Major Report

by H.c. Taneja

Submission date: 23-May-2020 10:01PM (UTC+0530)
Submission ID: 1330530383
File name: Major_Report_-_Main_Content.pdf (1.7M)
Word count: 8871
Character count: 45200

ABSTRACT

The amount of data that is exchanged, transmitted, produced and stored in today's world has increased exponentially thus forcing intrusion detection systems to become an integral part of modern-day network security systems. Considerable expense, time and effort are spent in ensuring timely detection and denial of malicious network activity in order to preserve the key objectives of system security; confidentiality, integrity, and availability. In this project, we propose a computationally light framework for NIDS by operating on a reduced feature space.

Research into network intrusion detection systems dates back to the early 1990s where researchers initially developed rule-based algorithms such as SNORT and TCPDUMP. As the subject gained traction and importance, researchers began to shift efforts towards creating anomaly detection systems using benchmarked datasets to test their algorithms. A brief of the approaches used by researchers is summarized below:

- **Clustering with ANNs:** The data points are divided into 'n' clusters, with 'n' representing the number of malicious classes. A binary ANN is then trained for each cluster to classify each data point into one of the clusters. This is followed by an aggregating ANN solely to reduce the number of false positives.
- **Feature Space Compression followed by ANNs:** ANNs are trained to work on a feature space that has been compressed using common feature reduction techniques such as autoencoding or PCA. Autoencoding has proven to be more beneficial in this regard as each one can be trained individually.
- **RNNs/LSTMs**: Some papers have used unrolled versions of sequential deep learning units such as RNNs, LSTMs or GRUs. By doing so, the authors have introduced context amongst the features. Thus, if a particular value of a feature is important in deciding the class label while another feature takes a different value, the temporal activations in these units are able to represent them best. The drawback however is that as standard feed-forward networks become extremely heavy for a large set of features; it becomes difficult for networks running on low compute servers to catch every attack. This problem is further

exaggerated for RNN and LSTM as forward propagation becomes more computationally expensive when the concept of context is brought in. Essentially, all information passes through each cell, making the network extremely heavy hence requiring more processing time.

Our research focused on building an IDS using the benchmarked NSL-KDD dataset. The aim was to develop an efficiently compressed feature space with the ultimate goal was to use sequential models to our benefit and build a computationally light classification model capable of operating on the aforementioned compressed feature set. The methodology we followed consisted of four steps:

1. **Feature Engineering:** The dataset consisted of 3 categorical and 39 numeric features, giving rise to a total of 41 features. We opted for One-Hot Encoding of the categorical features, which caused our feature space to explode to a sum of 84. On the numerical ones, we applied min-max scaling so that the features can be weighed on the same range. After this, PCA to bring down the number of features to 67 while retaining 99% variance of the original dataset.
2. **Binary Classification:** We applied binary classification on the dataset to divide it into two classes, malicious and benign. We used factors like precision, recall, area under the ROC curve to compare the various ensemble methods that were employed. The aim of this step is to increase recall so as to reduce false negatives.
3. **Oversampling:** U2R and R2L had a lower number of datapoints in comparison to the rest, so oversampling using SVMSMOTE was done which essentially synthesized new datapoints.
4. **Multi Class Classification:** We employed an ANN on the output class of binary classification containing both malicious and benign datapoints in an effort to reduce the number of false positives.

We conclude that compared to already given methods, we have achieved better evaluation results with our two-stage classification algorithm. A reduced feature space ensures forward propagation through the model to work fast so that detection overhead is minimized.

CHAPTER 1 INTRODUCTION

1.1 OBJECTIVE

The primary objective of this project is to develop an efficiently compressed feature space with the ultimate goal being to develop a computationally light classification model capable of operating on the compressed feature set using sequential models to our benefit.

1.2 CONCEPTS

1.2.1 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are a form of connected units that are capable of learning complex mathematical functions. They receive multidimensional input and are capable of producing multidimensional output. ANNs are inspired by how the human mind works. Signals in the form of high-dimensional matrices are propagated from the input layer to the output layer, adding some non-linearity at each stage. This enables them to learn mathematical functions in the form of small linear computations.

A standard feed forward ANN consists of multiple neuron-like structures, known as perceptrons. A perceptron in computer science is a unit that takes input from multiple sources, performs a certain computation on them and produces an output.

Typically, perceptrons are arranged in layers to facilitate stage by stage learning. The ANN is then fed with training samples, which are essentially pairs in the form of (input, output). This allows the ANN to recognize patterns in an iterative fashion. For e.g. If we give the task of identifying facial structures to an ANN, the first few layers

might learn very low-level information like edges and curves. These learnings will then be consolidated to identify small facial structures like boundaries and facial hair. In this fashion, layers will keep learning complex structures as we move ahead and the final few layers learn how to identify a face.

ANNs come in multiple forms, each with its own unique use case. A standard feed forward ANN is good for a standard feature set, as in the case of the NSL-KDD dataset. For images on the other hand, feed forward networks pose a fundamental problem: high dimensionality. Due to this, convolutional models are used to train machine learning models on images. Sequential data like in the case of natural language processing requires linear context inbuilt, which is why a separate class of models known as recurrent models are used to train on such data. There are also multiple forms of training a neural network, depending on the task. For e.g. Sometimes it may not be so important to classify an image as compared to knowing whether that image is legitimate or not. A generative adversarial network does exactly this task. For the purpose of facial recognition, a separate class of networks known as Siamese networks is used.

1.2.2 DEEP LEARNING

Deep learning is a part of a larger class of learning algorithms known as machine learning. These algorithms basically enable a computer to learn complex mappings from input to output. Deep learning enables computers to learn very complex mappings. There are 2 basic requirements to perform deep learning:

- A lot of computational power
- A lot of data

The reason is that since deep learning is learning such complex mathematical functions, the computations can get very hard to manage for a computer with low memory. Since data is constantly being swapped in and out of RAM, it tends to heat up the machine too. In more than one way, any deep learning task pushes a machine to its limits. A special class of instructions, known as SIMD instructions, supported by GPUs are considered much more suitable for deep learning tasks. These instructions allow true parallelism in the form of GPU cores. Since forward propagation in a single epoch is independent of the training sample, it is possible to run multiple samples in parallel. This speeds up the process by a considerable factor and is beneficial for the computer as well since it leaves the CPU to perform tasks more fundamental to the OS.

The second requirement stems from the highly complex nature of the function that is being learned. Since the complexity of a mathematical function is directly proportional to its non-linearity, a highly complex function is highly non-linear too. This means that even a task as strict as standard interpolation would require training samples proportionate to the degree of non-linearity. Since in case of deep learning algorithms, generality is also a factor, the function that's actually being learned is an average of multiple functions, depending on the conditions the training samples are taken from, which adds to the complexity of the problem.

Deep learning, due to its high requirements has only started to grow recently due to the possibility of increasing memory and speed. Big tech firms have started to incorporate artificial intelligence into their systems by using deep learning. The field has immense scope in fields like signal and image processing, natural language processing, data and business analysis, risk analysis, etc. It enables humans to understand how the human brain works and to model for tasks based on that information. Some authors also postulate that deep learning will grow exponentially in the coming times, in the same way as computers grew during the 80s. This poses great avenues for companies as well as individuals.

1.2.3 MACHINE LEARNING

Machine learning is the field of computer science that gives computers the ability to learn real life functions like speaking and identifying objects to name a few. It uses mathematics and probability theory along with computer science to accomplish this. It mostly requires data to be represented in a specific mathematical form, which is a criterion that has already been fulfilled to large extent for most forms of data. A large array of algorithms has been devised to work on different kinds of data. The kind of algorithm used also depends on what the format of the output is.

Machine learning algorithms mostly consist of some feedback mechanism using which an algorithm trains itself. Training examples are iteratively fed into the model and the results are compared to the actual results. Based on this difference, a loss is computed and is fed back to the model, which allows the model to retune its parameters.

Machine learning algorithms can be classified in 3 forms based on the kind of data they work on and the nature of the algorithm:

- **Supervised Learning:** Supervised learning is subclass of machine learning algorithms that works on data that consists of (input, output) pairs. This means that for every input, we know what the correct output will be. This enables the algorithm to use the feedback mechanism as mentioned earlier.
- **Unsupervised Learning:** Unsupervised learning algorithms work on data in which only an input is provided. Output of these kinds of algorithms depends only on the input distribution and feedback works in a different manner. For e.g. In clustering, feedback to the algorithm is generally given in form of distance to the centre of the various clusters. Clustering is broadly the only machine learning algorithm that falls under this category.

- **Reinforcement Learning:** Reinforcement learning is a subclass of machine learning that train an algorithm in a dynamic environment where there is an agent which makes decisions and receives feedback based on those decisions. There is no training set as such because the agent is repeatedly making decisions and receiving feedback that may be a reward or a penalty. Through multiple iterations in the same environment, the agent learns how to navigate it and make decisions that would fetch it long term rewards. Generally, reinforcement learning is used in cases like teaching a computer how to play a game or how to navigate a maze. These scenarios pose a challenge when it comes to developing training sets because there are a lot of combinations and local decisions to make.

1.2.4 CONVOLUTIONAL NEURAL NETWORKS

Convolutional layers in deep learning are normally used to train models on images. Their work initially created to exploit the contextual nature of pixels in an image, i.e., a pixel is normally not independent of its neighbours. This allows deep learning models to use filters to learn features through convolutional layers. The reason standard feed forward networks don't generally work well with images is:

- Generally, images are large and consist of potentially millions of pixels. Hence, a machine learning application would normally require to work on every pixel individually, which would shoot up the dimensional of the ANN by a large factor.
- Due to the reasons mentioned in above, working on individual pixels independently may not be such a great idea as in that case not only are we ignoring a lot of information that the image is giving us, nodes in the ANN will actually start memorizing values of each specific pixel. This may create a problem for situations like when the object to be recognized has been translated a bit in some other direction.

Convolutional layers operate on images of fixed size, which is why the input stream generally has to perform some resizing/cropping operation on the image so it can be fed to the CNN. Each convolutional layer consists of a filter, sometimes also referred to as a kernel. Each kernel tries to learn a single feature, the complexity of which is dependent on which layer the filter is applied on. When a filter is applied on an image, it outputs a single pixel depending on the weights and biases in each cell of the kernel. This reduces the size of the incoming image. Once the image reaches a size that is small enough, it is flattened and fed through a series of fully connected layers to compute the final output.

1.2.5 RECURRENT NEURAL NETWORKS

RNNs are a class of neural networks that typically operate on sequential or time- series data. The structure of this layer type allows one to exploit the linear context while analysing data that is sequential in nature. For e.g. If we speak a certain sentence, the nth word in the sentence is not independent of its surrounding words. Similarly, the weather on some nth day is not independent of the weather that has occurred on previous days. For these purposes this special class of neural networks are required.

RNNs are broadly of two kinds, unidirectional and bidirectional, both have their own use cases. Unidirectional RNNs take into account context from only 1 end of the sequential input, whereas bidirectional RNNs take into account inputs from both ends of the sequential input. As an e.g. the language analysis case mentioned earlier may require the bidirectional RNNs, but weather analysis strictly requires a unidirectional RNN, as weather on the nth day cannot be influenced by the weather on the (n+1)th day.

Each RNN cell consists of broadly 2 kinds of operations: an activation that is passed on to the further layers and another activation that is passed on to the RNN cells ahead of it to incorporate context. In a bidirectional RNN, these activations are 3 in count. One for the layer ahead of it, one for the cells ahead of it, and one more for the cells before. This naturally increases the performance overhead of a bidirectional RNN a lot, which is why it should be used when absolutely necessary.

A great disadvantage of an RNN is that it doesn't take into account long term dependencies, which basically means that an event that occurred a long time ago, will have minimal effect on the output of the current time step. This problem is addressed in much more detail in GRUs and LSTMs.

1.2.6 LABEL ENCODING

One major area of research with respect to data representation in numerical forms has been around label encoding and one hot encoding. Both have their own uses and areas of benefit. Label encoding assigns categorical variables sequential numbers. For e.g. suppose some machine learning application consists of an attribute that can take values like "low", "medium", "high". Since there is a natural ordering in these values, it makes sense to assign 0 to low, 1 to medium and 2 to high. This process is known as label encoding, and one of its many benefits include the preservation of dimensionality. The downside of using label encoding to represent an attribute that does not have ordering is that the machine learning application will then get confused between low and high numbers representing that attribute.

Based on the above analysis, it is clear that label encoding can be used with ordinal variables only. Using it with nominal variables may give rise to catastrophic failures in code. It is not necessary that label encoding be done with numbers at regular

intervals. Most encoders allow the programmer to specify their own ordering, which maintains monotonicity in the attribute.

1.2.7 ONE-HOT ENCODING

One-hot encoding is another way of representing non-numeric categorical variables in numeric forms for the computer to understand. One-hot encoding works in situations where the attributes in question are nominal in nature, i.e., they don't have an inherent ordering among them. A good example of this would be an attribute which specifies the type of a car, i.e., "sedan", "hatchback", etc. Now in most cases it won't make sense for there to be an ordering among these values. So, in such situations, one-hot encoding is used. One-hot encoding basically replaces categorical attributes with 'm' binary attributes, each specifying if that attribute takes a particular value or not, where 'm' is the total number of unique values the attribute can take.

A major disadvantage of using one-hot encoding is that it dramatically explodes the dimensionality of the dataset. For e.g. In our case, without any dimensionality reduction, a feature set of 41 features went up to 122, an increase by a factor of 3, after doing one-hot encoding. To resolve this problem, it is often a good idea to use dimensionality reduction techniques, since such a dramatic increase is often complemented with a very sparse feature set.

One-hot encoding is useful in almost every case, since it does not assume anything about the attribute. But due to the dimensionality increase problem, it is often a good idea to go through the attributes and try to understand for which attribute would label encoding work, since that maintains dimensionality.

1.3 TECH-STACK

1.3.1 GOOGLE COLABORATORY

Google Colaboratory (or Google Colab) is a free python based Jupyter notebook environment that runs on the Google Cloud Platform (GCP). It allocates separate containers, that host the runtime environment for a single notebook inside which all the code is run. This means that any packages that are not provided by Google have to be installed every time a new instance of the runtime is allocated. Since code is run within containers, it is possible to install other languages like Swift and R inside the container, which also have to be reinstalled each time a new instance is created.

Google Colab was initially started as an internal project for R&D at Google, but was made available for everyone to use in 2017. It hosts a great environment to run standalone code along with GPU support. Google also increased the RAM quota for Colab in 2018. As on date, Google Colab has GPU support with a single core Tesla K80 processor and up to ~26GB of RAM.

1.3.2 JUPYTER NOTEBOOKS

Jupyter Notebooks are a web based interactive environment for creating IPython files (.ipynb). It allocates an interpreter kernel to the running notebook and hence, has support for all interpreted languages like R, Python. While Python supports comes inbuilt with the installation, support for any other language can be installed easily. A running Jupyter notebook basically has the interpreter kernel loaded into memory of the machine for faster computation. This means that any changes to the existing kernel requires the kernel to be restarted.

Jupyter Notebooks can be converted into a number of standard output formats for publishing code. These formats include but are not limited to:

- LaTeX
- PDF
- ReStructuredText
- Markdown
- Python

These conversions can be done either from the UI or using the jupyter nbconvert command from the CLI.

Throughout the project Jupyter Notebooks have been extensively used along with GCP kernels for better performance. One major benefit of using Jupyter Notebooks is that the programmer need not worry about file level changes, and is free to think of cell level changes, which makes work much easier. The kernel has garbage collection inbuilt and loading variables into memory and releasing unused memory is handled well by the kernel.

1.3.3 MATPLOTLIB

Matplotlib is a famous plotting library for Python. Other libraries include Plotly, Seaborn, etc. Matplotlib is a very extensive drawing library and has support for all kinds of charts that one might need to plot during data or result analysis of an algorithm. One major benefit of using Matplotlib as the plotting library of choice is that it is integrated with Pandas. Since all of the project data had initially been loaded into Pandas DataFrames, plotting with Matplotlib was very convenient.

1.3.4 NUMPY

NumPy is a Python library for loading and manipulation high dimensional Python arrays. It has a large number of functions for manipulation of these arrays, which include most mathematical operations, like vector arithmetic as well as statistical functions, like mean and covariance. It is mostly the standard array form to use while performing machine learning tasks and is completely integrated with Pandas. This means NumPy arrays can be converted into Pandas Data Frames and vice versa with ease.

All NumPy functions are centred around the most essential NumPy datatype, the ndarray, short for “n-dimensional array”.

All NumPy operations are natively run on the CPU. An alternative of the NumPy library is the CuPy library and supports NumPy operations on GPUs. It does this using the Nvidia CuDNN C++ based library and is much faster than standard NumPy.

1.3.5 PANDAS

Pandas is a Python library written for data manipulation and analysis. It supports loading large data files into Pandas DataFrames, the central Pandas datatype. Data can be loaded into memory from an extensive set of formats. These include but are not limited to csv, xlsx and arff formats. Jupyter Notebooks complement Pandas by displaying Pandas Data Frames in a much more human readable format as compared to simple console output.

Pandas can be used for a variety of things, including data cleaning, pre-processing and partitioning. Pre-processed data can then be saved in separate files that may be used by the deep learning algorithm later. This modularizes code and enables the programmer to think of pre-processing as a black box.

1.3.6 SCIKIT-LEARN

Scikit Learn is a Python based machine learning library and has inbuilt functions for simple machine learning tasks like classification, regression, etc. Each sub package of the main sklearn package consists of a variety of classes which can be used for unique computations on native NumPy arrays. Scikit-Learn is mostly just used for simple machine learning tasks and not deep learning.

Scikit Learn uses NumPy and SciPy arrays as its native datatype. Classes are inter-functional which enables easy scoring and comparison of functions. It also provides a wide range of pre-processing and analysis related tools that can be used to transform the dataset. It also consists of various result analysis related functions along with tools for statistical analysis of attribute dependencies. This allows the user to transform and remove attributes.

1.3.7 TENSORFLOW

Tensorflow is a free, open source differentiable programming library used for deep learning tasks released by Google in 2015. Tensorflow was developed by the Google Brain team, mostly for internal use, but was released in public domain later on. It has stateful dataflow graphs which allow for easy backpropagation of gradients

and has been in use in the industry for some time now. It is used by many AI companies as their framework of choice. Recently, Tensorflow 2.0 has been released which makes Tensorflow code much easier to write for the programmer and combines it with important OOPS concepts.

Tensorflow has been one of the industry standard frameworks for a very long time now and since is extensively used within the open source community as well. It is available to use with all Python versions and C++ as well. The framework is robust in nature but has a very steep learning curve. It takes a while to get accustomed to the framework and is hence used for limited tasks in the project.

Tensorflow also has support for Nvidia CuDNN libraries so any model developed using Tensorflow can be trained and run on a GPU as well.

1.4 DATASET

1.4.1 FEATURES

The NSL-KDD data set is analysed and categorized into four different clusters depicting the four common different types of attacks. It is a dataset is a collection of downloadable files at the disposal for researches. They are listed as follows:

Table 1.1 List of NSL-KDD dataset files and their description

S. No.	Name of the file	Description
1	KDDTrain+.ARFF	The full NSL-KDD train set with binary labels in ARFF format.
2	KDDTrain+.TXT	The full NSL-KDD train set including attack-type labels and difficulty level in CSV format.

3	KDDTrain+_20Percent.ARFF	A 20% subset of the KDDTrain+.arff file.
4	KDDTrain+_20Percent.TXT	A 20% subset of the KDDTrain+.txt file.
5	KDDTest+.ARFF	The full NSL-KDD test set with binary labels in ARFF format.
6	KDDTest+.TXT	The full NSL-KDD test set including attack-type labels and difficulty level in CSV format.
7	KDDTest-21.ARFF	A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21.
8	KDDTest-21.TXT	A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21.

The following points are note-worthy about the dataset:

1. So that the classifiers produced an unbiased result, redundant records have been removed.
2. The train and test data sets both contain a sufficient number of records so as to enable the execution of experiments.
3. The number of selected records from each difficult level group is inversely proportional to the percentage of records in the original KDD dataset.

The following table represents the 41 attributes of each record as well as a label assigned to each either as an attack type or as normal. The 4 attack classes are DoS, Probe, R2L and U2R details on which is contained in the 42nd attribute, classifying each as either normal or benign. The details of the attributes their description and sample data are listed in the following tables.

Table 1.2 Basic features of each network connection vector

Attribute No.	Attribute Name	Description	Sample Data
1	Duration	Length of time duration of the connection.	0
2	Protocol_type	Protocol used in the connection.	Tcp
3	Service	Destination network service used.	ftp_data
4	Flag	Status of the connection – Normal or Error.	SF
5	Src_bytes	Number of data bytes transferred from source to destination in single connection.	491
6	Dst_bytes	Number of data bytes transferred from destination to source in single connection.	0
7	Land	If source and destination IP addresses and port numbers are equal then, this variable takes value 1 and 0.	0
8	Wrong_fragment	Total number of wrong fragments in this condition.	0
9	Urgent	Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated.	0

Table 1.3 Content related features of each network connection vector

Attribute No.	Attribute Name	Description	Sample Data
10	Hot	Number of 'hot' indicators in the content such as: entering a system	0
11	Num_failed_logins	Count of failed login attempts	0
12	Logged_in	Login Status:1 if successfully logged in; 0 otherwise	0
13	Num_compromised	Number of "compromised" conditions	0
14	Root_shell	1 if root shell is obtained; 0 otherwise	0
15	Su_attempted	1 if "su root" command attempted or used; 0 otherwise	0
16	Num_root	Number of "root" accesses or number of operations	0

		performed as a root in the connection	
17	Num_file_creations	Number of file creation operations in the connection	0
18	Num_shells	Number of shell prompts	0
19	Num_access_files	Number of operations on access control files	0
20	Num_outbound_cmds	Number of outbound commands in an ftp session	0
21	Is_hot_login	1 if the login belongs to the "hot" list i.e., root or admin; else 0	0
22	Is_guest_login	1 if the login is a "guest" login; 0 otherwise	0

Table 1.4 Time related traffic features of each network connected vector

Attribute No.	Attribute Name	Description	Sample Data
23	Count	Number of connections to the same destination host as the current connection in the past two seconds	2
24	Srv_count	Number of connections to the same service (port number) as the current connection in the past two seconds	2
25	Serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23)	0
26	Srv_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24)	0
27	Rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)	0
28	Srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)	0
29	Same_srv_rate	The percentage of connections that were to the same service, among	1

		the connections aggregated in the count (23)	
30	Diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in count (23)	0
31	Srv_diff_host_rate	The percentage of connections that were to different destination machines among the connections aggregated in srv_count (24)	0

Table 1.5 Host based traffic features in a network connection vector

Attribute No.	Attribute Name	Description	Sample Data
32	Dst_host_count	Number of connections having the same destination host IP address	150
33	Dst_host_srv_count	Number of connections having the same port number	25
34	Dst_host_same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32)	0.17
35	Dst_host_diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)	0.03
36	Dst_host_same_src_port_rate	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)	0.17

37	Dst_host_srv_diff_host_rate	The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33)	0
38	Dst_host_serror_rate	The percentage of connections that have activated the flag (4) so, s1, s2 or s3, among the connections aggregated in dst_host_count (32)	0
39	Dst_host_srv_serror_rate	The percent of connections that have activated the flag (4) so, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (33)	0
40	Dst_host_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32)	0.05
41	Dst_host_srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33)	0

The attack classes in the NSL-KDD dataset are grouped into four categories:

1. **DoS:** Denial of Service is an attack category, which depletes the victim's resources thereby making it unable to handle legitimate requests – e.g. syn flooding.

   Relevant features: "source bytes" and "percentage of packets with errors".

2. **Probing:** Surveillance and other probing attack's objective is to gain information about the remote victim. e.g. port scanning.

   Relevant features: "duration of connection" and "source bytes".

3. **U2R:** unauthorized access to local super user (root) privileges is an attack type, by which an attacker uses a normal account to login into a victim system and tries to gain root/administrator privileges by exploiting some vulnerability in the victim e.g. buffer overflow attacks.

   Relevant features: "number of file creations" and "number of shell prompts invoked".

4. **R2L:** unauthorized access from a remote machine, the attacker intrudes into a remote machine and gains local access of the victim machine. E.g. password guessing.

   Relevant features: Network level features – "duration of connection" and "service requested" and host level features - "number of failed login attempts".

Table 1.6 Attribute Value Type

Type	Features
Nominal	Protocol_type(2), Service(3), Flag(4)
Binary	Land(7), logged_in(12), root_shell(14), su_attempted(15), is_host_login(21), is_guest_login(22)
Numeric	Duration(1), src_bytes(5), dst_bytes(6), wrong_fragment(8), urgent(9), hot(10), num_failed_logins(11), num_compromised(13), num_root(16), num_file_creations(17), num_shells(18), num_access_files(19), num_outbound_cmds(20), count(23) srv_count(24), serror_rate(25), srv_serror_rate(26), rerror_rate(27), srv_rerror_rate(28), same_srv_rate(29) diff_srv_rate(30), srv_diff_host_rate(31), dst_host_count(32), dst_host_srv_count(33), dst_host_same_srv_rate(34), dst_host_diff_srv_rate(35), dst_host_same_src_port_rate(36), dst_host_srv_diff_host_rate(37), dst_host_serror_rate(38), dst_host_srv_serror_rate(39), dst_host_rerror_rate(40), dst_host_srv_rerror_rate(41)

Table 1.7 Mapping of attack class with attack type

Attack Class	Attack Type
DoS	Back, Land, Neptune, Pod, Smurf,Teardrop,Apache2, Udpstorm, Processtable, Worm (10)
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint (6)
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named (16)
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps (7)

Table 1.8 Details of normal and attack data in different types of NSL-KDD dataset

Dataset Type	Total Number of					
	Record	Normal Class	DoS Class	Probe Class	U2R Class	R2L Class
KDDTrain+ 20%	25192	13449	9234	2289	11	209
		53.39%	36.65%	9.09%	0.04%	0.83%
KDDTrain+	125973	67343	45927	11656	52	995
		53.46%	36.46%	9.25%	0.04%	0.79%
KDDTest+	22544	9711	7458	2421	200	2754
		43.08%	33.08%	10.74%	0.89%	12.22%

1.4.2 STATISTICS

The following graph depicts the log(Class Distribution) of the Train Set in terms of the datapoints for each attack class:

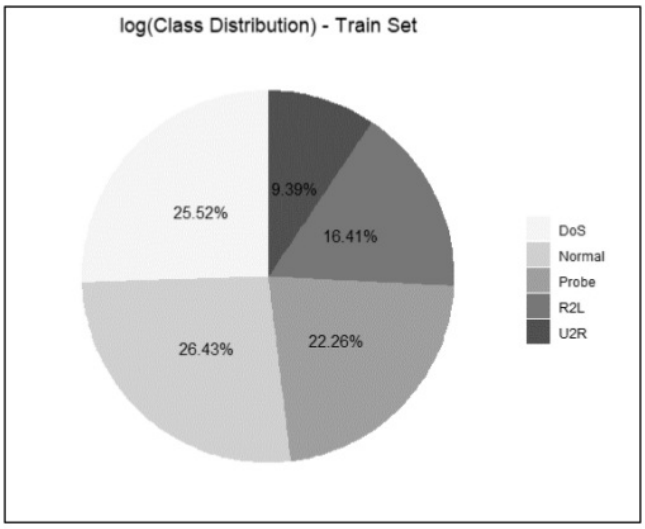

Fig 1.1 log(class distribution) of training set

The following graph depicts the class distribution of the Test Set in terms of the datapoints for each attack class:

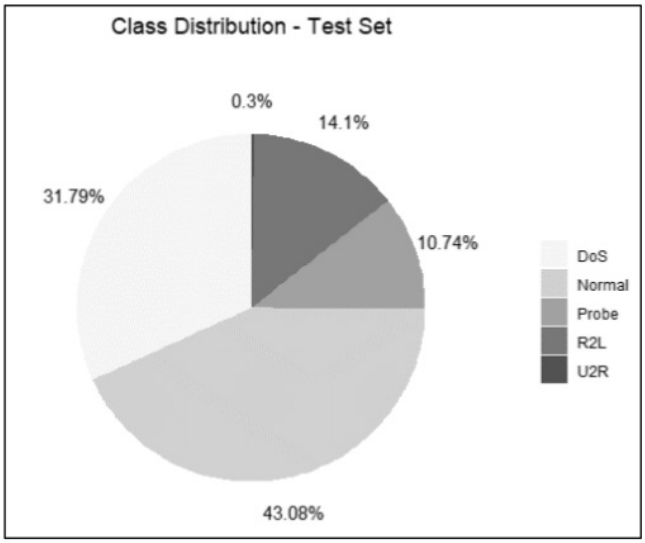

Fig 1.2 Class distribution of test set

The following graph shows the log(counts) of each category of the "flag" attribute in both datasets:

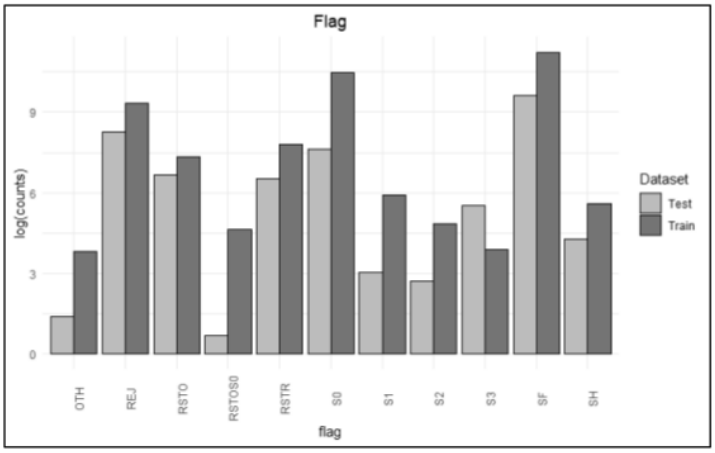

Fig 1.3 log(counts) of each category of flag attribute

The following graph shows the counts of each category of the "protocol_type" attribute in both datasets:

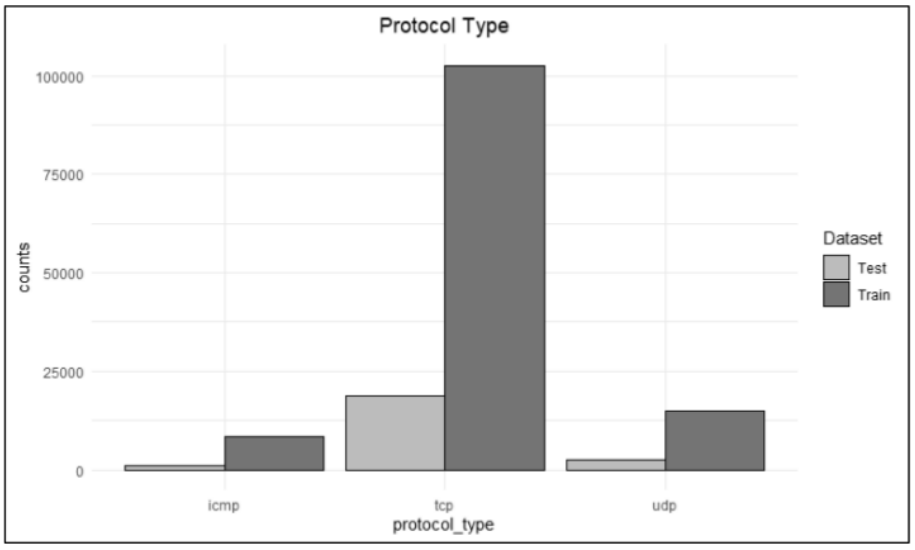

Fig 1.4 Count of each category of protocol_type attribute

The following graph shows the log(counts) of each category of the “service” attribute in both datasets:

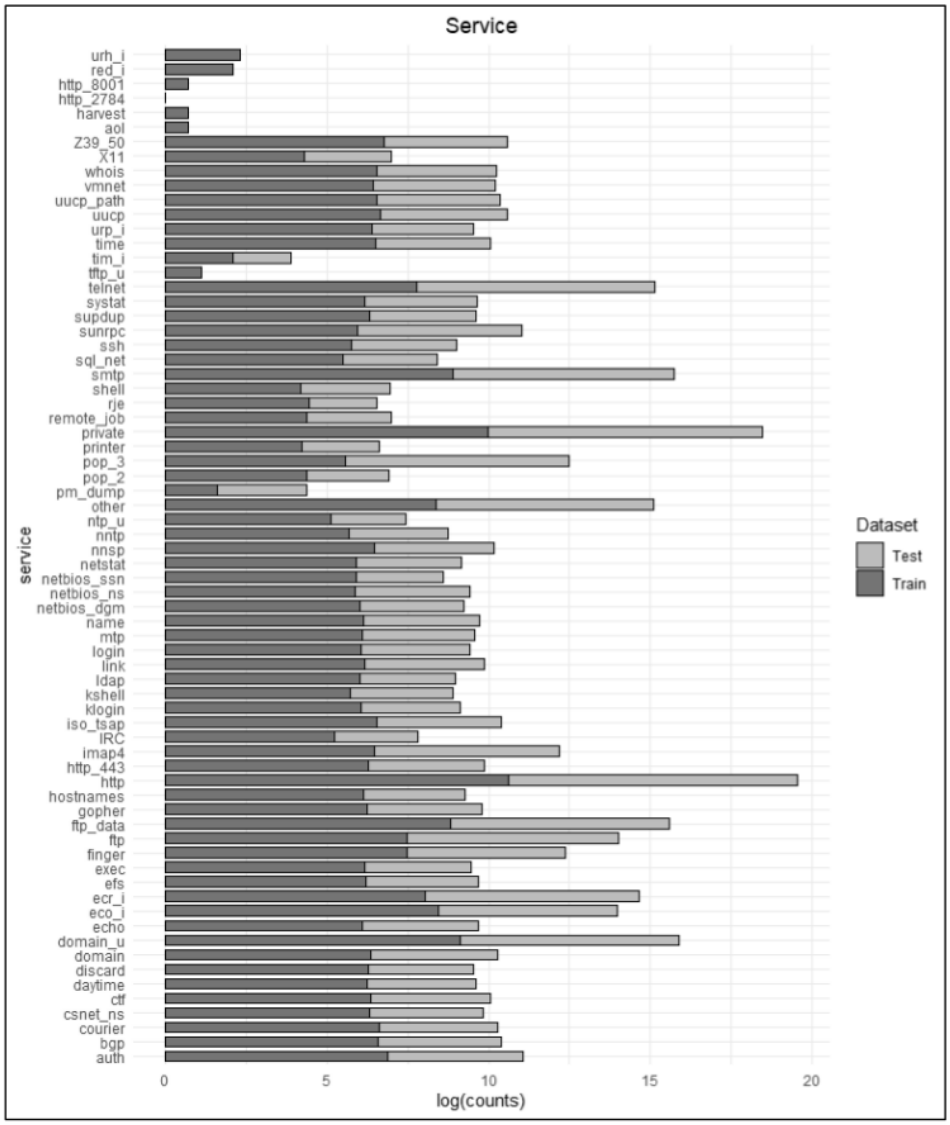

Fig 1.5 log(count) of each category of service attribute

CHAPTER 2 LITERATURE REVIEW

There exists extensive literature delving into the problem of developing extremely accurate and reliable network intrusion detection systems which can be installed and deployed with minimal overhead, and require minimal resources to run. The last requirement is crucial as most NIDS run round the clock and hence need to be affordable and efficient. While early detection systems used rule-based architecture, contemporary research has focused on the use of deep learning techniques, following the proposal of Hinton [9] in 2006. Owing to the popularity of deep learning techniques in contemporary NIDS research, we referred several papers which suggest the use of various techniques to build detection systems, yielding successful results.

Wang[3] and Shraddha[4] suggested the use of ensemble models consisting of fuzzy clustering followed by a classification ANN trained on each class. This provided a drastic reduction in false positives compared to previous techniques. The model suggested by Shraddha[4] performs exceptionally well in this regard owing to the use of an aggregating ANN after the individual classification ANNs. The benefit of using such ensemble models is that they use the output of multiple classifiers and combine them using some rule (such as majority rule or logical AND) to obtain the final result. As the final classification depends on multiple classifiers, it is less prone to errors.

Potluri and Diedrich[6] implemented autoencoders to encode the feature space into a lower dimensional and leverage parallel processing to execute instructions. However, the central theme of their paper revolved around the complexity and execution time of the algorithm, owing to which they opted for a drastic reduction in the feature space. Experiments show that such drastic reduction can lead to loss of crucial information and hence is not ideal. However, the benefit of autoencoders is that they are a simple way of compressing the feature space. Depending on the level of compression of reduction, it is possible to preserve most of the information in the data-

set. Despite being a black box, autoencoders are a commonly used tool for feature extraction and compression

Yin[7] and Kim[10] have suggested the use of sequential neural networks such as LSTMs and RNNs which are implemented 'bidirectionally', along with the introduction of context in the first layer itself. This allows the depth of the network to be restricted to 2 or 3 levels. While forward propagation in the suggested networks can be time consuming due to the bidirectional connections, the limited number of nodes and layers causes a drastic reduction in space overhead. LSTMs and RNNs are advanced Deep Learning Frameworks that use different types of nodes. A major benefit of these Networks are their ability to take into consideration the context represented by the data. It does this through its ability to 'remember' previous data that has been fed into the network and use the previous data and automatically learn dependencies amongst features. RNNs and LSTMs find extensive use in the problems of text prediction, NLP, Stock Market prediction etc due to this unique ability.

Chiba[11] built upon the idea of using clustering techniques followed by ANNs by introducing the use of genetic algorithms such as crossover and mutation in order to find the optimal feature space or parameter set. The idea of genetic algorithms is inspired by the theory of evolution and 'survival of the fittest', using an appropriate 'fitness' measure to evaluate and compare feature spaces and choose the most optimal one when run for several epochs. However, genetic algorithms are very complex and time consuming as it trains the underlying classification model over the entire dataset multiple times to calculate fitness measures for different feature spaces.

Several scholars used similar methods to build models using deep learning techniques and have delivered ground-breaking results each time. Different papers have focused on improving accuracy, reducing false positives or reducing the time or space complexity of their algorithm. It is our opinion that too a large extent, this space is almost saturated, with scholars delivering up to 99% accuracy through their models.

Their research has laid the foundation of our work. Spurred by our research, we decided to create a robust classification model that can work with a reduced feature space without compromising on performance.

CHAPTER 3 METHODOLOGY

To reach the results that finally received publication, we experimented with a wide variety of methods. Initially, we took the task of intrusion detection to be one of anomaly detection, hence, some efforts were made in the direction of clustering and z-score analysis. We also considered using signature-based detection methods that would compute some norm based on the signature generated for the given feature tuple and the signature for that class. Since similarity was one area the problem could be formulated into, we also tried to use a Siamese network.

In the following sections, we explain each method in detail and the reasons why we decided to abandon each approach. We also discuss why we think the final method worked best out of all.

3.1 PLOTTING UTILITIES

To analyse the data in the best way possible, we used visual aids throughout the project. From data analysis to result analysis, we utilised the python plotting runtime extensively. Following were the charts we plotted:

3.1.1 LOSS PLOTTING

Since the project involved a lot of neural network training, it made sense to make a wrapper function to plot losses. This function was made in such a way that it worked for scikit-learn models as well as tensorflow models. Loss plotting was done by representing 2 lines, one for the training set losses and one of the testing set losses. A similar plot was the accuracy plot, which plotted the training set and testing set accuracies.

3.1.2 CLUSTER PLOTTING

As mentioned above, we tried to formulate an anomaly/outlier detection problem for the task at hand. To accomplish the same, we used the widely recognized clustering technique. It was natural that clusters could be computed for n-dimensional vectors, but only a maximum of 3 dimensions could be plotted. We used the following procedure for plotted clusters:

- Cluster points based on the n dimensions obtained after dimensionality reduction
- Apply PCA on all points, obtaining 3 dimensions for each point
- Plot on a 3D axis for each cluster

The above procedure proved to work for every algorithm that required clustering. We applied clustering on 2 kinds of datasets. In one method, we used label encoding to represent categorical variables and in another we used one hot encoding to do the same. As mentioned before, it made sense to use one hot encoding for the given dataset and clustering proved the same. The clusters obtained for the one hot encoded vectors have much more inter-cluster separation and much less intra-cluster separation.

Another very important cluster related plot that we made was the K-Elbow Visualizer plot. The K-Elbow Visualizer plot recognizes the optimal number of clusters for a dataset.

3.1.3 CONFUSION MATRIX

Confusion matrices are famous for their concise representation of the results of a particular model. So much so, that close to about 10 metrics are calculated using the confusion matrix alone. As our task was to eventually classify points into 1 of 5 classes: Normal, DoS, Probe, R2L and U2R, we used confusion matrices throughout are project.

3.1.4 ROC CURVE

A Receiver Operator Characteristic (ROC) curve is a graphical plot to analyse the diagnostic ability of binary classifiers. When we switched to the dual stage methodology, the first stage of the classification process was binary in nature, we used curves like the ROC curve to analyse components like optimal threshold and final precision and recalls for any given method.

The ROC Curve of a model plots the true positive rate against the false positive rate for varying thresholds. Typically, a good ROC curve would close as much as possible towards the point (0,1) for false positive rate on the x-axis and the true positive rate on the y-axis. A good model would therefore have a very steep ROC curve for points beneath the line $y = -x$ and a very flat curve for points above this line.

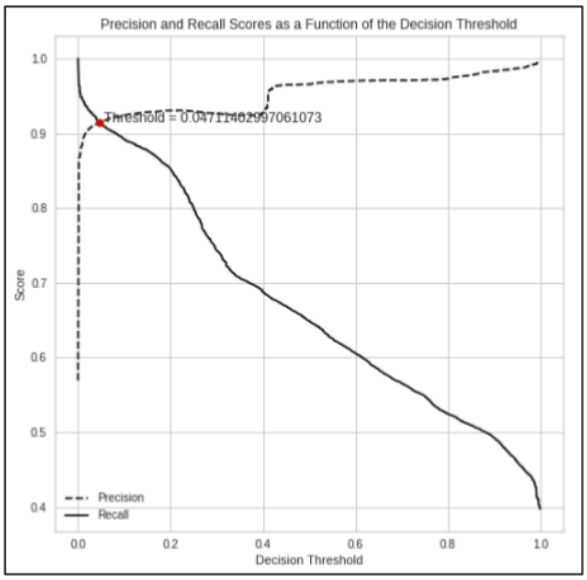

Fig 3.1 Precision and Recall Vs Threshold Curve

Precision and recall are two extremely important model evaluation metrics. While precision refers to the percentage of your results which are

relevant, recall refers to the percentage of total relevant results correctly classified by your algorithm.

Typically, precision and recall are considered to have a somewhat inverse relationship. A very high precision would result in extremely low recall and vice versa. The precision recall curve allows one to analyse this trade-off against varying threshold values. This curve allows us to choose a suitable threshold for the model to achieve a balance between precision and recall.

3.1.5 BAR CHARTS

Bar Charts are by far one of the simplest plots used in the project. From analysing class and category distributions to plotting scores for univariate selection, bar charts have been used extensively throughout the project.

3.1.6 HEATMAP

Heatmaps are plots that closely resemble a confusion matrix but in the context of our project have a completely different utility. We used heatmaps to analyse the correlation between various attributes as part of our data analysis portion. We plotted the heatmaps using the seaborn package in python.

3.2 DATA ANALYSIS

3.2.1 GINI INDEX IMPORTANCE FOR CATEGORICAL VARIABLES

Following the One-Hot Encoding of categorical variables, the Gini index criterion was used to construct a decision tree classifier with the aim of evaluating the importance of the individual features. The importance of each feature is quantified by measuring the total reduction of the Gini importance criterion due to the respective feature.

Gini Importance, or Mean Decrease in Impurity is calculated as the number of times a feature is used to split a node, weighted according to the number of samples split at each node of the decision tree. Mathematically,

$$W_G(A) = \sum_{a_i \in A} w(a_i) G(a_i) \tag{3.1}$$

$$w(a_i) = \frac{entries\ with\ value\ a_i\ in\ column\ A}{total\ entries\ in\ dataset} \tag{3.2}$$

$$G(a_i) = Gini\ Index\ calculated\ for\ a_i \tag{3.3}$$

The following graph was obtained, which marks the Weighted Gini Metric for each of the three categorical features. The results suggested that the Flag variable is the most crucial categorical feature, with the Service variable is not as crucial but can still be important. The Protocol Type variable however can be removed as it provides almost no information for classification.

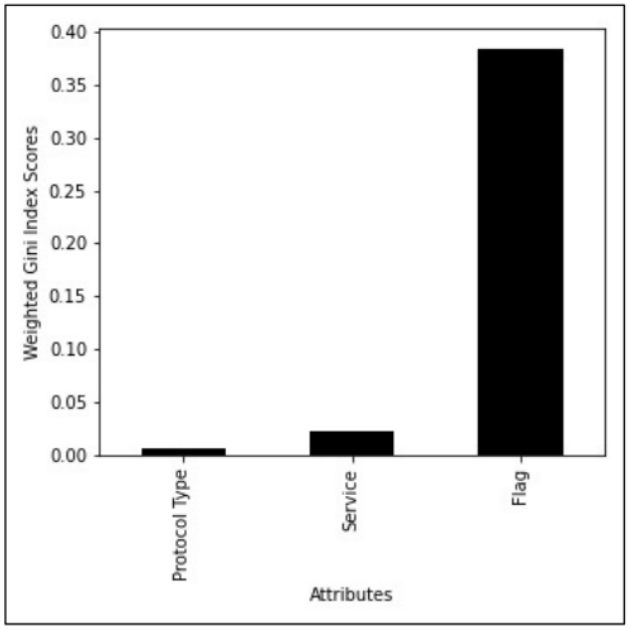

Fig 3.2 Weighted Gini Index scores for each categorical feature

3.2.2 UNIVARIATE SELECTION

Univariate selection works on the same concept as the ANOVA technique, where we compare each variable (or column or feature) to the target variable (The class of the attack) to see if there is any significant relation between the two variables. We used the chi-square variate to calculate a 'score' for each feature. The benefit of this is that the chi-square test is able to measure stochastic dependence between two features. Hence, features with the lowest chi-square scores are the most independent of the target variable and can hence be eliminated, as they are irrelevant for classification.

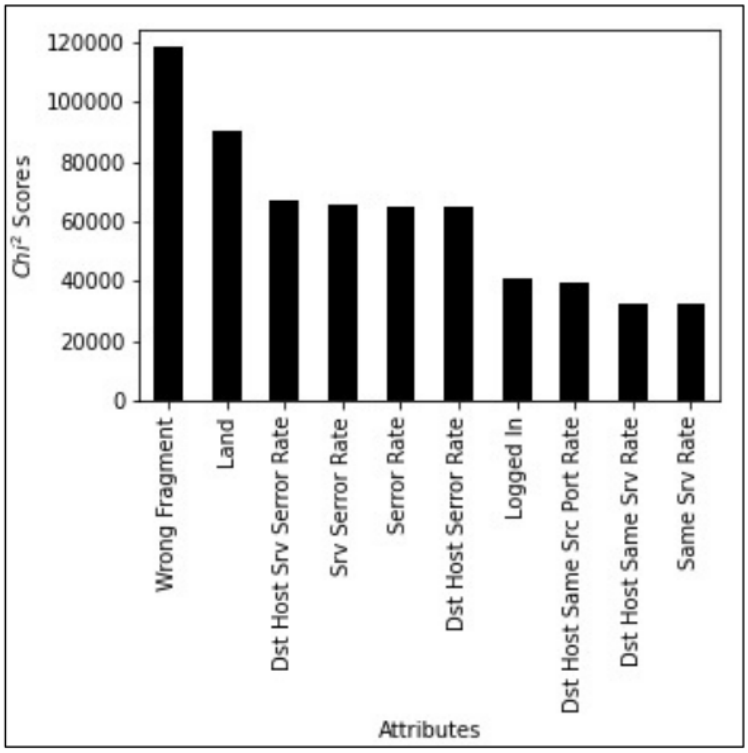

Fig 3.3 Chi-Square scores for the top 10 features

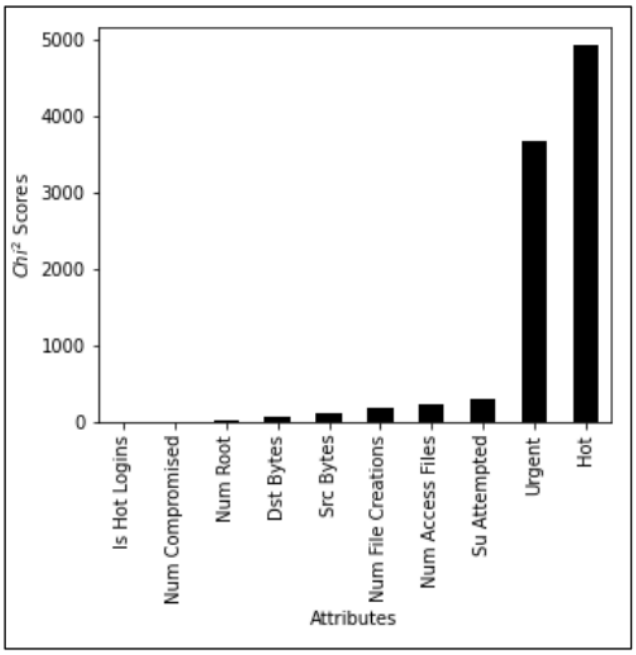

Fig 3.4 Chi-Square scores for the bottom 10 features

The above plots show the Chi-Square scores for the top 10 and the bottom 10 features. We took a conservative approach and suggested the removal of the last 2 features, namely Is_Hot_Logins and Num_Compromised, as their chi square scores were close to 0, implying complete irrelevance for classification.

3.2.3 CORRELATION PLOT

A correlation plot is essentially a symmetric matrix where the rows and columns represent features, and each element of the matrix is the Pearson's coefficient of correlation between the features in the row and the column. Correlation plots help visualize the degree of correlation between each pair of features. A higher absolute value implies a stronger linear relationship between the variables whereas the sign represents the direction of the relationship. Ideally, we wanted a linearly independent

set of features to maximize information present in the data while minimizing redundancy. Following this belief, we assumed a threshold of 0.95 (95 % correlation) and identified 5 pairs of highly correlated features.

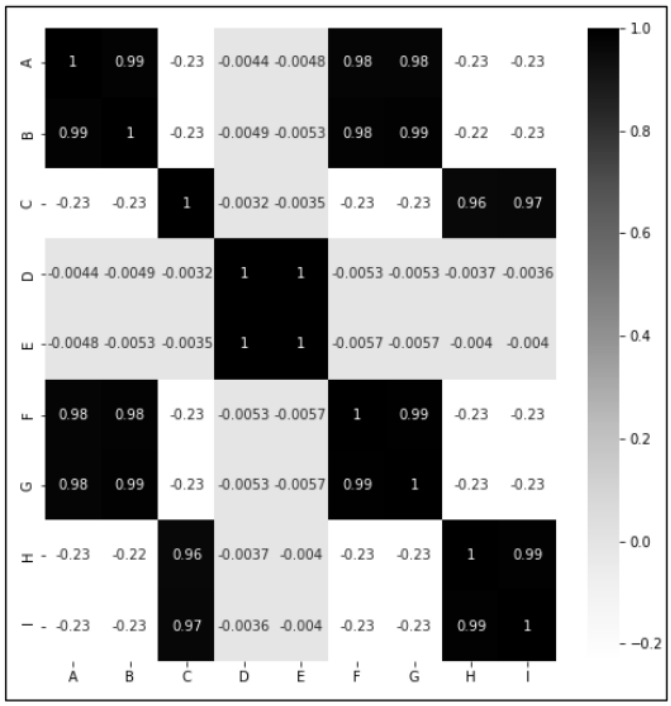

Fig 3.5 Correlation heatmap with a threshold of 0.95

Table 3.1 Corresponding legend for the above given heatmap

A	Dst_Host_Serror_Rate
B	Dst_Host_Srv_Serror_Rate
C	Dst_Host_Srv_Rerror_Rate
D	Num_Compromised
E	Num_Root

F	Serror_Rate
G	Srv_Serror_Rate
H	Rerror_Rate
I	Srv_Rerror_Rate

3.2.4 CONCLUSION

The conclusion of this analysis suggested that a total of 9 features of which 8 are numerical and 1 nominal are redundant. This reduces our feature space from 122 to 11 features. The nominal feature was chosen based on the decision tree classifier, 2 numeric features were chosen through univariate selection and 5 from the corrplot. The final numerical feature was removed as it consisted of only 0s and is hence redundant. Based on choice of threshold thresholds, further reduction is possible. For example, we could choose to remove the 'Service' variable which alone would reduce our feature space by 70. Similarly, more features could be eliminated by increasing the minimum score threshold for univariate selection. There is also further scope for feature reduction which would require further analysis of the correlation plot to identify the best features from correlated pairs or groups and retain them while eliminating the others.

3.3 DUAL STAGE INTRUSION DETECTION

The intuition behind using dual stage network intrusion detection is that it's much easier to classify malicious points first and then categorize them into 1 out of 4 classes as compared to directly classifying them into 1 out of 5 classes. There is another major benefit of using this technique. Since the percentage of some classes in the dataset are

very low, it might raise an issue of a lot of false positives. For e.g. the R2L class is only about 1% of the whole dataset as compared to the Normal class, which spans nearly 51% of the dataset. Dual stage classification would give us the following structure:

1. First, normal points are separated from the malicious ones. Since all malicious points make up nearly 49% of the dataset, this would not be a difficult task as the dataset is balanced in nature.
2. Second, classify the malicious points into 1 out of 4 classes. This problem still has data a little unbalanced since the proportion of DoS classes is much less as compared to R2L, but it is still much better than single stage classification.

We made a small change to the second stage and introduced classification into the normal class as well. This allowed us to reduce the number of false positives by a drastic factor.

The first stage was a fairly easy machine learning problem. We therefore used a few simple machine learning algorithms to achieve good accuracy. Following were the attempts we made at accomplishing stage one:

- Outlier Detection
- Random Forest Classifier
- AdaBoost Classifier

3.4 OUTLIER DETECTION

As mentioned earlier in the report, we tried to formulate an anomaly detection problem for intrusion detection. The intuition behind this was that generic web requests would have a similar feature structures, allowing them to be clustered around

a central point. Anomalous points would tend to act like outliers. Most importantly, we had to ensure that no outlier is classified as normal as that would be catastrophic. A normal point on the other hand can be classified as malicious, mainly for 2 reasons: one, the second stage of the classification process also classifies points as normal, so few normal points recognized as malicious by the first stage will get filtered out during the second stage. Two, even if some normal points get classified as malicious, it won't be an issue as typically an intrusion detection system doesn't act on the detection, it just informs the network administrator about the data packet. To accomplish outlier detection, we employed 2 methods.

3.4.1 CLUSTERING

Clustering is widely regarded as one of the most robust approaches to outlier detection. Points are clustered into n clusters, where the variable n can be controlled by the user. Each point can then be classified as an outlier based on its distance from its respective cluster centroid. Since clustering doesn't assume the probability distribution of the dataset, it normally is a very robust method, but it is time taking, as in many cases the user has to manually analyse data point distances and figure out what could be regarded as an outlier and what cannot.

We used K-Means clustering to view 3 things:

1. Binary Class Distribution
2. Multi Class Distribution
3. Malicious Data Distributions

The following images show the clusters that we obtained:

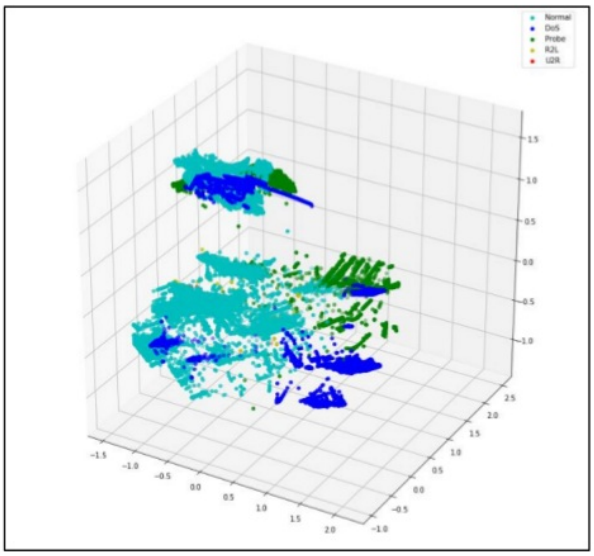

Fig 3.6 Multi Class Distribution using One Hot Encoded Vectors

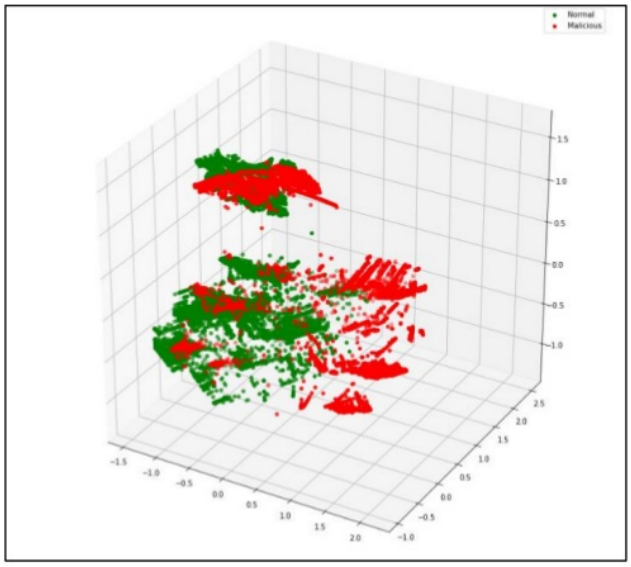

Fig 3.7 Binary Class Distribution using One Hot Encoded Vectors

The following images show the malicious class distributions:

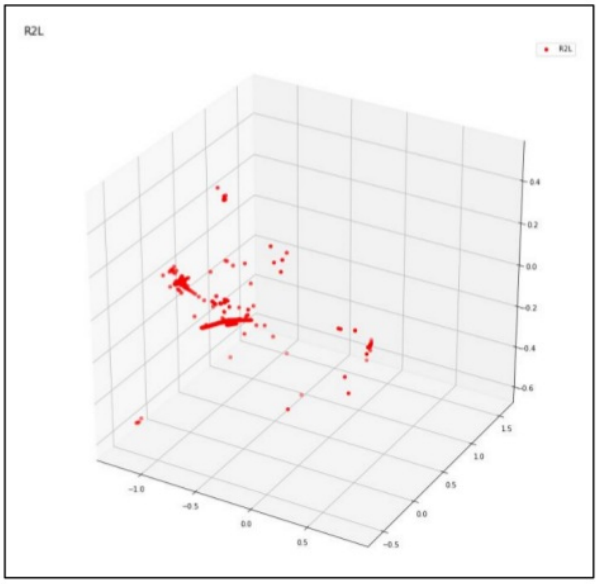

Fig 3.8 R2L Cluster

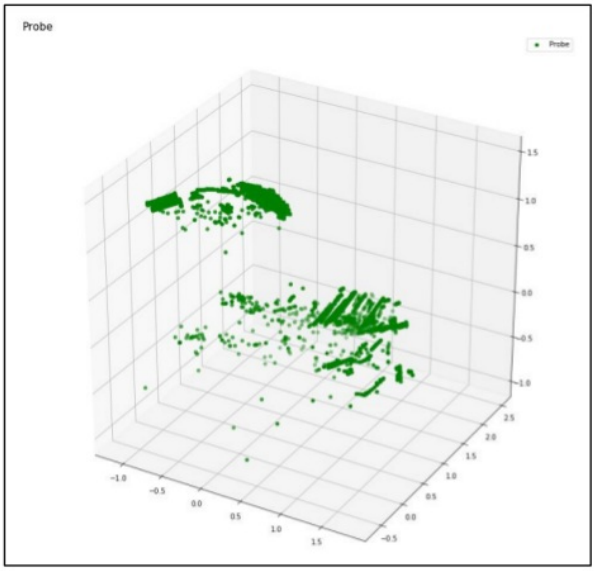

Fig 3.9 Probe Cluster

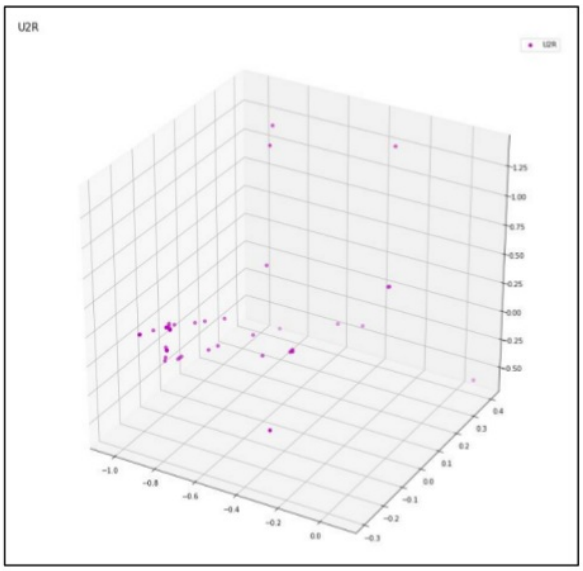

Fig 3.10 U2R Cluster

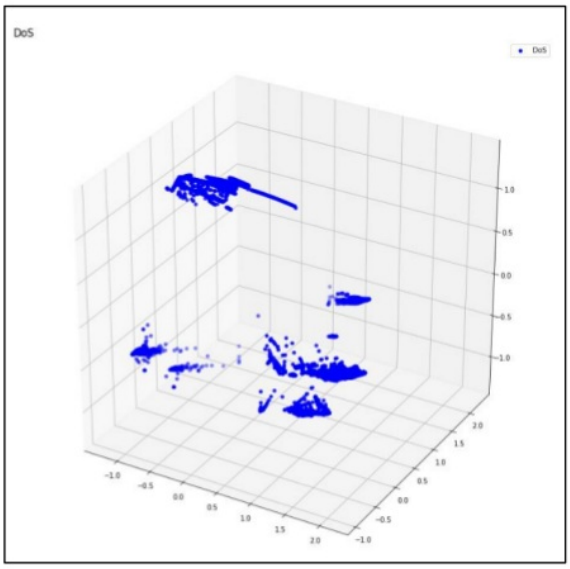

Fig 3.11 DoS Cluster

We also clustered all normal points into 4 clusters, which was the optimal number of clusters as identified by the K-Elbow Visualizer:

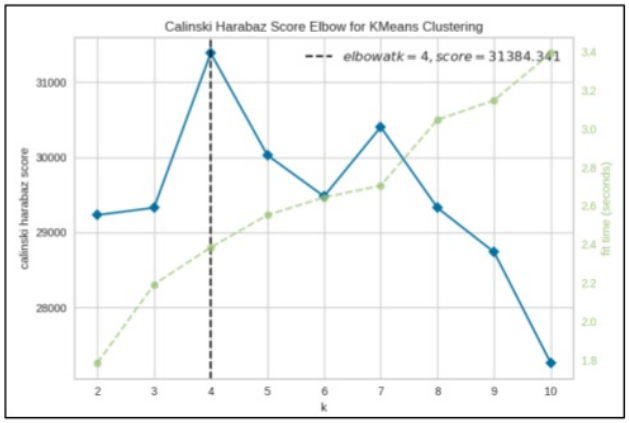

Fig 3.12 Calinski Harabaz Score for clustering

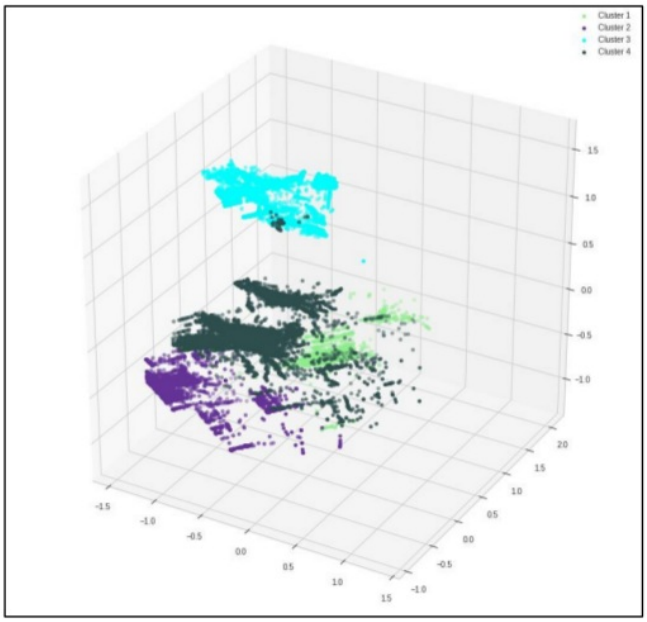

Fig 3.13 K-Means Clustering on normal data points of training data

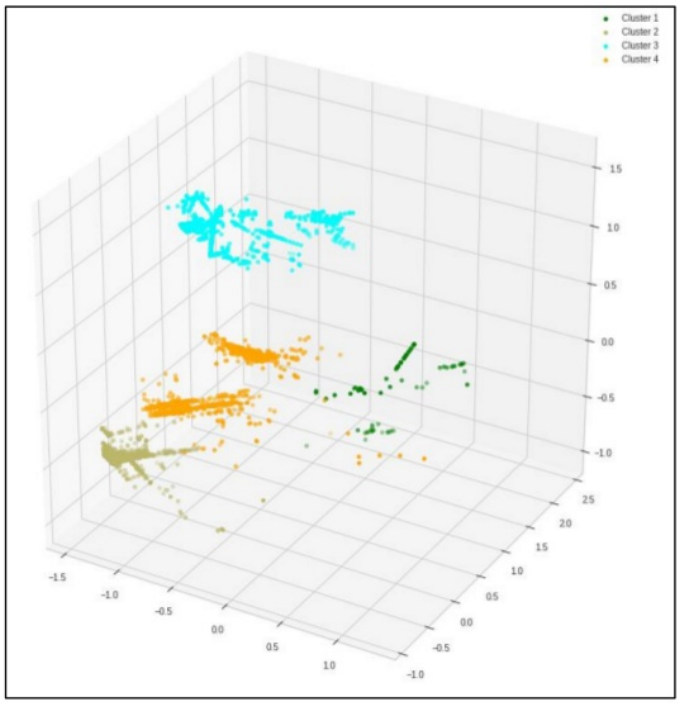

Fig 3.14 K-Means Clustering on normal data points of testing data

3.4.2 Z-SCORE ANALYSIS

Z-Score analysis is a popular method in the data science community for anomaly detection. It takes one fundamental assumption about the dataset, that it roughly represents the normal distribution. Since the applications of the normal distribution are wide, in some cases it is reasonable to assume such a thing.

Following is the formula to compute the z-score for a point:

$$z = \frac{x-\mu}{\sigma} \qquad (3.4)$$

where,

- x is the raw score
- μ is the population mean
- σ is the population standard deviation

The raw score is something that is left to the user to compute. In our case we used the distance from the cluster centroids as the raw score. Assuming that the distances to each cluster centroid is roughly normal in nature is much safer as compared to assuming the distribution of the dataset itself.

As can be seen, this allows the user to focus on the more algorithmic portions of the system and not how outlier detection will be done. This can give bad results if the distribution of the raw score is not normally distributed.

3.5 RANDOM FOREST CLASSIFIER

A random forest classifier is a collection of decision tree classifiers, each with its own set of parameters. It falls under the category of ensemble classifiers and aggregates votes from all of its constituent classifiers.

Random forest classifiers are very robust to outliers and hence were suitable to our application. We used grid search to search for the best set of parameters and optimized the configuration over the recall score.

Following are the plots we obtained for the optimal random forest classifier:

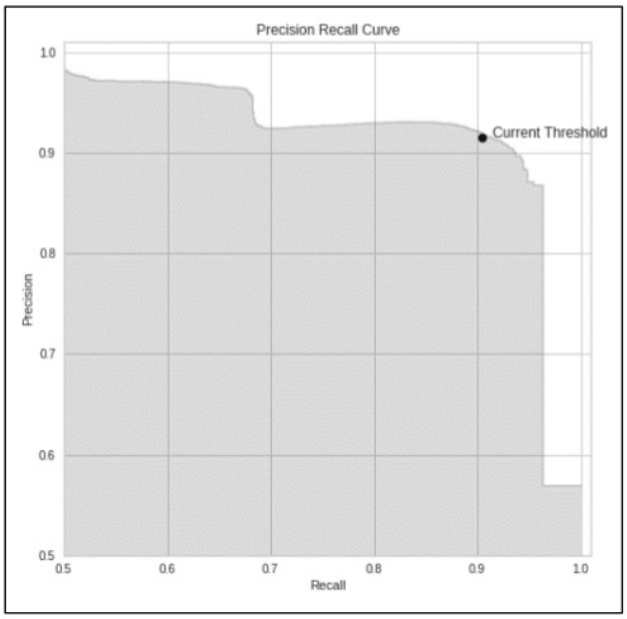

Fig 3.15 Precision recall curve for Random Forest

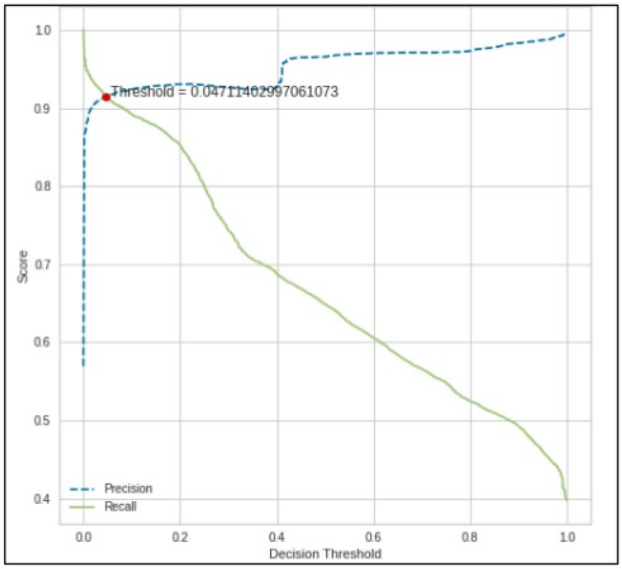

Fig 3.16 Precision and recall scores as a function of the decision threshold for Random Forest

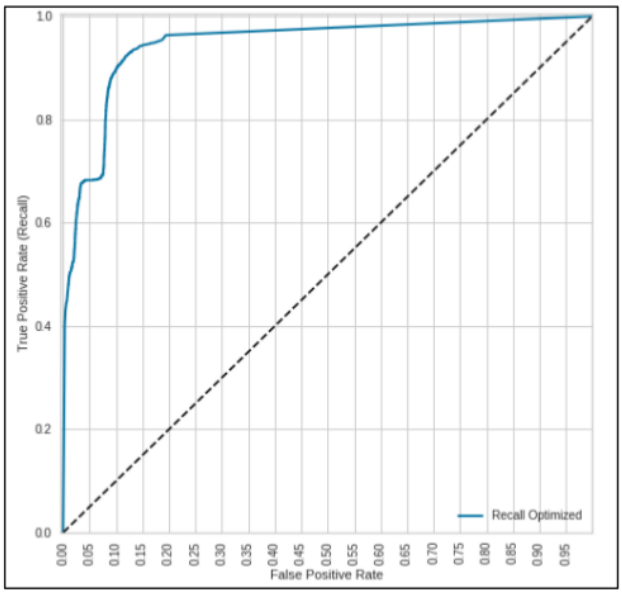

Fig 3.17 ROC Curve for Random Forest

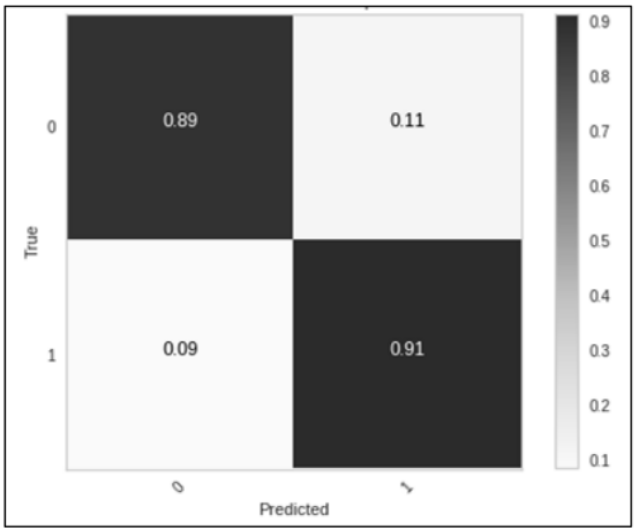

Fig 3.18 Random Forest classifier with optimized recall

3.6 ADABOOST CLASSIFIER

Boosting is another ensemble technique. AdaBoost also works on the principle of combining a lot of weak classifiers to make one strong classifier. The weak learners in AdaBoost are decision tree with a single split, known as decision stumps. It works by putting more weight on training instances that were difficult to classify and less weight on those that were handled well.

To obtain an optimal set of parameters, we used grid search with optimization over the recall score.

The following plots were obtained for the most optimal AdaBoost classifier:

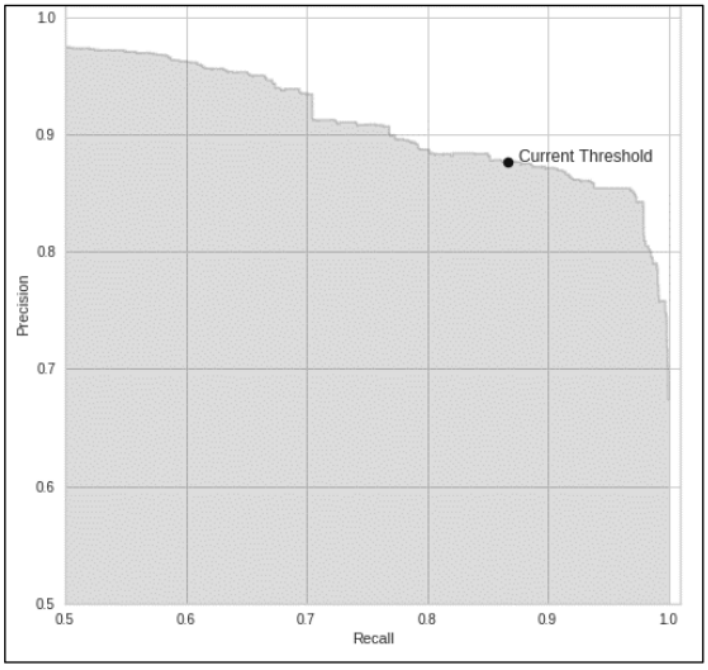

Fig 3.19 Precision recall curve for AdaBoost

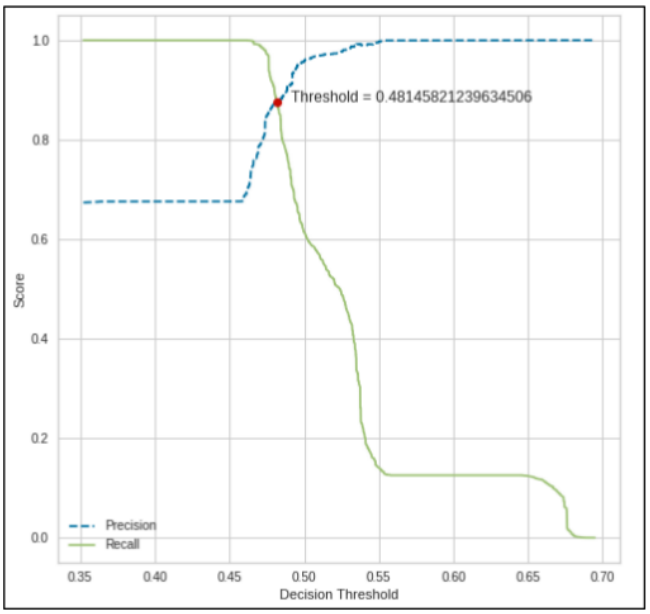

Fig 3.20 Precision and recall scores as a function of the decision threshold for AdaBoost

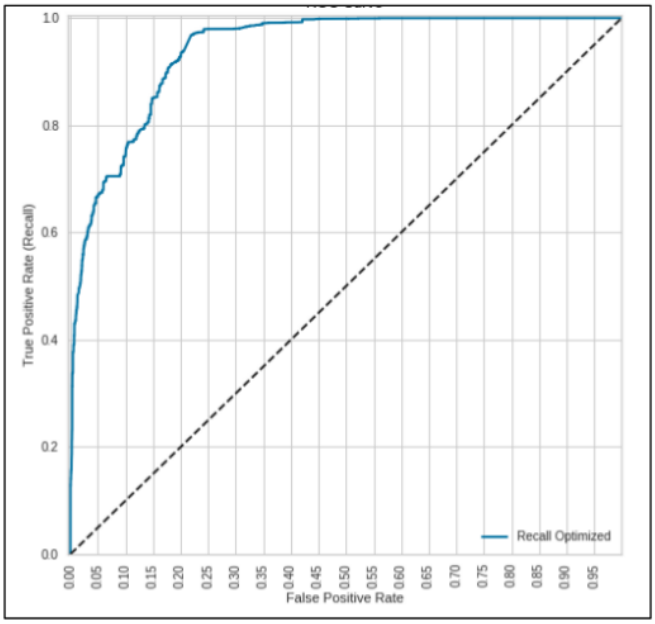

Fig 3.21 ROC Curve for AdaBoost

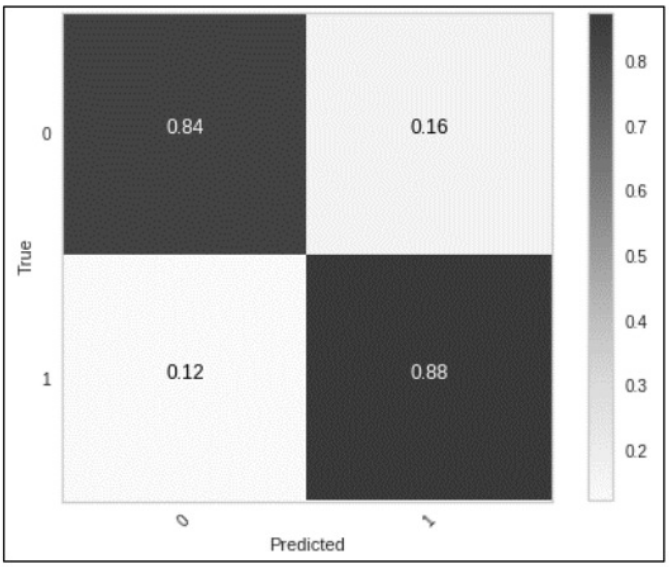

Fig 3.22 AdaBoost Classifier with optimized recall

3.7 CONVOLUTIONAL NETWORK

For the second stage of the classification process, we experimented with a lot of network architectures but CNNs worked out best for us. They yielded the final results published. We used the following architecture for the CNN:

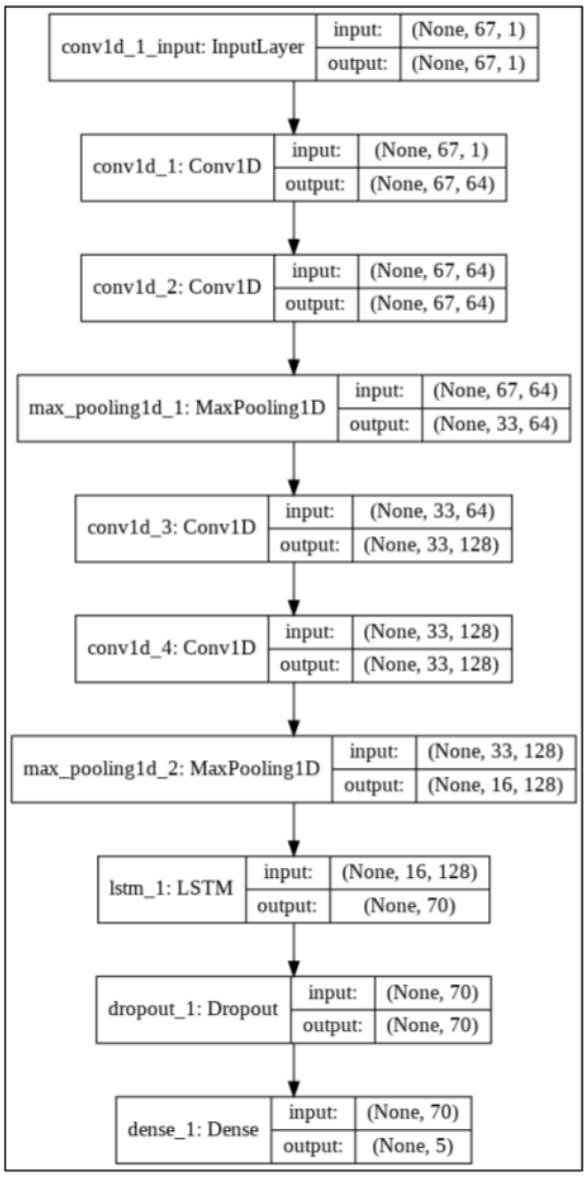

Fig 3.23 Architecture of the network

Following are the plots for the final results that we got after both stages of classification are done:

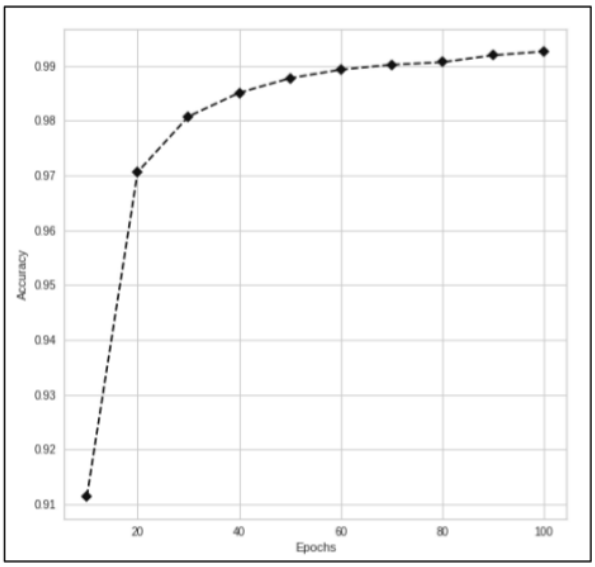

Fig 3.33 Graph representing accuracy vs number of epochs

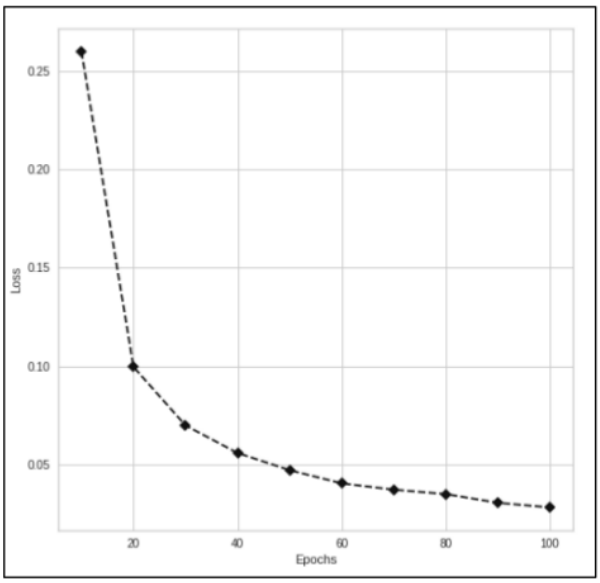

Fig 3.34 Graph representing loss vs number of epochs

CHAPTER 4 RESULTS AND FUTURE SCOPE

In our experiment, we compared three different methods for binary classification, namely the Random Forest Classification, AdaBoost and XGBoost. All three are special ensemble models that combine several simple models to deliver the final result. Parameter tuning was done using grid search on a suitable search space, while using the model's recall as the metric to be optimized. Subsequently, we plotted the precision and recall of each model to find an optimal threshold for classification into malicious or benign.

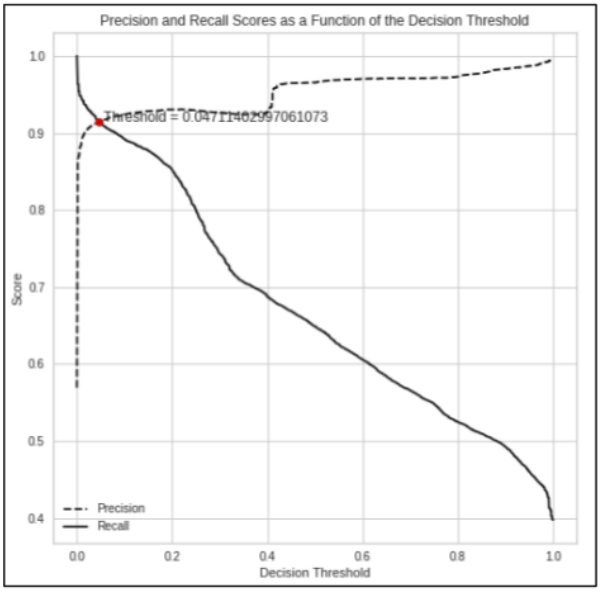

Fig 4.1 Precision and recall scores as a function of the decision threshold for Random Forest

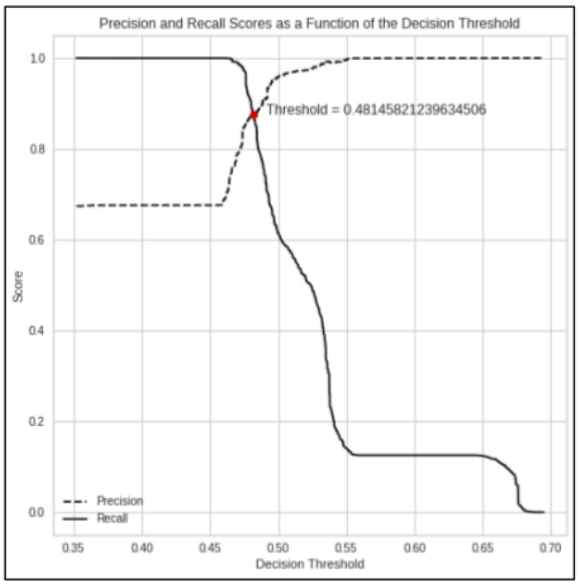

Fig 4.2 Precision and recall scores as a function of the decision threshold for AdaBoost

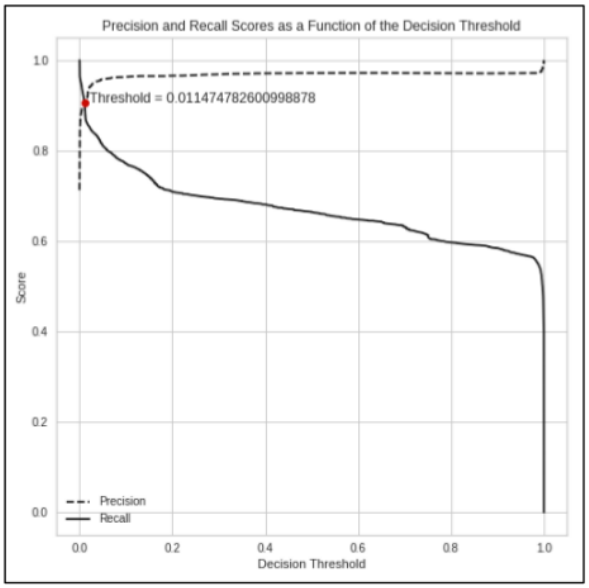

Fig 4.3 Precision and recall scores as a function of the decision threshold for XGBoost

The threshold value for each model was chosen as the point of intersection of recall and precision scores as experiments have proved such points to be optimal.

The Random Forest Classifier proved to be the best model for classification, with a recall of 91% and precision of 89%. While XGBoost had the same recall, its precision score was lower at 88%. AdaBoost performed the worst of the three, with recall and precision scores of 88% and 84%. Random forest gave an accuracy of 89%.

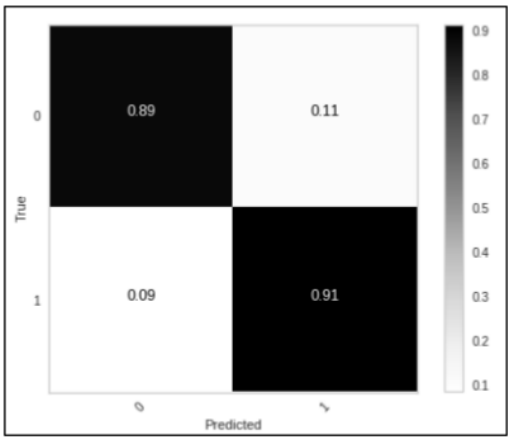

Fig 4.4 Confusion matrix for Random Forest

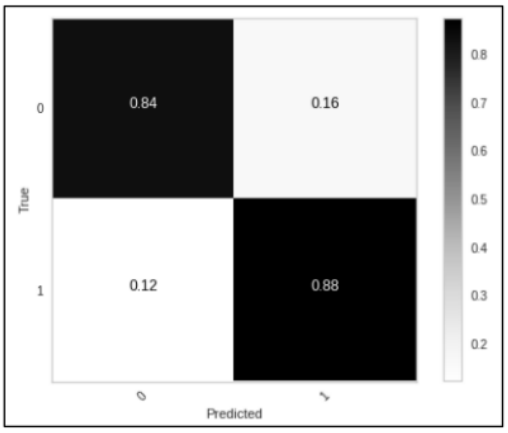

Fig 4.5 Confusion matrix for AdaBoost

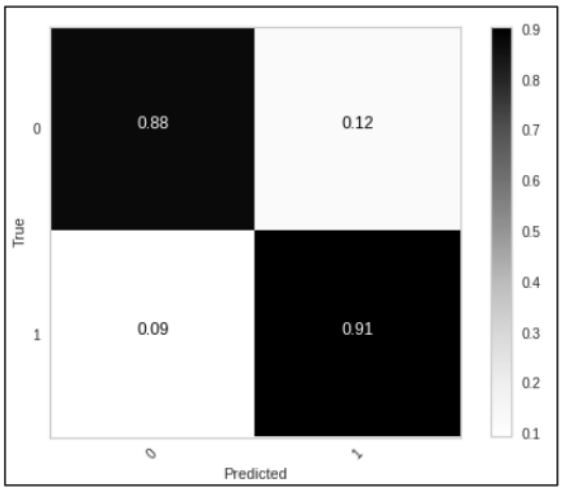

Fig 4.6 Confusion matric for XGBoost

We considered using various network architectures for the purpose of multi class classification. After looking into the use, applicability and complexity of various models such as CNNs, RNNs, LSTMs, ANN etc., we decided to use the following ANN architecture as it yielded the most favorable results when trained over 100 epochs. Figs 8 and 9 show the loss and accuracy convergence of the multi class classifier over 100 epochs. Figure 10 shows the confusion matrix of the classifier. It can be seen that the classifier performs perfectly when classifying datapoints as Normal, or belonging to the DoS or Probe attack classes, yielding an accuracy of 100% . It performs well on the U2R class with an accuracy of 98% but falters in the R2L class, with an accuracy of only 67%. This can be attributed to the disproportionately small number of examples of R2L attack types in the training set due to which the classifier is unable to memorize the distinct patterns pertaining to the class.

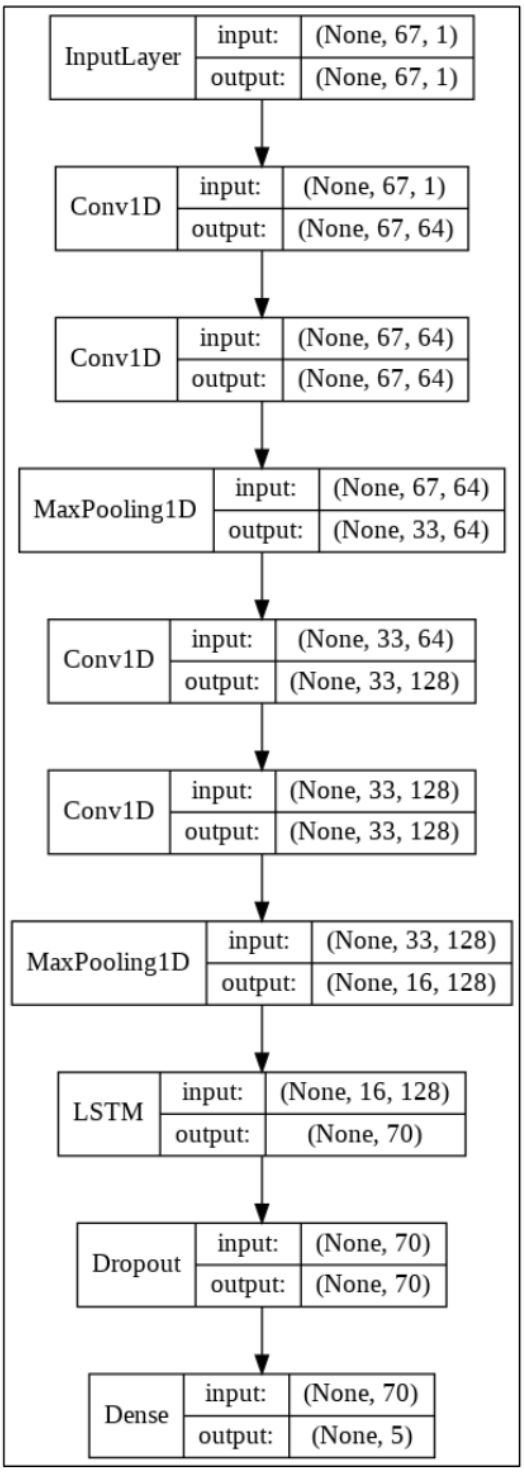

Fig 4.7 Architecture of the network

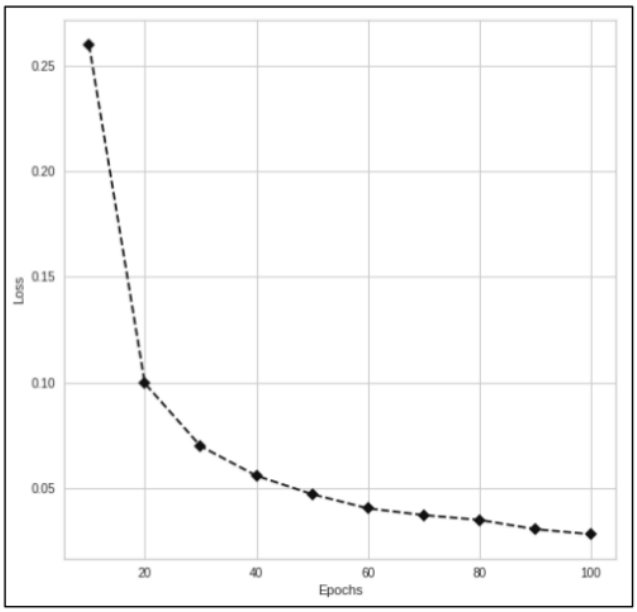

Fig 4.8 Graph representing loss vs number of epochs

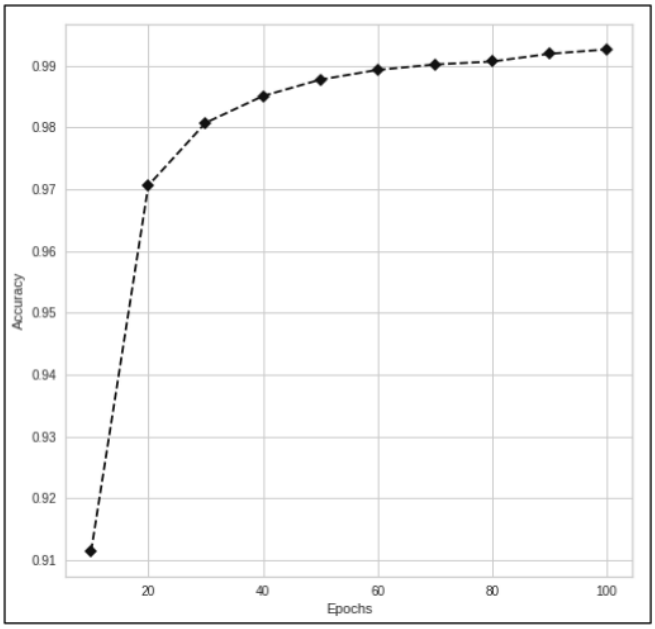

Fig 4.9 Graph representing accuracy vs number of epochs

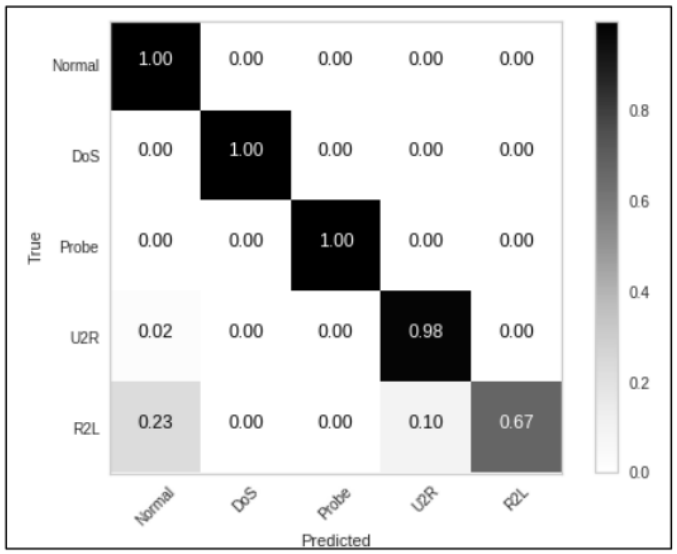

Fig 4.10 Confusion matrix of the different classes of attacks

In conclusion, the two-stage classification algorithm suggested in the paper resulted in improved evaluation results than existing methods. A reduced feature space helps ensure that forward propagation through the model is fast in order to minimize detection overhead and reduce costs.

Our data analysis suggested that a total of 9 features, and maybe even more could be eliminated from the feature space. As this elimination leads to removal of redundancies in the dataset, we can expect our model to perform better. However, this hypothesis is yet to be tested via experimentation.

Future scopes for this architecture also include the use of GAN to generate samples to compensate for the low proportion of samples belonging to the U2R and R2L classes and generate an improved feature space to enhance the performance of the model.

CHAPTER 5 ACHIEVEMENTS

Our efforts bore fruit when our original paper was accepted to be presented at the 4th International Conference on Mathematical Models and Computational Techniques in Science and Engineering (MMCTSE) held from February 22 to February 24, 2020 in London, UK. Our group member, Somya Gupta attended the conference to present the paper in front of an accomplished audience of scholars and professors from all over the world. The paper submitted and accepted by the conference is available in Appendix 1, along with the email confirming the acceptance (Appendix 2). The proceedings of the conference will also be published in the Scopus indexed Journal of Physics: Conference Series – IOPScience. The mail confirming the same is available in the Appendix.

Following the conference, an extended version of the paper was accepted to be published in the Scopus indexed journal: World Scientific and Engineering Academy and Society (WSEAS) transactions on Systems and Control, and has received the ID 298 in Volume 15 of the Journal. The link for the same is given in Appendix 4. The paper submitted for the journal is also available in Appendix 3, along with confirmation email for the same (Appendix 4).

Major Report

ORIGINALITY REPORT

12%	9%	7%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.wseas.org Internet Source	3%
2	Submitted to Rajiv Gandhi Proudyogiki Vishwavidyalaya Student Paper	3%
3	www.ijarcce.com Internet Source	<1%
4	towardsdatascience.com Internet Source	<1%
5	Submitted to Sunway Education Group Student Paper	<1%
6	Submitted to University of Sunderland Student Paper	<1%
7	docs.rapidminer.com Internet Source	<1%
8	Dipanjan Sarkar, Raghav Bali, Tushar Sharma. "Practical Machine Learning with Python", Springer Science and Business Media LLC, 2018	<1%

	Publication	
9	www.mif.vu.lt Internet Source	<1%
10	Soham Chatterjee, Vaidheeswaran Archana, Karthik Suresh, Rohit Saha, Raghav Gupta, Fenil Doshi. "Detection of non-technical losses using advanced metering infrastructure and deep recurrent neural networks", 2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe), 2017 Publication	<1%
11	Submitted to Pentecost University College Student Paper	<1%
12	samikshyagautam.com Internet Source	<1%
13	nagoya.repo.nii.ac.jp Internet Source	<1%
14	Submitted to De Montfort University Student Paper	<1%
15	Submitted to Universiti Putra Malaysia Student Paper	<1%
16	Submitted to University of Warwick Student Paper	<1%

17	Submitted to Swinburne University of Technology Student Paper	<1%
18	Submitted to University of Exeter Student Paper	<1%
19	Submitted to University of KwaZulu-Natal Student Paper	<1%
20	Submitted to Dr. S. P. Mukherjee International Institute of Information Technology (IIIT-NR) Student Paper	<1%
21	Submitted to Chester College of Higher Education Student Paper	<1%
22	blog.successwithagile.com Internet Source	<1%
23	Ossama Embarak. "Chapter 2 The Importance of Data Visualization in Business Intelligence", Springer Science and Business Media LLC, 2018 Publication	<1%
24	link.springer.com Internet Source	<1%
25	Submitted to University of Cape Town Student Paper	<1%
	Submitted to Virginia International University	

26	Student Paper	<1%
27	John Vince. "Chapter 4 Trigonometry", Springer Science and Business Media LLC, 2014 Publication	<1%
28	Submitted to University College London Student Paper	<1%
29	www23.us.archive.org Internet Source	<1%
30	Sireesha Rodda, Uma Shankar Rao Erothi. "Class imbalance problem in the Network Intrusion Detection Systems", 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), 2016 Publication	<1%
31	"Emerging Research in Computing, Information, Communication and Applications", Springer Science and Business Media LLC, 2019 Publication	<1%
32	Submitted to University of Pretoria Student Paper	<1%
33	Submitted to University of Queensland Student Paper	<1%
34	Submitted to University of Aberdeen Student Paper	<1%

35	www.napier.ac.uk Internet Source	<1%
36	Submitted to University of Derby Student Paper	<1%
37	Submitted to University of the Western Cape Student Paper	<1%
38	www.gangboard.com Internet Source	<1%
39	"Advances in Information and Communication", Springer Science and Business Media LLC, 2020 Publication	<1%
40	Submitted to University of Hertfordshire Student Paper	<1%
41	Submitted to University of Florida Student Paper	<1%
42	Bini Alias, R Karthika, Latha Parameswaran. "Classification of High Resolution Remote Sensing Images using Deep Learning Techniques", 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2018 Publication	<1%
43	Rupal Bhargava, Yashvardhan Sharma. "Deep Extractive Text Summarization", Procedia	<1%

Computer Science, 2020

Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

Major Report

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60