University of Piraeus

Department of Digital Systems

Postgraduate Program of Studies

MSc Digital Systems Security

Adversarial machine learning attacks against network intrusion detection systems

Supervisor Professor: Christos Xenakis

Vasileios Pantelakis

Piraeus

21/03/2023

Table of Contents

Table of Figures

Table of Tables

Abstract

Machine learning techniques have become increasingly popular in intrusion detection systems (IDS) due to their ability to automatically learn patterns and behaviors of normal and anomalous network activities. IDS aims to detect and prevent cyberattacks that can potentially cause significant damage to computer systems, networks, and sensitive information. Traditional intrusion detection methods rely on manually designed signatures and rules to identify known attacks, but they often fail to detect novel and sophisticated attacks. Machine learning-based IDS can automatically learn from large volumes of network traffic data and detect anomalies that may indicate an intrusion attempt.

The thesis focuses on the application of machine learning algorithms in developing an IoT intrusion detection system. The study explores various types of machine learning algorithms, including supervised and unsupervised learning, deep learning, and ensemble learning, and discusses how they can be used to detect anomalous activities in IoT networks. The study aims to enhance the accuracy and effectiveness of intrusion detection systems by exploring the robustness of deep learning model to adversarial attacks. The research implements three types of adversarial attacks: Jacobian Saliency Map attack (JSMA), Fast Gradient Sign Method (FGSM), and DeepFool, to evaluate the robustness of the deep learning-based intrusion detection system. The results of the study demonstrate that the deep learning model can effectively detect intrusion attacks in IoT networks with high accuracy although, it highlights the vulnerability of deep learning models to adversarial attacks and the need for developing robust and resilient intrusion detection systems.

1. Introduction

In recent years, the Internet of Things (IoT) has become a vital part of our daily lives, providing us with numerous benefits, including improved efficiency, comfort, and convenience. However, the increased connectivity and reliance on IoT devices have also made them more vulnerable to cyberattacks. To address this issue, intrusion detection systems (IDS) have been developed to detect and prevent cyberattacks. Traditional IDS methods rely on pre-defined signatures and rules, which may not be sufficient to detect novel and sophisticated attacks. Thus, machine learning-based IDS has emerged as a promising approach to automatically detect anomalies that may indicate an intrusion attempt.

In this context, this thesis explores the application of machine learning algorithms in developing an IoT intrusion detection system. Specifically, this study compares the effectiveness of supervised and unsupervised learning techniques in detecting anomalous activities in IoT networks. Furthermore, the research examines the robustness of deep learning-based IDS against adversarial attacks, which can deceive the system and make it classify malicious traffic as benign. The contribution of this study is twofold: firstly, it evaluates the performance of different machine learning techniques in detecting intrusions in the IoT domain. Secondly, it highlights the vulnerability of machine learning models to adversarial attacks and the need for developing more robust and resilient intrusion detection systems.

Moreover, the code used to perform the experiments is publicly available in GitHub Adversarial-Attacks-against-NIDS [34]. The availability of the code ensures the reproducibility of the experiments and allows for the validation and extension of the study's findings. Overall, this thesis presents an in-depth analysis of the application of machine learning techniques in IoT intrusion detection systems, highlighting the challenges and opportunities for developing more effective and robust security solutions.

This study aims to contribute to a deeper understanding of the effectiveness and limitations of machine learning-based IDS in the context of IoT security by answering in the following research questions:

- RQ1: How effective are adversarial attacks against intrusion detection systems in the IoT domain, and what impact do they have on the accuracy and effectiveness of the IDS?
- RQ2: Which adversarial attack has the biggest impact on the performance of intrusion detection systems in the IoT domain?
- RQ3: Which machine learning method (supervised or unsupervised learning) is the most vulnerable to adversarial attacks in the context of IoT intrusion detection?

1.1 Machine Learning

A dream researchers had was to teach computers to reason and make decisions in the way humans do, by drawing generalizations concepts from complex information sets without explicit instructions. Machine learning refers to one aspect of this dream, with processes and algorithms that can learn from past data and experiences in order to predict future outcomes and results. Is a set of mathematical techniques, implemented on computer systems that enables a process of mining information, drawing inferences from data, and pattern discovery.

The term dates back to 1959 when it was first coined by Arthur Samuel at the IBM Artificial Intelligence Labs. In the 1980s, machine learning gained much more prominence with the success of ANNs, Artificial neural networks, and glorified in the 1990s when researchers started using it to solve daily life problems. In the early 2000s, the internet and digitization made it more and more popular, and over the years companies like Google, Amazon, and Facebook started leveraging machine learning to improve the interactions between humans and computers.

1.2 Machine Learning in Cybersecurity

In order to detect threats and anomalies, threat detection systems used static signatures on a large amount of data logs. By doing this, analysts should be able to know how normal data logs look and needed to go through extraction, transformation, and load phase. Data that are transformed are analyzed by analysts who create the signatures. The signatures are then evaluated by passing more data. If an error occurred in the evaluation process they had to rewrite the rules.

Today signature-based systems are being gradually replaced by intelligent cybersecurity agents. Machine learning started to be used for malware detection, zero-day attacks, anomaly detection, and so on. New machine-learning cybersecurity products have been proactive in strengthening systems like virtual machines. In general, these products are created to predict attacks before they occur. Machine learning helps to recognize the attack at its initial stages and prevent it from spreading across the entire organization. Many cybersecurity companies are relying on advanced analytics, such as user behavior analytics and predictive analytics, to identify advanced persistent threats early on in the threat life cycle. Predictive analytics predicts threats by comparing current threat logs with historic threat logs. Prescriptive analytics deals with situations where an attack is already in play and analyzes data to suggest what measure could be best fit for the situation, to have the smallest possible impact.

Although alerts generated need to be tested by the SOC team, false alerts could make humans tired especially if we are talking about a large number of them. One way to solve this problem is with the use of SIEM. Signals from SIEM systems are compared with those from advanced analytics to reduce duplicate alerts and false signals to a minimum.

1.3 Data in Machine Learning

Data is the most important part of all Data Analytics, Machine Learning, and Artificial Intelligence. Without data, we can’t train any model. Data helps Machine learning in detecting patterns and mining data. This data can be in any form and comes in frequency from any source. Big companies are spending lots of money to gather data as certain as possible. We can split data into three categories, the training data, the validation data, and the testing data.

Training data are those we use to train our model to accurately predict an answer or an outcome that we want. This is the data that our model learns from. The validation data is the part of data where we evaluate our model and fit it on the training dataset in order to give an estimate of model skill while tuning the model's hyperparameters (initially set parameters before the model begins learning). The testing data provides an evaluation when our model is completely trained. When we feed in the inputs of testing data, our model will predict some values. After prediction, we evaluate our model by comparing it with the actual output present in the testing data. This is how we evaluate and see how much our model has learned from training data.

1.3.1 Structured and unstructured data

Data can either be structured or unstructured data. When structured it can be easily mapped to identifiable column headers, are in a standardized format, and can easily be accessed by humans and computer programs. Unstructured data can not be mapped to any identifiable data model, has no format or rules, and can not be stored in any logical way.

Structured data is more easily used by machine learning algorithms since is easier to understand when compared to unstructured. Also, manipulation and querying of data become even easier.

1.3.2 Labelled and unlabelled data

Another categorization for data is labeled and unlabelled data. When data has been manually tagged with headers is called labeled and when they are not is called unlabelled.

Unlabelled data is when we know nothing of the data, the environment, or the way in which is collected. We do not know which sensors collected them or the status of the environment in which they were retrieved. So, we do not have any knowledge associated with unlabelled data. Regarding the labeled data, a researcher or an automatic tagger must use their knowledge to add extra information to the data. Data know the way the environment operates. Specifying in the data that, for example, network traffic is an attack, is labeled. When we mention what kind of attack this is, for example, DDoS, it is also labeled.

2. Types of Machine Learning algorithms

Machine learning systems can be mainly categorized into two types, supervised approaches, and unsupervised approaches, based on the types of learning they provide, figure 1. The differences between these learning types are attributable to the type of result that we intend to achieve.

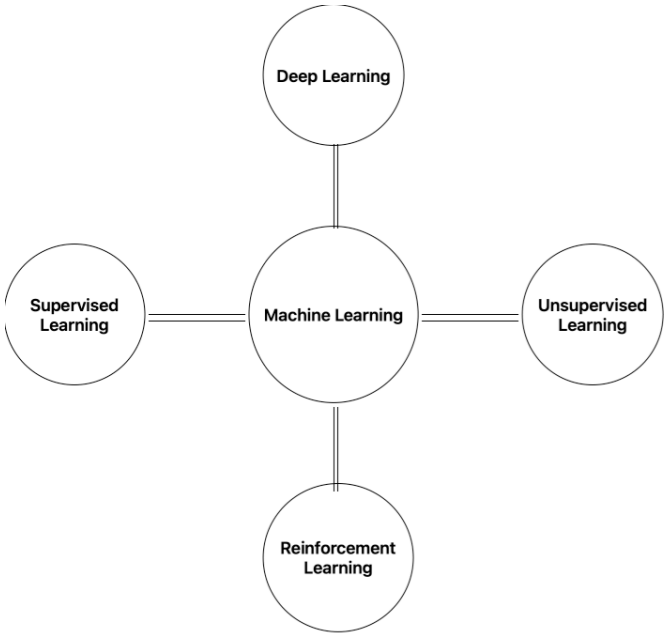

Figure 1. Types of Machine Learning

2.1 Supervised learning

Supervised learning is when we train our model using data that are labeled. The training of the algorithm is conducted by using an input dataset where the type of output that we want to obtain is already known. What we want from the algorithm is to be able to identify the relationship between the variables being trained and make an optimization of learning parameters on the basis of the labels. After that, the model is provided with a new set of data so the machine learning algorithm analyzes the training data and produces a valid result from labeled data. Supervised machine learning helps solve various types of real-world problems although classifying the data and training process can take a lot of time.

An example of a supervised learning algorithm is the classification algorithm which can be used for spam classification. A dataset containing many examples of emails that have been classified as malicious or spam or genuine is provided, for training, in the spam filter. A classification problem is when the result is a category such as “DDoS” or “Malicious”. Another example of supervised algorithms is regression algorithms. A regression problem is when the result is a real value, like “weight”. The following are the main supervised algorithms:

- Regression (linear and logistic) [1]

- k-Nearest Neighbors (k-NNs)

- Support vector machines (SVMs)

- Decision trees and random forests Neural networks (NNs)

2.1.1 Decision Tree

Decision trees are supervised learning models that are easy to interpret [2]. As its name suggests, decision tree is a binary tree data structure that is used to make a decision. Being a very popular choice for machine learning, even outside of these fields, decision trees have the ability to predict both categorical and real values such as classification and regression trees respectively. They can also take in numerical and categorical data without any normalization.

The first step in the learning decision construction is to split the dataset based on a binary condition into two child subsets. Then, the child subsets are partitioned into smaller subsets based on other conditions. We calculate how much accuracy each split will cost us and the split with the least cost is chosen. There are some common metrics where we can measure the quality of the split since are automatically selected at each step depending on the condition that best splits the dataset.

One metric is the information gain which measures the purity of the subsets that we have after a split. We can calculate this by subtracting the weighted sum of each decision tree child node's entropy from the parent node's entropy. In this case, the split is better when the entropy of the children is smaller thus the information gain is greater. Variance reduction is another metric that defines the total reduction in variance as we split into two subsets. We will have the best split in a decision tree when it results in the greatest variance.

We do not care only about the splitting method but also to understand when to stop. If each node contains samples that belong to the same class or when the maximum depth of the node is reached the splitting is stopped. Also, there is a case where child nodes will contain samples that are fewer than the minimum number and the node will not be split, figure 2.

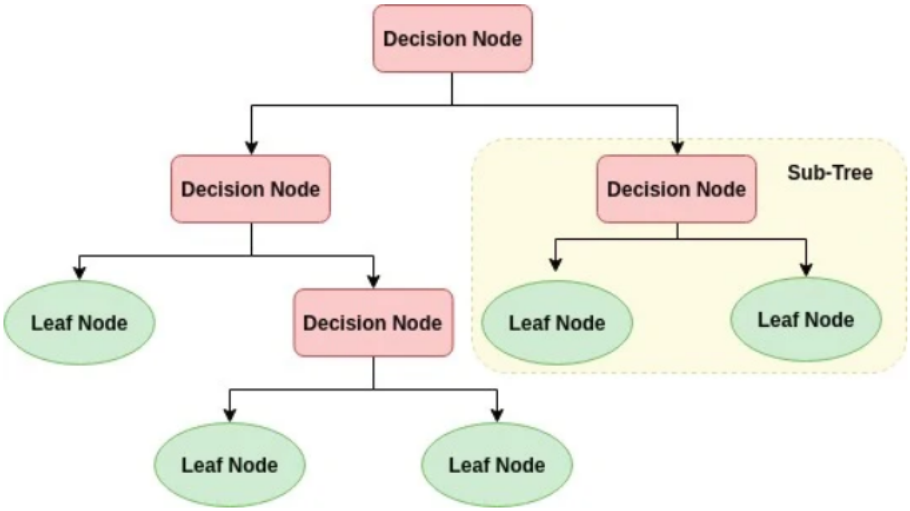

Figure 2. Decision Tree [1]

We can easily explain the classification or regression results of decision tree, since every prediction can be expressed in a series of boolean conditions that starts from the root node of the tree to a leaf node. The root node is from where the decision tree starts and leaf nodes are the final output node and the tree can not be segregated further after getting a leaf node.

They have great performances in large datasets since they are very efficient for training and making predictions. Although, decision trees have also their limitations. They often suffer from the problem of overfitting as decision-tree learners can create over-complex trees that do not generalize well beyond the training set. Also, decision trees are less accurate and robust compared to other supervised learning techniques as small changes to the training dataset can result in large changes to the tree which change the model's predictions. This is called variance, which needs to be lowered by methods like bagging and boosting. This makes decision trees unsuitable for online learning.

---

1 . Decision Tree figure from https://devopedia.org/decision-trees-for-machine-learning

2.1.2 Support Vector Machines

Support vector machines, known as SVM [3], is a linear classifier which means that it produces a hyperplane in a vector space that tries to separate the two classes in the dataset. The best decision that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category is called a hyperplane. The same goes for the Logistic regression. It uses a hinge loss, which penalizes only the points that are found on the wrong side of the hyperplane or very near to the correct side. In contrast, the Logistic regression which uses a log-likelihood function that penalizes all the points in proportion to the error in the probability estimate. SVM is one of the most popular supervised learning algorithms which is used for classification but also for regression problems, but mostly classification.

SVM classifier has as a goal to find the maximum distance from the separating plane to the closest data points on each side which separates the two classes. These cases are called support vectors. When we have data that is not linearly separable, the points within the margin are penalized proportionately to their distance from the margin. We can understand how the SVM classifier works by an example. In the following figure 3 shows two classes that are represented by white and black points.

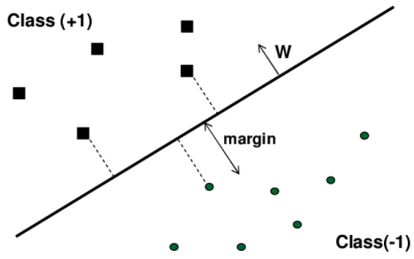

Figure 3. SVM separates two classes [2]

---

2. Figure from "Active learning to improve the detection of unknown computer worms activity" - Robert Moskovitch

The straight line is the dividing line, hyperplane, and the dotted lines are the margins. Since there can be many lines that can separate the classes, SVM can help to find the best line. Thus, the algorithm initially finds the closest points of the two classes on the line, which we call support vectors, and their distance from the line is called the margin. The hyperplane with the maximum margin is the optimal hyperplane.

In this example, we are referring to linearly separable data. When the data is non-linear, the SVM solves this problem by creating a new variable using a kernel. The SVM kernel is a function that takes a low dimensional input space and transforms it into a higher dimensional space, i.e. it transforms a non-separable problem into a separable one. It finds the process, after it has done some complex data transformations, and separates the data according to their labels or outputs defined.

SVMs have a very good performance in practice, especially in high-dimensional spaces. Also, since they can be described in terms of support vectors, a subset of training points in the decision function, this results in us having memory-efficient implementations for scoring new data points. However, when we train a kernelized SVM the complexity grows quadratically with the number of training samples. So, with large training set sizes kernels are rarely used and the decision boundary is linear. Another disadvantage is that the scores output by SVM is not interpretable as probabilities and converting them to probabilities requires additional computation and cross-validation.

2.2 Unsupervised learning

In unsupervised learning, we do not have the classification provided by the analyst and the algorithm must try to classify the data on one's own and unassisted. In cybersecurity unsupervised learning algorithms are very important for identifying new

malware attacks or email spamming. The machine has to group unsorted information, unlabeled data, from patterns and differences without any prior training data and the help of experience. These are more complex processes since the system learns by itself without any intervention. In contrast to supervised learning where we use labeled data, unsupervised learning, also known as self-organization allows for the modeling of probability densities over inputs. Also, unsupervised learning can work with real-time data to identify patterns. Some disadvantages are that it is not always certain that the obtained results will be useful as there is no label and often have lesser accuracy. It is also costlier since it might require human intervention to understand the patterns and correlate them.

Clustering is a category of unsupervised learning. Clustering is the process of grouping data that has not been labeled, classified, or categorized and putting similar data into the same group [4]. The clustering techniques are most popular in pattern recognition and information retrieval. These techniques use data parameters and go through many stages before they can group the data. Cluster analysis identifies commonalities in the data and reacts based on the presence or absence of such commonalities.

There are four clustering algorithms which are exclusive clustering, overlapping clustering, hierarchical clustering, and probabilistic clustering. The exclusive clustering data are grouped in an exclusive way where if a data point belongs to a definite cluster then it could not be included in another cluster. In case of overlapping, it uses unclear sets to cluster data so that each point can belong to more than one cluster with different characteristics. In this situation, the data will be associated with an appropriate subscription price. A hierarchical clustering algorithm is based on connecting the two nearest clusters. We start by setting every data point as a cluster and after a few iterations it reaches the final clusters that we want. For the probabilistic, the cluster includes data objects that have a higher probability to be in it.

2.2.1 K-means

One of the most used clustering algorithms is K-means [5]. The goal of this unsupervised learning algorithm is to assign each data point to a cluster such that the sum of distances from each point to its cluster centroid, cluster centers, is minimized. When we know how many clusters we expect, we define this number of clusters to *k*. Now, when we talk about the distance we are referring to Euclidean distance in a vector space. The k-means computes a cluster assignment that minimizes the loss function:

$$L(X) = \sum_i d(x_i, c_{f(x_i)})$$

where X = $\{x_1,\ldots, x_n\}$ is our dataset, the $c_{f(x_i)}$ is the centroid in f (x i) and the d is the distance between two points. The value L(X) is called inertia. The way in which the kmeans clusters are computed is by first selecting the k centroid, assigning each data point to the closest centroid, and again computing the c centroids by taking the average of all the data points that were assigned to the cluster. Finally, we repeat this process until the difference of L(X) in successful iterations is below a predetermined threshold. That way each cluster has a data point with some commonalities and is away from other clusters. So, we can just say that the algorithm takes the unlabeled dataset as input, and divides the dataset into k clusters until it can not process better clusters, figure 4.

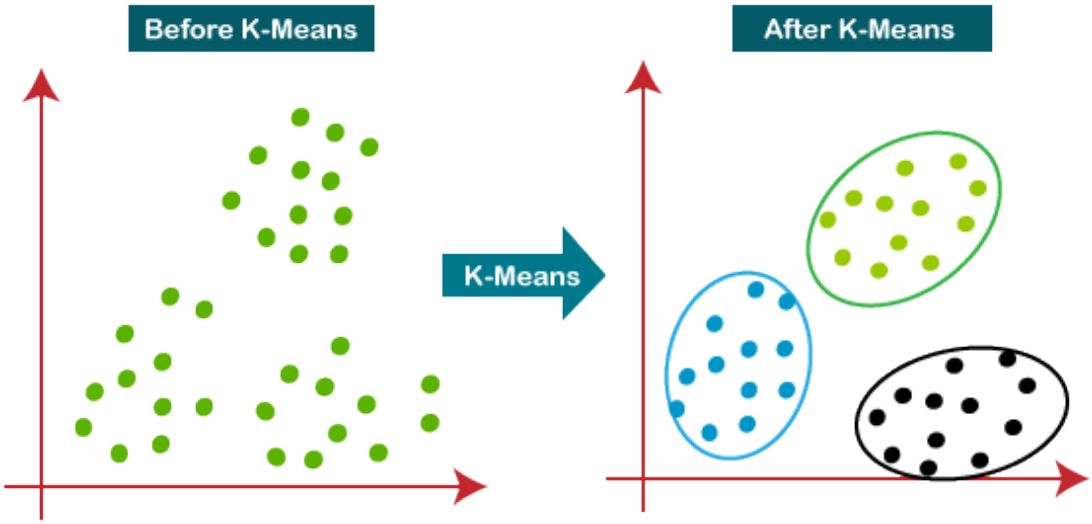

Figure 4. K-means [3]

Kmeans has multiple applications such as document clustering, image segmentation, image compression, etc. Before implementing it, we should bear in mind that it is proposed to normalize our data since such algorithms do the distance-based measurement and that due to the initialization of centroids of kmeans there is a possibility that it will get stuck in a local optimum and fail to converge to the global optimum and so we should employ distinct centroids initializations.

K-means is a simple and very effective clustering algorithm although we should keep some things in mind. We should normalize our data before using k-means as if we had a two-dimensional dataset where the first coordinate has a range from 0 to 1 and a second one that has a range from 0 to 100, surely the second one will have a much greater impact on the loss function. Also, when using binary features results can be unpredictable as they can become the dominant feature determining the cluster or we could lose all the information. K-means takes for granted that the clusters are spherical so it does not work well on non-spherical distributions.

---

3. Figure from "K Means Clustering Simplified in Python" - https://www.analyticsvidhya.com/blog/2021/04/k-means-clustering-simplified-in-python/

As mentioned previously, the choice of the optimal number of clusters is a serious task. As the performance of the K-means clustering algorithm depends significantly on the efficient clusters it forms, there is a way we can find the ideal number of clusters or the value of k. One of the most well-known methods is the elbow method. This method uses the logic of Within Cluster Sum of Squares (WCSS) which defines the total variations in a cluster. The way it works is that it first performs K-means clustering on a dataset that we give with different K values, from 1 to 10, then for each k the WCSS value is calculated and we make a plot, a curve, between the calculated WCSS values and the number of k. At the point of the curve where a corner is created, in figure 5, if we liken the curve to a hand then at the point of the elbow, then that point is considered the ideal value for k [6].

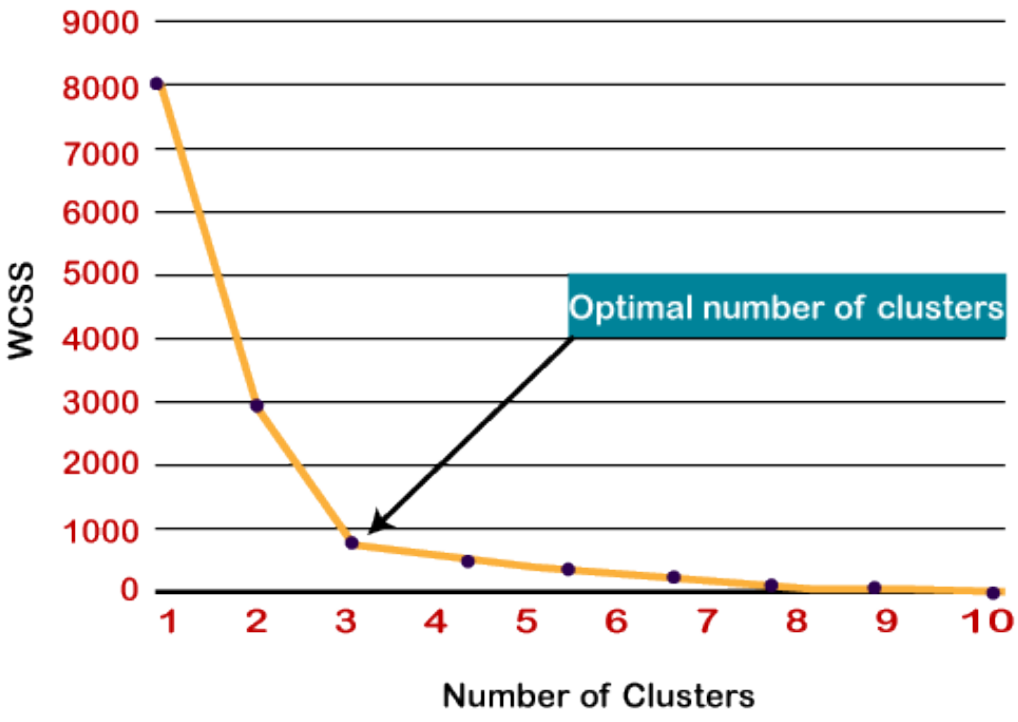

Figure 5. Elbow method for optimal number of k [4]

4. *Figure from* https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning

2.3 Deep learning

Deep learning is another popular term that is commonly conflated with machine learning. Deep learning is a strict subset of machine learning referring to a specific class of multilayered models that use layers of simpler statistical components to learn representations of data. “Neural network” is a more general term for this type of layered statistical learning architecture that may have or not have many layers, that why it is deep [7].

Deep learning is a class of machine learning which in turn is a core building block for Artificial intelligence (AI), figure 6. AI is a popular but loosely defined term that indicates algorithmic solutions to complex problems typically solved by humans. It's like enabling computers to mimic human behavior. Below we can see the correlation of AI with machine learning and deep learning.

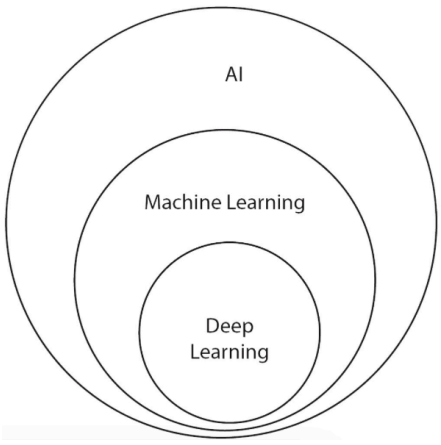

Figure 6. AI related to Machine learning and Deep Learning [5]

---

5. *Figure from* *https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781789802993/1/ch01lvl1sec02/the-relationship-between-ai-machine-learning-and-deep-learning*

Humans arrive at some conclusions by analyzing some data with some logical structure, so can deep learning also achieves this with neural networks, i.e. using a multi-layered structure of algorithms. as we mentioned, neural networks try to mimic the human brain, so their design structure is based on the human brain. With neural networks, we can perform many tasks. To group or sort unlabelled data, to train a network on a labeled dataset to classify the data into different categories, but also clustering. These capabilities of neural networks can make Deep learning able to solve problems that a machine learning model cannot.

Today, deep learning is used in many applications such as Google's voice and image recognition, Netflix, Amazon or Youtube recommendation engines, Apple's Siri, Alexa, automatic email and text replies, and chatbots. Without deep learning, we would not have self-driving cars, fraud detection, health care, adding sound to silent movies, automatic machine translation, text-to-image translation, image-to-image synthesis, image colorization, earthquake prediction, market rate forecasting, news aggregation, and fraud news detection. All recent advances in artificial intelligence are due to Deep learning.

The increase of high-performance computing has a lot to do with the increasing popularity of Deep learning. When dealing with unstructured data Deep learning achieves higher power due to its ability to process a large number of features. Deep learning algorithm passes the data from several layers where each one extracts features which in turn were passed to the next layer. Another reason they are so popular is that feature extraction is not needed. When we use algorithms such as Decision tree or SVM they cannot be applied directly to raw data and one more preprocessing step is needed, the feature extraction. After that, the given raw data can be used. In the case of Deep learning, this extra step is not needed. The layers are able to learn on their own an implicit representation of the raw data. Through the several layers, an abstract and compressed representation of raw data is produced which in the end produces the result. During the training process, this step is also optimized by the neural network in order to have the best possible abstract

representation of the data. So, in a Deep learning model, it is not required any effort to perform and optimize the feature extraction. We can say that it is already part of the process that takes place in a neural network.

As we said the term “deep” usually refers to the number of layers in the neural network. The typical neural network architecture consists of several layers, figure 7. The first layer is the input layer which receives the data that the neural network learns from. The last layer is called the output layer which outputs a vector representing the result that the neural network came up with. For example, if we have classification the number of neurons in the last layer would be the number of classes. The prediction vector is obtained by a number of mathematical operations that are performed in the layer between the input and the output layers, called hidden layers. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

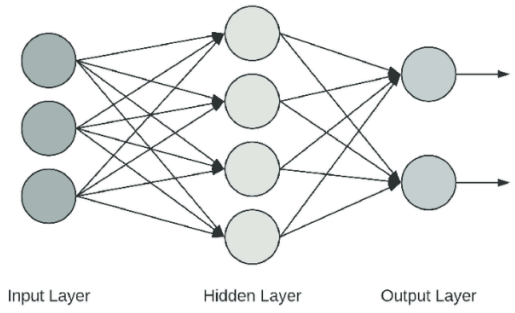

Figure 7. Artificial Neural Network [6]

A neural network can be described as a set of connected nodes which are called neurons. Connections between the neurons are made with so-called weights. When a neural network learns the weights, numeric values, between the neurons are changing and with that, it also changes the strength of the connection.

---

6 . Figure obtain from ESTIMATION WITH ARTIFICIAL NEURAL NETWORK ON ELECTRONIC WORD OF MOUTH - Ibrahim Topal

For every action taken, we need a certain set of weights that will allow the neural network to perform, for example, a classification. The set of weights is different for every task and every dataset. During the training process, we obtain these weights and it is impossible to predict them in advance.

2.4 Ensembling

Rather than using individual classifiers, in presence of non-stationary data, it may be useful to apply an ensemble of classifiers in order to improve overall prediction accuracy. For that reason, ensembling learning aims to combine different classifiers to get a final classifier that will have better predictions than what we would get from an individual classifier. Let's think that we have a number of binary classifiers, all of the same type, they could make a correct prediction in 75% of the cases and not so accurate in the remaining 25% of the cases. By using an ensemble classifier the probability of getting correct results is improved and accordingly the probability of error is reduced.

Ensemble learning combines the mapping functions learned by different classifiers to generate an aggregated mapping function. There are several methods that we can use to combine classifiers, the most popular that commonly used are Bagging, Boosting, Stacking, and Majority Voting.

The bagging ensemble technique [8], also known as bootstrap aggregating, is one of the earliest ensemble methods proposed. It is a method that has the property of parallel processing. Using replacements, random subsets of a dataset are created and we call them bootstrap sampling. Now, these subsets are treated as independent datasets where several Machine learning models will be fit. The predictions from all models are collected and using an aggregation mechanism we will compute the final prediction. What we want to achieve with the bagging method is to reduce variance in the ensemble predictions. So, the chosen ensemble classifier usually has high variance

and low bias. Random forest [12] is an example of bagging with additional features in the learning process.

Unlike bagging which is a parallel ensemble method Boosting method [10] are sequential ensemble algorithm. At first, our first classifier is fed with the entire dataset and makes the predictions. After that, where the first classifier failed to produce correct predictions is fed to the second classifier. In this way, the second classifier will learn an appropriate decision boundary focusing only on the problematic ones. We continue in the same way until we finally compute the ensemble of all classifiers and make the final prediction. The chosen ensemble classifier in this case usually has low variance and high bias. An algorithm based on this approach is the Gradient Boosting Machines [13].

In the Stacking [11] ensemble method, as in the case of the bagging mechanism, we have the creation of bootstrapped data subsets. The difference here, however, is that the outputs of all the models are used as input to another classifier called a meta-classifier, which in turn makes the final prediction. The reason this stacking of the two classifiers is done is to determine if the training data have been appropriately learned. If the first classifier has issues recognizing some classes, the meta-classifier can then correct this behavior before making the final prediction. We can also add more layers of classifiers in the stacking ensemble method, although this will result in a very expensive computation without having any significant performance boost.

The Majority Voting method [9] refers to the fact that from predictions made by individual classifiers, we select the one that shows the highest frequency. It is one of the earliest ensemble schemes. An odd number of classifiers are chosen and predictions of each one are computed. Then, the class that has achieved the highest frequency is considered the predicted class of the ensemble. If we have, for example, a binary classification problem where we will have only two candidates for whom the classifier can vote, then Major voting is quite effective. However, when we have several classes, quite a problem arises since it is very difficult for one class to receive the total number of votes.

3. Intrusion Detection System

As people are using more and more intelligent systems in their daily lives the risks around cyberattacks have increased significantly. We are in a current state where traditional security measures have started losing their effectiveness since connections between different devices, such as the Internet of Things (IoT), have reached a very high level of complexity. Thus, the need for something new for detecting network anomalies was created[19]. To ensure safe communication Intrusion Detection Systems (IDS) were developed using machine learning algorithms that have the ability to detect attacks against network security. IDS is the continuous monitoring of the traffic for unusual events and critical characteristics and raises an alarm that alerts the defenders when it determines any intrusion, a threat, in the network. It is also useful as an alternative to traditional firewalls. Intrusion detection systems have a long history in the defense of networks, although machine learning came to assist and improved the accuracy, performance, and the discovery of existing or new attacks[18].

A network-based IDS monitors and detect threats that find through the communication that travels into and out of the network. A host-based IDS makes its scans in a particular server. In the area of network intrusion detection, we can find two different types of IDS Signature-based IDS and Anomaly-based IDS. The signature-based IDS analyzes network traffic for already known malicious signatures from attacks that were previously detected [20]. Although, having a strategy with common antivirus software its disadvantage is that requires a constant update of these known signatures in order to work efficiently and to be able to detect new types of attacks. The Anomaly-based IDS learns from existing data and compares the network traffic against a user's known patterns to raise an alert [21]. So, it identifies a network traffic behavior as normal and detects differences in behavior as anomalous. With this

approach, it is possible to detect new types of attacks as it is analyzing the traffic that deviates from the pattern. Some common patterns that can be taken into consideration are the number of connections from a specific IP, unusual communication ports, communication bandwidth from particular hosts, etc. All these events may be considered as suspicious if we compare them with normal traffic ones. This approach is more widely used in the design of intrusion detection applications. Machine learning is used to develop the anomaly-based IDS since it works on the principle of training the model with known data points and testing it with benign samples. In our implementation, we will create a Network Intrusion detection system using neural networks.

4. IoT Intrusion Dataset

The dataset we used to train the machine learning models is the IoTID20. It is a dataset that generated by Hyunjae et. al [22] which refers to anomalous activity in IoT. We will develop our intrusion detection system in IoT networks.

4.1 Internet of Things (IoT)

The Internet of things, or IoT, [22] is a system of connected devices, provided with unique identifiers, that communicate with the internet and do most of the work without requiring human intervention. Although people can interact with the devices by setting them up, giving them instructions, or accessing the data. Everyday devices like cars, vacuums, and lights are collecting and sending data that acquire from their environment with their sensors, communication hardware, or processors and respond to users in a smart way. IoT becomes an important technology to develop smart infrastructure since organizations operate in a more efficient way, improve decision-making and achieve consistent and effective operations, reducing operational costs significantly. Home automation can use IoT to monitor mechanical and electrical devices in a building. Smart cities can tackle problems regarding infrastructure, health, and the environment. Reducing energy with smart lights, detecting maintenance needs in streets or bridges, or reducing driver indolence with efficient parking management.

Although all these implementations provide a large attack surface for intruders to attack and exhaust the IoT network with malicious activity. The difference with a traditional network is that IoT devices operate without manual supervision. Their wider adoption in our lives makes it very critical to address the security threats before

their deployment. IoT attacks can be physical, network, software, or encryption. In network attacks, we can have the Man in the Middle attacks where the attacker intercepts the communication between two devices and obtain sensitive information [23]. Denial of service where the attacker floods the network with large traffic and makes the devices unavailable [24]. Since the impact of these attacks is huge we can not work with traditional detection systems. As mentioned in the previous section, a signature-based IDS will not have a big effect since it would be able to identify new attacks and we want them to be stopped before their implementations. An anomaly-based IDS is well suited to the current environment since it could detect zero-day attacks. Data mining and machine learning help develop the capabilities of an anomaly-based IDS. In order to have an effective IDS we will need a new sophisticated dataset. The IoTID20 dataset will help us in detecting malicious activity in the IoT network.

4.2 IoTID20 dataset

The IoTID20, generated from the dataset [21], has a more comprehensive network and flow-based features which will provide a reference point to identify anomalous activity in the IoT network. In order to generate the dataset, a smart home environment was implemented having two smart devices, a Wi-Fi camera EZVIZ and a smart home device SKT NGU, and some laptops, tablets, and smartphones. We consider the two smart devices as the IoT victim devices and all other devices in the environment are considered the attacker's devices. The dataset contains 8 types of attacks to evaluate the Intrusion Detection system in IoT Networks. The way in which the dataset was created, and more specifically the dataset in CSV format, is by starting with the Pcap files from [1]. Then, the CICflowmeter application was used to extract the features and generate the CSV file. The IoTID20 dataset consists of 83 network

features and three label features. The three label features are the Binary, category, and sub-category. In binary, we have normal and anomalous traffic. In Category, we have the five attacks, Normal, Dos, Mirai, MITM, and Scan. And in Subcategory, we have subcategories of the attacks which are Normal, Syn Flooding, Brute force, HTTP Flooding, UDP Flooding, ARP Spoofing, Host Port, and OS. The IoTID20 is one of the few publicly available IoT intrusion detection datasets which replicates the IoT network communication. In the below tables 1, 2, and 3 we can find the binary, category, and subcategory instances distributions.

Binary label distribution	
Type	Instances
Normal	40073
Anomaly	585710

Table 1. Binary label distribution

Category label distribution	
Type	Instances
Normal	40073
DoS	59391
Mirai	415677
MITM	35377
Scan	75265

Table 2. Category label distribution

Subcategory distribution	
Type	Instances
Normal	40073
DoS	59391
Mirai Ack Flooding	55124
Mirai Brute Force	121181
Mirai UDP Flooding	55818
MITM	35377
Scan Host Port	22192
Scan Port OS	53073

Table 3. Subcategory label distribution

Features							
Flow_ID	Src_IP	Src_Port	Dst_IP	Dst_Port	Protocol	Timestamp	Flow_Duration
Tot_Fwd_Pkts	Tot_Bwd_Pkts	TotLen_Fwd_Pkts	TotLen_Bwd_Pkts	Fwd_Pkt_Len_Max	Fwd_Pkt_Len_Min	Fwd_Pkt_Len_Mean	Fwd_Pkt_Len_Std
Bwd_Pkt_Len_Max	Bwd_Pkt_Len_Min	Bwd_Pkt_Len_Mean	Bwd_Pkt_Len_Std	Flow_Byts	Flow_Pkts	Flow_IAT_Mean	Flow_IAT_Std
Flow_IAT_Max	Flow_IAT_Min	Fwd_IAT_Tot	Fwd_IAT_Mean	Fwd_IAT_Std	Fwd_IAT_Max	Fwd_IAT_Min	Bwd_IAT_Tot
Bwd_IAT_Mean	Bwd_IAT_Std	Bwd_IAT_Max	Bwd_IAT_Min	Fwd_PSH_Flags	Bwd_PSH_Flags	Fwd_URG_Flags	Bwd_URG_Flags
Fwd_Header_Len	Bwd_Header_Len	Fwd_Pkts/s	Bwd_Pkts/s	Pkt_Len_Min	Pkt_Len_Max	Pkt_Len_Mean	Pkt_Len_Std
Pkt_Len_Var	FIN_Flag_Cnt	SYN_Flag_Cnt	RST_Flag_Cnt	PSH_Flag_Cnt	ACK_Flag_Cnt	URG_Flag_Cnt	CWE_Flag_Count
ECE_Flag_Cnt	DownUp_Ratio	Pkt_Size_Avg	Fwd_Seg_Size_Avg	Bwd_Seg_Size_Avg	Fwd_Byts/b_Avg	Fwd_Pkts/b_Avg	Fwd_Blk_Rate_Avg
Bwd_Byts/b_Avg	Bwd_Pkts/b_Avg	Bwd_Blk_Rate_Avg	Subflow_Fwd_Pkts	Subflow_Fwd_Byts	Subflow_Bwd_Pkts	Subflow_Bwd_Byts	Init_Fwd_Win_Byts
Init_Bwd_Win_Byts	Fwd_Act_Data_Pkts	Fwd_Seg_Size_Min	Active_Mean	Active_Std	Active_Max	Active_Min	Idle_Mean
Idle_Std	Idle_Max	Idle_Min	Label	Cat	Sub_Cat		

Table 4. IoT dataset features

5. Adversarial Attacks

Deep learning was able to solve difficult problems that traditional machine learning techniques could not. With their evolution and the availability of high-performance hardware to train complex models, its usage increased radically in day-to-day applications. Having great accuracy, deep learning models had a big impact on AI-based services on the Internet, including Google and Alibaba. Although these systems are considered secure, network proliferation makes them vulnerable to many external threats and their security and integrity pose a great concern. In today's digital world, people are adopting AI systems into their daily lives and the cyberattacks on their user-specific information have grown.

More specifically, the security of deep learning systems is vulnerable to adversarial attacks that craftily manipulate legitimate inputs, by adding careful perturbation to the data known as adversaries and forcing a trained model to produce incorrect results. Slightly perturbed input can fool the Deep neural network into making misclassified outputs with high confidence [14]. These attacks also became very critical and raised safety concerns when we are talking about DNN models that apply in applications such as autonomous driving [15] and medical diagnosis [16]. Szedy et al., [17] first discovered that deep neural networks are prone to adversarial attacks as they map inputs to outputs which could be intermittent based on the data used. Adversarial samples can reduce the trust of a classifier, force the classifier to generate results that resemble the targeted output class, and adds noise to output in order to misclassify it as another class.

The attack generation mechanisms can fall into different categories. One of these categories is what access the attacker has to the model. We have White-Box attacks and Black-Box attacks. In a White-Box scenario, the attacker has some information regarding the model or its training data, like, machine learning algorithm, model parameters, network structure, etc. This information allows him to exploit the gradient of the loss function to form an adversarial sample. In a black box scenario,

the attacker has no knowledge about the model, training data, or parameters. The adversary although has the ability to probe the model with a series of carefully crafted inputs to observe the outputs.

Another category has to do with which phase of the model the attack will be implemented. We have attacks in the training phase where the attacker attempts to learn the model by accessing a summary, partial, or all of the training data. And attacks in the inference phase where the attacker collects information and evidence about the model characteristics by observing the inferences made by it. Finally, we have the Passive and Active attacks. In a Passive attack, the attacker passively observes the model and its updates, without making any changes to the training procedure, and performs inference Regarding the Active attack, the attacker changes the way the model operates having an impact on its results.

In our case, the neural networks, that we will use to create our intrusion detection system, are also prone to adversarial attacks as they map inputs to the outputs which could be intermittent based on the data used. Several techniques are developed to craft adversarial samples which can be used to dodge the detection capabilities of a system. In our implementation, we used two techniques, white-box methods, the Jacobian Saliency Map Attacks (JSMA), and Fast Gradient Sign (FGSM) algorithms.

5.1 Jacobian Saliency Map Attack

Papernot et al. [26] proposed an algorithm to craft adversarial samples using the Jacobian Matrix the Jacobian Saliency Map Attacks (JSMA). The way the JSMA algorithm works is that a direct mapping is established from the input vector, A, to the

desired output, B, which then generates adversaries. We can have an activation function F which is A -> B. An adversary, A*, can be generated from the following mathematical optimization problem:

$$\arg\max_{\sigma_a} \|\sigma_a\| \text{s.t.} F(A+\sigma_a)=B^*, \quad (1)$$

Where we have:

- σa as the perturbation vector,

- $\|\,.\,\|$ the relevant norm for Neural Networks input comparison,

- B^* the adversarial output data points,

- $A + \sigma_a = A^*$ the adversarial sample.

Our goal is to generate adversaries that should be almost similar to the original sample but our Neural Network model misclassifies it. So, if our original dataset is F(A) = B what we want is F(A*) = B* which is different from B. Since σa is the perturbation that creates the adversarial sample of output class B* forward derivative is computed for each feature.

To generate an adversarial example from the original input, JSMA first computes the gradient Z(x) for a saliency map. This is the Jacobian Matrix of a given function which is learned during the training phase. If we have a one-dimensional vector space we have the following equation:

$$\nabla F(A) = [\frac{\delta F(A)}{\delta d_1}, \frac{\delta F(A)}{\delta d_2}] \quad (2)$$

Taking forward derivate reduces the adversarial data search space and demonstrated the amount of change that happened in the original features. So, it uses a saliency map that will show the impact each data has on the classification result. It finds the most salient component that will be changed (saliency map: input perturbations -> output variations). For example, If we are referring to a picture as input, it chooses and changes the most likely pixel that makes the largest increase, which means the largest gradient.

$$S[A,t][i] = \begin{cases} 0 \text{ if } \dfrac{\delta Ft(A)}{\delta d_i} < 0 \text{ or } \sum\limits_{k=/t} \dfrac{\delta Fk(A)}{\delta d_i} \\ \dfrac{\delta Ft(A)}{\delta d_i} \cdot \left| \sum\limits_{k=/t} \dfrac{\delta F(A)}{\delta d_i} \right| \text{ otherwise.} \end{cases} \quad (3)$$

Where we have the S[A, t][i] being the attacker's target class, the upper part is the when feature moves away from the target or towards other labels, and in the lower part measures how much output moves towards the target. Therefore, we have the maps

$\dfrac{\delta Ft(A)}{\delta d_i}$ and $\sum\limits_{k=/t} \dfrac{\delta Fk(A)}{\delta d_i}$ that quantify how much F(A) will increase given an alteration modification of input A. The iterations continue until it succeeds in output label changes or when the maximum allowed number of iterations is reached.

5.2 Fast Gradient Sign Method (FGSM)

Fast Gradient Sign Method is a very simple and efficient method of generating adversarial samples. Goodfellow et. Al [27] proposed a fast gradient sign methodology that calculates the gradient of the cost function with respect to the input of the neural network. They posited that linear behavior in high-dimensional input spaces are capable to cause adversarial perturbations and in order to find them we estimate the dimensions of the input space that are most sensitive to class change. This is where fast gradient methodology takes place. When it comes to training neural networks, gradients are how we determine the direction in which to push our weights

to reduce the loss value. With the intention of maximizing the error of the network, the input is modified by changing the values of these dimensions in the opposite direction of the gradient to maximize the loss function. A way to determine the direction in which to adjust a weight deep in the network is by back-propagating the gradients from the output layer to the weight. The adversarial examples are generated using the following equation:

$$X_* = X + e * \text{sign}(\nabla_x J(X, y_{true}))$$

Where we have:

- J as the cost function of the trained model
- ∇_x indicates the gradient of the model with respect to the normal sample X
- y_{true} the correct label
- *e* indicates the input variation parameter which controls the perturbation's amplitude.

Normally, in a neural network in order to nudge the weights to decrease the loss value we use the following:

New weights = old weights - learning rate * gradients

For FGSM we want to maximize the loss so we nudge the weights accordingly in the following equation:

New weights = old weights + epsilon * gradients

A difference between the two above equations is that one has addition and the other has subtraction. By using the addition in equation two we nudge the wights in the opposite direction from the direction that minimizes the loss. Regarding the epsilon, if for example, we have as input an image, the degree of the noise in the resulting image depends on the epsilon parameter. The larger the value the more intense the noise.

There are also some variations of FGSM, where we have the targeted and basic methods. The Target Class method maximizes the probability of a target class.

Rather than mislabel a class, a sample can be misclassified to a specific label by calculating the J with reference to the target label. Next, we add the negative of that result to X.

$$X_* = X - \varepsilon * \text{sign}(\nabla_x J(X, y_{\text{target}}))$$

The Basic Iterative method is a straightforward extension of the basic FGSM which in this case generates adversarial examples iteratively using a small step size. Since in this case, we are applying FGSM in multiple iterations instead of one, in order to reduce change we use the aT = ε in each iteration. For the new weight to be within the epsilon (ε) max normal of the original input, we apply clipping[28].

$$X^0 = X;\ X^{n+1} = \text{Clip}_{X,e}\ \{X^n + \alpha * \text{sign}(\nabla_x\ J(X^n, y_{\text{true}}))\}$$

Where, α is the step size and $\text{Clip}_{X,e}$ indicates the clipping of X. Most of cases this method does not rely on any approximation of the model and produces additional adversarial examples, one with the largest change in cost, when run for more iterations.

5.3 DeepFool

Moosavi-Dezfoll et al. [32] introduced Deepfool an efficient method to compute the minimal perturbations to cause a classifier to misclassify an input. DeepFool is an iterative and efficient algorithm for generating adversarial examples in deep neural networks. It specifically aims to find the smallest perturbation that can cause a misclassification, making it a "minimal" attack.

The basic idea behind DeepFool is to iteratively compute the direction in which a sample needs to be perturbed in order to misclassify it. At each iteration, the algorithm computes the linearized decision boundary of the current model at the current sample. The perturbation is then set to be in the direction of the decision boundary but with a magnitude that is minimized subject to the constraint that the

resulting perturbed sample is misclassified. This is done by solving a linear optimization problem that finds the minimum distance between the current sample and the decision boundary, subject to the constraint that the perturbed sample is misclassified. The DeepFool algorithm repeats this process until a misclassification is achieved or a maximum number of iterations is reached. The result is a perturbed sample that is misclassified by the model, with the perturbation being as small as possible.

For binary Classifiers, as we see in figure 8, the algorithm works by calculating the gradient *w* of the loss function for the input and then dividing the output prediction of the network *f(xo)* by the L2-norm of that gradient. This gives a scalar value that is used to determine the size of the perturbation. The perturbation is then multiplied by the unit vector of the gradient *w* and its sign is inverted so that the loss of the classifier *f* is increased. This process is repeated iteratively by adding the previous perturbation to the next perturbation until the label changes or a maximum number of iterations is reached. To avoid convergence close to zero, the algorithm uses a parameter called overshoot *n*, which increases the size of the perturbation beyond zero to ensure a label change.

$$r*(x0) = \frac{f(x0)}{\|w\|2} w$$

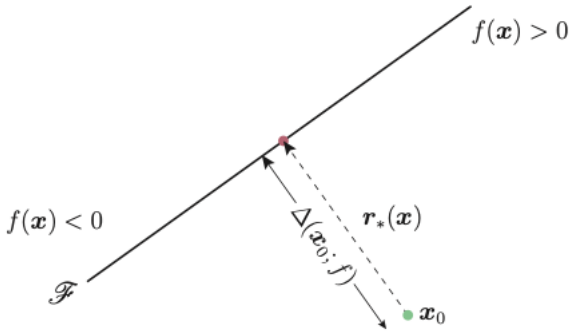

Figure 8. DeepFool for Binary Classifier [7]

---

7 . From https://arxiv.org/pdf/1511.04599.pdf

For a multi-class classifier, the algorithm treats each class as a binary classifier. The decision boundary for each class is represented as a polyhedron, which is formed by the intersection of multiple hyperplanes, as shown in figure 9. To find the minimum perturbation needed to cause misclassification, the algorithm computes the differences between the classifier outputs and gradients for each class and the original predicted class. This difference is used to determine which hyperplane is the closest to the input image. Once the closest hyperplane is identified, the algorithm computes the minimum perturbation in a similar way to the binary case, using the absolute value of the model output. The perturbation is then added to the previous perturbation *r∗(x0)* and multiplied by the overshoot scalar, and this process is repeated until the maximum number of iterations is reached or the predicted class changes.

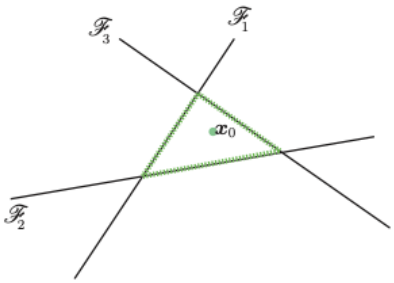

Figure 9. DeepFool for Multi-Class Classifier [8]

One of the strengths of DeepFool is its ability to generate adversarial examples that are effective across different deep neural networks trained on the same dataset. This is because the algorithm is based on a generalization of the linear decision boundary concept, which is independent of the specific architecture of the neural network being attacked. However, DeepFool is not perfect and has limitations. For example, it can be less effective against models with strong defenses against adversarial attacks, and it may not always find the "most malicious" perturbation that causes the most harmful misclassification.

---

8 . *https://arxiv.org/pdf/1511.04599.pdf*

6. Set up Environment

Machine learning model creation and training were done in Ubuntu 20.04.3 operating system with Python version 3.8. Jupiter Notebook is used for the development of Python code which is a very useful tool for development. We can have it in a single document with the Python code and the results of its execution, including images and graphics, making it very helpful since we get immediate feedback on the development activity in progress. It is a server-client application that allows editing and running notebooks via a web browser. The Jupiter notebook can be executed on a local desktop requiring no Internet access or on a remote server accessed through the internet. We will describe below the main applications used to carry out our implementations.

6.1 Tensorflow

In 2015, Google released its open-source framework for machine learning and named it Tensorflow [29]. It bundles together with Machine learning, Deep learning models, and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++. It is at present the most popular software library with several real-world applications. From DeepFace, Facebook's image recognition system, to Apple's Siri for voice recognition. It is also used in every Google app to improve our experience. It supports numerical computation, and large-scale machine learning on CPUs, GPUs, and clusters of GPUs.

Tensorflow makes it faster and easier for developers to implement machine learning models, as it assists the process of acquiring data, serving predictions at scale, or refining future results. It allows them to create a graph of computations to perform, where each node in the graph represent a mathematical operation and each connection represents data. Each connection between nodes represents

multidimensional vectors or matrices, creating what is known as tensors. All the computations associated with Tensorflow involve the use of tensors. Computations are made possible through interconnections of tensors. The mathematical operations are carried out by the node of the tensor and its edge explains the input and output relationship between nodes. Therefore, Tensorflow takes input in a form of a multidimensional array or matrix, which goes through a system of several operations and comes out as output.

6.2 Numpy

Numpy is a python library used for working with arrays. It stands for Numerical Python and was created in 2005 by Travis Oliphant. It provides a multidimensional array object, masked arrays and matrices, and fast operations on arrays such as mathematical, logical, shape manipulation, sorting, basic linear algebra and much more.

Some difference between Numpy arrays and the standard python sequences is that Numpy arrays facilitate advanced mathematical and other types of operations on a large number of data. It executes more efficiently and with less code such operations compared to python's built-in sequences. In addition, Python lists can grow dynamically when Numpy arrays have a fixed size when they are created. When we will need to change the size of an array, it will create a new one and delete the old one. Vectorization is the main reason why Numpy is considered very fast, due to its absence of any explicit looping or indexing in the code. It is more concise and easier to read.

6.3 Pandas

Pandas is an open-source python package that is mainly used for data science or analysis and machine learning tasks. Developed by Wes Mckinney in 2008. It provides various data structures and operations for numerical data and time series. It is built on top of Numpy, which we mentioned earlier. Pandas is fast and efficient for analyzing data, loading data from different objects, handling data that are missing represented as NaN in floating and nonfloating points, and providing time-series functionality. Moreover, it has the ability to merge and join datasets, reshape and pivoting of data sets, and perform split, apply, and combine data sets. Pandas makes it also simple to execute tasks such as data normalization, data visualization, and data inspection.

6.4 Sklearn

Sklearn or Scikit-learn is one of the most useful python libraries for machine learning. It is built upon Numpy, Scipy and Matplotlib. It features various algorithms and a lot of efficient tools for machine learning. Scikit-learn aims on modeling the data, we can find some of the most popular categories of models:

- Supervised algorithms, where we can have all the popular supervised learning algorithms such as Decision Tree, Support Vector Machines, Linear Regression, etc.
- Unsupervised algorithms, where we can have a large spread of unsupervised machine learning algorithms in the context of clustering, factor analysis, and principal component analysis to unsupervised neural networks.
- Cross-validation, where we can check the accuracy of supervised models on test data.
- Feature extraction, where it is used to extract features from data(images and text).

6.5 Cleverhans

Cleverhans is an open-source software library that provides techniques for generating adversarial examples and adversarial training. It is mostly used for two reasons. First, for developers to build robust models by using adversarial training, which requires the construction of adversarial examples during the training phase. Second, having a standard reference implementation helps the researchers who report the accuracy of their models in the adversarial settings to be more comparable with other benchmarks, and will be no cases of mistakes or weaker attacks. Two of the most important modules of cleverhans are the attacks and model.

The attacks module contains the Attack class where we can find all implementations of adversarial example crafting algorithms, such as the Fast Gradient Sign Method (FGSM) and Jacobian Saliency Map Attacks (JSMA). The model module contains the Model class where we can have examples of model implementations for Tensorflow models that are not implemented using a modeling framework library or model implementations for Keras Sequential models.

6.6 Adversarial Robustness Toolbox

The Adversarial Robustness Toolbox (ART) is an open-source software library designed to help researchers and developers better understand, evaluate, and improve the security and robustness of machine learning models. ART provides a suite of tools and techniques for detecting, generating, and defending against adversarial attacks.

ART includes various attack techniques, such as FGSM, PGD, DeepFool, and Carlini-Wagner, for generating adversarial examples, as well as various defense techniques, such as adversarial training, input transformation, and feature squeezing, for improving model robustness against such attacks. ART supports a wide range of deep learning frameworks, including TensorFlow, PyTorch, Keras, and scikit-learn.

7. Implementations

In this section, we will refer to the implementations that took place and their results. The IoTID20 dataset was used to build the IoT network intrusion detection system. The IoTID20 dataset was split into a training dataset and a test dataset in CSV format to build my machine learning models. The training and test dataset split can be seen in table 6. The CSV file of the dataset can be read using pandas.

	Training Dataset	Test Dataset
Mirai	5699	3490
Scan	1023	627
DoS	788	491
Normal	517	350
MIMT ARP Spoofing	468	304
Total	10495	6262

Table 5. Training/Test dataset split.

7.1 Data preparation

In figures 10 and 11 we can see the training data class distributions.

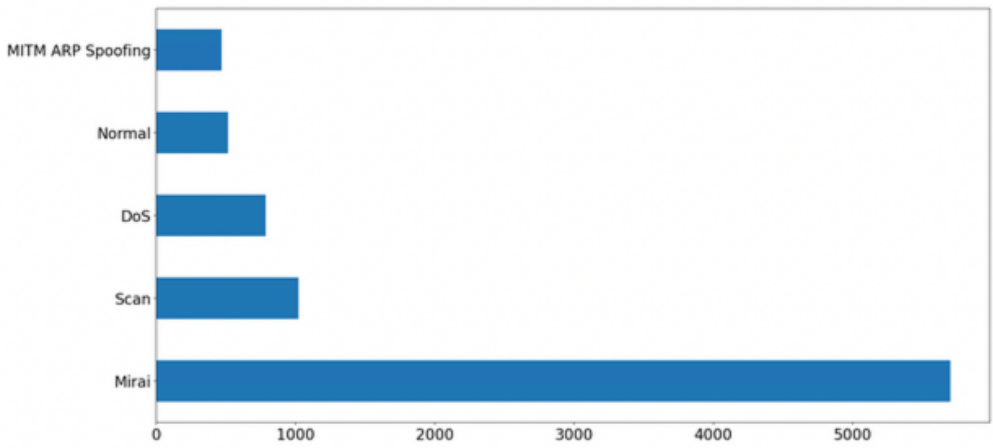

Figure 10. Training data attacks category distributions

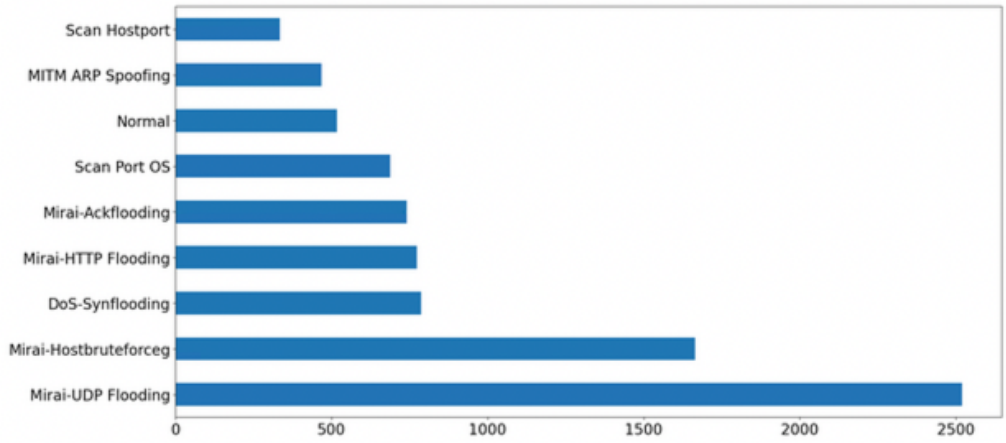

Figure 11. Training data attacks Sub_category distributions

And now we can see the test data class distribution in below figures 12 and 13.

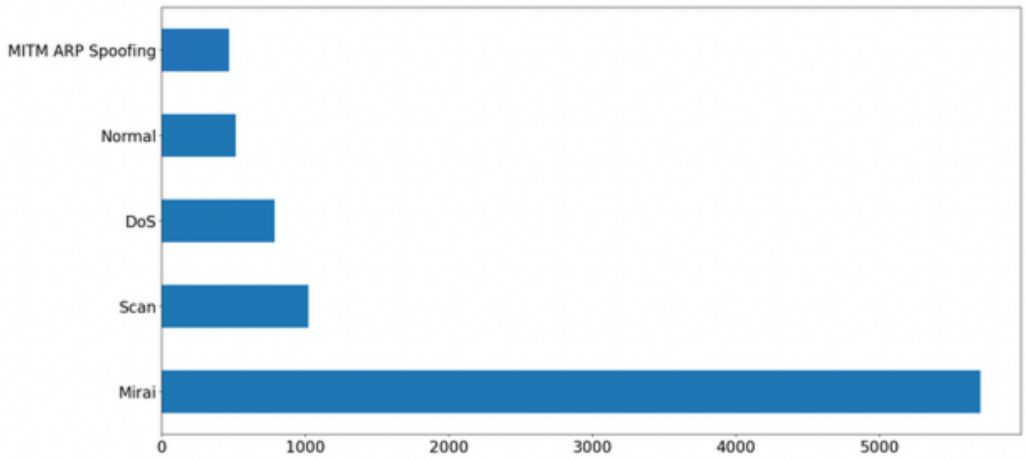

Figure 12. Test data attacks category distributions

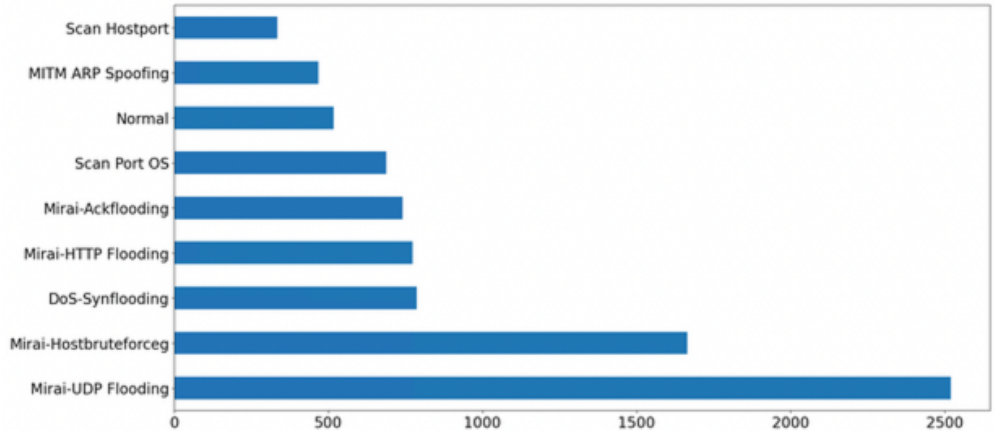

Figure 13. Test data attacks Sub_category distributions

Next step we are going to investigate our training and test set features. Below we find some of our training set features. All dataset features can be seen in Table 4.

	Src_Port	Dst_Port	Protocol	Flow_Duration	Tot_Fwd_Pkts	Tot_Bwd_Pkts	TotL
count	8495.000000	8495.000000	8495.000000	8495.000000	8495.000000	8495.000000	
mean	35221.611418	16445.180812	10.044026	622.771160	1.719482	1.463096	
std	24748.247798	17615.124310	5.388329	3348.040653	4.479508	0.719419	
min	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	
25%	9020.000000	8899.000000	6.000000	76.000000	0.000000	1.000000	
50%	51991.000000	9020.000000	6.000000	130.000000	1.000000	1.000000	
75%	56361.000000	10101.000000	17.000000	217.000000	2.000000	2.000000	
max	64783.000000	64955.000000	17.000000	99839.000000	131.000000	9.000000	

Figure 14. Training set features distribution

Looking at the distributions of our training set features we can understand that each feature varies widely. This will have an impact on our results if we use any distance-based methods for classification. For example, the std of Src_Port is significantly larger than the std of Tot_Fwd_Pkts. In that point, if we do not perform feature value standardization, the Src_Port feature would dominate, having as a result our model could possibly miss out on potentially important information from Tot_Fwd_Pkts.

Standardization is a process that rescales a data series to have a mean of 0 and a standard deviation of 1. Therefore, it will normalize the features individually so each of them will have $\mu = 0$ and $\sigma = 1$. This will allow us to compare multiple features together and get more relevant information since all the data will be on the same scale. To apply standardization we will use the StandardScaler class of Sklearn library. We can see the results of standardization in Figure 15 below where now the std of Scr_Port and Tot_Fwd_Pkts do not have that big of a difference.

	Src_Port	Dst_Port	Protocol	Flow_Duration	Tot_Fwd_Pkts	Tot_Bwd_Pkts	TotL
count	8.495000e+03	8.495000e+03	8.495000e+03	8.495000e+03	8.495000e+03	8.495000e+03	
mean	6.126811e-17	-1.150084e-17	1.758060e-16	-2.300168e-18	-2.049240e-17	1.233726e-16	
std	1.000059e+00	1.000059e+00	1.000059e+00	1.000059e+00	1.000059e+00	1.000059e+00	
min	-1.423280e+00	-9.336380e-01	-1.864143e+00	-1.857229e-01	-3.838777e-01	-6.437465e-01	
25%	-1.058788e+00	-4.284174e-01	-7.505600e-01	-1.633204e-01	-3.838777e-01	-6.437465e-01	
50%	6.776389e-01	-4.215479e-01	-7.505600e-01	-1.471906e-01	-1.606258e-01	-6.437465e-01	
75%	8.542275e-01	-3.601765e-01	1.291009e+00	-1.212037e-01	6.262618e-02	7.463467e-01	
max	1.194554e+00	2.754035e+00	1.291009e+00	2.963586e+01	2.886213e+01	1.047700e+01	

Figure 15. Training set features distribution after Standardization

7.1 Model Evaluation

To find the results regarding the algorithms, some metrics of the sklearn library will be looked into.

7.1.1 Common Evaluation Metrics

A Confusion Matrix is a technique for summarizing the performance of a classification algorithm. Having more than 2 classes or an unequal number of observations, classification accuracy alone can be misleading. With a confusion matrix, we can understand better how our classification model performs, summarizing all the correct and incorrect predictions with count values and broken down by each class. When creating a 2x2 Confusion matrix we get four different combinations from the predicted values of a classifier.

- True Positive: These are cases in which we predicted a positive value and it is correct. For example, we predicted that the network traffic is an attack and it was correct.

- True Negative: These are cases in which we predicted a negative value and it is actually negative. For example, we predicted that the network traffic is not an attack and it was indeed normal traffic.

- False positive: We predicted a positive value and it was actually a negative value. For example, we predicted that the network traffic is an attack and it was eventually normal traffic.

- False Negative: We predicted a negative value and it was actually a positive value. For example, we predicted that the network traffic is normal traffic but it was eventually an attack.

Having the Confusion matrix we can not really understand the performance of our model. Below we can find some rates that can be computed from a confusion matrix.

- Accuracy: The accuracy is used to find the portion of correctly classified values. It is an overall rate, where we see how often is the classifier correct. To compute it we add all the true values and divide by the total values:

  (True Positive + True Negative)/total

- Precision: Precision is used to calculate the model's ability to classify positive values correctly. It means that we find how often we predicted that it was an attack and it was correct. To do so, we divide the true positives by the total number of predicted positive values.

  True Positive/(True Positive + False Positive)

- Recall: Recall is the ability of the model to predict positive values. To calculate it we divide the True Positive by the sum of actually positive values.

  True Positive/(True Positive + False Negative)

- F1-Score: This is a weighted average of recall and precision. It is useful when we want to have both Precision and Recall

  2*Precision*Recall/(Precision + Recall)

7.1.2 V-measure

For measuring the performance of unsupervised learning we can use the V-measure to evaluate the models. V measure or so-called Normalised Mutual Information is an average of the other two measures, homogeneity, and completeness.

Homogeneity measures how many data points a cluster has that belong to the same class label. A cluster, in order to be referred to as homogeneous should contain only samples belonging to a single class. When all samples in cluster k have the same label the homogeneity equals 1. The score can be between 0 and 1. A low value indicates low homogeneity and a high value indicates high homogeneity.

Completeness describes the closeness of a cluster where all data points belonging to the same class are clustered into the same cluster. The completeness score provides information regarding the assignment of samples belonging to the same class. A good cluster algorithm should assign all samples to the same cluster.

A complete cluster does not mean it is also homogeneous. For example, regarding homogeneity, we can have data samples with only one feature in a cluster, but this does not mean that it could not exist also another cluster having again only the same data samples. On the other side, regarding completeness, we can have points with the same property classified together, but this does not mean that there may not be other samples in the same cluster. Now, we understand that we need a metric that we could measure as homogeneous and complete. V-measure is the harmonic average between homogeneity and completeness. The score can be between 0 - 1 that we can

evaluate our clustering algorithm. If a cluster does not satisfy both homogeneity and completeness the score will be 0.

7.1.3 Attack Success Rate

The attack success rate is a measure of the effectiveness of an adversarial attack on a machine learning model. It is defined as the proportion of adversarial examples that are misclassified by the target model.

To compute the attack success rate, we first generate a set of adversarial examples by applying a perturbation to the input data in order to cause misclassification by the target model. We then feed these adversarial examples into the target model and record the number of examples that were misclassified. The attack success rate is then computed as the ratio of misclassified adversarial examples to the total number of adversarial examples generated.

The attack success rate is an important metric for evaluating the security of machine learning models and assessing their susceptibility to adversarial attacks. A high attack success rate indicates that the model is vulnerable to adversarial attacks, while a low attack success rate indicates that the model is more robust against such attacks. To compute the attack success rate given the accuracy on normal and adversarial examples, you can use the following formula:

$$ASR = 1 - \frac{AccAdv}{AccNorm}$$

Where:

- AccAdv is the accuracy of adversarial examples
- AccNorm is accuracy on normal examples

7.2 Supervised Machine learning models

The supervised machine learning models, namely the Decision Tree classifier, the K-Nearest neighbors classifier, and the Linear Support Vector classifier were used using sklearn.

7.2.1 Results - Decision Tree Classifier

The Confusion Matrix of the Decision Tree Classifier can be seen below in Figure 16. It is a 5x5 confusion matrix, as it is a 5-class classification. The diagonal values, from the upper left to lower right, are the counts of the correctly classified samples. All the values in the matrix add up to 4.909 which is the size of the test set. The rows represent the true class and the columns the predicted class.

```
[[ 488    3    0    0    0]
 [   0  304    0    0    0]
 [   0    0 3490    0    0]
 [   0    0    0  350    0]
 [   0    0    0    0  627]]
```

Figure 16. Decision Tree Confusion Matrix

	Accuracy	F1 score	K-Nearest
Decision Tree	100%	100%	100%

Table 6. Decision Tree Metrics

7.2.2 Results - K-Nearest classifier

The Confusion Matrix of the K-Nearest classifier can be seen in Figure 17 below.

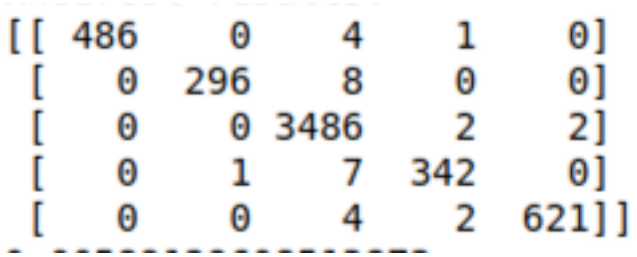

Figure 17. K-Nearest Confusion Matrix

	Accuracy	F1 score	K-Nearest
K-Nearest	99.97%	99.97%	99.97%

Table 7. K-Nearest Metrics

7.2.3 Results - Support Vector Classifier

The Confusion Matrix of the Support Vector Classifier can be seen in Figure 18 below.

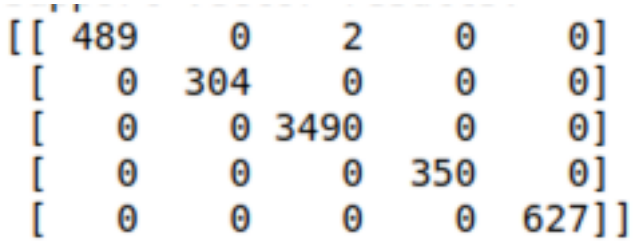

Figure 18. Support Vector Confusion Matrix

	Accuracy	F1 score	K-Nearest
SVM	100%	100%	100%

Table 8. SVM Metrics

7.3 Unsupervised Machine learning model

For unsupervised machine learning, the K-means clustering method was implemented using sklearn.

7.3.1 Results - K-means

As said in the previous chapter, each group of similar points is a cluster and each cluster represents a category. The k number in k-means represents the number of clusters. In case we do not have any knowledge regarding the dataset and its classes, in order to find the optimal k number we would use the elbow method. In our case, plotting the elbow method for k from 1 to 10 we get the following results in figure 19.

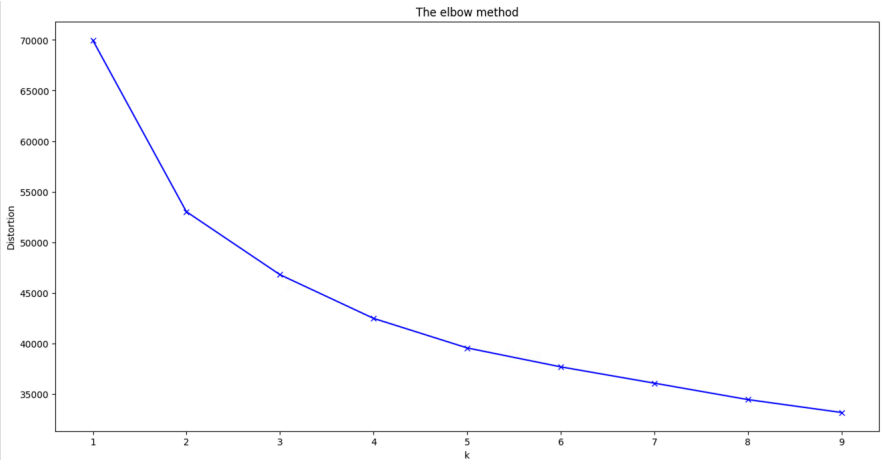

Figure 19. Elbow method finding k.

From the plot, it is not very clear where the “elbow” is. Since it was known that the dataset contains 5 categories of samples, k was chosen to be 5. We can see the clustering results below in figure 20.

```
1    2544
0    2388
3     319
2       9
4       2
```

Figure 20. Clustering results

Here, as we may see we have a lot of differences from supervised learning. Here we have 5 clusters labeled with an arbitrary index, whereas in supervised we would have the attack category. We can see that cluster 1 has 2544 samples, cluster 0 has 2388 samples, cluster 3 has 319 samples, cluster 2 has 9 samples and cluster 4 has 2 samples. We did not pass any labels to the algorithm, thus evaluating the results of the cluster is not as easy as in supervised since now we can not compare expected and predicted labels. To evaluate our model we will compute the completeness score, homogeneity score, and finally the V-measure. Also, for comparison reasons, we will compute the accuracy score using the accuracy_score of sklearn metrics. The results are below:

	Accuracy	Completeness	Homogeneity	V-measure
KMeans	34.21%	32.39%	48.70%	38.91%

Table 9. KMeans Metrics

The V-measure score of 38.91% is a really bad result. It seems that the data in its current state is not suitable for unsupervised learning. Visualization is always useful when we are talking about clustering. Since we have 6770 features it would be impossible to plot all dimensions, so dimensionality reduction was performed to reduce the data to two dimensions, in order to be able to represent them in two-dimensional Cartesian axes using the PCA function from sklearn. The Principal Component Analysis (PCA) is a popular technique for reducing the dimensionality of data. It increases interpretability but at the same time, it minimizes information loss. It has the ability to make data easy for plotting in 2D and 3D. In figure 21 we can see the plot of training data with predicted 5 classes of k-means.

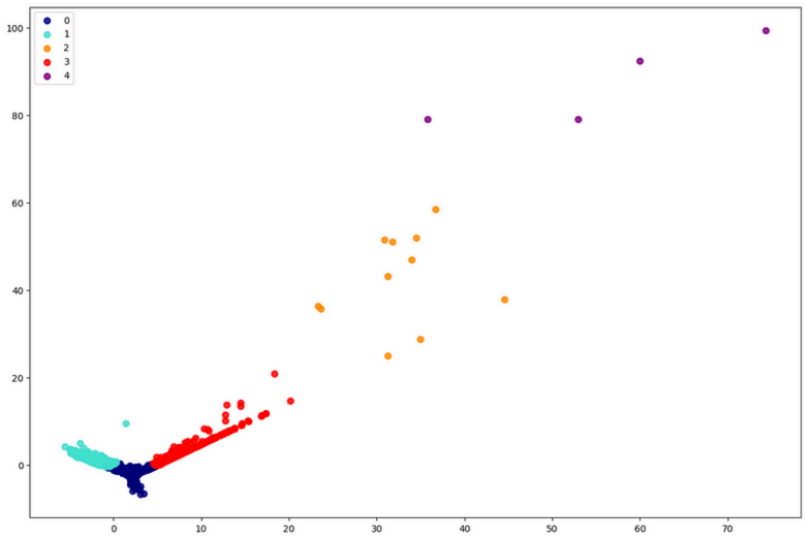

Figure 21. The plot of training data with the predicted class of k-means

In general, the k-means clustering algorithm assumes that the clusters have spherical shapes and similar sizes. If the clusters are non-spherical or have different sizes, then the k-means algorithm may not work well. In Figure 21, it is easy to see that this dataset is not very suitable for clustering. Samples are scattered unpredictably and do not form any strong clusters. To better understand this, below in Figure 22 we can see the 3D plot of clusters.

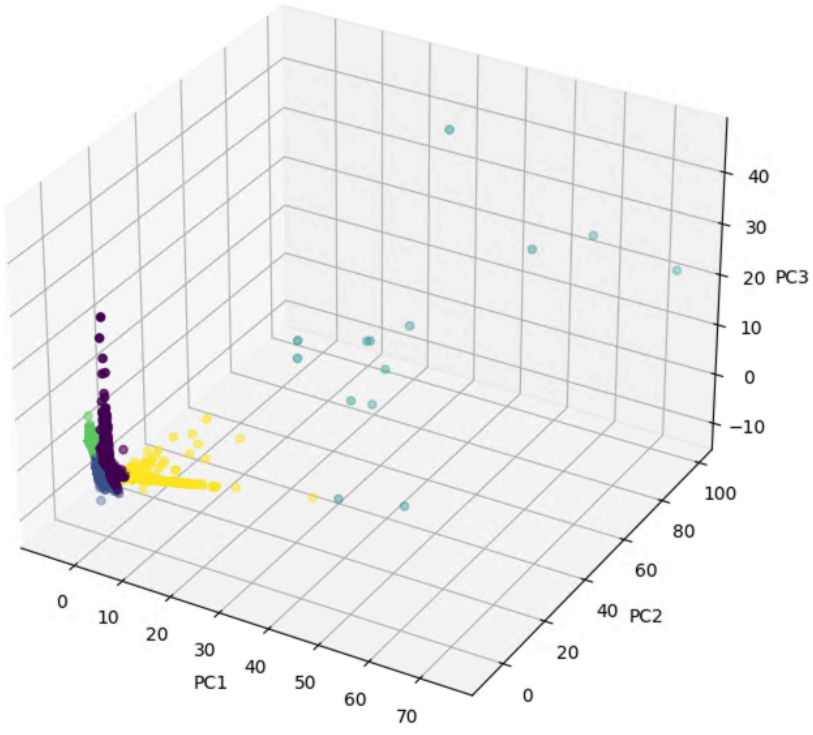

Figure 22. The 3D plot of training data with the predicted class of k-means

7.5 Deep Neural Network

In order to create our Deep learning model we used the Keras library. Models in Keras are defined as a sequence of layers. Our model is a Sequential model, which means that from input to output, passing through a series of neural layers one after the other. We will have a fully-connected network structure with three Dense layers. The dense layer is the regular deeply connected neural network layer which is the most commonly used layer. To create a layer we will need to specify the number of its neurons and its activation function. The activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output of that input. In other words, it will decide whether the neuron's input is important to the network, using simpler mathematical operations.

To create our model, first, we have the input layer where we get as input the training data. We will have 256 neurons or nodes, and the activation function will be ReLU. Rectified Linear Unit (ReLU) is a default activation function for many types of neural networks since models that use it are easier to train and often achieve better performance. It is a linear function that will output the input directly if it is positive, or, it will output zero. Our second layer will have also 256 nodes and ReLu as an activation function. Our third and final layer, the output layer, has 5 nodes and uses the softmax activation function. The softmax activation function calculates the relative probabilities. Below we have the equation for Softmax:

$$softmax(Z_i) = \frac{exp(Z_i)}{\Sigma_j \, exp\,(Z_j)}$$

Z represents the values from the neurons of the output layer that will be divided by the sum of exponential values in order to normalize and afterward convert them into probabilities. The softmax activation function is popularly used in multi-class classification problems.

Between the three dense layers, we have two dropout regularizations [31]. Dropout is a technique where randomly selected neurons are ignored during training, which means that their contribution to the activation of downstream neurons is

dropped out on the forward pass as well as any right updates are not applied to the neuron on the backward pass. Weights of the neurons are tuned for specific features providing some specialization on which neighboring neurons rely. If taken too far model could be too specialized for the training data. In order to make our network less sensitive to the weights of neurons, we could randomly drop out neurons from the network during training having as a result, other neurons step in and handle the representation required to make predictions for the missing ones. This brings a network capable of better generalization having lower probabilities to have overfilling in our training data. Below in figure 22, we can see how our modes are configured.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense (Dense)                (None, 256)               12032

 dropout (Dropout)            (None, 256)               0

 dense_1 (Dense)              (None, 256)               65792

 dropout_1 (Dropout)          (None, 256)               0

 dense_2 (Dense)              (None, 5)                 1285

=================================================================
Total params: 79,109
Trainable params: 79,109
Non-trainable params: 0
```

Figure 22. Deep learning model setup

After defining our model the next step is to compile it using the efficient numerical libraries of Tensorflow in order to be able to train and make predictions with our model. We will have to specify the loss function, the optimizer, and the metrics. The loss function is used to calculate the quantity the model should aim to minimize during training. In our model, we will use the Cross entropy loss function which predicts the probability of whether the data belongs to one class or the other. More specifically we will use the categorical cross-entropy type which is used for multiclass models. The output label of the model is converted into a categorical encoding in form of zeros and ones. For the optimizer parameter, we will use the

“Adam” which is a popular version of the gradient descent algorithm and has the ability to configure itself to give good results in a wide range of problems. Finally, for the metrics argument, we define accuracy.

To perform the training of the model we have to define three parameters, the epochs number, the batch size, and the learning rate. The epochs are the number of times that the learning algorithm will go through the entire training dataset. One epoch means that each sample in the training dataset had the opportunity to update the internal model parameters. The batch size defines the number of samples to work through before updating the internal model parameters. In most cases, the number of epochs is large allowing the learning algorithm to run until the error in the model is minimized. Our epoch number is 20 and the batch size is 128. The learning rate will be 0.1, which is a common value, where the network will be updated 0.1 (or 10%) of the estimated weight error, each time the weights are updated. Starting the training we will see the output of every epoch performed accompanied by the time it took to complete and the test accuracy. The test accuracy gives us an idea of how well we have modeled the dataset. Is an important metric as it helps evaluate the generalization performance of the model. A model with a high test accuracy indicates that it can effectively generalize to new, unseen data and is likely to perform well in real-world applications. It's important to note that the test accuracy should only be calculated and reported after the model has been fully trained. Below in figure 23, we can see the output of the first 5 epochs. Our Deep learning model test accuracy is 99.97%.

```
[INFO 2023-02-04 10:30:02,207 cleverhans] Epoch 0 took 0.12333822250366211 seconds
[INFO 2023-02-04 10:30:02,295 cleverhans] Epoch 1 took 0.023437976837158203 seconds
[INFO 2023-02-04 10:30:02,333 cleverhans] Epoch 2 took 0.02649664878845215 seconds
[INFO 2023-02-04 10:30:02,372 cleverhans] Epoch 3 took 0.029691696166992188 seconds
[INFO 2023-02-04 10:30:02,413 cleverhans] Epoch 4 took 0.026059627532958984 seconds
[INFO 2023-02-04 10:30:02,446 cleverhans] Epoch 5 took 0.024161338806152344 seconds

Test  accuracy : 0.9069767441860465
Test  accuracy : 0.9966777408637874
Test  accuracy : 0.9966777408637874
Test  accuracy : 0.9966777408637874
Test  accuracy : 0.9933554817275747
Test  accuracy : 0.9833887043189369
```

Figure 23. First 5 epochs completion time and test accuracy

7.6 Results - Jacobian Silency Map Attack

We performed the Jacobian Silency Map attack using the Cleverhans library. When executing the attack we will see in the output of the program, figure 24, the creation of the adversarial examples and also some failed attempts to find adversarial attacks after a number of iterations.

```
[INFO 2023-02-05 14:52:47,678 cleverhans] Failed to find adversarial example after 2 iterations
[INFO 2023-02-05 14:52:47,690 cleverhans] Failed to find adversarial example after 2 iterations
[INFO 2023-02-05 14:52:47,703 cleverhans] Failed to find adversarial example after 2 iterations
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Creating adv . example for target class0
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Creating adv . example for target class0
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Creating adv . example for target class0
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Creating adv . example for target class0
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Creating adv . example for target class0
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Creating adv . example for target class0
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Creating adv . example for target class0
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 24. JSMA execution

The results of the attack in test accuracy of the model, in classification algorithms, and in clustering algorithm are described below followed by the ROC curve. ROC (Receiver Operating Characteristic) curve is a graph that plots the True Positive Rate and False Positive Rate that shows the performance of the model. The ROC curve provides a visual representation of the trade-off between the true positive rate and the false positive rate for a given classifier. The area under the ROC Curve, known as AUC, is used as a measure of the overall performance of a classifier. Classifiers whose curve is close to the top-left corner indicate better performance. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test is.

Test Accuracy on Adversarial examples	93.34%

Table 10. JSMA test accuracy on adversarial examples

Decision Tree Classifier	
Accuracy Score in Adversarial examples	92.51%
F1 score in adversarial examples	96.14%
AUC score in adversarial examples	49.56%

Table 11. JSMA Decision Tree Classifier metrics

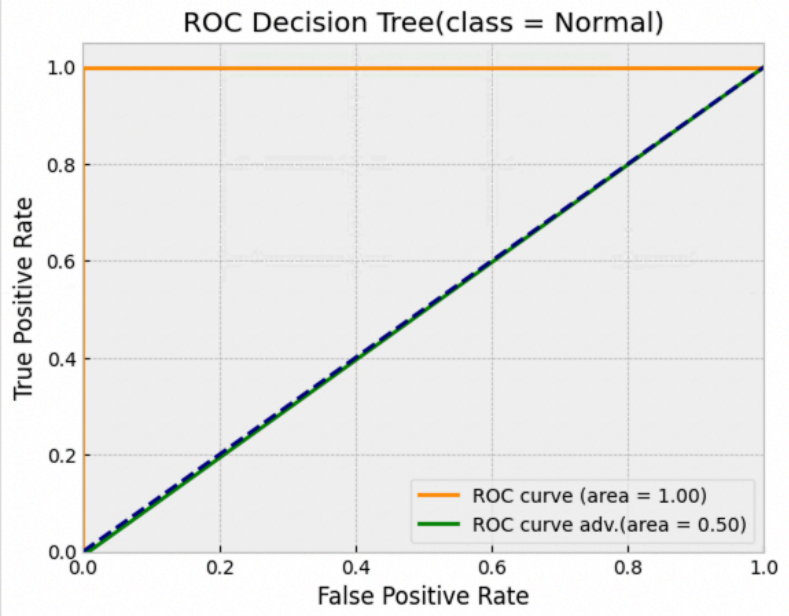

Figure 25. ROC curve Decision Tree

SVM Classifier	
Accuracy Score in Adversarial examples	93.06%
F1 score in adversarial examples	96.41%
AUC score in adversarial examples	49.56%

Table 12. JSMA SVM Classifier metrics

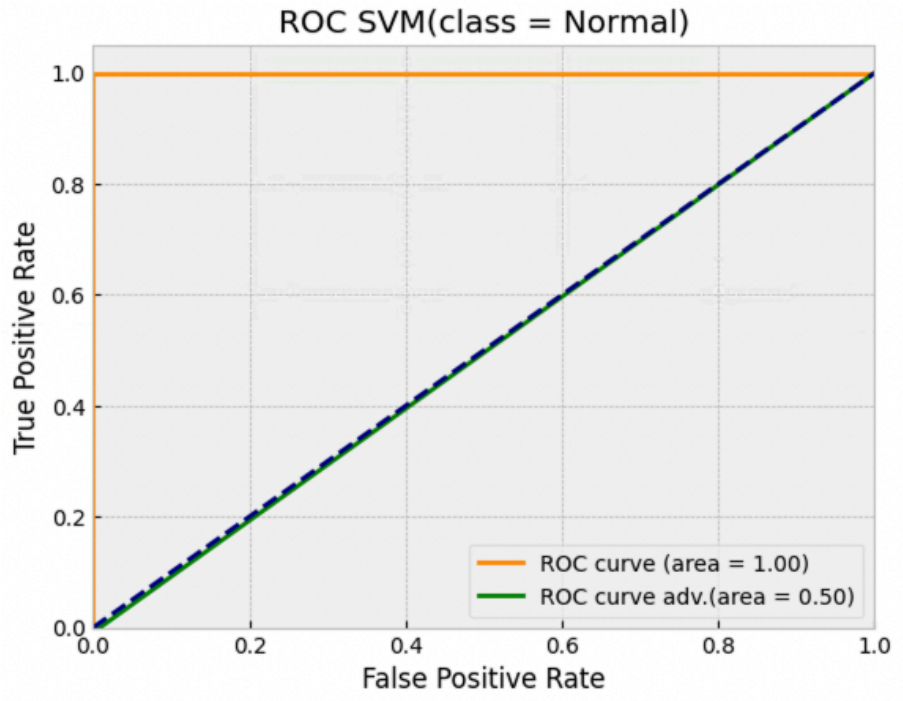

Figure 26. ROC curve SVM

K-Nearest Classifier	
Accuracy Score in Adversarial examples	93.06%
F1 score in adversarial examples	93%
AUC score in adversarial examples	49.84%

Table 13. JSMA K-Nearest Classifier metrics

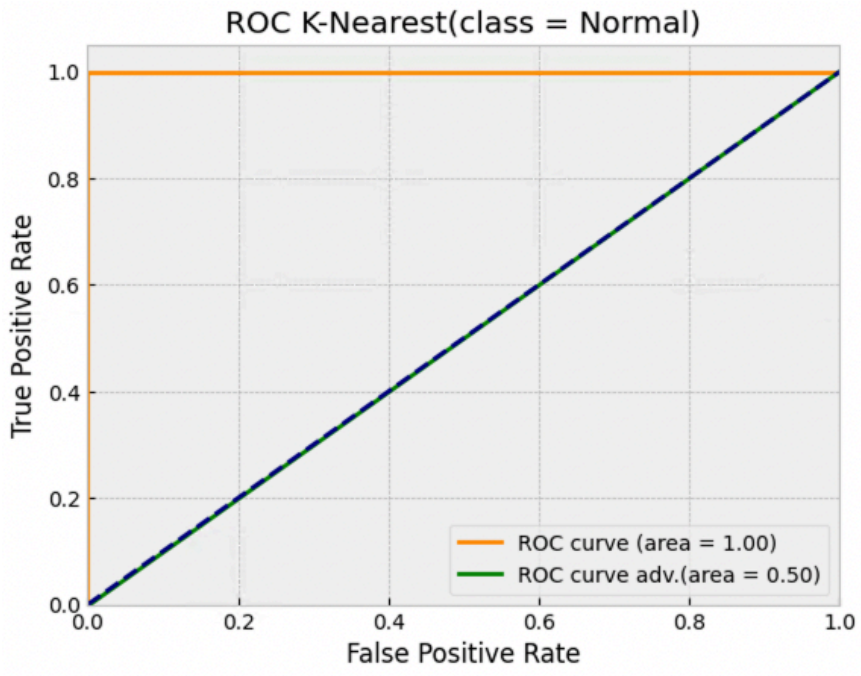

Figure 27. ROC curve K-Nearest

KMeans				
	Accuracy	Completeness	Homogeneity	V_measure
JSMA	22.59%	37.90%	66.15%	48.19%

Table 14. JSMA K- means accuracy

7.7 Results - Fast Gradient Sign Method

We performed the Fast Gradient Sign Method attack using the Cleverhans library. For the epsilon parameter, we choose three values 0.4, 0.6, and 0.8, as we have said in the previous chapter the bigger the epsilon the bigger will be the noise. Below we can see the results of the attack in test accuracy of the model and in the classifiers followed by their ROC curve.

Test Accuracy on Adversarial examples (e=0.4)	91.36%
Test Accuracy on Adversarial examples (e=0.6)	82.72%
Test Accuracy on Adversarial examples (e=0.8)	75.08%

Table 15. FGSM test accuracy on adversarial examples

Decision Tree Classifier	
Accuracy Score in Adversarial examples (e=0.4)	100%
Accuracy Score in Adversarial examples (e=0.6)	80.39%
Accuracy Score in Adversarial examples (e=0.8)	69.76%
F1 score in adversarial examples (e=0.4)	100%
F1 score in adversarial examples (e=0.6)	74.27%
F1 score in adversarial examples (e=0.8)	55.29%
AUC score in adversarial examples (e=0.4)	100%
AUC score in adversarial examples (e=0.6)	61.49%
AUC score in adversarial examples (e=0.8)	45.78%

Table 16. FGSM Decision Tree classifier metrics

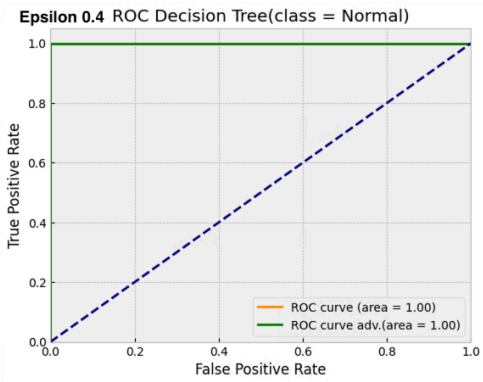

Figure 28. ROC Curve Decision Tree FGSM e0.4

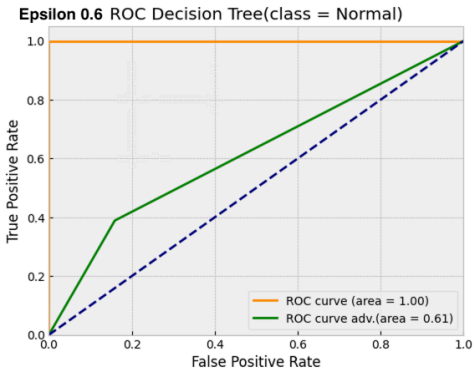

Figure 29. ROC Curve Decision Tree-FGSM e0.6

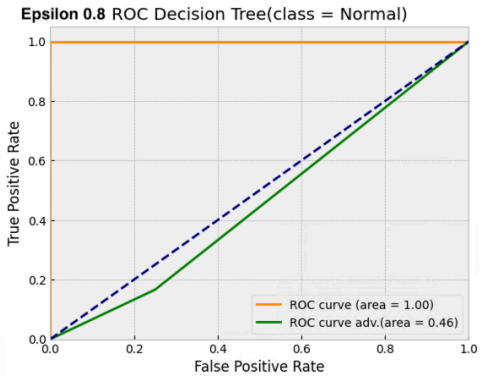

Figure 30. ROC Curve Decision Tree - FGSM - e 0.6

SVM Classifier	
Accuracy Score in Adversarial examples (e=0.4)	93.68%%
Accuracy Score in Adversarial examples (e=0.6)	82.39%
Accuracy Score in Adversarial examples (e=0.8)	69.76%
F1 score in adversarial examples (e=0.4)	94.89%
F1 score in adversarial examples (e=0.6)	82.65%
F1 score in adversarial examples (e=0.8)	63.62%
AUC score in adversarial examples (e=0.4)	100%
AUC score in adversarial examples (e=0.6)	61.49%
AUC score in adversarial examples (e=0.8)	45.78%

Table 17. FGSM SVM Classifier metrics

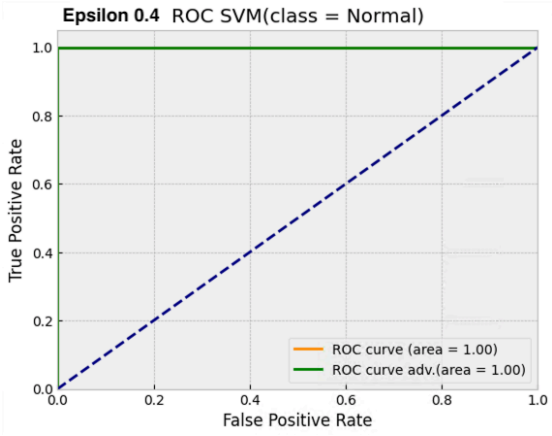

Figure 31. ROC Curve SVM - FGSM - e 0.4

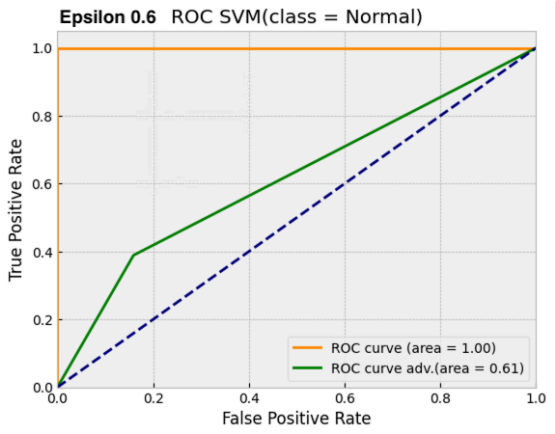

Figure 32. ROC Curve SVM - FGSM - e 0.6

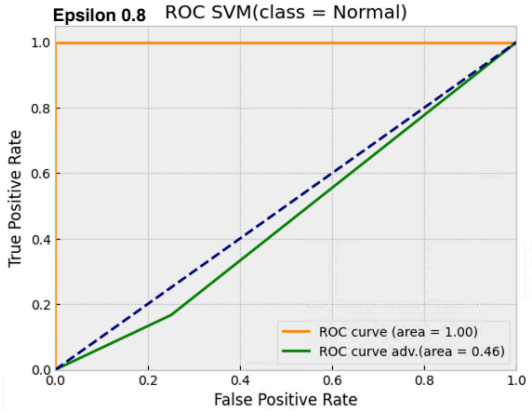

Figure 33. ROC Curve SVM - FGSM - e 0.8

K-Nearest Classifier	
Accuracy Score in Adversarial examples (e=0.4)	98.33%
Accuracy Score in Adversarial examples (e=0.6)	94.35%
Accuracy Score in Adversarial examples (e=0.8)	71.76%
F1 score in adversarial examples (e=0.4)	98.33%
F1 score in adversarial examples (e=0.6)	94.35%
F1 score in adversarial examples (e=0.8)	71.76%
AUC score in adversarial examples (e=0.4)	100%
AUC score in adversarial examples (e=0.6)	72.04%
AUC score in adversarial examples (e=0.8)	61.76%

Table 18. FGSM K-Nearest Classifier metrics

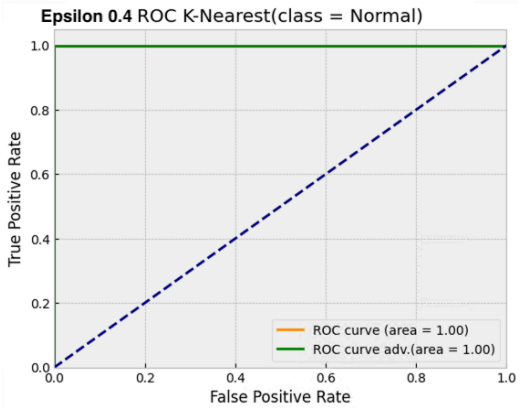

Figure 34. ROC Curve K-Nearest-FGSM-e 0.4

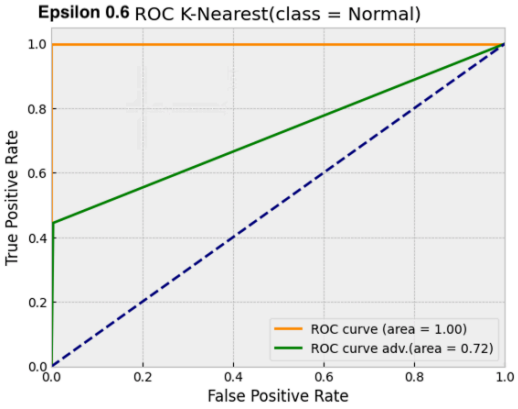

Figure 35. ROC Curve K-Nearest-FGSM-e0.6

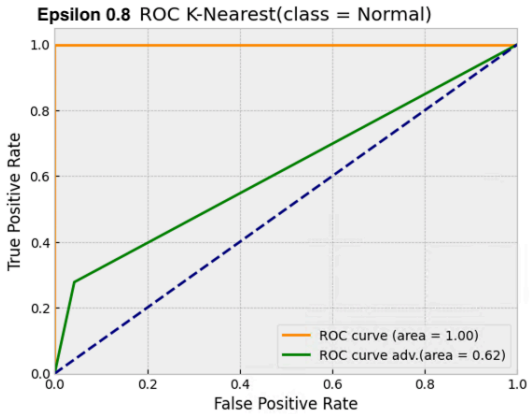

Figure 36. ROC Curve K-Nearest-FGSM-e 0.8

KMeans				
	Accuracy	Completeness	Homogeneity	V-measure
FGSM (e=0.4)	27.24%	25.87%	36.37%	30.24%
FGSM (e=0.6)	24.85%	37.05%	54.12%	43.99%
FGSM (e=0.8)	22.25%	13.41%	17.86%	15.31%

Table 19. FGSM K-Means accuracy on adversarial examples

7.8 Results - DeepFool

We performed the DeepFool attack using the Adversarial Robustness Toolbox (ART). Below we can see the results of the attack in test accuracy of the model and in the classifiers followed by their ROC curve.

Test Accuracy on Adversarial examples	61.60%

Table 20. DeepFool test accuracy on adversarial examples

Decision Tree Classifier	
Accuracy Score in Adversarial examples	62.55%
F1 score in adversarial examples	44.26%
AUC score in adversarial examples	86.35%

Table 21. DeepFool Decision Tree Classifier metrics

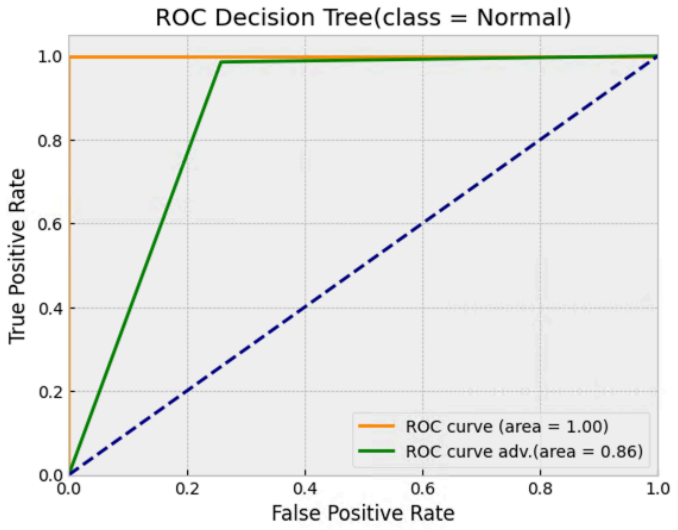

Figure 37. ROC Curve Decision Tree DeepFool

SVM Classifier	
Accuracy Score in Adversarial examples	60.93%
F1 score in adversarial examples	44.87%
AUC score in adversarial examples	86.35%

Table 22. DeepFool SVM Classifier metrics

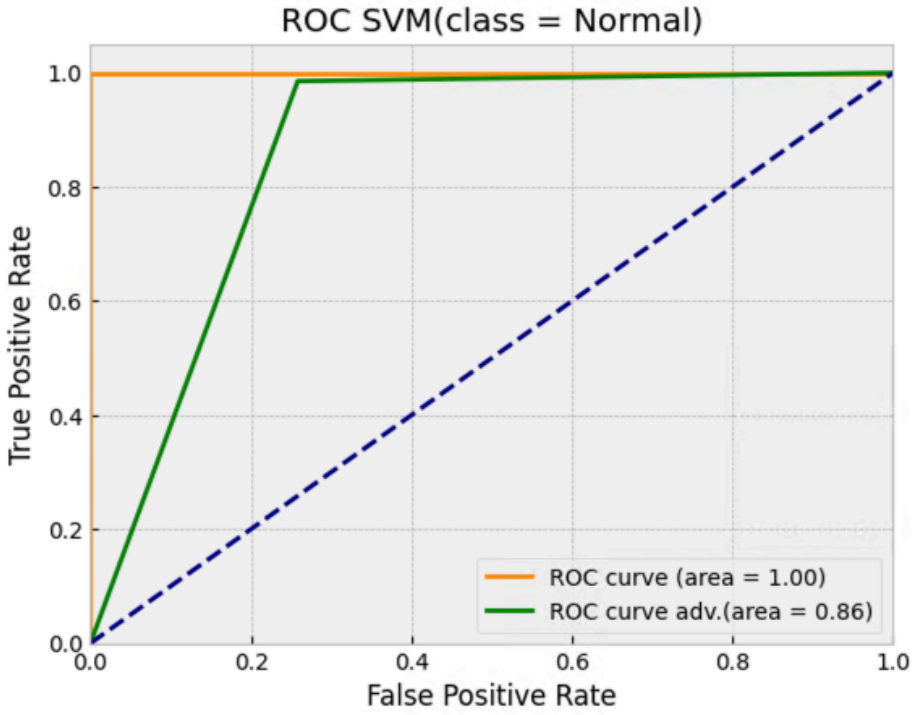

Figure 38. ROC Curve SVM DeepFool

K-Nearest Classifier	
Accuracy Score in Adversarial examples	63.77%
F1 score in adversarial examples	63.77%
AUC score in adversarial examples	91.88%

Table 23. DeepFool K-Nearest Classifier metrics

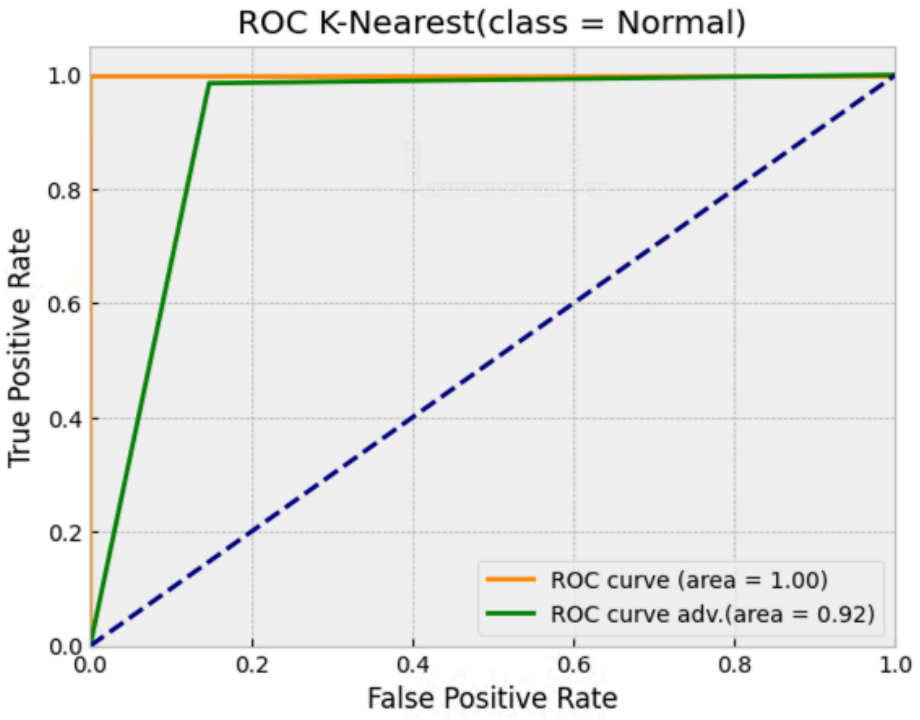

Figure 45. ROC Curve K-Nearest DeepFool

KMeans				
	Accuracy	Completeness	Homogeneity	V-measure
DeepFool	15.21%	20.40%	23.65%	21.91%

Table 24. DeepFool K-Means accuracy on adversarial examples

8. Adversarial Attacks - Overall Results

Test Accuracy of model

	Percentage	Difference
Test Accuracy on normal data	99.97%	
JSMA	93.34%	-6.63%
FGSM(e=0.4)	91.36%	-8.61%
FGSM (e=0.6)	82.72%	-17.25%
FGSM (e=0.8)	75.08%	-24.89%
DeepFool	61.60%	-38.37%

Table 25. Test Accuracy of model - Overall results

Decision Tree Classifier - Accuracy

	Percentage	Difference
Accuracy Score	100%	
JSMA	92.51%	-7.49%
FGSM (e=0.4)	100%	-0.0%
FGSM (e=0.6)	80.39%	-19.61%
FGSM (e=0.8)	69.76%	-30.24%
DeepFool	62.55%	-37.45%

Table 26. Decision Tree Classifier - Accuracy overall results

Decision Tree Classifier - F1 Score

	Percentage	Difference
Accuracy Score	100%	
JSMA	96.14%	-3,86%
FGSM(e=0.4)	100%	-0.0%
FGSM(e=0.6)	74.27%	-25.73%
FGSM(e=0.8)	55.29%	-44.71%
DeepFool	44.26%	-55.74%

Table 27. Decision Tree Classifier - F1 Score overall results

Decision Tree Classifier - AUC Score

	Percentage	Difference
Accuracy Score	100%	
JSMA	49.56%	-50.44%
FGSM(e=0.4)	100%	-0.0%
FGSM(e=0.6)	61.49%	-38.51%
FGSM(e=0.8)	45.78%	-54,22%
DeepFool	86.35%	-13,65%

Table 28. Decision Tree Classifier - AUC Score overall results

SVM Classifier - Accuracy

	Percentage	Difference
Accuracy Score	100%	
JSMA	93.06%	-6.94%
FGSM(e=0.4)	93.68%%	-6,32%
FGSM(e=0.6)	82.39%	-17,61%
FGSM(e=0.8)	69.76%	-30,24%

DeepFool	60.93%	-39,07%

Table 29. SVM Classifier - Accuracy overall results

SVM Classifier - F1 Score

	Percentage	Difference
Accuracy Score	100%	
JSMA	96.41%	-3.59%
FGSM(e=0.4)	94.89%	-5.11%
FGSM(e=0.6)	82.65%	-17.35%
FGSM(e=0.8)	63.62%	-36.38%
DeepFool	44.87%	-55.13%

Table 30. SVM Classifier - F1 Score overall results

SVM Classifier - AUC Score

	Percentage	Difference
Accuracy Score	100%	
JSMA	49.56%	-50.44%
FGSM(e=0.4)	100%	0.0%
FGSM(e=0.6)	61.49%	-38.51%
FGSM(e=0.8)	45.78%	-54.22%
DeepFool	86.35%	-13.65%

Table 31. SVM Classifier - AUC Score overall results

K-Nearest Classifier - Accuracy

	Percentage	Difference
Accuracy Score	99.97%	
JSMA	93.06%	-6.94%
FGSM(e=0.4)	98.33%	-1.67%

FGSM(e=0.6)	94.35%	-5.65%
FGSM(e=0.8)	71.76%	-28.24%
DeepFool	63.77%	-36.23%

Table 32. K-Nearest Classifier - Accuracy overall results

K-Nearest Classifier - F1 Score

	Percentage	Difference
Accuracy Score	99.97%	
JSMA	93%	-7%
FGSM(e=0.4)	98.33%	-1.67%
FGSM(e=0.6)	94.35%	-5.65%
FGSM(e=0.8)	71.76%	-28.24%
DeepFool	63.77%	-36.23%

Table 33. K-Nearest Classifier - F1 Score overall results

K-Nearest Classifier - AUC Score

	Percentage	Difference
Accuracy Score	100%	
JSMA	49.84%	-50.16%
FGSM(e=0.4)	100%	0.0%
FGSM(e=0.6)	72.04%	-27.96%
FGSM(e=0.8)	61.76%	-38.24%
DeepFool	91.88%	-8.12%

Table 34. K-Nearest Classifier - AUC Score overall results

K-Means

	Accuracy	Completeness	Homogeneity	V-measure
Normal samples	34.21%	32.39%	48.70%	38.91%
JSMA	22.59%	37.90%	66.15%	48.19%
FGSM(e=0.4)	27.24%	25.87%	36.37%	30.24%
FGSM(e=0.6)	24.85%	37.05%	54.12%	43.99%
FGSM(e=0.8)	22.25%	13.41%	17.86%	15.31%
DeepFool	15.21%	20.40%	23.65%	21.91%

Table 35. KMeans — Accuracy overall results

Attack Success Rate

	JSMA	FGSM(0.4)	FGSM(0.6)	FGSM(0.8)	DeepFool
Test Accuracy on model	6.64%	8.62%	17.26%	24.9%	38.39%
Decision Tree	7.49%	0%	19.61%	30.24%	37.45%
SVM	6.94	6.32%	17.61%	30.24%	39.07%
K-Nearest	6.92%	1.67%	5.65%	28.22%	36.23%
K-Means	34%	21%	28%	35%	56%

Table 36. Attack Success Rate overall results

9. Results Discussion

Overall, the models perform well on clean data, but their vulnerability to adversarial attacks varies significantly depending on the attack type and the classification algorithm used. The Decision Tree classifier outperforms the other classifiers in terms of accuracy, achieving 100% accuracy on all types of attacks except for FGSM with e=0.6 and DeepFool. However, its F1 score and AUC score are relatively lower than the other classifiers, indicating that it may not perform well in detecting adversarial attacks. The SVM classifier also performs relatively well, achieving 100% accuracy on normal data and FGSM with e=0.4, but its accuracy drops significantly under other types of attacks. The K-Nearest Classifier achieves high accuracy on normal data and FGSM with e=0.4, but its accuracy drops significantly under other types of attacks, especially under FGSM with e=0.8 and DeepFool. The K-Means algorithm performs the worst among all the classifiers, achieving a very low accuracy of 32.77% on normal data and dropping significantly under adversarial attacks, especially the FGSM attack with epsilon 0.8 and the DeepFool attack.

Regarding the attack success rate of each attack:

- The JSMA attack has the lowest success rate of all models, except for K-Means, where it has a relatively high success rate of 30.08%.
- The FGSM attack with epsilon 0.4 has a low success rate on all models, except for SVM, where it has a moderate success rate of 6.32%.
- The FGSM attack with epsilon 0.6 has a moderate success rate on all models, with the highest success rate being on the Decision Tree and SVM models.
- The FGSM attack with epsilon 0.8 has a high success rate on all models, with the highest success rate being on the K-Means model.

- The DeepFool attack has the highest success rate on all models, with the highest success rate being on the K-Means model.

These results suggest that the K-Means model is the most vulnerable to adversarial attacks, especially the FGSM attack with epsilon 0.8 and the DeepFool attack. The Decision Tree and SVM models are relatively less vulnerable, while the K-Nearest model is in between. The JSMA attack is the least effective among the attacks considered in this analysis.

10. Conclusion

In conclusion, machine learning has revolutionized the field of intrusion detection systems, providing powerful tools to detect and prevent attacks in real time. There are several types of machine learning techniques, including supervised and unsupervised learning, deep learning, and ensemble methods, which can be used to detect anomalies and identify potential threats.

In this thesis, we have focused on building an IoT intrusion detection system. Our experiments showed that deep learning techniques can effectively detect attacks with high accuracy. We have also implemented adversarial attacks, including Jacobian Saliency Map attack, Fast Gradient Sign Method, and DeepFool, to evaluate the robustness of the system. Our results demonstrated that the Machine learning models were vulnerable to these attacks, highlighting the need for improving the system's resilience to adversarial attacks. Addressing the research questions made in the Introduction section:

RQ1: The study revealed that the effectiveness of adversarial attacks on intrusion detection systems in the IoT domain varies significantly depending on the attack type and the classification algorithm used. While the models perform well on clean data, their vulnerability to adversarial attacks is a concern, particularly for the K-Means algorithm, which is the most susceptible to such attacks.

RQ2: Among the attacks considered in this analysis, the DeepFool attack had the biggest impact on the performance of intrusion detection systems in the IoT domain, achieving the highest success rate on all models.

RQ3: The study found that the supervised learning methods (Decision Tree and SVM) are relatively less vulnerable to adversarial attacks than the unsupervised learning method (K-Means), while the K-Nearest model is in between. However, the performance of these models in detecting adversarial attacks varies depending on the attack type and the classification algorithm used.

When an adversarial attack is performed on an unsupervised algorithm, it can cause the model to produce incorrect outputs or completely fail. This is because the algorithm is highly sensitive to small changes in the input data, and the attack can exploit this sensitivity to manipulate the model's output. In contrast, supervised algorithms have been shown to be more robust to adversarial attacks because they are trained on labeled data, which provides them with more context and information about what is expected from the input. They can use this contextual information to detect and resist adversarial attacks. Adversarial attacks on unsupervised algorithms reveal the importance of carefully considering the robustness of machine learning models to adversarial attacks, especially in situations where security and reliability are critical.

Future research in this area can focus on developing new techniques to enhance the security of the IoT intrusion detection system, such as using generative models, improving the quality of the dataset, and integrating explainable AI to understand the model's decision-making process. Also, could explore the effectiveness of other attack types or investigate alternative machine learning algorithms. Overall, this thesis has shown the potential of using machine learning in the development of IoT intrusion detection systems, and the importance of addressing security challenges to ensure the reliability and effectiveness of these systems in real-world scenarios.

References

[1] Logistic Regression pp. 223 – 237. Available at: https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12 .pdf

[2] Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica 31 (2007). Pp. 249 – 268.

[3] Vapnik, V. N. (1995). The Nature of Statistical Learning

[4] Jain, A.K.; Murty, M.N.; Flynn, P.J. Data clustering: A review. ACM Comput. Surv. 1999, 31, 264–323.

[5] Tapas Kanungo, D. M. (2002). A local search approximation algorithm for k-means clustering. Proceedings of the eighteenth annual symposium on Computational geometry. Barcelona, Spain: ACM Press.

[6] "A comparison of clustering criteria for document datasets" by A. Jain and R. Dubes (1988)

[7] "Deep Learning" by Y. LeCun, Y. Bengio, and G. Hinton (2015): This paper provides an overview of deep learning, including the historical background, basic concepts, and recent developments.

[8] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996) 123–140.

[9] Ruta, Dymitr, and Bogdan Gabrys. "Classifier selection for majority voting." Information fusion 6, no. 1 (2005): 63-81.

[10] Freund Y, Schapire R. Experiments with a new boosting algorithm. In: Proceedings of the Thir- teenth National Conference on Machine Learning; 1996. pp. 148–156.

[11] Wolpert DH. Stacked generalization. Neural Networks. 1992; 5(2):241–259.

[12] Breiman L (2001) Random forests.

[13] Friedman, J. (2001). Greedy boosting approximation: a gradient boosting machine.

[14] Szegedy et al., 2014; Goodfellow et al., 2015

[15] Eykholt et al., 2018

[16] inlayson et al., 2019

[17] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks.

[18] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep learning approach for network intrusion detection in software defined networking.

[19] Qureshi, A.U.H.; Larijani, H.; Mtetwa, N.; Javed, A.; Ahmad, J. RNN-ABC: A New Swarm Optimization Based Technique for Anomaly Detection.

[20] S. Kumar, E. Spafford, "A Software architecture to Support Misuse Intrusion Detection" in The 18th National Information Security Conference, pp. 194-204. 1995.

[21] S. Kumar, "Classification and Detection of Computer Intrusions", Purdue University, 1995.

[22] K. Ashton, "That 'internet of things' thing.", RFID Journal 22, no. 7, 2009, pp.97-114.

[23] I. Andrea, C. Chrysostomou and G. Hadjichristofi, "Internet of Things: Security vulnerabilities and challenges," 2015 IEEE Symposium on Computers and Communication (ISCC), pp.180-187,Larnaca, 2015.

[24] Wahid, Abdul, P. Kumar, "A Survey on attacks, Challenges and Security Mechanism In wireless Sensor Network", JIRST- International Journal for Research in Science & Technology, Volume 1, Issue 8, pp. 189-196,January 2015.

[25] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. arXiv 2013, arXiv:1312.6199.

[26] Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The

Limitations of Deep Learning in Adversarial Settings.

[27] Goodfellow, I., J. Shlens, and C. Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv 1412.6572. December.

[28] Alexey Kurakin, Ian J. Goodfellow, Samy Bengio. 2017. ADVERSARIAL EXAMPLES IN THE PHYSICAL WORLD

[29] Martin Abadi Paul Barham Jianmin Chen Zhifeng Chen Andy Davis Jeffrey Dean Matthieu Devin Sanjay Ghemawat Geoffrey Irving Michael Isard Manjunath Kudlur Josh Levenberg Rajat Monga Sherry Moore Derek G. Murray Benoit Steiner Paul Tucker Vijay Vasudevan Pete Warden Martin Wicke Yuan Yu Xiaoqiang Zheng - TensorFlow: A system for large-scale machine learning

[30] BREIMAN, L., 1999, Random forests - Random Features. Technical Report 567, Statistics Department, University of California, Berkeley,

[31] Srivastava et al. 2014, Dropout: A simple way to prevent neural networks from overfitting.

[32] DeepFool: a simple and accurate method to fool deep neural networks - Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard E´colePolytechniqueFe´de´raledeLausanne

[33] Performance Evaluation of Adversarial Attacks: Discrepancies and Solutions - Jing Wu, Mingyi Zhou, Ce Zhu, Yipeng Liu, Mehrtash Harandi, Li Li

[34] VasileiosPant/Adversarial-Attacks-against-NIDS: https://github.com/VasileiosPant/Adversarial-Attacks-against-NIDS