 'int64']).columns
sc_train = scaler.fit_transform(train.select_dtypes(include=['float64',
'int64']))
sc_test = scaler.fit_transform(test.select_dtypes(include=['float64',
'int64']))
```

```
sc_traindf = pd.DataFrame(sc_train, columns=cols) sc_testdf
= pd.DataFrame(sc_test, columns=cols)
# print(np.shape(z))
# print(z)
encoder = LabelEncoder()

# extract categorical attributes from both training and test sets
cattrain = train.select_dtypes(include=['object']).copy() cattest
= test.select_dtypes(include=['object']).copy()

# encode the categorical attributes traincat =
cattrain.apply(encoder.fit_transform) testcat =
cattest.apply(encoder.fit_transform)

# separate target column from encoded data enctrain
= traincat.drop(['class'], axis=1) enctest =
testcat.drop(['class'], axis=1) cat_Ytrain =
traincat[['class']].copy() train_x =
pd.concat([sc_traindf, enctrain], axis=1) train_y =
train['class']


le = LabelEncoder() le.fit(train_y)
train_y = le.transform(train_y)

test_x = pd.concat([sc_testdf, enctest], axis=1)
test_y = test['class'] counter =
Counter(train_y) plt.bar(counter.keys(),
counter.values()) plt.title('Training Dataset')
plt.xlabel('Attack Type') plt.ylabel('No: of
records') plt.show() counter1 = Counter(test_y)
plt.bar(counter1.keys(), counter1.values())
plt.title('Testing Dataset')
plt.xlabel('Attack Type') plt.ylabel('No:
of records') plt.show()
test_y = le.transform(test_y)
pca2 = PCA() pca2.fit(train_x)
plt.plot(np.cumsum(pca2.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show() pca = PCA(0.95) pca.fit(train_x) train_x
= pca.transform(train_x) test_x =
pca.transform(test_x)

forest = RandomForestClassifier(random_state = 1)
n_estimators = [50, 100, 150, 200, 250] max_depth
= [5, 8, 15, 25, 30] min_samples_split = [2, 5,
10, 15, 20] min_samples_leaf = [1, 2, 5, 10, 15]
max_features= [2, 3, 4, 5, 6]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
min_samples_split = min_samples_split,
min_samples_leaf = min_samples_leaf,              max_features =
max_features)
```

```
gridF = GridSearchCV(forest, hyperF, cv=3, verbose=1,
n_jobs=-1, scoring="neg_log_loss") bestF = gridF.fit(x_val,
y_val)
print("Best: %f using %s" % (bestF.best_score_, bestF.best_params_)) rfc
= RandomForestClassifier(**bestF.best_params_, n_jobs = -1)

rfc.fit(train_x, train_y)
disp = plot_confusion_matrix(rfc, train_x, train_y, display_labels=["dos",
"normal", "probe"],cmap=plt.cm.get_cmap('Blues')) plt.show()

start = time.time() y_pred =
rfc.predict(train_x) stop =
time.time()
print(f"Time Taken by Random Forest to Predict on Training Data: {stop -
start}s")
print(metrics.confusion_matrix(train_y, y_pred))
print(metrics.classification_report(train_y, y_pred))
print(metrics.accuracy_score(train_y, y_pred))



Model = XGBClassifier()
n_estimators = [50, 100, 150, 200, 250]
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2]
reg_lambda = [1, 2, 3, 4, 5] min_child_weight =
[1, 5, 10, 15, 20] param_grid =
dict(learning_rate=learning_rate,
n_estimators=n_estimators,min_child_weight=min_child_weight,reg_lambda=reg_
lambda )
grid_search = GridSearchCV(Model, param_grid, scoring="neg_log_loss",
n_jobs=-1, cv=3, verbose= 1)
grid_result = grid_search.fit(x_val, y_val) print("Best:
%f using %s" % (grid_result.best_score_,
grid_result.best_params_))

model = XGBClassifier(objective='multi:softmax',
**grid_result.best_params_,
n_jobs=-1)

model.fit(train_x, train_y)
start1 = time.time() y_pred2 =
model.predict(train_x) stop1=
time.time()
print(f"Time Taken by XBoost to Predict on Training Data: {stop1 -
start1}s") print(y_pred2)
print(metrics.confusion_matrix(train_y, y_pred2))
print(metrics.classification_report(train_y, y_pred2))
print("Accuracy:", metrics.accuracy_score(train_y, y_pred2))
print(model.classes_)

start2 = time.time()
y_pred3 = rfc.predict(test_x) stop2=
time.time()
print(f"Time Taken by Random Forest to Predict on Testing Data: {stop2 -
start2}s") print(metrics.confusion_matrix(test_y, y_pred3))
print(metrics.classification_report(test_y, y_pred3))
```

```
start3 = time.time() y_pred4 =
model.predict(test_x) stop3=
time.time()
print(f"Time Taken by XGBoost to Predict on Testing Data: {stop3 -
start3}s")
print(metrics.confusion_matrix(test_y, y_pred4))
print(metrics.classification_report(test_y, y_pred4))
```

9. Glossary

Accuracy of a classification model is the proportion of true results to total records.

Algorithm: embody scientific mathematical theories reduced to a set of rules to be followed for problem solving.

Artificial Intelligence: is a broad science of imitating human intelligence.

Artificial Neural Networks: algorithm inspired by the workings of neural networks in living beings.

Bias signifies how far off predicted values are from actual values. A larger bias indicates that an algorithm has not grasped relationships between features and labels.

Binary Classification / Two-Class Classification is the grouping of objects in to two classes depending on the features that they either possess or lack.

Confusion Matrix: is also referred to as an error matrix is a visualization of the performance of an algorithm.

Deep Learning: is a combination of Artificial Neural Networks with advanced computing power to handle large amounts of data.

Entropy or Gini Impurities are selection criterion for decision trees that help in determining split points for root/decision nodes on classification/regression trees.

Feature: Datapoints in a dataset that are used as variables for analysis.

Feature Selection also called variable selection or attribute selection, is the selection of features that are most relevant to the ML model problem.

KNN K-Nearest Neighbors is a non-parametric algorithm used for classification and regression.

Label: Datapoints in a dataset that represent the 'answer' to the problem.

Machine Learning: ML is a subset of AI focused on imparting cognitive abilities to a machine or making a machine learn.

ML Model: An ML model is an entity that has been trained to recognize patterns over a set of data, providing it an algorithm that it can use to reason over and learn from those data elements.

ML Model Training: is the process of usually parsing a large volume of labeled data through an ML model for it to learn the relationships between the features and labels based on a specified algorithm.

Precision is the proportion of true results to total positive results.

Random Decision Forest or simply Random forests are an ensemble learning method that construct a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Recall is the fraction of all correct results returned by the model.

Reinforcement Learning: algorithm utilizes a trial and error approach to determine which actions result is the greatest reward.

Supervised Learning: is a class of algorithms that are usually applied where historical data is used to predict likely future outcomes.

Un-supervised Learning: is usually applied where transactional data needs to be segmented or outliers need to be identified.

Variance signifies how scattered the predicted values are from actual values. Higher variance implies that the model will perform well on the training dataset but well be less accurate on a test dataset.