
**Arab American University – Palestine**

**Faculty of Graduate Studies**

# Detecting Complex Intrusion Attempts Using Hybrid Machine Learning Techniques

By

**Nizar S. Shanaah**

Supervisor

**Dr. Mustafa Abusalah**

**This thesis was submitted in partial fulfillment of the requirements for the Master`s degree in**

**Data Science and Business Analytics**

**August 2021**

# Detecting Complex Intrusion Attempts Using Hybrid Machine Learning Techniques

By

**Nizar S. Shanaah**

This thesis was defended successfully on 14/8/2021 and approved by:

| Committee members | Signature |
|---|---|
| 1. **Mustafa Abusalah, Supervisor** |  |
| 2. **Mohammed Maree, Internal Examiner** |  |
| 3. **Ayser Armiti, External Examiner** |  |

# Declaration

I declare that this thesis has been composed solely by me and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

Name: Nizar Shanaah

Signature: 

Date: Jan 11, 2022

# Abstract

Organizations are in a constant race to secure their services and infrastructure from ever-evolving information security threats. The primary security control of the arsenal is the Intrusion Detection System (IDS), which can automatically detect attacks and intrusion attempts. For the IDS to be effective, it needs to detect all kinds of attacks while not disturbing legitimate traffic by erroneously classifying them as attacks and affecting normal operations. Additionally, IDS needs to detect previously unknown attacks that do not exist in its knowledge base. This capability is traditionally achieved by anomaly detection based on trends and baselines; an approach that is prone to high false-positive rates. This dissertation will explore the most appropriate machine learning algorithms and techniques, specifically hybrid machine learning. This hybrid approach will combine unsupervised and supervised machine learning to detect previously unknown attacks while minimizing false positives by analyzing events generated by different connected systems and devices.

Keywords: Intrusion Detection System, Anomaly, Machine Learning, Unsupervised, Performance, Evaluation, Big Data

# Contents

# Table of Tables

# Table of Figures

# Chapter 1

# Introduction

## 1.1. Intrusion detection systems

Intrusion detection is the process of detecting misuse or abuse of an organization's assets through the monitoring and analysis of events generated by various connected systems and devices. Organizations rely on Intrusion Detection Systems (IDSs) to detect such intrusion and misuse attempts (Friedberg, et al., 2015).

Traditionally IDS systems operate in two modes, signature-based and anomaly detection. The signature-based relies on a list of previously known attacks for detection. The anomaly detection works by looking for events that fall out of normal system operation and the established normal baseline to detect misuse (Syarif, et al., 2012) (Repalle & Kolluru, 2017).

The signature-based primary advantage is its ability to accurately identify well-known attacks because it relies on previously programmed indicators like hashes, traffic content, and known byte sequences. However, it falls short in detecting new attacks as it does not have the signatures of those attacks in its database. In contrast to signature-based detection, anomaly detection systems do not require a signature database. It learns normal behavior by establishing a normalized baseline to be used as a benchmark to detect abnormal behavior. Hence, it does a better job of detecting new attacks but suffers from a high false-positive detection rate (Amoli, et al., 2016).

## 1.2. Supervised and Unsupervised Learning

The main difference between supervised and unsupervised learning is the existence of labels in the training data set. Supervised machine learning learns from labeled training data for prediction and classification. There are two main categories for

supervised learning algorithms: classification and regression, in addition to logistic regression, decision tree, random forest, naive bayes classifiers, K-nearest neighbors, and support vector machine. The main advantage of supervised learning is helping optimize the performance criteria with the help of experience. Also, it solves various types of real-world computation problems. Conversely, supervised learning has some disadvantages when classifying big data as it requires a lot of computation time to train the model. Furthermore, it is costly and challenging to obtain labeled data for emerging real-world problems.

Meanwhile, unsupervised machine learning recognizes patterns without the need to a target variable. All the variables used in the analysis are included as inputs, and the algorithm needs to find the hidden structure by itself. Hence, the techniques are suitable for clustering and association mining application domains. Here the challenge will be to group unsorted information based on similarities, patterns, and differences. There are main categories of unsupervised learning: clustering and association. The clustering algorithms identify inherent groupings within the unlabeled data and then assigns a grouping label to each data value. However, association algorithms aim to identify rules that accurately represent relationships between attributes, and it discovers rules that describe large portions of data. Clustering contains hierarchical clustering and k-means.

However, unsupervised learning is computationally complex, and results are less accurate than supervised learning algorithms. (Berry, et al., 2019)

### 1.3. The Hybrid Approach

The hybrid approach involves combining supervised and unsupervised learning to evade overfitting and reduce the high computational costs in high-dimensional big data set. The hybrid approach involves combining supervised and unsupervised learning to evade

overfitting and reduce the high computational costs in high-dimensional big data set. In our approach, we will use unsupervised machine learning to create the clusters. After that, the label feature will be introduced back to create a new training dataset for supervised machine learning. Moreover, a hard voting technique will be used on the results obtained from different hybrid combinations to judge whether a request is benign or an attack.

### 1.4. Problem Statement

Malicious entities are increasingly evolving in their techniques to evade detection by exploiting novel vulnerabilities that the IDS do not yet have their signatures, also known as zero-day attacks (Singh, et al., 2019). While the anomaly detection approach can help detect previously unknown attacks, it suffers from high false-positive rates (Grill, et al., 2017). Such threats require algorithms and techniques to augment the capabilities of IDS to detect such unseen attack attempts while not disrupting legitimate operational processes.

### 1.5. Research Question

Most of the research for anomaly detection utilized various datasets, ranging from the KDD Cup dataset collected in 1999 to the latest CSE-CIC-IDS2018. (Kumar, et al., 2020) (Leevy & Khoshgoftaar, 2020) (Thakkar & Lohiya, 2020)

Using older datasets has a significant drawback; most systems and device makers updated their systems and products with the latest security patches, and older attacks are no longer a concern. On the other hand, a new class of services and applications emerged, introducing new security vulnerabilities. (Leevy & Khoshgoftaar, 2020)

This thesis aims to determine the best approach to find previously unknown attacks while reducing false positives by using a hybrid approach that combines different

supervised and unsupervised machine learning techniques. Moreover, we will use real-world and recently generated security events made available by the Canadian Establishment for Cybersecurity (CIC), the CSE-CIC-IDS2018 dataset captured in 2018.

### 1.6. Contribution

This research will suggest whether combining different unsupervised and supervised machine learning methods and strategies will improve network access security using the real-world and recent CSE-CIC-IDS2018 dataset with 16 million records. We also utilize the hold-out method to create training, validation, and testing sets.

We will use hierarchal clustering for feature selection and apply different unsupervised machine learning algorithms to create clusters. Then the cluster label feature will be introduced back to the training dataset for the supervised machine learning algorithms.

Furthermore, we will be developing a hard-voting method to obtain better results from the different hybrid approaches.

Finally, we will explore the deep learning models performance in the same dataset utilizing auto encoders; complete autoencoder, stacked autoencoder and denoising autoencoders.

# Chapter 2

# Literature Review

In this chapter, we will review the research on the problem of detecting complex intrusion attempts. There are different approaches to tackle this challenge using different machine learning techniques. We will also highlight the primary publicly available datasets researchers used to evaluate the proposed approaches.

The research in the intrusion detection field is based on publicly available datasets discussed thoroughly in this chapter.

(Laskov, et al., 2005) presented an experimental evaluation framework for supervised and unsupervised learning methods for intrusion detection applications. They used the KDD'99 [1] dataset to demonstrate that the supervised learning methods significantly outperform the unsupervised when dealing with known attacks. Consequently, they highlighted those unsupervised methods are better suited to handle unknown attacks as supervised methods' performance drops significantly.

(Zhang & Zulkernine, 2006) highlight the performance issues with supervised algorithms in the ever-changing real-world network environments as it is difficult to obtain attack-free training data. Also, they highlight high false-positive rates when normal traffic patterns change. They suggest a framework based on an unsupervised random forest algorithm to overcome such limitations using the KDD'99 dataset.

(Gogoi, et al., 2010) performed a literature analysis of various existing machine learning approaches and the ability to detect attacks in network traffic data using unsupervised and supervised learning approaches using the KDD'99 dataset. Their

[1] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

analysis suggests that unsupervised learning has a higher detection rate than supervised learning; however, the results of their analysis suggest that they are susceptible to a high false-positive rate due to the changing network environment or services, patterns of normal traffic.

(Kumar, et al., 2020) investigate the application of a mean shift algorithm to detect an attack on a network using the KDD'99 dataset. Their experimental result demonstrates an accuracy of 81.2% and a detection rate of 79.1%. Their research provides an initial application of a mean shift unsupervised algorithm on a network traffic dataset to detect an attack.

(Singh & Tiwari, 2014) review the approaches used for detecting anomalies using the KDD'99 dataset, highlighting their advantages and disadvantages, concluding that most of them, in general, are less efficient in reducing false alarms and take extensive training time.

(Jamadar, 2018) proposes a model for building the network intrusion detection system for anomaly-based detection using decision tree. The precision of the proposed system indicates that the True-Positive-Rate (TPR) is 99.9%, and the False-Positive-Rate (FPR) is 0.1%. The researcher discussed the limitations of the outdated KDD'99 dataset. Since then, a lot has changed with the introduction of newer technologies and protocols, and many of the systems and their vulnerabilities are no longer used and not being actively attacked.

(Blanco, et al., 2019) propose the use of multiple simple Gaussian Mixture Models (GMM) to model normal behavior along with an asymmetric voting scheme that aggregates individual anomaly detectors to decide whether the sample is normal or not. They also compare the proposed methods, other state-of-the-art algorithms like K-Means,

SVMs, Decision Tree, and MLP. They achieved a Fl-score over 0.9 using the outdated NSL-KDD dataset.

(Verkerken, et al., 2020) proposed four unsupervised techniques: principal component analysis, isolation forest, one-class support vector machine, and autoencoder on the CIC-IDS-2017 dataset. The training dataset used data collected on Monday to generate a sample of 50,000 benign only for training and around 200,000 benign to validation and testing combined with around 500,000 attacks divided on validation and testing; attacks were not included in the training phase.

(Thakkar & Lohiya, 2020) reviewed advancement in intrusion detection datasets developed in the field of intrusion detection systems. Their review revealed the shortcomings of KDD'99 and NSL-KDD datasets and the need to update the underlying dataset to identify recent attacks in realistic network scenarios. The authors introduced the use of attacks CIC-IDS-2017 and CSE-CICIDS-2018 datasets.

(Leevy & Khoshgoftaar, 2020) surveyed and analyzed the use of intrusion detection models and IDS datasets, starting with ISCX2012 and its limitations and lack of realistic and recent nature. They have also discussed the CIC-IDS2018 dataset, which was prepared from a much larger network of simulated client targets and attack machines. They also determined that the best performance scores for each study were unexpectedly high overall and attributed it to overfitting. They also found that information on the data cleaning of CSECIC-IDS2018 was insufficient across the board. They also suggested significant research gaps in large data processing frameworks, concept drift, and transfer learning.

(Farhan, et al., 2020) discussed the large data size, Higher dimensionality, and Data preprocessing challenges of the real-world CSE-CIC-IDS2018 dataset. The use of

fully connected dense deep neural network (DNN) was proposed, achieving 0.90 accuracy of detection. In addition, the researchers recommended utilizing a proper feature selection method to increase accuracy, detection rate, decrease False Alarm Report (FAR), and minimize computing time. Hyperparameter tuning is also recommended for further efficiency.

A hybrid approach that combines unsupervised and supervised techniques was proposed (Soheily-Khah, et al., 2018). Their experimental work showed high accuracy over 99% using k-means and random forest; those results are based on the ISCX2012 dataset containing around two million records. K-Means was effectively used as an under-sampling technique on the ISCX dataset. DNS flows were reduced from 309K to 55K, while HTTPWeb flows were reduced from 881K to 367K. The ISCX2012 doesn't include HTTPS requests which account for most web traffic in modern applications and environments.

# Chapter 3

# Methodology

## 3.1. The Dataset

As discussed in the literature review, the most used intrusion detection evaluation dataset is the KDD99 dataset generated from the DARPA 98 dataset. Later, the NSL-KDD dataset was proposed to address the limitations of the KDD99, but the two versions are outdated and do not reflect modern-day attacks. Many of the attacks in the dataset are irrelevant to current systems and security platforms (Tavallaee, et al., 2009, July) (Maciá-Fernández, et al., 2018).

To overcome such limitations, the Communications Security Establishment (CSE) & the Canadian Institute for Cybersecurity (CIC) collaborated to create a more realistic and modern evaluation dataset called CSE-CIC-IDS2018[2] hosted on the Amazon web services and freely available for researchers to download. The dataset was compiled through traffic generated by 50 attacking machines to a victim organization that has five departments, 30 servers, and 420 end-user workstations to generate seven different attack scenarios, including, Heartbleed, Brute-force, DoS, DDoS, Web attacks, Botnet, and infiltration (Ferrag, et al., 2020) (Kanimozhi & Jacob, 2019). Those attacks are discussed in detail below.

---

[2] https://www.unb.ca/cic/datasets/ids-2018.html

### 3.1.1. Brute-force Attack

A Brute-force using a large dictionary of 90 million words is used on a Kali Linux attacking machine targeting a victim Ubuntu 14.0 system using FTP and SSH modules.

### 3.1.2. Heartbleed Attack

In this scenario, Heartleech is used to scan and exploit systems vulnerable to the Heartbleed bug to exfiltrate data. To be able to exploit the vulnerability using Heartleech, a vulnerable OpenSSL version 1.0.1f is used.

### 3.1.3. Botnet

In this attack type, Zeus Trojan horse and Zeus Crypto-Locker ransomware are used for collecting screenshots from infected machines at regular intervals.

### 3.1.4. Denial-of-Service

This attack involves using one machine to take down another machine's web server using a tool called Slowloris created by Robert Hansen without affecting other services.

### 3.1.5. Distributed Denial-of-Service

This attack involves using the High Orbit Ion Cannon(HOIC), an open-source network stress testing and denial-of-service attack application written in BASIC.

### 3.1.6. Web Attacks

This type of attack involved using a readily available security testing platform called the Damn Vulnerable Web App (DVWA). DVWA is a PHP/MySQL web application that is vulnerable and used for security and

penetration testers and researchers. To automate the attacks in XSS and Brute-force, the Selenium framework was used for automation.

#### 3.1.7. Infiltration of the Network from Inside

The attack involves exploiting a vulnerable application using the Metasploit framework by delivering a malicious document to the end-user to create a backdoor on the victim's network that can be used to further reconnaissance activities.

### 3.2. Proposed Solution

This thesis will use unsupervised methods to detect attacks among 16 million network flows in the CSE-CIC-IDS2018 dataset. We will use clustering methods like K-means and mean shift algorithms. In addition, unsupervised outlier detection methods will be used, like isolation forest and local outlier factor.

Moreover, a hybrid approach will be used. First, we will apply K-means algorithm on the training data, and then we will perform two supervised models: logistic regression and random forest. We will then apply local outlier factor and then implement logistic regression, random forest, and gradient boost. Hard voting will be used to aggregate the classifications, and results will be evaluated and compared. After that, synthetic minority oversampling (SMOTE), a commonly used standard for learning from imbalanced data (Fernández, et al., 2018), will be performed on the training data, and then the hybrid approach will be repeated based on the new resampled dataset.

Finally, we will use deep learning approach to distinguish between benign and attacks. For this purpose, we will apply three designed autoencoder algorithms which will be trained on the benign data only to have a better understanding of the benign and to be able to detect them correctly so that the other records will be considered as attacks.

### 3.3. Evaluation Criteria

Validation techniques are essential to evaluate a model's accuracy. In supervised learning, the validation is done mainly by measuring the performance metrics like accuracy, precision, recall, AUC, etc. Conversely, the validation process will not be simple since we do not have the correct labels in unsupervised learning.

For unsupervised learning, two statistical techniques can be used for validation: internal validation and external validation. Internal validation is used to measure cluster quality without the need to external data and is used when we have no further information. Internal validation metrics are further broken down into cohesion and separation. Cohesion assesses the closeness of the elements of the same cluster and is defined by:

$$cohesion(C_i) = \sum_{x_i, y_i \in C_i} proximity(x_i, y_i).$$

*Where $x$ and $y$ are an example in clusters with centroids of $C_i$ and $C_j$*

Separation measures quantify the level of separation between clusters and is defined by:

$$separation(C_i, C_j) = \sum_{x \in C_i, y \in C_j} proximity(x_i, y_i)$$

The proximity function is used to determine how similar a pair of examples are, where we can use similarity, dissimilarity, and distance functions. Clustering is good when it has a high separation between clusters and high cohesion within clusters. Internal indices are usually used in two families of clustering algorithms: hierarchical clustering algorithms and partitional algorithms, and there are several metrics that try to combine separation and cohesion in a single measure. For partitional algorithms, we can use silhouette coefficient, Calisnki-Harabasz coefficient, Dunn index, Xie-Beni score, and Hartigan index. In hierarchical cluster algorithms, Cophenetic Correlation Coefficient and Hubert Statistic can be used to assess the results (Palacio-Niño & Berzal, 2019).

On the other hand, external validation methods can be used when we have additional information, i.e., external class labels for training examples. So external validation methods are not used on most clustering problems. External metrics also depend on the clustering algorithm used. It is used to compare the set of clusters obtained from the clustering algorithm denoted C and results obtained from other partition denoted P, which is obtained by the expert knowledge of the analyst, prior knowledge of the data in the form of class labels, or the results obtained by another clustering algorithm. Then, a contingency matrix built to evaluate the clusters used the terms; T P: The number of data pairs found in the same cluster, both in C and in P, F P: number of data pairs in the same cluster in C but in different clusters in P, F N: The number of data pairs which is in different clusters in C but in the same cluster in P and T N: number of data pairs found in different clusters, both in C and in P. One of the external validation methods is matching sets which are used to compare the clusters from C with ones with the same nature from P and then measure the similarity by precision, recall and purity. Another external validation method is Peer-to-peer Correlation, which depends on the correlation between pairs, i.e., measuring similarity between the result of a grouping process for the same set and using two different methods. The metric used to measure the correlation between pairs are Jaccard coefficient, Rand coefficient, Folkes and Mallows, and Hubert statistical coefficient.to-peer Correlation which depends on correlation between pairs, i.e., measuring similarity between the result of a grouping process for the same set, but by using two different methods.

Since our dataset contains the actual labels, as outlined in section 3.1, we will apply external validation methods using the precision, recall, F1, and average precision score.

# Chapter 4

# Data Cleanup and Exploratory Data Analysis

## 4.1. Data Cleanup

This chapter will discuss dataset exploration, data cleaning, and feature engineering. The dataset consists of 10 CSV files; each contains benign and malicious network traffic collected on different days.

One of 10 CSV files will be read and examined to inspect data and identify potential problems. Once the issues and remedies are identified, the procedure will be applied to the rest of the files.

### 4.1.1 Column Headers and Data Types

The first step in exploring the data is to take a sample file to understand the dataset structure and features.

By examining the sample file, it is noticeable that the column headers like (protocol and cwe_flag_count) are repeated as row values multiple times, as shown in the figure below.

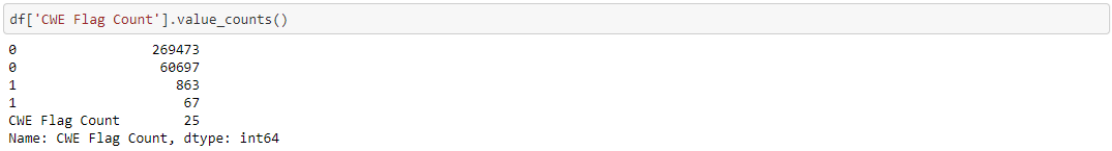


*Figure 1 – Column headers repeated*

Therefore, it must be removed for the algorithms to be run properly. It is also notable that the data points are stored as objects. Hence, features need to be examined to identify the correct data types.

As a matter of simplicity, column headers will be modified by removing whitespaces and non-word characters while converting all the characters to the lower case.

### 4.1.2 Invalid Data

The data type for every variable is examined for validity; two features (flow_byts_s and flow_pkts_s) contain “infinity” as a value. Since python cannot recognize such value, it is replaced with (inf), which is an internal data type in python.

Also, the columns (Flow ID, Src IP, Dst IP, Src Port) in the dataset file (Thuesday-20-02 2018_TrafficForML_CICFlowMeter.csv) will be dropped because they do not appear in the other files, and they are ID's columns that will not have any influence on predicting attacks.

### 4.1.3 Applying the Cleaning Steps to All other Files

The cleaning procced needs to be applied to all the files, and the files will be saved in a new directory. In summary, the following steps should be performed:

1. Remove the duplicated headers.
2. Replace "Infinity" with "inf".
3. Rename column names to remove whitespaces and non-word characters.

Finally, files will be renamed to reflect the attack contained with each file.

## 4.2. Exploratory Data Analysis

In this section, data analysis will be performed; to observe the percentage of malicious records in the data. Also, we will identify network flows attack types. In addition, we will examine if there is a correlation between features and if there is a strong correlation between features and the target variable (if the network flow is benign or malicious).

### 4.2.1 Dataset Import

Firstly, the dataset from all the files will be imported (which includes all different types of attacks), and we will define the data type for each column.

### 4.2.2 Data Imputation for Null Values and Infinity

Datasets' quality and integrity are essential to machine learning and data mining techniques to produce reliable and correct predictions. Imputation can be utilized to address missing, invalid, and null values before learning and testing are applied instead of dropping or deleting such occurrences to avoid the loss of potentially valuable data (Zhang, et al., 2005).

Like many real-world datasets, the CSE-CIC-ID2018 dataset does have some data quality issues that need addressing, while the dataset does not contain records with null values; however, columns "flow_byts_s" and "flow_pkts_s" contain some infinity values which will be substituted by null, and since the two variables represent the number of bytes and packets per each second, the null values will be imputed by the mean of each column.

### 4.2.3 Negative Values

The dataset has 11 columns containing negative values, which will also be substituted by null, and because of the type and the nature of these columns the nulls will be eventually replaced by the mean of each column.

## 4.3 Data Analysis and Visualization

This section will overview the data, like how many attacks there are in the data, how do the features correlate with each other and with the target variable. We will also visualize the data to understand data trends, outliers and patterns.

### 4.3.1 Number and Percentage of Benign Network Flows in relation to Malicious Flows

The first thing we want to examine is how many attacks and benign records are there in the dataset. Figure 2 shows there are 13,484,708 benign, representing 83% of the dataset, while attacks are 2,748,235, making only 17% of the dataset. Hence, we can see that the data is highly imbalanced.

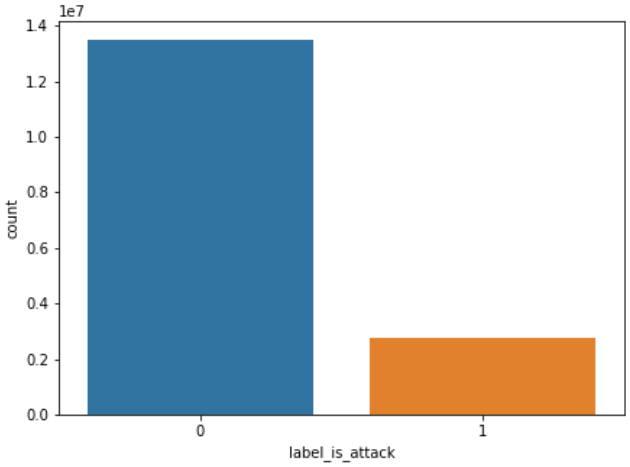


*Figure 2 benign and attack percentage*

### 4.3.2 Number of Flows Per Attack Type

In this section we will examine the different types of attacks in the dataset and the number of records corresponding to each type.

The following graph shows the number of flows accounting for the different attack types. As we can see, DoS attacks-GoldenEye, DoS attacks-Slowloris, DDOS attack-LOIC-UDP, Brute Force-Web, Brute Force-XSS, and SQL Injection has the lowest records in the dataset, which could be challenging to train the multi-class classifier to detect those kinds of network traffic.

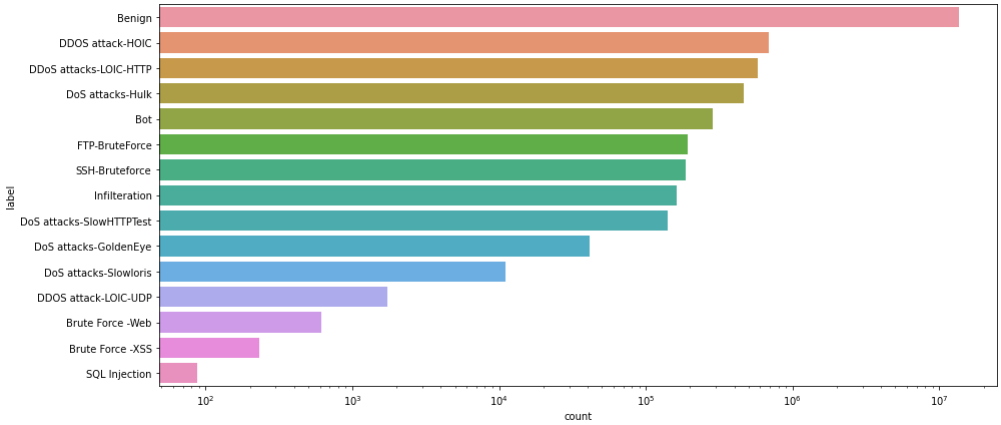


*Figure 3 attack types*

### 4.3.3 Correlation between features

Correlation refers to how close two features are to having a linear relationship with each other. A heat map is produced to examine the Pearson correlation between all features. We notice that there is a clear correlation between some of the feature pairs. High correlation between two features will affect the independent features and bias the prediction.

For example, tot_fwd_pkts (Total packets in the forward direction) and fwd_header_len (Total bytes used for headers in the forward direction) are correlated,

i.e., an increase in the number of packets leads to a higher number of bytes used as headers.

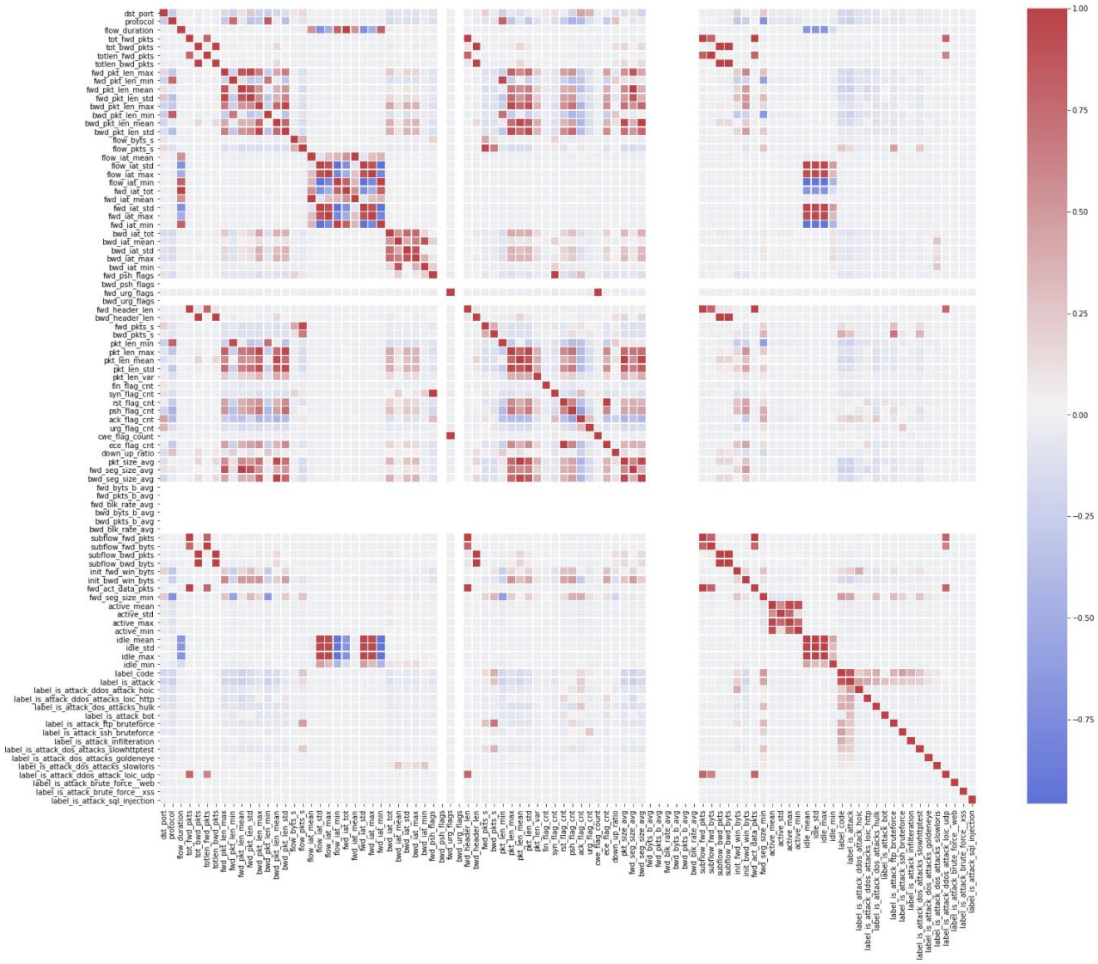

*Figure 4 correlation between the features*

Next, we will see the pair of features which have correlations with an absolute value > 0.8; those features seem to be redundant and can be removed. This helps us in feature selection for the modeling stage. For example, fwd_urg_flags and cwe_flag_count have 1 absolute correlation.

### 4.3.4 Correlation between Features and the Target Variable of Network Flows

To examine the features with the highest correlations, basic statistics can be utilized to examine if those features can predict the network flow label. P-value, which is the probability of obtaining a result that equals or exceeds one observed assuming the null hypothesis of no effect is true; it will give us a measure of the strength to see if the variable affects the target label or not (Goodman, et al., 2019).

fwd_seg_size_min, bwd_pkts_s, ack_flag_cnt, init_fwd_win_byts, and flow_pkts_s are the top 5 features correlated with the target variable (is attack) those features obtained from the correlation matrix explained in the previous section. We will discuss these features in more details in the coming sections.

### 4.3.5 Kolmogorov–Smirnov Test

Validation and test statistics are performed to see if the distribution for each class (benign or not) for each variable with a good correlation with the binary class is the same. The cumulative distribution function $F_n(x) = \frac{1}{n}\sum_{i=1}^{n} 1_{(-\infty,x)}(X_i)$, where $1_{(-\infty,x)}(X_i)$ is the indicator function, equal to 1 if $X_i \leq x$ and equal to 0 otherwise. The Kolmogorov–Smirnov statistic for a given cumulative distribution function F(x) is $D_n = \sup_x |F_n(x) - F(x)|$ where $\sup_x$ is the supremum of the set of distances. If the Kolmogorov–Smirnov (KS) statistic is small or the p-value is high, then we cannot reject the hypothesis that the distributions of the two samples are the same. Else, we reject the hypothesis, and we consider the variable to be good in predicting if there is an attack or not (Moscovich, 2020).

- Minimum segment size observed in the forward direction (fwd_seg_size_min).

Here, we see some descriptive statistics of the fwd_seg_size_min for the benign and attacks like the mean of fwd_seg_size_min for both benign and attacks, the standard deviation, which shows how the values are far away from each other, the minimum and maximum values and the values of the 25%, 50%, and 75% percentiles

*Table 1 fwd_seg_size_min for the benign and attacks.*

| label_is_attack | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| 0 | 13484708 | 16.50157 | 6.757963 | 0 | 8 | 20 | 20 | 56 |
| 1 | 2748235 | 25.30901 | 7.817982 | 0 | 20 | 20 | 32 | 40 |

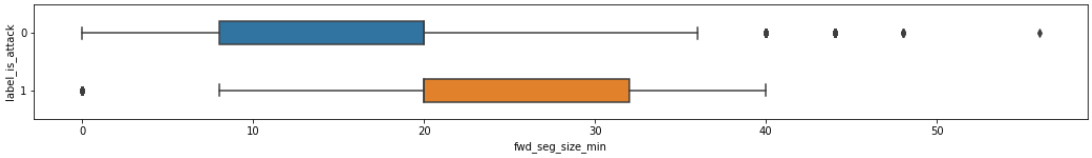


*Figure 5 box plot of benign and attacks- per fwd_seg_size_min*

Next, we plot the distribution of attacks and benign, and we calculate the distribution similarity for attacks and benign with respect to fwd_seg_size_min. The KS statistic is .342, and the p-value is 0, which means that values from both classes do not originate from the same distribution, so the distributions of the feature for each of the classes differ significantly. Hence, fwd_seg_size_min might be a good predictor of the attacks.

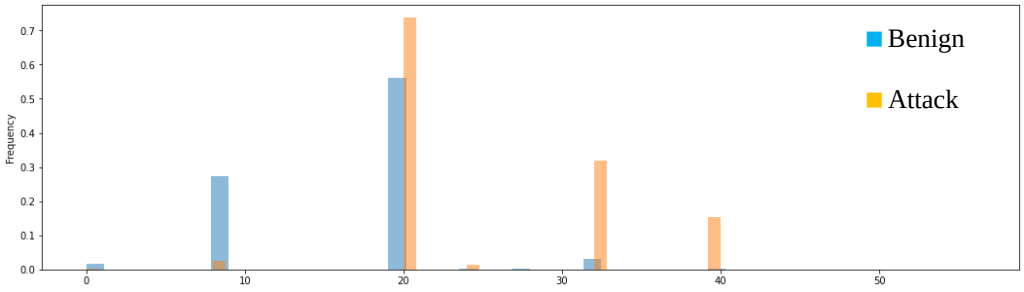


*Figure 6 distribution of benign and attacks- per fwd_seg_size_min*

- Number of backward packets per second

Another variable bwd_pkts_s is tested. The following table and chart show descriptive statistics of bwd_pkts_s.

*Table 2 descriptive statistics of bwd_pkts_s*

| label_is_attack | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| 0 | 13484708 | 4408.273 | 43276.78 | 0 | 0.161656 | 4.30558 | 882.6125 | 2000000 |
| 1 | 2748235 | 68462.94 | 200881.2 | 0 | 0 | 0 | 350.8772 | 2000000 |

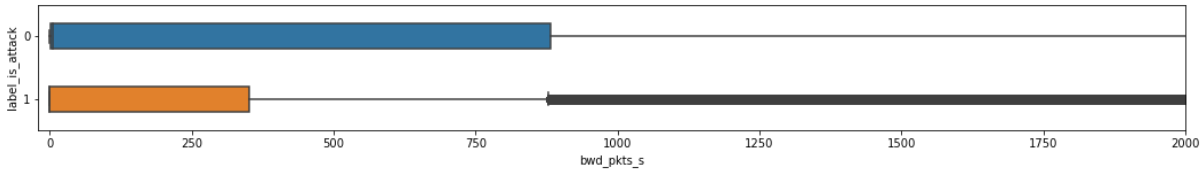


*Figure 7 box plot of benign and attacks- per bwd_pkts_s*

The distribution plot shows the distribution of attacks and benign with respect to bwd_pkts_s. The KS statistic is .3205, and the p-value is 0. Thus, the distributions of the feature for each of the classes differ significantly, and so bwd_pkts_s might also be a good predictor of attacks.

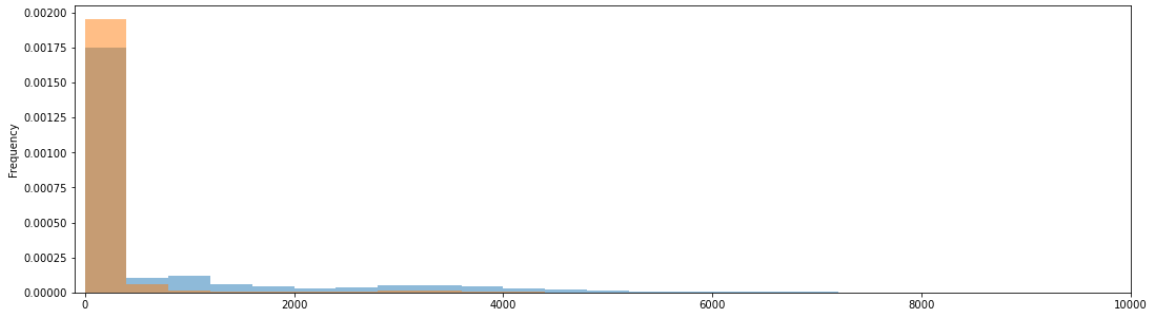


*Figure 8 distribution of benign and attacks- per bwd_pkts_s*

Following the same procedure, we can also see that both ack_flag_cnt and bwd_pkt_len_min could be good predictors of the target variable (is attack).

# Chapter 5

# Modeling

This chapter outlines the methods to select the most relevant features for the model to achieve the highest attack prediction accuracy. Also, we will divide the dataset into training, validation, and testing (James, et al., 2013). After that, we will start modeling; several unsupervised models like K-means, mean shift, isolation forest, and local outlier factor are tested. Furthermore, we will examine a hybrid approach, which is to perform an unsupervised approach then use the results as inputs in the supervised approach. Then, SMOTE oversampling will be applied to the training dataset, and then we will make the hybrid approach again and examine the effect on the results. Finally, we will perform three deep learning models: complete autoencoder, a stacked autoencoder, and denoising autoencoder.

## 5.1 Feature Engineering and Selection

The dataset contains of around 16 million records, and 82 features with three columns representing labels (benign or attack, attack type and attack code). Seven of the features have zero variation and hence will be removed as they do not influence the prediction of the target variable. Moreover, timestamp and dst_port will be removed as they do not influence the prediction of the target variable.

Features identified in the exploratory analysis section to have high correlation are redundant. Therefore, we will do clustering based on the correlation between features and pick one feature from each cluster. After doing that, we will work on the 31 remaining features. The following figure shows the correlation heat map after the removal of highly correlated features.

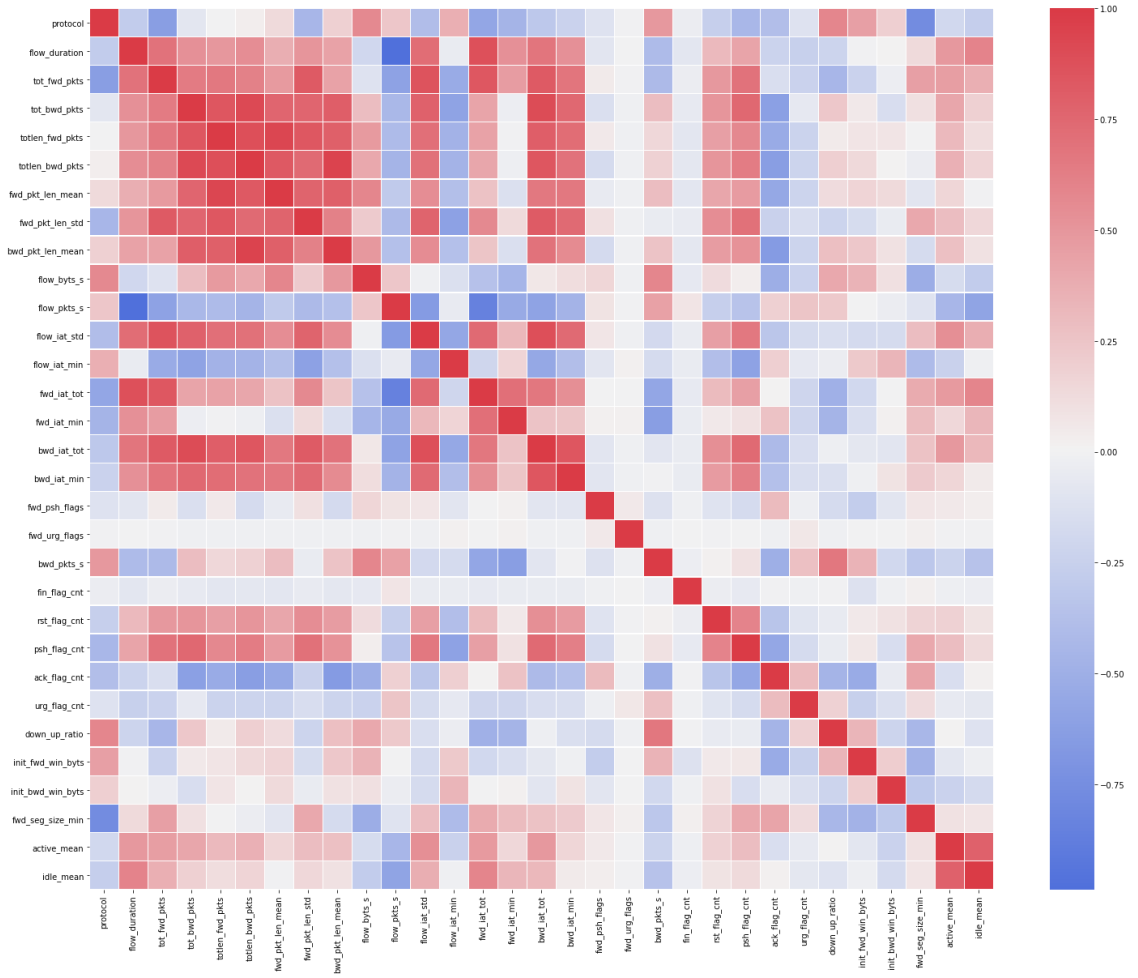


*Figure 9 correlation between features after feature selection*

## 5.2 Train – Test Split

We used the Holdout evaluation by dividing the dataset into three sub-datasets: training, evaluation, and testing, with the ratio of 80, 10,10 consequently. Holdout evaluation aims to test a large data model (16 million records) on different data than the data used to train the model; this provides an unbiased estimate of learning performance. Recall, Precision, F1-score, and Average Precision Score were used to evaluate the performance of the classifiers (Abusalah, 2008). Furthermore, the split is

stratified using the attack category to guarantee that all attacks are represented in the training and test set based on their occurrences in the dataset.

### 5.3 Metrics for Evaluation

There are several ways to evaluate the unsupervised models depending on the algorithm used and other information about the proper labels. External validation methods can be used when we have additional information, i.e., External class labels for training examples. In our case, we have the actual labels, and since the distribution of classes shows that the dataset is highly imbalanced with class 0 - Benign contributing to ~83% of all the samples. For this reason, the metric accuracy is not suitable to measure the performance, and so to evaluate the performance of a classifier, two metrics will be used: recall (weighted average) will be used as the primary metric since the goal of the classifier should be to detect as many attacks as possible while reducing the false positive rate, and this is the metric that classifiers will be optimized for. Precision (weighted average) will be used as a secondary classifier as the number of false-positives should be kept to a minimum. The average precision score will also be calculated.

### 5.4 Models

In this section, we will perform several unsupervised models. Since the data we have is big data (Leevy & Khoshgoftaar, 2020), not all the unsupervised algorithms can be applied, so we will choose algorithms that can be applied to such data like K-means, mean shift, local outlier factor, and isolation forest. The outcomes obtained from the applied algorithms will be compared to the actual labels for the evaluation.

### 5.4.1 K-means

K-means clustering is one of the simplest and most popular unsupervised machine learning algorithms. It simply groups similar data points and discovers underlying patterns. In this method, we must define k (the number of clusters), where a cluster refers to a collection of data points aggregated together because of certain similarities. The k number represents the number of centroids desired in the dataset. A centroid is the imaginary or real location representing the center of the cluster. So, the k number of centroids will be identified, and then every data point will be assigned to the nearest cluster, with the aim of keeping the number of clusters as small as possible.

We first standardize the data for our case and reduce the dimensionality to 2 dimensions to reduce complexity and computational time. Then we fit the algorithm on the dataset and choose the k = 2 after using the elbow method to determine the best number of clusters. After that, the data points are plotted before clustering. The right below figure shows the data points before clustering and after clustering to the right (0-benign, 1-attack).

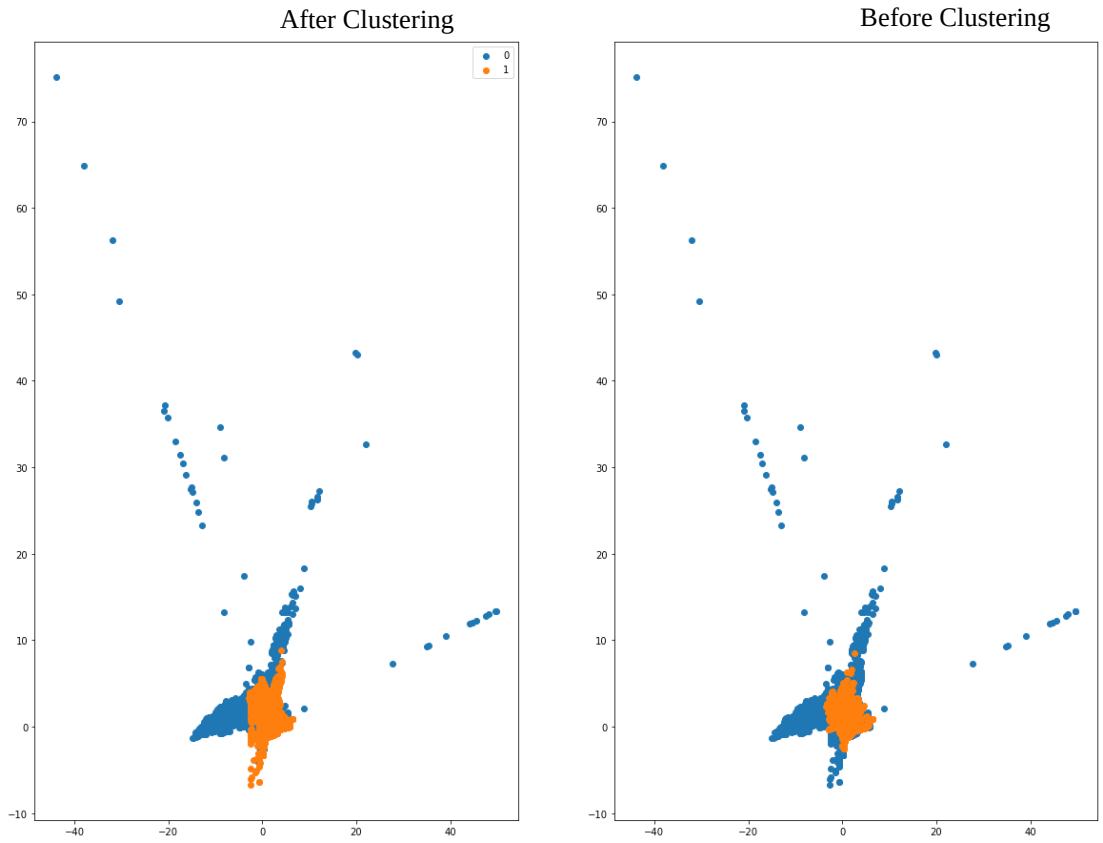


*Figure 10 distribution of points before (right) and after (left) k-means clustering*

Comparing the results with the true labels, we have the following results:

*Table 3 results with the true labels*

| labels | precision | recall | F1-score |
|---|---|---|---|
| 0 | 0.89 | 0.74 | 0.81 |
| 1 | 0.31 | 0.56 | 0.4 |
| accuracy | | 0.71 | |
| macro average | 0.6 | 0.65 | 0.6 |
| weighted average | 0.79 | 0.71 | 0.74 |

### 5.4.2 Mean Shift

Another clustering algorithm is the mean shift algorithm which assigns the data points to the clusters iteratively by shifting points towards the mode. As such, it is also known as the Mode-seeking algorithm. It has applications in the field of image processing and computer vision. Mean shift is a centroid-based algorithm, which

works by performing multiple iterations to select candidates of centroids to be the mean of points within a given region. To eliminate near-duplicates, the candidate centroids are filtered in a post-processing step. Unlike K-Means clustering, mean shift does not require specifying the number of clusters in advance. Instead, the number of clusters is determined by the algorithm with respect to the data. Mean shift is that it is computationally expensive (Vatturi & Wong, 2009, June).

Mean shift created 44 clusters. the majority cluster contains 8,804,512 records, while the remaining records are distributed among 43 clusters. The following figure shows the 44 clusters. When analyzing the clusters it is clear benign records are heavily distributed in more than one cluster, which will affect the overall results.

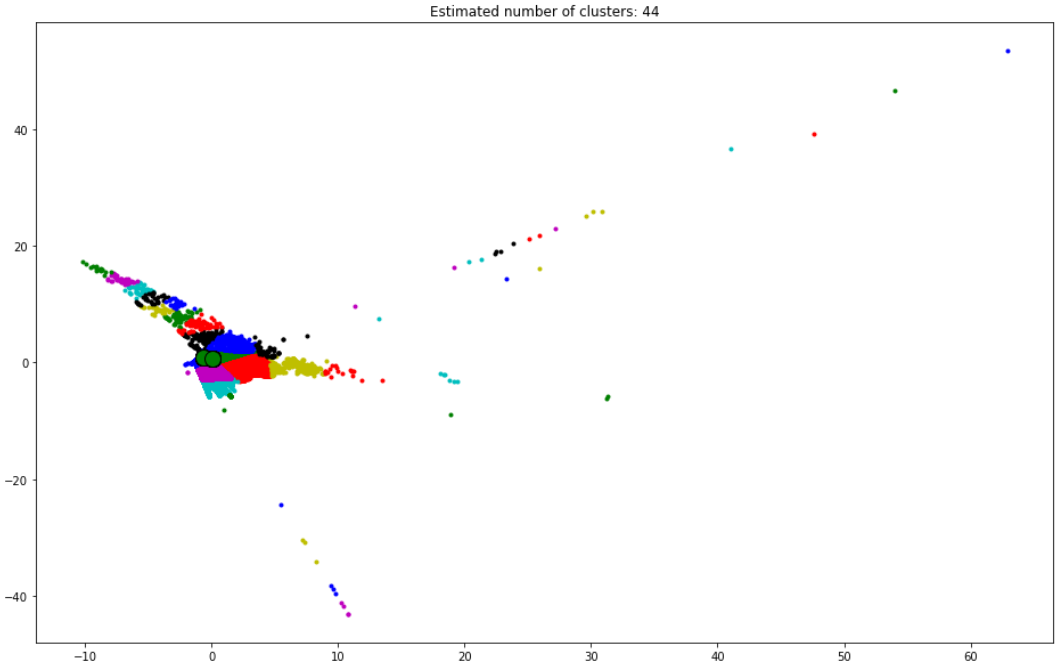


*Figure 11 estimated number of clusters after mean shift clustering*

The results obtained with respect to the true labels are follows:

*Table 4 mean shift results*

| labels | precision | recall | F1-score |
|---|---|---|---|
| 0 | 0.83 | 0.54 | 0.66 |
| 1 | 0.17 | 0.46 | 0.25 |
| accuracy | | | 0.53 |
| macro average | 0.5 | 0.5 | 0.45 |
| weighted average | 0.72 | 0.53 | 0.59 |

As we can see the results of k-means performs better precision score while mean shift provided better recall since it is more robust for outlier detection.

### 5.4.3 Local Outlier Factor

Local Outlier Factor (LOF) is another unsupervised machine learning algorithm intended for outlier detection. It computes the local density deviation of a given data point with respect to its neighbors. It considers outliers as the samples that have a substantially lower density than their neighbors. It works well on high-dimensional datasets.

After standardizing our dataset and after reducing the dimensionality, we fit the algorithm on the data. The figure shows the outliers that have been detected in the dataset.

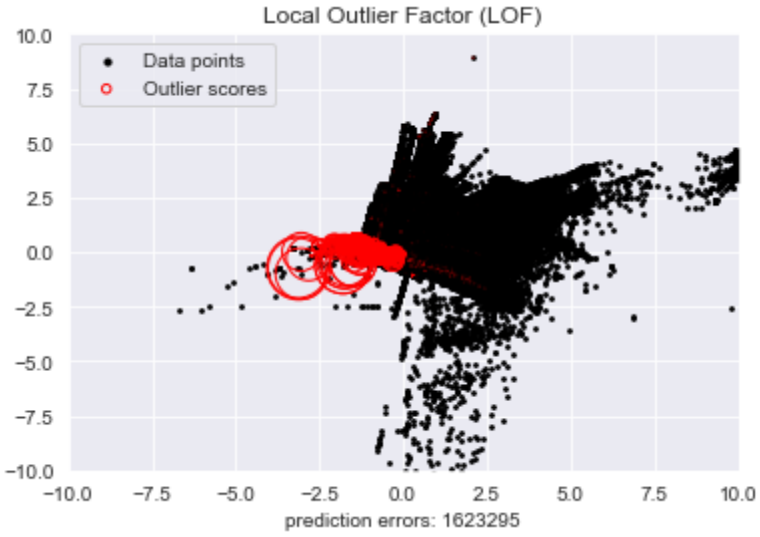


*Figure 12 outliers detected by local outlier factor*

We choose the contamination parameter, which represents the percentage of attacks in the data set, to be 0.1, representing the expected number of outliers; the model detects 1,623,295 outlier observations. The below table compares the results with the actual labels. The model did well in detecting inliers (normal traffic) but performed poorly in detecting the outliers (attacks).

*Table 5 local outlier factor results*

| **labels** | **precision** | **recall** | **F1-score** |
|---|---|---|---|
| **0** | 0.82 | 0.89 | 0.86 |
| **1** | 0.09 | 0.05 | 0.07 |
| **accuracy** | | 0.75 | |
| **macro average** | 0.46 | 0.47 | 0.46 |
| **weighted average** | 0.7 | 0.75 | 0.72 |

Since the data is highly imbalanced, the below table shows LOF performance after performing under-sampling. We did the under-sampling of the normal data by using only 20% of benign data and all the attacks to obtain a balanced dataset of around 5 million records of benign and attack records. The algorithm performed better in

detecting attack cases. However, it showed a decrease in the performance for identifying benign data.

*Table 6 local outlier factor after under-sampling*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.40 | 0.41 | 0.40 |
| **1** | 0.41 | 0.41 | 0.41 |
| **accuracy** | | 0.41 | |
| **macro average** | 0.41 | 0.41 | 0.41 |
| **weighted average** | 0.41 | 0.41 | 0.41 |

### 5.4.4 Isolation Forest

Isolation forest is another unsupervised algorithm effective for outlier and novelty detection in high-dimensional data. It isolates observations by randomly selecting a feature and split value between the maximum and minimum values of the selected feature.

Here, we will train the isolation forest model on the training dataset and then we will predict the outliers on the testing dataset. The following figure shows the outliers (attacks) on the original dataset and the predicted ones on the testing data. The algorithm performs well for normal traffic and poorly for detecting attacks.

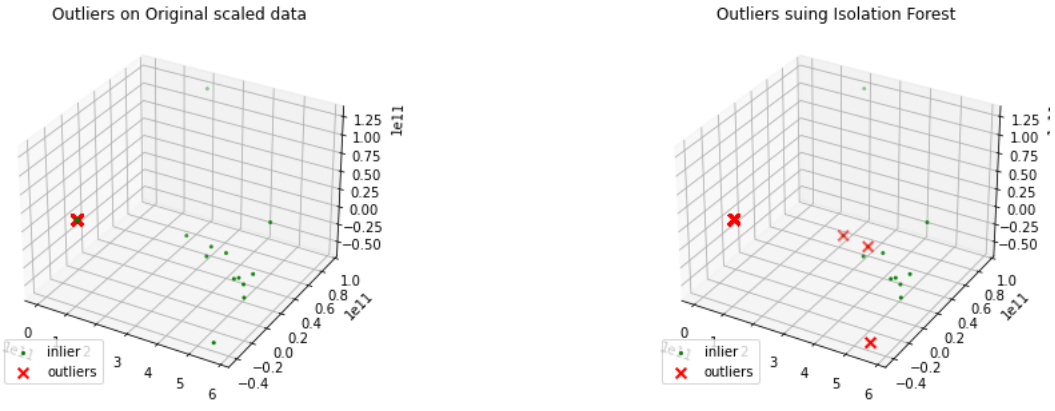


*Figure 13 outlier detected by isolation forest*

*Table 7 isolation forest results*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.81 | 0.82 | 0.81 |
| **1** | 0.05 | 0.05 | 0.05 |
| **accuracy** | | 0.69 | |
| **macro average** | 0.43 | 0.43 | 0.43 |
| **weighted average** | 0.68 | 0.69 | 0.68 |

### 5.5 Hybrid Approach (Unsupervised then Supervised)

This section will apply a hybrid approach by combining unsupervised models (K-means and LOF) and then using the results as input to the supervised models (logistic regression, random forest, and gradient boost). After that, a voting classifier is used to combine the outcome of those three classifiers. Furthermore, synthetic oversampling will be used for the attacks, and then will repeat the hybrid approach to examine if it performs better after the dataset imbalance issue is addressed. The aim of using a hybrid approach is not only to have the better performance it also to avoid the overfitting problem.

### 5.5.1 K-means and logistic regression

After applying k-means on the training and testing data, then, the results of training data are used as training data in the logistic regression model for prediction. The logistic model is used to model the probability of a specified event occurring such as pass/fail, attack/benign (Tolles & Meurer, 2016).

While this combination of algorithms provides good precision and recall for benign, it performs poorly for detecting previously unknown attacks.

*Table 8 k-means with logistic regression performance*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.91 | 0.96 | 0.94 |
| **1** | 0.74 | 0.55 | 0.63 |
| **accuracy** | | | 0.89 |
| **macro average** | 0.82 | 0.76 | 0.78 |
| **weighted average** | 0.88 | 0.89 | 0.88 |
| **average precision score** | | | 0.48 |

### 5.5.2 K-means and random forest

Random forests or random decision forests are an ensemble learning method that works by generating several classifiers and combining their results via a majority vote to classify a new instance to achieve better performance on the same training data (Oshiro, et al., July, 2012).

As shown in the table below, K-Means perform better in precision and recall when coupled with random forest for both benign and attacks than logistic regression. The better results can be attributed to how random forest works by creating a set of decision trees from a randomly selected subset of the training set and then aggregating the votes from different decision trees to decide the final class of the test object.

*Table 9 k-means with random forest performance*

| **labels** | **precision** | **recall** | **F1-score** |
|---|---|---|---|
| **0** | 0.99 | 0.99 | 0.99 |
| **1** | 0.97 | 0.95 | 0.96 |
| **accuracy** | | | 0.99 |
| **macro average** | 0.98 | 0.97 | 0.98 |
| **weighted average** | 0.99 | 0.99 | 0.99 |
| **average precision score** | | | 0.93 |

### 5.5.3 Local Outlier Factor and Random Forest

As shown in the table below, local outlier factor provides comparable results to k-means when combined with random forest. Again such results can be attributed to the nature of random forest described earlier.

*Table 10 local outlier factor and random forest performance*

| **labels** | **precision** | **recall** | **F1-score** |
|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 |
| **1** | 0.96 | 0.95 | 0.96 |
| **accuracy** | | | 0.99 |
| **macro average** | 0.98 | 0.97 | 0.97 |
| **weighted average** | 0.99 | 0.99 | 0.99 |
| **average precision score** | | 0.988 | |

### 5.5.4 Local Outlier Factor and Logistic Regression

Applying local outlier factor with logistic regression produces an average precision score of 0.94 compared to 0.63 achieved by logistic regression alone.

*Table 11 local outlier factor and logistic regression performance*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.94 | 0.97 | 0.96 |
| **1** | 0.84 | 0.70 | 0.76 |
| **accuracy** | | | 0.93 |
| **macro average** | 0.89 | 0.84 | 0.86 |
| **weighted average** | 0.92 | 0.93 | 0.92 |
| **average precision score** | | 0.94 | |

### 5.5.5 Local Outlier Factor and Gradient Boost

Gradient boosting is used for regression and classification by generating a prediction model by combining ensembles of weak learners -typically decision trees into a single strong learner in an iterative fashion to build a more robust prediction model (Bentéjac, et al., 2021).

Gradient boost with local outlier factor offers improvements in the precision of detecting attacks and benign recall with an average precision score of 0.986 compared to 0.93 obtained from applying only gradient boost.

*Table 12 local outlier factor and gradient boost*

| **labels** | **precision** | **recall** | **F1-score** |
|---|---|---|---|
| **0** | 0.99 | 1.00 | 0.99 |
| **1** | 1.00 | 0.93 | 0.97 |
| **accuracy** | | | 0.99 |
| **macro average** | 0.99 | 0.97 | 0.98 |
| **weighted average** | 0.99 | 0.99 | 0.99 |
| **average precision score** | 0.9867 | | |

### 5.5.6 Voting Classifier

Here, we will apply hard voting or majority voting. The main idea of hard voting is that every individual classifier vote for a class and the majority wins. In statistical terms, the predicted target label of the ensemble is the mode of the distribution of individually predicted labels. Therefore, we combine the results obtained from the three classifiers: logistic regression, random forest, and gradient boost to see if the performance will be improved. Then, we take the results (predicting attacks and benign) from each classifier and then use the mode as a voting mechanism (i.e., if at least two out of three models rule it is an attack or a benign).

The table below shows that the voting classifier offers similar results to those obtained by combining local outlier factor with gradient boost.

*Table 13 hard voting classifier*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.99 | 1.00 | 0.99 |
| **1** | 1.00 | 0.93 | 0.96 |
| **accuracy** | | | 0.99 |
| **macro average** | 0.99 | 0.97 | 0.98 |
| **weighted average** | 0.99 | 0.99 | 0.99 |
| **average precision score** | 0.9863 | | |

### 5.5.7 Oversampling using Synthetic Minority Oversampling Technique "SMOTE"

The nature of the dataset is imbalanced. Therefore, the model cannot effectively learn the decision boundary because there are too few examples of the minority class (attacks). To overcome this challenge, we can utilize SMOTE for oversampling instead of simply duplicating attacks in the training dataset to be balanced with benign before fitting the model. Such a process can balance the class distribution without providing any additional information to the model (Chawla, et al., 2002).

In our dataset, we will do the oversampling on some types of attacks that have less than 100,000 records in the training data, after SMOTE and hybrid implementation of local outlier factor followed by random forest and gradient boost. As a result, there was an improvement in the results without overfitting.

LOF with random forest after SMOTE provided a recall of 0.97 compared to 0.95 obtained without SMOTE.

*Table 14 local outlier & random forest*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.99 | 0.99 | 0.99 |
| **1** | 0.97 | 0.97 | 0.97 |
| **accuracy** | | | 0.99 |
| **macro average** | 0.98 | 0.98 | 0.98 |
| **weighted average** | 0.99 | 0.99 | 0.99 |
| **average precision score** | 0.992 | | |

Similarly, after applying SMOTE with LOF and gradient boost, a recall result of 0.94 compared to 0.93 with SMOTE was achieved.

*Table 15 local outlier factor & gradient boost*

| **labels** | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.99 | 1 | 0.99 |
| **1** | 1 | 0.94 | 0.97 |
| **accuracy** | | | 0.99 |
| **macro average** | 0.99 | 0.97 | 0.98 |
| **weighted average** | 0.99 | 0.99 | 0.99 |
| **average precision score** | 0.988 | | |

### 5.6 Deep Learning Approach

In a real-world environment, obtaining labeled data for previously unknown attacks, known as zero-day attacks, is quite challenging and, in some cases, impractical. By definition, zero-day attacks target novel and emerging vulnerabilities unknown to the systems' developers and technology managers. To overcome such limitations, deep learning can help by learning the normal behavior of systems. Deviation from the learned normal and expected behavior can then be flagged as anomalous behavior (He, et al., 2018).

We will use the deep learning approach to create a binary classifier based on representation learning and anomaly detection ideas. We will train multiple deep

learning models on benign data and a small proportion of attacks to fully represent the benign data. We want to create a model that can classify network traffic as benign or malicious, for the network traffic is similar or dissimilar to the data the model has been trained with. The reason for using unsupervised learning, in this case, is that benign data is usually easier to obtain and hence can be provided in higher volumes than malicious data.

### 5.6.1 Autoencoders

An autoencoder is a neural network with the goal of learning a dense representation of the input data. It reconstructs the given inputs by initially encoding the input features as dense representations (latent representations or coding) and then decoding the dense representations to reconstruct the initial inputs. Using this approach, the model should learn the identity function of the input data. To ensure that the model learns a meaningful representation of the data and does not just copy the input features to the output, an autoencoder restricts the model in different ways, for example, by limiting the dimensionality of the latent representations (Nicolau & McDermott, 2016, September).

Here, we will apply three models: under complete, stacked, and denoising autoencoders. The transformation produced by PCA will be ignored since it is a linear transformation. Autoencoders will be applied to all features, except those with zeros and zero variances, so that we will have around 60 features. The idea here is that autoencoders provide an alternative non-linear transformation to PCA that better captures the non-linearity correlation between features, especially when working with anomaly detection (Mohamed, et al., 2019).

The training data will contain only 0.1 attacks, and the rest is benign data. The models will be trained on benign data only and use the autoencoders as an unsupervised approach. Then, we will use the portion of the validation set to identify if there is overfitting. The average precision score is used to perform early stopping of the training process to obtain the best performing model.

### 5.6.2 Under Complete Autoencoder

The under complete autoencoder represents the simplest type of autoencoders, as it uses a single restricted hidden layer containing the latent representations of the data with the following hyper-parameters:

- Loss function: binary cross entropy
    - Optimizer: Adam
- Learning rate: 0.0001
- Hidden layers: 1
    - Containing latent layer with 30 units
    - Activation function: elu

However, inspecting the learning curves, we observe that the model does not overfit the training data.

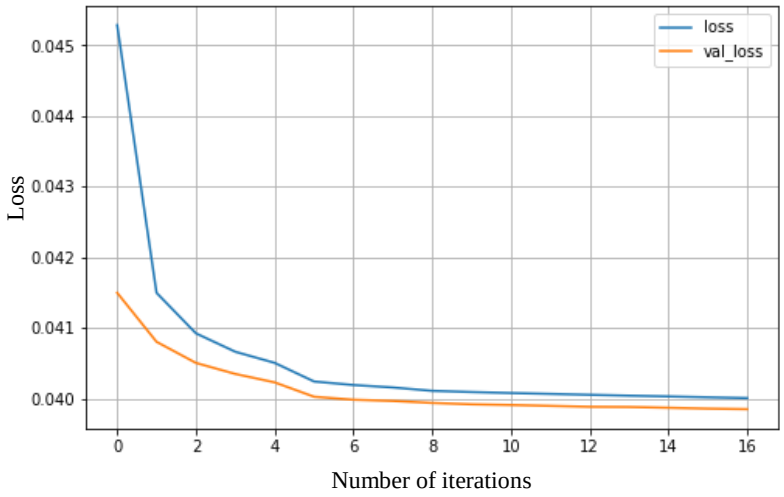


*Figure 14 learning curve- under complete autoencoder*

For this model we obtain an average precision score of 0.79.

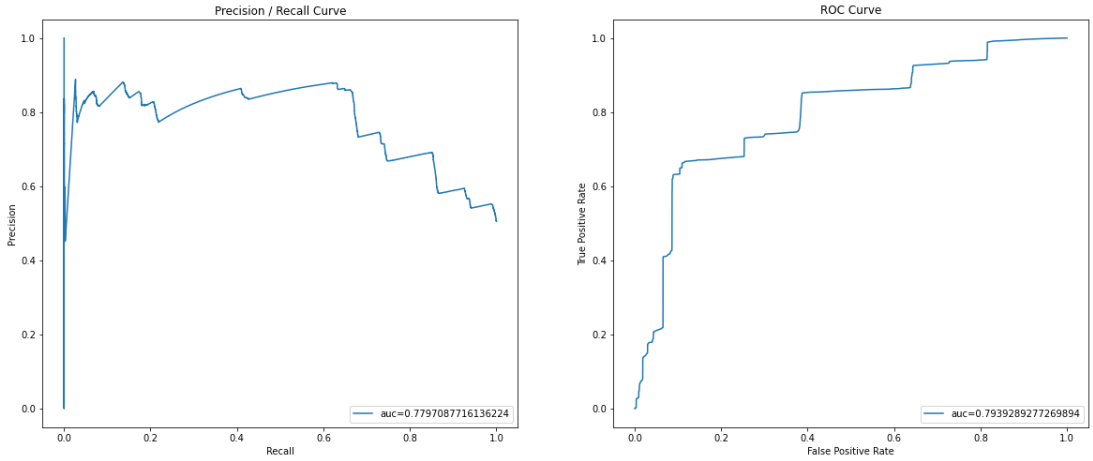


*Figure 15 precision- recall and roc curves- under complete autoencoder*

In the next step, we choose a threshold to define the decision boundary, used to separate benign and malicious data by utilizing the validation data. The threshold is chosen to obtain a minimum recall of 0.85.

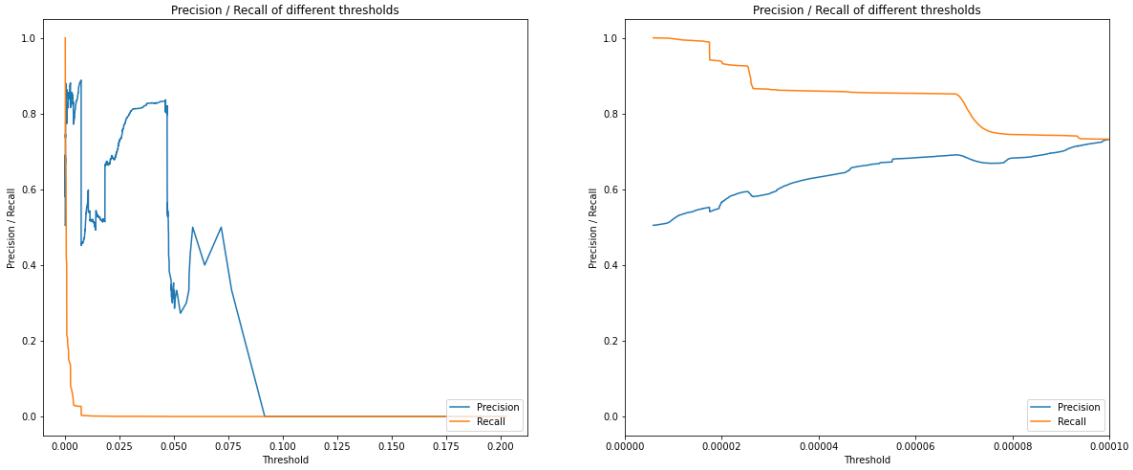


*Figure 16 precision- recall for different thresholds- under complete autoencoder*

The following plot illustrates the distributions of benign and malicious data, outlining the decision boundary. We can observe that the overlap of both traffic classes is quite significant.

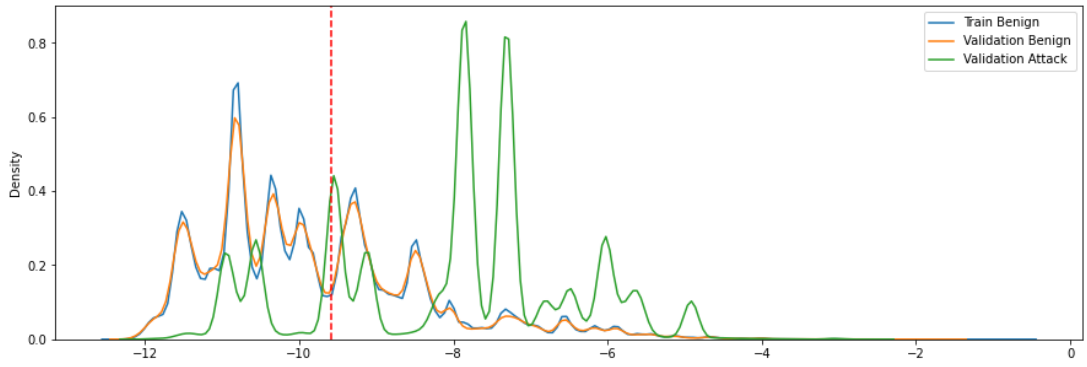


*Figure 17 benign and attack distribution with the decision boundary- under complete auto encoder*

Now, we will see the model's performance on the validation data, and we will see the misclassification per attack type to see the types that can be predicted in high percent by the model.

*Table 16 under complete autoencoder performance*

| labels | precision | recall | F1-score |
|---|---|---|---|
| 0 | 0.80 | 0.61 | 0.69 |
| 1 | 0.69 | 0.85 | 0.76 |
| accuracy | 0.73 | | |
| macro average | 0.75 | 0.73 | 0.73 |
| weighted average | 0.75 | 0.73 | 0.73 |

*Table 17 under complete autoencoder per attack type performance*

| attack type | misclassified | total | percent misclassified |
|---|---|---|---|
| SQL Injection | 27 | 44 | 0.61 |
| Bot | 71,639 | 143,095 | 0.50 |
| Infilteration | 39,148 | 80,967 | 0.48 |
| Brute Force -XSS | 49 | 115 | 0.43 |
| Benign | 522,254 | 1,348,471 | 0.39 |
| Brute Force -Web | 93 | 306 | 0.30 |
| DDOS attack-HOIC | 81,866 | 343,006 | 0.24 |
| DDoS attacks-LOIC-HTTP | 13,295 | 288,095 | 0.05 |
| DoS attacks-GoldenEye | 1 | 20,754 | 0.00 |

As we can see, the model could predict DDoS and DoS attacks with a high accuracy and low misclassification rate.

### 5.6.3 Stacked Autoencoder

The next model we will try is the stacked auto encoder. The main idea of the stacked auto encoder is to modify an under complete auto encoder by adding additional hidden layers, transforming the under complete auto encoder into a deep learning model. Adding more layers allows the auto encoder to create more complex representations, which can enhance performance. However, one must pay attention not to make the auto encoder too powerful as it might simply recreate the outputs by copying the given inputs if possible. The following hyper-parameters were used:

- Loss function: binary cross entropy

- Optimizer: Adam

- Learning rate: 0.0001
- 4 Hidden layers:
  - Containing latent layer with 10 units
  - Activation function: elu

Here we can see the learning curve.

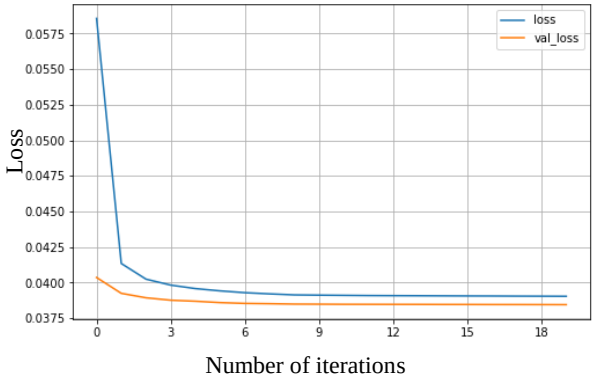


*Figure 18 learning curve- stacked autoencoder*

For this model, we obtain an average precision score of 0.86.

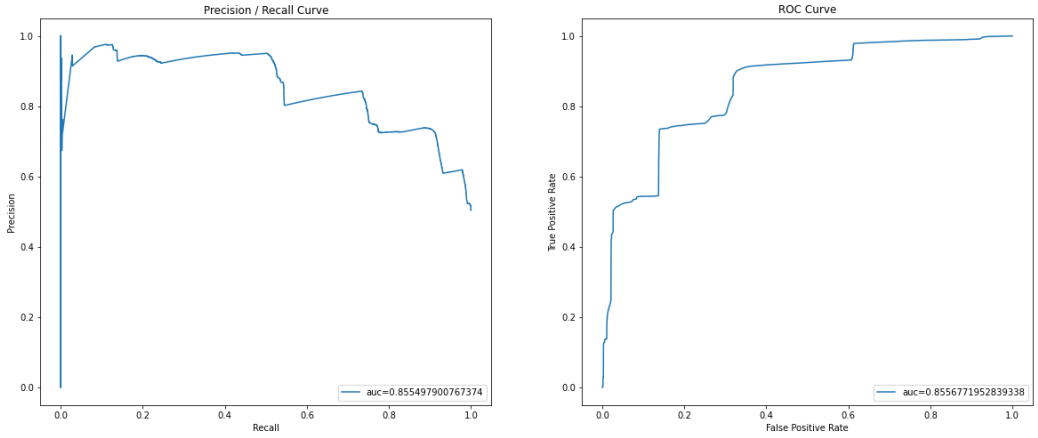


*Figure 19 precision- recall and roc curves- stacked autoencoder*

The following graphs show the precision-recall curve. Here, we will choose the threshold to have recall .90, and we will see the distributions of benign and malicious data. Again, we can observe that the overlap of both traffic classes is quite significant.

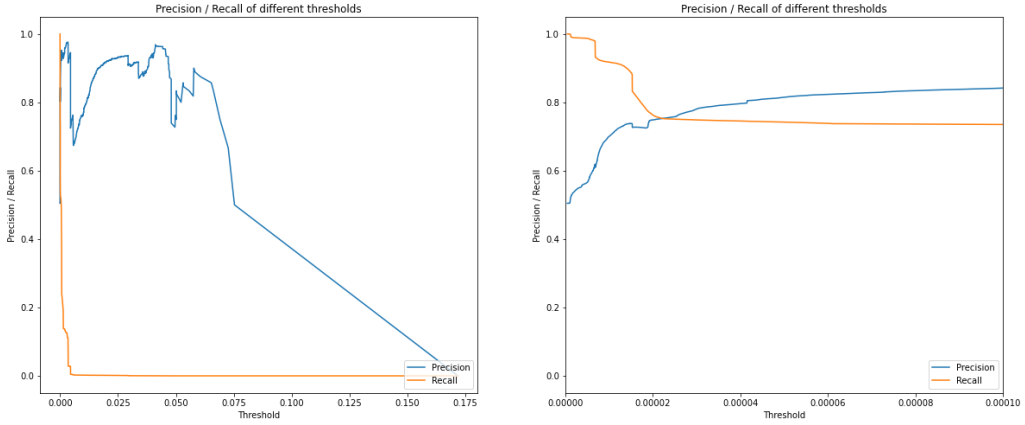


*Figure 20 precision- recall for different thresholds- stacked autoencoder*

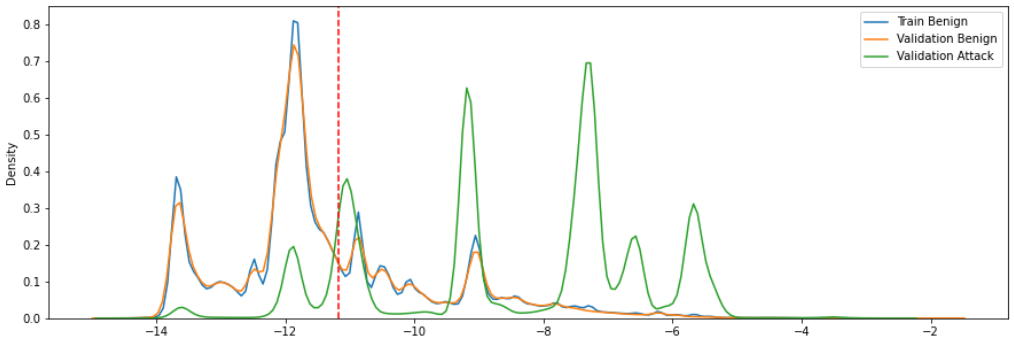


*Figure 21 benign and attack distribution with the decision boundary- stacked auto encoder*

Now, we will see the model's performance on the validation data, and we will see the misclassification per attack type.

*Table 18 stacked autoencoder performance*

| **labels** | **precision** | **recall** | **F1-score** |
|---|---|---|---|
| **0** | 0.87 | 0.67 | 0.76 |
| **1** | 0.74 | 0.90 | 0.81 |
| **accuracy** | | 0.79 | |
| **macro average** | 0.80 | 0.79 | 0.78 |
| **weighted average** | 0.80 | 0.79 | 0.78 |

*Table 19 stacked autoencoder per attack type performance*

| attack type | misclassified | total | percent misclassified |
|---|---|---|---|
| **Infilteration** | 44,269 | 80,967 | 0.55 |
| **Bot** | 71,924 | 143,095 | 0.50 |
| **Brute Force -XSS** | 49 | 115 | 0.43 |
| **Benign** | 442,137 | 1,348,471 | 0.33 |
| **Brute Force -Web** | 82 | 306 | 0.27 |
| **SQL Injection** | 11 | 44 | 0.25 |
| **DDoS attacks-LOIC-HTTP** | 21,070 | 288,095 | 0.07 |
| **DDOS attack-LOIC-UDP** | 6 | 865 | 0.01 |
| **DoS attacks-GoldenEye** | 1 | 20,754 | 0.00 |

We can see that the performance of this model is better than the previous one. Also, we can see that the model can better predict DDoS and DoS attacks.

### 5.6.4 Denoising Autoencoder

Another variant of autoencoders is the denoising auto encoder. In this version, the model is restricted not by limiting the size of the hidden layers but by adding additional noise to the inputs. The aim is to recreate the inputs without containing the noise. They are typically used in image processing to denoise images but can also be applied to tabular data using a simple method originally introduced by Porto Seguro as the winning strategy in a Kaggle competition (Seguro, 2018). The denoising auto encoder we will use is over complete, hence it contains more units in the hidden layers than in the input layer. The model consists of five layers containing 512 units per layer. The hyper parameters used:

- Loss function: binary cross entropy
    - Optimizer: Adam
- Learning rate: 0.0001
- 5 Hidden layers:
    - Containing 512 unit per layer
    - Activation function: elu

- The likelihood for introducing noise = 15%

Here we can see the learning curve and for this model we obtain .80 as average precision score.

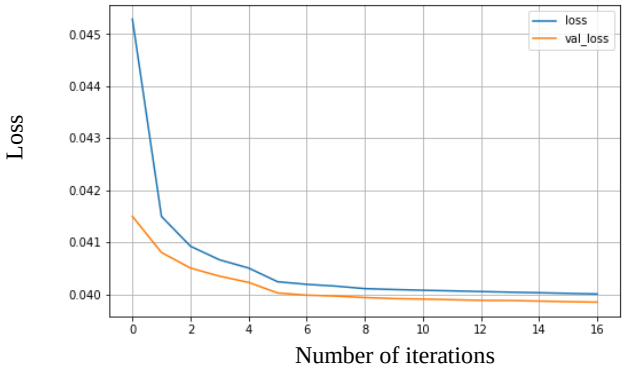


*Figure 22 learning curve- denoising autoencoder*

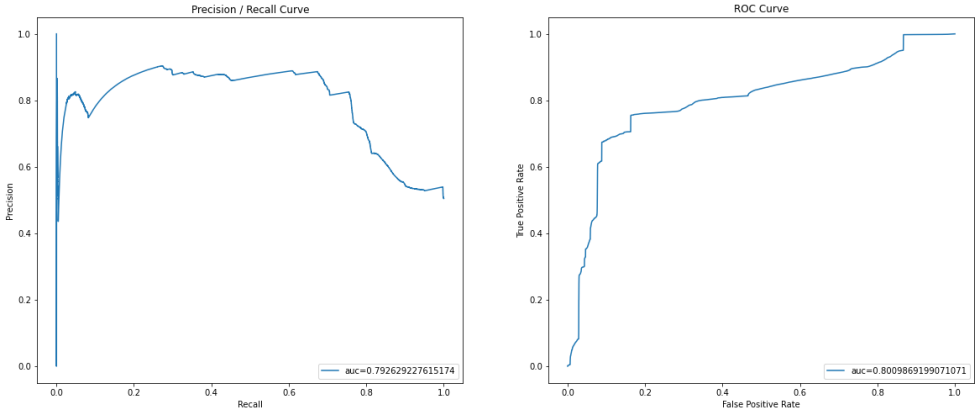


*Figure 23 precision- recall and roc curves- denoising autoencoder*

Next, we will see the precision recall curve. Here, we will choose the threshold to have recall 0.80, and we will see the distributions of benign and malicious data. Again, we can observe that the overlap of both traffic classes is quite significant.

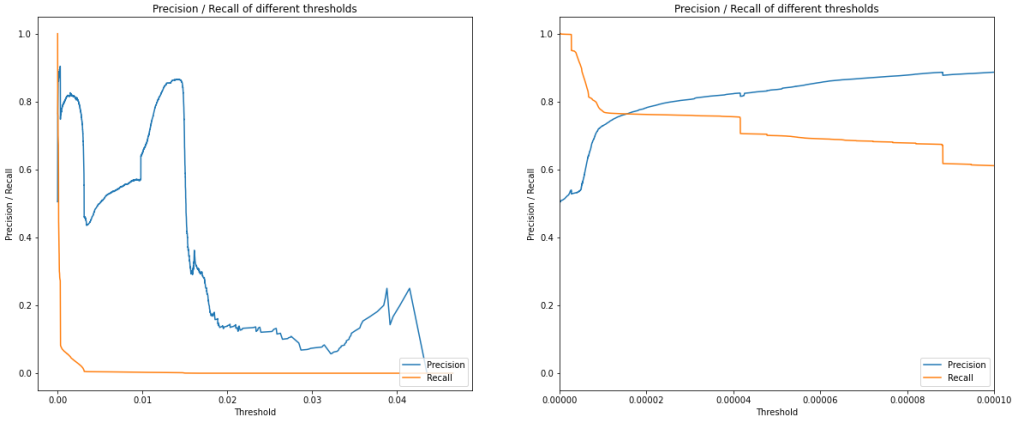


*Figure 24 precision- recall for different thresholds- denoising autoencoder*

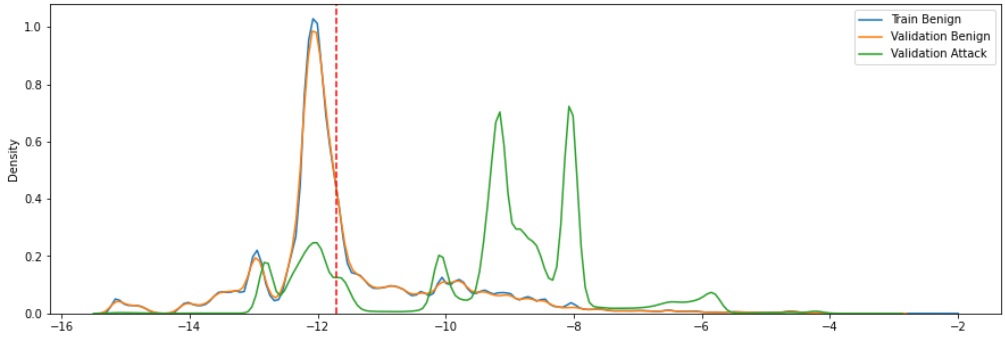


*Figure 25 benign and attack distribution with the decision boundary- denoising auto encoder*

Now, we will see the model's performance on the validation data, and we will see the misclassification per attack type.

*Table 20 denoising auto encoder performance*

| **labels** | **precision** | **recall** | **F1-score** |
|---|---|---|---|
| **0** | 0.76 | 0.66 | 0.71 |
| **1** | 0.70 | 0.80 | 0.75 |
| **accuracy** | 0.73 | | |
| **macro average** | 0.73 | 0.73 | 0.73 |
| **weighted average** | 0.73 | 0.73 | 0.73 |

*Table 21 denoising auto encoder per attack type performance*

| attack type | misclassified | total | percent misclassified |
|---|---|---|---|
| **DDoS attacks-LOIC-HTTP** | 152,788 | 288,095 | 0.53 |
| **Infilteration** | 42,685 | 80,967 | 0.53 |
| **Bot** | 73,741 | 143,095 | 0.52 |
| **Brute Force -XSS** | 48 | 115 | 0.42 |
| **Benign** | 463,712 | 1,348,471 | 0.34 |
| **Brute Force -Web** | 65 | 306 | 0.21 |
| **SQL Injection** | 5 | 44 | 0.11 |
| **SSH-Bruteforce** | 5,491 | 93,794 | 0.06 |
| **DoS attacks-GoldenEye** | 1 | 20,754 | 0.00 |

We can see that the performance of the previous model is better than this one. However, we can see that the model could predict some DDoS and DoS attacks with a high percentage.

### 5.6.5 Performance Comparison and Model Selection

Comparing the models' performance, we obtain the following performance characteristics:

*Table 22 auto encoder performance comparison*

| **Model** | PR Score | Precision | Recall | F1 | Precision (Attack) | Recall (Attack) |
|---|---|---|---|---|---|---|
| **Under complete Auto encoder** | 0.79 | 0.75 | 0.73 | 0.73 | 0.69 | 0.85 |
| **Stacked Auto encoder** | 0.86 | 0.80 | 0.79 | 0.78 | 0.74 | 0.90 |
| **Denoising Auto encoder** | 0.80 | 0.73 | 0.73 | 0.73 | 0.70 | 0.80 |

We see that the stacked autoencoder achieves the best performance, however under complete auto encoder performs better than the denoising autoencoder, which is rather impressive given that the under complete autoencoder is based on a much simpler architecture with one hidden layer compared to the five hidden layers for denoising. This

could be attributed to the fact denoising is mainly used for image processing use cases (Gondara, December, 2016).

We should mention here that these models will have better results if the training set contains only benign records because the model will better understand benign records and distinguish them from other records. In that case, the performance of the models will be different, and we may have the best results when applying denoising autoencoder.

### 5.6.6 Performance on the Testing Set

This section evaluates the performance on the test set using the stacked, denoising, and the under complete autoencoder. The results of the under complete autoencoder are shown in the following tables:

*Table 23 under complete auto encoder performance on testing set*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.82 | 0.61 | 0.7 |
| **1** | 0.67 | 0.85 | 0.75 |
| **accuracy** | 0.73 | | |
| **macro average** | 0.74 | 0.73 | 0.72 |
| **weighted average** | 0.75 | 0.73 | 0.72 |
| **average precision score** | 0.76 | | |

*Table 24 under complete auto encoder performance on testing set per attack type*

| attack type | misclassified | total | percent misclassified |
|---|---|---|---|
| **SQL Injection** | 24 | 39 | 0.62 |
| **Brute Force -XSS** | 56 | 104 | 0.54 |
| **Bot** | 64,030 | 128,786 | 0.50 |
| **Infilteration** | 35,270 | 72,870 | 0.48 |
| **Benign** | 522,294 | 1,348,471 | 0.39 |
| **Brute Force -Web** | 86 | 275 | 0.31 |
| **DDOS attack-HOIC** | 73,699 | 308,705 | 0.24 |
| **DDoS attacks-LOIC-HTTP** | 12,032 | 259,286 | 0.05 |
| **DoS attacks-GoldenEye** | 1 | 20,754 | 0.00 |

Archived results from the stacked autoencoder are highlighted in the following tables:

*Table 25 stacked auto encoder performance on testing set*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.88 | 0.67 | 0.76 |
| **1** | 0.72 | 0.9 | 0.8 |
| **accuracy** | 0.78 | | |
| **macro average** | 0.8 | 0.79 | 0.78 |
| **weighted average** | 0.8 | 0.78 | 0.78 |
| **average precision score** | 0.84 | | |

*Table 26 stacked auto encoder performance on testing set per attack type*

| attack type | misclassified | total | percent misclassified |
|---|---|---|---|
| **Infilteration** | 39,926 | 72,870 | 0.55 |
| **Brute Force -XSS** | 55 | 104 | 0.53 |
| **Bot** | 64,284 | 128,786 | 0.50 |
| **Benign** | 442,414 | 1,348,471 | 0.33 |
| **Brute Force -Web** | 78 | 275 | 0.28 |
| **SQL Injection** | 11 | 39 | 0.28 |
| **DDoS attacks-LOIC-HTTP** | 18,756 | 259,286 | 0.07 |
| **DDOS attack-LOIC-UDP** | 1 | 779 | 0.00 |
| **DoS attacks-GoldenEye** | - | 20,754 | 0.00 |

Finally, we will see the results of the denoising auto encoder.

*Table 27 denoising auto encoder performance on testing set*

| labels | precision | recall | F1-score |
|---|---|---|---|
| **0** | 0.78 | 0.66 | 0.71 |
| **1** | 0.68 | 0.8 | 0.74 |
| **accuracy** | 0.73 | | |
| **macro average** | 0.73 | 0.73 | 0.72 |
| **weighted average** | 0.73 | 0.73 | 0.72 |
| **average precision score** | 0.78 | | |

*Table 28 denoising auto encoder performance on testing set per attack type*

| attack type | misclassified | total | percent misclassified |
|---|---|---|---|
| **DDoS attacks-LOIC-HTTP** | 137,285 | 259,286 | 0.53 |
| **Infilteration** | 38,446 | 72,870 | 0.53 |
| **Bot** | 65,839 | 128,786 | 0.51 |
| **Brute Force -XSS** | 52 | 104 | 0.50 |
| **Benign** | 463,410 | 1,348,471 | 0.34 |
| **Brute Force -Web** | 66 | 275 | 0.24 |
| **SQL Injection** | 8 | 39 | 0.21 |
| **SSH-Bruteforce** | 4,939 | 84,415 | 0.06 |
| **DoS attacks-GoldenEye** | - | 20,754 | 0.00 |

Again, we get the best results from the stacked auto encoder with average precision score of .84. Some types of attacks have been misclassified with high percent. However, we could detect other types with high accuracy like DoS, SSH-Bruteforce and DDoS attacks.

# Chapter 6

# Results and Conclusions

## 6.1 Evaluation of Results

This thesis performed several unsupervised ML methods to detect Network attacks based on the CSE-CIC-IDS2018 dataset. With such big data dataset k-means performed better than mean shift, producing an F1-score of 0.74 compared to 0.59 and 0.56 recall of attacks compared to mean shift's 0.46.

Unsupervised outlier detection methods were examined as well, like local outlier factor and isolation forest. Local outlier factor produced an F1-score of 0.72 but performed poorly in recall of attacks with a score of 0.05; nevertheless, it scored 0.89 for recall for benign. Since the dataset is highly imbalanced, under-sampling was performed by taking only 20% of benign data and all attacks, applying the local outlier factor resulted in a poor F1-score of 0.41, 0.41 attack recall, and 0.41 benign recall. On the other hand, isolation forest produced an F1-score of 0.68, 0.05 attack recall, and 0.82 for benign recall.

Next, a hybrid strategy was performed combining supervised and unsupervised algorithms. After applying k-means to the training data, two supervised models are implemented: logistic regression and random forest. While the former produced 0.89 accuracy and F1-score of 0.88, the latter performed better, producing an accuracy of F1-score of 0.99. Furthermore, the hybrid approach was performed by combining local outlier factor with logistic regression, random forest, and gradient boost; with logistic regression, the approach resulted in 0.93 accuracy and 0.92 F1-score; the attack recall was 0.70 with an average precision score of 0.94. Local outlier factor with random forest resulted in 0.99 accuracy, 0.99 F1-score, 0.95 attack recall, and 0.988 for the average precision score. Local outlier factor and gradient boost resulted in 0.99 for accuracy, 0.99

for F1-score, 0.93 attack recall, and 0.987 for average precision score. The gradient boost results are close to random forest.

In general, using local outlier factor with supervised models produced better results than using K-means with supervised models. After applying the local outlier factor with the three supervised models, hard voting was used in an attempt to obtain better results; the results of hard voting were: 0.99 accuracy, 0.99 F1-score, 0.93 recall for attacks, 1.0 recall for benign, and 0.986 average precision score. Thus, the results were close to the ones obtained when applying local outlier factor and gradient boost.

To overcome the challenges (Leevy & Khoshgoftaar, 2020) witnessed in their research discussed earlier in the literature, in addition to using hybrid ML approach which heavily contributed to the reduction of dataset overfitting challenge, we have used feature correlation analysis to locate the critically important features on which others depend to reduce possible overfitting caused by features. And feature engineering in order to optimize the dataset. We have also used SMOTE to overcome the overfitting challenge caused by the nature of the dataset which is heavily unbalanced.

Synthetic minority oversampling (SMOTE) was applied to the training data, and the hybrid approach was performed again, which improved the accuracy. For example, the attack recall for local outlier factor and random forest was 0.97 with an average precision score of 0.99. The recall attack for local outlier factor and gradient boost was 0.94 with 0.988 for an average precision score.

Finally, we used deep learning approach to distinguish between benign and attacks. For this purpose, three designed autoencoder algorithms were applied. The training was on the benign data only to better recognize and correctly detect benign records, i.e., other records will be considered attacks. Under complete autoencoder achieved 0.73 accuracy,

0.72 F1-score, 0.85 recall for attacks, 0.61 recall for benign, and 0.76 for an average precision score. The stacked autoencoder delivered 0.78 accuracy, 0.78 F1-score, 0.90 recall for attacks, 0.67 recall for benign, and 0.84 for an average precision score. Lastly, the denoising autoencoder achieved 0.73 accuracy, 0.72 F1-score, 0.80 attack recall, 0.66 benign recall, and .78 average precision score.

Comparing our performance to the results obtained in (Verkerken, et al., 2020), although they have achieved better results with 0.94 F1-score, 0.95 precision, and 0.93 recall compared to our results of 0.68, 0.68, and 0.69, respectively. On the other hand, we achieved a 0.78 F1-score when applying stacked autoencoder with 0.80 precision, 0.78 recall, and 0.90 recall for attacks. Our work used both attacks and benign with the training data containing 11 million records compared to only 50,0000 benign-only training set. Applying the algorithms on such a small dataset will give better results. It is worth noting that using the benign data only in training will make the algorithm distinguish between the benign and the attacks even better. However, it may be impractical in real-world scenarios where it is difficult to separate benign from attacks.

The autoencoder deep learning approach provided the best results among the unsupervised machine learning methods. Combining an unsupervised machine learning algorithm with a supervised machine learning algorithm after resampling the dataset demonstrated high accuracy, specifically when using Local Outlier as an Unsupervised ML and Random Forest as Supervised ML. (Soheily-Khah, et al., 2018) suggest that using k-means combined with Random Forest on ICX 2012 Dataset outperformed the other state-of-the-art methods through the high accuracy, high detection rate, and low false alarm rate, overall. In our experiments, local outlier & random forest after applying SMOTE overperformed all other methods on CSE-CIC-IDS2018 dataset.

The proposed solution by (Farhan, et al., 2020) was to perform deep neural network algorithm after doing simple data cleaning and removing some unnecessary features. The algorithm was trained only on around 162,500 records compared with around 11 million records trained with our models. Although the researchers achieved an accuracy of 0.90 compared to the 0.78 we achieved with the stacked autoencoder approach, the precision and recall for the proposed algorithm were 0.65 and 0.59, respectively, compared to the stacked autoencoder 0.72 precision and 0.90 recall. Furthermore, we achieved significantly better results with our hybrid approach before and after using SMOTE.

The following table summarizes our results and compares them to the results obtained by other researchers:

*Table 29 Comparing our results with other research*

| | **Proposed models** | **Precision** | **Recall** | **F1-Score** | **Notes** |
|---|---|---|---|---|---|
| Our results | LOF & gradient boost | 1 | 0.93 | 0.99 | • Complete CSE-CIC-IDS2018 dataset of 16 million records<br>• Hierarchal clustering for feature selection<br>• Hard voting to combine results from different hybrid approaches |
| | Voting Classifier (for hybrid approach) | 1 | 0.93 | 0.99 | |
| | LOF with gradient boost | 0.97 | 0.97 | 0.99 | |
| | SMOTE (LOF & gradient boost) | 1 | 0.94 | 0.99 | |
| | Stacked Autoencoder | 0.74 | 0.90 | 0.78 | |
| Soheily-Khah, Marteau, & Béchet, 2018 | Hybrid: K-means and Random Forest (kM-RF) | 0.999 | 1 for SSH | 1 for SSH | • ISCX2012 dataset, doesn't include https with only 2M records<br>• K-Means was used as under sampling technique<br>• DNS flows reduced by 82% (309K to 55K) |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | • HTTPWeb flows reduced by 46% ( 881K to 367K) |
| Verkerken, D'hooge, Wauters, Volckaert, & De Turck, 2020 | Autoencoders (stacked) | 0.95 | 0.98 | 0.96 | • CIC-IDS-2017 dataset<br>• The training dataset used data collected on Monday (benign day) to generate a sample of 50,000 records benign only for training<br>• 200,000 records for validation and testing |
| Farhan, Maolood, & Hassan, 2020 | Dense deep neural network (DNN) | FTP_Brutforce : P:0/R:0/ F1-score:0<br>SSH_Brutforce: P: 0.87/ R: 0.27 / F1-score: 0.41<br>DDOS attack_LOTC_UDP : 1/1/1 | | | • CSE-CIC-IDS2018 dataset<br>• The algorithm was trained only on around 162,500 records<br>• We achieved comparable results for DDoS |
| Kumar, Glisson, & Benton, 2020 | Mean shift | 81.2 | 0.75 | 0.99+ | • KDD'99 outdated dataset |

### 6.2 Limitations and Future Work

The data we used in this thesis is big data, and applying the unsupervised methods has some limitations as some unsupervised algorithms need extensive computations, which makes them unsuitable for big data. We faced challenges detecting attacks using outlier detection algorithms since the records are not far from benign.

For the deep learning approach, more benign records are needed for better performance, which could be problematic in real-world scenarios since labeling is needed to ensure that the percentage of attacks is reasonable. Also, the threshold level for a record to be considered an attack could be challenging and needs careful tuning.

We have applied hard voting on the hybrid classifications, trying soft voting by giving different weights to classifiers based on their performance could be of benefit. Also, we may combine deep learning outcome with supervised ML, this will improve the performance, as it did in the hybrid experiments.

# References

Abusalah, M., 2008. *Cross language information retrieval using ontologies.* Sunderland: University of Sunderland.

Amoli, P. et al., 2016. Unsupervised network intrusion detection systems for zero-day fast-spreading attacks and botnets. *International Journal of Digital Content Technology and its Applications 10(2),* pp. 1-13.

Bentéjac, C., Csörgő, A. & Martínez-Muñoz, G., 2021. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review,* pp. 54(3), 1937-1967..

Berry, M. W., Mohamed, A. & Yap, B. W. (., 2019. *Supervised and unsupervised learning for data science..* s.l.:Springer Nature.

Blanco, R., Malagón, P., Briongos, S. & Moya, J. M., 2019. *Anomaly Detection Using Gaussian Mixture Probability Model to Implement Intrusion Detection System.* Cham, Springer, pp. 648-659.

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P., 2002. SMOTE: synthetic minority over-sampling technique.. *Journal of artificial intelligence research,* pp. 16, 321-357..

Farhan, R. I., Maolood, A. T. & Hassan, N., 2020. Performance analysis of flow-based attacks detection on CSE-CIC-IDS2018 dataset using deep learning.. *Indonesian Journal of Electrical Engineering and Computer Science,* pp. 20(3), 1413-1418.

Fernández, A., Garcia, S., Herrera, F. & Chawla, N. V., 2018. SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary.. *Journal of artificial intelligence research,* pp. 61, 863-905.

Ferrag, M. A., Maglaras, L., Moschoyiannis, S. & Janicke, H., 2020. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study.. *Journal of Information Security and Applications,* pp. 50, 102419.

Friedberg, I., Skopik, F., Settanni, G. & Fiedler, R., 2015. Combating advanced persistent threats: From network event correlation to incident detection.. *Computers & Security, Vol 48,* pp. 35-57.

Gogoi, P., Borah, B. & Bhattacharyya, D. K., 2010. Anomaly detection analysis of intrusion data using supervised & unsupervised approach. *J. Convergence Inf. Technol., 5(1),* pp. 95-110.

Gondara, L., December, 2016. *Medical image denoising using convolutional denoising autoencoders..* s.l., IEEE, pp. 241-246.

Goodman, W. M., Spruill, S. E. & Komaroff, E., 2019. A proposed hybrid effect size plus p-value criterion: empirical evidence supporting its use. The American Statistician, 73(sup1), 168-185.. *The American Statistician,* pp. 73(sup1), 168-185..

Grill, M., Pevný, T. & Rehak, M., 2017. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing.. *Journal of Computer and System Sciences,* pp. 83(1), 43-57.

He, Z., Raghavan, A., Chai, S. & Lee, R., 2018. Detecting zero-day controller hijacking attacks on the power-grid with enhanced deep learning. *arXiv,* p. 1806.06496.

Jamadar, R. A., 2018. Network intrusion detection system using machine learning.. *Indian Journal of Science and Technology,* pp. 7(48), 1-6..

James, G., Witten, D., Hastie, T. & Tibshirani, R., 2013. *An introduction to statistical learning (Vol. 112, p. 178)..* New York: Springer.

Kanimozhi, V. & Jacob, T. P., 2019. Calibration of various optimized machine learning classifiers in network intrusion detection system on the realistic cyber dataset cse-cic-ids2018 using cloud computing. *International Journal of Engineering Applied Sciences and Technology,* pp. 4(6), 2455-2143..

Kumar, A., Glisson, W. & Benton, R., 2020. *Network attack detection using an unsupervised machine learning algorithm.* s.l., s.n.

Laskov, P., Düssel, P., Schäfer, C. & Rieck, K., 2005. *Learning intrusion detection: supervised or unsupervised?.* Berlin, Heidelberg., Springer, pp. 50-57.

Leevy, J. L. & Khoshgoftaar, T. M., 2020. A survey and analysis of intrusion detection models based on CSE-CIC-IDS2018 Big Data. *Journal of Big Data,* pp. 7(1), 1-19.

M. Tavallaee, E. B., Lu, W. & Ghorbani, A. A., 2009. *A detailed analysis of the kdd cup 99 data set, in: Computational Intelligence for Security and Defense Applications.* s.l., IEEE, p. pp. 1–6.

Maciá-Fernández, G. et al., 2018. UGR ‘16: A new dataset for the evaluation of cyclostationarity-based network IDSs. *Computers & Security,,* pp. 73, 411-424.

Mohamed, S., Ejbali, R. & Zaied, M., 2019. *Denoising autoencoder with dropout based network anomaly detection..* s.l., s.n., p. 110.

Moscovich, A., 2020. Fast calculation of p-values for one-sided Kolmogorov-Smirnov type statistics.. *arXiv,* p. 2009.04954.

Nicolau, M. & McDermott, J., 2016, September. *A hybrid autoencoder and density estimation model for anomaly detection..* s.l., Springer, Cham., pp. pp. 717-726.

Oshiro, T. M., Perez, P. S. & Baranauskas, J. A., July, 2012. *How many trees in a random forest?.* Berlin, Heidelberg., Springer, pp. pp. 154-168.

Palacio-Niño, J. O. & Berzal, F., 2019. Evaluation metrics for unsupervised learning algorithms. *arXiv,* p. 1905.05667.

Repalle, S. A. & Kolluru, V. R., 2017. Intrusion detection system using ai and machine learning algorithm. *International Research Journal of Engineering and Technology (IRJET),* pp. 1709-1715.

Seguro, P., 2018. *Safe Driver Prediction.* [Online]
Available at: https://www.kaggle.com/c/porto-seguro-safe-driver-prediction

Singh, P. & Tiwari, A., 2014. A review intrusion detection system using KDD'99 Dataset. *Int J Eng Res Technol,* pp. 3(11), 1103-1108.

Singh, U. K., Joshi, C. & Kanellopoulos, D., 2019. A framework for zero-day vulnerabilities detection and prioritization. *Journal of Information Security and Applications,* pp. 46, 164-172.

Soheily-Khah, S., Marteau, P. F. & Béchet, N., 2018. *Intrusion detection in network systems through hybrid supervised and unsupervised machine learning process: A case study on the iscx dataset.* s.l., IEEE, pp. pp. 219-226.

Syarif, I., Prugel-Bennett, A. & Wills, G., 2012. *Unsupervised clustering approach for network anomaly detection.* Berlin, Heidelberg, Springer, pp. 135-145.

Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A. A., 2009, July. *A detailed analysis of the KDD CUP 99 data set..* s.l., IEEE, pp. pp. 1-6.

Thakkar, A. & Lohiya, 2020. A Review of the Advancement in Intrusion Detection Datasets. *Procedia Computer Science,* pp. 167, 636-645.

Tolles, J. & Meurer, W. J., 2016. Logistic regression: relating patient characteristics to outcomes.. *Jama,* pp. 316(5), 533-534..

Vatturi, P. & Wong, W. K., 2009, June. *Category detection using hierarchical mean shift..* s.l., s.n., pp. 847-856.

Verkerken, M. et al., 2020. *Unsupervised Machine Learning Techniques for Network Intrusion Detection on Modern Data..* s.l., IEEE., pp. 1-8.

Zhang, J. & Zulkernine, M., 2006. *Anomaly based network intrusion detection with unsupervised outlier detection.* s.l., s.n., pp. Vol. 5, pp. 2388-2393.

Zhang, S., Qin, Z., Ling, C. X. & Sheng, S., 2005. Missing is useful": Missing values in cost-sensitive decision trees.. *IEEE transactions on knowledge and data engineering,* pp. 17(12), 1689-1693..

# Appendix A

## 1) Experimental Environment

The Environment used for this thesis comprised of the following components:

- Hardware/OS used:
    - Server : Processor Intel(R) Xeon(R) Gold 6262V CPU @ 1.90GHz with 48 Core.
    - Installed Memory : 256 GB RAM.
    - OS: Windows Server 2019 - Version 1809 (OS Build 17763.1757)
- Toolkit:
    - Anaconda : 2020.11
    - jupyter-notebook : 6.1.4
    - python : 3.8.5
    - sklearn : 0.24.1
    - pandas : 1.1.3
    - numpy : 1.19.2
    - tensorflow : 2.4.1
    - keras : 2.4.3
    - seaborn : 0.11.0
    - matplotlib : 3.3.2

## 2) Source Code

The source code repositories can be found on GitHub using the following link: https://github.com/nizarpal/Hybrid-Anomaly-Detection

**3) Dataset**

Canadian Establishment for Cybersecurity CSE-CIC-IDS2018 dataset can be found on Amazon aws ( https://registry.opendata.aws/cse-cic-ids2018/ )

## ملخص

يقع على عاتق الشركات والمؤسسات مسؤولية حفظ أمن خدماتها وبناها التحتية من الاخطار التي وبشكل متسارع تهدد امنها المعلوماتي. ولكي تتغلب على مثل هذه التهديدات توظف المؤسسات ترسانة من أدوات الحماية من أهمها نظام كشف التسلل (Intrusion Detection System) والذي يمكنه اكتشاف الهجمات ومحاولات التسلل والاختراق تلقائيًا. ولكي يكون نظام IDS فعالاً، فإنه يحتاج إلى اكتشاف جميع أنواع الهجمات مع عدم الإخلال والتعرض لعمليات الأنظمة العادية من خلال تصنيفها خطأً على أنها هجمات. كما يتحتم على نظام IDS اكتشاف الهجمات غير المعروفة مسبقا لعدم وجودها في قاعدة البيانات التي يعتمد عليها. لكي تتمكن أنظمة الحماية من اكتشاف مثل هذه المحاولات المتطورة، تقليديا يتم الاعتماد على تعلم السلوك الطبيعي والمعروف لكشف وعزل التهديدات (Anomaly Detection)، في المقابل تنتج هذه الالية نتائج إيجابية وهمية (false-positives). تستطلع هذه الرسالة خوارزميات وتقنيات التعلم الآلي المناسبة، وتحديداً التعلم الآلي الهجين (hybrid learning) والذي يجمع بين التعلم الآلي الخاضع والغير الخاضع للإشراف لاكتشاف الهجمات غير المعروفة مسبقا لتقليل النتائج الوهمية عن طريق تحليل الأحداث الناتجة عن الأنظمة والأجهزة المتصلة المختلفة.