Exploring the Intersection of Machine Learning and Explainable Artificial Intelligence: An Analysis and Validation of ML Models Through XAI For Intrusion Detection

1st Masroor Rahman
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
masroor.rahman@g.bracu.ac.bd

2nd Reshad Karim Navid
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
reshad.karim.navid@g.bracu.ac.bd

3rd Md Muballigh Hossain Bhuyain
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
md.muballigh.hossain.bhuyain@g.bracu.ac.bd

4th Farnaz Fawad Hasan
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
farnaz.fawad.hasan@g.bracu.ac.bd

5th Naima Ahmed Nup
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
naima.ahmed.nup@g.bracu.ac.bd

***Abstract*—The use of machine learning models has greatly enhanced the capability to recognize patterns and draw conclusions. However, due to their black-box nature, it can be difficult to comprehend the factors that affect their decisions. XAI methods offer transparency into these models and aid in enhancing comprehension, examination, and trust in their outcomes. In this paper, we present a study on the use of machine learning (ML) models for intrusion detection in Windows 10 Operating systems using the ToN-IoT dataset. We investigate the performance of different ML models including tree-based models such as Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), and K-Nearest Neighbors (KNN) in detecting these attacks. Furthermore, we use Explainable Artificial Intelligence (XAI) techniques to understand how the attacks influence the processes in the Windows 10 systems and how they can be identified and prevented. Our study highlights the importance of using XAI techniques to make ML models more interpretable and trustworthy in high-stakes applications such as intrusion detection. We believe that this work can contribute to the development of more robust and secure operating systems.**

***Index Terms*—Machine Learning, Explainable Artificial Intelligence (XAI), ToN-IoT, Windows OS, Data analysis, Intrusion detection**

I. Introduction

Microsoft developed Windows, a widely used operating system that has undergone several updates over the years. The latest version being Windows 11 which was released in 2021. Windows is known for its user-friendly interface and wide range of software and applications that are compatible with it. It is also the most widely used operating system in the world [1]. However, due to its widespread usage and complexity, Windows is a prime target for cyber attacks for various reasons [2], including its popularity as a target for attackers, difficulty in securing due to legacy components and third-party software, and vulnerability caused by outdated systems and human error.

When a process starts, there are numerous other processes that fork up [3]. As so, it is very difficult to actually understand which processes in terms of features are actually important in detecting an intrusion since numerous processes start when a process begins. The main problem is that the list of features/processes is multifarious and it is very hard to narrow down the important ones.

To address this, intrusion detection systems (IDS) are used [4], which come in two primary forms: network-based and host-based. These systems can be classified by their detection methods such as signature-based, anomaly-based, and behavior-based detection [5]. We compared our findings with a similar paper which focused on the network part of the ToN-IoT dataset to detect and keep intrusion away from VANETs [6]. In this report, we aim to investigate which features and processes are crucial for intrusion detection using machine learning and explainable artificial intelligence (XAI) techniques.

In doing so, the following steps were implemented:

- Choosing the ToN-IoT [20] and meticulously evaluating the dataset by removing the flow identifier attributes in order to prevent any bias towards attacks and to avoid overfitting [6].
- Making the dataset free from noise, inconsistency, and inaccuracy by fixing imbalances in classes, categorical features, missing values, and other irrelevant features to boost the performance of our algorithms
- Using different ML models to assess the metrics and to compare them with other studies
- Explaining our results better by using Local Interpretable Model Agnostic Explanation (LIME)
- Compare and contrast our findings from ML and XAI with other research and studies

The following is the order in which the report is arranged: Chapter 2 focuses on related works and provides the background of our research. Section 3 provides a brief description of the ToN-IoT dataset and details the feature preparation process. Section 4 highlights the experimental methodology setup. Section 5 presents the results and findings of all the ML models and XAI. Lastly, Section 6 provides the conclusion of the research. Figure 1.1 shows our work flowchart.

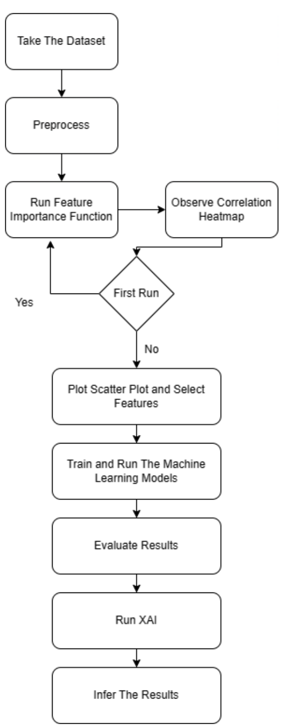

Fig 1.1: WorkFlow

II. Background

Cyber attacks of every from, like DDoS, DoS, and injection attacks, aim to disrupt the availability of a network or website by sending a huge amount of traffic or injecting harmful code. These forms of attacks can have a significant negative effect on organizations and individuals, resulting in financial losses, harm to reputation, and disruption of services.

According to [7], DDoS attacks are still a major concern for organizations globally. The report highlighted that the number of DDoS attacks in Q3 2021 rose by 8% compared to the previous quarter. [8] also states that DDoS attacks are becoming more complex and larger in scale, with an increase in volumetric attacks which are intended to overload the network and disrupt services.

Despite Windows 10 having a variety of security features to keep it safe from intrusion and malware attacks, it is still a frequent target of cyber attackers. Despite these protective measures, vulnerabilities can still be found and exploited by cybercriminals. A recent intrusion attack on Windows 10 is the Zerologon vulnerability [9], which is a vulnerability in the Netlogon Remote Protocol (MS-NRPC) in Windows Server that can be exploited to remotely take control of domain controllers and gain access to the entire domain. A recent intrusion attack on Windows 10 is the BlueKeep vulnerability [10], which is a vulnerability that allows remote execution of arbitrary code on systems using the Remote Desktop Protocol (RDP) ofWindows Server 2008, Windows 7, and Windows Server 2008 R2. It was first identified in May 2019 and attackers can exploit it remotely by executing code on the affected system.

A. Related Works

1) Intrusion Detection using ML: The article [6] presents a method for creating an Intrusion Detection System (IDS) for VANETs (Vehicular Ad Hoc Networks) that utilizes Machine Learning (ML) and is trained and tested using the ToN-IoT dataset. The dataset entails issues regarding missing values and imbalanced classes, but it covers a wider range of attack types than earlier datasets such as UNSW-NB15, KDD-CUP99, and NSL-KDD. The IDS model employs preprocessing techniques, such as Chi1 for selecting features, which reduces the amount of features, and for balancing the class SMOTE is used to enhance performance. The IDS uses various ML methods and the best results were obtained using XGBoost. In future work, the model will be deployed using Kafka Hadoop and Apache Spark, and experiments with deep learning methods and optimization algorithms for dimensionality reduction will be conducted.

According to [11], this paper shows the research of performance enhancement of a Machine Learning model is influenced by the choice of datasets and features, where the task of categorizing Linux Binaries as being potentially harmful. The dataset utilizes 4 categories of IoT files which are system, application, botnet, and general malware files. These files are utilized for any ML model. They developed a system that was trained on these data and outperformed earlier approaches using a set of features that include static and dynamic network information. According to the article, training on system files or IoT application files is no longer adequate, but priming a model on IoT botnets can help identify zero-day assaults dramatically.

According to [12], this research conducted an experimental investigation on ml methods for DDoS intrusion detection for botnets, with the algorithms examined including DecisionTree, USML, NB, SVM, and ANN. The evaluation was performed using the KDD99 and UNBS-NB 15 datasets. It demonstrates that in terms of Accuracy, False Alarm Rate (FAR), Sensitivity, Specificity, MCC, FPR(False Positive Rate), and AUC, USML is very accurate at identifying botnet and regular network traffic.

The objective of this paper [13] is to construct a classifier that can detect anomalous traffic with comparatively higher general accuracy from the N_BaIoT dataset. To produce the outcome, four binary classifiers are evaluated and validated: Support Vector Machines(SVM), Random Forests, Extra Trees Classifiers, and Decision Trees. The results show that the classifiers perform very well when all of the classifiers are utilized to train and evaluate the irregularity within each device. To detect vulnerabilities on unrelated devices, Random Forests Classifier is very efficient.

This paper [21] emphasizes the methods of optimization of Logistic regression and its mathematical model to reduce the required time to train a large magnitude of data. They reduced the number of iterations by defining the error function, improving the sigmoid function as well as using gradient descent to find the regression coefficient. This resulted in a better classification effect while keeping the accuracy the same and less time to train. Additionally, a vehicle evaluation prediction model is developed in this article to predict whether or not buyers would approve of a certain car. It offers a specific point of reference for the binary classification issue. After optimizing, they concluded that, if the value of n is larger in the Sigmoid function $\sigma(z) = \frac{1}{1+e^{-nz}}$, the amount of iterations that is necessary to obtain a similar accuracy becomes smaller.

From these papers, we understand that the mentioned machine learning frameworks are effective in terms of accurately detecting bots but require categories of dataset files. The results are noticeably accurate compared to normal detection methods.

B. eXplainanble Artificial Intelligence (XAI)

According to [15], research done on Enhancing Cybersecurity (Intrusion detection) by using Random Forest and Explainable AI, the need for Intrusion Detection Systems (IDS) in light of the growing vulnerability of cyber networks is growing tremendously (Wali, 2021). While traditional ML-based IDS have proven effective against standard cyber threats, they are vulnerable to adversarial attacks. As a solution, the article suggests a new IDS framework that integrates conventional ML-based systems with Explainable AI (XAI) to better handle adversarial attacks. This framework uses a technique called SHAP to identify and filter out malicious network traffic and to increase transparency and trust in the process of decision-making. The proposed IDS is tested and shown to have a 98.5% and 100% accuracy rate against the CICIDS dataset and Hop Skip Jump Attack, respectively. The results of this comparison with conventional algorithms support the credibility of the proposed framework and suggest that integrating regular IDS along with XAI can improve the integrity, credibility, and availability of cyber networks.

The paper [16] highlights the rising use of machine learning models in cyber-security applications, specifically intrusion detection systems (IDS), but also notes the difficulty in interpreting these models and understanding their decision-making process (Mahbooba, 2021). It's stated that previous studies have primarily focused on the accuracy of these models, but not on their explainability. The article proposes utilizing decision tree models in intrusion detection systems, along with straightforward decision tree algorithms that imitate human thinking, to solve the problem mentioned. The effectiveness of this method is evaluated using a commonly used KDD benchmark dataset, and the results are then compared to those of other advanced algorithms.

The paper [17], talks about research done on Surveying the Use of Explainable Artificial Intelligence in Cybersecurity and examines the application of Artificial Intelligence (AI) in various aspects of daily life and the issue of transparency in AI systems, which do not meet the principles of Explainable Artificial Intelligence (XAI) (Capuano, 2022). In the field of CyberSecurity, the lack of transparency in AI presents a risk as important decisions are made by systems that cannot explain their actions. The article reviews various methods in the literature that aim to provide transparency in AI results but also emphasizes the potential vulnerability of the system to adversarial attacks. This study examines the current state of Explainable Artificial Intelligence (XAI) in the field of CyberSecurity by examining over 300 papers. The study examines the main areas of application of XAI such as zero-day vulnerabilities, spam and phishing detection, crypto-jacking, botnet detection, and Intrusion Detection. The study specifically looks at the methods used to make these systems explainable and identifies promising work and areas for further research.

III. The ToN-IoT Dataset

In order to gauge the effectiveness, efficiency, and fidelity of cybersecurity applications based on Artificial Intelligence oriented Deep Learning and ML algorithms, a new generation of datasets was developed by Dr. Nour Moustafa, called the TON_IoT Datasets, UNSW Research, n.d. [18]. The datasets are referred to as ”ToN IoT” since they comprise a variety of data sources including Ubuntu 18 and 14 Transport Layer Security and Network Traffic Datasets, Windows 10 and 7 operating system datasets and telemetry datasets retrieved from IIoT sensors and IoT devices. The datasets were gathered from a large-scale, realistic network created at the Australian Defense Force Academy’s IoT Labs & Cyber Range, School of Engineering and Information Technology (SEIT), UNSW Canberra (ADFA). The IoT and IIot networks which constitute the industrial 4.0 network have a new testbed network. To supervise the connection amongst the three levels of Fog or Edge Systems and Cloud, IoT, the deployment of the testbed was done utilizing several VMs (virtual machines) and hosts of Kali, Linux, and Windows Operating Systems. We took decided to select the Windows 10 dataset as it had a lot of features and was in addition a decent-sized dataset. The IoT and Linux datasets had more data as a whole but they lacked a sufficient amount of features while the Windows 7 dataset seemed irrelevant since a very low percentage of people use the Windows 7 operating system. After choosing the datasets we formulated a work plan to achieve the highest accuracy and interpretability.

Windows Intrusion Detection:
The Performance Monitoring Tool was used to trace the utilization of several resources like Memory, Disk, Ram, Processor, and Network of Windows 10 machines. [19] The dataset we chose to work with was extremely rich, bearing more than 125 features and over 35,000 rows of data.

This dataset had a lot of features. In table 3.1 [20], we tried to showcase a few of them:

Windows 10 Feature Description

ID	Feature	Description
1	LogicalDisk Total Free_Megabytes	The quantity of unused space on the storage device, expressed in mega bytes.
2	Memory Pool Paged Resident Bytes	This feature shows the amount of memory, measured in bytes, currently being used by the part of the system’s virtual memory called the paged pool. The paged pool is used for storing objects that can be moved to disk when not in use. It should be noted that this counter only displays the last recorded value and is not an average.
3	Memory Committed Bytes	The quantity of virtual memory that has been designated for use, represented in terms of bytes.
4	Memory Standby Cache Core Bytes	This counter shows the amount of physical memory, measured in bytes, that is allocated to the core standby cache page lists. This memory is used to store cached data and code that is not currently being used by any processes, the system or the system cache.
5	Memory Standby Cache Normal Priority Bytes	This counter displays the amount of physical memory, measured in bytes, allocated to the standby cache page lists of normal priority. This memory contains cached data and code that is not currently being used by processes, the system or the system cache. It is ready for immediate use by a process or the system.
6	Memory Long-Term Average Standby Cache Lifetime (s)	Over a lengthy period, the average lifespan of the data in the standby cache is calculated.
7	Memory Cache Bytes	This feature indicates the amount of physical memory, measured in bytes, being used by the system file cache This cache stores frequently used files for quick access. It should be noted that the counter only displays the most recent recorded value and not an average.
8	Network_I (Intel R _82574L_GNC) Bytes Sent sec	This counter measures the rate at which data is being transmitted over eac network adapter, including any additional data added for framing purposes.
9	Process_IO Read Bytes_sec	This feature measures the speed at which a process is reading bytes from input/output operations. It takes into account all types of I/O actions carried out by the process, including reading from files, network, and devices. It gives a sense of how much input/output operations are being performed by the process and at what rate.
10	Process_IO Data Operations_sec	This feature measures the rate of read and write I/O operations initiated by a process, including file, network and device I/Os, giving an understanding of the number and frequency of the I/O operations performed by the process.
11	Process_Pool Nonpaged Bytes	This feature shows the amount of memory, measured in bytes, being used by the nonpaged pool, which is a part of the system’s virtual memory that stores objects that cannot be moved to disk and must stay in physical memory as long as they are allocated. The calculation of this counter is different than that of the ”Process/Pool Nonpaged Bytes” so the two might not be the same. It should be noted that this counter only displays the last recorded value and is not an average.
12	Process_pct_ User_Time	The portion of total execution time that process threads spent running code in user mode.
13	Process_IO_Write Operations_sec	I/O operations to write data are being sent by the process. Including file, network, and device I/Os, this feature tracks all I/O activity produced by the process.
14	Process_IO Read_ Operations_sec	Read I/O operations are being sent by the process. This feature records every file, network, and device I/O activity produced by the process.
15	label	Marked records for normal and attacks, where 0 denotes normal and 1 denotes attacks.

A. *Preprocessing*

The ToN IoT dataset was enormous and the data was unbalanced. There were null values disguised as singular spaces, in addition to random unnecessary spaces before and after a lot of the numerical entries. This made some of the columns in the dataset a mixture of both string and numerical inputs. In order to run a tree-based model, we first had to preprocess the dataset, so that the model could go through the dataset without any issues and perform the computation.

1) Formating Data: The first order of business was to replace all the null values disguised as empty strings and also get rid of all the trailing and leading spaces around the numerical values. We noticed that the empty spaces for all the columns seemed to be in multiple columns and on the same rows. So we decided to remove those particular rows entirely as the number of such rows was very small compared to the overall size of the dataset.

Furthermore, the concatenated string and numerical values were dealt with by removing the string values using the ‘apply map’ function with the help of the lambda function. The resulting boolean data frame was then negated using the ’$\sim$’ operator and then used to index the original data frame, keeping only the rows where all cells are not empty.

2) Feature Selection: Since there were a total of 127 columns, including the ’label’ of attack and the ’type’ of attack, it was imperative that we conduct feature selection before advancing further. From the start, we removed the column which kept track of exactly ‘when’ the attack took place. Afterward, we plotted a feature importance bar chart for the remaining features using the Random Forest Classifier model.

From this huge feature importance map, we decided to keep around 30 features according to their feature importance. We then plotted a heatmap to cross reference.

From the heatmap, we realized there were some weakly correlated data to the ‘label’ column, which is our target variable.

To further narrow down our list of suitable features for our target variable, we plotted to scatter plots for all the remaining columns to get a feel for the variance of the data in each of the columns. From these scatter plots, we found that a lot of the values in the feature columns are not evenly distributed and have quite a number of outliers. Such a scatter plot is shown in figure 3.1.

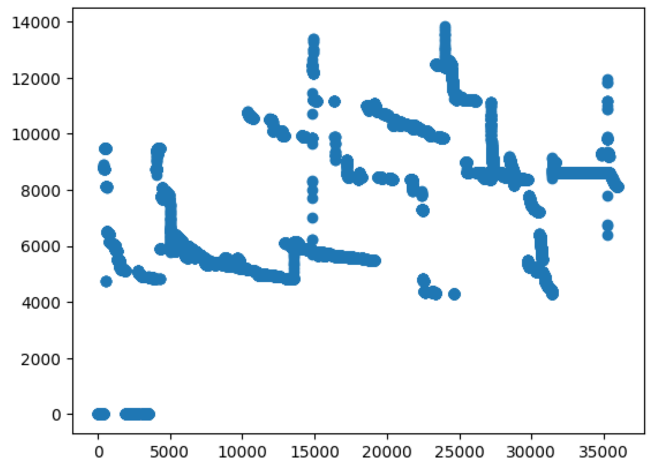

Fig 3.1: Feature 'Network I(Intel R 82574L GNC)TCP APS' Scatter Plot

The y-axis of the scatter plot denotes the value, while the x-axis is the row that a particular value is in. The feature whose values resulted in this scatter plot is not even remotely distributed equally in any range. Thus, features that have to scatter plots like this were dropped. While very few plots were perfect initially, a lot of the features did give out plots that had consistent values at certain ranges. Such as in figure 3.2

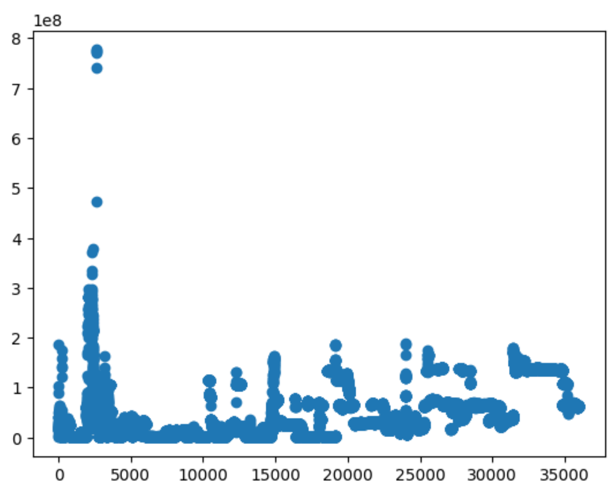

Fig: 3.2: Feature 'Memory Cache Bytes' Scatter Plot Before

This feature's scatter plot had a high density of values from 0 to $2*10^{8}$. Thus, we took this feature, and others liked it and adjusted their range accordingly. The above scatter plot with the new modified range is shown in figure 3.3.

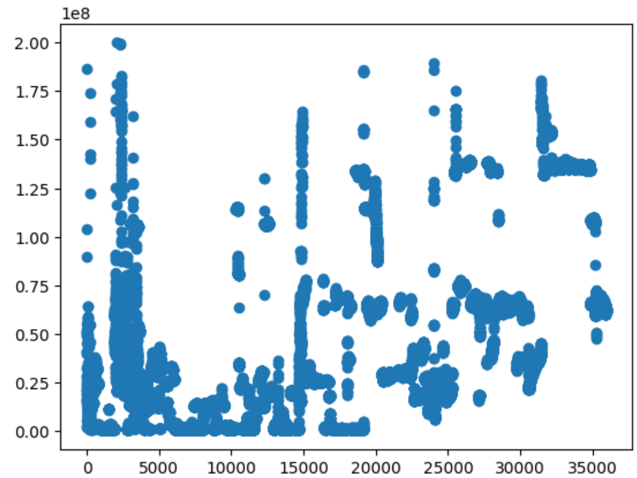

Fig 3.3: Feature 'Memory Cache Bytes' Scatter Plot After

After getting rid of all the high-variance columns and adjusting the range of the relatively lower-variance columns, we were left with 14 feature columns. We then plotted a feature importance bar chart using Random Forest Classifier to get the final feature importance of our selected features. The chart and its corresponding heatmap are shown in figure 3.4 and figure 3.5.

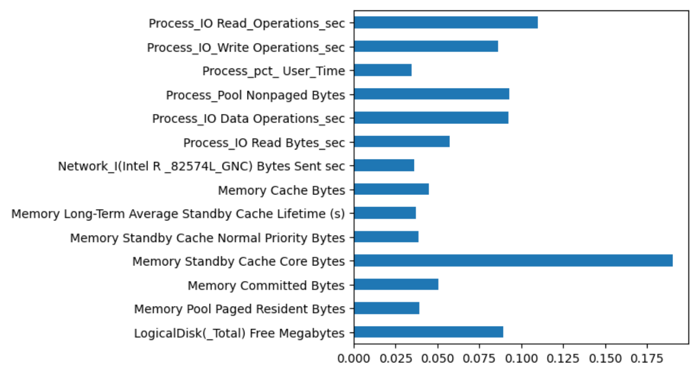

Fig 3.4: Windows 10 Remaining Features

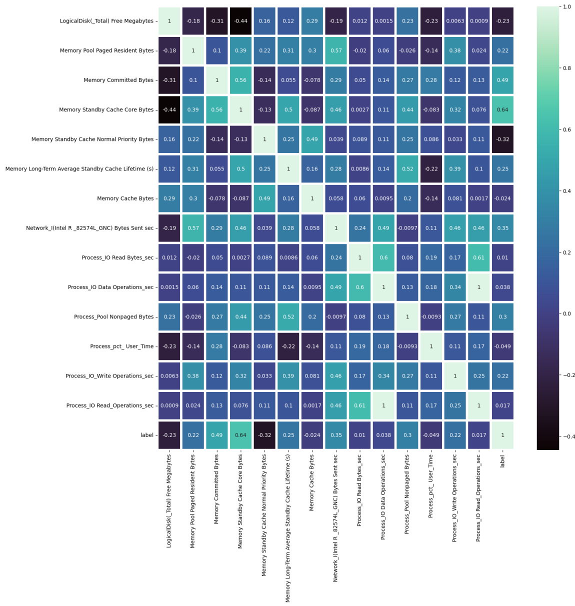

Fig 3.5: Windows 10 Final Heatmap

After preprocessing was done, it was time to train our models. We used 15 folds (Stratified Kfold) for our models across the board as it gave us the highest scores in the relatively low time.

IV. Model Implementation

A. Tree-Based Models

Random Forests and Decision Trees are non-linear machine learning models. They are used for mainly three purposes, feature selection, regression, and classification. In our research, we used Random forest to select the features using Feature Importance and then performed classification using the RandomForestClassifier and Decision Tree Classifier.

In a random forest, data instances are selected randomly from the data set and this process is called bootstrapping. Furthermore, it uses several decision trees (similar to “Forest”) to build a decision model and this process is known as the bagging method.

The random forest classifier and the decision tree classifier have several parameters. Depending on the dataset, the parameters vary. In order to determine the best parameters for our use-case scenario, we used RandomizedSearchCV instead of GridSearchCV. RandomizedSearchCV is superior to GridSearchCV for a number of reasons, one of which is that it utilizes random sampling to select a subset of parameters for testing, which is faster than testing all possible combinations as GridSearchCV does. Additionally, RandomizedSearchCV allows for more exploration of the parameter space, and it can handle continuous parameter distributions, whereas GridSearchCV can only handle discrete values. Furthermore, RandomizedSearchCV can evaluate multiple metrics simultaneously and return the best results based on the specified metric.

The optimal parameters for the Decision Tree Classifier and Random Forest Classifier were determined respectively:

DecisionTreeClassifier (criterion= ’gini’, max_depth= 5000, min_samples_leaf= 8)

Random Forest Classifier (’bootstrap’: False, ’criterion’: ’entropy’, ’max_depth’: 3, ’max_features’: 2, ’min_samples_leaf’: 2, ’n_estimators’: 10).

We ran our tree-based models with the aforementioned parameters and with default parameters. We also ran the model without scaling and with Standard Scaling. Tree-based models typically do not necessitate feature scaling.

The decision tree algorithm aims to assign a target value to an item by mapping its observations. The algorithm accomplishes this by repeatedly dividing the data into smaller groups based on one or more input features. The ultimate goal is to create groups or subsets that have similar target values. These division or partitioning decisions are represented by the nodes in the tree, and each final group or subset is represented by a leaf of the tree. It is crucial to pay attention to overfitting when utilizing decision tree algorithms as it can happen when the tree is overly complex with too many branches and leaves. Thus we have tweaked the parameters to optimize the accuracy of the model. After running the model, the decision tree algorithm gave us the following scores:

Accuracy of Decision Tree: 95.43%
F1_Score of Decision Tree: 96.0%
AUC Score of Decision Tree: 95.31%

The key factors in determining the behavior of a random forest include the number of decision trees in the forest (n_estimators), how deep each tree can grow (max_depth), the minimum number of samples required for a split to occur at internal nodes (min_samples_split), the minimum number of samples required for a leaf node (min_samples_leaf), and the number of features to be taken into consideration when finding the best split (max_features). Additionally, there are other parameters that can be set such as the method used to evaluate the quality of split and bootstrapping options, among others. In a random forest, Gini impurity and entropy are both ways to measure the impurity of a set of data. Gini impurity is a measure of how often a randomly selected element would be mislabeled if it were randomly labeled according to the class labels distribution, it can be calculated by subtracting the sum of squares of the class probabilities from one. Entropy, on the other hand, is a measure of uncertainty or disorder in a set of data, it’s calculated by summing the product of the probability of each class and the algorithm of the probability of that class. The decision tree algorithm uses one of these criteria to split the data based on feature values. Generally, Gini impurity is used for binary classification problems, and entropy is used for multi-class classification problems. However, in practice, the choice of Gini impurity or entropy as a criterion for splitting in a decision tree may not have a significant impact on performance, as other factors such as the number of trees, max_depth, and max_features often have a greater effect on the overall performance of the random forest.

We ran our Random Forest Model with the aforementioned parameters found from running RandomizedSearchCV. We also ran the model without scaling. The Random Forest algorithm typically does not necessitate feature scaling. The method is based on building multiple decision trees and merging them to produce a final prediction. Each decision tree is created independently, using a random subset of the data and a random subset of the features. As the algorithm is not based on distance measurements, it does not require scaling and can handle data with different scales and units. However, in certain situations, it can be advantageous to scale the features before using them in a random forest. For example, if one feature has a much larger scale than the others, it can overpower the split criterion and cause overfitting. Scaling the features can prevent this from happening and make the interpretation of feature importance more meaningful. The accuracy of our model with parameters from the RandomizedSearchCV was:

Accuracy of rfc: 96.26%
F1_Score of rfc: 96.77%
AUC Score of rfc: 95.92%

Figure 4.1 and 4.2 illustrates the learning curve for the models:

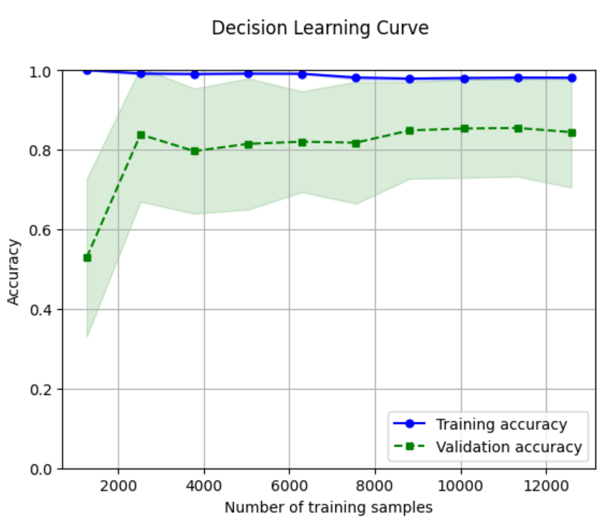

Fig 4.1: Learning Curve of Decision Tree Classification

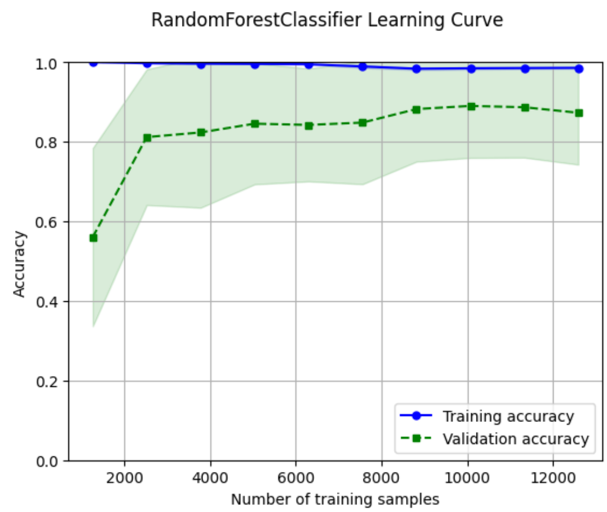

Fig 4.2: Learning Curve of Random Forest Classification

B. Logistic Regression

We have tuned the parameters of the model in order to get the best results. The parameters of Logistic regression contain 'C' which is the inverse of regularization strength, 'penalty' which is the type of regularization, and the 'solver' which is the algorithm used for the optimization. Solvers such as "Liblinear" is a decent option for small datasets, whereas "sag" and "saga" are quicker for large datasets. The 'lbfgs' solver also known as Limited memory Broyden Fletcher Goldfarb Shannois solver, is used for the optimization. It uses 'L2' and 'none' penalties and is good for unscaled datasets. Most of the parameters are set as default as they are optimal for our dataset. The class weight is tweaked to "balanced" as the outcome feature of the dataset has imbalance sets of 0's and 1's.
For training and splitting the model, we have used Stratified K-Fold cross-validation as it is good for the distribution of the target variable. The features of the dataset were selected using the feature importance map generated for the random forest classifier. We further scaled the features by observing the scatter plot and histogram of the features.
We have also implemented a min-max Scaler for normalizing the independent variables. Since each variable will be on the same scale, it may be possible to avoid having any one variable have a significant influence on the model. By making it simpler for the optimizer to locate the global minimum of the cost function, it also aids in enhancing the performance of the logistic regression model. We ran our model with and without min-max Scaler and found that the best results are when the Scaler is used.

After training and testing our model, we found the accuracy, F1 and Cross-Validation, and AUC scores which are given below-

Accuracy of LR: 85.15%

F1_score of LR: 87.11%
AUC Score of LR: 84.69 %

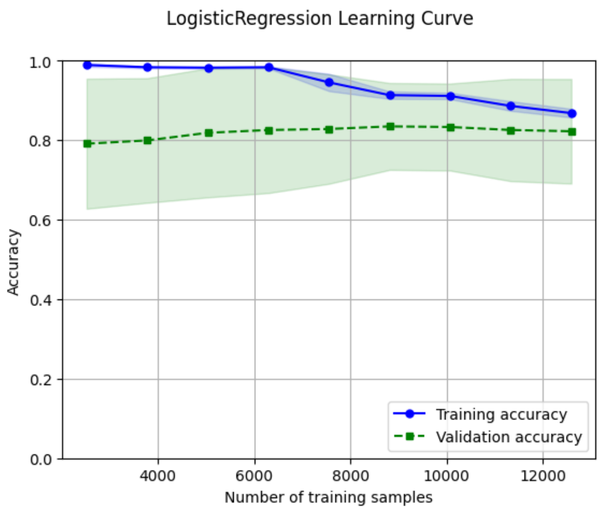

Fig 4.3: Learning Curve Logistic Regression

We have also plotted the learning curve of our model shown in figure 4.3 from which we could perceive that as the model is initially trained on the data, the accuracy of logistic regression often increases rapidly at the beginning of the learning curve. The accuracy of the model often rises slowly as additional data is supplied and finally approaches a level when the model performs at its peak on the provided dataset. As we can see the training accuracy is very close to the validation accuracy at the end of the training session.

C. K-Nearest Neighbors

Since the ToN-IoT dataset is a classification-based dataset, we preprocessed our dataset according to the requirements. Stratified k-fold cross-validation was used to evaluate the performance of the K-Nearest Neighbors (KNN) algorithm by dividing the data into k partitions or ”folds” and training the algorithm k-1 of the folds while testing the remaining one, this process is repeated k times. The results are then averaged to get an overall measure of the algorithm’s performance.

To avoid data leakage, the dataset was scaled after verifying that the data used for training and testing were appropriately separated, and techniques such as cross-validation were employed to guarantee that the model was not overfitting. Data leaking is a phenomenon that occurs when information from the test set is used to train a model, resulting in overfitting and models that cannot generalize well to new, unknown data. It can occur when the data used for training and testing are not adequately segregated, or when the model is not properly validated.

If all features are equal in importance but not on the same scale, they must be normalized; otherwise, the features with the greatest magnitude will dominate the overall euclidean distance unless we employ the Manhattan distance. Since distances are measured in KNN, feature scaling is a must. We used MixMaxScaler which scales the value between 0 and 1. Both MinMaxScaler and StandardScaler gave similar results in this dataset so either one can be used.

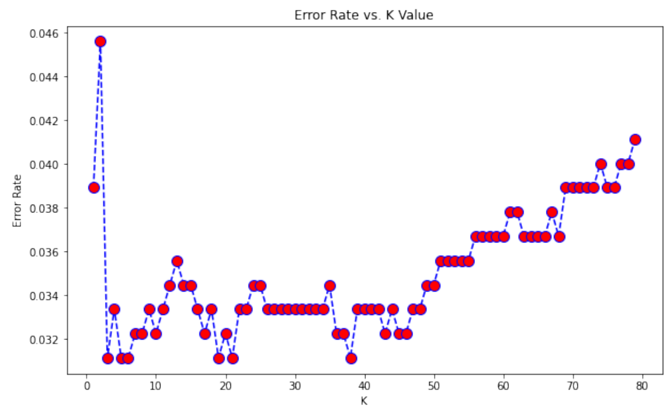

Fig 4.4: Finding the exact value for k

The value of k is often determined through experimentation, and common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance. The weighting schemes are uniform weighting and distance weighting. Figure 4.4 shows the illustration of error rate vs K value.

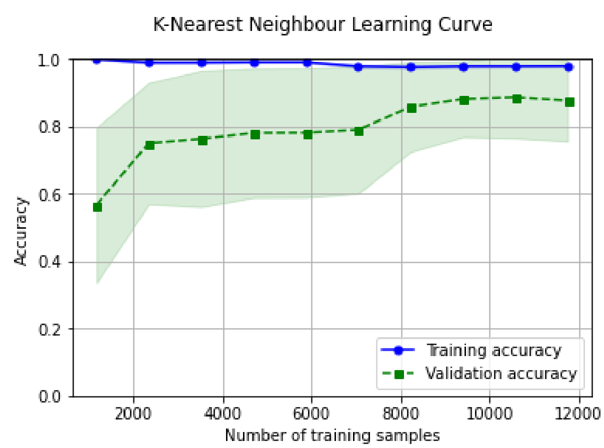

Fig 4.5: Learning Curve of KNN

The n_neighbor was set to 22 after detecting the best value using GridSearchCV. A graph of the error rate was plotted which shows that as the value of k increases, the error rate goes down and after a certain point the values start oscillating between a range. If the value of k is set from that range there might be a possibility that the model might be overfitted and the accuracy will deteriorate. We chose the value for which this model gives the best results. The distance was set as

'Minkowski' by default and the value of p was chosen to be 1 considering Manhattan distance. The accuracy came out to be:

Accuracy of KNN: 97.22%
f1_score of KNN: 97.6%
AUC Score of KNN: 96.9%

After this, we visualized the learning curve to assess the results shown in figure 4.5.

V. Result

A. Analysis Of Windows10 Dataset:

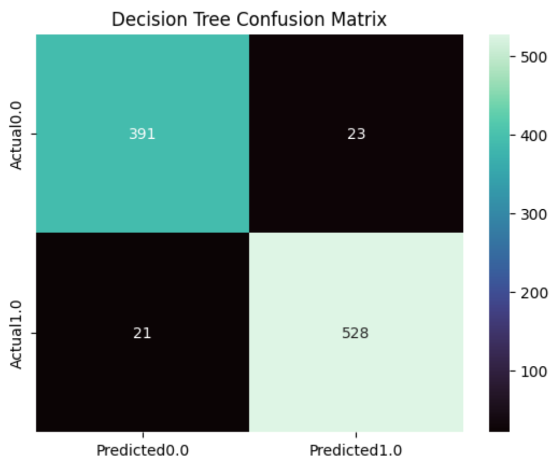

Fig 5.01: Confusion Matrix of Decision Tree

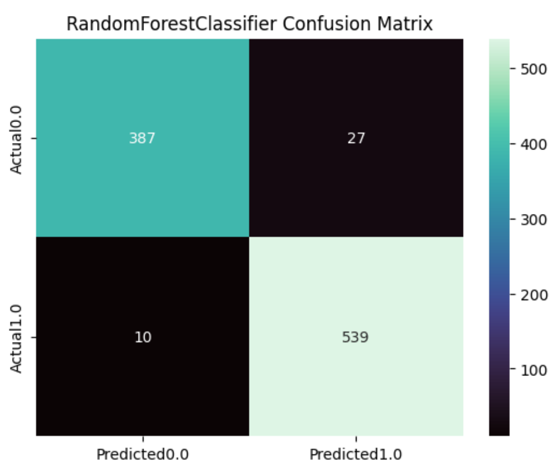

Fig 5.02: Confusion Matrix of Random Forest

The confusion matrix for the Decision Tree, Random Forest, Logistic Regression and KNN on the Windows10 dataset are shown in Figure 5.01, 5.02, 5.03 aand 5.04 respectively.

It's conceivable that false negatives (missed intrusions) in an intrusion detection dataset are perceived as more dangerous

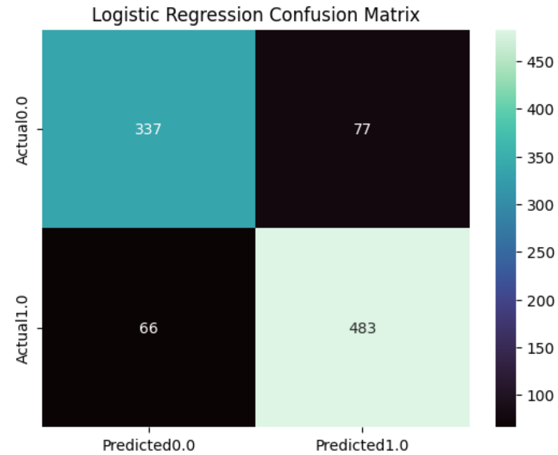

Fig 5.03: Confusion Matrix of Logistic Regression

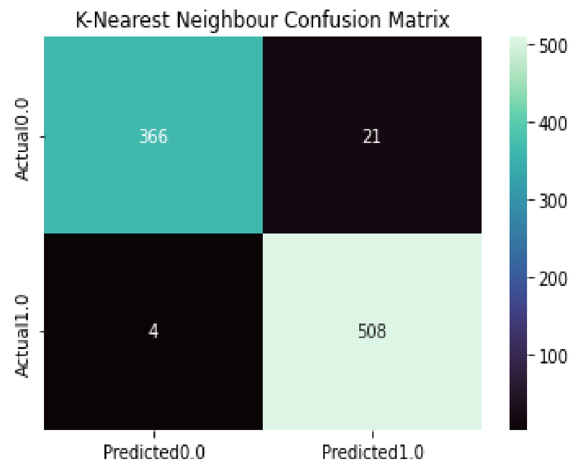

Fig 5.04: Confusion Matrix of KNN

than false positives (false alarms). Although the algorithm predicted that the device had been attacked, the false positives show that the device is not actually under attack. Therefore, even if the algorithm's prediction was incorrect, we won't be negatively impacted by it. But it's very risky if an attack is falsely anticipated as a non-attack. The low amount of false negatives also suggests that the algorithm will operate more accurately in the event of an attack scenario. Therefore, it would be best to use the model with the lowest false negative rate.

In the confusion matrix for the random forest, we can observe that out of a total of 414 non-attack records, 27 were incorrectly predicted as attacks (FP). Moreover, out of a total of 549 attack records, 10 were incorrectly predicted as non-attacks (FN). This method yields the fewest possible mistakes. When compared to the findings from other methods, k-NN's 4 false negative instances provide substantially good accurate estimates too. However, k-NN would place some strain on the system; there are a few more false positives; 21

to be exact. As a result, it can't compete with methods like random forest classifiers.

When compared to other classifiers, the decision tree's accuracy is around average, with about 27 false negatives and 23 false positives. When compared to other methods, logistic regression yields the least trustworthy results, with a high rate of false negatives (66) and false positives (77) overall Logistic regression would put a heavy load on the infrastructure.

It's vital to keep in mind that having a low false negative rate is not the only element to take into account; the model's overall accuracy, precision, recall, and other metrics like F1-score, and AUC, should also be taken into account to evaluate the model's performance.

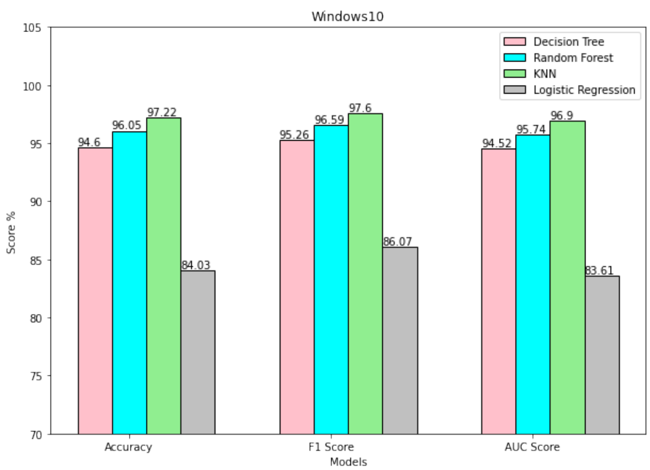

Fig 5.05: All Scores

The resulting Accuracy, F1, and AUC scores from each method are shown in figure 5.05 in percentage form in the aforementioned table 5.06, allowing for easy comparison between them. It appears that all four models have high accuracy and F1-score, which indicates that all models are doing a good job of classifying the data. However, looking at the numbers in the table, it's clear that the K-Nearest Neighbor(KNN) algorithm outperformed the other classifiers. The K-Nearest Neighbor(KNN) algorithm has shown tremendous potential by exceeding all other methods on all three measures of the matrix. It provides the highest accuracy and F1 scores among the models given. The K-Nearest Neighbor(KNN) algorithm also has the highest AUC score among the models.

Table 5.06: Accuracy, F1-Score and AUC Score table

Windows 10	Accuracy	F1-Score	AUC
Random Forest	96.16	96.68	95.83
Decision Tree	94.81	95.43	94.76
k-Nearest-Neighbor	97.22	97.6	96.9
Logistic Regression	85.15	87.11	84.69

The performance of a binary classifier system as the discrimination threshold is changed is depicted graphically by ROC curve. The true positive rate (TPR) against the false positive rate (FPR) at various threshold values are plotted on the ROC curve.

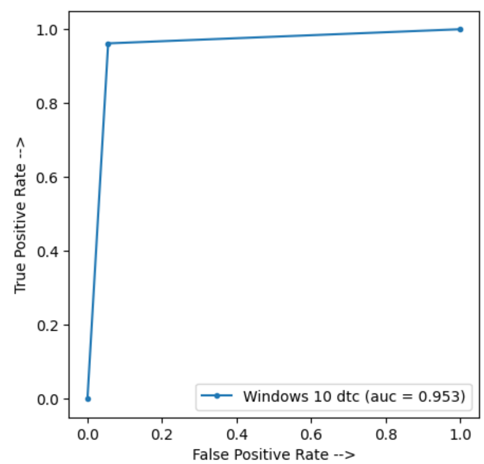

Fig 5.07: ROC curve of Decision Tree Classifier

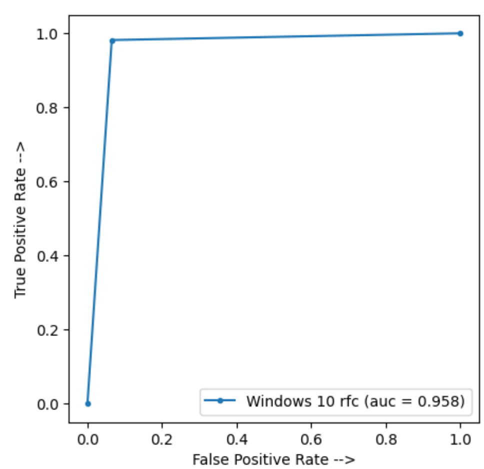

Fig 5.08: ROC curve of Random Forest Classifier

From Figure 5.04, we can see that the ROC curve of the K-Nearest Neighbor(KNN) indicates that the classifier has a very high ability to distinguish between positive and negative classes. In this case, the AUC of 0.969 for the K-Nearest Neighbor(KNN) is very close to 1, which means that the classifier is able to correctly classify a high proportion of positive and negative cases. The ROC curve is likely to be close to the top-left corner of the plot, which represents a high true positive rate (TPR) and a low false positive rate (FPR) at various threshold settings. It means that the K-Nearest Neighbor(KNN) is able to identify most of the positive cases as positive and most of the negative cases as negative. With

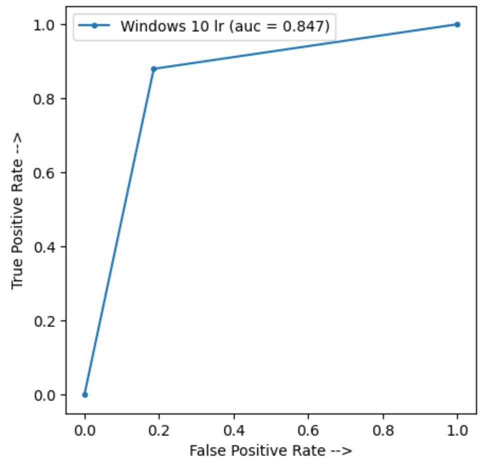

Fig 5.09: ROC curve of Logistic Regression

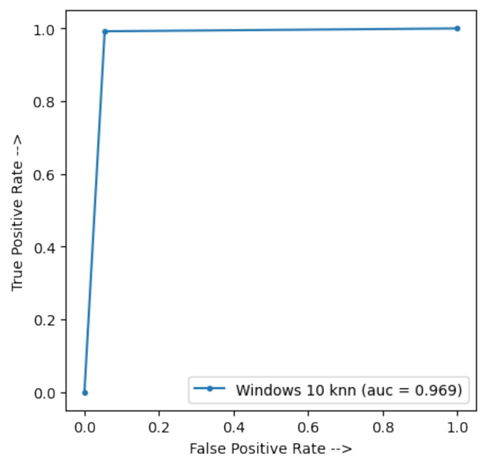

Fig 5.1: ROC curve of KNearest Neighbour

the exception of Logistic regression Classification, all of the models have produced a nearly ideal ROC curve. Again, the best ROC curve was produced by K-Nearest Neighbor(KNN).

B. XAI Obeservations

We ran LIME on all our machine-learning models and then plotted the correlation bar charts of 200 observations. It is important to note that LIME by default only shows the 10 most correlated features per observation. The observations included the actual value, the predicted value, and the residue. The residue was the difference between the actual value and the predicted value. So, if the actual value was 1 and the predicted value was 1, then the residue would be 0 and vice versa. Features that are on the left (colored blue) are denying the prediction while features that are on the right (colored orange) are approving the prediction.

We took a subset of observations and then looked at multiple instances for when the model accurately predicted 1 and 0. Then we looked at the ten most correlated feature correlations for those instances. We made sure to look at multiple instances of correct observations and made sure to look out for any biased features(features whose correlation changed little to none, even when the output has changed). Below are some of the lime observations for each model.

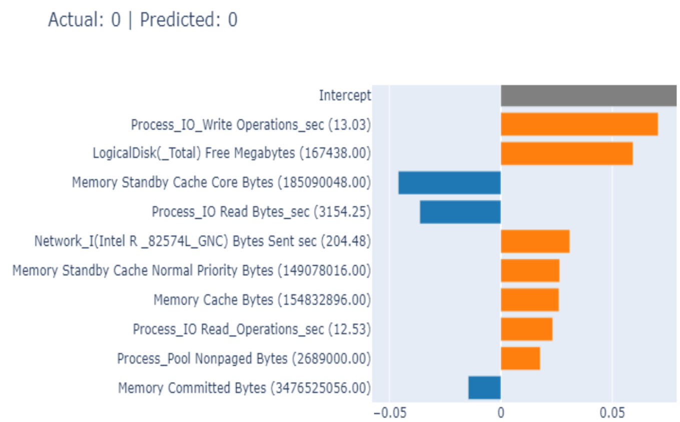

Fig 5.11: Lime Graph of Decision Tree Prediction 0

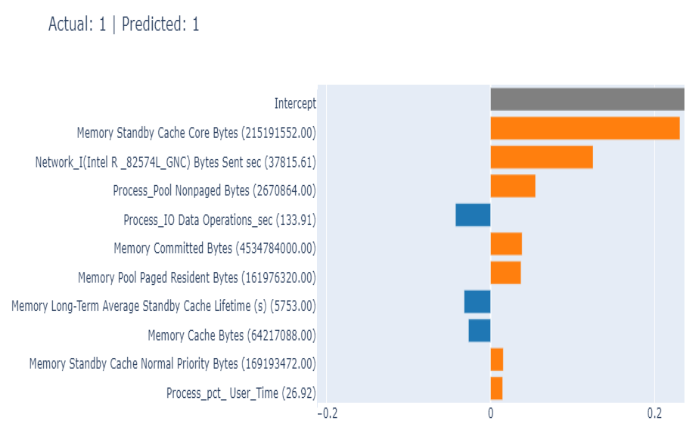

Fig 5.12: Lime Graph of Decision Tree Prediction 1

As we can see from the figure 5.11, 'Memory Standby Cache Core Bytes', 'Process_IO Read Bytes_sec', 'Memory Committed Bytes' have a negative correlation to the predicted value 0 and hence are on the left side and in essence against the predicted value.

Whereas in figure 5.12, when it is 1, the same parameters are seen to be on the left side or the positive side. This indicates that these parameters are positively correlated to output 1.

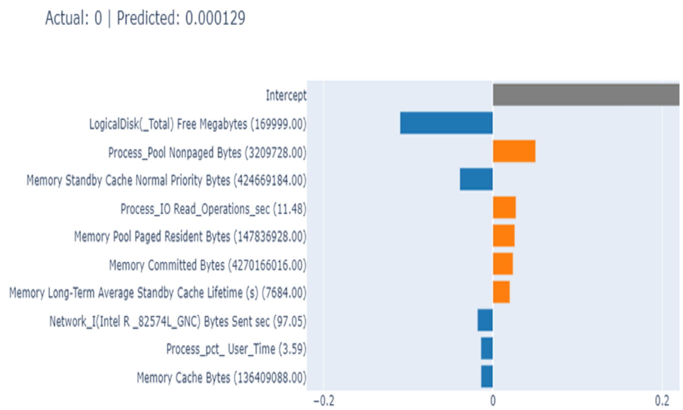

Fig 5.13: Lime Graph of Random Forest Classifier Prediction 0

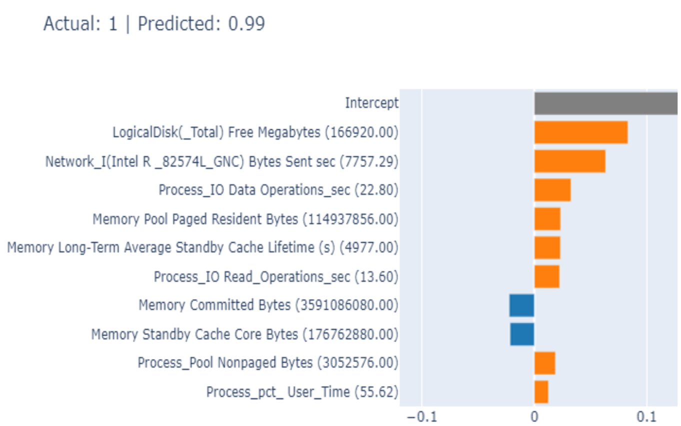

Fig 5.14: Lime Graph of Random Forest Classifier Prediction 1

From the above Figure 5.13, we can see that when it is observed 0, the features ’LogicalDisk(_Total) Free Megabytes’, and ’Memory Committed Bytes’, are some of the features that are on the left side, signifying negative correlation while ’Process_Pool Nonpaged Bytes’ can be observed to be slightly correlated positively.

But in figure 5.14, when the observation is 1, both the features ’LogicalDisk(_Total) Free Megabytes’, and ’Memory Committed Bytes’ become highly skewed to the right and thus positively correlated. But, the feature ’Process_Pool Nonpaged Bytes’ can be observed to be still positive but with a much lower correlation. Hence, it can be concluded that this feature has a slight bias.

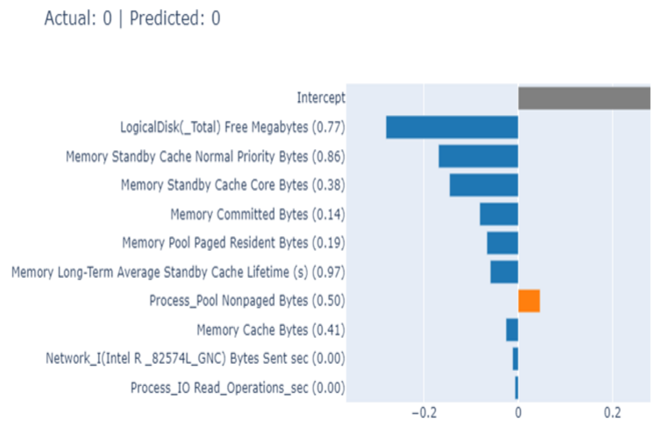

Fig 5.15: Lime Graph of KNearest Neighbour Prediction 0

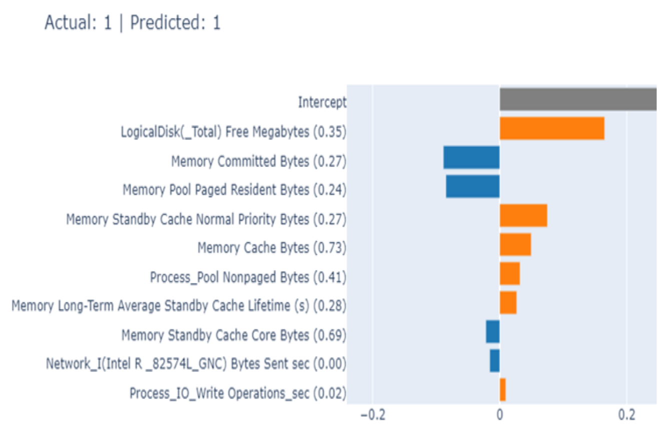

Fig 5.16: Lime Graph of KNearest Neighbour Prediction 1

Similarly in figure 5.15, for the KNN model, we can see all the features except the feature ’Process Pool NonPaged Bytes’ are all on the left side of the prediction, indicating a negative correlation.

But in figure 5.16, when the prediction is 1, it is important to note that some of the features such as ’LogicalDisk(_Total) Free Megabytes’ are now on the right side, further confirming that it is positively correlated with the predicted output 1.

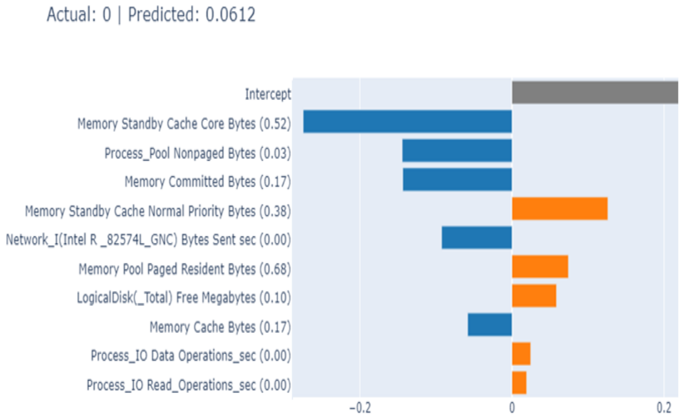

Fig 5.17: Lime Graph of Logistic Regresion Prediction 0

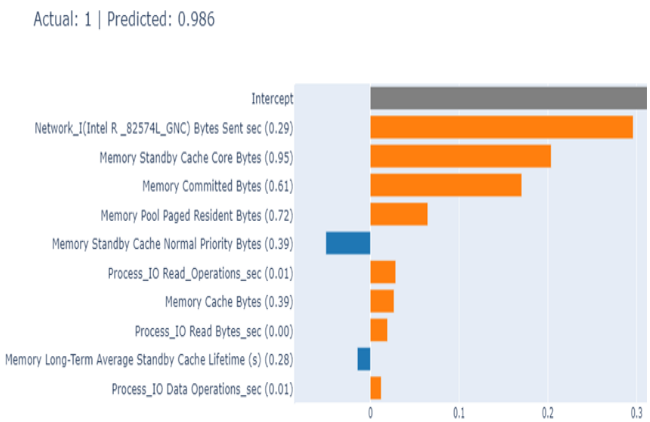

Fig 5.18: Lime Graph of Logistic Regresion Prediction 1

Again in Figures 5.17 and 5.18, a similar trend of negative and positive correlations can be seen for the model of logistic regression as well. But one thing to note is that model has given a lot of weight to a feature 'Network I(Intel R 82574L GNC)TCP APS', something all the other models have not done.

In summary, a complete picture of how each feature interacts with each model can not be painted with an XAI such as LIME, as it can only interpret local observations. But, some patterns among certain features can clearly be seen from the figures above. It is important to note that LIME only displays the 10 most correlated features for each observation. Hence, if some features were present when it was predicted to be 1, the same features will not always be present when observing the features when it is predicted to be 0.

VI. Conclusion

Machine Learning (ML) and Explainable Artificial Intelligence (XAI) are two crucial fields of research that are shaping the future of technology. ML has the potential to revolutionize various industries by providing highly accurate predictions and automating decision-making processes. However, the black-box nature of many ML models can make them difficult to interpret and trust, which is where XAI comes in. XAI aims to make ML models more transparent and accountable, which is crucial in high-stakes applications such as healthcare, finance, and autonomous vehicles. Achieving this goal is challenging, but ongoing research in XAI is making progress in creating more interpretable and explainable ML models. Future research should focus on developing methods that can balance interpretability, accuracy, and complexity, as well as addressing privacy concerns and the potential for bias in ML models.

In the report we referred to [6], the Chi-square (Chi2) method was utilized for feature selection, and the Synthetic Minority Oversampling Technique (SMOTE) was employed for class balancing. XGBoost was the most effective among all boosting algorithms that were tested, achieving an accuracy rate of over 0.979. Due to the class imbalance issue in ToN-IoT, an additional evaluation method was conducted using the SMOTE technique. Both XGBoost and kNN achieved the best result with an accuracy of 99%. On the other hand, we employed the in-built module of the Random Forest classifier and after preprocessing, our data was no longer imbalanced, so no class balancing techniques such as SMOTE were used. In our experimentation, we discovered that KNN achieved superior results in terms of accuracy (97.22%), F1 score (97.6%), and precision when compared to other models. Random Forest was second-best, with results very similar to KNN, specifically 96.05% .

Our research findings align with previous studies, making it useful for comprehending how intrusion happen in Windows 10 and providing a foundation for future versions of both newer and older operating systems. We intend to improve the accuracy of the models by adjusting the parameters further and aim for over 98% accuracy in future works. After that, an intrusion detection system (IDS) could be created based on our findings.

Acknowledgment

We would like to express our sincere gratitude to all those who have supported us throughout the completion of this thesis.

First and foremost, praise be to the Almighty Allah for granting the uninterrupted completion of our thesis.

We would like to extend our deepest appreciation to our advisor, Dr. Muhammad Iqbal Hossain, for his direction, support, and encouragement throughout the entire process. His expertise and invaluable feedback have been instrumental in shaping this research and we are truly grateful for his time and dedication.

Finally, we would like to acknowledge the contributions and extend our gratitude to our great friend and mentor, Md. Fahmid-Ul-Alam Juboraj, for his unwavering support and guidance throughout the entire process, without which our thesis would not be possible.

REFERENCES

[1] A. Saab, V. Nakhle, G. A. H. Chehade, and H. Al Moussawi, Testing and comparing the performances of windows server 2022, ubuntu 20. 04 and centos 8 under ddos attacks, 2022.

[2] H. Upadhyay, H. A. Gohel, A. Pons, and L. Lagos, "Windows virtualization architecture for cyber threats detection," in 2018 1st International Conference on Data Intelligence and Security (ICDIS), 2018, pp. 119–122. doi: 10.1109/ ICDIS.2018.00025.

[3] A. Silberschatz, P. B. Galvin, G. Gagne, and A. Silberschatz, Operating system concepts. Wiley, 2010.

[4] Z. Abou El Houda, B. Brik, and L. Khoukhi, ""why should i trust your ids?": An explainable deep learning framework for intrusion detection systems in internet of things networks," IEEE Open Journal of the Communications Society, vol. 3, pp. 1164–1176, 2022.

[5] M. Pradhan, C. K. Nayak, and S. K. Pradhan, "Intrusion detection system (ids) and their types," in Securing the internet of things: concepts, methodologies, tools, and applications, IGI Global, 2020, pp. 481–497.

[6] A. R. Gad, A. A. Nashat, and T. M. Barkat, "Intrusion detection system using machine learning for vehicular ad hoc networks based on ton-iot dataset," IEEE Access, vol. 9, pp. 142 206–142 217, 2021.

[7] A. Technologies, Akamai reports third quarter 2021 financial results, https://www.akamai.com/newsroom/press-release/akamai-reports-third-quarter2021-financial-results, 2021.

[8] Imperva, Q3 2021 ddos attacks and trends, https://www.imperva.com/blog/ q3-2021-ddos-attacks-and-trends-report/, 2021.

[9] Microsoft, Windows zerologon vulnerability, https : / / msrc . microsoft . com /update-guide/en-us/vulnerability/CVE-2020-1472, 2020

[10] Microsoft, Windows rdp bluekeep vulnerability, https://msrc.microsoft.com/update-guide/en-us/vulnerability/CVE-2019-0708, 2019.

[11] G. Kirubavathi and R. Anitha, "Structural analysis and detection of androidbotnets using machine learning techniques," International Journal of Information Security, vol. 17, no. 2, pp. 153–167, 2018.

[12] T. A. Tuan, H. V. Long, L. H. Son, R. Kumar, I. Priyadarshini, and N. T. K.Son, "Performance evaluation of botnet ddos attack detection using machine learning," Evolutionary Intelligence, vol. 13, no. 2, pp. 283–294, 2020.

[13] S. Joshi and E. Abdelfattah, "Efficiency of different machine learning algorithms on the multivariate classification of iot botnet attacks," in 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), IEEE, 2020, pp. 0517–0521.

[14] F. V. Alejandre, N. C. Cort´es, and E. A. Anaya, "Feature selection to detect botnets using machine learning algorithms," in 2017 International Conference on Electronics, Communications and Computers (CONIELECOMP), IEEE,2017, pp. 1–7.

[15] S. Wali and I. Khan, "Explainable ai and random forest based reliable intrusion detection system," 2021.

[16] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, "Explainable artificial intelligence (xai) to enhance trust management in intrusion detection systems using decision tree model," Complexity, vol. 2021, 2021.

[17] N. Capuano, G. Fenza, V. Loia, and C. Stanzione, "Explainable artificial intelligence in cybersecurity: A survey," IEEE Access, vol. 10, pp. 93 575–93 600, 2022.

[18] J. Petch, S. Di, and W. Nelson, "Opening the black box: The promise and limitations of explainable machine learning in cardiology," Canadian Journal of Cardiology, vol. 38, no. 2, pp. 204–213, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0828282X21007030.

[19] I. Tareq, B. M. Elbagoury, S. El-Regaily, and E.-S. M. El-Horbaty, "Analysis of ton-iot, unw-nb15, and edge-iiot datasets using dl in cybersecurity for iot," Applied Sciences, vol. 12, no. 19, p. 9572, 2022.

[20] U. of New South Wales, Toniot datasets, 2021. [Online]. Available: https : //research.unsw.edu.au/projects/toniot-datasets%7D,.

[21] X. Zou, Y. Hu, Z. Tian, and K. Shen, "Logistic regression model optimization and case analysis," in 2019 IEEE 7th international conference on computer science and network technology (ICCSNT), IEEE, 2019, pp. 135–139.