DUAL STAGE NETWORK INTRUSION DETECTION SYSTEMS THROUGH FEATURE REDUCTION

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY
IN
MATHEMATICS AND COMPUTING

Submitted by:

Manan Lohia (2K16/MC/049)
Raghuvansh Raj (2K16/MC/062)
Somya Gupta (2K16/MC/079)

Under the supervision of

DR. H.C. TANEJA

Department of Applied Mathematics
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

MAY, 2020

Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CANDIDATE'S DECLARATION

We, Manan Lohia (2K16/MC/049), Raghuvansh Raj (2K16/MC/062) and Somya Gupta (2K16/MC/079) students of B.Tech. (Mathematics and Computing), hereby declare that the project Dissertation titled "Dual Stage Network Intrusion Detection Systems through Feature Reduction" which is submitted by us to the Department of Mathematics, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Manan	[illegible]	[illegible]
MANAN LOHIA	RAGHUVANSH RAJ	SOMYA GUPTA

Place: Delhi
Date: 22/5/2020

Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "Dual Stage Network Intrusion Detection Systems through Feature Reduction" which is submitted by Manan Lohia (2K16/MC/049), Raghuvansh Raj (2K16/MC/062) and Somya Gupta (2K16/MC/079), Department of Mathematics, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

[illegible]

Place: Delhi
Date: 22/5/2020

Dr. H.C. Taneja
SUPERVISOR
Professor
Department of Applied Mathematics
Delhi Technological University, Bawana Road, Delhi - 110042

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.

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without acknowledging the people who made it all possible and whose constant guidance secured us the success.

We are grateful to Dr H.C. Taneja (Professor, Department of Applied Mathematics), Delhi Technological University (formerly Delhi College of Engineering), New Delhi for his aspiring guidance, invaluably constructive criticism and friendly advice during the project work.

We also place on record, our immense gratitude to one and all who directly or indirectly have lent their helping hand in this venture. We feel proud and privileged in expressing our deep sense of gratitude to all those who have helped us in presenting this project.

Last but never the least, we thank our parents for always being with us, in every sense.

Manan	Raghuvansh Raj	Somya
MANAN LOHIA	RAGHUVANSH RAJ	SOMYA GUPTA

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS, ABBREVIATIONS

3D – 3 Dimensional

ANN – Artificial Neural Network

ANOVA – Analysis of Varaiance

CLI – Command Line Interface

CNN – Convolutional Neural Network

CPU – Central Processing Unit

DoS – Denial of Service

GAN – General Adversarial Network

GCP – Google Cloud Platform

GRU – Gated Recurrent Unit

LSTM – Long Short-Term Memory

ML – Machine Learning

NIDS – Network Intrusion Detection System

NLP – Natural Language Processing

OS – Operating System

PCA – Principal Component Analysis

PDF – Portable Document Format

R2L – Remote to Local

RAM – Random Access Memory

RNN – Recurrent Neural Network

ROC – Receiver Operating Characteristics

SIMD – Single Instruction, Multiple Data

SkLearn – SciKit-Learn

U2R – User to Root

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. Redundant records are removed to enable the classifiers to produce an un-biased result.
2. Sufficient number of records are available in the train and test data sets, which is reasonably rational and enables the execution of experiments on the complete set.
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 42nd attribute contains data about the various 5 classes of network connection vectors and they are categorized as one normal class and four attack classes. The 4 attack classes are further grouped as

DoS, Probe, R2L and U2R. 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 performed as a root in the connection	0
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 the connections aggregated in the count (23)	1
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

Table 1.5 (continued)

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:

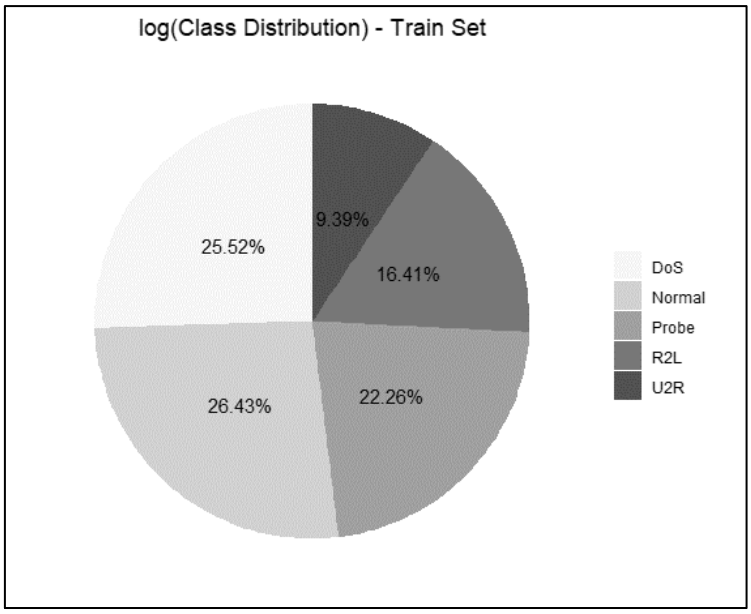


Fig 1.1 log(class distribution) of training set

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

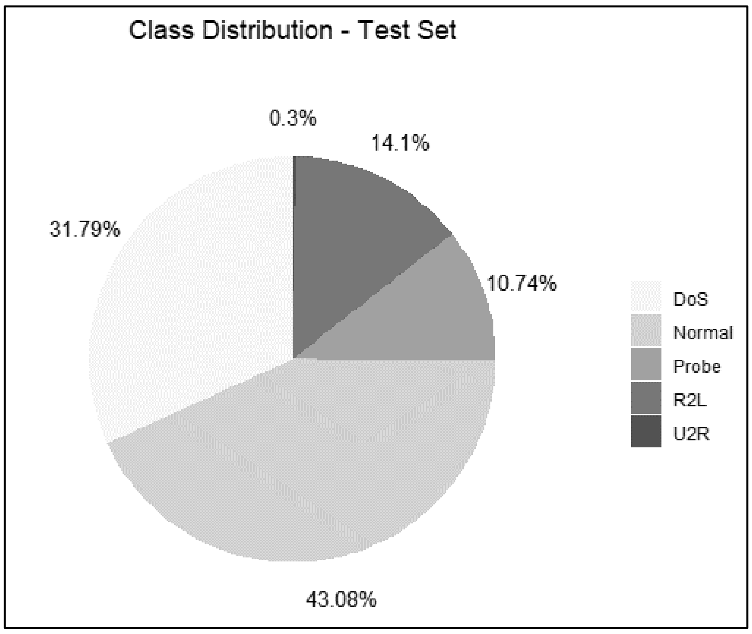


Fig 1.2 Class distribution of test set

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

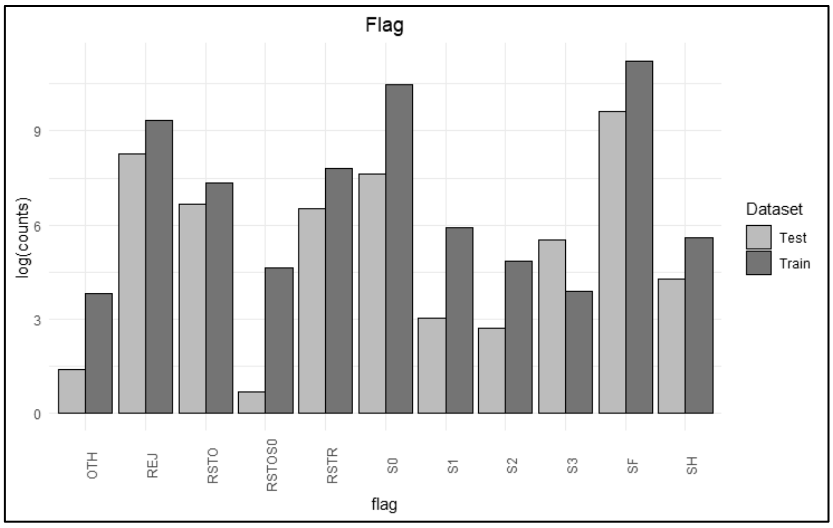


Fig 1.3 log(counts) of each category of flag attribute

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

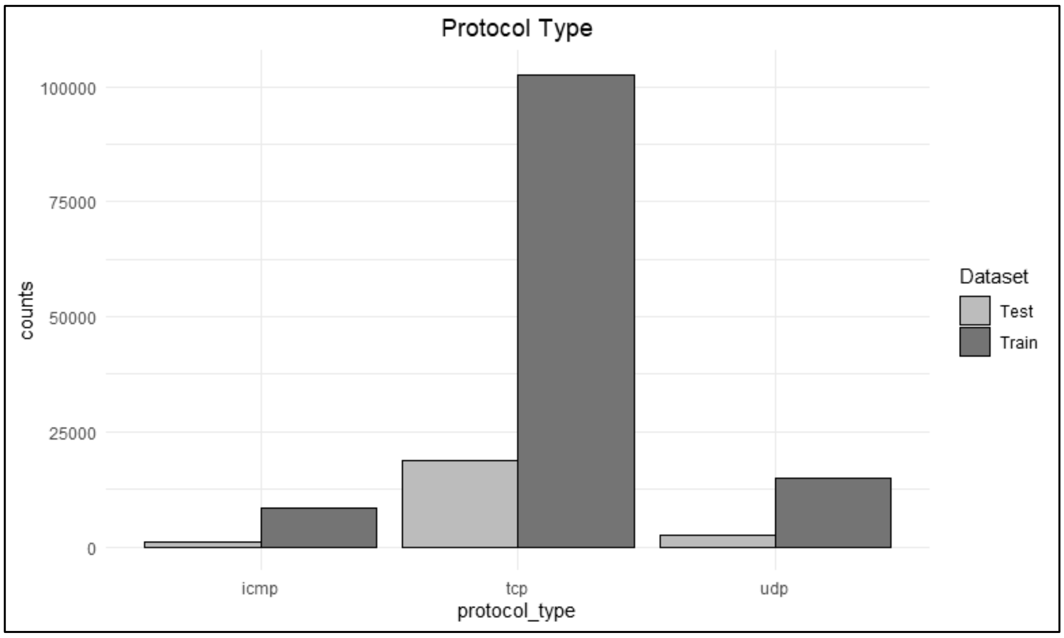


Fig 1.4 Count of each category of protocol_type attribute

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

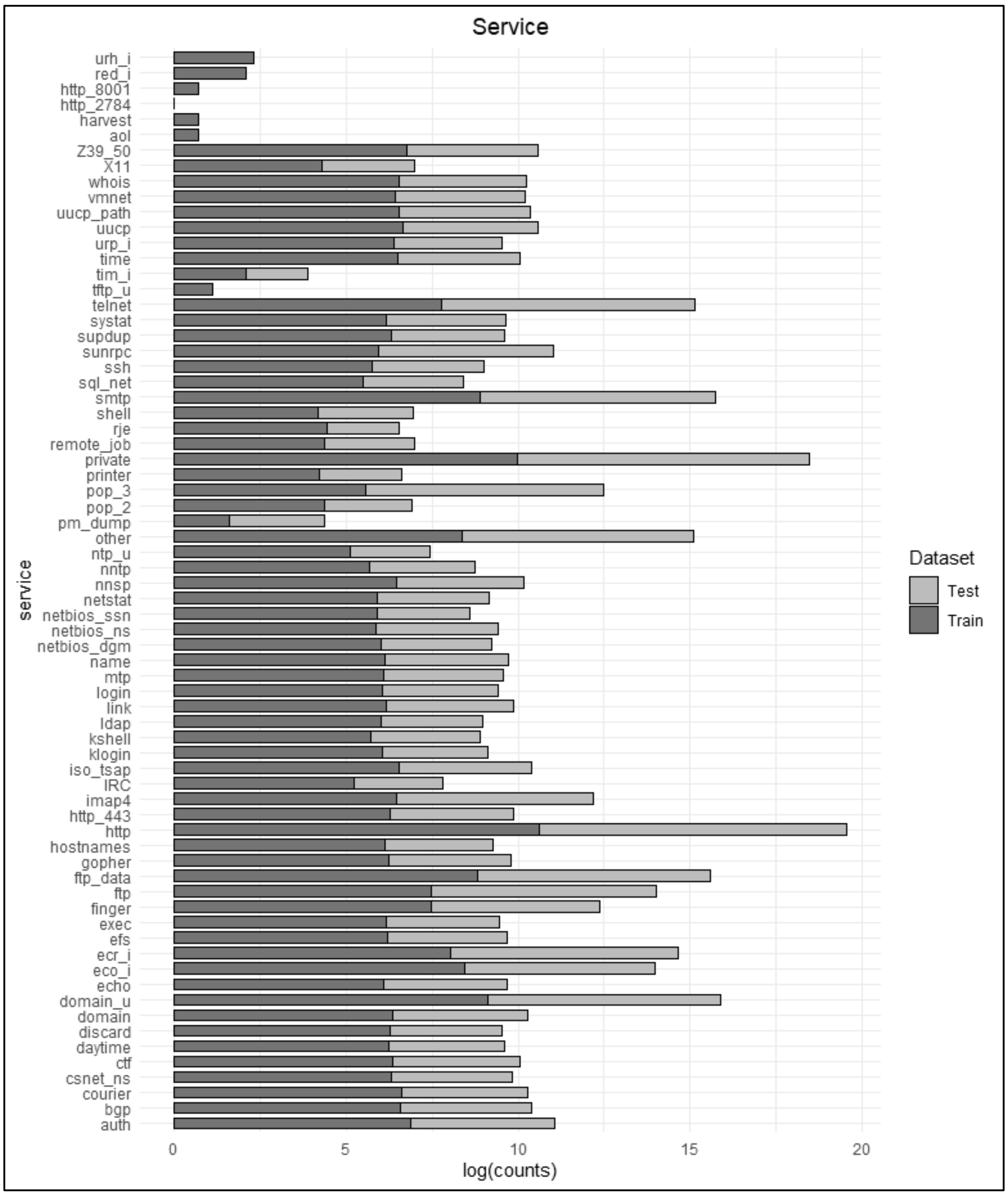


Fig 1.5 log(count) of each category of service attribute

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.

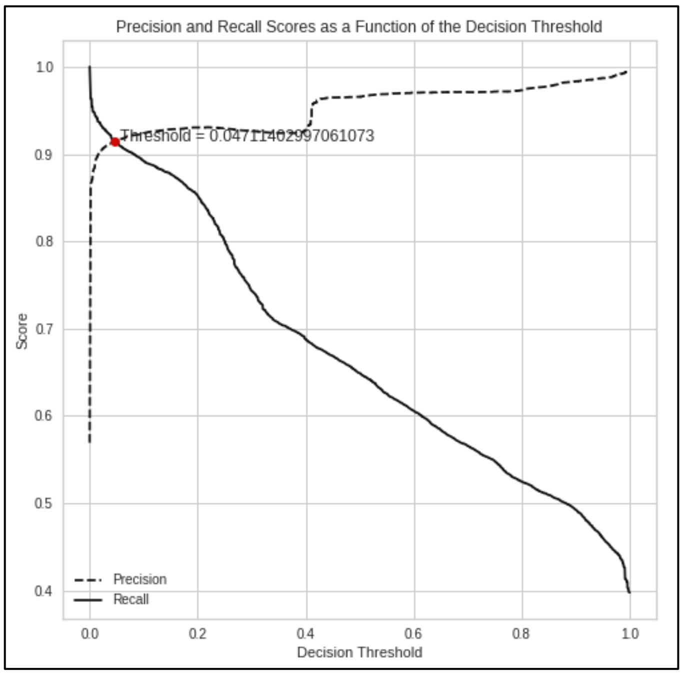


Fig 3.1 Precision and Recall Vs Threshold Curve

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.

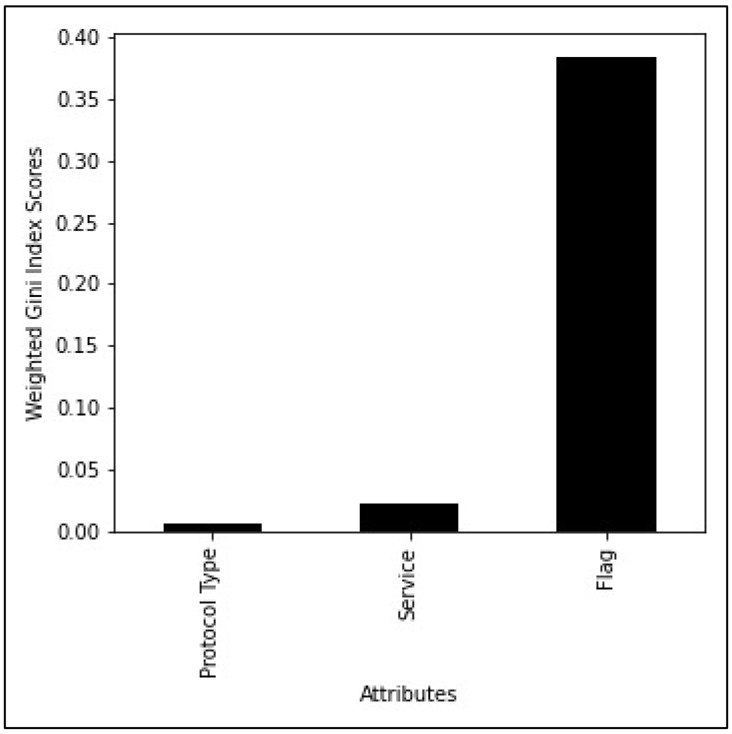


Fig 3.2 Weighted Gini Index scores for each categorical feature

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.

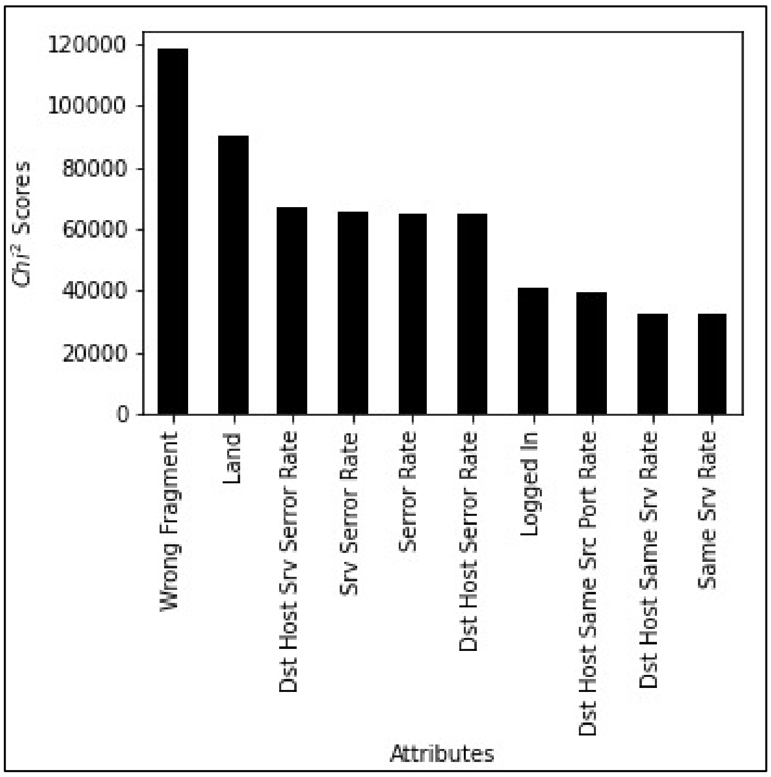


Fig 3.3 Chi-Square scores for the top 10 features

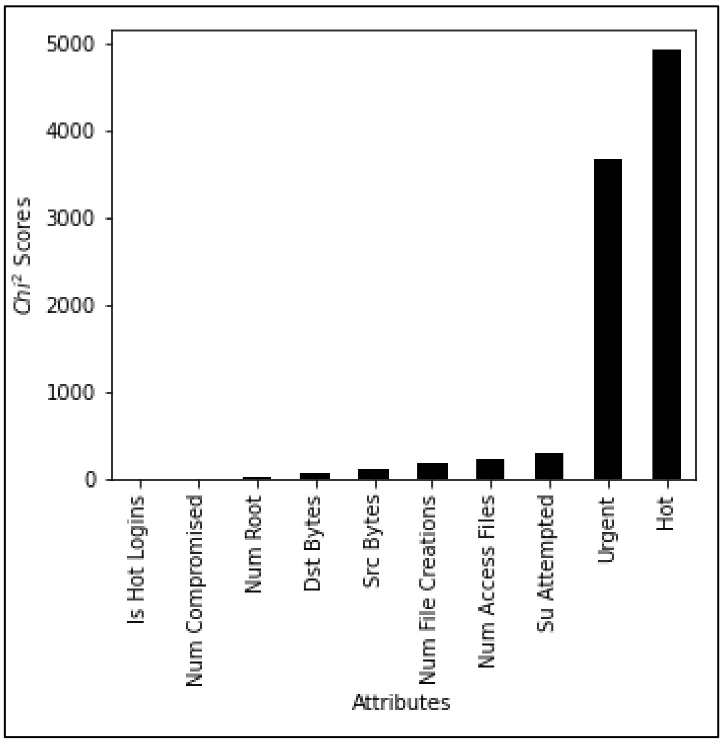


Fig 3.4 Chi-Square scores for the bottom 10 features

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.

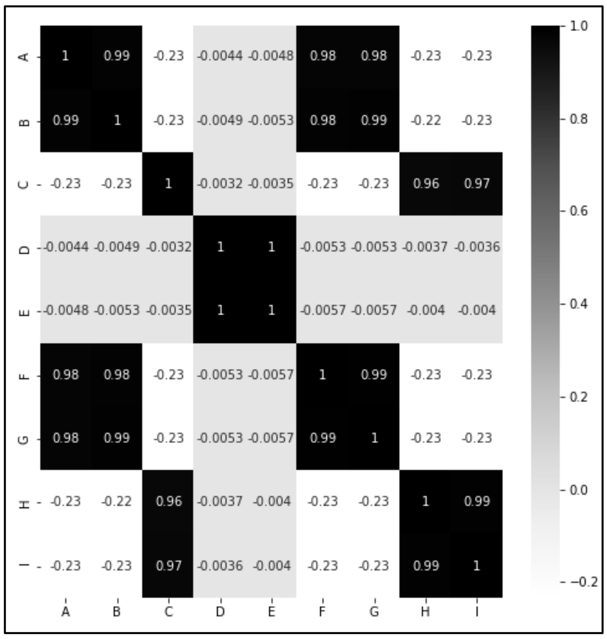


Fig 3.5 Correlation heatmap with a threshold of 0.95

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:

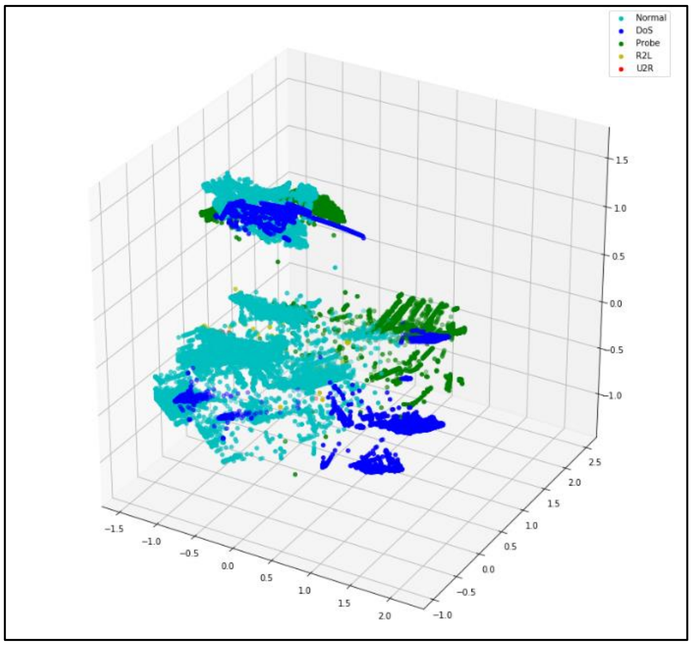


Fig 3.6 Multi Class Distribution using One Hot Encoded Vectors

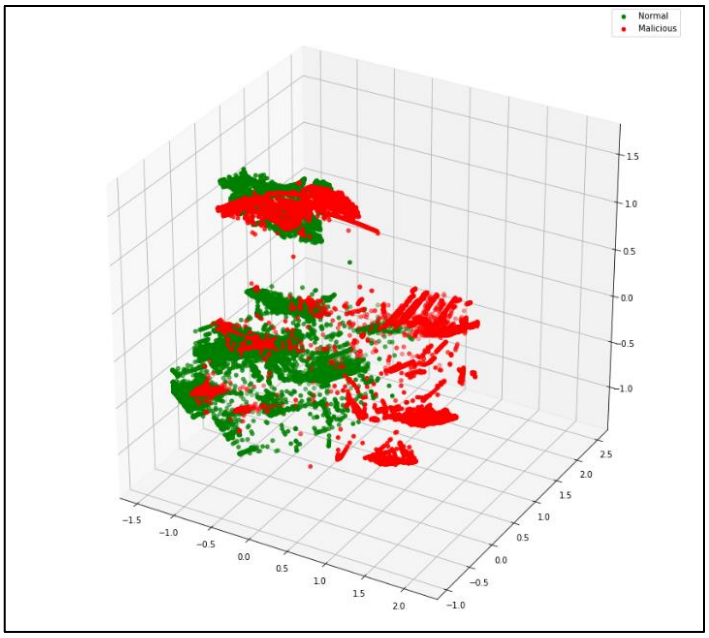


Fig 3.7 Binary Class Distribution using One Hot Encoded Vectors

The following images show the malicious class distributions:

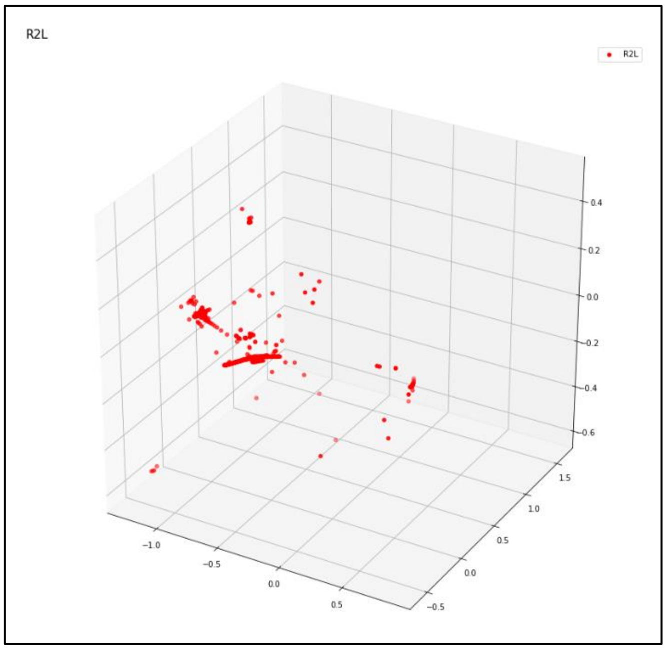


Fig 3.8 R2L Cluster

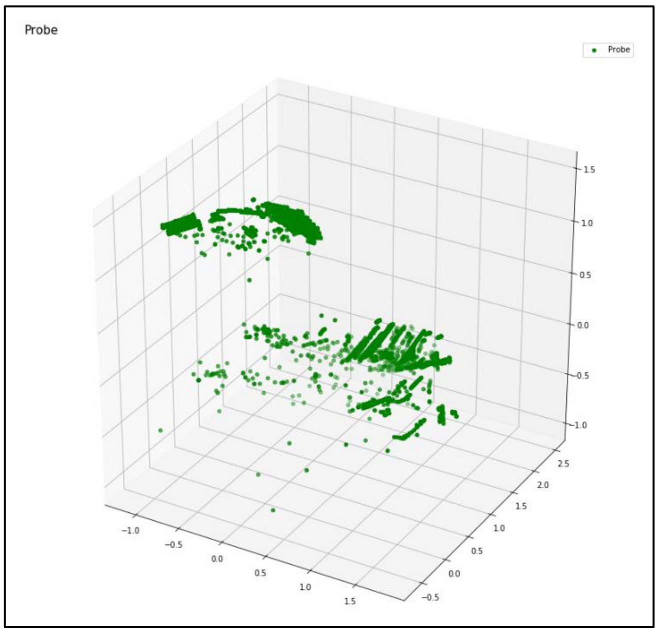


Fig 3.9 Probe Cluster

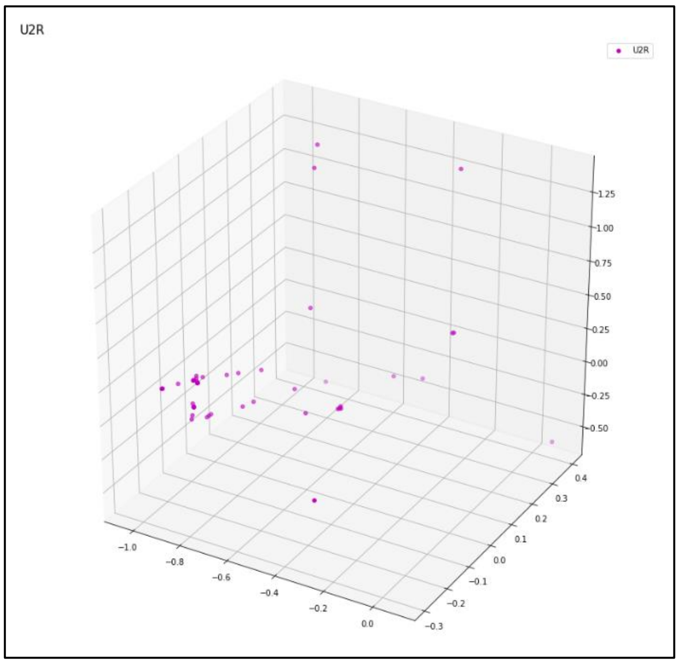


Fig 3.10 U2R Cluster

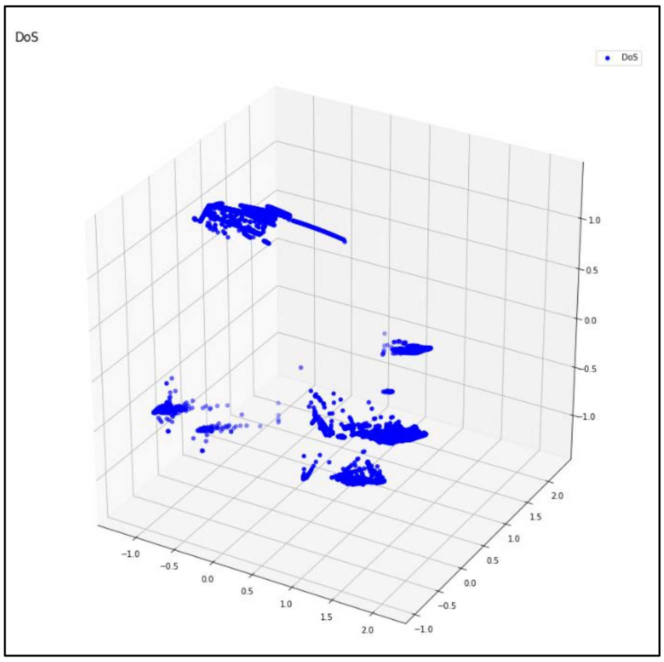


Fig 3.11 DoS Cluster

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

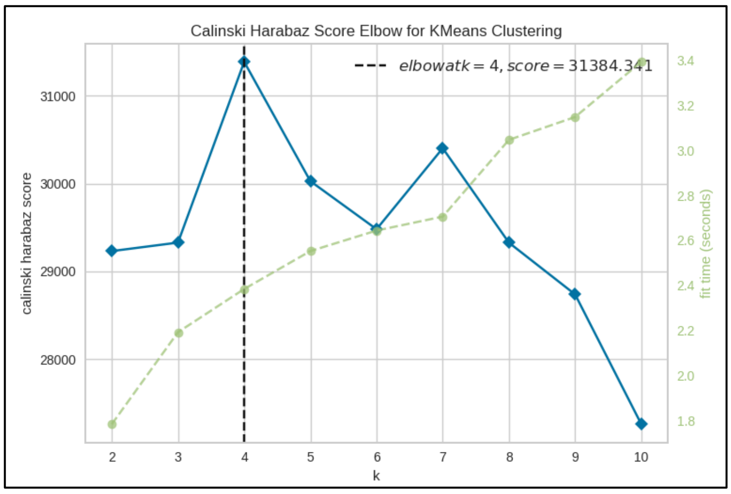


Fig 3.12 Calinski Harabaz Score for clustering

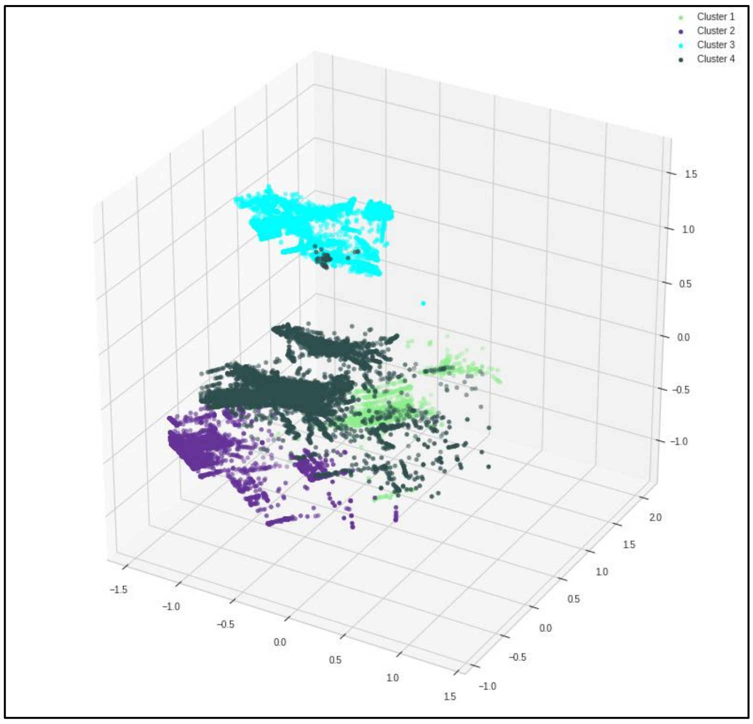


Fig 3.13 K-Means Clustering on normal data points of training data

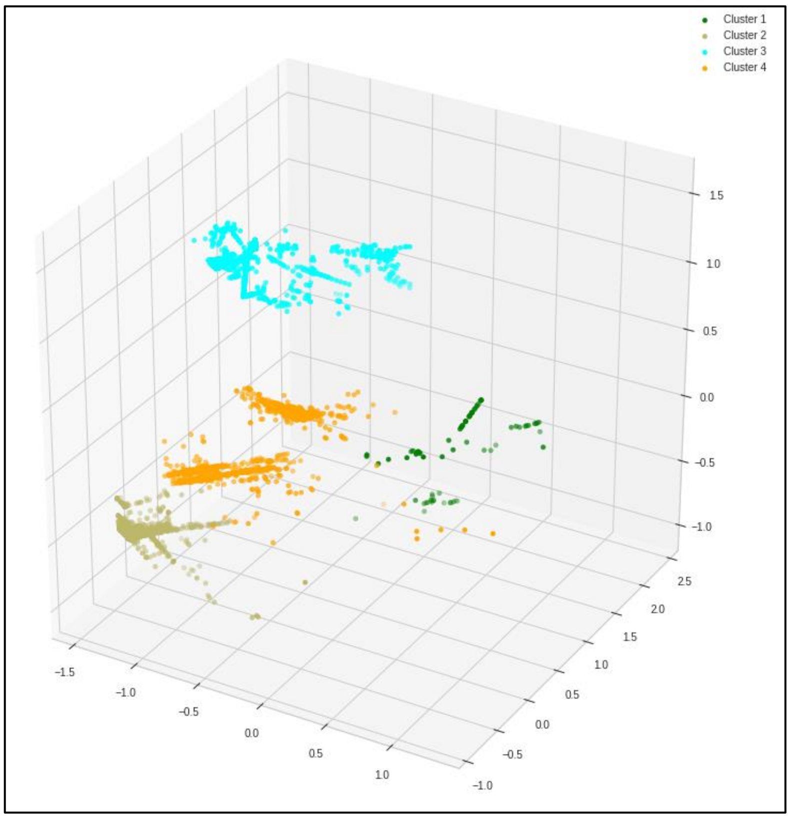


Fig 3.14 K-Means Clustering on normal data points of testing data

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:

$$z = \frac{x-\mu}{\sigma} \tag{3.4}$$

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:

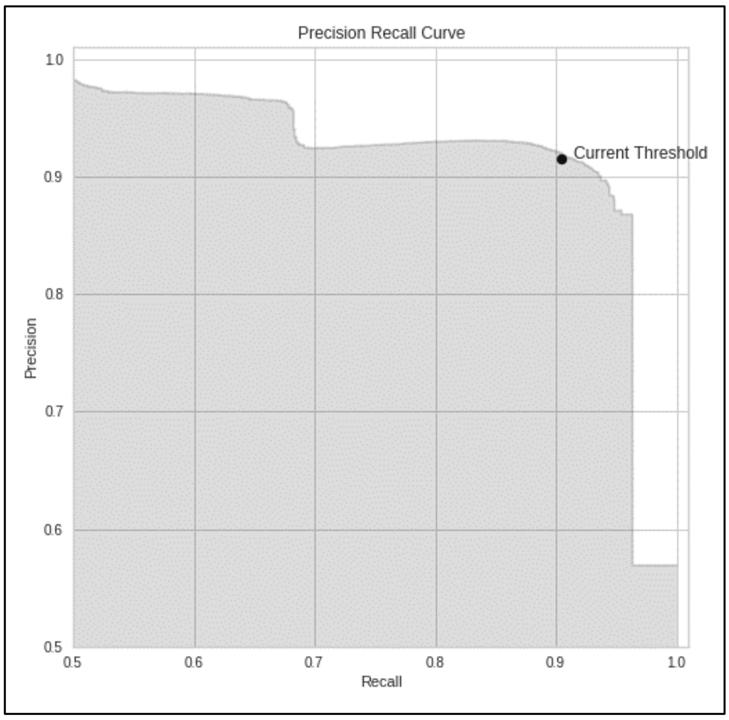


Fig 3.15 Precision recall curve for Random Forest

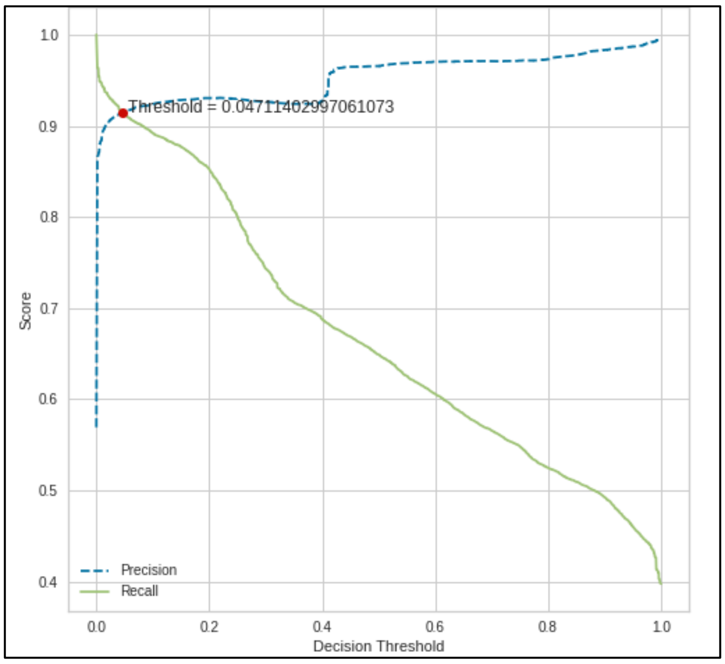


Fig 3.16 Precision and recall scores as a function of the decision threshold for Random Forest

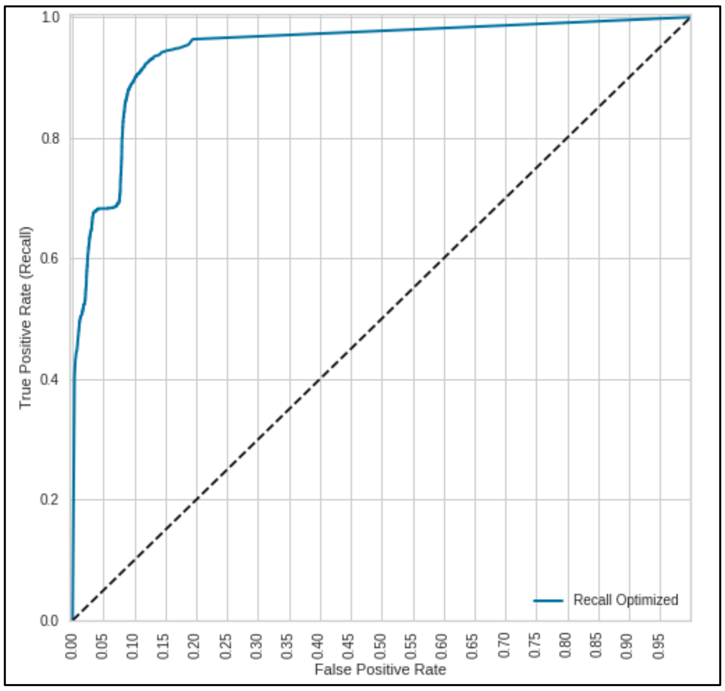


Fig 3.17 ROC Curve for Random Forest

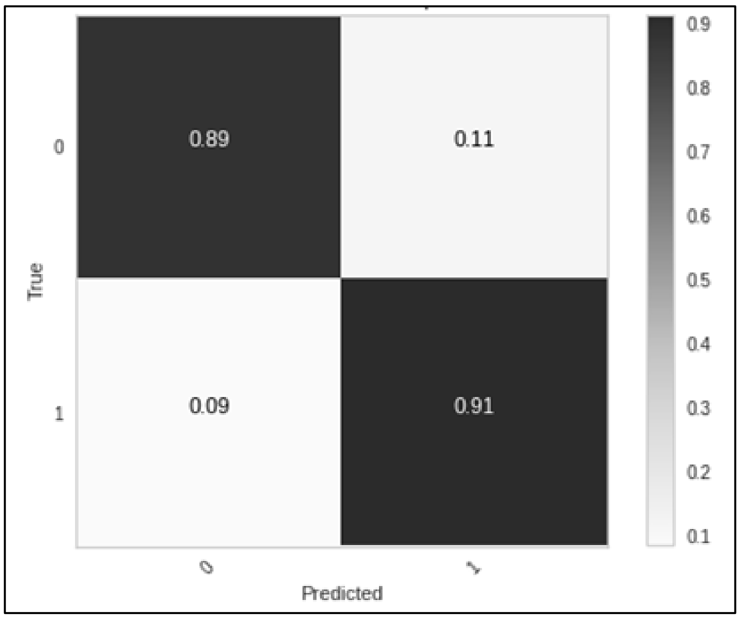


Fig 3.18 Random Forest classifier with optimized recall

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:

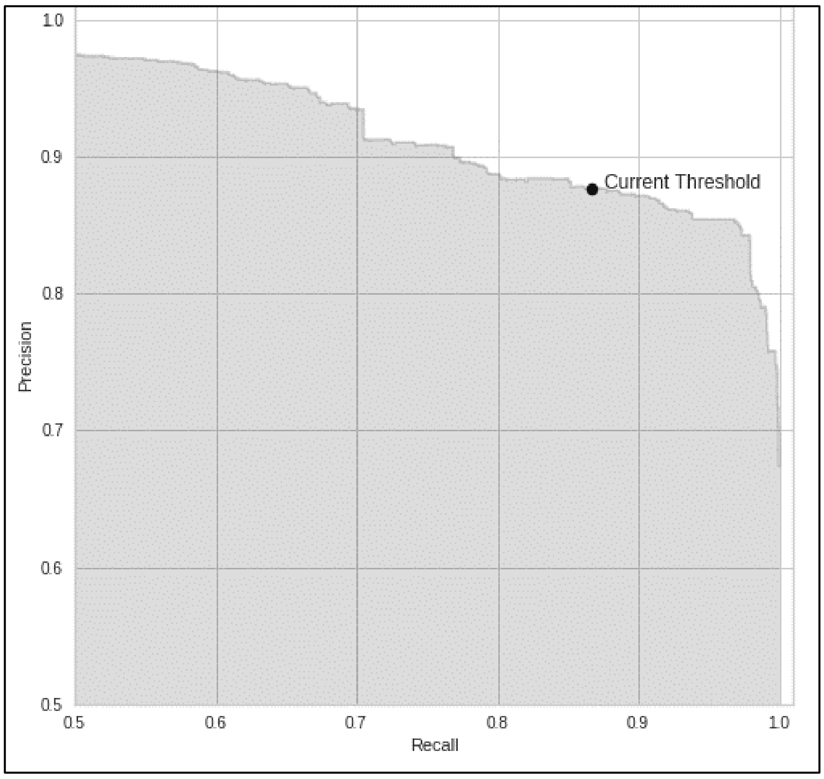


Fig 3.19 Precision recall curve for AdaBoost

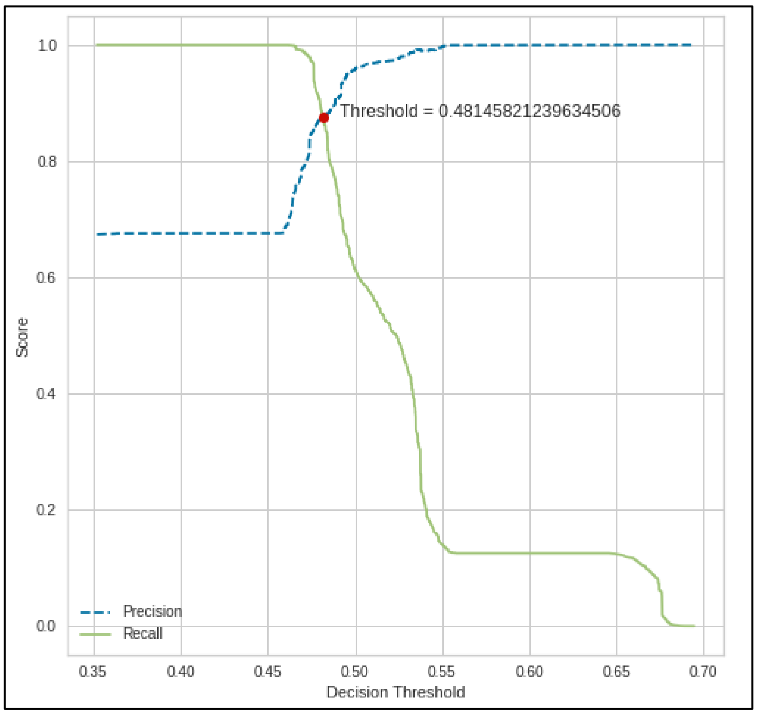


Fig 3.20 Precision and recall scores as a function of the decision threshold for AdaBoost

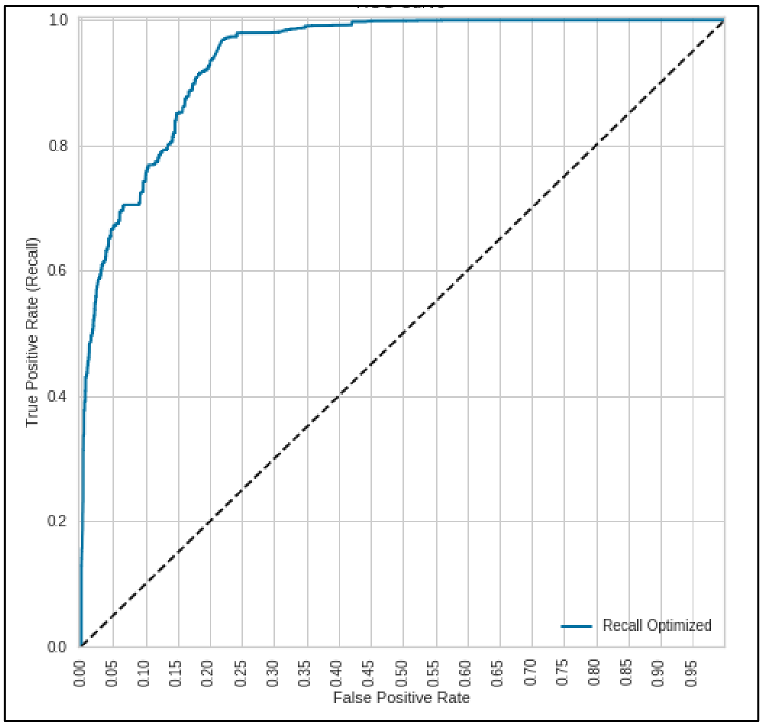


Fig 3.21 ROC Curve for AdaBoost

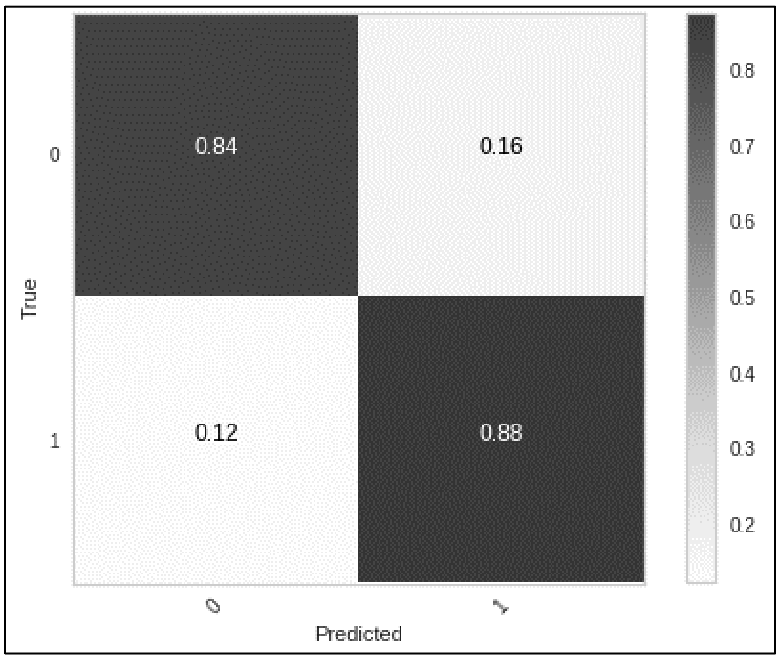


Fig 3.22 AdaBoost Classifier with optimized recall

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:

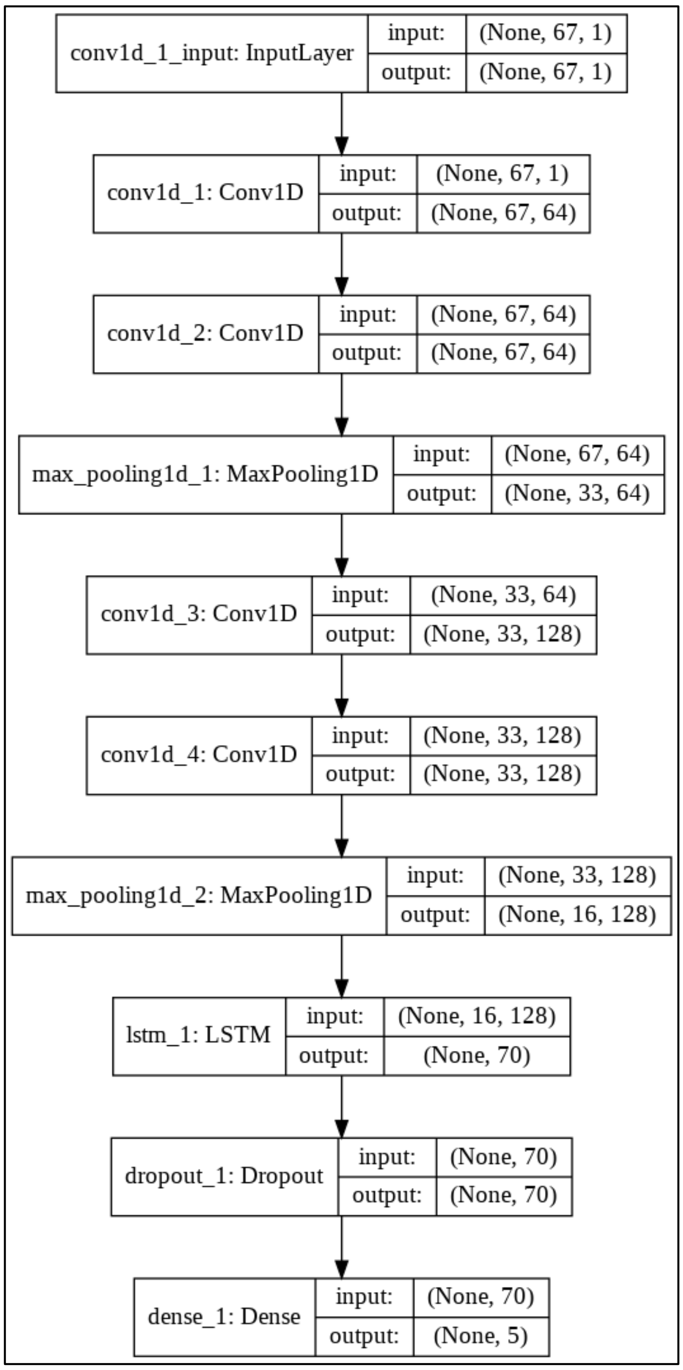


Fig 3.23 Architecture of the network

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

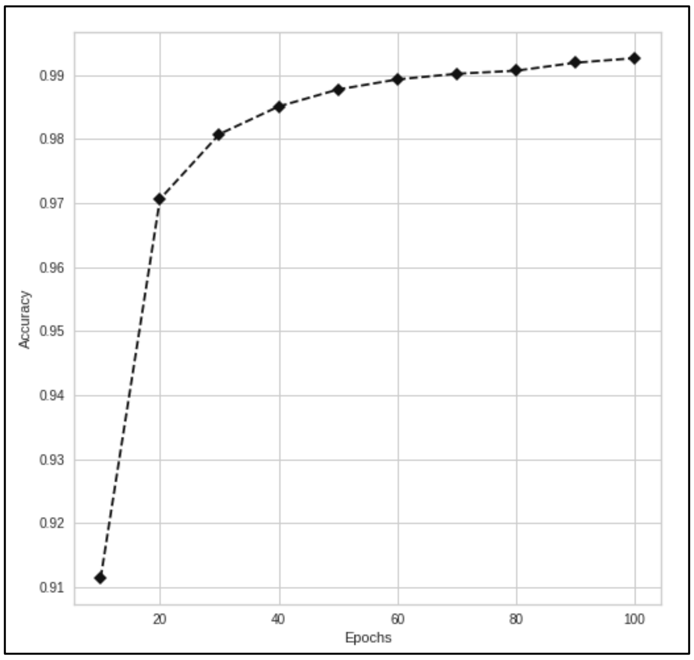


Fig 3.33 Graph representing accuracy vs number of epochs

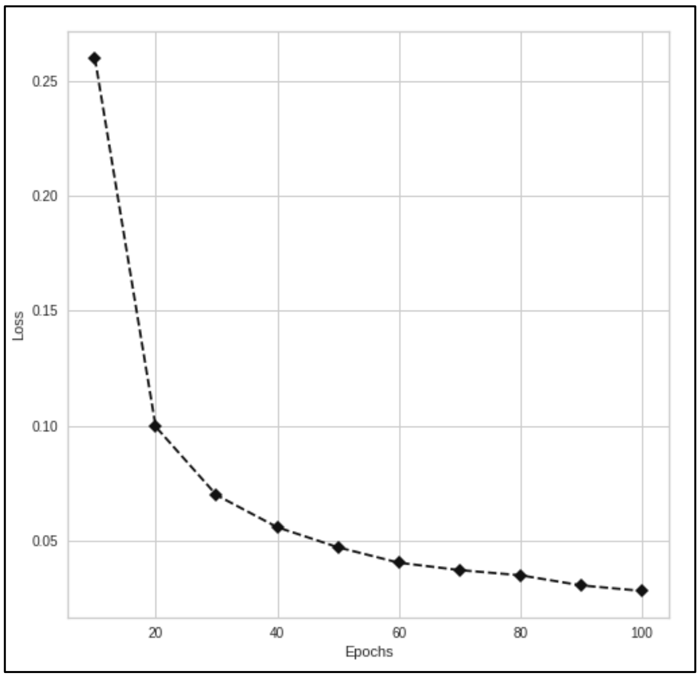


Fig 3.34 Graph representing loss vs number of epochs

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.

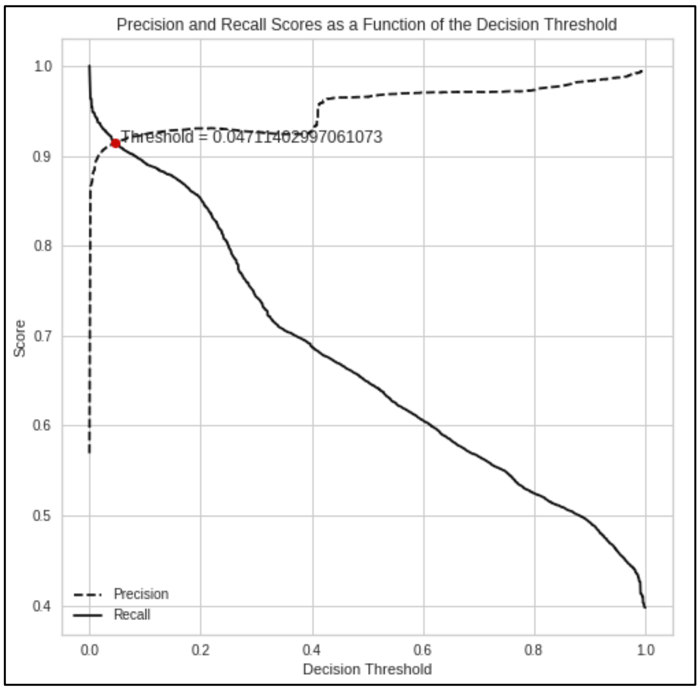


Fig 4.1 Precision and recall scores as a function of the decision threshold for Random Forest

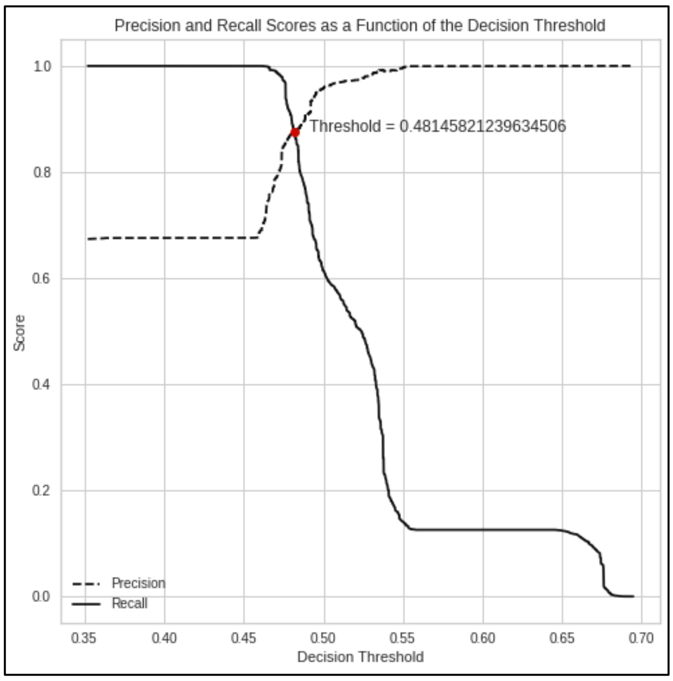


Fig 4.2 Precision and recall scores as a function of the decision threshold for AdaBoost

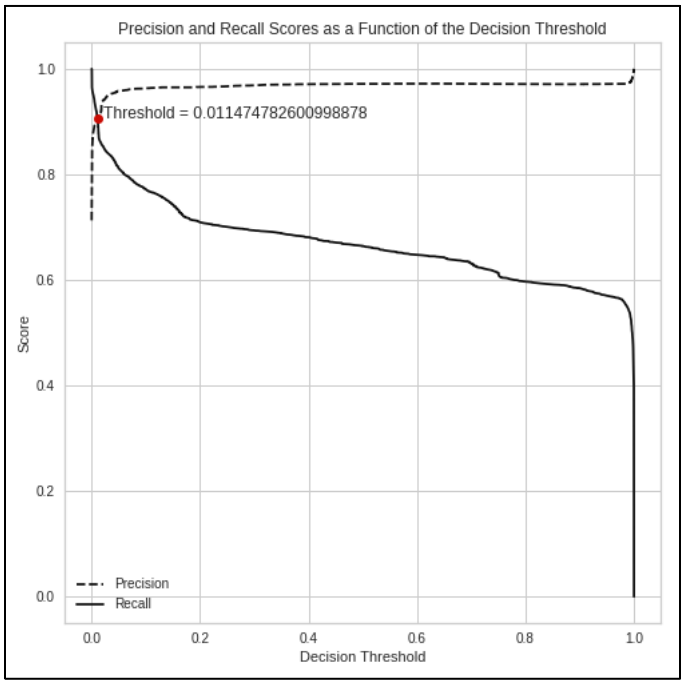


Fig 4.3 Precision and recall scores as a function of the decision threshold for XGBoost

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%.

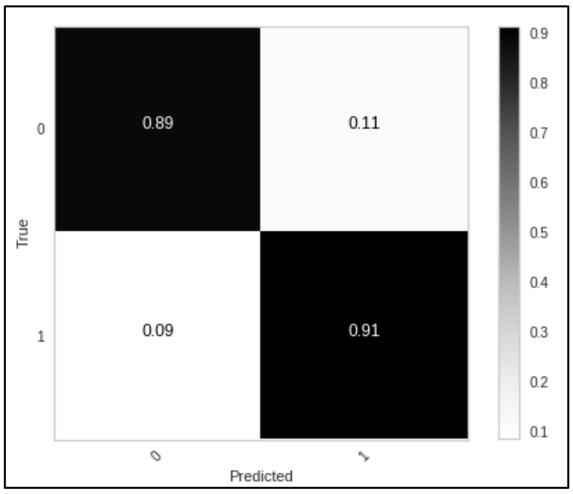


Fig 4.4 Confusion matrix for Random Forest

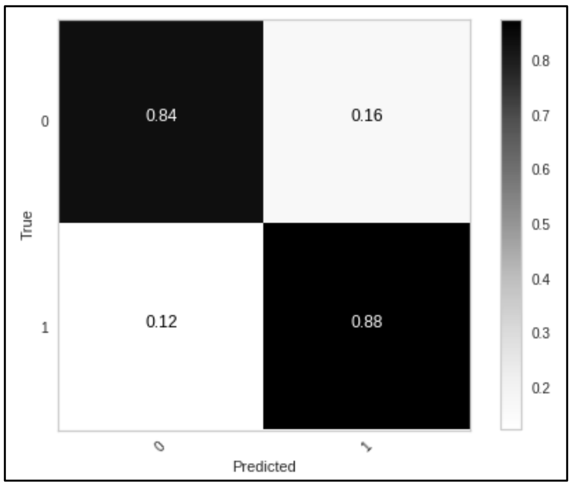


Fig 4.5 Confusion matrix for AdaBoost

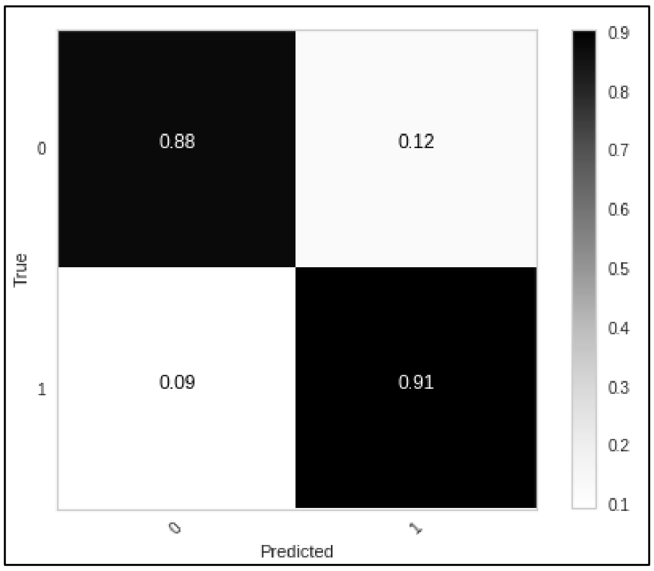


Fig 4.6 Confusion matric for XGBoost

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.

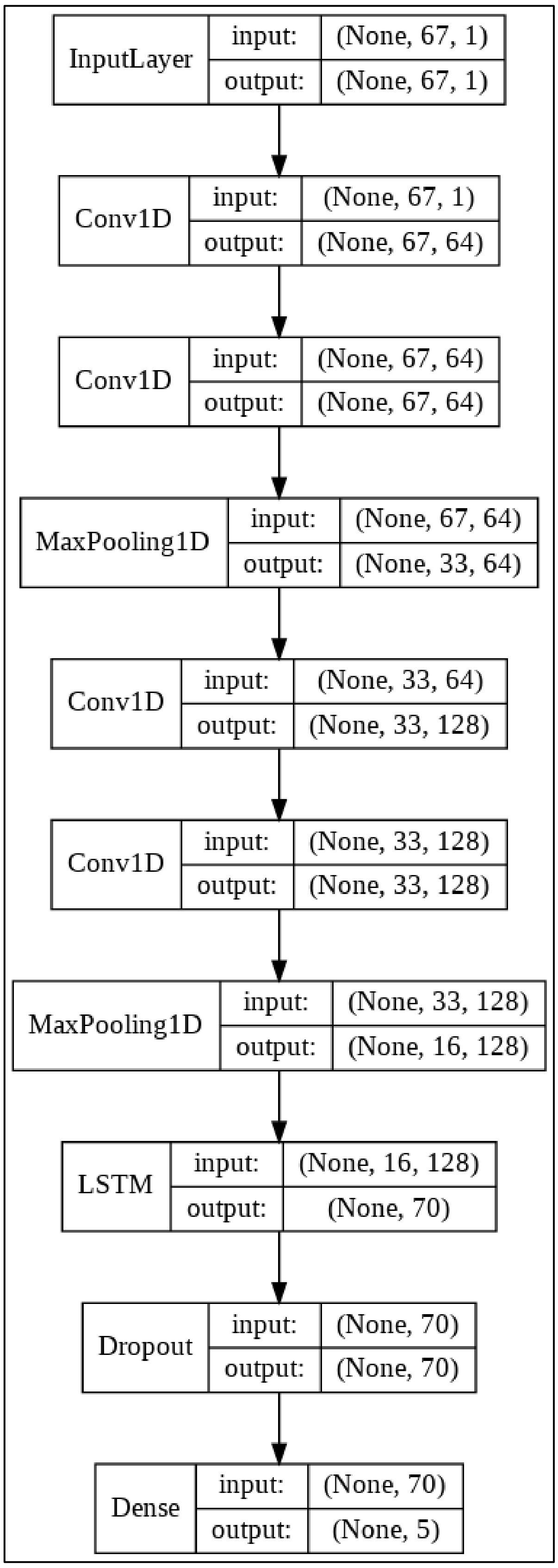


Fig 4.7 Architecture of the network

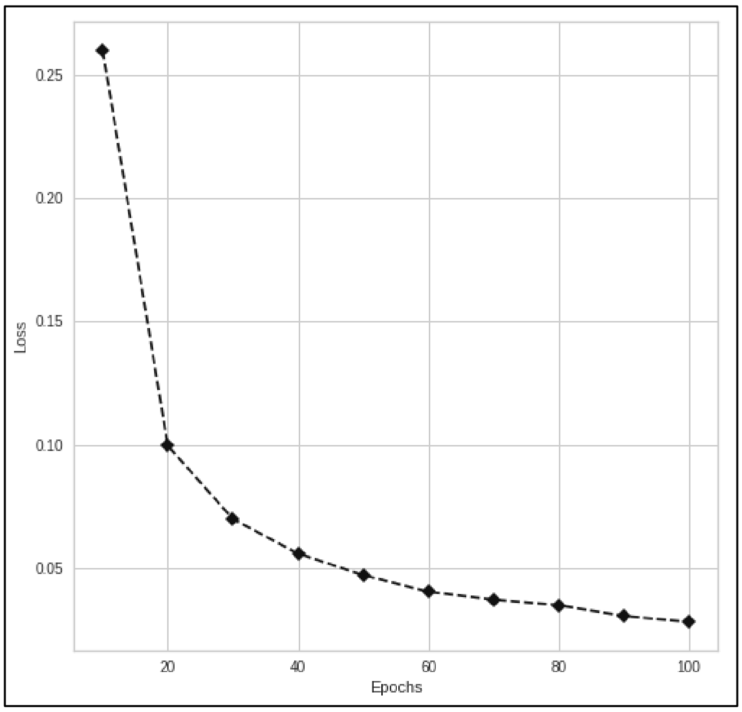


Fig 4.8 Graph representing loss vs number of epochs

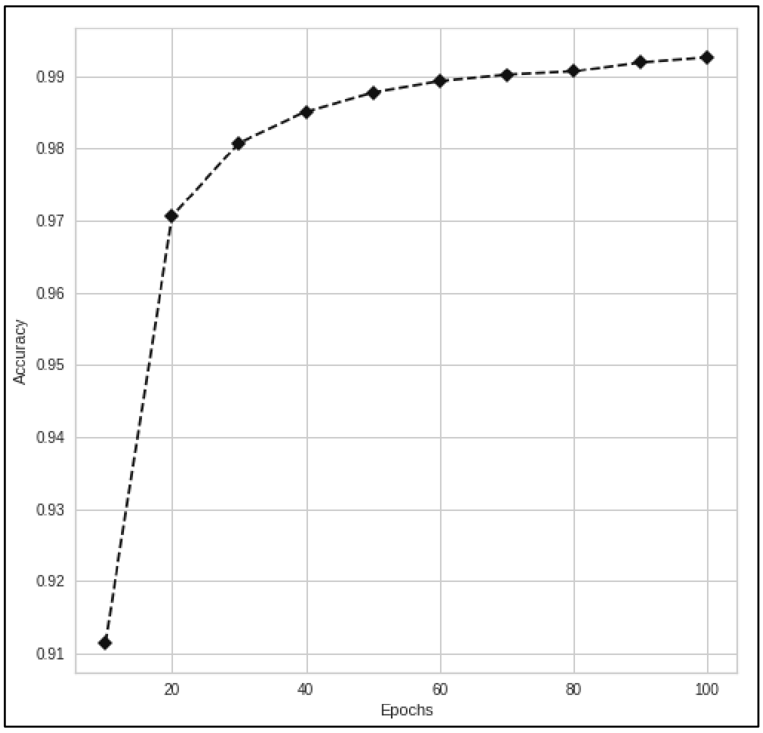


Fig 4.9 Graph representing accuracy vs number of epochs

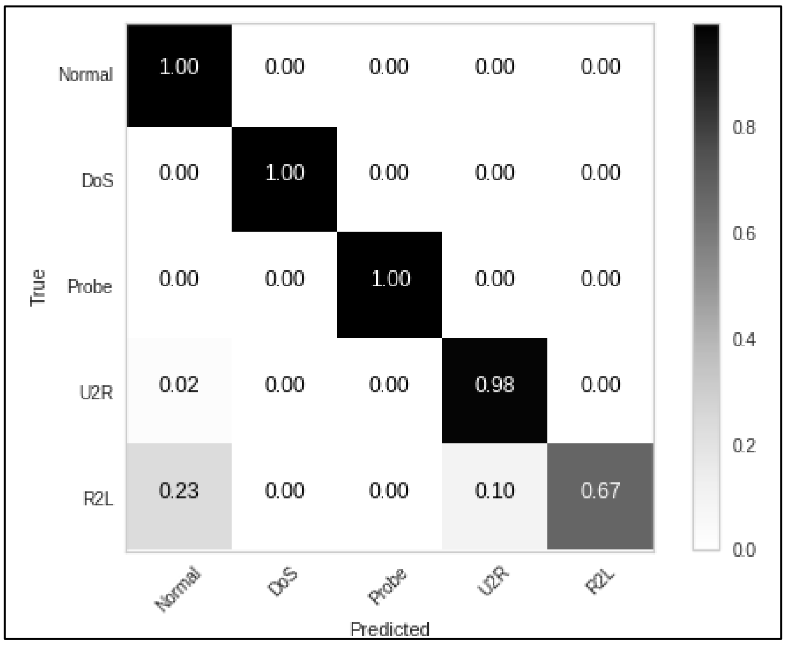


Fig 4.10 Confusion matrix of the different classes of attacks

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).

APPENDIX 1

Copy of the paper submitted for the 4th International Conference on Mathematical Models and Computational Techniques in Science and Engineering (MMCTSE)

Dual Stage Network Intrusion Detection System Through Feature Reduction

Raghuvansh Raj, Somya Gupta, Manan Lohia and H C Taneja

Department of Applied Mathematics, Delhi Technological University, Delhi, India, 110042

E-mail: somya_bt2k16@dtu.ac.in

Abstract: With an exponential increase in the amount of data produced, transmitted, stored and exchanged over the internet, intrusion detection systems have formed an integral part of modern-day network security systems. Considerable expense, time and efforts are spent in ensuring timely detection and denial of malicious users in order to preserve the key objectives of system security; confidentiality, integrity, and availability. In this paper, we intend to propose a dual stage algorithm to tackle the problem of NIDS. Our aim in this paper is to construct an algorithm that results in few false positives and fewer false negatives, as any IDS should be.

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. In the early 2010s, several papers pertaining to NIDS were published, and a majority of the technological breakthroughs in this field were influenced by the theory of deep learning.

We have focused on building an IDS using the benchmarked NSL-KDD dataset. The dataset consists of 41 features, of which 3 are categorical and the remaining, numeric. Having opted for One-Hot Encoding of the categorical features, our feature space explodes to a sum of 122 features. Despite the obvious drawbacks, this was necessary as the categorical features do not contain any implicit ordering within their values. The aim 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. Introduction

In the current scenario, the Internet has become a necessity for daily life and is used for various purposes such as education, business, banking, social networking etc. The Internet is extremely vast and diversified, connecting millions of devices with each other to create and save data but simultaneously making users vulnerable to attacks on sensitive information. Exposure of one system could lead to other computers on the network becoming compromised. With the advancement of the Internet, the requirement for network security has also evolved. This field is extremely versatile and has grown in complexity to prevent the various attacks that have been created over time and put into action. Intrusion detection systems (IDS) monitor networks for malicious activity or privacy breaches to protect data confidentiality and integrity, i.e. any form of intrusion that may negatively affect privacy or safety. Depending upon the deployed location, IDS are majorly of two types; host-based and network-based. Host-based IDS are installed on PCs and network-based IDS are situated on networks.

These systems employ machine learning techniques to differentiate the malicious usage patterns from the benign ones. In these techniques, a model is trained on a training set consisting of data points of both classes then applied on the test set to compare the predicted classes to the actual ones then tuned upon to improve the accuracy and complexity. Once the desired output is achieved, the trained model is employed in real-life scenarios to prevent malicious data packets from going through the network. To keep the model up-to-date, it is trained on newer samples to protect against the ever-evolving attacks.

This paper employs the NSL-KDD dataset which is a balanced resampling of the KDD CUP 99 dataset [1]. This dataset contains 5 classes with 4 of them malicious attacks and 1 benign. The malicious classes are DoS, Probe, R2L and U2R.

Most algorithms in effect make use of rule-based methods such as SNORT [2], clustering methods such as fuzzy clustering [3], [4] and mini batch k-means [5], and neural networks [6], [7], [8]. The dataset has high variance and no clear decision boundary between the different classes, making the classification process difficult, requiring a denser/complex model network.

In this paper, we set out to propose and implement a new method which first consists of pre-processing via feature engineering using one hot encoding followed by dimensionality reduction using Principal Component Analysis (PCA). Outlier detection is implemented as a binary classification problem with the two classes being malicious and benign to classify the malicious attacks with less complexity. The last stage consists of multi-class classification with all 5 classes in an effort to further classify the attack and reduce the rate of false positives.

2. Related Work

Due to the increasing demand of computationally inexpensive and extremely accurate network intrusion detection systems, many algorithms have been proposed, with many of them yielding high accuracies. Since the proposal of deep learning as a method for intrusion detection [9] by Professor Hinton in 2006, there have been huge advancements in every field of computational intelligence, as has been the case with intrusion detection systems.

[3], [4], make use of fuzzy clustering augmented with Artificial Neural Networks (ANN) which causes a drastic reduction in false positives. The technique used by [4] results in extremely few false positives due to the presence of an aggregating ANN after the classification ANNs.

[6] uses autoencoders to reduce the feature space and executes instructions on parallel processors. This paper focuses more on the complexity of the algorithm and the time taken by the algorithm, which is why feature space reduction is done on a rather drastic scale. Experiments show that such drastic reduction in feature space result in stark information loss which results in a less than ideal situation.

[7], [10] make use of Long Short-Term Memory (LSTM) and Recurrent Neural Network (RNN) to introduce context among the features. These kinds of neural networks are implemented by these papers in a bidirectional sense and since context is introduced in the first layer itself, the depth of the network can be limited to 2-3. Although forward propagation in this kind of network takes a substantial amount of time due to bidirectional connections in

each layer, it causes minimal space overhead due to a radical reduction in the number of nodes and layers.

[11] uses genetic algorithms like crossover and mutation to search for the optimal hyperparameter space in neural networks. The idea of genetic algorithms is inspired by evolution and how it results in the fittest population (hyperparameters in this case). Genetic algorithms simply need a fitness measure that results in the optimal feature space when run for multiple epochs but the drawback with this method lies in the complexity and time taken by the model as it is essentially running over the entire dataset multiple times.

In this paper we have attempted to create a model that runs on a reduced feature space and works as a robust classifier while providing minimum false negatives. For this, we use a two-step approach. First, anomaly detection is conducted to segregate malicious and benign data points. Then, a multi class classification model classifies data points into respective classes and reduces the number of false positives.

3. Preliminaries

In this section, a basic overview is provided of all relevant terms such as IDS, dimensionality reduction, decision trees, ensemble methods and the NSL-KDD dataset.

3.1 Intrusion Detection Systems

Intrusion Detection Systems are classified into two major categories based on their approach and the techniques and algorithms used to analyze traffic data and detect intrusions, namely anomaly-based and signature-based.

Signature based IDS compare live traffic data to a stored database and try to match it with known intrusion patterns in order to detect attacks, i.e., they follow the well-known paradigm of rule-based systems. Signature-based IDS deliver high accuracy when detecting known intrusion patterns, but can fail to detect newer types of attacks due to a lack of any existing, match able patterns in the database. In an era where attackers are always finding newer ways to take control of a system or a network of systems, signature-based IDS fail to offer an adequate level of protection without a substantial degree of risk. Another drawback of such frameworks is the large administrative overhead required to regularly maintain and update the signature database to ensure that the system is able to detect even the newest types of attacks [12].

Contrarily, anomaly-based IDS rely on detecting deviations from what is considered to be normal network behavior to identify malicious activity. Originally, anomaly-based systems used statistical methods to model data representing ‘normal behavior’ and used these developed models to identify and flag any significant deviations as suspicious. However, recent efforts have pivoted to focus on using machine learning and deep learning along with advanced mathematical techniques to develop improved anomaly-based IDS. Major challenges that arise when attempting to build such models is the high false alarm rates, owing to the difficulty of modeling the ‘normal behavior’ due to non-linearity of data and a certain degree of obscurity present in the data [13].

3.2 Dimensionality Reduction

Dimensionality reduction is an umbrella term for the various processes aimed at reducing the number of random variables in a dataset. This serves multifarious purposes such as enabling visualization of data and easing computational requirements. It can also help remove contextual redundancies in the dataset and simplify the overall process of data analysis and model construction. Dimensionality reduction is split majorly into two kinds of techniques, feature selection and feature extraction.

Feature selection involves the selection of a subset of the existing features. The selection of features can be guided by strategies involving information gain, gain ratio, scoring features based on their importance etc.

Feature extraction is a more complex set of techniques that involve combining the existing variables through various statistical, algebraic or deep learning techniques to form a new, lower dimensional set of features which can be used to represent the dataset effectively. However, this technique suffers with a loss of context as it is not always possible to determine what exactly the new features represent and some information is always lost due to the decrease in number of features.

3.2.1 Principal Component Analysis

Principal Component Analysis (PCA) [5], [14] is a kind of dimensionality reduction technique that combines concepts from statistics and linear algebra to perform feature extraction. The process involves the use of covariance matrix and eigenvalues to transform the existing dataset into a new dataset using principal components derived from eigenvectors. The following steps are involved in PCA:

- For the dataset $D_{m \times n}$ containing *m* datapoints with *n* features, subtract from each element the mean of the column to which it belongs to center the datapoints at the origin. Let this new matrix be D`.
- Calculate the covariance matrix od D`, denoted by C.
- Obtain the eigenvalues and eigenvectors of the matrix C.
- If *d* denotes the required number of dimensions for the new dataset, select the first *d* eigenvalues with the largest magnitudes, and combine their corresponding eigenvectors to form the transformation matrix $E_{n \times d}$.
- The reduced dataset is given by $F_{m \times d} = [\ D`_{m\,x\,n}\ ] \times [\ E_{n\,x\,d}\ ]$.

3.3 Decision Trees

A decision tree is a classification algorithm that forms a flowchart in a tree-like graph where at each internal node an attribute is chosen on which to split the dataset on. Consequently, each branch represents one outcome of the split and any subsequent leaf nodes constitute the classes. The entire path from the root node to a leaf node can be regarded as a classification rule [15]. They map both linear as well as non-linear relationships.

The split at each internal node can be done on the basis of information gain and Gini index. The model proposed in this paper uses Gini index. Gini index forces a binary tree which is ideal in our case as we are initially working on a binary classification model. The underlying principle for this splitting algorithm is based on the fact that the probability of two randomly chosen data tuples from the dataset belonging to the same class is 1 if the dataset is pure.

3.4 Ensemble Methods

The usage of ensemble methods is ideal in the combination of a number of base models to produce an optimal model. These methods take into consideration the outputs from a number of weak classifiers and use aggregated outputs using simultaneous or sequential voting procedures to form the final output.

Bagging [16] is an example of an ensemble method that combines Bootstrapping and Aggregation. From a sample of the dataset, multiple bootstrapped samples are taken upon which individual decision trees are trained. An algorithm is then used to aggregate the trees and find the most efficient predictor.

Random Forests [17] are another example. These use the same concept as bagging except that they work with the entire dataset rather than a sample. Each tree in this method splits at different features thus overall providing a larger ensemble to aggregate over, eventually producing a more accurate predictor.

3.5 NSL-KDD

The NSL-KDD dataset [18] is a refined version of the original KDD-CUP 99 dataset, which, after a thorough analysis, was revealed to contain a number of redundancies and other issues such as imbalanced classes and a huge number of records which offered many challenges to the construction of a classification model. The improved NSL-KDD dataset consists of 41 features and is split into the test set and training set. The training set has 125,973 datapoints while the test set has 22,544 datapoints. The test data is classified into 38 different classes, of which 21 are present in the training data. One of these classes corresponds to 'normal activity' while the others are various attack types. For the sake of brevity, the 37 other classes are categorized into 4 different attack types, as follows [19]:

Table 1. Categorization of the 37 different classes into the 4 different attack types.

S. No	Attack Type	Class in Dataset
1	DoS	back, land, neptune, pod, smurf, teardrop, mailbomb, processtable, udpstorm, apache2, worm
2	Probe	satan, ipsweep, nmap, portsweep, mscan, saint
3	R2L	guess_password, ftp_write, imap, phf, multihop, warezmaster, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named
4	U2R	buffer_overflow, loadmodule, rootkit, perl, sqlattack, xterm, ps

We follow the aforementioned classification in our model. The following figures show the distribution of datapoints belonging to each class. As the distribution is highly skewed, we have used the log of the counts for each class where necessary.

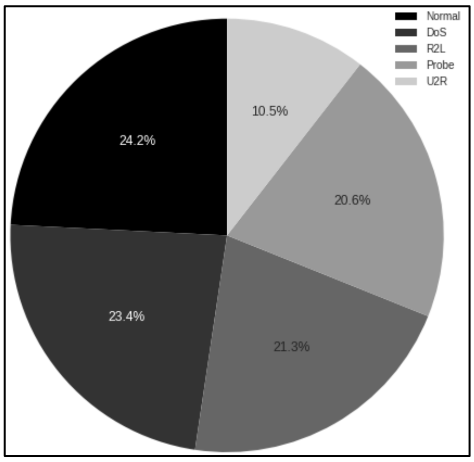


Figure 1. Class distribution of the test set

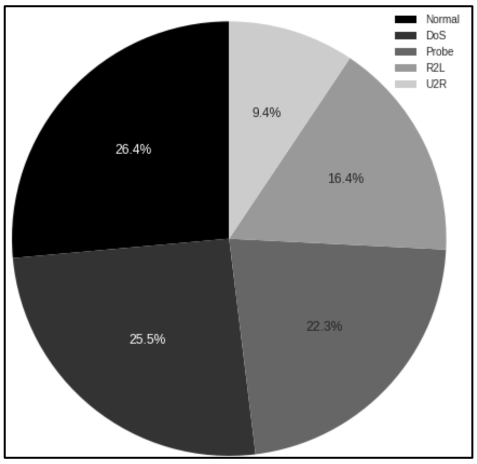


Figure 2. Class distribution of the training set.

The 41 features can be divided into 4 major sets, consisting of the basic features for any network connection, content related features, time related traffic features and host-based traffic features [20]. A description of features is given in [20]. The remaining features are majorly binary variables which provide additional information about the network connection represented by the corresponding datapoints.

4. Proposed Methodology

In this section, we provide the proposed methodology for intrusion detection. The proposed schematic is shown in figure 3.

The proposed architecture consists of 4 steps; each of these are explained in detail further.

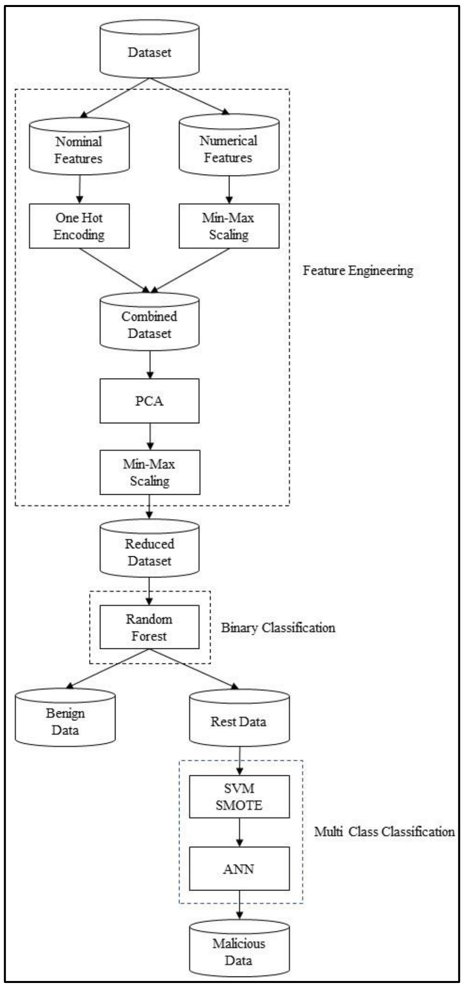


Figure 3. Flow of the model

4.1 Feature Engineering

The initial phase of the model is the feature engineering step. The NSL-KDD dataset consists of a training set and a testing set. The training set has 41 features, out of which the 20th feature, i.e., num_outbound_cmds which is the number of outbound commands in an ftp session [20], consists of only zeros so the column is dropped. After dropping the 20th column, we divide the dataset into nominal and numerical features depending on the data type. The nominal features are one-hot encoded which expands the number of features to 84. The numerical features on the other hand undergo min-max scaling so that features with a broader range do not outweigh the features with a smaller range. There are a total of 37 such features. The combined dataset of both numerical and nominal features gives a total of 121 features. The feature matrix formed is sparse due to one-hot encoding so dimensionality reduction is carried out using PCA on the 121 features. Retention of 67 features retains 99% variance of the original dataset. Then min-max scaling once again to bring all the features within the same range.

4.2 Binary Classification

In this step, binary classification is done to divide the dataset into two classes namely, malicious and benign. Ensemble methods have been tried and tested to see which one works best. Random Forest, AdaBoost and XGBoost have been compared in terms of precision, recall and area under the receiver operating characteristics (ROC) curve. The precision-recall curves have been plotted to calculate the threshold for the confidence score. The optimal threshold value is the intersection of the precision and recall curves in order to optimize both metrics. The aim of binary classification was to increase recall so as to reduce false negatives.

After applying binary classification, we have divided the dataset into two categories; one with completely benign data points and another with both benign and malicious data points as the number of false positives are still relatively high.

4.3 Oversampling

Before we reach the step where we reduce the number of false positives, we oversample the dataset as the number of data points of U2R and R2L are extremely less in comparison to the rest, i.e., Normal, DoS and Probe. This could potentially lead to the problem of the network not being able to effectively categorize datapoints of those 2 classes. Since the samples of U2R and R2L are few, the network is not able to classify points into these 2 classes properly resulting in false negatives. To solve this problem, we used SVMSMOTE which oversampled for the 2 classes and synthesized new data points.

4.4 Multi Class Classification

The last step is multi class classification. This is done to reduce the number of false positives occurring at the output of the model. An artificial neural network (ANN) is fitted on the output classes of binary classification that had both malicious and benign data points after oversampling.

The datapoints are classified into the 5 classes, namely Normal, DoS, Probe, U2R and R2L after being fed through an ANN. The ANN architecture that we have implemented consists of Convolutional and LSTM layers and outputs the class probabilities for all the 5 classes using SoftMax Activation at the final layer and Rectified Linear Unit (ReLU) activations at every other layer. Datapoints classified as normal here as well as at the binary classification stage form the combined population of benign data points. Every other data point is categorized first as malicious and then further into the type of attack.

5. Experiment and Results

In this research, we have used scikit-learn and keras with a tensorflow backend for the construction of our models and have used matplotlib for generating graphs. All of the models have been trained on Google Colaboratory using a Tesla K80 single core GPU with 25.51 GB RAM, although RAM usage throughout our experiments was limited to 10 GB.

During feature engineering, we applied PCA for dimensionality reduction. The following graph shows the retention of variance corresponding to the number of components retained.

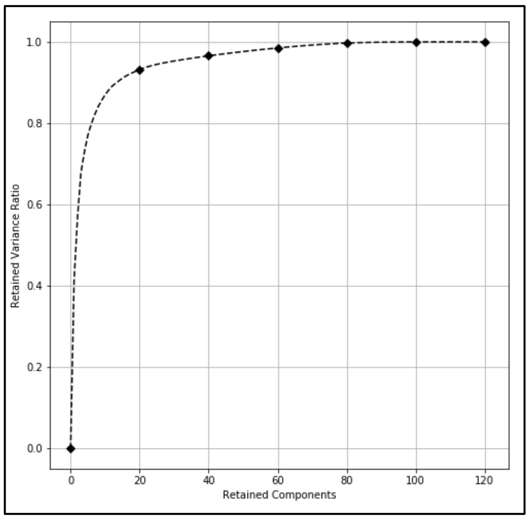
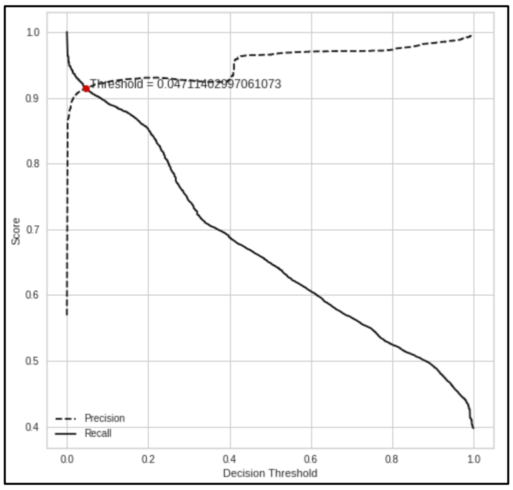
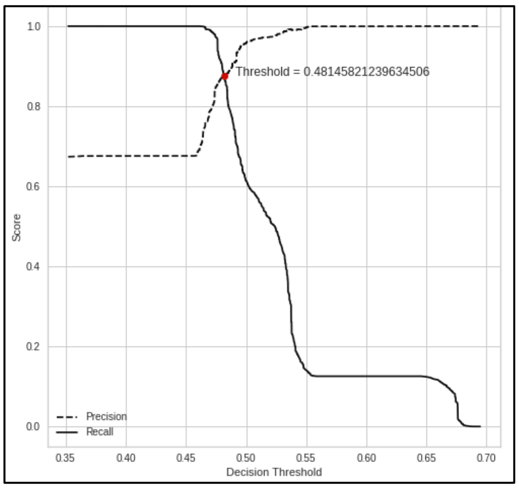


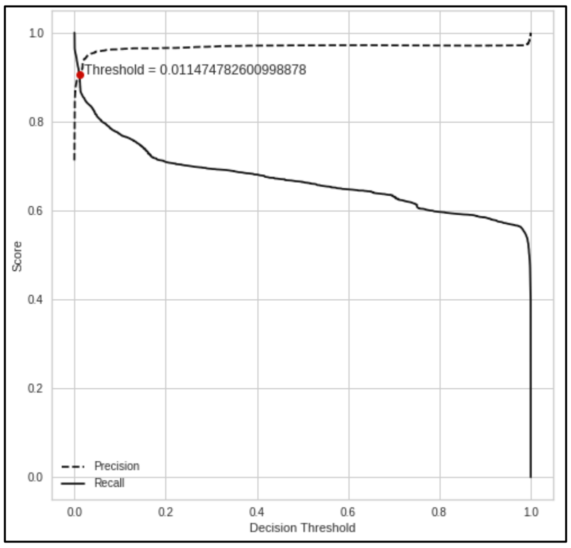
Figure 4. Variance Retention vs. Number of Components.

For binary classification we have compared 3 methods: Random Forest classifiers and boosting algorithms AdaBoost and XGBoost. To obtain the optimal set of parameters for each of these classifiers, grid search was run on a suitable search space with recall as the optimizing metric. This was followed by comparing plots (Figure 5) of precision and recall and finding the optimal threshold for a point to be classified as malicious. Threshold tuning was based on the fact that we weren't looking for very high precision values at this stage; recall was of primary importance. Experiments suggested that threshold values that brought precision and recall closest together could be considered as optimal threshold points as the relation between these 2 metrics is almost inverse in nature, which means that increasing recall via threshold tuning would result in reduced precision.





(a) **(b)**

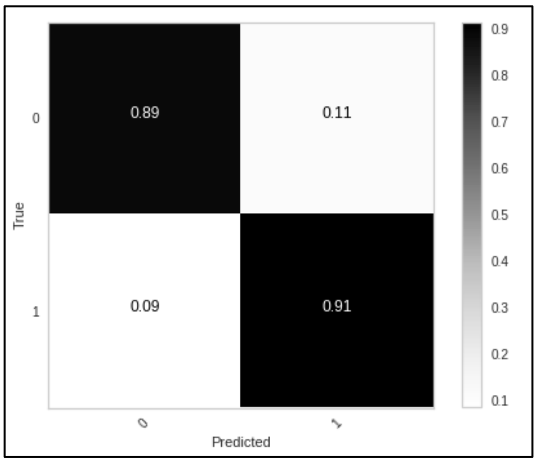


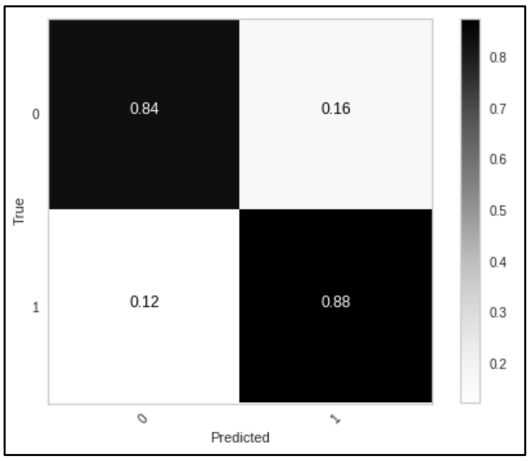
(c)

Figure 5. Precision and recall scores as a function of the decision threshold of (a) Random Forest, (b) AdaBoost and (c) XGBoost

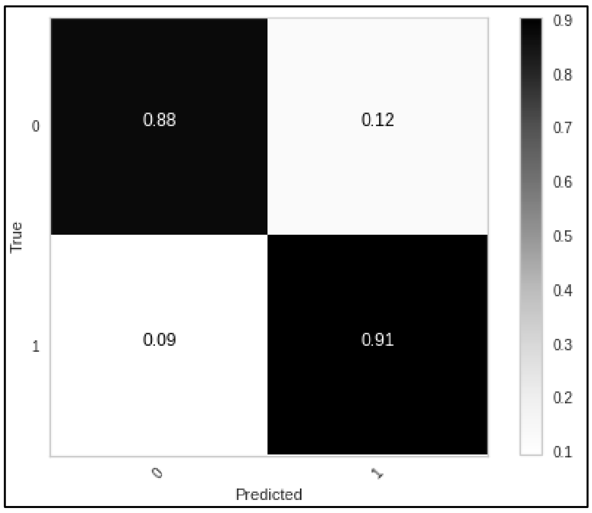
Random Forest classifiers gave the best classification output overall (Figure 6). It and XGBoost had a recall of 91%, while AdaBoost gave a recall of 88%. Random Forest Classifiers gave a maximum precision of 89% while AdaBoost and XGBoost gave a precision of 84% and 88% respectively.

XGBoost had the maximum area under curve (AUC), with a value of 0.96275, while Random Forest Classifiers and AdaBoost had an AUC of 0.94587 and 0.94139 respectively (Figure 7).



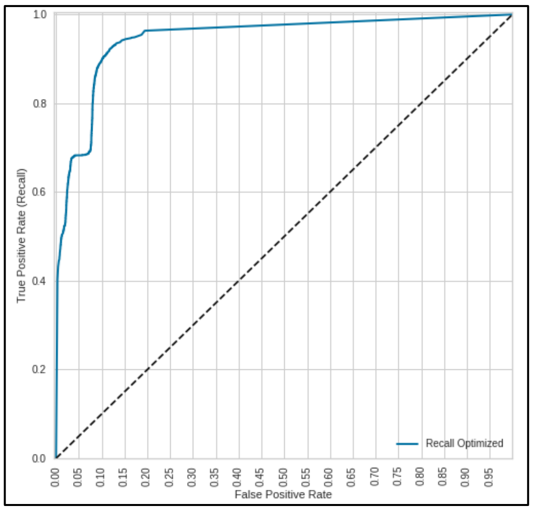


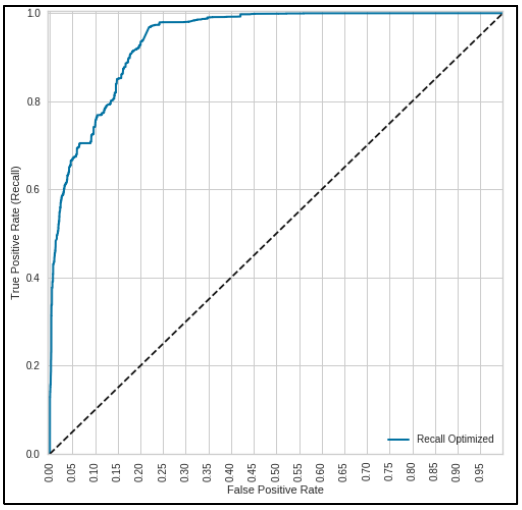
(a) **(b)**



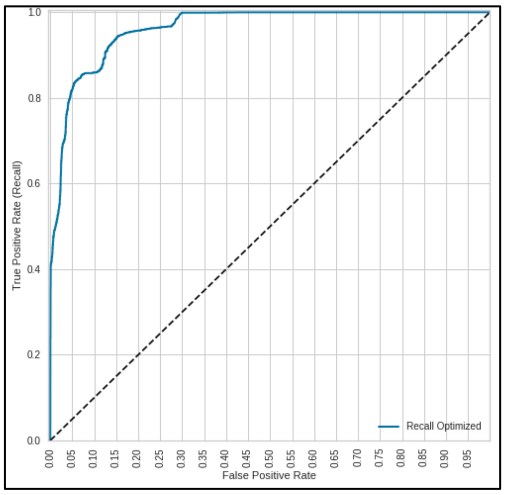
(c)

Figure 6. Confusion matrices of (a) Random Forest, (b) AdaBoost and (c) XGBoost





(a) **(b)**



(c)

Figure 7. ROC curves of (a) Random Forest, (b) AdaBoost and (c) XGBoost

For multi-class classification, multiple neural network architectures were considered comprising of CNNs, RNNs, LSTMs, etc. The following architecture, as suggested by [21] yielded best results when trained on 100 epochs.

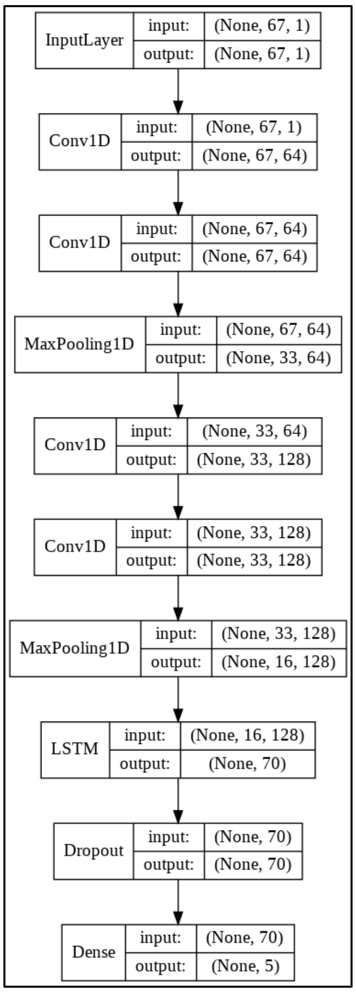

InputLayer | input: (None, 67, 1) | output: (None, 67, 1)
Conv1D | input: (None, 67, 1) | output: (None, 67, 64)
Conv1D | input: (None, 67, 64) | output: (None, 67, 64)
MaxPooling1D | input: (None, 67, 64) | output: (None, 33, 64)
Conv1D | input: (None, 33, 64) | output: (None, 33, 128)
Conv1D | input: (None, 33, 128) | output: (None, 33, 128)
MaxPooling1D | input: (None, 33, 128) | output: (None, 16, 128)
LSTM | input: (None, 16, 128) | output: (None, 70)
Dropout | input: (None, 70) | output: (None, 70)
Dense | input: (None, 70) | output: (None, 5)

Figure 8. Architecture of the ANN

The loss convergence and accuracy over a course of 100 epochs are shown in the following graph.

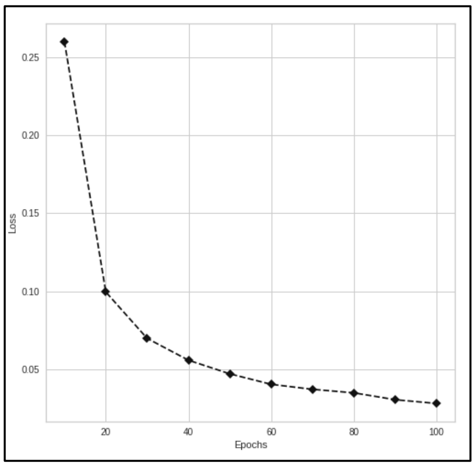


Figure 9. Loss convergence of the model.

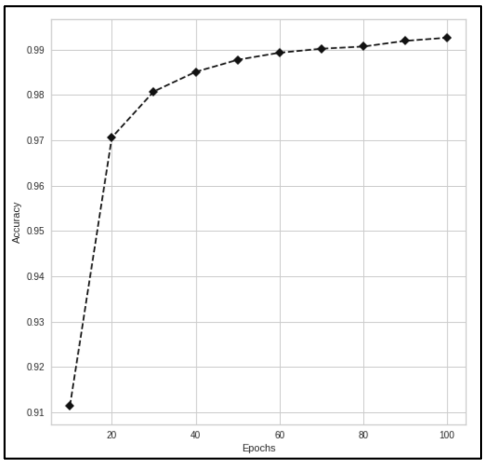


Figure 10. Accuracy convergence of the model.

The final results obtained by our overall model, are outlined below.

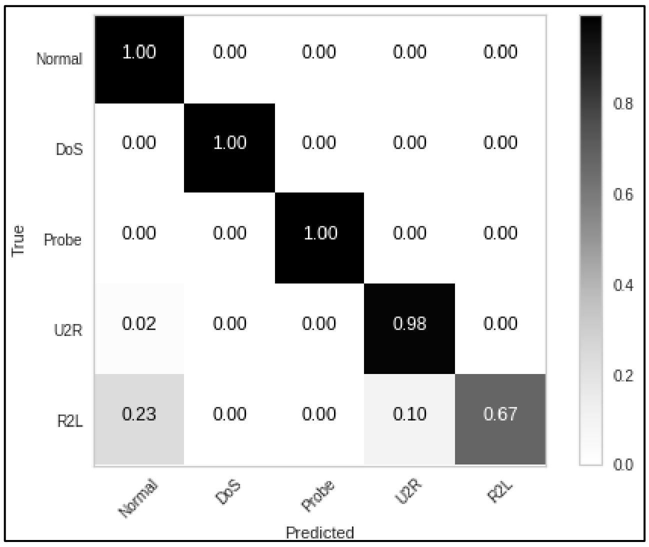


Figure 11. Confusion matrix of the multi class classifier.

6. Conclusion and Future Scope

The two-stage classification algorithm we considered here has resulted in much better evaluation results than the methods previously explored. A reduced feature space ensures forward propagation through the model to work fast so that detection overhead is minimized.

Future scopes for this architecture include using GANs for generating adversarial samples that can compensate for the lesser proportion of U2R and R2L samples and generate a more global feature space that would result in better model parameters and would detect a larger proportion of attacks. A number of other oversampling techniques can be explored and their accuracies studied.

7. References

[1] A. Divekar, M. Parekh, V. Savla, R. Mishra and M. Shirole, "Benchmarking datasets for anomaly-based intrusion detection: KDD cup 99 alternatives," in *Proceedings on 2018 IEEE 3rd International Conference on Computing, Communication and Security, ICCCS 2018,* 2018.

[2] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of LISA '99: 13th Systems Administration Conference,* Seattle, Washington, USA, 1999.

[3] G. Wang, J. Hao, J. Ma and L. Huang, "A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering," *Expert Systems with Applications,* vol. 37, no. 9, pp. 6225-6232, 2010.

[4] S. Shraddha, "Intrusion detection using fuzzy clustering and artificial neural network," *Advances in Neural Networks, Fuzzy Systems and Artificial Intelligence,* pp. 209-217.

[5] K. Peng, V. C. M. Leung and Q. Huang, "Clustering approach based on mini batch kmeans for intrusion detection system over big data," *IEEE Access,* vol. 6, no. 1, pp. 11897-11906, 2018.

[6] S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced intrusion detection system," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA,* 2016.

[7] C. Yin, Y. Zhu, J. Fei and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access,* vol. 5, no. 1, pp. 21954-21961, 2017.

[8] J. Kim, N. Shin, S. Y. Jo and S. H. Kim, "Method of intrusion detection using deep neural network," in *IEEE,* 2017.

[9] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, no. 1, pp. 436-444, 2015.

[10] J. Kim, J. Kim, H. L. T. Thu and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *IEEE,* 2016.

[11] Z. Chiba, A. Noreddine, K. Moussaid, A. E. Omri and M. Rida, "Intelligent and improved self-adaptive anomaly based detection system for networks," *International Journal of Communication Networks and Information Security,* vol. 11, no. 2, pp. 312-330, 2019.

[12] P. Manandhar and Z. Aung, "Intrusion detection based on outlier detection method," in *International conference on Intelligent Systems, Data Mining and Information Technology* , Bangkok, Thailand, 2014.

[13] J. Vacca, Computer and information security handbook, Elsevier, 2009.

[14] M. E. Tipping and C. M. Bishop, "Mixtures of probabilistic principal component analysers," *Journal of the Royal Statistical Society,* pp. 443-482, 1999.

[15] K. Rai, M. S. Devi and A. Guleria, "Decision tree based algorithm for intrusion detection," *Advanced Networking and Applications,* vol. 7, no. 4, pp. 2828-2834, 2016.

[16] D. P. Gaikwad and R. C. Thool, "Intrusion detection system using bagging ensemble method of machine learning," in *2015 International Conference on Computing Communication Control and Automation*, Pune, India, 2015.

[17] J. Zhang, M. Zulkernine and A. Haque, "Random Forests based network intrusion detection systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 38, no. 5, pp. 649-659, 2008.

[18] U. o. N. Brunswick, "Datasets," [Online]. Available: http://nsl.cs.unb.ca/NSL-KDD/.

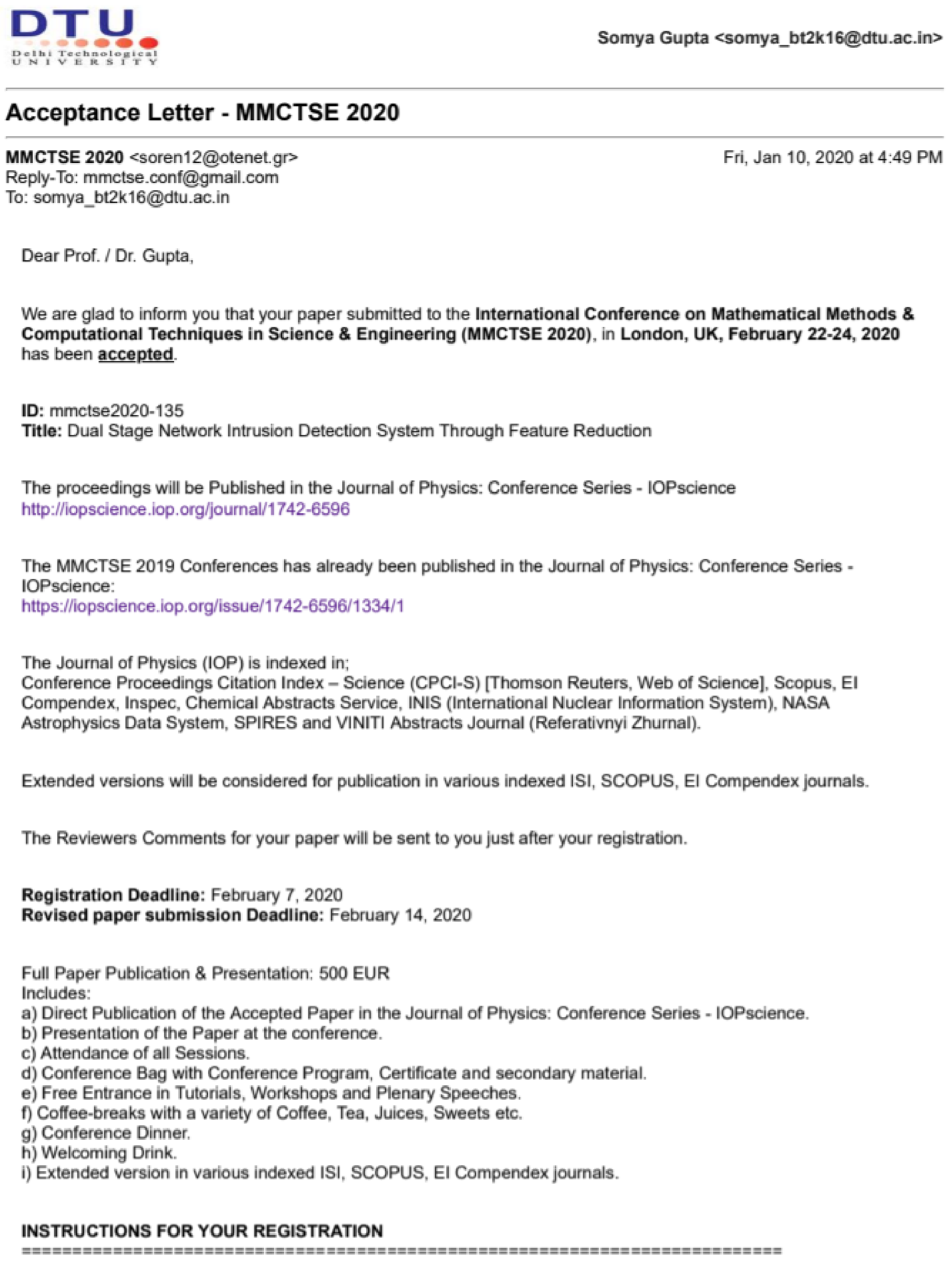
[19] S. Revathi and A. Malathi, "A detailed analysis on NSL-KDD dataset using various machine learning techniques for intrusion detection," *International Journal of Engineering Research and Technology,* vol. 2, no. 12, pp. 1848-1853, 2013.

[20] L. Dhanabal and S. S. P, "A study on NSL-KDD dataset for intrusion detection system based on classification algorithms," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 4, no. 6, pp. 446-452, 2015.

[21] R. VinayaKumar, K. P. Soman and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *IEEE,* Udupi, India, 2017.

APPENDIX 2

Copy of the acceptance letter and proof of Scopus indexing received from the 4th International Conference on Mathematical Models and Computational Techniques in Science and Engineering (MMCTSE)



DTU
Delhi Technological University

Somya Gupta <somya_bt2k16@dtu.ac.in>

Acceptance Letter - MMCTSE 2020

MMCTSE 2020 <soren12@otenet.gr>
Reply-To: mmctse.conf@gmail.com
To: somya_bt2k16@dtu.ac.in

Fri, Jan 10, 2020 at 4:49 PM

Dear Prof. / Dr. Gupta,

We are glad to inform you that your paper submitted to the **International Conference on Mathematical Methods & Computational Techniques in Science & Engineering (MMCTSE 2020)**, in **London, UK, February 22-24, 2020** has been **accepted**.

ID: mmctse2020-135
Title: Dual Stage Network Intrusion Detection System Through Feature Reduction

The proceedings will be Published in the Journal of Physics: Conference Series - IOPscience
http://iopscience.iop.org/journal/1742-6596

The MMCTSE 2019 Conferences has already been published in the Journal of Physics: Conference Series - IOPscience:
https://iopscience.iop.org/issue/1742-6596/1334/1

The Journal of Physics (IOP) is indexed in;
Conference Proceedings Citation Index – Science (CPCI-S) [Thomson Reuters, Web of Science], Scopus, EI Compendex, Inspec, Chemical Abstracts Service, INIS (International Nuclear Information System), NASA Astrophysics Data System, SPIRES and VINITI Abstracts Journal (Referativnyi Zhurnal).

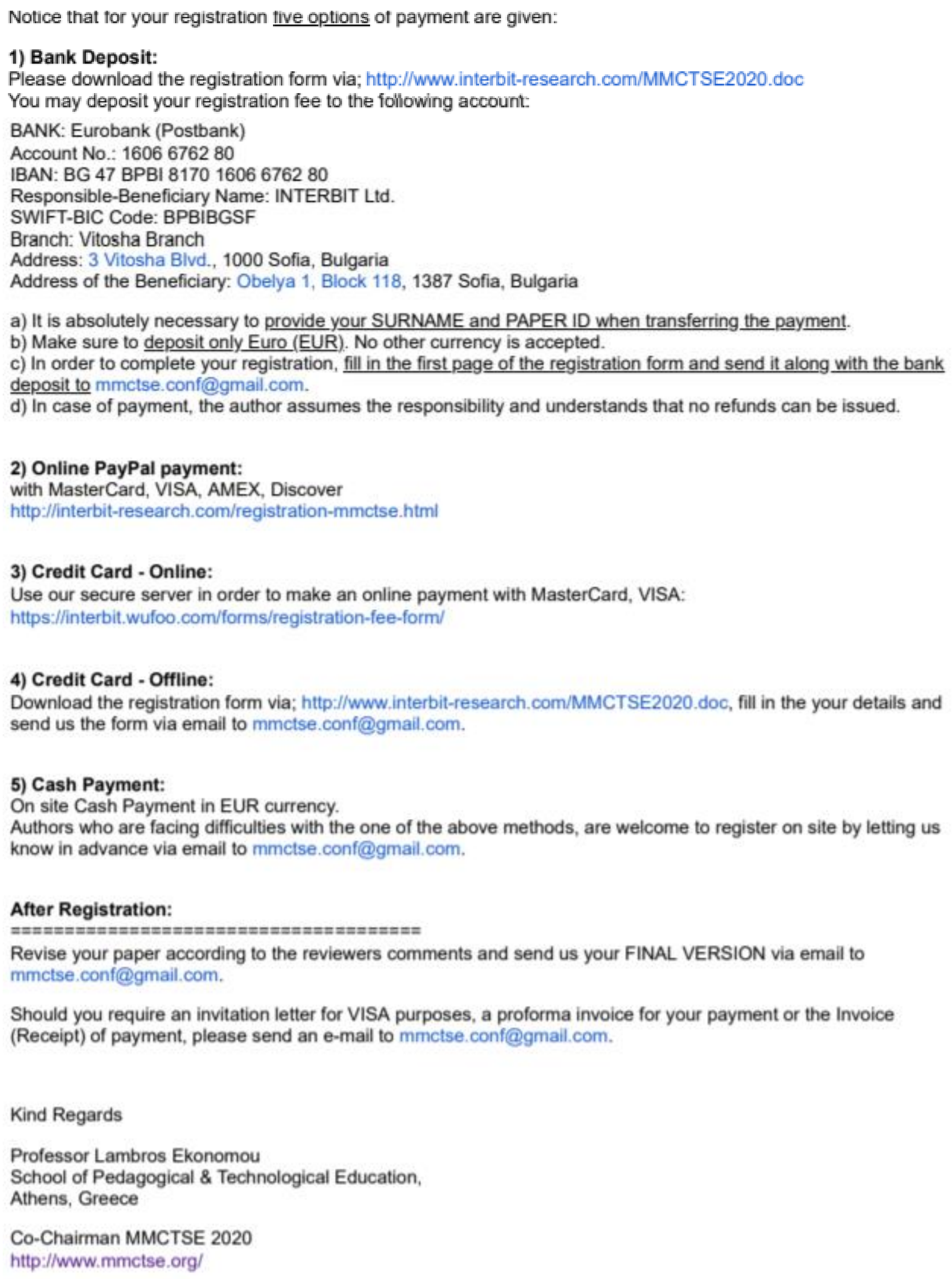
Extended versions will be considered for publication in various indexed ISI, SCOPUS, EI Compendex journals.

The Reviewers Comments for your paper will be sent to you just after your registration.

Registration Deadline: February 7, 2020
Revised paper submission Deadline: February 14, 2020

Full Paper Publication & Presentation: 500 EUR
Includes:
a) Direct Publication of the Accepted Paper in the Journal of Physics: Conference Series - IOPscience.
b) Presentation of the Paper at the conference.
c) Attendance of all Sessions.
d) Conference Bag with Conference Program, Certificate and secondary material.
e) Free Entrance in Tutorials, Workshops and Plenary Speeches.
f) Coffee-breaks with a variety of Coffee, Tea, Juices, Sweets etc.
g) Conference Dinner.
h) Welcoming Drink.
i) Extended version in various indexed ISI, SCOPUS, EI Compendex journals.

INSTRUCTIONS FOR YOUR REGISTRATION
==



Notice that for your registration five options of payment are given:

1) Bank Deposit:
Please download the registration form via; http://www.interbit-research.com/MMCTSE2020.doc
You may deposit your registration fee to the following account:

BANK: Eurobank (Postbank)
Account No.: 1606 6762 80
IBAN: BG 47 BPBI 8170 1606 6762 80
Responsible-Beneficiary Name: INTERBIT Ltd.
SWIFT-BIC Code: BPBIBGSF
Branch: Vitosha Branch
Address: 3 Vitosha Blvd., 1000 Sofia, Bulgaria
Address of the Beneficiary: Obelya 1, Block 118, 1387 Sofia, Bulgaria

a) It is absolutely necessary to provide your SURNAME and PAPER ID when transferring the payment.
b) Make sure to deposit only Euro (EUR). No other currency is accepted.
c) In order to complete your registration, fill in the first page of the registration form and send it along with the bank deposit to mmctse.conf@gmail.com.
d) In case of payment, the author assumes the responsibility and understands that no refunds can be issued.

2) Online PayPal payment:
with MasterCard, VISA, AMEX, Discover
http://interbit-research.com/registration-mmctse.html

3) Credit Card - Online:
Use our secure server in order to make an online payment with MasterCard, VISA:
https://interbit.wufoo.com/forms/registration-fee-form/

4) Credit Card - Offline:
Download the registration form via; http://www.interbit-research.com/MMCTSE2020.doc, fill in the your details and send us the form via email to mmctse.conf@gmail.com.

5) Cash Payment:
On site Cash Payment in EUR currency.
Authors who are facing difficulties with the one of the above methods, are welcome to register on site by letting us know in advance via email to mmctse.conf@gmail.com.

After Registration:
==
Revise your paper according to the reviewers comments and send us your FINAL VERSION via email to mmctse.conf@gmail.com.

Should you require an invitation letter for VISA purposes, a proforma invoice for your payment or the Invoice (Receipt) of payment, please send an e-mail to mmctse.conf@gmail.com.

Kind Regards

Professor Lambros Ekonomou
School of Pedagogical & Technological Education,
Athens, Greece

Co-Chairman MMCTSE 2020
http://www.mmctse.org/

APPENDIX 3

Copy of the paper submitted for the journal: World Scientific and Engineering Academy and Society (WSEAS) transactions on Systems and Control

Building a NIDS Using a Two-Stage Classifier and Feature Reduction through Statistical Methods

RAGHUVANSH RAJ, SOMYA GUPTA, MANAN LOHIA, H C TANEJA
Department of Mathematics
Delhi Technological University
Shahbad Daulatpur, Main Bawana Road, Delhi - 110042
INDIA
somya_bt2k16@dtu.ac.in   http://www.dtu.ac.in

Abstract: - Today, data is produced, transmitted and stored across the internet in abundant quantities. In such a world, modern-day network security systems have had to develop at an equally astounding rate in order to keep up with the data deluge. Subsequently, companies spend a substantial amount of money, time and effort in developing intrusion detection systems to ensure timely detection and prevention of malicious activity in order to preserve system security. In our paper, we propose a two-stage algorithm to solve the problem of NIDS resulting in fewer false positives and false negatives. We built and evaluated the performance of our IDS using the benchmarked NSL-KDD dataset, which consists of 41 features, of which 3 are categorical and the remaining numeric. Categorical features have been transformed via One Hot Encoding due to which the feature space explodes to 122 features. Our aim henceforth is to reduce this feature space through methods of feature selection and dimensionality reduction to develop a computationally inexpensive classifier capable of operating on the reduced feature space using sequential models. The work has involved using autoencoders for feature space reduction followed by a feed forward network, and has delivered encouraging results. We have then extended our analysis to identify features which can be eliminated without any substantial loss of information available for the classification algorithm. The remaining set of features can then be input into a different model to possibly provide better results or reduce training and evaluation time.

Key-Words: Intrusion Detection, Feature Reduction, Artificial Neural Networks, NSL-KDD

1 Introduction

In the current scenario, the Internet has become a necessity for daily life and is used for various purposes such as education business and banking. The Internet is extremely vast and diversified, connecting millions of devices with each other to create and save data but at the same time making it vulnerable to attacks on sensitive information. Exposure of one system could lead to the subsequent contamination of another. With the advancement of the Internet, the equipment for network security has also evolved. This field is extremely versatile and has grown in complexity to prevent the various attacks that have been created over time and put into action. Intrusion Detection Systems (IDS) monitor networks for malicious activity or privacy breaches to protect data confidentiality and integrity, i.e. any form of intrusion that may negatively affect privacy or safety. Depending upon the deployed location, IDS are majorly of two types; host-based and network-based. Host-based IDS are installed on PCs and network-based IDS are situated on networks.

These systems employ machine learning techniques differentiating the malicious usage patterns from the benign ones. In these techniques, a model is trained on a training set consisting of data points of both classes then applied on the test set to compare the predicted classes to the actual ones then tuned upon to improve the accuracy and complexity. Once the desired output is achieved, the trained model is employed in real-life scenarios to prevent malicious data packets from going through the network. To keep the model up-to-date, it is trained in further samples to identify new types of malicious attacks.

This paper employs the NSL-KDD dataset which is a balanced resampling of the KDD CUP 99 dataset [1]. This dataset contains 5 classes with 4 of them malicious attacks and 1 benign. The malicious classes are DoS, Probe, R2L and U2R.

The main algorithms in effect make use of rule-based methods such as SNORT [2], clustering methods such as fuzzy [3], [4] and mini batch kmeans [5], and neural networks [6], [7], [8]. These methods make use of the direct dataset which has a high variance and no clear decision boundary between the different classes, making it a tougher classification process and requiring a denser model network.

In this paper, we set out to propose and implement a new method which first consists of pre-processing via feature engineering using one hot encoding followed by dimensionality reduction using Principal Component Analysis (PCA) to improve the efficiency of the resultant dataset for the following steps. Secondly, outlier detection is implemented as a binary classification problem with the two classes being malicious and benign to classify the malicious attacks with less complexity. The last stage consists of multi-class classification with all 5 classes in an effort to further classify the attack and reduce the rate of false positives.

2 Related Work

Due to the increasing demand of computationally inexpensive and extremely accurate network intrusion detection systems, many algorithms have been recently proposed that yield unbeatable accuracies. Since the proposal of deep learning as a method for intrusion detection [9] by Professor Hinton in 2006, there have been huge advancements in every field of computational intelligence, as has been the case with intrusion detection systems.

[3], [4], make use of fuzzy clustering augmented with Artificial Neural Networks (ANN) which causes a drastic reduction in false positives. The technique used by [4] results in extremely less false positives due to the presence of an aggregating ANN after the classification ANNs.

[6] uses autoencoders to reduce the feature space and executes instructions on parallel processors. This paper focuses more on the complexity of the algorithm and the time taken by the algorithm, which is why feature space reduction is done on a rather drastic scale. Experiments show that such drastic reduction in feature space result in stark information loss which results in a less than ideal situation.

[7], [10] make use of Long Short-Term Memory (LSTM) and Recurrent Neural Network (RNN) to introduce context among the features. These kinds of neural networks are implemented by these papers in a bidirectional sense and since context is introduced in the first layer itself, the depth of the network can be limited to 2-3. Although forward propagation in this kind of network takes a substantial amount of time due to bidirectional connections in each layer, it causes minimal space overhead due to a radical reduction in the number of nodes and layers.

[11] uses genetic algorithms like crossover and mutation to search for the optimal feature space in neural networks. The idea of genetic algorithms is inspired by evolution and how it results in the fittest population (features in this case). Genetic algorithms simply need a fitness measure that results in the optimal feature space when run for multiple epochs but the drawback with this method lies in the complexity and time taken by the model as it is essentially running over the entire dataset multiple times.

In this paper we have attempted to create a model that runs on a reduced feature space and works as a robust classifier while providing minimum false negatives. For this, we use a two-step approach. First, anomaly detection is conducted which detects malicious data points. Then, multiclass classification which classifies data points into respective classes and reduces the number of false positives.

3 Preliminaries

In this section, a basic overview is provided of all the related terms such as IDS, dimensionality reduction, decision trees, ensemble methods and the NSL-KDD dataset.

3.1 Intrusion Detection Systems

Intrusion Detection Systems are classified into two major categories based on their approach and the techniques and algorithms used to analyze traffic data and detect intrusions, namely anomaly-based and signature-based.

Signature based IDS compare live traffic data to a stored database and try to match it with known intrusion patterns in order to detect attacks, i.e., they follow the well-known paradigm of rule-based systems. Signature-based IDS deliver high accuracy when detecting known intrusion patterns, but can fail to detect newer types of attacks due to a lack of any existing, match able patterns in the database. In an era where attackers are always finding newer ways to take control of a system or a network of systems, signature-based IDS fail to offer an adequate level of protection without a substantial degree of risk. Another drawback of such frameworks is the large administrative overhead required to regularly maintain and update the signature database to ensure that the system is able to detect even the newest types of attacks [12].

Contrarily, anomaly-based IDS rely on detecting deviations from what is considered to be normal network behaviour to identify malicious activity. Originally, anomaly-based systems used statistical methods to model data representing 'normal behaviour' and used these developed models to identify and flag any significant deviations as suspicious. However, recent efforts have pivoted to focus on using machine learning and deep learning along with advanced mathematical techniques to develop improved anomaly-based IDS. Major challenges that arise when attempting to build such models is the high false alarm rates, owing to the difficulty of modelling the 'normal behaviour' due to non-linearity of data and a certain degree of obscurity present in the data [13].

3.2 Dimensionality Reduction

Dimensionality reduction is an umbrella term for the various processes aimed at reducing the number of random variables in a dataset. This serves multifarious purposes such as enabling visualization of data and easing computational requirements. It can also help remove contextual redundancies in the dataset and simplify the overall process of data analysis and model construction. Dimensionality reduction is split majorly into two kinds of techniques, feature selection and feature extraction.

Feature selection involves the selection of a subset of the existing features. The selection of features can be guided by strategies involving information gain, gain ration, scoring features based on their importance etc.

Feature extraction is a more complex set of techniques that involve combining the existing variables through various statistical, algebraic or deep learning techniques to form a new, lower dimensional set of features which can be used to represent the dataset effectively. However, this technique suffers with a loss of context as it is not always possible to determine what exactly the new features represent and some information is always lost due to the decrease in number of features.

3.2.1 Principal Component Analysis

Principal Component Analysis (PCA) [5], [14] is a kind of dimensionality reduction technique that combines concepts from statistics and linear algebra to perform feature extraction. The process involves the use of covariance matrix and eigenvalues to transform the existing dataset into a new dataset using principal components derived from eigenvectors. The following steps are involved in PCA:

- For the dataset $D_{m \times n}$ containing *m* datapoints with *n* features, subtract from each element the mean of the column to which it belongs to centre the datapoints at the origin. Let this new matrix be D`.
- Calculate the covariance matrix od D`, denoted by C.
- Obtain the eigenvalues and eigenvectors of the matrix C.
- If *d* denotes the required number of dimensions for the new dataset, select the first *d* eigenvalues with the largest magnitudes, and combine their corresponding eigenvectors to form the transformation matrix $E_{n \times d}$.
- The reduced dataset is given by $F_{m \times d} = [D`_{m \times n}] \times [E_{n \times d}]$. (1)

3.2 Decision Trees

A decision tree is a classification algorithm that forms a flowchart in a tree-like graph where at each internal node an attribute is chosen on which to split the dataset on. Consequently, each branch represents one outcome of the split and any subsequent leaf nodes constitute the classes. The entire path from the root node to a leaf node can be regarded as a classification rule [15]. They map both linear as well as non-linear relationships.

The split at each internal node can be done on the basis of information gain and Gini index. The model proposed in this paper uses Gini index. Gini index forces a binary tree which is ideal in our case as we are initially working on a binary classification model. The underlying principle for this splitting algorithm is based on the fact that the probability of two randomly chosen data tuples from the dataset belonging to the same class is 1 if the dataset is pure.

3.4 Ensemble Methods

The usage of ensemble methods is ideal in the combination of a number of base models to produce an optimal model. These methods take into consideration a number of decision trees and use those to find the ideal split at each internal node by calculating which features to use.

Bagging [16] is an example of an ensemble method that combines Bootstrapping and Aggregation. From a sample of the dataset, multiple bootstrapped samples are taken upon which a decision tree is formed for each one. An algorithm is then used to aggregate the trees and find the most efficient predictor.

Random Forests [17] are another example. These use the same concept as bagging except that they work with the entire dataset rather than a sample. Each tree in this method splits at different features thus overall providing a larger ensemble to aggregate over, eventually producing a more accurate predictor.

3.5 NSL-KDD

The NSL-KDD dataset [18] is a refined version of the original KDD-CUP 99 dataset, which, after a thorough analysis revealed, a number of redundancies and other issues such as imbalanced classes and a huge number of records which offered many challenges to the construction of a classification model. The improved NSL-KDD dataset consists of 41 features and is split into the test set and training set. The training set has 125,973 datapoints while the test set has 22,544 datapoints. The test data is classified into 38 different classes, of which 21 are present in the training data. 1 of these classes corresponds to 'normal activity' while the others are various attack types. For the sake of brevity, the 37 other classes are categorized into 4 different attack types, as follows [19]:

Table 1. Categorization of the 37 different classes into the 4 different attack types.

S. No.	Attack Type	Class in Dataset
1	DoS	back, land, neptune, pod, smurf, teardrop, mailbomb, processtable, udpstorm, apache2, worm
2	Probe	satan, ipsweep, nmap, portsweep, mscan, saint
3	R2L	guess_password, ftp_write, imap, phf, multihop, warezmaster, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named
4	U2R	buffer_overflow, loadmodule, rootkit, perl, sqlattack, xterm, ps

We follow the aforementioned classification in our model. The following figures show the distribution of datapoints belonging to each class. As the distribution is highly skewed, we have used the log of the counts for each class where necessary.

Fig.1 Class distribution of the test set.

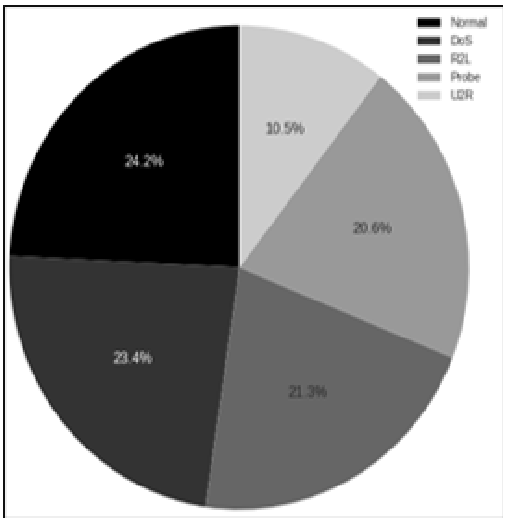
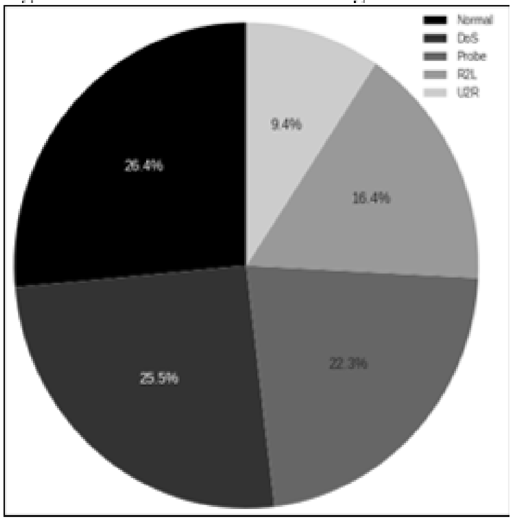


Fig.2 Class distribution of the training set.



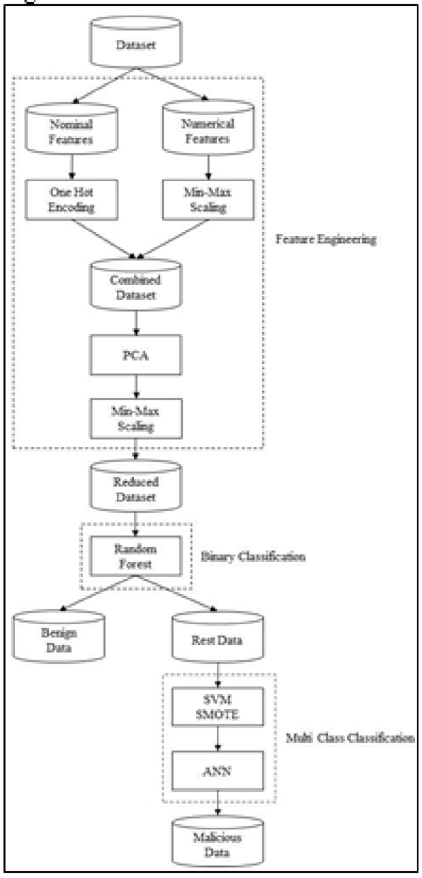
The 41 features can be divided into 4 major sets, consisting of the basic features for any network connection, content related features, time related traffic features and host-based traffic features [20]. A description of the basic features is given in [20] The remaining features are majorly binary variables which provide additional information about the network connection represented by the corresponding datapoints.

4 Proposed Methodology

In this section, we provide the proposed methodology for intrusion detection. The proposed schematic is shown in figure 3.

The proposed architecture consists of 4 steps; each of these are explained in detail further:

Fig.3 Flow of the model



4.1 Feature Engineering

The initial phase of the model is the feature engineering step. The NSL-KDD dataset consists of a training set and a testing set. The training set has 41 features, out of which the 20th feature, i.e., num_outbound_cmds which is the number of outbound commands in an ftp session [20], consists of only zeros so the column is dropped. After dropping the 20th column, we divide the dataset into nominal and numerical features depending on the data type. The nominal features undergo one-hot encoding which expands the number of features to become 84. The numerical features on the other hand undergo min-max scaling so that features with a broader range do not outweigh the features with a smaller range. They are a total of 37 features. The combined dataset of both numerical and nominal features gives a total of 121 features. The feature matrix formed is sparse due to one-hot encoding so dimensionality reduction is carried out using PCA on the 121 features. Retention of 67 features retains 99% variance of the original dataset. Then min-max scaling once again to bring all the features within the same range.

4.2 Binary Classification

In this step, binary classification is done to divide the dataset into two classes namely, malicious and benign. Ensemble methods have been tried and tested to see which one works best. Random Forest, AdaBoost and XGBoost have been compared in terms of precision, recall and area under the receiver operating characteristics (ROC) curve. The precision-recall curves have been plotted to calculate the threshold for the confidence score. The optimal threshold value is the intersection of the precision and recall curves in order to optimize both metrics. The aim of binary classification was to increase recall so as to reduce false negatives.

After applying binary classification, we have divided the dataset into two categories one with completely benign data points and another with both benign and malicious data points as the number of false positives are still relatively high.

4.3 Oversampling

Before we reach the step where we reduce the number of false positives, we oversample the dataset as the number of data points of U2R and R2L are extremely less in comparison to the rest, i.e., Normal, DoS and Probe. This could potentially lead to the problem of the network not being able to learn the datapoints of those 2 classes. Since the samples of U2R and R2L are less, the network is not able to classify points into these 2 classes properly resulting in false negatives. To solve this problem, we used SVMSMOTE which oversampled for the 2 classes and synthesized new data points.

4.4 Multi Class Classification

The last step is multi class classification. This is done to reduce the number of false positives occurring at the output of the model. An artificial neural network (ANN) is fitted onto the output class of binary classification that had both malicious and benign data points after oversampling.

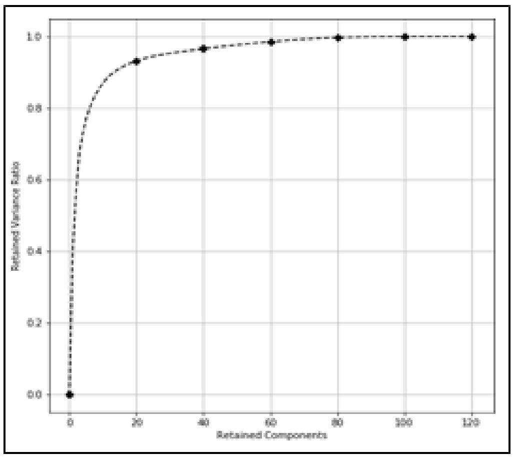
The datapoints are classified into the 5 classes, namely Normal, DoS, Probe, U2R and R2L after being fed through an ANN. The ANN architecture that we have implemented consists of Convolutional and LSTM layers and outputs the class probabilities for all the 5 classes using SoftMax Activation at the final layer and Rectified Linear Unit (ReLU) activations at every other layer. Datapoints classified as normal here as well as at the binary classification stage form the combined population of benign data points. Every other data point is categorized first as malicious and then further into the type of attack.

5 Experiment and Results

In this research, we have used scikit-learn and keras with a tensorflow backend for the construction of our models and we have used matplotlib for generating graphs. All of the models have been trained on Google Colaboratory using a Tesla K80 single core GPU with 25.51 GB RAM, although RAM usage throughout our experiments was limited to 10 GB.

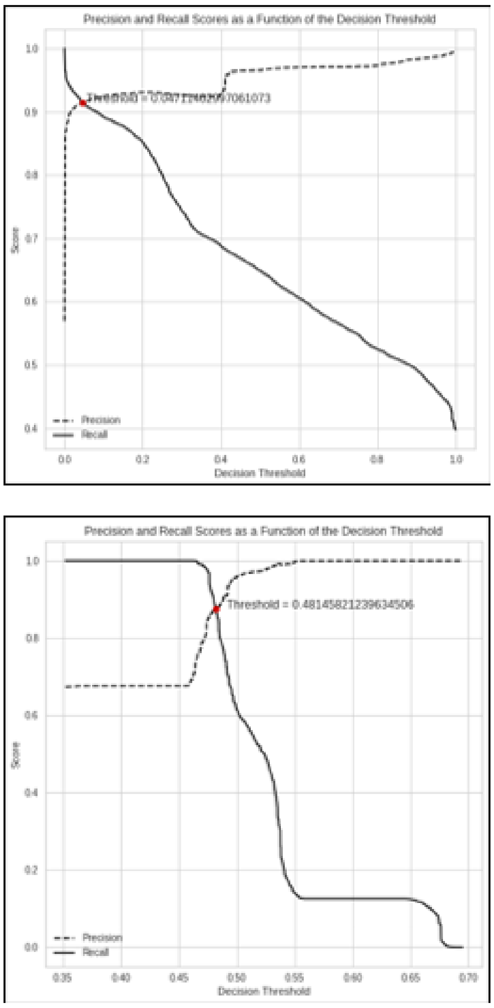
During feature engineering, we applied PCA for dimensionality reduction. The following graph shows the retention of variance corresponding to the number of components retained.

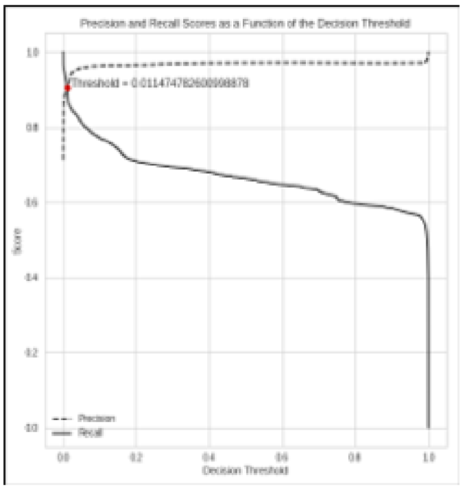
Fig.4 Variance Retention vs. Number of Components.

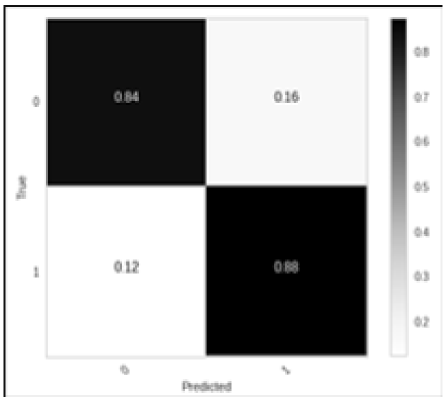


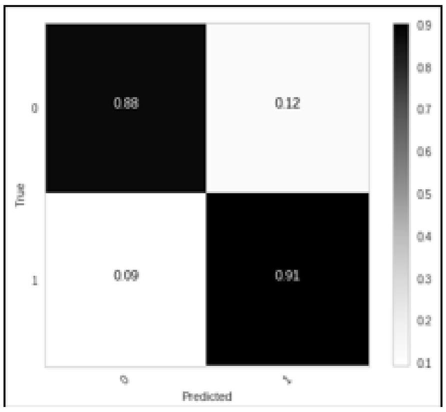
For binary classification we have compares 3 methods: Random Forest Classifiers and boosting algorithms AdaBoost and XGBoost. To obtain the optimal set of parameters for each of these classifiers, grid search was run on a suitable search space with recall as the optimizing metric. This was followed by comparing plots (Figure 5) of precision and recall and finding the optimal threshold for a point to be classified as malicious. Threshold tuning was based on the fact that we weren't looking for very high precision values at this stage; recall was of primary importance. Experiments suggested that threshold values that brought precision and recall closest together could be considered as optimal threshold points as the relation between these 2 metrics is almost inverse in nature, which means that increasing recall via threshold tuning would result in reduces precision.

Fig.5 Precision and recall scores as a function of the decision threshold of Random Forest, AdaBoost, and XGBoost respectively.









Random Forest Classifiers gave the best classification output overall (Fig.6) It and XGBoost had a recall of 91%, while AdaBoost gave a recall of 88%. Random Forest Classifiers gave a maximum precision of 89% while AdaBoost and XGBoost gave a precision of 84% and 88% respectively.

XGBoost had the maximum area under curve (AUC) of 0.96275, while Random Forest Classifiers and AdaBoost had an AUC of 0.94587 and 0.94139 respectively (Fig.7).

Fig.6 Confusion matrices of Random Forest, AdaBoost and XGBoost respectively.

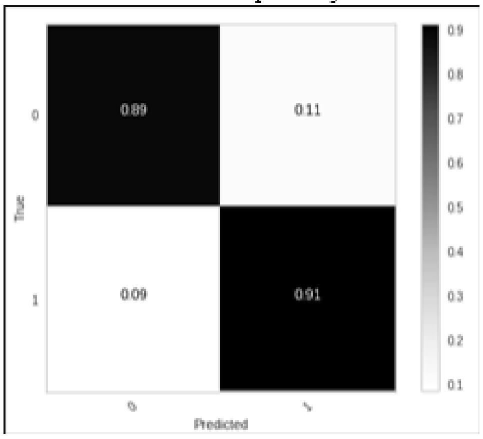
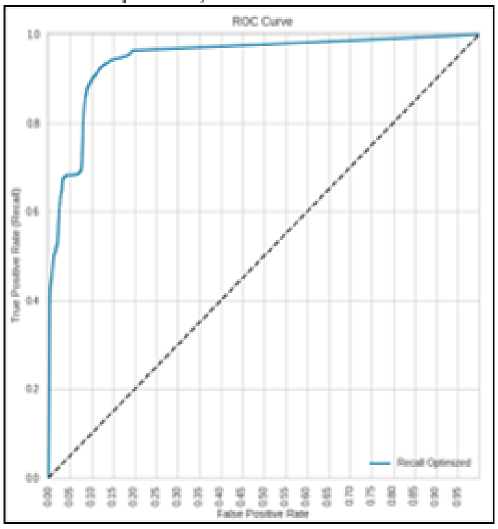
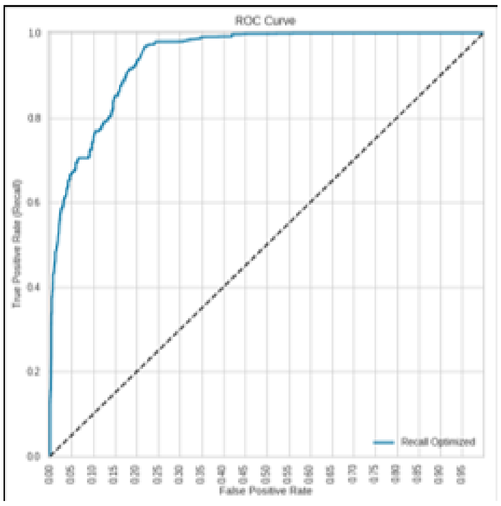
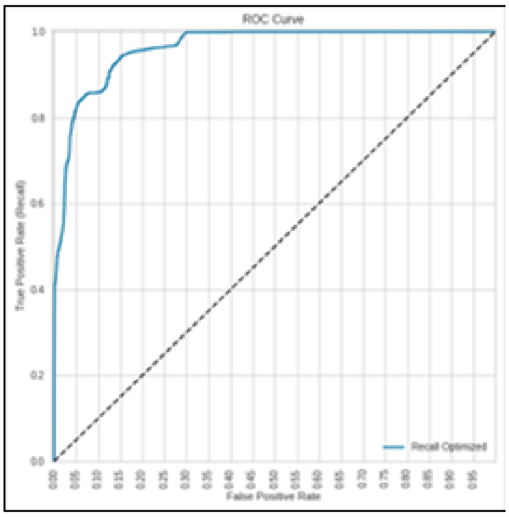


Fig.7 ROC curves of Random Forest, AdaBoost and XGBoost respectively

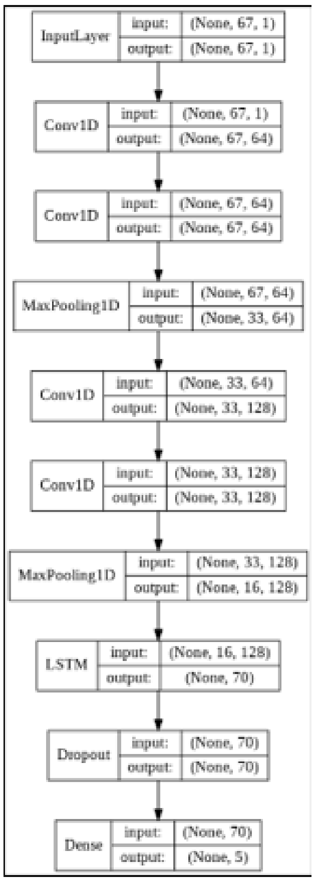






For multi-class classification, multiple neural network architectures were considered comprising of CNNs, RNNs, LSTMs, etc. The following architecture, as suggested by [21] yielded best results when trained on 100 epochs.

Fig.8 Architecture of the ANN



The loss convergence and accuracy over a course of 100 epochs are shown by the following graph.

Fig.9 Loss convergence of the model

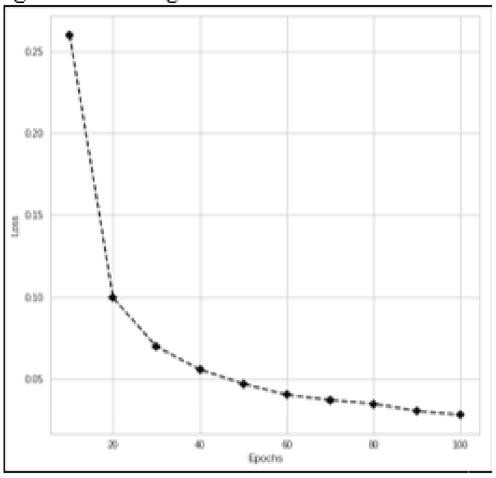
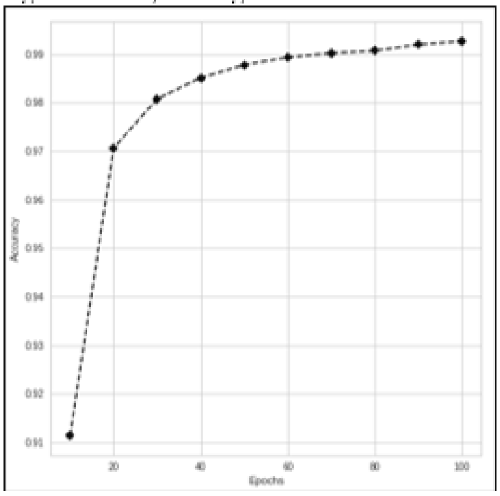
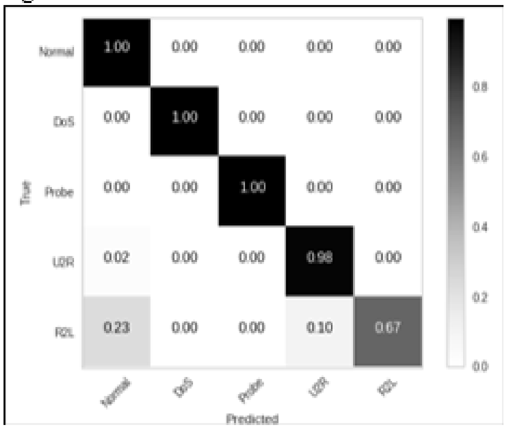


Fig.10 Accuracy convergence of the model.



The combined model, along with all binary classifiers in consideration combined with the multi class classifier gave the following results.

Fig.11 Confusion matrix of the multi class classifier.



6 Further Analysis

In this section, we resort to using various methods to help understand the importance of each feature viz a viz the information it provides in aiding the classification process. Three standard techniques have been used for the same, which are mentioned and further elaborated upon below:

6.1 Decision Tree Classifiers

For categorical variables, we used the weighted Gini index criterion after doing one hot encoding on the variables. Each individual categorical variable was separated into binary attributes. The weighted Gini index was calculated using the following formula:

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

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

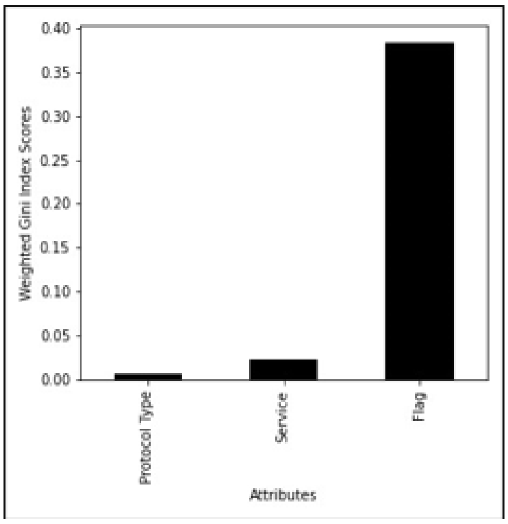
$$G(a_i) = Gini\ Index\ calculated\ for\ a_i$$

(2)

Categorical variables in the dataset have been inputted into a decision tree classifier to evaluate the importance of each in the classification process. This is measured as the total reduction of the Gini importance criterion caused due to each feature. The Gini Importance, also known as the 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 in a decision tree.

The conclusion in this regard is that while the Flag and Service variables are crucial in detecting intrusions, while the Protocol Type is not so important owing to its low Gini scores.

Fig.12 Feature Importance's using Decision Tree Classifier



6.2 Univariate Selection

For nominal features, we compared their importance using a chi square test (χ^2) which is commonly used for testing relationships between categorical variables. The null hypothesis of the Chi-Square test is that no relationship exists on the categorical variables in the population; they are independent.

We see from the following graph that these are the top 10 features we have identified in order of their importance.

Fig.13 Univariate Selection – Top 10 Features

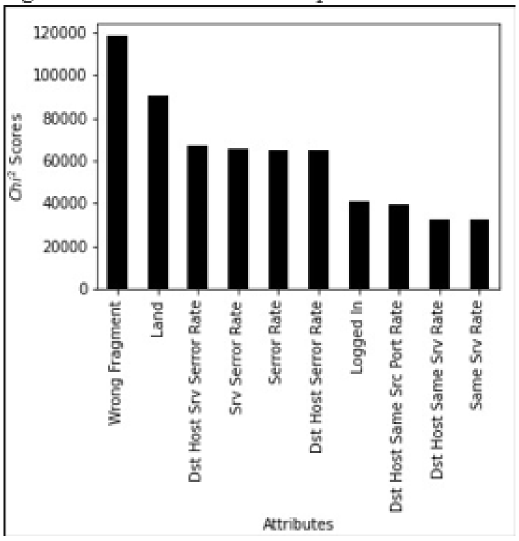
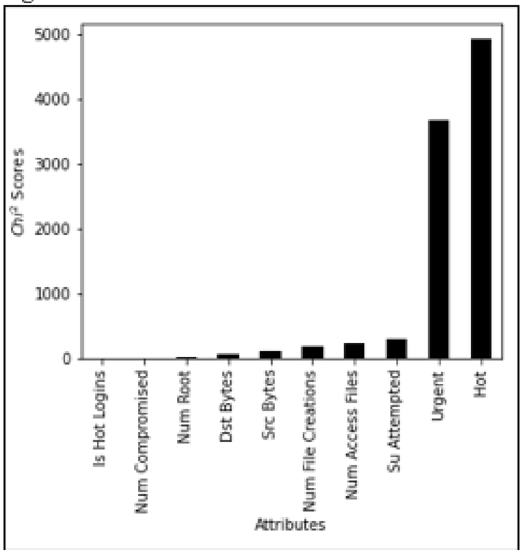


Fig.14 Univariate Selection – Bottom 10 Features



6.2 Correlation Plot

Correlation plots are used to determine the degree of correlation between every pair of features in a dataset. Each element corresponds to the Pearson's correlation coefficient between the two variables. Naturally, it is a symmetric matrix. Higher the correlation coefficient between two features, the stronger the linear relationship between them. Hence, each coefficient can be considered as a measure of the linear dependency between two variables. Ideally, we want a linearly independent set of features to maximize information present in the data and minimize redundancy. Assuming a threshold of 0.95 (or 95% correlation), we can identify 5 pairs of highly correlated features.

Fig.15 Correlation Heatmap with 95% Threshold

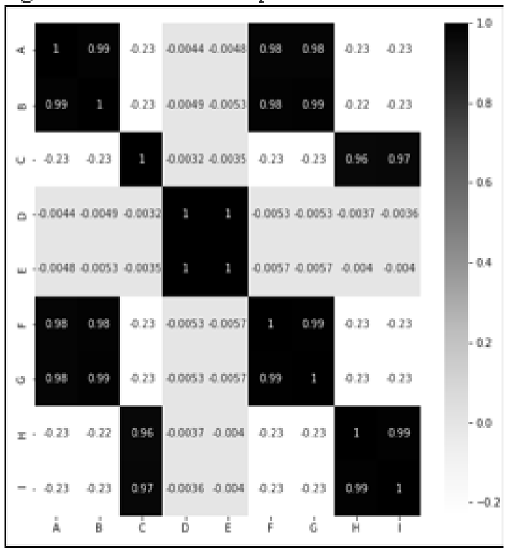


Table 2. Legend for Correlation Heatmap

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

7 Conclusion and Future Scope

The two-stage classification algorithm we considered here has resulted in much better evaluation results than the methods we explored. A reduced feature space ensures forward propagation through the model to work fast so that detection overhead is minimized.

Based on the further analysis, a total of 9 features: 8 numerical and 1 nominal, can be eliminated, reducing our feature space from 122 to 111. As the elimination has helped reduce redundancies in the dataset, we can expect our model to perform better due to advanced deep learning algorithms on the new dataset. There is further scope for removal based on correlation plot data which would require further analysis.

Future scopes for this architecture include using GANs for generating adversarial samples that can compensate for the lesser proportion of U2R and R2L samples and generate a more global feature space that would result in better model parameters and would detect a larger proportion of attacks. A number of other over sampling techniques can be compared to the results we have obtained in this paper. The model can be trained on the new feature space for an expected enhanced performance.

References:

[1] A. Divekar, M. Parekh, V. Savla, R. Mishra and M Shirole, Benchmarking datasets for Anomaly-based Intrusion Detection: KDD CUP 99 alternatives, *Proceedings on 2018 IEEE 3rd International Conference on Computing, Communication and Security, ICCCS 2018*, 2018

[2] M. Roesch, Snort – Lightweight Intrusion Detection for Networks, *Proceedings of LISA '99: 13th Systems Administration Conference,* Seattle, Washington, USA, 1999.

[3] G. Wang, J. Hao, J. Ma, L. Huang, A New Approach to Intrusion Detecting using Artificial Neural Networks and Fuzzy Clustering, *Expert Systems with Application,* vol. 37, no. 9, pp. 6225-6232, 2010

[4] S. Shraddha, Intrusion Detection using Fuzzy Clustering and Artificial Neural Network, *Advances in Neural Networks, Fuzzy Systems and Artificial Intelligence,* pp. 209-217

[5] K. Peng, V. C. M. Leung and Q. Huang, Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection System Over Big Data, *IEEE Access,* vol. 6, no. 1, pp. 11897-11906, 2018

[6] S. Potluri and C. Diedrich, Accelerated Deep Neural Networks for Enhanced Intrusion Detection System, *IEEE International Conference on Emerging Technologies and Factory Automation, EFTA,* 2016

[7] C. Yin, Y. Zhu, J. Fei and X. He, A Deep Learning Approach for Intrusion Detection using Recurrent Neural Networks, *IEEE Access,* vol. 5, no. 1, pp. 21954-21961, 2017

[8] J. Kim, N. Shin, S. Y. Jo and S. H. Kim, Method of Intrusion Detection Using Deep Neural Network, in *IEEE*, 2017

[9] Y. LeCun, Y. Bengio and G. Hinton, Deep Learning, *Nature,* vol. 521, no. 1, pp. 436-444, 2015

[10] J. Kim, J. Kim, H. L. T. Thu and H. Kim, "Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection," *IEEE*, 2016

[11] Z. Chiba, A. Noreddine, K. Moussaid, A. E. Omri, M. Rida, Intelligent and Improved Self-Adaptive Anomaly Based Detection System for Networks, *International Journal of Communication Networks and Information Security,* vol. 11, no. 2, pp. 312-330, 2019

[12] P. Manandhar and Z. Aung, Intrusion Detection Based on Outlier Detection Method, *International conference on Intelligent Systems, Data Mining and Information Technology,* Bangkok, Thailand, 2014

[13] J. Vacca, Computer and Information Security Handbook, *Elsevier*, 2009.

[14] M. E. Tipping and C. M. Bishop, Mixtures of Probabilistic Principal Component Analysers, *Journal of the Royal Statistical Society,* pp. 443-482, 1999

[15] K. Rai, M. S. Devi and A. Guleria, Decision Tree Based Algorithm for Intrusion Detection, *Advanced Networking and Applications,* vol. 7, no. 4, pp. 2828-2834, 2016

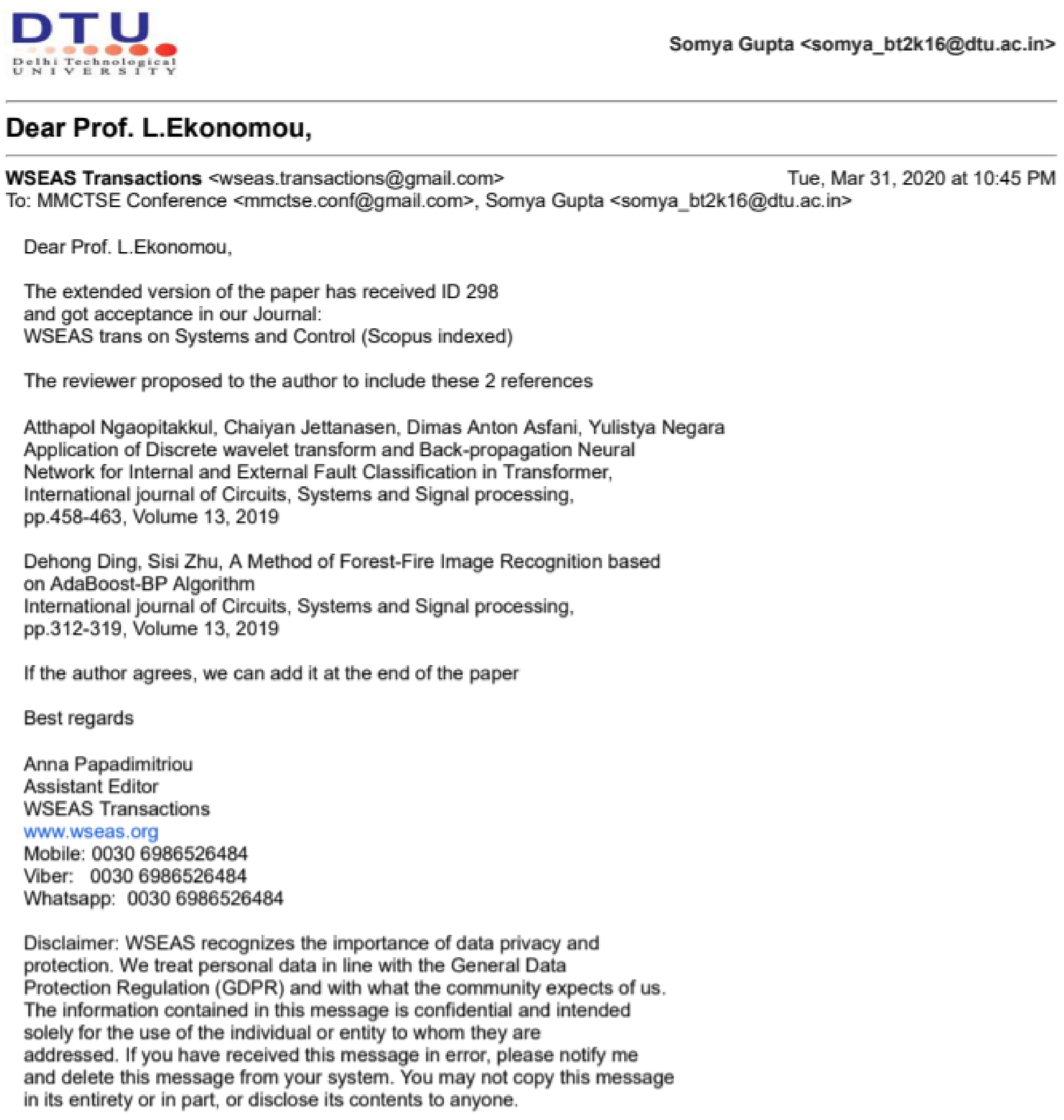
[16] D. P. Gaikwad and R. C. Thool, Intrusion Detection System using Bagging Ensemble Method of Machine Learning, *2015 International Conference on Computing Communication Control and Automation,* Pune, India, 2015

[17] J. Zhang, M. Zulkernine and A. Haque, Random Forests Based Network Intrusion Detection Systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 38, no. 5, pp. 649-659, 2008

APPENDIX 4

Copy of the acceptance letter received from the journal: World Scientific and Engineering Academy and Society (WSEAS) transactions on Systems and Control

The link for the paper accepted in the above mentioned journal is as given below: http://www.wseas.org/multimedia/journals/control/2020/a245103-055.pdf



DTU Delhi Technological University

Somya Gupta <somya_bt2k16@dtu.ac.in>

Dear Prof. L.Ekonomou,

WSEAS Transactions <wseas.transactions@gmail.com> Tue, Mar 31, 2020 at 10:45 PM
To: MMCTSE Conference <mmctse.conf@gmail.com>, Somya Gupta <somya_bt2k16@dtu.ac.in>

Dear Prof. L.Ekonomou,

The extended version of the paper has received ID 298
and got acceptance in our Journal:
WSEAS trans on Systems and Control (Scopus indexed)

The reviewer proposed to the author to include these 2 references

Atthapol Ngaopitakkul, Chaiyan Jettanasen, Dimas Anton Asfani, Yulistya Negara
Application of Discrete wavelet transform and Back-propagation Neural
Network for Internal and External Fault Classification in Transformer,
International journal of Circuits, Systems and Signal processing,
pp.458-463, Volume 13, 2019

Dehong Ding, Sisi Zhu, A Method of Forest-Fire Image Recognition based
on AdaBoost-BP Algorithm
International journal of Circuits, Systems and Signal processing,
pp.312-319, Volume 13, 2019

If the author agrees, we can add it at the end of the paper

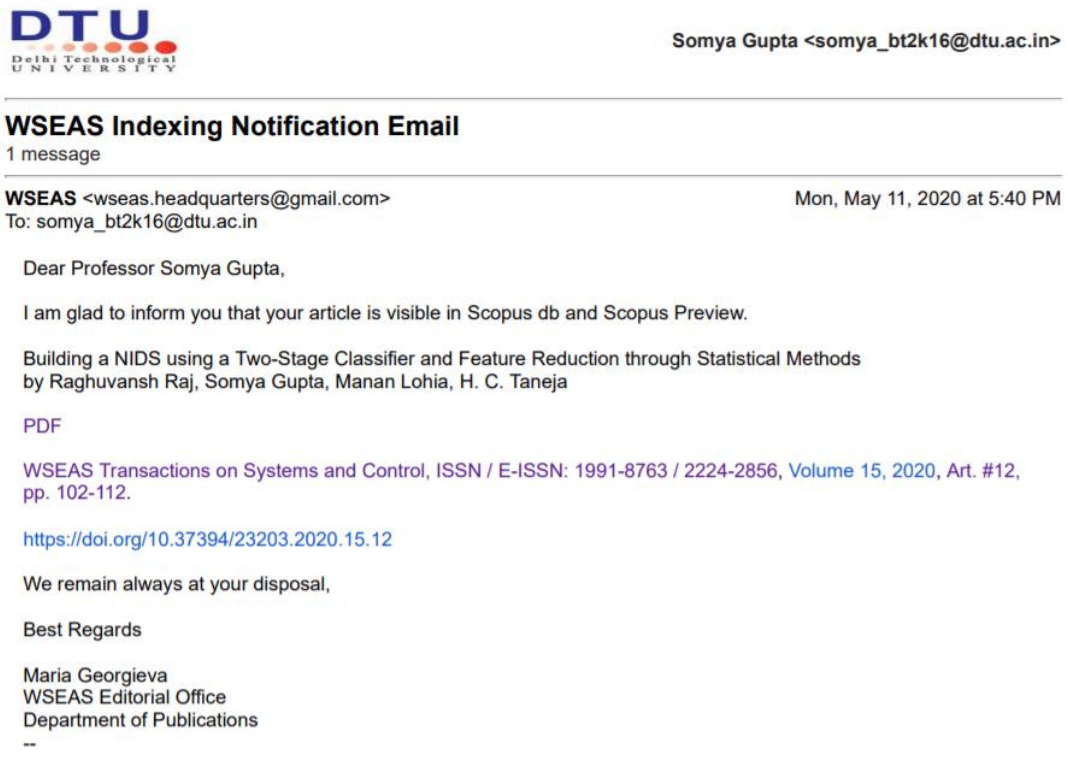
Best regards

Anna Papadimitriou
Assistant Editor
WSEAS Transactions
www.wseas.org
Mobile: 0030 6986526484
Viber: 0030 6986526484
Whatsapp: 0030 6986526484

Disclaimer: WSEAS recognizes the importance of data privacy and
protection. We treat personal data in line with the General Data
Protection Regulation (GDPR) and with what the community expects of us.
The information contained in this message is confidential and intended
solely for the use of the individual or entity to whom they are
addressed. If you have received this message in error, please notify me
and delete this message from your system. You may not copy this message
in its entirety or in part, or disclose its contents to anyone.

APPENDIX 5

Proof that journal is Scopus Indexed



Somya Gupta <somya_bt2k16@dtu.ac.in>

WSEAS Indexing Notification Email
1 message

WSEAS <wseas.headquarters@gmail.com> Mon, May 11, 2020 at 5:40 PM
To: somya_bt2k16@dtu.ac.in

Dear Professor Somya Gupta,

I am glad to inform you that your article is visible in Scopus db and Scopus Preview.

Building a NIDS using a Two-Stage Classifier and Feature Reduction through Statistical Methods
by Raghuvansh Raj, Somya Gupta, Manan Lohia, H. C. Taneja

PDF

WSEAS Transactions on Systems and Control, ISSN / E-ISSN: 1991-8763 / 2224-2856, Volume 15, 2020, Art. #12, pp. 102-112.

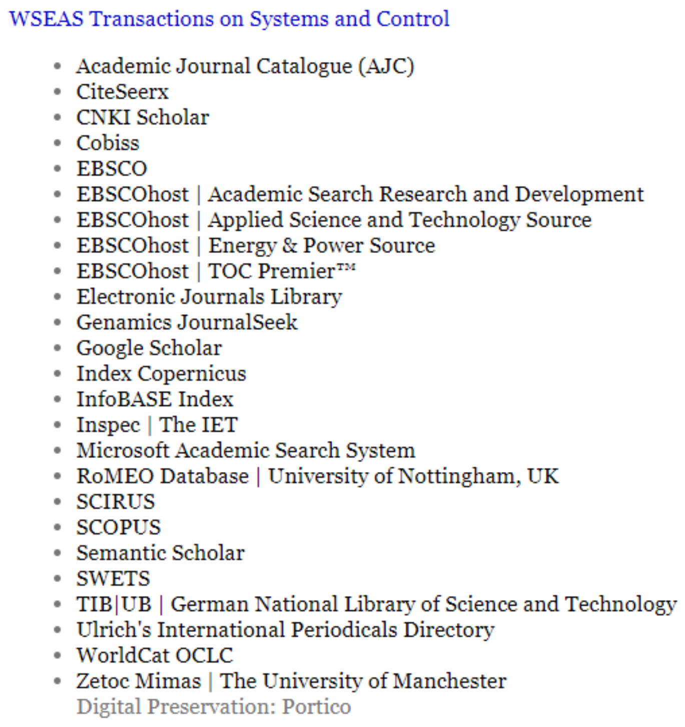
https://doi.org/10.37394/23203.2020.15.12

We remain always at your disposal,

Best Regards

Maria Georgieva
WSEAS Editorial Office
Department of Publications
--

Screenshot taken from their website: http://www.wseas.org/cms.action?id=6



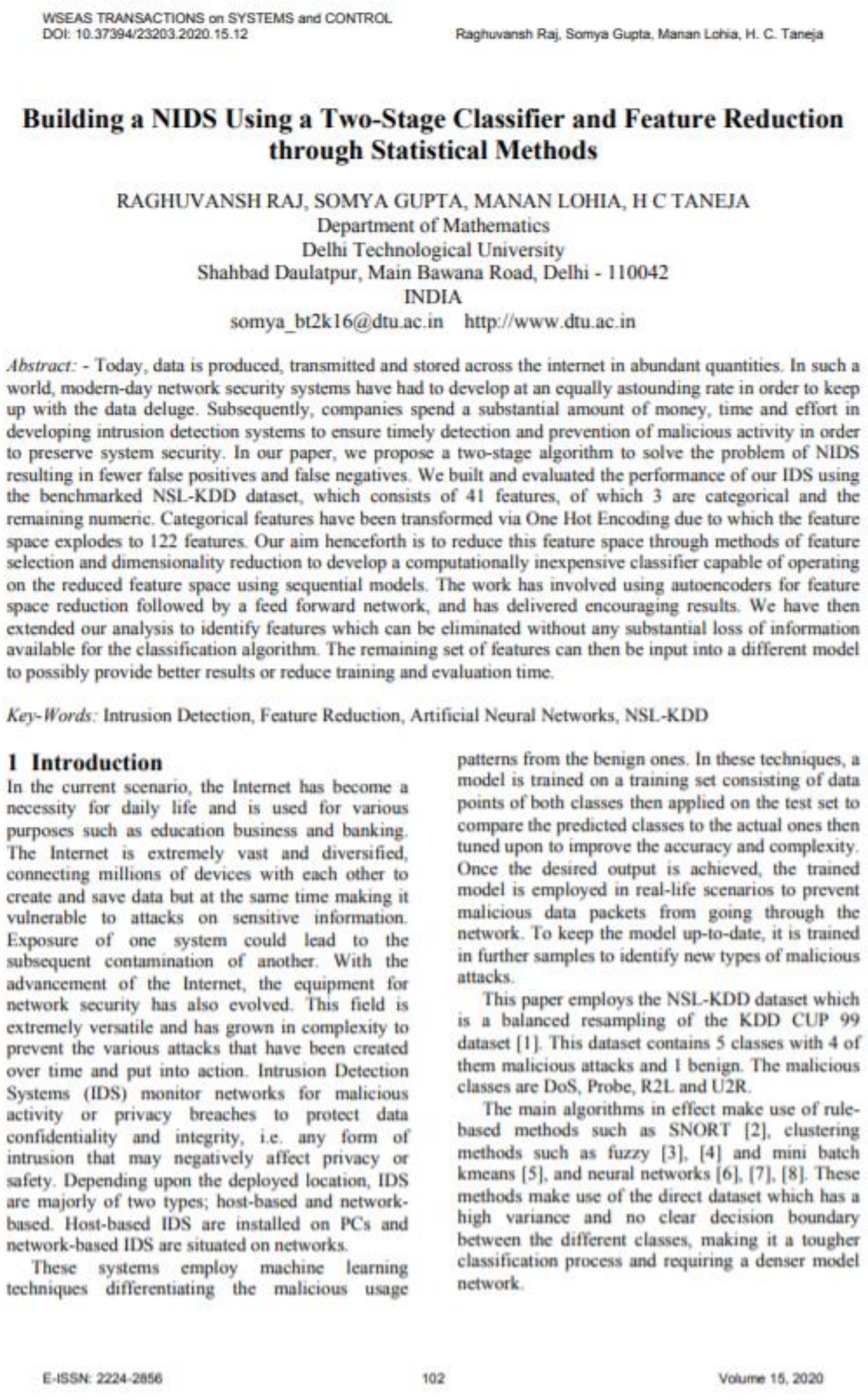
WSEAS Transactions on Systems and Control

- Academic Journal Catalogue (AJC)
- CiteSeerx
- CNKI Scholar
- Cobiss
- EBSCO
- EBSCOhost | Academic Search Research and Development
- EBSCOhost | Applied Science and Technology Source
- EBSCOhost | Energy & Power Source
- EBSCOhost | TOC Premier™
- Electronic Journals Library
- Genamics JournalSeek
- Google Scholar
- Index Copernicus
- InfoBASE Index
- Inspec | The IET
- Microsoft Academic Search System
- RoMEO Database | University of Nottingham, UK
- SCIRUS
- SCOPUS
- Semantic Scholar
- SWETS
- TIB|UB | German National Library of Science and Technology
- Ulrich's International Periodicals Directory
- WorldCat OCLC
- Zetoc Mimas | The University of Manchester

Digital Preservation: Portico

APPENDIX 6

Proof that journal has been published



WSEAS TRANSACTIONS on SYSTEMS and CONTROL
DOI: 10.37394/23203.2020.15.12

Building a NIDS Using a Two-Stage Classifier and Feature Reduction through Statistical Methods

RAGHUVANSH RAJ, SOMYA GUPTA, MANAN LOHIA, H C TANEJA
Department of Mathematics
Delhi Technological University
Shahbad Daulatpur, Main Bawana Road, Delhi - 110042
INDIA
somya_bt2k16@dtu.ac.in http://www.dtu.ac.in

Abstract: - Today, data is produced, transmitted and stored across the internet in abundant quantities. In such a world, modern-day network security systems have had to develop at an equally astounding rate in order to keep up with the data deluge. Subsequently, companies spend a substantial amount of money, time and effort in developing intrusion detection systems to ensure timely detection and prevention of malicious activity in order to preserve system security. In our paper, we propose a two-stage algorithm to solve the problem of NIDS resulting in fewer false positives and false negatives. We built and evaluated the performance of our IDS using the benchmarked NSL-KDD dataset, which consists of 41 features, of which 3 are categorical and the remaining numeric. Categorical features have been transformed via One Hot Encoding due to which the feature space explodes to 122 features. Our aim henceforth is to reduce this feature space through methods of feature selection and dimensionality reduction to develop a computationally inexpensive classifier capable of operating on the reduced feature space using sequential models. The work has involved using autoencoders for feature space reduction followed by a feed forward network, and has delivered encouraging results. We have then extended our analysis to identify features which can be eliminated without any substantial loss of information available for the classification algorithm. The remaining set of features can then be input into a different model to possibly provide better results or reduce training and evaluation time.

Key-Words: Intrusion Detection, Feature Reduction, Artificial Neural Networks, NSL-KDD

1 Introduction

In the current scenario, the Internet has become a necessity for daily life and is used for various purposes such as education business and banking. The Internet is extremely vast and diversified, connecting millions of devices with each other to create and save data but at the same time making it vulnerable to attacks on sensitive information. Exposure of one system could lead to the subsequent contamination of another. With the advancement of the Internet, the equipment for network security has also evolved. This field is extremely versatile and has grown in complexity to prevent the various attacks that have been created over time and put into action. Intrusion Detection Systems (IDS) monitor networks for malicious activity or privacy breaches to protect data confidentiality and integrity, i.e. any form of intrusion that may negatively affect privacy or safety. Depending upon the deployed location, IDS are majorly of two types; host-based and network-based. Host-based IDS are installed on PCs and network-based IDS are situated on networks.

These systems employ machine learning techniques differentiating the malicious usage patterns from the benign ones. In these techniques, a model is trained on a training set consisting of data points of both classes then applied on the test set to compare the predicted classes to the actual ones then tuned upon to improve the accuracy and complexity. Once the desired output is achieved, the trained model is employed in real-life scenarios to prevent malicious data packets from going through the network. To keep the model up-to-date, it is trained in further samples to identify new types of malicious attacks.

This paper employs the NSL-KDD dataset which is a balanced resampling of the KDD CUP 99 dataset [1]. This dataset contains 5 classes with 4 of them malicious attacks and 1 benign. The malicious classes are DoS, Probe, R2L and U2R.

The main algorithms in effect make use of rule-based methods such as SNORT [2], clustering methods such as fuzzy [3], [4] and mini batch kmeans [5], and neural networks [6], [7], [8]. These methods make use of the direct dataset which has a high variance and no clear decision boundary between the different classes, making it a tougher classification process and requiring a denser model network.

APPENDIX 7

Code implemented over the course of the project

```
# # Imports
import pandas as pd
import os
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.decomposition import PCA
import joblib
import matplotlib.pyplot as plt

# # Reading Data
DATA_DIR = '/content/drive/My Drive/Colab Notebooks/Intrusion
Detection/data/NSL-KDD'

# ## Training Data
train_data = pd.read_csv(os.path.join(DATA_DIR, 'KDDTrain+.txt'), header=None)

# ## Testing Data
test_data = pd.read_csv(os.path.join(DATA_DIR, 'KDDTest+.txt'), header=None)

# # Splitting Data Into Numeric and Nominal Features
# ## Dropping Redundant Columns
def remove_redundant_attributes(train_data, test_data):
    drop_cols = []
    for i in range(41):
        if train_data.loc[:, i].min() == train_data.loc[:, i].max():
            drop_cols.append(i)
    train_data_dropped = train_data.drop(drop_cols, axis=1)
    test_data_dropped = test_data.drop(drop_cols, axis=1)
    return train_data_dropped, test_data_dropped, drop_cols
train_data_dropped, test_data_dropped, dropped_cols =
remove_redundant_attributes(
    train_data, test_data)

# ## Numeric Features
numeric_features = np.asarray(
    pd.concat(
        [pd.DataFrame(train_data_dropped.loc[:,0]), train_data_dropped.loc[:, 4:40]],
        axis=1))
# ### MinMaxScaler
min_max_scaler_numeric = MinMaxScaler()
```

```
min_max_scaler_numeric.fit(numeric_features)

# ### Extraction
def extract_numeric_features(data, min_max_scaler):
    numeric_features = np.asarray(
        pd.concat(
            [pd.DataFrame(data.loc[:,0]), data.loc[:, 4:40]],
            axis=1 ))
        numeric_features_scaled = min_max_scaler.transform(numeric_features)
    numeric_features_final = numeric_features_scaled.astype('float64')
        return np.asarray(numeric_features_final)
numeric_features_train = extract_numeric_features(train_data_dropped,
min_max_scaler_numeric)
numeric_features_train.shape
numeric_features_test = extract_numeric_features(test_data_dropped,
min_max_scaler_numeric)
numeric_features_test.shape

# ## Nominal Features
nominal_features = np.asarray(train_data.loc[:, 1:3])

# ### One Hot Encoder
one_hot_encoder = OneHotEncoder(sparse=False)
one_hot_encoder.fit(nominal_features)
list(map(len, one_hot_encoder.categories_))

# ### Extraction
def extract_nominal_features(data, one_hot_encoder):
    nominal_features = np.asarray(data.loc[:, 1:3])
    nominal_features_one_hot = one_hot_encoder.transform(nominal_features)
    nominal_features_final = nominal_features_one_hot.astype('float64')
    return nominal_features_final
nominal_features_train = extract_nominal_features(train_data, one_hot_encoder)
nominal_features_train.shape
nominal_features_test = extract_nominal_features(test_data, one_hot_encoder)
nominal_features_test.shape

# # Final Features
# ## Training
final_features_train = np.concatenate([numeric_features_train,
nominal_features_train], axis=1)
final_features_train.shape

# ## Testing
final_features_test = np.concatenate([numeric_features_test, nominal_features_test],
axis=1)
final_features_test.shape
```

```
# ## PCA
pca = PCA()
pca.fit(final_features_train)
def get_components(pca, threshold):
    if threshold >= 1:
        threshold /= 100
    ratio_sum = 0
    i = 0
    for ratio in pca.explained_variance_ratio_:
        i += 1
        ratio_sum += ratio
        if ratio_sum >= threshold:
            return i, ratio_sum
    return None, None
get_components(pca, threshold=0.99)
variance_ratios = [0]
curr_sum = 0
for ratio in pca.explained_variance_ratio_:
    curr_sum += ratio
    variance_ratios.append(curr_sum)
len(variance_ratios)
np.linspace(10, 120, 12, dtype='int64') - 1
plt.figure(figsize=(8,8))
plt.plot(
    np.arange(122),
    variance_ratios,
    'k--',
    markevery=20,
    marker='D'
)
plt.grid()
plt.xlabel('Retained Components')
plt.ylabel('Retained Variance Ratio')
plt.savefig('Principal Component Analysis.png')

# ### 67 Components
pca_67dims = PCA(n_components=67)
pca_67dims.fit(final_features_train)
train_features_pca = pca_67dims.transform(final_features_train)
test_features_pca = pca_67dims.transform(final_features_test)

# ## MinMaxScaling
min_max_scaler_pca = MinMaxScaler()
min_max_scaler_pca.fit(train_features_pca)
train_features_scaled = min_max_scaler_pca.transform(train_features_pca)
test_features_scaled = min_max_scaler_pca.transform(test_features_pca)
train_features_scaled.shape
test_features_scaled.shape
```

```
# # Processing Labels
def combine_classes(labels, combine_dict):
    labels_multiclass = np.zeros((labels.shape[0], 5))
    for i in range(labels.shape[0]):
        labels_multiclass[i, combine_dict[labels[i]]] = 1
    labels_binary = np.zeros((labels.shape[0], 2))
    for i in range(labels.shape[0]):
        if labels[i] == 'normal':
            labels_binary[i, 0] = 1
        else:
            labels_binary[i, 1] = 1
    return pd.DataFrame(labels_multiclass).astype('int64'),
pd.DataFrame(labels_binary).astype('int64')
def extract_labels(data):
    labels = np.asarray(data.loc[:, 41])
    combine_dict = {
        'normal': 0,
        'neptune': 1,
        'warezclient': 3,
        'ipsweep': 2,
        'portsweep': 2,
        'teardrop': 1,
        'nmap': 2,
        'satan': 2,
        'smurf': 1,
        'pod': 1,
        'back': 1,
        'guess_passwd': 3,
        'ftp_write': 3,
        'multihop': 3,
        'rootkit': 4,
        'buffer_overflow': 4,
        'imap': 3,
        'warezmaster': 3,
        'phf': 3,
        'land': 1,
        'loadmodule': 4,
        'spy': 3,
        'perl': 4,
        'saint': 2,
        'mscan': 2,
        'apache2': 1,
        'snmpgetattack': 3,
        'processtable': 1,
        'httptunnel': 3,
        'ps': 4,
        'snmpguess': 3,
        'named': 3,
        'sendmail': 3,
        'xterm': 3,
        'worm': 1,
        'xlock': 3,
        'xsnoop': 3,
        'sqlattack': 4,
        'udpstorm': 1,
        'mailbomb': 3}
    final_labels, final_binary_labels = combine_classes(labels, combine_dict)
    return final_labels, final_binary_label
train_labels_multiclass, train_labels_binary = extract_labels(train_data)
train_labels_multiclass.shape, train_labels_binary.shape
test_labels_multiclass, test_labels_binary = extract_labels(test_data)
test_labels_multiclass.shape, test_labels_binary.shape

##DATA ANALYSIS
DATA_COLS = [
    'Duration',
    'Src Bytes',
    'Dst Bytes',
    'Land',
    'Wrong Fragment',
    'Urgent',
    'Hot',
    'Num Failed Logins',
    'Logged In',
    'Num Compromised',
    'Root Shell',
    'Su Attempted',
    'Num Root',
    'Num File Creations',
    'Num Shells',
    'Num Access Files',
    'Num Outbound Cmds',
    'Is Hot Logins',
    'Is Guest Login',
    'Count',
    'Srv Count',
    'Serror Rate',
    'Srv Serror Rate',
    'Rerror Rate',
    'Srv Rerror Rate',
    'Same Srv Rate',
    'Diff Srv Rate',
    'Srv Diff Host Rate',
    'Dst Host Count',
    'Dst Host Srv Count',
    'Dst Host Same Srv Rate',
    'Dst Host Diff Srv Rate',
    'Dst Host Same Src Port Rate',
    'Dst Host Srv Diff Host Rate',
    'Dst Host Serror Rate',
    'Dst Host Srv Serror Rate',
    'Dst Host Rerror Rate',
    'Dst Host Srv Rerror Rate',
    'Protocol Type',
    'Service',
    'Flag']
train_data = pd.read_csv(os.path.join(DATA_DIR, 'KDDTrain+.txt'), header=None)
train_data.head()

# # Data Segregation
numeric_features = pd.concat((train_data.iloc[:,0], train_data.iloc[:,4:41]), axis=1)
nominal_features = train_data.iloc[:,1:4]
labels = train_data.iloc[:,41]
numeric_cols = DATA_COLS[:-3]
nominal_cols = DATA_COLS[-3:]
numeric_features_np = numeric_features.to_numpy()
nominal_features_np = nominal_features.to_numpy()
labels_np = labels.to_numpy()

# ## MinMaxScaler
min_max_scaler = MinMaxScaler()

numeric_features_scaled = min_max_scaler.fit_transform(numeric_features_np)

# ## Get Dummies
nominal_features_one_hot = pd.get_dummies(nominal_features)
nominal_features_one_hot.head()
nominal_features_one_hot_np = nominal_features_one_hot.to_numpy()

# # Numeric Features
# ## Univariate Selection
select_k_best = SelectKBest(score_func=chi2, k='all')
k_best = select_k_best.fit(numeric_features_scaled, labels_np)
numeric_feature_univariate_selection = pd.Series(k_best.scores_,
index=numeric_cols)
top_n = 10
title = f'Univariate Selection - Top {top_n} Features'
```

```
fig = plt.figure()
numeric_feature_univariate_selection.nlargest(top_n).plot(
    kind='bar',
    figsize=(5,5),
    color='k'
    # title=title)
plt.ylabel('$Chi^2$ Scores')
plt.xlabel('Attributes')
fig.tight_layout()
plt.savefig(title + '.png')
last_n = 10
title = f'Univariate Selection - Last {last_n} Features'
fig = plt.figure()
numeric_feature_univariate_selection.nsmallest(last_n).plot(
    kind='bar',
    figsize=(5,5),
    color='k'
    # title=title)
plt.ylabel('$Chi^2$ Scores')
plt.xlabel('Attributes')
fig.tight_layout()
plt.savefig(title + '.png')

# ## Correlation Matrix with Heatmap
def plot_heatmap(features, cols, thresh=0.75, figsize=(20,20)):
    features_pd = pd.DataFrame(features)
    correlation_mat = features_pd.corr()
    top_corr_features = set()
    for i in range(38):
        for j in range(i+1,38):
            if abs(correlation_mat[i][j]) >= thresh:
                top_corr_features.add(i)
                top_corr_features.add(j)
    top_corr_features_list = list(top_corr_features)
    top_correlation_mat = features_pd.iloc[:, top_corr_features_list].corr()
    top_corr_cols = [cols[i] for i in top_corr_features_list]
    short_col_list = list()
    short_cols = []
    ch = 'A'
    for i in range(len(top_corr_cols)):
        short_col_list.append([chr(ord(ch) + i), top_corr_cols[i]])
        short_cols.append(chr(ord(ch) + i))
    fig = plt.figure(figsize=figsize)
    sns.heatmap(top_correlation_mat, xticklabels=short_cols, yticklabels=short_cols,
annot=True, cmap=plt.cm.Greys)
    title = f'Correlation Heatmap with {thresh*100}% Threshold'
    fig.tight_layout()
    plt.savefig(title + '.png')
    return short_col_list
short_col_list = plot_heatmap(numeric_features_scaled, numeric_cols, thresh=0.95,
figsize=(7,7))
short_col_list

# # Nominal Features
# ## Decision Tree Classifier
nominal_features.head()
dtc = DecisionTreeClassifier()
dtc.fit(nominal_features_one_hot_np, labels_np)
weights = []
for col in nominal_features.columns:
    weights.append(nominal_features[col].value_counts().to_dict())
for curr_val in weights:
    for key in curr_val.keys():
        curr_val[key] /= nominal_features.shape[0]
pprint(weights)
importances = [0] * 3
for i in range(84):
    curr_column = nominal_features_one_hot.columns[i]
    importances[int(curr_column[0])-1] += weights[int(curr_column[0])-
1][curr_column[2:]] * dtc.feature_importances_[i]
importances
importances_series = pd.Series(importances, index=DATA_COLS[-3:])
fig = plt.figure()
title = 'Feature Importances using Decision Tree Classifier'
importances_series.plot(
    kind='bar',
    # title=title,
    figsize=(5,5),
    color='k')
plt.ylabel('Weighted Gini Index Scores')
plt.xlabel('Attributes')
fig.tight_layout()
plt.savefig(title + '.png')

##INTRUSION DETECTION
# # Imports
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import os
import numpy as np
import pandas as pd
import torch.optim as optim
import torch.nn.functional as F
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import sklearn.cluster as cluster
from mpl_toolkits.mplot3d import axes3d
# from random import sample, shuffle
import joblib
from yellowbrick.cluster import KElbowVisualizer
from scipy.stats import zscore, multivariate_normal
import seaborn as sns
from sklearn.metrics import confusion_matrix, f1_score, make_scorer,
accuracy_score, precision_score, recall_score,
precision_recall_curve, roc_curve, auc
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.utils import shuffle
from keras import Sequential
from keras.layers import Dense, Dropout, Convolution1D, MaxPooling1D, LSTM
from keras.utils import to_categorical
from keras.regularizers import l2
from keras.optimizers import Adam
from keras.utils import plot_model
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE, SMOTENC, SVMSMOTE
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# # Constants
CSV_DIR = '/content/drive/My Drive/Colab Notebooks/Intrusion
Detection/data/NSL-KDD/csv'
BATCH_SIZE = 128
FIG_TITLE_SIZE = 15
LEGEND_TEXT_SIZE = 11
MODELS_DIR = '/content/drive/My Drive/Colab Notebooks/Intrusion
Detection/saved_models/grid_search'

# # Device Check for GPU
```

```
device = torch.device('cuda: 0' if torch.cuda.is_available() else 'cpu')
print(device)

# # Dataset & DataLoader
# ## NSLKDDDataset
class NSLKDDDataset(Dataset):
        def __init__(self, features, labels):
        self.features = np.asarray(features)
        self.labels = np.asarray(labels)
    def __len__(self):
        return self.features.shape[0]
    def __getitem__(self, idx):
        return (torch.from_numpy(self.features[idx]),
torch.from_numpy(self.labels[idx]))

# ## NSLKDDFeatures
class NSLKDDFeatures(Dataset):
    def __init__(self, features):
        self.features = np.asarray(features)
    def __len__(self):
        return self.features.shape[0]
    def __getitem__(self, idx):
        return torch.from_numpy(self.features[idx])

# ## NSLKDDSiameseNetDataSet
class NSLKDDSiameseNetDataSet(Dataset):
    def __init__(self, partitioned_data, indices, labels, nsl_kdd_features):
        self.partitioned_data = partitioned_data
        self.indices = indices
        self.labels = labels
        self.nsl_kdd_features = nsl_kdd_features
    def __len__(self):
        return len(self.indices)
    def __getitem__(self, idx):
        idx_tuple = self.indices[idx]
        if idx_tuple[0] == idx_tuple[1]:
            ret_list = self.partitioned_data[idx_tuple[0]][
                np.random.choice(
                    self.partitioned_data[idx_tuple[0]].shape[0], 2, replace=False   )]
        else:
            ret_list = list(map(
                lambda x: self.partitioned_data[x][
                    np.random.choice(self.partitioned_data[x].shape[0], 1)[0]],
                idx_tuple))
    return (
        torch.from_numpy(ret_list[0]),
        torch.from_numpy(ret_list[1]),
        torch.tensor(self.labels[idx], dtype=torch.int64))

# ## Training Data
train_features_pd = pd.read_csv(os.path.join(CSV_DIR, 'training',
'train_features_pca.csv'), header=None)
train_labels_multiclass_pd = pd.read_csv(os.path.join(CSV_DIR, 'training',
'train_labels_multiclass.csv'), header=None)
train_labels_binary_pd = pd.read_csv(os.path.join(CSV_DIR, 'training',
'train_labels_binary.csv'), header=None)
train_features_pd.shape
train_multiclass_ds = NSLKDDDataset(train_features, train_labels_multiclass)
train_binary_ds = NSLKDDDataset(train_features, train_labels_binary)
train_features_ds = NSLKDDFeatures(train_features)
train_multiclass_dl = DataLoader(
    train_multiclass_ds,
    batch_size=BATCH_SIZE,
    shuffle=True)
train_binary_dl = DataLoader(
    train_binary_ds,
    batch_size=BATCH_SIZE,
    shuffle=True)
train_features_dl = DataLoader(
    train_features_ds,
    batch_size=BATCH_SIZE,
    shuffle=True)

# ## Testing Data
test_features_pd = pd.read_csv(os.path.join(CSV_DIR, 'testing',
'test_features_pca.csv'), header=None)
test_labels_multiclass_pd = pd.read_csv(os.path.join(CSV_DIR, 'testing',
'test_labels_multiclass.csv'), header=None)
test_labels_binary_pd = pd.read_csv(os.path.join(CSV_DIR, 'testing',
'test_labels_binary.csv'), header=None)
test_features_pd.shape
test_multiclass_ds = NSLKDDDataset(test_features, test_labels_multiclass)
test_binary_ds = NSLKDDDataset(test_features, test_labels_binary)
test_features_ds = NSLKDDFeatures(test_features)
test_multiclass_dl = DataLoader(
    test_multiclass_ds,
    batch_size=BATCH_SIZE,
    shuffle=True)
```

```
test_binary_dl = DataLoader(
    test_binary_ds,
    batch_size=BATCH_SIZE,
    shuffle=True)
test_features_dl = DataLoader(
    test_features_ds,
    batch_size=BATCH_SIZE,
    shuffle=True)

# ## NumPy Arrays
train_features_np = np.asarray(train_features_pd)
test_features_np = np.asarray(test_features_pd)
train_labels_binary_np = np.asarray(train_labels_binary_pd)
test_labels_binary_np = np.asarray(test_labels_binary_pd)
train_labels_multiclass_np = np.asarray(train_labels_multiclass_pd)
test_labels_multiclass_np = np.asarray(test_labels_multiclass_pd)

# # Data Plots
# ## Plot Pie Chart
def plot_pie_chart(labels_one_hot, title, figure_size=(8,8), have_title=True,
log_scaling=False):
    labels = np.argmax(labels_one_hot, axis=1)
    names = ['Normal', 'DoS', 'Probe', 'R2L', 'U2R']
    _, sizes = np.unique(labels, return_counts=True)
    names_sizes_dict = dict(zip(names, sizes))
    sorted_names_sizes_dict = {k: v for k, v in sorted(
        names_sizes_dict.items(),
        key=lambda item: item[1],
        reverse=True )}
    names = list(sorted_names_sizes_dict.keys())
    sizes = list(sorted_names_sizes_dict.values())
    if log_scaling:
        sizes = np.log(sizes)
    fig = plt.figure(figsize=figure_size)
    ax = fig.add_subplot()
    theme = plt.get_cmap('gray')
    ax.set_prop_cycle('color', [theme(1. * i / len(sizes))
                         for i in range(len(sizes))])
    patches, texts, autotexts = ax.pie(
        sizes,
        autopct='%1.1f%%',
        startangle=90)
    autotexts[0].set_color('w')
    autotexts[1].set_color('w')
    # for text in autotexts:
    #     text.set_fontsize(11)
    ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    if have_title:
        plt.title(
            title,
            fontsize=LEGEND_TEXT_SIZE  )
    plt.legend(
        names,
        fontsize=11)
    plt.savefig(title + '.png')
plot_pie_chart(
    train_labels_multiclass_np,
    title='Class Distributions: Training Data (log scaled)',
    figure_size=(10,10),
    have_title=False,
    log_scaling=True)
plot_pie_chart(
    test_labels_multiclass_np,
    title='Class Distributions: Testing Data (log scaled)',
    figure_size=(10,10),
    have_title=False,
    log_scaling=True)

# # Model
# ## Autoencoder
class Autoencoder(nn.Module):
    def __init__(self, in_features, out_features):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(in_features=in_features, out_features=out_features),
            nn.Sigmoid())
                self.decoder = nn.Sequential(
            nn.Linear(in_features=out_features, out_features=in_features),
            nn.Sigmoid())
    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# ## AnomalyDetector
class AnomalyDetector(nn.Module):
    def __init__(self):
        super(AnomalyDetector, self).__init__()
        self.activation = nn.LeakyReLU(inplace=True)
        self.model = nn.Sequential(
            # nn.BatchNorm1d(num_features=67),
            nn.Linear(in_features=67, out_features=100),
            self.activation,
            nn.BatchNorm1d(num_features=100),
            nn.Linear(in_features=100, out_features=120),
            self.activation,
            nn.BatchNorm1d(num_features=120),
            nn.Linear(in_features=120, out_features=60),
            self.activation,
            nn.BatchNorm1d(num_features=60),
            nn.Linear(in_features=60, out_features=30),
            self.activation,
            nn.BatchNorm1d(num_features=30),
            nn.Linear(in_features=30, out_features=10),
            self.activation,
            nn.BatchNorm1d(num_features=10),
            nn.Linear(in_features=10, out_features=2),
            nn.Softmax(dim=1)    )
    def forward(self, x):
        x = self.model(x)
        return x

# ## SignatureGenerator
class SignatureGenerator(nn.Module):
    def __init__(self):
        super(SignatureGenerator, self).__init__()
        self.activation = nn.LeakyReLU(inplace=True)
        self.model = nn.Sequential(
            nn.Linear(in_features=67, out_features=100),
            self.activation,
            nn.BatchNorm1d(num_features=100),
            nn.Linear(in_features=100, out_features=80),
            self.activation,
            nn.BatchNorm1d(num_features=80),
            nn.Linear(in_features=80, out_features=60),
            self.activation,
            nn.BatchNorm1d(num_features=60),
            nn.Linear(in_features=60, out_features=40),
            self.activation,
            nn.BatchNorm1d(num_features=40),
            nn.Linear(in_features=40, out_features=20),
            self.activation,
            nn.BatchNorm1d(num_features=20),
            nn.Linear(in_features=20, out_features=10),
            nn.Sigmoid() )
    def forward(self, x):
        x = self.model(x)
        return x

# ## L1Difference
class L1Difference(nn.Module):
    def __init__(self):
        super(L1Difference, self).__init__()
    def forward(self, x1, x2):
        return abs(x1 - x2)

# ## SiameseNet
class SiameseNet(nn.Module):
    def __init__(self):
        super(SiameseNet, self).__init__()
        self.signature_gen = SignatureGenerator()
        self.l1_diff = L1Difference()
        self.fc = nn.Linear(in_features=10, out_features=2)
    def forward(self, x1, x2):
        x1_sig = self.signature_gen(x1)
        x2_sig = self.signature_gen(x2)
        l1_diff = self.l1_diff(x1_sig, x2_sig)
        out = torch.softmax(self.fc(l1_diff), dim=1)
        return out

# ## MultiClassClassifier
class MultiClassClassifier(nn.Module):
    def __init__(self):
        super(MultiClassClassifier, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(in_features=67, out_features=120),
            nn.ReLU(inplace=True),
            nn.BatchNorm1d(num_features=120),
            nn.Linear(in_features=120, out_features=100),
            nn.ReLU(inplace=True),
            nn.BatchNorm1d(num_features=100),
            nn.Linear(in_features=100, out_features=80),
            nn.ReLU(inplace=True),
            nn.BatchNorm1d(num_features=80),
            nn.Linear(in_features=80, out_features=40),
            nn.ReLU(inplace=True),
            nn.BatchNorm1d(num_features=40),
            nn.Linear(in_features=40, out_features=20),
            nn.ReLU(inplace=True),
            nn.BatchNorm1d(num_features=20),
            nn.Linear(in_features=20, out_features=10),
            nn.ReLU(inplace=True),
            nn.BatchNorm1d(num_features=10),
            nn.Linear(in_features=10, out_features=5),
            nn.Softmax(dim=1)        )
    def forward(self, x):
        return self.model(x)

# # Training Utilities
# ## Network Loss
def get_network_loss(net, dataloader):
    agg_loss = 0.0
    criterion = nn.CrossEntropyLoss()
    for i, data in enumerate(dataloader, 0):
        x, y = data
        y = torch.argmax(y, dim=1)
        x = x.to(device)
        y = y.to(device)
        outputs = net(x)
        loss = criterion(outputs, y)
        agg_loss += loss.item()
        return agg_loss / len(dataloader)

# ## Train Network
def train_network(net, trainloader, testloader, n_epochs, print_every=500):
    net.double()
    net.to(device)
    optimizer = optim.Adam(net.parameters())
    criterion = nn.CrossEntropyLoss()
    train_losses = []
    test_losses = []
    for epoch in range(n_epochs):
        running_loss = 0.0
        for i, data in enumerate(trainloader):
            x, y = data
            y = torch.argmax(y, dim=1)
            x = x.to(device)
            y = y.to(device)
            optimizer.zero_grad()
            outputs = net(x)
            loss = criterion(outputs, y)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if i % print_every == print_every-1:
                print('[%d, %5d] loss: %.6f' %
                  (epoch + 1, i + 1, running_loss / print_every))
                running_loss = 0.0
        net.eval()
        train_loss = get_network_loss(net, trainloader)
        test_loss = get_network_loss(net, testloader)
        net.train()
        train_losses.append(train_loss)
        test_losses.append(test_loss)
        print('Training Loss: %.6f' % (train_loss))
        print('Testing Loss: %.6f' % (test_loss))
    return train_losses, test_losses

# ## Encoder Loss
def get_encoder_loss(autoencoder, dataloader, criterion, autoencoder1=None,
autoencoder2=None):
    agg_loss = 0.0
    for i, data in enumerate(dataloader, 0):
        x = data
        x = x.to(device)
        if autoencoder1:
            x = autoencoder1.encoder(x)
        if autoencoder2:
            x = autoencoder2.encoder(x)
        outputs = autoencoder(x)
        loss = criterion(outputs, x)
        agg_loss += loss.item()
        return agg_loss / len(dataloader)

# ## Train Encoder
def train_encoder(autoencoder, trainloader, testloader, n_epochs, print_every=500,
autoencoder1=None, autoencoder2=None):
    autoencoder.double()
    autoencoder.to(device)
    optimizer = optim.Adam(autoencoder.parameters())
    criterion = nn.MSELoss()
    train_losses = []
    test_losses = []
    for epoch in range(n_epochs):
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            x = data
            x = x.to(device)
            with torch.no_grad():
                if autoencoder1:
                    x = autoencoder1.encoder(x)
                if autoencoder2:
                    x = autoencoder2.encoder(x)
            optimizer.zero_grad()
            outputs = autoencoder(x)
            loss = criterion(outputs, x)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if i % print_every == print_every-1:
                print('[%d, %5d] loss: %.6f' %
                  (epoch + 1, i + 1, running_loss / print_every))
                running_loss = 0.0
        autoencoder.eval()
        train_loss = get_encoder_loss(autoencoder, trainloader, criterion,
autoencoder1=autoencoder1, autoencoder2=autoencoder2)
        test_loss = get_encoder_loss(autoencoder, testloader, criterion,
autoencoder1=autoencoder1, autoencoder2=autoencoder2)
        autoencoder.train()
        train_losses.append(train_loss)
        test_losses.append(test_loss)
    return train_losses, test_losses

# ## SiameseNet Loss
def get_siamese_net_loss(siamese_net, dataloader, val_batches):
    agg_loss = 0.0
    criterion = nn.CrossEntropyLoss()
    for i, data in enumerate(dataloader, 0):
        x1, x2, y = data
        x1 = x1.to(device)
        x2 = x2.to(device)
        y = y.to(device)
        outputs = siamese_net(x1.double(), x2.double())
        loss = criterion(outputs, y)
        agg_loss += loss.item()
        if i == val_batches-1:
            break
    return agg_loss / val_batches
```

```
# ## Train SiameseNet
def train_siamese_net(siamese_net, trainloader, testloader, n_epochs,
print_every=500, val_batches=1):
    siamese_net.double()
    siamese_net.to(device)
    optimizer = optim.Adam(siamese_net.parameters())
    criterion = nn.CrossEntropyLoss()
    train_losses = []
    test_losses = []
    for epoch in range(n_epochs):
        running_loss = 0.0
        for i, data in enumerate(trainloader):
            x1, x2, y = data
            x1 = x1.to(device)
            x2 = x2.to(device)
            y = y.to(device)
            optimizer.zero_grad()
            outputs = siamese_net(x1.double(), x2.double())
            outputs = torch.squeeze(outputs)
            loss = criterion(outputs, y)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if i % print_every == print_every-1:
                print('[%d, %5d] loss: %.6f' %
                  (epoch + 1, i + 1, running_loss / print_every))
                running_loss = 0.0
        siamese_net.eval()
        train_loss = get_siamese_net_loss(siamese_net, trainloader, val_batches)
        test_loss = get_siamese_net_loss(siamese_net, testloader, val_batches)
        siamese_net.train()
        train_losses.append(train_loss)
        test_losses.append(test_loss)
       return train_losses, test_losses

# ## Partition Dataset
def partition_dataset(data, labels):
    unique_labels = np.unique(labels)
    final_data = [None] * len(unique_labels)
    n_samples = len(labels)
    for i in range(n_samples):
        curr_class = labels[i]
        if final_data[curr_class]:
            final_data[curr_class].append(data[i])
        else:
            final_data[curr_class] = [data[i]]
    return list(map(lambda x: np.asarray(x), final_data))

# ## Plotting Losses
def plot_losses(train_losses, test_losses, title, xlabel, ylabel, start_idx=1,
interval_length=1, figure_size=(14,14)):
    train_losses_aggregated = []
    test_losses_aggregated = []
    n_epochs = len(train_losses)
    start_idx -= 1
        i = start_idx
    while i < n_epochs:
        curr_train_sum = sum(train_losses[i : min(n_epochs, i + interval_length)])
        curr_test_sum = sum(test_losses[i : min(n_epochs, i + interval_length)])
        train_losses_aggregated.append(curr_train_sum / interval_length)
        test_losses_aggregated.append(curr_test_sum / interval_length)
        i += interval_length
    n = len(train_losses_aggregated)
    x = list(map(lambda x: (x * interval_length) + start_idx, np.arange(1, n+1)))
    plt.figure(figsize=figure_size)
    plt.plot(x, train_losses_aggregated, marker='o', linestyle='dashed')
    plt.plot(x, test_losses_aggregated, marker='o', linestyle='dashed')
    plt.legend(['Training Loss', 'Testing Loss'])
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid()
    plt.xticks(x)
    plt.savefig(title + '.png')

# ## Saving and Loading Models
def save_state_dict(model, filename):
    torch.save(model.state_dict(), filename)
def load_state_dict(model, state_dict_path):
    state_dict = torch.load(state_dict_path)
    model.load_state_dict(state_dict)
    model.double()
    model.to(device)

# # Training
# ## Autoencoders
# ### Label Encoding
# #### Autoencoder 1
```

```
autoencoder_label_1 = Autoencoder(in_features=40, out_features=20)
print(autoencoder_label_1)
train_losses_encoder_label_1, test_losses_encoder_label_1 = train_encoder(
    autoencoder_label_1,
    trainloader=train_label_loader,
    testloader=test_label_loader,
    n_epochs=30)
plot_losses(
    train_losses_encoder_label_1,
    test_losses_encoder_label_1,
    # start_idx=11,
    title='autoencoder_label_1',
    xlabel='Epochs',
    ylabel='Losses',
    interval_length=1,
    figure_size=(14,8))
save_state_dict(autoencoder_label_1, 'autoencoder_label_1.pth')

# #### Autoencoder 2
autoencoder_label_2 = Autoencoder(in_features=20, out_features=10)
print(autoencoder_label_2)
train_losses_encoder_label_2, test_losses_encoder_label_2 = train_encoder(
    autoencoder_label_2,
    autoencoder1=autoencoder_label_1,
    trainloader=train_label_loader,
    testloader=test_label_loader,
    n_epochs=30)
plot_losses(
    train_losses_encoder_label_2,
    test_losses_encoder_label_2,
    # start_idx=11,
    title='autoencoder_label_2',
    xlabel='Epochs',
    ylabel='Losses',
    interval_length=1,
    figure_size=(14,8))
save_state_dict(autoencoder_label_2, 'autoencoder_label_2.pth')

# ### Loading Autoencoder 1
autoencoder_label_1 = Autoencoder(in_features=40, out_features=20)
load_state_dict(autoencoder_label_1, os.path.join(LOAD_DIR,
'autoencoder_label_1.pth'))

# ### Loading Autoencoder 2
```

```
autoencoder_label_2 = Autoencoder(in_features=20, out_features=10)
load_state_dict(autoencoder_label_2, os.path.join(LOAD_DIR,
'autoencoder_label_2.pth'))

# ### AnomalyDetector
anomaly_detector = AnomalyDetector()
print(anomaly_detector)
train_losses_anomaly_detector, test_losses_anomaly_detector = train_network(
    anomaly_detector,
    trainloader=train_binary_dl,
    testloader=test_binary_dl,
    n_epochs=50,
    print_every=100)
plot_losses(
    train_losses_network_label,
    test_losses_network_label,
    # start_idx=37,
    title='network_label_double_encoder',
    xlabel='Epochs',
    ylabel='Losses',
    interval_length=1,
    figure_size=(14,8))

# ## Data Distributions
# ### PCA
pca = PCA(n_components=3)
pca.fit(train_features_np)
train_features_3dims = pca.transform(train_features_np)
test_features_3dims = pca.transform(test_features_np)

# ### Groups
groups_multiclass = [
    'Normal',
    'DoS',
    'Probe',
    'R2L',
    'U2R']
groups_binary = [
    'Normal',
    'Malicious']
groups_malicious_classes = [
    'DoS',
    'Probe',
    'R2L',
    'U2R']

# ### Plot Points
def plot_points(
        features,
        labels,
        title,
        groups,
        class_idx=None,
        one_hot_labels=True,
        figure_size=(8,8)    ):
    fig = plt.figure(figsize=figure_size)
    ax = fig.add_subplot(111, projection='3d')
    if one_hot_labels:
        labels_indices = np.asarray(list(map(lambda x: np.argmax(x), labels)))
    else:
        labels_indices = labels
    data = []
    if class_idx:
        i = class_idx
        curr_data_indices = np.where(labels_indices == i)[0]
        curr_data = features[curr_data_indices]
        data.append(curr_data)
    else:
        for i in range(len(groups)):
            curr_data_indices = np.where(labels_indices == i)[0]
            curr_data = features[curr_data_indices]
            data.append(curr_data)
    all_colors = (
        'grey',
        'orange',
        'cyan',
        'green',
        'maroon',
        'chocolate',
        'lightgreen',
        'dodgerblue',
        'darkslategray',
        'lightseagreen',
        'mediumspringgreen',
        'rebeccapurple',
        'hotpink',
        'indigo',
        'midnightblue',
        'gold',
        'black',
        'crimson',
        'darkkhaki',
        'aqua' )
    colors = sample(all_colors, len(groups))
    for features, color, group in zip(data, colors, groups):
        x = list(map(lambda x: x[0], features))
        y = list(map(lambda x: x[1], features))
        z = list(map(lambda x: x[2], features))
        ax.scatter(
            x, y, z,
            c=color,
            label=group,)
    plt.legend()
    plt.title(
        title,
        fontsize=15,
        loc='left')
    plt.savefig(title + '.png')

# ### Training Data Plots
# #### Binary Class Distribution
plot_points(
    train_features_3dims,
    train_labels_binary_np,
    groups=groups_binary,
    title='Binary Class Distribution',
    figure_size=(14,14))

# #### Multi Class Distribution
plot_points(
    train_features_3dims,
    train_labels_multiclass_np,
    groups=groups_multiclass,
    title='Multi Class Distribution',
    figure_size=(14,14))

# #### Malicious Data Point Distributions
# ##### DoS
plot_points(
    train_features_3dims,
    train_labels_multiclass_np,
    class_idx=1,
    groups=['DoS'],
    title='DoS',
    figure_size=(14,14))

# ##### Probe
plot_points(
    train_features_3dims,
    train_labels_multiclass_np,
    class_idx=2,
    groups=['Probe'],
    title='Probe',
    figure_size=(14,14))

# ##### R2L
plot_points(
    train_features_3dims,
    train_labels_multiclass_np,
    class_idx=3,
    groups=['R2L'],
    title='R2L',
    figure_size=(14,14))

# ##### U2R
plot_points(
    train_features_3dims,
    train_labels_multiclass_np,
    class_idx=4,
    groups=['U2R'],
    title='U2R',
    figure_size=(14,14))

# ## Anomaly Detection
# ### Utilities
# #### Plot Confusion Matrix
def plot_confusion_matrix(
        cm, classes,
        normalize=False,
        title='Confusion Matrix',
        cmap=plt.cm.Blues,
        print_cm=False,
        save=False):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    if print_cm:
        print(cm)
       plt.imshow(cm, interpolation='nearest', cmap=cmap)
    # plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True')
    plt.xlabel('Predicted')
    # plt.xticks([])
    # plt.yticks([])
    plt.grid(False)
    if save:
        plt.savefig(title + '.png')

# #### One Hot to Label Encoding
def to_label(arr):
    return np.argmax(arr, axis=1)

# #### Get Predictions
def get_predictions(estimator, features, labels):
    return to_label(labels), estimator.predict(features),
estimator.predict_proba(features)

# #### Get Scores
def get_scores(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1}
```

```
# #### Get Adjusted Classes
def get_adjusted_classes(y_probs, thresh=0.5):
    return [int(y_prob >= thresh) for y_prob in y_probs]

# #### Plot ROC Curve
def plot_roc_curve(fpr, tpr, title, label=None, figure_size=(8,8)):
    plt.figure(figsize=figure_size)
    plt.title('ROC Curve')
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.005, 1, 0, 1.005])
    plt.xticks(np.arange(0,1, 0.05), rotation=90)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate (Recall)")
    plt.legend(loc='best')

# #### Plot Precision Vs Recall
def plot_precision_recall_curve(p, r, y_scores, y_test, thresholds, title, t=0.5,
figure_size=(8,8)):
    y_pred_adj = get_adjusted_classes(y_scores, t)
    plt.figure(figsize=figure_size)
    plt.title(title)
    plt.step(r, p, color='b', alpha=0.2,
             where='post')
    plt.fill_between(r, p, step='post', alpha=0.2,
                     color='b')
    plt.ylim([0.5, 1.01]);
    plt.xlim([0.5, 1.01]);
    plt.xlabel('Recall');
    plt.ylabel('Precision');
    close_default_clf = np.argmin(np.abs(thresholds - t))
    plt.plot(r[close_default_clf] - 0.01, p[close_default_clf], '.', c='k',
            markersize=15)
    plt.text(r[close_default_clf], p[close_default_clf], 'Current Threshold')
    plt.savefig(title + '.png')

# #### Plot Precision and Recall Vs Threshold
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds, title,
figure_size=(8,8)):
    idx = np.argmin(abs(precisions - recalls))
    plt.figure(figsize=figure_size)
    plt.title(title)
    plt.plot(thresholds, precisions[:-1], "k--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "k-", label="Recall")
    plt.plot(thresholds[idx], precisions[idx], 'ro')
    plt.text(thresholds[idx] + 0.01, precisions[idx], f'Threshold = {thresholds[idx]}')
    plt.ylabel("Score")
    plt.xlabel("Decision Threshold")
    plt.legend(loc='best')
    return thresholds[idx]

# ### Clustering Normal Points
plot_points(
    train_features_3dims,
    train_labels_multiclass_np,
    class_idx=0,
    groups=['Normal'],
    title='Normal',
    figure_size=(14,14))
train_labels = np.asarray(list(map(lambda x: np.argmax(x), train_labels_binary_np)))
train_indices_normal = np.where(train_labels == 0)[0]
train_features_normal = train_features_np[train_indices_normal]
test_labels = np.asarray(list(map(lambda x: np.argmax(x), test_labels_binary_np)))
test_indices_normal = np.where(test_labels == 0)[0]
test_features_normal = test_features_np[test_indices_normal]
train_features_normal.shape
test_features_normal.shape

# #### K-Means
# ##### Finding Optimal Clusters using K-Elbow Visualizer
model = cluster.KMeans(random_state=42)
visualizer = KElbowVisualizer(
    model,
    k=(2,11),
    metric='calinski_harabasz')
visualizer.fit(train_features_normal)

# ##### Fitting
N_CLUSTERS = 4
kmeans_normal = cluster.KMeans(n_clusters=N_CLUSTERS, random_state=42)
kmeans_normal.fit(train_features_normal)
kmeans_normal.inertia_
train_labels_normal = kmeans_normal.predict(train_features_normal)
plot_points(
    features=pca.transform(train_features_normal),
    labels=train_labels_normal,
    groups=['Cluster ' + str(i+1) for i in range(N_CLUSTERS)],
    one_hot_labels=False,
    title='K-Means Clustering on Normal Data Points - Training',
    figure_size=(14,14))
test_labels_normal = kmeans_normal.predict(test_features_normal)
plot_points(
    features=pca.transform(test_features_normal),
    labels=test_labels_normal,
    groups=['Cluster ' + str(i+1) for i in range(N_CLUSTERS)],
    one_hot_labels=False,
    title='K-Means Clustering on Normal Data Points - Testing',
    figure_size=(14,14))
joblib.dump(kmeans_normal, 'kmeans_normal.joblib')

# ##### Partitioning Dataset
partitioned_training_data_normal = partition_dataset(train_features_normal,
train_labels_normal)
for data in partitioned_training_data_normal:
    print(data.shape)
partitioned_testing_data_normal = partition_dataset(test_features_normal,
test_labels_normal)
for data in partitioned_testing_data_normal:
    print(data.shape)

# ##### SiameseNet
# ###### Generating Data for NSLKDDSiameseNetDataset Instance
TRAIN_MULT_FACTOR = 100000
TEST_MULT_FACTOR = 20000
indices = []
for i in range(N_CLUSTERS):
    for j in range(i, N_CLUSTERS):
        indices.append((i,j))
train_indices_final = indices * TRAIN_MULT_FACTOR
shuffle(train_indices_final)
test_indices_final = indices * TEST_MULT_FACTOR
shuffle(test_indices_final)
len(train_indices_final), len(test_indices_final)
train_labels = list(map(lambda x: int(x[0] == x[1]), train_indices_final))
test_labels = list(map(lambda x: int(x[0] == x[1]), test_indices_final))
len(train_labels), len(test_labels)

# ###### NSLKDDSiameseNetDataset Instance
train_data_siamese_net = NSLKDDSiameseNetDataSet(
    partitioned_training_data_normal,
    train_indices_final,
    np.asarray(train_labels),
    train_features)
trainloader_siamese_net = DataLoader(
    train_data_siamese_net,
    batch_size=256,
    shuffle=True)
test_data_siamese_net = NSLKDDSiameseNetDataSet(
    partitioned_testing_data_normal,
    test_indices_final,
    np.asarray(test_labels),
    train_features)
testloader_siamese_net = DataLoader(
    test_data_siamese_net,
    batch_size=256,
    shuffle=True)

# ###### Training Signatures for Each Cluster
siamese_net = SiameseNet()
print(siamese_net)
train_losses_siamese_net, test_losses_siamese_net = train_siamese_net(
    siamese_net,
    trainloader=trainloader_siamese_net,
    testloader=testloader_siamese_net,
    n_epochs=50)

# #### Z-Score
curr_cluster = partitioned_training_data_normal[0]
curr_cluster.shape
z_scores = zscore(curr_cluster)
z_scores.shape
sns.kdeplot(z_scores[:,5])

# ### Random Forest Classifier
# #### Grid Search
# ##### Recall
rfc = RandomForestClassifier(
    class_weight='balanced',
    max_depth=58,
    min_samples_leaf=2    )
param_grid = {
    'n_estimators': np.arange(100, 120),}
rfc_grid_search_recall = GridSearchCV(
    estimator=rfc,
    param_grid=param_grid,
    scoring=make_scorer(recall_score),
    verbose=2,
    cv=5,
    n_jobs=5)
rfc_grid_search_recall.fit(train_features_np, to_label(train_labels_binary_np))
rfc_grid_search_recall.best_score_
rfc_grid_search_recall.best_params_
joblib.dump(rfc_grid_search_recall, 'rfc_grid_search_recall.joblib')

# #### Model
rfc_grid_search_recall = joblib.load(os.path.join(MODELS_DIR,
'rfc_grid_search_recall.joblib'))
rfc = rfc_grid_search_recall.best_estimator_
y_true, y_pred, y_probs = get_predictions(
    rfc,
    test_features_np,
    test_labels_binary_np)
cm = confusion_matrix(
    y_pred=y_pred,
    y_true=y_true)
plot_confusion_matrix(
    cm,
    classes=[0,1],
    title='Random Forest Classifier with Optimized Recall',
    save=True,
    normalize=True,
    cmap=plt.cm.Greys)
p, r, thresholds = precision_recall_curve(y_true, y_probs[:,1])
best_threshold_rfc = plot_precision_recall_vs_threshold(
    p, r, thresholds,
    title='Precision and Recall Scores as a Function of the Decision Threshold',)
plot_precision_recall_curve(
    p, r, y_probs[:,1], y_true, thresholds,
    title='Precision Recall Curve',
    t=best_threshold_rfc,)
fpr, tpr, auc_thresholds = roc_curve(y_true, y_probs[:,1])
auc(fpr, tpr)
plot_roc_curve(
    fpr, tpr,
    title='ROC Curve',
    label='Recall Optimized')
y_pred_adjusted = get_adjusted_classes(y_probs[:,1], thresh=best_threshold_rfc)
cm = confusion_matrix(
    y_pred=y_pred_adjusted,
    y_true=y_true)
```

```
plot_confusion_matrix(
    cm,
    classes=[0,1],
    title='Random Forest Classifier with Optimized Recall',
    save=True,
    normalize=True,
    cmap=plt.cm.Greys)

# ### AdaBoost Classifier
# #### Grid Search
# ##### Recall
adaboost_clf = AdaBoostClassifier()
param_grid = {
    'n_estimators': np.arange(50, 70)}
adaboost_clf_grid_search_recall = GridSearchCV(
    estimator=adaboost_clf,
    param_grid=param_grid,
    scoring=make_scorer(recall_score),
    verbose=2,
    cv=5,
    n_jobs=5)
adaboost_clf_grid_search_recall.fit(train_features_np,
to_label(train_labels_binary_np))
joblib.dump(adaboost_clf_grid_search_recall,
'adaboost_clf_grid_search_recall.joblib')

# #### Model
adaboost_clf_grid_search_recall = joblib.load(os.path.join(MODELS_DIR,
'adaboost_clf_grid_search_recall.joblib'))
adaboost_clf = adaboost_clf_grid_search_recall.best_estimator_
y_true, y_pred, y_probs = get_predictions(
    adaboost_clf,
    test_features_np,
    test_labels_binary_np)
cm = confusion_matrix(
    y_pred=y_pred,
    y_true=y_true)
plot_confusion_matrix(
    cm,
    classes=[0,1],
    title='AdaBoost Classifier with Optimized Recall',
    save=True,
    normalize=True,
    cmap=plt.cm.Greys)
```

```
p, r, thresholds = precision_recall_curve(y_true, y_probs[:,1])
best_threshold = plot_precision_recall_vs_threshold(
    p, r, thresholds,
    title='Precision and Recall Scores as a Function of the Decision Threshold',)
plot_precision_recall_curve(
    p, r, y_probs[:,1], y_true, thresholds,
    title='Precision Recall Curve',
    t=best_threshold,)
fpr, tpr, auc_thresholds = roc_curve(y_true, y_probs[:,1])
auc(fpr, tpr)
plot_roc_curve(
    fpr, tpr,
    title='ROC Curve',
    label='Recall Optimized')
y_pred_adjusted = get_adjusted_classes(y_probs[:,1], thresh=best_threshold)
cm = confusion_matrix(
    y_pred=y_pred_adjusted,
    y_true=y_true)
plot_confusion_matrix(
    cm,
    classes=[0,1],
    title='AdaBoost Classifier with Optimized Recall',
    save=True,
    normalize=True,
    cmap=plt.cm.Greys)

# ## MultiClassClassifier
# #### Utilities
def print_counts(labels):
    labels = np.argmax(labels, axis=1)
    for i in range(5):
        print(len(np.where(labels == i)[0]))
print_counts(test_labels_multiclass_np)

# #### Model
over_sampler = SVMSMOTE()
x_train, y_train = over_sampler.fit_resample(
    train_features_np, np.argmax(
        train_labels_multiclass_np,
        axis=1))
x_train, y_train = shuffle(x_train, y_train)
Counter(y_train)
x_test, y_test = test_features_np, test_labels_multiclass_np
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
lstm_output_size = 70
cnn = Sequential()
cnn.add(Convolution1D(64, 3,
border_mode="same",activation="relu",input_shape=(67, 1)))
cnn.add(Convolution1D(64, 3, border_mode="same", activation="relu"))
cnn.add(MaxPooling1D(pool_length=(2)))
cnn.add(Convolution1D(128, 3, border_mode="same", activation="relu"))
cnn.add(Convolution1D(128, 3, border_mode="same", activation="relu"))
cnn.add(MaxPooling1D(pool_length=(2)))
cnn.add(LSTM(lstm_output_size))
cnn.add(Dropout(0.1))
cnn.add(Dense(5, activation="softmax"))
cnn.summary()
cnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history_1 = cnn.fit(
    np.expand_dims(x_train, axis=2), to_categorical(
        y_train,
        num_classes=5),
    batch_size=128,
    epochs=5,
    validation_data=(
        np.expand_dims(x_test, axis=2), y_test))
all_losses = []
for loss in history_1.history['loss']:
    all_losses.append(loss)
all_acc = []
for acc in history_1.history['accuracy']:
    all_acc.append(acc)
len(all_losses), len(all_acc)
plt.figure(figsize=(8,8))
plt.plot(
    np.linspace(1, 10, 10) * 10,
    all_losses,
    'k--',
    marker='D')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.savefig('Loss.png')
plt.figure(figsize=(8,8))
plt.plot(
    np.linspace(1, 10, 10) * 10,
    all_acc,
    'k--',
    marker='D')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.savefig('Accuracy.png')
cnn.save('conv_model.h5')
import pickle as pk
with open('history.pkl', 'wb') as f:
   pk.dump(history, f)
with open('history_1.pkl', 'wb') as f:
   pk.dump(history_1, f)
plot_model(cnn, show_layer_names=False, show_shapes=True, t)
y_pred_rfc = rfc.predict(test_features_np)
normal_samples_indices = np.where(y_pred_rfc == 0)[0]
x_train_normal = test_features_np[normal_samples_indices]
y_train_multiclass_normal = test_labels_multiclass_np[normal_samples_indices]
y_train_multiclass_normal
x_train_normal.shape, y_train_multiclass_normal.shape
malicious_samples_indices = np.where(y_pred_rfc == 1)[0]
x_train_malicious = test_features_np[malicious_samples_indices]
y_train_multiclass_malicious =
test_labels_multiclass_np[malicious_samples_indices]
x_train_malicious.shape, y_train_multiclass_malicious.shape
y_true, y_pred, _ = get_predictions(
   cnn,
   np.expand_dims(x_train_malicious, axis=2),
   y_train_multiclass_malicious)
y_true_with_zeros = np.concatenate((y_true, np.argmax(y_train_multiclass_normal,
axis=1)))
y_true_with_zeros.shape
pred = np.argmax(y_pred, axis=1)
pred_with_zeros = np.concatenate((pred, np.asarray([0] * 13920)))
pred_with_zeros.shape
pred_with_zeros
cm = confusion_matrix(
   y_pred=pred_with_zeros,
   y_true=y_true_with_zeros)
plot_confusion_matrix(
   cm,
   classes=['Normal', 'DoS', 'Probe', 'U2R', 'R2L'],
   title='Final Confusion',
   save=True,
   normalize=True,
   cmap=plt.cm.Greys)
```

REFERENCES

[1] A. Divekar, M. Parekh, V. Savla, R. Mishra and M Shirole, Benchmarking datasets for Anomaly-based Intrusion Detection: KDD CUP 99 alternatives, *Proceedings on 2018 IEEE 3rd International Conference on Computing, Communication and Security, ICCCS 2018,* 2018

[2] M. Roesch, Snort – Lightweight Intrusion Detection for Networks, *Proceedings of LISA '99: 13th Systems Administration Conference,* Seattle, Washington, USA, 1999.

[3] G. Wang, J. Hao, J. Ma, L. Huang, A New Approach to Intrusion Detecting using Artificial Neural Networks and Fuzzy Clustering, *Expert Systems with Application,* vol. 37, no. 9, pp. 6225-6232, 2010

[4] S. Shraddha, Intrusion Detection using Fuzzy Clustering and Artificial Neural Network, *Advances in Neural Networks, Fuzzy Systems and Artificial Intelligence,* pp. 209-217

[5] K. Peng, V. C. M. Leung and Q. Huang, Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection System Over Big Data, *IEEE Access,* vol. 6, no. 1, pp. 11897-11906, 2018

[6] S. Potluri and C. Diedrich, Accelerated Deep Neural Networks for Enhanced Intrusion Detection System, *IEEE International Conference on Emerging Technologies and Factory Automation, EFTA,* 2016

[7] C. Yin, Y. Zhu, J. Fei and X. He, A Deep Learning Approach for Intrusion Detection using Recurrent Neural Networks, *IEEE Access,* vol. 5, no. 1, pp. 21954-21961, 2017

[8] J. Kim, N. Shin, S. Y. Jo and S. H. Kim, Method of Intrusion Detection Using Deep Neural Network, in *IEEE*, 2017

[9] Y. LeCun, Y. Bengio and G. Hinton, Deep Learning, *Nature*, vol. 521, no. 1, pp. 436-444, 2015

[10] J. Kim, J. Kim, H. L. T. Thu and H. Kim, "Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection," *IEEE,* 2016

[11] Z. Chiba, A. Noreddine, K. Moussaid, A. E. Omri, M. Rida, Intelligent and Improved Self-Adaptive Anomaly Based Detection System for Networks, *International Journal of Communication Networks and Information Security,* vol. 11, no. 2, pp. 312-330, 2019

[12] P. Manandhar and Z. Aung, Intrusion Detection Based on Outlier Detection Method, *International conference on Intelligent Systems, Data Mining and Information Technology,* Bangkok, Thailand, 2014

[13] J. Vacca, Computer and Information Security Handbook, *Elsevier*, 2009.

[14] M. E. Tipping and C. M. Bishop, Mixtures of Probabilistic Principal Component Analysers, *Journal of the Royal Statistical Society*, pp. 443-482, 1999

[15] K. Rai, M. S. Devi and A. Guleria, Decision Tree Based Algorithm for Intrusion Detection, *Advanced Networking and Applications*, vol. 7, no. 4, pp. 2828-2834, 2016

[16] D. P. Gaikwad and R. C. Thool, Intrusion Detection System using Bagging Ensemble Method of Machine Learning, *2015 International Conference on Computing Communication Control and Automation*, Pune, India, 2015

[17] J. Zhang, M. Zulkernine and A. Haque, Random Forests Based Network Intrusion Detection Systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 38, no. 5, pp. 649-659, 2008

LIST OF PUBLICATIONS

1. R. Raj, S. Gupta, M. Lohia and H. Taneja, "Dual Stage Network Intrusion Detection System Through Feature Reduction," in *Mathematical Models and Computational Techniques in Science and Engineering*, London, 2020.

2. R. Raj, S. Gupta, M. Lohia and H. Taneja, "Building a NIDS Using a Two-Stage Classifier and Feature Reduction through Statistical Methods," *World Scientific and Engineering Academy and Society (WSEAS) transactions on Systems and Control,* vol. 15, pp. 102-112, 2020.