An Artificial Intelligence-based Intrusion Detection System

Thani Almuhairi[1,*], Ahmad Almarri[1], Khalid Hokal[1]

[1]American University in the Emirates, Dubai, UAE

Emails: 171110066@aue.ae; 171110025@aue.ae; 181120029@aue.ae

Abstract

Intrusion detection systems have been used in many systems to avoid malicious attacks. Traditionally, these intrusion detection systems use signature-based classification to detect predefined attacks and monitor the network's overall traffic. These intrusion detection systems often fail when an unseen attack occurs, which does not match with predefined attack signatures, leaving the system hopeless and vulnerable. In addition, as new attacks emerge, we need to update the database of attack signatures, which contains the attack information. This raises concerns because it is almost impossible to define every attack in the database and make the process costly also. Recently, research in conjunction with artificial intelligence and network security has evolved. As a result, it created many possibilities to enable machine learning approaches to detect the new attacks in network traffic. Machine learning has already shown successful results in the domain of recommendation systems, speech recognition, and medical systems. So, in this paper, we utilize machine learning approaches to detect attacks and classify them. This paper uses the CSE-CIC-IDS dataset, which contains normal and malicious attacks samples. Multiple steps are performed to train the network traffic classifier. Finally, the model is deployed for testing on sample data.

Keywords: Artificial Intelligence, Intrusion detection system, Machine learning, decision tree

1. Introduction

This paper explores multiple machine learning approaches to identify network attacks. To achieve our goal, we will use a dataset consisting of attacks and benign examples. We will analyze it completely to build a strong machine learning algorithm for classification and identification of the benign or attack behavior present in the given pcap file. In this process, we will compare different machine learning algorithms and judge the performance of all of them on a test-set of the given dataset and eventually choose the best performing machine-learning algorithm to classify the given pcap file as an attack or benign class. The advancement of technology and boom in the eCommerce sector has raised the concern related to security issues in different organizations worldwide. Meanwhile, the emergence of new chips and powerful computational hardware make it easy to launch an attack into any system. Previously, companies used to deploy two-layer approaches to tackle the issue of intrusion detection. Where the first layer had included intrusion prevention systems such as firewalls, while

intrusion detection systems are deployed at the second layer. But constant development of new attacks makes it impossible to

DOI:
https://doi.org/10.54216/JCIM.07.02.04

build a good preventive system while at the same time the old intrusion detection system which uses sign-based rules to identify the attacks becomes impractical. It is close to impractical to add the rules constantly as new attacks emerge. It is not time efficient and costly too, resulting in the current intrusion detection system becoming weaker day by day.

Due to many lacking in current systems, it made us think to use the power of other computer science research areas such as machine learning and quantum computing to tackle this issue. Machine learning has recently shown outstanding results in many domains like medical care, recommendation systems. Spam email detection and filtering [1] is one of the good recent practical applications of machine learning. Currently, much international research institutes research to use the power of machine learning to resolve many issues in network security and come up with new advanced and robust systems to detect new attacks automatically.

Machine learning algorithms themselves are divided into multiple branches mainly supervised and unsupervised algorithms. The supervised algorithms utilize the past data to learn the pattern present in the data. Then, based on history, they will classify the pattern. In the case of Intrusion detection systems, there is an abundance of data that comprises attacks and beginning examples. The supervised algorithms will find the difference between attacks and normal samples and will use the learned knowledge to classify any future sample. On the other hand, unsupervised learning algorithms [2] such as kmeans, learn the pattern of normal network samples and assign attacks to those samples which occur out of normal network distribution. This paper will use the supervised learning approaches and utilize the given labels in the dataset to classify the sample into attack or benign.

The main goal of this paper is to implement an artificial intelligence-based intrusion detection system to classify the data sample into a benign and attack class. To do that, we first explored many machine learning algorithms and read many research papers. Then, we find a suitable and enough sampled dataset to train a model. After training, we will evaluate our algorithms to choose the final model. Next, we create a features extraction script, which will extract required features from the given packet and pass it our classification model. Finally, we will test our systems on different pcap files having different attacks.

The rest of this paper is organized as follows: Section 2 discusses the literature review. Then, section 3 describes the background information in the topic of machine learning and intrusion detection. The developing environment, the used methodology, and the implementation details are discussed in sections 4, 5, and 6 respectively. Section 7 concludes the paper.

2. Literature review

The idea of using machine learning to build a robust intrusion detection has been presented in many past works of literature. Many research groups around the globe are constantly finding ways to improve the current intrusion detection system using the power of machine learning. "Machine learning Approach to IDS: A Comprehensive Review" [3] presented at IECA conference. This paper summarizes many important aspects of previous research on this topic. It also presents many machine learning algorithms and their performance on intrusion detection tasks. It is presented in the paper that Intrusion detection systems are improving year by year. The minimum accuracy presented in the paper was 88.0% and a maximum of 99.9%. Note, these numbers reflect the performance of machine learning algorithms on IDS tasks on a particular dataset. These datasets are recorded in a controlled environment and most of the time do not reflect actual world attacks. The paper which has achieved the highest accuracy (99.9%) [4] has used the KDD Cup 1999 dataset [5]. This dataset comprises two classes: attacks and benign. The paper has used a support vector machine (SVM) to train the classifier.

Despite some issues, many papers presented in the references used the KDD Cup 1999 dataset. It comprises around 5 million data points and each data point is either an attack or benign class. The attack class itself contains various types of attacks. The size of the dataset and various attacks made this dataset popular. Irrespective of the size and diversity of the dataset, it has

DOI:
https://doi.org/10.54216/JCIM.07.02.04

presented attacks that are easily detectable and making it completely out of date. The training of this dataset may give the highest accuracy but in the practical world, it will be unable to predict the new attacks. In addition, the absence of ample amounts of benign data and unrealized attacks make this dataset not usable for training models for production (Real World) purposes.

3. Background Information

Machine learning seems promising to be used for building robust intrusion detection research. However, this robustness has some challenges. Firstly, to train a robust and good machine learning model, we need to have a large amount of labeled data. A dataset like KDD cup 1998 or any other, does not contain all possible attacks. In addition, many threatening attacks happened in the real environment, on large companies. These companies often do not publicize their dataset due to privacy concerns Some researchers suggested anonymizing the data to not make any personal information public, encrypt the personal data present in the dataset, but sometimes it majorly affects the quality of data and ultimately lead to a bad model.

Historical rule-based intrusion detection systems have a database where all rules were saved. So, whenever a new attack occurs, they update the rules to identify the new attacks. In the case of machine learning-based systems, it is very hard to find out when it needs to update the training data to build a new model. We can be alerted about it when a new attack occurs, but no one wants to be attacked first to find out the time to update the model with new training data. Also, the scalability of the proposed system must be considered before deploying the learning-based algorithms.

Eventually, it is safe to say that machine learning-based systems are challenging when it comes to testing. We believe that machine learning systems are a better solution against the traditional rule-based approaches and help to detect new attacks, but it is sad to see that almost all papers presented results based on datasets of control environments which make it hard to see whether these algorithms work in the real world or not. In other words, we are hoping for a system to detect new attacks which it has never seen nor knowledge about. In summary, there is a higher probability of errors occurring when these systems are deployed in the real world than compared to other domains where machine learning has shown promising results. Although we are not sure on which attacks it will fail so the best

Implementing a machine learning algorithm is not very easy to do and has a couple of challenges. The first challenge is to gather a proper and appropriate dataset. The dataset must be relevant to the problem being solved. Moreover, there has to be abundant data so the model can train and learn efficiently. Not having enough data, it takes a lot of data for most of the algorithms to function properly. For a simple task, it needs thousands of examples to make something out of it, and for advanced tasks like image or speech recognition, it may need many more examples. Poor training data is also another ML challenge. If your training data has lots of errors, outliers, and noise, it will make it impossible for your machine learning model to detect a proper underlying pattern. Hence, it will not perform well. Feature engineering, the credit for a successful machine learning paper goes to coming up with a good set of features on which it has been trained, which includes feature selection, extraction, and creating new features.

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize. Underfitting is the opposite of overfitting. It happens when our model is too simple to learn something from the data. To overcome this, we can tune the parameters, choose a better model, reduce the constraints and we can also train in more data. As previously mentioned, the data used might contain personal data such as IP addresses, so it is preferable to remove such data. Also, there are multiple types of network attacks that classify as harmful, but this is a challenge since there are many variations, we need to be able to create a model with high performance to avoid compromises in the network.

DOI:
https://doi.org/10.54216/JCIM.07.02.04

Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model. Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of your model. The data features that you use to train your machine learning models have a huge influence on the performance you can achieve. Irrelevant or partially relevant features can negatively impact model performance. Feature selection and Data cleaning should be the first and most important step of your model designing.

3.1 Networking Theory

TCP/IP Reference Model is a four-layered suite of communication protocols. It is named after the two main protocols that are used in the model, namely, TCP and IP. TCP stands for Transmission Control Protocol and IP stands for Internet Protocol. The four layers in the TCP/IP protocol suite are shown in Table 2.2. Each layer has its functionalities. Network attacks are mostly associated with application and transport layers.

Table 1. Different layers of Network information

Layer	Function	Protcols
Application Layer	Interacts with the application program	FTP, HTTP, SMTP
Transport Layer	Transports data across the network	TCP, UDP
Network Layer	Defines the routing of the packets in the network	IP, ICMP
Link Layer	Describes how to physically send data throught the network	Ethernet, ARP

At the transport layer, there are namely 2 protocols that are used for data transmission: TCP & UDP. Each has its pros and cons. TCP is much more reliable but a slow mechanism whereas UDP is a very quick but not very reliable mechanism. At the application layer, a few of the protocols possible are Telnet/SSH for remote terminal access, FTP/TFTP for file transfer & SMTP for email services.

4. Development environment

A few goals which were necessary for this paper to be a success were:

- Implementation of a Machine Learning algorithm and evaluate its performance.
- Develop a whole interface for Intrusion Detection, which assesses the pcap file and checks the file for possible attacks.
- Evaluate the detection accuracy for the intrusion when applied to real-time network traffic

Python seemed the most appropriate language of choice for the paper; its use is common for machine learning programs due to the availability of many libraries, which makes implementing easier. Although it meant that we had to spend some time at the start familiarizing ourselves with the language, it was the right choice for this paper. One of the aspects that makes Python such a popular choice in general, is its abundance of libraries and frameworks that facilitate coding and save development time. Machine learning and deep learning are exceptionally well catered for.

For the initial learning part of the paper, I used Jupyter Notebook. Jupyter Notebooks allow for efficient testing since you can run the cells individually, and so do not have to run the whole program each time you make a change. It is beneficial when performing machine learning since you do not have to run the pre-processing each time. Also, the notebook provides visual outputsso that you can see graphs or tables directly. Jupiter redesigned architecture that allows the notebook to speak dozens

DOI:
https://doi.org/10.54216/JCIM.07.02.04

of programming languages - a fact reflected in its name, which was inspired, according to co-founder Fernando Pérez, by the programming languages Julia (Ju), Python (Py) and R.
As for version control, Git and GitHub were used as it is almost a necessary thing to have in huge papers.

4.1 Libraries/Packages used

For processing the dataset, the libraries used in our paper were mainly NumPy & Pandas. NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays. Pandas is a fast, powerful, flexible, and easy-to-use data analysis and manipulation tool, built on the python programming language.

For the machine learning part, we used Scikit-learn, also known as Sklearn. Scikit-learn [6] is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy, and Matplotlib. Since we did not want to write code from scratch, we used this library for model generation and tuning.

And for the information retrieval for our IDS system, the Scapy library was used. Scapy is a library made in Python, with its command-line interpreter (CLI), which allows to create, modify, send and capture of network packets. It can be used interactively through the command line interface or as a library by importing it into Python programs.

4.2 Dataset Utilization

Network traffic can be analyzed mainly into two categories: flow-level data and packet-level data. Flow level data when analyzed provides a high-level overview of the network activity performed on the network. Whereas, when analyzing packet-level data it captures the actual packets that were carrying the data and therefore these provide more information regarding whether a malicious attack is probable. But, even though there is more data in these than flow level data, the information would be too large for the processing and monitoring in an intrusion detection system. Hence, for our paper flow level data would be a much more appropriate choice.

For training our model, we used the CICIDS2017 dataset. It is a labelled dataset that contains benign and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). It also includes the results of the network traffic analysis using CICFlowMeter with labeled flows based on the timestamp, source, and destination IPs, source and destination ports, protocols, and attack. It has nearly 3 million data points, 80 features(flow-level data), and has 14 types of various malicious network attacks. The CICIDS2017 dataset comes from a research paper [7]. When presented it was aimed to solve the issues with other IDS datasets such as the KDD99. One of the reasons that CICIDS2017 is a good choice is because it was up to date and was proposed by the Canadian Institute of Cybersecurity.

5. Methodology

This section will cover the necessary steps for shaping the dataset for training. In addition, I will cover the approach which I followed to implement the machine learning algorithms as well as Intrusion detection systems. Finally, I will explain the evaluation methods to choose a robust algorithm.

DOI:
https://doi.org/10.54216/JCIM.07.02.04

5.1 Data Preprocessing

The used dataset was available in a raw form. To use it for training the machine learning algorithm, we need to perform certain steps. We call these steps Preprocessing. It contains the deletion of null values, scaling and normalization, removing redundant information. These processing steps ensure that training will not be influenced by errors present in the dataset. The dataset consists of around 80% benign samples and the remaining percentage represents different types of attacks. Some attacks are present in abundances like DoS and DDOS compared to Heartbleed and Web Attacks. Because of less representation of these attacks in the dataset, we had to drop these rows. These attacks will not contribute much to the training but they can worsen the classifier by adding some noise

We have evaluated our models in 3 different ways. First, We treated every class of attack as an individual class with the Benign class in training. Secondly, We have grouped similar attacks into one group. Lastly, I have combined all attacks under the umbrella of an attack class which resulted in only 2 classes for classification: Benign and attack. Evaluation models on 3 different combinations will inform us of a better approach or combination to tackle this problem.

The missing values in the dataset can be troublesome for models. These missing values can be occurred by different errors like in the recording of data or when extracting the particular features. Luckily, the dataset is huge enough, so we can remove these missing values which include NaN, Inf, or Null. Removing these rows will not affect the quality of our model at all.

After training, we also need to know how good our trained model is. So to evaluate, we split our dataset into 3 portions before training: training, validation, and testing. We use a training dataset to train the dataset. The validation set is mostly used for hyperparameter tuning and model initialization. The testing set is used to judge the performance of our trained model and it is never used during training. I have split the dataset into 60% training, 20% validation, and 20% testing. Due to the difference in the number of labels between classes, unbalanced, we preferred to use stratified sampling. Unlike random sampling, Stratified sampling splits the labels equally into equal numbers of classes. This ensures the presence of enough minority classes in each split to avoid unbalanced classification problems.

In machine learning modeling, the redundant columns do not provide any information during training; they are either being ignored during training or add a random noise. In any way, they just increase our computational resources and time. So, I have checked all redundant columns and dropped them before training.

Features in the dataset vary in scale. Some of them are present in ranges of 0 and some are quite big numbers like 10000. We must ensure that every feature in the dataset must be in the same range. There are multiple models which especially generate poor results if feature normalization is not done. They prefer to give high importance to those features which have higher values compared to lower ones. To do normalization, we can leverage min-max normalization present in the sci-kit-learn library to make all features in one range. Min-max brings all the features in the range of [0-1]. It is also easy to calculate and apply to a dataset. It just finds the minimum and maximum then scales the whole dataset. It uses the following equation.

DOI:
https://doi.org/10.54216/JCIM.07.02.04

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

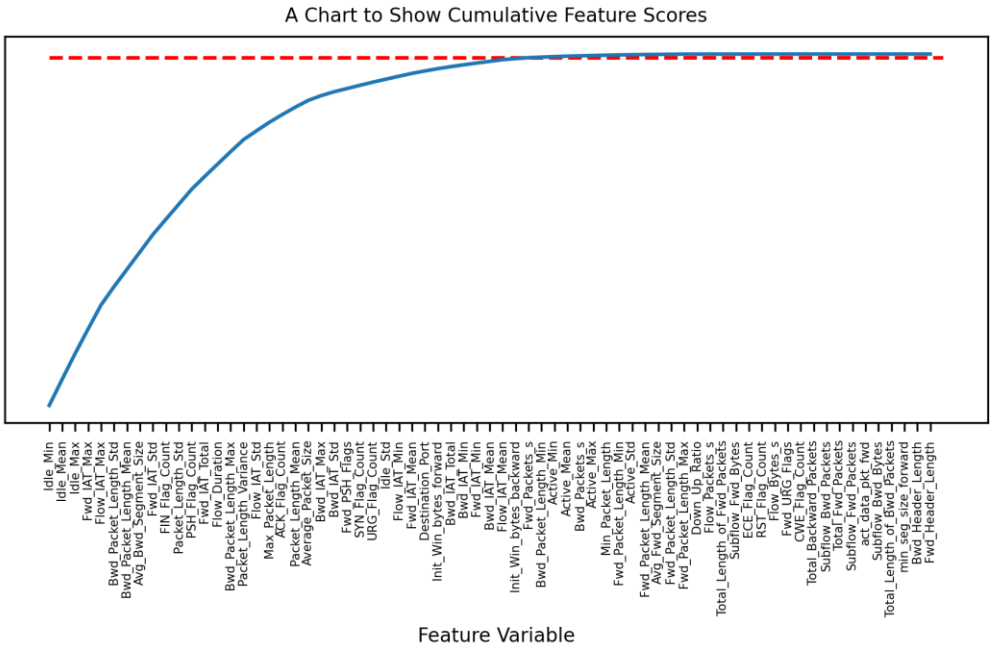

Figure 1: Feature selection scores

5.2 Feature Selection

In the dataset, we have a lot of features that do not contribute much to the output prediction. These features instead of adding the value in the prediction increase the computational time. We will select useful features to reduce the dimensionality of the dataset. The chi-squared test is one of the useful methods for feature selection. It tests how much independence a feature has against the class label. The independence between a class label and features decides the selection of the feature. If they are independent, then that feature will not be considered and discarded during feature selection. There are handy functions available in the sci-kit-learn library to choose the top 'K' number of dependent features from our dataset basis on corresponding feature selection tests like Chi-squared. In this way, we get the top 'K' dependent features.

We have added figure 1 to show the cumulative score for features. This figure shows the sorted features scores, and these are obtained from the chi-square test present in the sci-kit-learn library. The dotted red line is set as a threshold to discard features. We have set its value to 99%. So, any feature above that threshold will be discarded. In other words, it contributes close to no value in model prediction. After applying the threshold, we end up with 40 dependent features. This number is obtained using

DOI:
https://doi.org/10.54216/JCIM.07.02.04

the sci-kit-learn SelectKBest library. The chosen 40 features mostly represent some statistics like mean, sum, or max for a particular packet property like packet length or inter-arrival time between packets. Table 2 illustrates the chosen features with their summary.

Table 2: Selected features and their summary

Feature	Description	Feature	Description
dest port	Destination port number	fwd PHS flags	PSH (push) ag count (forward direction)
flow duration	Duration of ow in microseconds	fwd packets s	Number of forward packets per second
bwd packet len max	Maximum packet length (backward direction)	max packet len	Maximum packet length
bwd packet len min	Minimum packet length (backward direction)	packet len mean	Mean packet length
bwd packet len mean	Mean packet length (backward direction)	packet len std	Standard deviation of packet length
bwd packet len std	Packet length standard deviation (backward direction)	packet len var	Packet length variance
flow IAT mean	Mean packet inter-arrival time	FIN flag count	FIN (nished) ag count
flow IAT std	Standard deviation of packet inter-arrival time	SYN flag count	SYN (synchronisation) ag count
flow IAT max	Maximum packet inter-arrival time	PSH flag count	PSH (push) ag count
flow IAT min	Minimum packet inter-arrival time	ACK flag count	ACK (acknowledgement) ag count
fwd IAT total	Total packet inter-arrival time (forward direction)	URG flag count	URG (urgent) ag count
fwd IAT mean	Mean packet inter-arrival time (forward direction)	avg packet size	Average size of a packet
fwd IAT std	Standard deviation of packet inter-arrival time (forward direction)	avg bwd segment size	Average size (backward direction)
fwd IAT max	Maximum packet inter-arrival time (forward direction)	init win bytes forward	Number of bytes sent in the initial window (forward direction)
fwd IAT min	Minimum packet inter-arrival time (forward direction)	init win bytes backward	Number of bytes sent in the initial window (backward direction)
bwd IAT total	Total packet inter-arrival time (backward direction)	active min	Minimum time a ow was active before becoming idle
bwd IAT mean	Mean packet inter-arrival time (backward direction)	idle mean	Mean time a ow was idle before becoming active
bwd IAT std	Standard deviation of packet inter-arrival time (backward direction)	idle std	Standard deviation of time a ow was idle before becoming active
bwd IAT max	Maximum packet inter-arrival time (backward direction)	idle max	Maximum time ow idle before becoming active
bwd IAT min	Minimum packet inter-arrival time (backward direction)	idle min	Minimum time ow idle before becoming active

6. Implementation of Classification

First, we have completed dataset preprocessing which included cleaning data, normalization, and other steps. Then, we have selected the best features out of all features present in the dataset. Now we have a dataset ready for training and evaluation [8].

6.1 Implemented Models

We have explored many algorithms and ended up with some chosen algorithms. We have implemented 3 different algorithms and the explanation behind the preference of using these algorithms for present tasks is discussed in brief below.

6.1.1 Decision Tree

The decision tree is a very intuitive learning algorithm and debugging is also understandable. That's why it is always considered to be used first in classification problems and due to this, we started with the decision tree. The implementation is also straightforward and doesn't have so many hyperparameters. Unfortunately, decision trees can result in an overfitted model on some datasets. But it still surpasses the problem of normalization and scaling. Decision trees can produce good performance even on a very unscaled features dataset. Decision tree created nodes and based on features it adds conditions on each node where the several conditions on various nodes lead to different labels. At every node, the Decision tree splits the dataset randomly and evaluates the split using the Entropy or Gini impurity.

When all examples correspond to the same class or there is a little bit of randomness then Gini impurity outputs a value between

DOI:
https://doi.org/10.54216/JCIM.07.02.04

zero. On the other hand, it outputs a large value when there is a lot of randomnesses which means there is an equal number of examples of each class [9]. We want the Gini impurity to output a minimum value, so we get a perfect split.

6.1.2 Random Forest

The decision tree is simple and intuitive, but it has some problems like overfitting. These problems were addressed and improved in the shape of the Random Forest algorithm. Random Forest is the most used algorithm in research and competition. The random forest inside of relying on one decision tree for prediction, it uses multiple decision trees trained on many subsets of features. The final output will not be influenced by nodes but it will always be a prediction made by the majority of decision trees. In this way, it deals with problems like overfitting and outliers.

6.1.3 Naive Bayes

The naive Bayes algorithm works using conditional probability and it is one of the most commonly used algorithms. Naive Bayes was once ranked to be one of the best working algorithms for spam email detection. As our task is close to the classification of traffic into benign and attack, then it is worth a try to use the Naive Bayes algorithm for the intrusion detection system. In general, naive Bayes assumes features present in the dataset must be independent such that changing of one feature should not influence other features except the label column. We already know that there might be some features present in our dataset which are not independent, but we cannot just drop them. Because these features might be having information which is necessary for a good model training. So, at least we are not hoping for great results to get from the Naive Bayes algorithm.

6.2 Assessment

Now, we have implemented 3 different algorithms after performing multiple preprocessing steps on the dataset. Now, this section will cover the designing of those algorithms and importantly the results of these algorithms [10]. These evaluation numbers will give us a good indication of choosing a final model for our intrusion detection systems.

6.2.1 Evaluation of Different Learning Algorithms

I have used multiple evaluation metrics to compare the result of our machine learning algorithms. These evaluation metrics will compare different performance measures of our models and will help us to choose the most capable final model for deployment.

Initial Testing Results

As we discussed in the previous sections, we have split our dataset into 3 portions and we will use the validation portion of our dataset for initial testing of our methods.

We already have the label for our target prediction so all of the algorithms that we have used in making intrusion detection systems are supervised learning algorithms. So, we have used four evaluation metrics: Accuracy, precision, recall, and F-score for judging the performance of our models.

Figure 2 demonstrates the performance of every supervised machine learning algorithm used for modeling each metric. The graph clearly illustrates that accuracy is not a good measure as it does not count the unbalanced dataset, so it ultimately shows the unrealistic number. For example, it shows the Naive Bayes algorithm is around 86% accurate but seeing its recall score gives an impression of not believing the accuracy number only. That's why, for classification, it is always recommended to compare the performance of the model on many metrics and choose

DOI:
https://doi.org/10.54216/JCIM.07.02.04

the model after considering all the results. In the case of the Naive Bayes algorithm, recall tells us that the naive Bayes is only predicting the correct label 1 in 5 times which means it is not a good choice for the intrusion detection system as it misses a lot of traffic attacks during the prediction. [11]

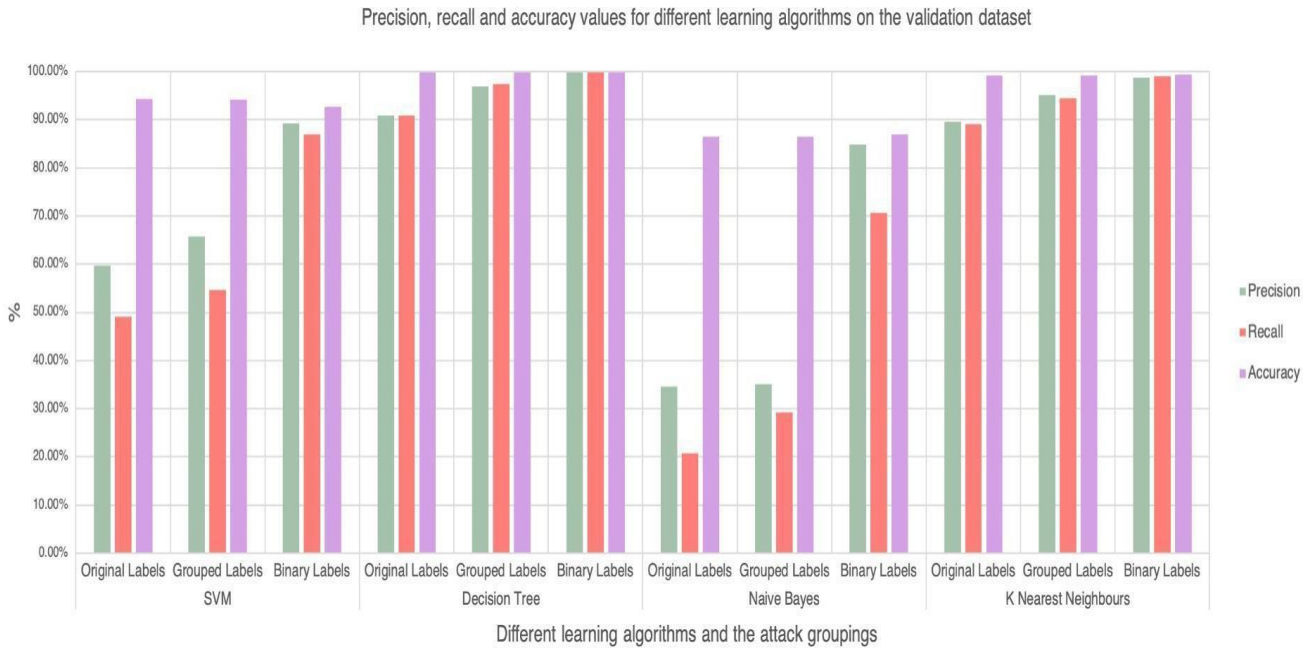

Figure 2: Experiments result of 4 different algorithm and their evaluation metric results

The jump in the percentage of recall and precision can be witnessed as we group many attacks into one class. This results in better performance of the classification between benign and malicious attacks compared to individual attacks labels.

It is clear from the graph that due to independent features assumption Naive Bayes algorithm was made. The naive Bayes has retained the consistent worst performance in all of the defined group categories. We assumed every feature in the dataset will be independent and the correlation between each of them will be zero as correlation shows the changes when we made in one feature affect the other features in the dataset. Unfortunately, this assumption does not hold in our dataset. We have some features where the independence condition is valid, for example, we have "variance of packet length" and other features like "standard deviation of packet length". Both features are completely dependent as we can derive variance from standard deviation and vice versa.

The SVM has shown promising results in earlier works on similar network security tasks and overall its ability to work against outliers and bias made us consider it for the implementation of

intrusion detection systems. SVM has achieved around 88% accuracy on the UNSW-NB15 [12]. We have achieved around 94% accuracy which is far higher than mentioned in the previous paper, the task is the same, but a difference is present in the dataset.

Nonetheless, the decision tree has shown promising results on our dataset. The graph also shows good results produced by KNN. So, it is good to say these two models can give higher accuracy along with good recall and precision for intrusion

DOI:
https://doi.org/10.54216/JCIM.07.02.04

detection systems.

6.2.2 Model Refinement

We have chosen two best performing classifiers among 4 machine learning algorithms. In this section, we will analyze them in more detail to choose a model for our final deployment.

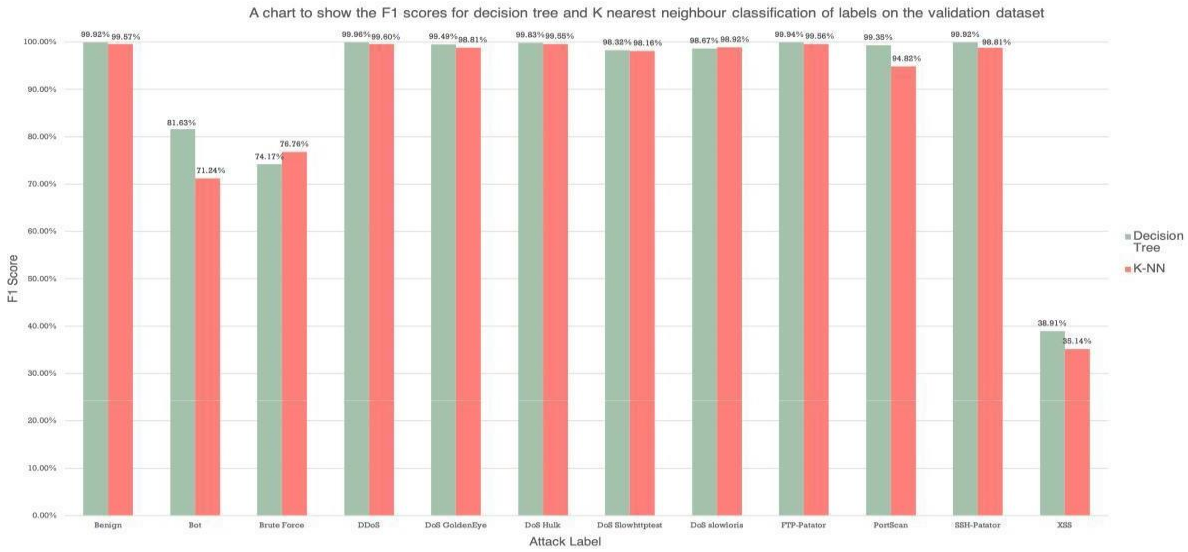

Figure 3: Decision tree and KNN f1 scores

6.2.2.1 Decision Tree vs K Nearest Neighbours

In this section, we will compare both algorithms because of their high accuracy in the intrusion detection dataset. We will compare them individually at the start and later we will compare both sides by side. It is evident from Figure 3 that the Decision tree has managed to get a high F1 score compared to KNN in almost all labels. KNN only managed to get high F1 in brute force and DoS slow loris. So, it is safe to assume that the Decision tree can be chosen for the final model because of its ability of prediction and high recall, and high precision which resulted in a high F1 score.

Some classes like Cross-site scripting have a smaller number of examples, which resulted in bad performance from both classifiers and the classifier finds it hard to predict them compared to others. It includes examples of Cross-site scripting and brute force attacks which were not

correctly classified most of the time by both high-performing classifiers. All these classification results are shown in Figure 3.

Almost all the features which are included in our prediction exist in the transport and network layer which can be troublesome for the attacks that rely on features of the application layer. Because of this, both best performing models may not be able to predict attacks like Cross-site scripting attacks during inference. If we have features like Script functions number and total references to JS-based files, we could get the URLs from these features and use them for training and prediction. It has been done in the past and achieved good results using algorithms like Naive Bayes. Since this dataset has not had these features so our classifier will not be accurate in terms of the prediction of these attacks.

The other important factor we should consider is the training and testing time. Training time is not that important, and we can

DOI:
https://doi.org/10.54216/JCIM.07.02.04

afford large training time. However, the classification of individual packet/traffic must be fast in an Intrusion detection system. In both scenarios, users also prefer timing should be less but as a user can wait for a model to be trained before it is deployed so, less testing time is always preferable. Table 3 provides a summary of the training and classification timing of both decision tree and K-Nearest Neighbor algorithms. The Decision tree passes the example input across nodes and gets an output that's why it is faster in both training and testing. On other hand, KNN first finds the closest K neighbors of given input by calculating a specific distance and then decide the class of output, resulting in a high training and testing time.

Table 3: Training and testing times for both algorithms

	DecisionTree			K-NN		
	Original	Grouped	Binary	Original	Grouped	Binary
Training Time (s)	105.93	100.01	92.33	2,306.96	2,507.44	2,457.47
Classi cation Time (s)	0.73	0.23	0.16	1,963.92	2,064.91	2,008.67

6.2.2.2 Decision Tree vs Random Forest:

The decision tree is chosen to be the most effective, highest accurate algorithm among all the algorithms we have implemented. The power of its prediction and less timing made it a clear winner over other algorithms. There is a more robust and advanced version of the decision tree
, Random Forest. Instead of using a single decision tree to perform the classification, the random forest used multiple decisions and split features among trees randomly. In the end, it decides the output label by counting most votes to a particular class by all built

DOI:
https://doi.org/10.54216/JCIM.07.02.04

individual decision trees. It is also effective against overfitting. In addition, decision trees also suffer from the problem of outlier data. The random forest also fixes this issue by having multiple decision trees.

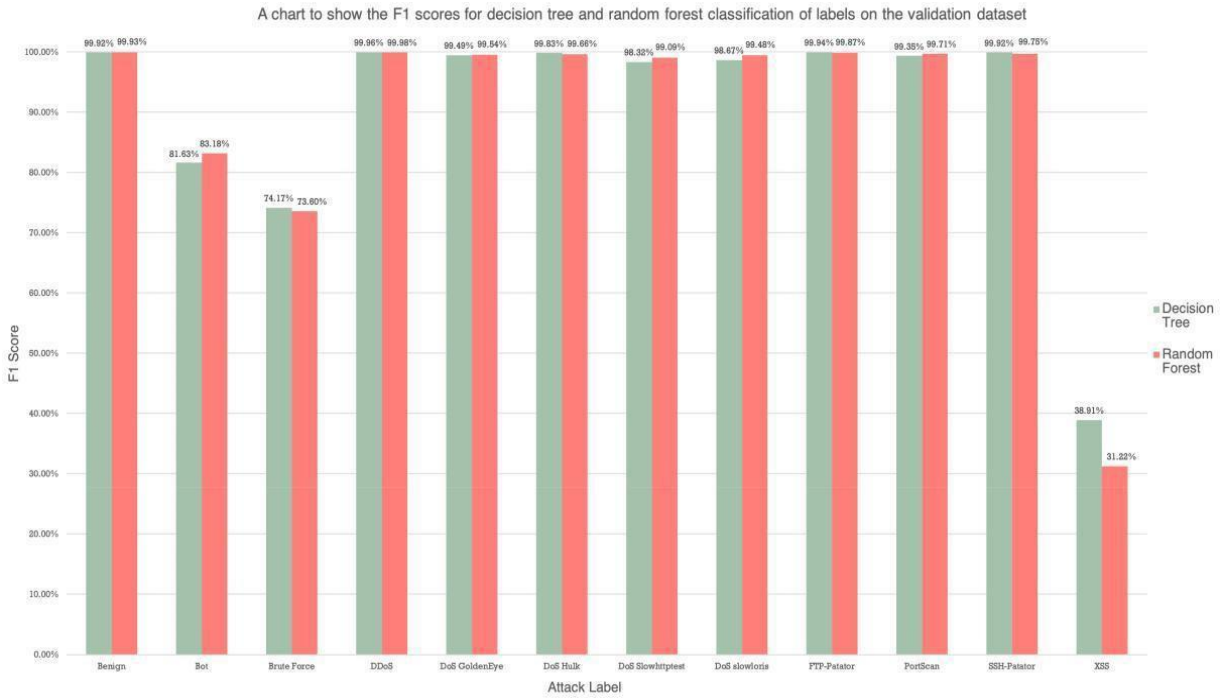

Figure 4: F1 scores of Random Forest and Decision Tree

Table 3: Testing and Training time of RF and DT algorithm

	DecisionTree			Random Forest		
	Original	Grouped	Binary	Original	Grouped	Binary
Training Time (s)	110	10	92	77	78	70

DOI:
https://doi.org/10.54216/JCIM.07.02.04

	5 . 9 3	0 . 0 1	. 3 3	. 3 6	. 2 6	. 1 9
Classi cation Time (s)	0 . 7 3	0 . 2 3	0 . 1 6	0 . 9 9	0 . 9 0	0 . 6 5

The results in Figure 4 are mostly similar because the random forest is just using many decision trees instead of only one for prediction. Still, the random forest has been able to achieve a higher F1 Score and consequently performed well on many labels. Out of 12 attacks, the decision tree only managed to get a higher score or similar score in 5 labels, and random forest managed to get a higher F1 score in the remaining labels. In binary and group labels, a random forest maintained its higher F1 score compared to a decision tree [13].

In the timing and computation comparison of table 3, they both have similar results. The random forest has a faster training time, but decision trees generate a prediction on inference faster. Nonetheless, they both share the same property, so this does not matter much.

It will not be wrong choosing a random forest because of its higher and better prediction power over the decision tree. The random forest is also faster to train and hence provide much more option to train many different hyperaerated models for testing. Hence, due to all the reasons, we decided to choose a random forest as the final model for an intrusion detection system.

6.2.2.3 The Final grouping of labels

As we have mentioned before, we have done our training experiments with three groups: All labels, categorization of labels, and binary labels. After experiments, we now chose one of them for our final classifiers. Overall, as we categorized our labels into fewer classes, we witnessed the jump in our evaluation metrics (recall, precision, and F1-score). So, it is safe to say that we should be using binary labels, classifying examples only into benign and attack classes. This way we can achieve better accuracy in intrusion detection systems with overall good computation and testing time.

Due to less number of examples of classes like brute force and Cross-site scripting, they were not predicted accurately by random forest and resulted in a low F1 score. Despite this, when these attacks combined under the umbrella of one class, web attack class, the F1 score has increased significantly. The combined reported score jumped to 98%. Previously, around 80% of examples of Cross-site script classes were classified as brute force. Similarly, 23% of brute force classes were predicted as Cross-site scripting attacks. Ultimately, combining both into one class got rid of these mistakes and increased the overall percentage of correct prediction, and improved the overall model also. Figure 5 illustrates the prediction of each class.

DOI:
https://doi.org/10.54216/JCIM.07.02.04

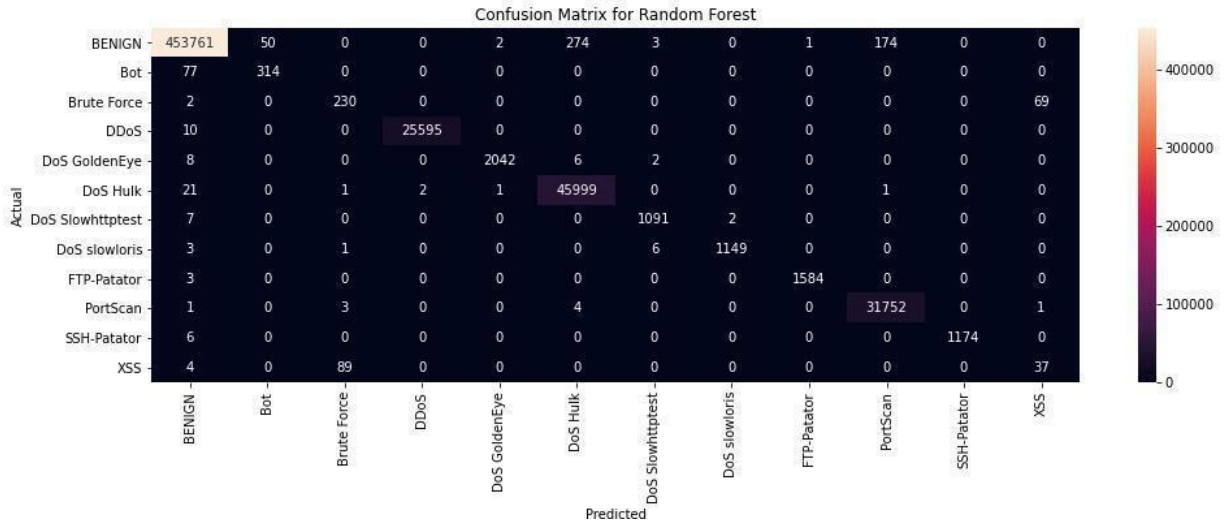

Figure 5: Confusion Matrix of each prediction

This also indicates that because of the unavailability of required features (Application layer information) for classifying the attacks Cross-site scripting and brute force into their corresponding classes, we were not able to predict them individually but when combined in one category class then we gathered information to predict them under the shadow of one attack.

We also have grouped all four DoS attacks into one class, to increase the overall score. Although without combining them they were classified accurately as we have witnessed in the case of web attacks, we can always get an increase in overall prediction which means our intrusion system performance increases to detect attacks.

6.2.2.4 Parameters Tuning

Every machine learning model has some parameters which can be tuned to get a better model. These parameters are known as hyperparameters. These parameters can increase or decrease the power and performance of a model. This is a complete research area in the field of machine learning and representation learning. A simple example is a linear regression, which takes a Learning rate hyperparameter. The learning rate can take an infinite number of possible values and finding the correct one can be troublesome. Linear regression is just a basic example. There are algorithms like neural networks which can take a lot of hyperparameters and finding the correct one requires a lot of time and computation and of course, the result can also be fruitful. In the case of our random forest model, we have several hyperparameters as well and we will fine-tune them to find the correct one for our final model prediction. The following is the list of the hyperparameters.

N_estimators: It defines the number of trees to be used in random forests. In the case of the decision tree, it is set to be 1.
Max_depth. It tells the random forest how many splits should be done.
Min_samples_split: We split the dataset at every node so it tells the random forest to check that the number of samples is greater than the minimum number for splitting.
Min_samples_leaf: The bare minimum of samples that must be present at a leaf node. A split point will be considered if it leaves at least Min_samples_leaf training samples in each of the left and right branches, regardless of depth. This has the potential to smooth the model, particularly in regression.
bootstrap: When creating trees, whether bootstrap samples are used. If true, each tree is constructed using the entire dataset. This is followed by a look at the number of estimators metric. Our ensemble evaluates a set number of decision trees. When n estimators = 800, the peak is at its highest. This is not the number of estimators we chose. When n estimators = 800, the classification time would be roughly 50 seconds when n estimators = 800. A randomized search is used to optimize the

DOI:
https://doi.org/10.54216/JCIM.07.02.04

remaining hyperparameters. For each hyperparameter, this search technique generates a grid of alternative values. It then performs training and testing on a set of random hyperparameter values, assigning a score to each combination. To implement parameter optimization, we use the sci-kit-learn library's RandomizedSearchCV, and then call best params to return the parameter setting that produced the best results on the hold-out data.

6.2.2.5 Building The Intrusion detection system:

To use the trained random forest model, we must have some interface to pass our data. Along with an interface, we also need a script that will extract necessary features from the packet and retain only features which are required by our model for prediction [14]. We have written a script that extracts useful information from the packet. The second script will convert that information into features that are required by our model. These features will be passed to models for predictions. We have also developed a Flask web application. We have provided an interface where you can pass the path of the PCAP format file, then it will process and let you know whether the file contains any malicious attacks or not. We have used some packets from the other dataset including DDOS, Botnet attacks. We have also tested it on a normal data packet, recording network traffic using Wireshark.

The machine learning flowchart contains all the steps being done in making a machine learning model for intrusion detection systems. This flowchart represents the steps being done after training a machine learning model. These steps are the final steps when we run our system on a local machine.

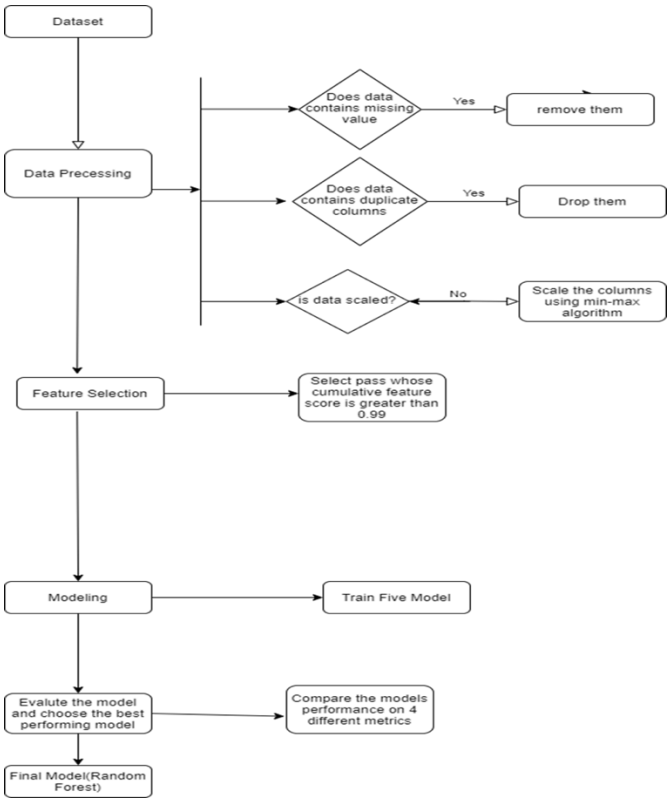

Figure 6: Flowchart for the proposed model

7. Conclusion

DOI:
https://doi.org/10.54216/JCIM.07.02.04

In this paper, we have used machine learning as a solution to be able to detect intrusions in network traffic. Results that were achieved can be called satisfactory as the model can effectively detect intrusions in the network. The proposed model has achieved a high F1 score and accuracy. The achieved results when compared to values of other models proposed in previous papers can be interpreted as high competitors to those models. The system also shows great results on previously unseen attacks and for attacks that were not detected correctly, the reasons and causes were discussed. We also addressed possible future works that can be implemented to obtain higher performance and achieve better results.
Multiple paths can help us achieve higher efficiency and greater results in detecting intrusion in-network data for smart applications by applying different machine learning and deep learning algorithms.

References

[1] E. G. J. S. B. H. C. A. O. A. a. O. E. A. Dada, "Machine learning for email spam filtering: review, approaches and open research problems," Heliyon 5, no. 6, 2019 .

[2] J. MacQueen, "Some methods for classification and analysis of multivariate observations," In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, 1967 .

[3] M. Dua, "Machine Learning Approach to IDS: A Comprehensive Review," In 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 117-121. IEEE, 2019 .

[4] Z. Liu, "A method of SVM with normalization in intrusion detection.," Procedia Environmental Sciences 11 , 2011 .

[5] K. C. 1. Dataset .Machine Learning Library - s.-l. [. A. 2021] .

[6] R. a. S. B. Panigrahi, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems."," International Journal of Engineering & Technology 7, no. 3.24 , 2018 .

[7] P. Flach, "Machine learning: the art and science of algorithms that make sense of data.," Cambridge University Press, 2012 .

[8] M. Maloof, " Machine learning and data mining for computer security: methods and applications.," Springer Science & Business Media, 2006 .

[9] R. Y. A. S. C. J. K.-C. M. F. C. a. J. P. C. Choi, "Introduction to machine learning, neural networks, and deep learning," Translational Vision Science & Technology 9, no. 2, 2020 .

[10] E. M. Ö. S. A. &. I. T. Karabulut, " A comparative study on the effect of feature selection on classification accuracy.," Procedia Technology, 1, 323-327., 2012 .

[11] N. a. J. S. Moustafa, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)."," In 2015 military communications and information systems conference (MilCIS, 2015 .

[12] T. I. K. a. I. T. Shatovska, "New module of text classification for IDA system.," In 2009 10th International Conference-The Experience of Designing and Application of CAD Systems in Microelectronics, pp. 481-482. IEEE, , 2009 .

[13] R. Y. A. S. C. J. K.-C. M. F. C. a. J. P. C. Choi, "Introduction to machine learning, neural networks, and deep learning," Translational Vision Science & Technology 9, no. 2, 2020 .

DOI:
https://doi.org/10.54216/JCIM.07.02.04