ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

ADAPTIVE AND HIERARCHICAL CLASSIFIER FUSION APPROACHES FOR NETWORK ATTACK DETECTION

M.Sc. THESIS

Erkan AS
(708151025)

Applied Informatics Department

Information and Communications Engineering Programme

Thesis Advisor: Assoc.Prof.Dr. Behçet Uğur TÖREYİN

SEPTEMBER 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

AĞ SALDIRISI TESPİTİ İÇİN UYARLANIR VE AŞAMALI SINIFLANDIRICI TÜMLEŞTIRME YAKLAŞIMLARI

YÜKSEK LİSANS TEZİ

Erkan AS
(708151025)

Bilişim Uygulamaları Anabilim Dalı

Bilgi ve Haberleşme Mühendisliği Programı

Tez Danışmanı: Doç.Dr. Behçet Uğur TÖREYİN

EYLÜL 2019

Erkan AS, an M.Sc. student of ITU Informatics Institute Engineering and Technology 708151025 successfully defended the thesis entitled "ADAPTIVE AND HIERARCHICAL CLASSIFIER FUSION APPROACHES FOR NETWORK ATTACK DETECTION", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc.Prof.Dr. Behçet Uğur TÖREYİN** Istanbul Technical University	.....
Jury Members :	**Assoc.Prof.Dr. Yusuf YASLAN** Istanbul Technical University	.....
	Assoc.Prof.Dr. Gökhan BİLGİN Yildiz Technical University	.....

Date of Submission : **9 September 2019**
Date of Defense : **10 September 2019**

To my dear family and advisor,

FOREWORD

I am thankful to Behçet Uğur TÖREYİN for his guidance and support. I also thank to the lecturers at the university who share their valuable knowledge. I would like to thank my business partners for their understanding during my master studies. Last but not least, I would like to thank my family for their endless support, guidance and unconditional love throughout my life.

September 2019

Erkan AS
(Electronic Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

IoT : Internet of Things
UTM : Unified Threat Managements
AI : Artificial Intelligence
kNN : k-Nearest Neighbors
CART : Classification And Regression Trees
ANN : Artificial Neural Network
ADF : Adaptive Decision Fusion
EADF : Entropy-Functional-Based Online Adaptive Decision Fusion
MLP : Multilayer Pertecptron
LR : Logistic Regression
CPU : Central Processing Unit
RF : Random Forrest
ERT : Extremely Randomized Trees
IDS : Intrusion Detection System
SVM : Support Vector Machines
GB : Gradient Boosting
GBM : Gradient Boosting Machines
ML : Machine Learning
LDA : Linear Discriminant Analysis
QDA : Quadratic Discriminant Analysis
NaN : Not a Number
NA : Not Available
O.S : Operating Systems
R2L : Remote to User
U2R : User to Root
DoS : Denial of Service
DDoS : Distributed Denial of Service
DR : Detection Rate
FP : False Positive
TP : True Positive
TN : True Negative
TP : True Positive
FPR : False Positive Rate
CIC : Canadian Institute for Cybersecurity
PAC : Probably Approximately Correct

LIST OF TABLES

Page

LIST OF FIGURES

Page

ADAPTIVE AND HIERARCHICAL CLASSIFIER FUSION APPROACHES FOR NETWORK ATTACK DETECTION

SUMMARY

In this thesis we aimed to develop, a machine learning based system that can detect anomalies and intrusions in computer networks with high performance. For this purpose, three-step hierarchical methods were developed. An attack type detection model was created by using different machine learning algorithms, and their outputs and weights were combined with decision fusion methods. In this way, a IDS with a high attack detection rate and a low false alarm rate was developed. The proposed method essentially has three steps. In the first step, we detect whether network traffic is normal or abnormal/attack. If the network traffic is normal, it does not enter the second step as normal, but if network traffic is abnormal data will be forward to the next step. Several machine learning algorithms are used for building anomaly detection models. In the second step, we built up models for determining attack type. In the first two steps, twelve different machine learning algorithms were used separately. The algorithms with the highest scores were used. In addition, in order to achieve better performance and solve the problem of class imbalance in the data set, attack classes were grouped in the second stage. In the last step, the best four classifiers with the highest scores are used for decision fusion, which aims to detect attack type better. Majority voting, weighted average based majority voting and, online adaptive decision fusion methods are compared. As experimental result, the proposed method has high intrusion detection rate and accuracy rate. A system with a 99.98% F1-score with a 99.98% detection rate is developed by regrouping the classes. Also, 99.84% accuracy score and 99.83% F1-score were obtained in the experiments conducted considering the original classes in the data set.

AĞ SALDIRISI TESPİTİ İÇİN UYARLANIR VE AŞAMALI SINIFLANDIRICI TÜMLEŞTIRME YAKLAŞIMLARI

ÖZET

Günümüzde bilgi teknolojilerinin gelişmesi ve yaygınlaşmasına paralel olarak bu sistemlerin güvenliğinin sağlanması sorunu ortaya çıkmıştır. Çok farklı cihaz ve teknolojinin internete bağlanması sistem ve ağı daha da karmaşıklaştırmıştır. Daha fazla bilgisayar, kişi ve cihazın internet ve bu ağa bağlanması saldırganların motivasyonunu artırmış ve daha fazla saldırıda bulunmaya ve güvenlik açığı bulmaya çalışmaktadırlar.

Saldırganlar sadece şirket ve kişilere saldırmamakta devlet kurumlarına da sızmaya, veri çalmaya veya sistemi devre dışı bırakarak onarılması zor zararlar vermektedirler. Bu artan güvenlik zafiyetlerine karşı zamanla çeşitli yazılımsal ve donanımsal güvenlik çözümleri geliştirilmiştir. Bu güvenlik çözümleri zaman içinde gelişmiş ve tümleşik güvenlik çözümleri haline gelmiştir. Ayrıca saldırıganların sayısının artması ve yazılım ve donanımdaki gelişmeler saldırganların elindeki araçların güçlenmesine yol açmıştır. Bu sürekli değişen ve farklı varyantları çıkan virus, malmware, trojan, ve saldırı araçlarının tespit edilmesi zorlaşmştır. Ayrıca saldırı araçları çok sık değiştiği için buna uygun olarak güvenlik sisteminin güncellenmesi belli bir zaman almakta ve bundan sistem zafiyet yaşamaktadır.

Bu sık değişen ve çeşitlenen saldırılar için mevcut sistem ve ağı öğrenen anomali tabanlı saldırı tespit sistemleri geliştirilmeye başlanmıştır. Makine öğrenmesi, yapay zeka ve bunları kullanarak model oluşturup entegre edecek yazılım ve donanımlardaki gelişmelerle beraber anomali tabanlı sadırı tespit sistemleri yaygınlaşmaya başlanmıştır.

Bu çalışmada makine öğrenme algoritmaları kullanılarak yüksek tespit ve düşük yanlış alarma sahip saldırı tespit sistemi geliştirilmsi hedeflenmiştir.İlgili çalışmaların gerçekleştirilmesi ve önerilen metodun sınanması için 2017 yılında Kanada Siber Güvenlik Ensitüsü tarfından oluşturulan veri kümesi kullanılmıştır. Bu veri kümesi laboratuvar ortamında oluşturulmuş günümüzün modern ağ trafiğini modellemiştir. Bu veri kümesi günlük normal aktivitelerini ve en yaygın saldırı türlerini içermektedir. Yaklaşık 2,8 milyon örnekten oluşmakta olup bu ağ trafik akışının 80 tane özelliği bulunmaktadır. Toplam paket boyutu, maksimum paket boyutu, ACK bayrak sayısı, geliş paket sayısı, oturum süresi vb. özellikleri içermektedir.

Önerilen sistemde öncellikle veri kümesi bir takım ön veri işleme teknikleri kullanılarak temizlenip, hatalardan giderilmiştir. Ayrıca bazı kayıtlar silinmiş ve boş olan özellikler sınıf ortalaması ile doldurulmuştur. Veri ön işleme tekniklerinden sonra sistem karmaşıklığını azaltmak ve başarımı artırmak için özellik seçimi yapılmıştır. Özellik seçme yöntemi olarak, Anova F-puanını temel alan ve en yüksek skora sahip özellikler seçilmiştir. Özellik seçiminden sonra veri kümesi sınama ve eğitim kümesi olarak ayrılmış (%75 eğitim, %25 sınama) olup bu eğitim kümesi model eğitilmesi,

doğrulama ve parametre optimizasyonunda kullanılmamıştır. Eğitim kümesi beş parçaya bölünmüş ve %15 doğrulama, geliştirme kümesi olarak kullanılmıştır. Sonuçlar kısmında belirtilen sonuçlar, sınama kümesi üzerindeki deney sonuçlarını göstermektedir.

Önerilen sistem 3.aşamadan oluşmaktadır. 1.aşamada anomali tespiti için tüm saldırı türleri tek sınıfta toplanmış ve anormal (-1) olarak yeniden etiketlendirilmiştir. Normal trafik verileri (1) olarak etiketlenmiştir. Daha sonra oluşturulan bu veri kümesi %25 sınama, %75 eğitim kümesi olarak sınıf ağırlıkları korunarak ayrılmıştır. 1. aşama için 12 tane sınıflandırıcı (k en yakın komşu, destek vektör makineleri, karar ağaçları, topluluk yöntemleri vb.) ile eğitilip sınanmıştır. Öncellikle her sınıflandırıcı için olası hiper parametre listesi oluşturulmuş ve scikit-learn kütüphanesinden randomizedsearchcv kullanılarak en iyi hiper parametreler belirlenmiştir. Beş katmanlı çapraz doğrulama yöntemi kullanılarak çapraz doğrulam işlemi uygulanmıştır. Yani veri kümesinin %15 doğrulama/geliştirme kümesi olarak kullanılmıştır. Sınıf ağırlıkları farklı olduğu için en iyi başarım F1 puanına göre belirlenmiştir. Veri kümesi büyük (2,8 milyon örnek, 70 özellik) ve olası parametre kümesi büyük olduğu için yüksek başarımlı bilgisayarlar (UHEM) kullanılmıştır. Bu adımda saldırı tespiti için %99,92 doğruluk ve F1-puanına sahip model geliştirilmiştir. Bu aşamda "Extremely Randomized Trees" algoritması en başarılı sonucu vermiştir.

İkinci adımda saldırı/girişim türünün belirlenmesi hedeflenmektedir. Bunun için sadece girişim/saldırı olan veri kullanılmıştır. Normal trafik bu adıma girmeden devam edecektir. Anormal olan trafik bu adımda işleme alınacaktır. Bunun için veri kümesinden sadece girişim/saldırı olan veriler alınmıştır. Sınıf etiketleri veri kümesi açıklamasında belirlenen etiketler bulunmaktadır. Bu adımda yine eğitim ve sınama aşamları için 12 tane sınıflandırıcı (k en yakın komşu, destek vektör makineleri, karar ağaçları, topluluk yöntemleri vb.) kullanılmıştır. Birinci aşamada gerçekleştirilen, sınıf ağırlıkları gözetilerek eğitim ve sınama kümelerine bölünmesi, en iyi hiper parametrelerin belirlenmesi, ve çapraz doğrulama işlemleri bu aşamada tekrar gerçekleştirilmiştir. Bunun yanında, veri kümesinde bazı saldırı türleri çok az örnek içermektedir. Bazı saldırı türleri de birbirine benzerlik göstermektedir. Daha iyi saldırı türü tespti için ikinci adımda ek olarak bu saldırı türleri tekrar sınıflandırılarak eğitim ve sınama işlemleri gerçekleştirilmiştir. Yapılan bu sınıf gruplandırılmasından sonra daha yüksek başarım ve tespit oranına sahip model elde edilmiştir. İkinci adım sonucunda en yüksek başarıma sahip dört tane sınıflandırı seçilmiştir. Bu sınıflandırıcılar kullanılarak son aşamada karar birleştirme metodları sınanmıştır. Veri kümesinin orjinal sınıfları kullanılarak yapılan deneylerde, %99,83 F1-puanına ve %99.84 doğruluğa sahip sistem geliştirilmiştir. Saldırı içeren örneklerin sınıfların yeniden gruplandırılması ile gerçekleştirilen deneylerde %99.98 doğruluk ve F1-puanına sahip sistem geliştirilmiştir.

Son adımda, ikinci adım sonunda sonunda seçilen en yüksek başarıma sahip dört sınıflandırıcı ile karar birleştirme metodları sınanmıştır. Bu adımda karşılaştırma için 3 tane farklı metod kullanılmıştır. (1) En çok oyu alan sınıfın seçilmesine dayanan "majority voting". (2) sınıflandırıcıların çıktılarında sınıf olasılıkların ağırlandırılarak karar birleştirme işlemini sağlayan metod. (3) Sınıflandırıcı ağırlıklarını her örnekte sınıflandırıcı hatasına göre güncelleyen "ADF" metodu kullanılmıştır. Bu üç karar birleştirme metodu ile sistem geliştirilmiştir.Bu farklı üç metodla ile yapılan deneylerde en yüksek başarıma, olasılık tabanlı ağırlıklandırılmış karar

birleştirme algoritması sahip olmuştur. %99,84 doğruluk ve F1-puanına sahip sistem geliştirilmiştir. Ayrıca Makro-F1 olarak adlandırılan her sınıfın ağırlıkları ve örnek sayısı gözetilmeden yapılan puanlamada %92,61 başarıma ulaşılmıştır.

Geliştirilen bu üç aşamalı hibrit saldırı tespit sistemi benzer çalışmalara göre daha yüksek doğruluk ve tespit oranına sahiptir. İlgili çalışmalar karşılaştırmalı olarak gösterilmiştir.

1. INTRODUCTION

1.1 Motivation

With the rapid developments in information technologies, the number of internet users has increased rapidly. According to Statista's April 2019 report, the number of internet users reached 4.3 billion [3]. Also, nowadays, mobile devices, things, cars, industrial devices have become connected to the Internet. Today, the number of devices connected to the Internet is almost 26 billion and is expected and planned to reach 75 billion in 2025. Also, the advancements in cloud technology, mobile devices, and the Internet of things increase this internet traffic [4].

With the development of the Internet, cybersecurity concern becoming important. Various solutions have been implemented against increased attacks. One of the most critical systems for defending and detecting internal and external intrusions/ attacks is Intrusion detection systems (IDS). IDS could provide detecting the various type of attacks in real-time.

Intrusion Detection systems could be classified as signature based and anomaly-based by their detection method. Signature based intrusion detection systems which contributed firstly, have a record in the database for each attack type. This attack signature consists of the actions had taken by the attacker. For each type of attack, a signature must be created, and the database must be updated. These class intrusion detection systems are vulnerable to unknown attacks (zero-day attacks). Because different versions of attack types are continually evolving from attackers, computer networks can be vulnerable to such attacks.

Moreover, today's internet traffic more than half is encrypted, which using SSL/TLS protocols, and the rate of encrypted traffic is increasing day by day [5]. Signature-based intrusion detection systems can not work efficiently because the content of this traffic has not been adequately investigated.

Due to such disadvantages of signature-based intrusion detection systems, anomaly based intrusion detection systems have been developing day by day and are being used more widely. Anomaly-based intrusion detection systems, the attack can detect the attack even if there is no attack signature in the existing database. It can also detect data from data properties, even if data traffic is encrypted. For example, It could detect attack with the amount of data in a specific time interval, idle time, packet size, etc. properties of the attack. Anomaly-based IDS can learn the normal flow of its existing network and identify devices and users with different behaviors.

In parallel with the development of machine learning methods, anomaly-based intrusion detection systems showed a rapid development. First, in 1998, sample data sets were published, and academic and business researchers began developing various models. These works continued with KDD Cup 1999 data set published in 1999. Various data sets have been published, and researches and improvements have been made. However, the studies focused on the updated version of the KDD99 data set, NSL-KDD (2007). Research has shown that today's internet traffic varies from ten years ago. Encrypted traffic has increased, the types of attacks have changed, and the characteristics of this internal and external traffic has changed. For this purpose, current data sets were created and published. UNSW-NB15 and CICIDS2017 data sets were created by considering the current network environment and attack types.

Today, the increase in Internet and internal network traffic has revealed the need for IDS systems to work faster and more efficiently. Besides, a very high attack detection rate is needed since even an attack can cause too much damage to the network request. In these systems, normal or abnormal detection of traffic is critical, not to identify the type of attack of the priority. However, it is necessary to determine the type of attack and to take into account for eliminating the deficits in the network system. Also, it should have a low false alarm rate under the heavy network traffic in today's conditions and should not increase the workload of the network administrator. Considering all these requirements, we build an intrusion detection system with a high detection rate (DR), low false alarm rate (FPR), and for the current network traffic.

1.2 Literature Review

The studies in the literature can be classified into several categories. First of all, in these studies, models were created by using different data sets, and experiments were performed. Table 1.1 provides detailed information about these data sets. As new data sets are created and used in academic studies; models were created, and experiments were carried out. KDD Cup 1999 [6] and NSL-KDD [7] data sets were used extensively in these studies. The NSL-KDD data set, which is an updated version of KDD Cup 1999 Data, which was created in 2007, is also included as a reference data set.

We can distinguish the studies in the literature according to the method used (single and multi-step), the machine learning methods used, and the use of deep learning. Single-step and multi-step methods were used for attack detection and determination of attack type. Besides, different feature selection methods and different machine learning methods are applied to the data sets mentioned above to increase the model performance. In addition to these classical methods, special methods have been developed for intrusion detection area. In these methods, different feature selection, partitioning of data set, classifier selection, etc. techniques were applied. In more recent studies, deep learning has been used and combined with the different algorithms mentioned above. There are a lot of academic studies as security is a trend and critical issue.

Another categorization is to divide the data sets into partitioning methods. First, the data set can be partitioned for binary classification as normal and abnormal. Or, a multi-class classifier can be created, assuming each attack type is a class. Another method is to perform binary classification for each attack type. Although this method has high performance, it is not possible to apply it for all attack types in real life. Since there are many types of attacks, it is not feasible to create and train models for each attack type.

Table 1.1 : Attacks With IDS Data Sets [1].

Data Set	Attacks	Publication Year
Botnet	botnets (Menti, Murlo, Neris, NSIS, Rbot, Sogou, Strom, Virut, Zeus)	2010, 2014
CIC-DoS	Layer7 DoS attacks (ddossim, Golden-eye, hulk, rudy, Slow-httptest, Slow-loris)	2012, 2017
CICIDS2017	botnet, cross-site-scripting, DoS (Hulk, Golden-eye, Slow-loris, Slow-httptest), DDoS, heartbleed, infiltration, SSH-brute-force, SQL injection	2017
CIDDS-001	DoS, port-scans (ping-scan, SYN-Scan), SSH-brute-force	2017
CIDDS-002	port scans (ACK-Scan, FIN-Scan, ping-Scan, UDP-Scan, SYN-Scan)	2017
CTU-13	botnets (Menti, Murlo, Neris, NSIS, Rbot, Sogou,Virut)	2013
DARPA	DoS, privilege escalation (remote-to-local and user-to-root), probing	1998, 1999
DDoS 2016	DDoS (HTTP flood, SIDDOS, ICMP flood, UDP flood)	2016
ISCX 2012	Four attack scenarios (Infiltrating the network from the inside; HTTP DoS; DDoS using an IRC botnet; SSH-brute-force)	2012
KDD CUP 99	DoS, U2R, Probe, U2L	1998
Kyoto 2006+	Var. attacks against honeypots (backscat-ter, DoS, exploits, malware, port-scans, shellcode)	2006
NSL-KDD	DoS, U2R, Probe, U2L	2007
UGR'16	botnet, DoS, port-scans, SSH-brute-force, spam	2016
UNSW-NB15	back-doors, DoS, exploits, fuzzers, generic, port-scans, reconnaissance, shellcode, spam, worms	2015

Many different studies have been carried out for intrusion detection [8–13]. These studies are mainly host-based and network-based. In parallel with the development of algorithms, these studies have been conducted primarily in the form of using single algorithms, hybrid approaches, multi-step approaches, decision fusion, development of feature selection algorithms. It is aimed to increase the detection rate of interference and to reduce false alarms. Also, new data sets were created and published in parallel with the developments in internet technologies. New studies have been carried out with

these data sets. Examples of these sets are the UNSW-NB15 [14], CICIDS2017 [15], CICIDS2018 [16] data sets.

With the increasing use of IoT, cloud technology, and mobile technologies internal and external network traffic created by these devices increased; studies on intrusion detection systems in these areas have increased in recent years. Nowadays, it is aimed to make fast transactions with larger data using big data processing tools besides these emerging areas. Besides, the studies have come to the fore with deep learning. Despite these developments, similar data sets continue to be used.

Hybrid IDS with two-stage intrusion detection/attack detection approach based on two-class classification and kNN methods proposed by Li and Yu. [11] In the first step for every attack category a binary classifiers (normal/attack category) trained by using C45 decision trees. If attack type could not be determined or is ambiguous kNN algorithm detect attack type. Experiments have evaluated on NSL-KDD data set. Four classifiers trained based on DoS, Probe, U2R, R2L attack categories.

Timcenko and Gajin [17] have evaluated ensemble methods and classification. Bagged trees LogitBoost, RUSboost, AdaBoost, supervised algorithms classification performance, analyzed and compared. The experiments have evaluated on small set of UNSW-NB15 data set. On that small set attack category and normal/anomaly data binarized. As results. Bagged Tree and GentleBoost perform with the highest accuracy and ROC-AUC score. RUSBoost has the lowest score.

In [9], the CICIDS2017 data set is used for anomaly detection and attack type classification. Proposed method is composed of three classifiers, the first for evaluating the data type as being attack or benign, the second classifier is for detecting attack category (include normal data) and the last classifier for combining two classifiers. In [9], the whole data set was not used. Data resampled from 2.5 million records to 40K records. The proposed method increases the detection rate of attack types. However, the resampling rate is too high and may lead to degrade data set. Detection rate is 94.47% and accuracy score is 96.66%.

In [18], authors aim to detect and to categorize different attacks on UNSW-NB15 data set. For that, firstly they use best-first feature selection for maximizing overall accuracy performance and reduce complexity of model. Linear Regression and

Random Forest supervised techniques used on UNSW-NB15 data set. Single-type attack categorization and step-wise attack categorization methods used in experiments. 99% anomaly detection accuracy and 93.6% attack- type detection achieved. However, the framework in [18], unrealistic for real-world implementation. Single-type attack categorization could not apply if apply system complexity too high to be used in a real network environment. Moreover, attack type detection is still between 2% to 97%.

In [19], extracting features of flow extracted from raw data. Then apply a deep hierarchical network method which a hybrid neural network using LeNet-5 and LTSM neural network structures. The first layer is based on LetNet-5 CNN to extract spatial features of the flow, and the second layer uses the LTSM network to extract temporal features of the flow. These two networks trained simultaneously. Binary classification (normal/abnormal) and attack type detection with normal flow classification experiments made on CTU and CICIDS2017 data sets. F1-score of on binary classification is up to %99.88, and multi-class classification is 99.99%. However, details of detailed classification does not share. The class weight of normal traffic and dos attacks are too high, and this could lead to show high detection rate.

In [12], a novel classifier model using stacked Nonsymmetric Deep Auto-Encoder (NDAE) and Random Forest Algorithm are used for detecting network attacks. NDAE is unsupervised feature learning which provides nonsymmetric data dimension reduction. The model evaluated on KDD Cup99 and NSL-KDD datasets. Five classes (normal, UR2, R2L, DoS, Probe) and 13 class classification models (well-known attacks and normal traffic) trained and evaluated. Experiments show that have higher f-score and detection rate compared to other deep learning methods. Also have less training and testing time.

1.3 Proposed Method

A hybrid three-stage system has been designed for the development of an intrusion detection system capable of providing the requirements specified in the introduction. The proposed method is shown in Figure 1.1. In the first stage, it will be determined by using machine learning algorithms whether the current traffic is normal or anomaly. At this stage, the algorithm with the highest success will be selected with performing data preprocessing operations and feature selection. If the traffic is normal, it will continue without entering into the second stage, and if it is an anomaly, the activity is further

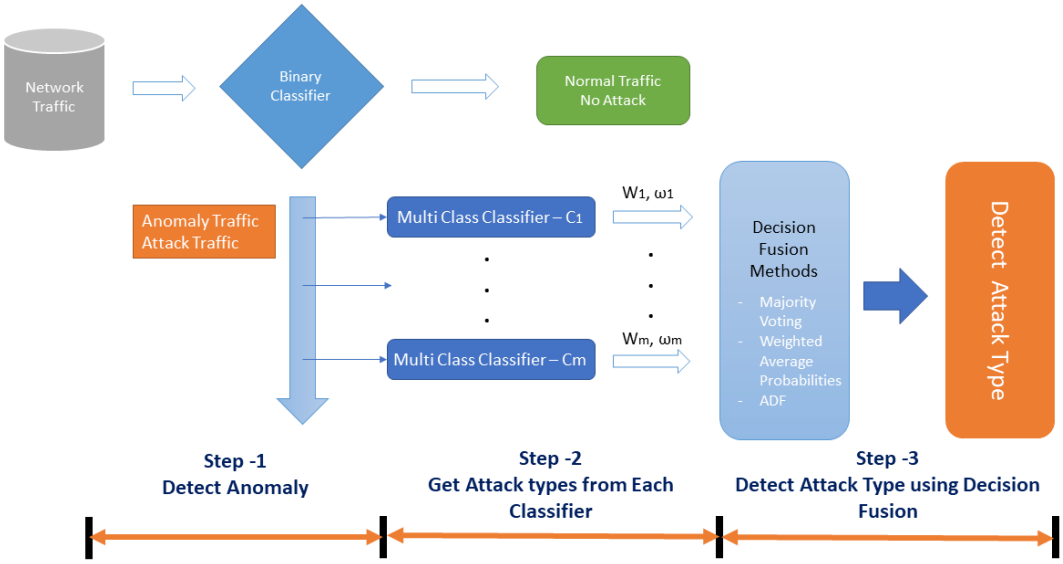

Figure 1.1 : Proposed Method

analyzed in the second step. In the second step, a multi-class multiple classifier is used, and input data is only anomaly data. Four classifiers with the highest estimation F1-score will be selected from these classifiers. The type/category of the attack will be determined by these classifiers. The reason for selecting more than one classifier in the second stage is the failure to achieve the required detection rate and accuracy rate with a single classifier. In the third stage, these models/classifiers that are trained and created in the second stage will be formed with a system with higher success by using decision fusion methods. In the third stage, three decision-fusion methods will be compared, namely, majority voting, weighted average based decision fusion, online adaptive decision fusion. In the majority voting algorithm, decision fusion is achieved by the selection of the class with the most votes among the four classifiers. In the second method, the output of classifiers is class probabilities. The type/category of the attack is determined according to the class probabilities coming from each classifier. With the third algorithm, online adaptive decision fusion [20], class weights are dynamically updated according to the amount of error. Class weights are updated for each instance, and the decision is made by classifiers results and weights. With this three-stage method, a higher performance system was designed.

CICIDS2017 data set have been studied in order to verify the proposed method. Besides, data preprocessing operations were performed for each data set. In addition, statistical, information gain, and select from model (SVM, RF) are used for feature selection. Test results are presented comparatively. Moreover, the best

hyperparameters of the classifiers used to make this study a reference study were determined. Also, data sets are divided into training, validation/development, and testing. For cross-validation and finding the best hyper-parameters, k-fold verification was used. In terms of reliability, the test set has not been used in these steps.

1.4 Thesis Layout

In this chapter of the thesis, the aim of the thesis, literature research, and the proposed method are given. In the second chapter, theoretical information about intrusion detection systems, machine learning algorithms used in experiments and decision combining algorithms are given. The third chapter contains a detailed description of the proposed method, feature selection, data preprocessing, and performance metrics. The fourth chapter contains information about the data set used, how it is obtained and created, and the explanations of the classes. The fifth chapter contains the experiments and the results for each step. It also includes a comparison with other studies in the literature. In the last chapter, information about the results and future work are given.

2. THEORETICAL BACKGROUND

2.1 Intrusion Detection Systems

Anderson proposed the concept of the intrusion detection system and the first IDS concept in 1980 in an article entitled "Monitoring and Surveillance of Computer Security Threats." James Anderson has suggested that audit trails (log records) could contain vital information in understanding user/device behavior and monitoring abuse. To develop this idea, understanding the importance of audit data has led to new developments in the monitoring of network and system of almost every network and operation systems [21]. Anderson's contributions was the basis for the design and development of future intrusion detection and prevention systems (IDS).

Intrusion Detection Systems (IDS) provide the security of computer or network systems which are software or hardware security systems. IDS could detect external and internal attacks or unauthorized access abuse within the organization. The other definition of IDS is: The Intrusion Detection System (IDS) is a security mechanism (software/hardware) that collects and analyzes information about events occurring in the computer system or computer network, detects security policy violations and detects the traces of the attack. The primary role of the intrusion detection system in computer systems or networks is to deal with network attacks and to help computer systems and IT staff.

While a Signature Based IDS can not identify unknown attacks or which have not signature in the database. Today's computer networks needs fast reaction to unknown attacks which calls zero-day attacks. Despite this weakness of signature-based IDS, it is still a classic security solution used for commercial/industrial products. Rather, Network Anomaly Based IDS detects serious intrusions but is often faced with potential development methodology challenges. The following challenges can be examined from a methodology based on anomalies, naming the creation of a purely legitimate profile with variations from an anomaly [22]. Figure 2.1 show types of intrusion detection systems. In this study, we focus on network-based anomaly

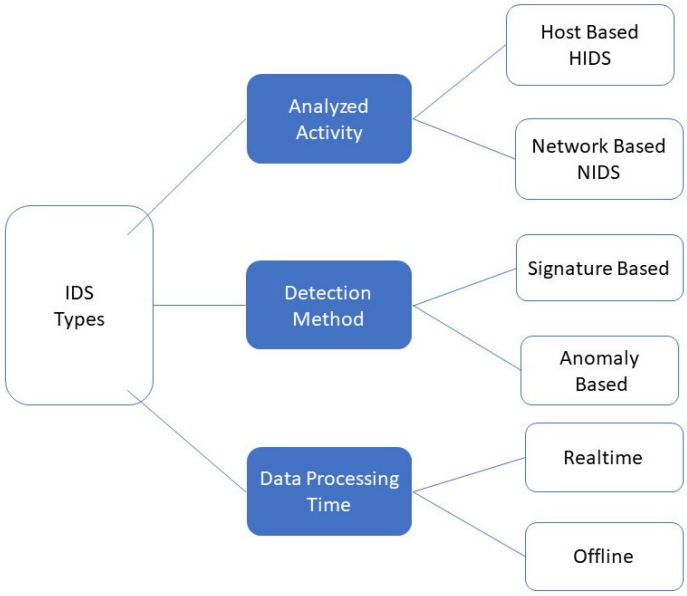

Figure 2.1 : Types of Intrusion Detection Systems.

detection based IDS. False Positive errors occur if a normal behavior detects in the attack area, and False Negative errors occur if abnormal one detects in the normal area. While both errors are unsafe for network data, IDS performance is usually measured by the False Negative, as attack alerts are not available on real network systems. When creating the architecture of a Network Anomaly Based IDS cognitive and extensible, attacks from the normal profile are very difficult to differentiate, because complex malicious activities such as stubborn and spy attacks may fit to almost match normal patterns. For several reasons, real-time intrusion detection is also challenging. First, the network traffic features may contain a variety of noisy or irrelevant features. Secondly, concerning the above problems, the lightweight of detection methods must be carefully taken.

Today, security solutions consisting of modern, distributed, and different layers have been developed against the rising security risks and threats. Instead of a firewall, unified threat management (UTM), including IDS, servers, clients, and software solutions, are used. These software solutions can include components such as anti-virus, anti-malware, firewall, threat detection and blocking, content access security. With improvements in machine learning and artificial intelligence, vendors have added ML-based prediction systems to classic threat and interference detection and prevention mechanisms. These include Trend Micro, Symantec, Kaspersky, and McAfee. Besides, there is a new generation of fully ML and AI-based security

solutions. Sentinel One [23] and Carbon Black [24] are examples of this generation of solutions. Threat detection and prevention, abnormal behavior detection process can do with case-based content and process control. These solutions are more successful than traditional methods, successful against zero-days and have a higher detection rate. Also, they can detect anomaly by learning the existing network, system, and user behavior.

2.2 Machine Learning Algorithms

In this section, the machine learning algorithms used in the experiments and used to develop the proposed method will be discussed.

2.2.1 k-Nearest neighbors

The k-nearest neighbor classification assigns an example to the class most represented among its neighbors. It is based on the concept that the closer the instances are, the more likely they are to be of the same class. We can use the same classification strategy, provided that we have a reasonable similarity or distance [25]. Most classification algorithms can be revised as distance-based classification methods [26]. Instead of a distance measurement per class or cluster, we can have a distinct one for every neighborhood, that is for every small region in the input space. In other words, we can identify locally adjustable distance functions, for instance, with kNN [27].

2.2.2 Decision tree

Decision tree learning is a method for approximating discrete-valued functions, in which a decision tree represents the learned function. Decision tree learning is one of the most common and practical techniques of inductive learning [28]. A decision tree is a procedure, which classifies the categorical data based on different features. In addition to this decision tree, a large quantity of data are also being processed and used in applications for data mining. No domain knowledge or parameter configuration is needed for decision-making trees. Decision trees are, therefore, adequate and suitable for exploratory understanding, and the tree type is intuitive and easily understood to represent the knowledge acquired.

The main task in those systems is to use inductive methods to determine the attribute values of an unknown object according to decision tree rules. Decision trees classify

cases from root node to leaf node throughout. We begin from the root node of the decision tree, test the attribute, and then move down the tree branch to the given set attribute value. It is repeated at the level of the sub-tree [29].

Entropy in the information theory sets out the minimum number of bits required to encode an instance's class code. Also note that the best split is always between adjacent points of different classes [26]. We are therefore trying them, and the purity of the attribute is done best. No such iteration is required in the situation of a discrete attribute. Therefore we calculate the impurity and choose the minimum entropy for all attributes, discrete and numeric, and for numeric attributes for all divided positions. And after that tree building continues recursively and in parallel to all the not purified branches until they are all pure. This is the basis of the classification and regression tree (CART) algorithm [30], regression tree ID3 algorithm [31], and its extension C4.5 [32]. CART algorithm is used in our experiments which is default algorithm used in scikit-learn [33] for decision tree classifier.

CART stood for Breiman in 1984. This is characterized by the fact that they are building binary trees, namely that each internal node has outgoing two edges. The splits are selected according to the criteria of towing, gini, entropy, and the obtained tree is pruned by cost-complexity trim. In case of misclassification, CART may consider costs when inducing the tree. It also allows users to supply prior probability distribution [34]. The ability of CART to produce regression trees is an important feature. Regression trees are trees whose leaves forecast an actual number rather than a class. CART seeks for splits to minimize the square error forecast in the event of regression. The weighted mean for node is the basis of a prediction in each leaf [35].

2.2.3 Support vector machines

There are many feasible linear separators for two-class, separable training data. While some learning techniques like the perceptron algorithm are based on just any linear separator, others are looking for the highest linear separator based on certain criteria like Naive Bayes. The Support Vector Machines (SVM) defines in specific the criterion for a decision surface as far as possible from any data point. The classifier margin is determined by this distance from the choice surface to the closest data point. The decision feature for an SVM is necessarily specified by a (generally small) subset of

the data, which defines the position of the separator. These points are known as the supporting vectors (a point may be considered as a vector between the origin and the point in a vector space). Figure 2.2 illustrates the sample issue margins and support vectors. Additional data points do not contribute to the determination of the decision surface [36].

It seems good to maximize the margin because the points close the decision surface represent very uncertain classification decisions. The classifier will be almost 50% likely to decide any way. A classifier with a big margin does not make choices on classification with a small level of certainty. This provides you an overall safety margin for classification; there is no misclassification due to a minor mistake in measuring.

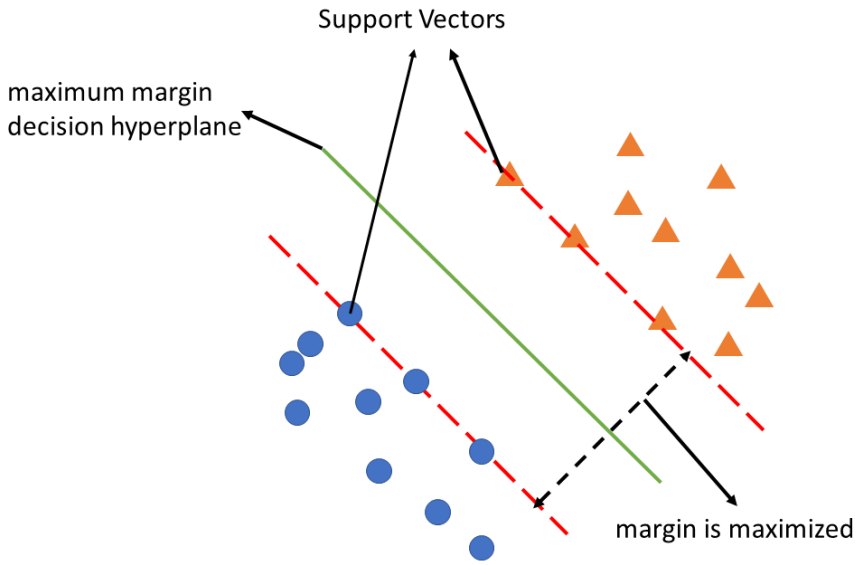

Figure 2.2 : Support Vector Machines for Classification.

For formalize Support Vector Machines; an intercept term d and a hyperplane decision normal vector $\vec{w}$, perpendicular to the hyperplane, can define a decision hyperplane. In the machine learning literature, this vector is commonly called the vector of weight. We specify the intercept term d to choose from all the hyperplanes perpendicular to the normal vector. Since the hyperplane is perpendicular to normal vector, the $\vec{w}^{\mathrm{T}}\vec{x} = -d$ is satisfied in all points $\vec{x}$ on the hyperplane. Now suppose we have a set of data points $\mathbb{D} = \{(\vec{x}_i, y_i)\}$ for each member to be paired with a point $\vec{x}_i$ and a class y_i. For SVM the two data classes are always called $+1$ and -1 (instead of 1 and 0); instead of being folded in to a weight vector, the intercept term is always expressly represented d. The

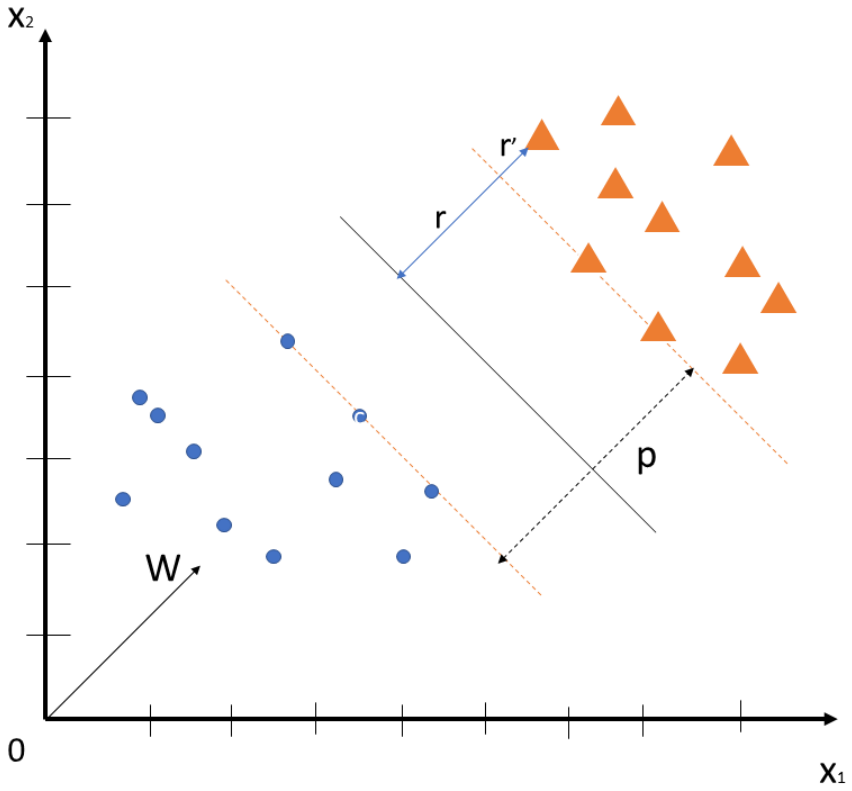

Figure 2.3 : The Margin of One Point and a Boundary of Decision.

linear function will:

$$f(\vec{x}) = \text{sign}\left(\vec{w}^{\text{T}}\vec{x} + d\right) \tag{2.1}$$

We are sure that an item is classified if it is far from the limit of decision. The functional margin of the i^{th} instance $\vec{x}_i$ with regards to a hyperplane $\langle \vec{w}, d \rangle$ is defined as the amount $y_i\left(\vec{w}^1\vec{x}_i + d\right)$ for a certain data set and choice hyperplane. With regard to the decision surface, the functional margin of a data set is double that of one point in the data series with a minimum functional margin (the factor of 2 is measured across the entire range, like in Figure 2.3).

2.2.4 Multilayer perceptron

Multilayer perceptrons (MLPs) are neural feed-forward networks with standard algorithms of backpropagation. They are supervised, so they need to be trained. If it is known how the data is converted to the desired result, it can be widely used to classify the pattern. Using one or two hidden layers, any input-output map is almost predictable. In hard problems, they were shown to estimate the efficiency of ideal statistical classifiers. Most apps in neural networks include MLPs [37].

The units each execute a biased weighted total of their outputs and proceed through a transfer function to generate their output, and the units are structured in a layered feed-forward structure. Therefore, the network has a straightforward definition as an input-output model, with the model's free weights and biases. Such networks can design almost random complexity features, which determine the amount of layers and units in each layer. Important problems in Multilayer Perceptrons (MLP) layout include the number of hidden layers and the number of units in these layers. This multilayer network has various descriptors: MLP, neural feed-forward networks, artificial neural networks, back-prop neural networks [38]. Figure 2.4 shows the general structure of the MLP algorithm. Let us define the input, output, hidden units mentioned on the figure. First, to generalize neural networks by implementing a number of fixed nonlinear transforms ϕ_j to input vector $\vec{x}$. Single output network will be:

$$y(\vec{x},\vec{w}) = g\left(\sum_{j=1}^{M} w_j \phi_j(\vec{x})\right) \tag{2.2}$$

g is nonlinear activation function like sigmoid. If $\phi_j(\vec{x})$ features are fixed and non-adaptive, they can sometimes be linked to as basic functions. Using such functions expands the class of discriminating features available. Our objective is to explore how to adapt these basic functions: and have their own parameters that can be measured instantly from a training data set. First layer includes M linear d dimensional inputs (shows in Figure 2.4):

$$b_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i, \quad j = 1,2,\ldots,M \tag{2.3}$$

As before, $x_0 = 1$, with its weights equal to the biases. b_j quantities are called "activations", and $w_{ji}^{(1)}$ parameters are the weights. The superscript '(1)' shows the network's first layer. Every activation is then converted by nonlinear activation g, normally a sigmoid:

$$z_j = h\left(b_j\right) = \frac{1}{1+\exp\left(-b_j\right)} \tag{2.4}$$

The z_j inputs equal to the base function inputs in Equation (**2.2**). In the scope of neural networks, the quantities z_j are defined as the output of hidden units so-called because they have no problem-specified values or target values used during training. The second layer contains a linear combination of the inputs of the concealed units in

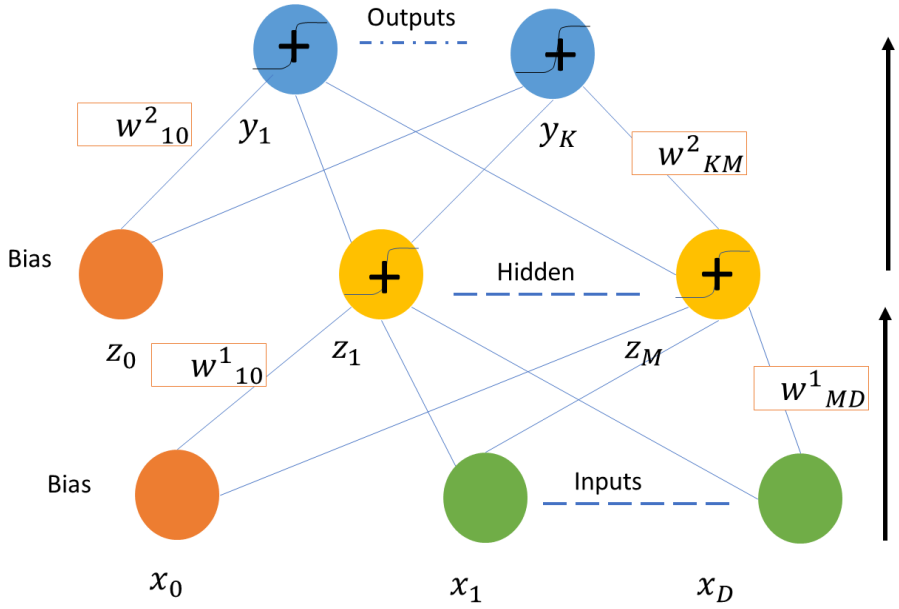

Figure 2.4 : Multi Layer Perceptron (MLP) Network Diagram with Two Weight Layers.

order to activate the output K units:

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \quad k = 1, 2, \ldots, K \tag{2.5}$$

$z_0 = 1$ which matches the bias. The second part of the neural network is the conversion parameterized for weights $w_{kj}^{(2)}$. A sigmoid may be used to transform the output units with an activation function:

$$y_k = g(a_k) = \frac{1}{1 + \exp(-a_k)} \tag{2.6}$$

Or a "softmax activation function" for multiclass problems:

$$g(a_k) = \frac{\exp(a_k)}{\sum_{\ell=1}^{K} \exp(a_\ell)} \tag{2.7}$$

These equations can be combined to describe the general equation describing the forward propagation via the network and how an output vector, given its weight matrices, is calculated from an input vector [39] where $y_k is$:

$$y_k = g\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{2.8}$$

2.2.5 Linear discriminant analysis

Linear Discriminant Analysis is methods of classification and reduction of dimensionality that can be understood from two points of view. The first of these is a probabilistic interpretation, and the second is due to Fisher's more methodical approach. The first knowledge of the LDA assumptions is helpful. The second knowledge of how LDA reduces dimensions enables for stronger comprehension [40]. LDA provides the following properties:

- LDA considers the data to be Gaussian. In particular, it presumes that all classes have the same matrix of covariance. In the dimensional subspace of $K-1$ LDA discovers linear decision boundaries. It is not appropriate, therefore, if the independent variables are interacted in a higher order.
- LDA is suitable for multiclass issues, and should be used with attention when the distribution of classes has been unbalanced, as priors from the observed numbers are calculated. Observations are therefore rarely classified into rarities.
- LDA can also be used as a technique for reduction of dimensionality as with PCA. Note that LDA's transformation is fundamentally distinct from PCA, as LDA is a supervised method considering the results.

2.2.6 Quadratic discriminant analysis

Quadratic Discriminant Analysis is an LDA version in which each class of findings estimates a single covariance matrix. QDA is especially helpful when it is known beforehand (prior) that each class has separate covariances. QDA is disadvantageous because it can not be used as a method for dimensional reduction [40].

2.2.7 Logistic regression classifier

The aim of binary logistic regression is to develop a model,which can decide binarily on the class of a future experiment. Consider a single input event $\vec{x}$, which represented by a vector of features $[x_1, x_2, \ldots, x_n]$ The output of the classifier y can be 1 (i.e., the observation is a class member) or 0 (the observation is not a class member). We'd like to know the probability that $P(y = 1|\vec{x})$ is a class member [41].

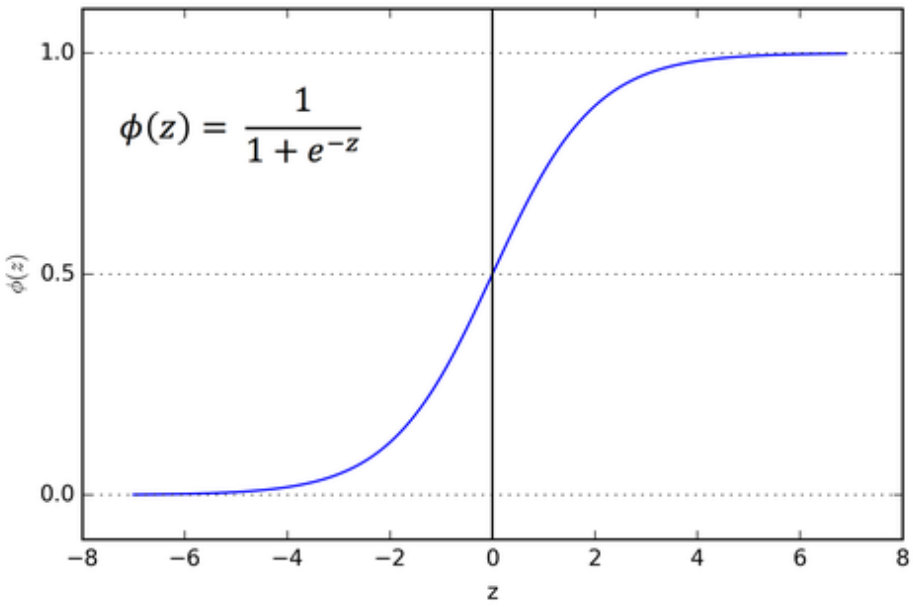

Figure 2.5 : Sigmoid Function [2].

Logistic regression resolves the classification problem by learning a vector of weight and a bias term from a training set. Each w_i is a real number and linked with one of the input features x_i. The weight w_i shows how significant the input feature is to the classification decision, and can be either positive (the feature is correlated with the class) or negative the feature is not correlated with the class). To decide on a test instance - after learning the weights in training the classifier first multiplies each x_i by its weight w_i, sums up the weighted features, and adds the preference term b. The subsequent single digit z represents the class weighted sum of proof.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b \tag{2.9}$$

But note that nothing at Equation (**2.9**) forces z, i.e., to lie between 0 and 1. Since weights are real value the output could be even negative ; z ranges from $-\infty$ to ∞. In fact, the output is negative. We transfer z through sigmoid, $\sigma(z)$, to create a probability. The sigmoid function, also known as the logistic function, gives the logistic regression its name because it looks like an s. The sigmoid has that equation and function shown in Figure 2.5.

$$y = \sigma(z) = \frac{1}{1+e^{-z}} \tag{2.10}$$

If the sigmoid is applied to the weighted function amount, we receive a value between 0 and 1. We want to know parameters (meaning $\vec{w}$ and b) that create $\hat{y}$ as close as appropriate to the real y. This needs two parts. The first is to measure the present

label's proximity to the actual label y. The difference between system output and actual output, and the loss function or cost function is known as this distance loss. The loss function (cross-entropy loss) frequently used for logistic regression and neural networks. Secondly, we need an optimization algorithm to upgrade weights iteratively so that this loss function is minimized. The default algorithm is gradient descent.

2.3 Ensemble Methods

In the last few decades, the computer intelligence and the machine learning community has paid more attention to multiple classifiers systems, also called ensemble systems. This attention has been well-deserved since ensemble systems in a broad range of problem fields, and real-world applications have proved themselves to be very useful and extremely versatile. Initially developed to reduce the variance thereby enhancing the accuracy of auto -made decision-making system, ensemble systems have since succeeded in tackling a range of machine learning issues such as selection of features, reliability estimation, missing features, incremental learning, error correction, class imbalance of data [42].

Data scientists have found the ensemble-based systems rather late and the benefits of these systems for decision-making. Although the results of several decades of extensive studies are now an essential component of our understanding and literature on ensemble systems, ensemble-based decision making has actually taken place and maybe part of our daily life as long as civilized populations exist. We, as humans, use such systems so many times in our everyday life that perhaps it is secondary for us. There are many examples. Indeed, the essence of democracy in which a group of individuals vote to decide whether to elect an elected officer or to decide a new law is based on the decision-making processes of an ensemble. In many nations, the judicial system is also founded on ensemble-based decision making, whether based on a jury of pairs or a jury of judges. Maybe more in practice, when we face an important choice, we often check for the views of different "specialists" to assist us make this choice. Before agreeing on a significant medical procedure, consult with several physicians, read customer reviews before buying a product, call references before employing a potential candidate are all examples of ensemble-based decision making.

Since the primary objective of using ensemble systems is in reality like that of our everyday life with such mechanisms, this is, by weighting different views and combining them through a particular thought process to make a final choice, we can build the trust that we make the right choice. Machine learning applications of ensemble systems are numerous. These include estimating confidence, addressing missing features, incremental sequence data learning, selecting the features, teaching in non-stationary settings, and addressing imbalanced data problems.

2.3.1 Random forest classifier

Random forests [43] constitute a significant change in bagging, which builds up and then averages an extensive collection of de-correlated forests. The performance of random forests is very identical in many cases to boost, and it is easier for training and tuning. Random forests are thus common and deployed in many packages.

The fundamental concept for bagging is to decrease the variance by averaging a number of noisy but nearly unbiased models. Trees are perfect bagging candidates because they are able to track complicated interaction constructions in the data and have comparatively low bias when adequately large. Because the trees are notable for their noise, the average is handy. Furthermore, since every tree produced in the bagging process is distributed identical, the average B expectations of these trees are identical to those of each. This implies that the distance between bagged trees is the same as that of individual trees, and the only chance of enhancement is by reducing variance. In comparison to boosting, trees are grown dramatically to prevent bias, so they are not identically distributed [44].

On average, B identically distributed random values, with σ^2 variance respectively, have $\frac{1}{B}\sigma^2$ variance. Whether the variables are identically distributed with positive pairwise correlation, the variation of the average is (identically distributed):

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{2.11}$$

As B rises, the second term fades, but the first stays, and thereby the magnitude of bagged trees pair correlation limits the advantages of averaging. The concept is to enhance variance reductions in bagging in random forests by decreasing correlations

between trees without too much-increased variance. This is done by randomly selecting the input variables in the tree growing method.

2.3.2 Bagging

Bagging [45] is the bootstrap [46] method for the ensemble which, by training every classifier in a random distribution of the set, creates individuals for the ensemble. Every training set for a classifier is produced by random drawings with substitute N examples where N is the size of the original training set; many of the original examples can be replicated while others can be abandoned in the resulting training set. With a distinct random selection of the training set, each independent classifier in the selection is created [47].

In statistics, the bootstrap is an essential re-sampling instrument. The fundamental concept behind the bootstrap is that, given the training data $\mathbb{D} = \{(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)\}, i = 1, \ldots n$, we can assess actual $\mathscr{F}$ of so-called empirical distribution $\hat{\mathscr{F}}$. The feature empirically is just [48]:

$$P_{\hat{F}}\{(X,Y) = (\vec{x},y)\} = \begin{cases} \frac{1}{n} & \text{if } (\vec{x},y) = (\vec{x_i},y_i) \text{ for some } i \\ 0 & \text{otherwise} \end{cases} \quad \textbf{(2.12)}$$

This is a discrete distribution of probabilities, which gives equivalent importance $(1/n)$ to every learning point observed. A test bootstrap m from the data is $(\vec{x_i}^*, y_i^*), i = 1, \ldots m$, in which each $(\vec{x_i}^*, y_i^*)$ is evenly taken from $(\vec{x_1}, y_1), \ldots (\vec{x_n}, y_n)$ with substitute.

Data from a training $(\vec{x_i}, y_i), i = 1, \ldots n$, bagging compares the predictions of a bootstrap sample of classification trees. In the case of $b = 1, \ldots B$ we take n samples of boostraps $\left(\vec{x_i}^{*b}, y_i^{*b}\right), i = 1, \ldots n$ specimens.And on this sampled training data set we fit a $\hat{f}^{\text{tree},b}$ ranking tree. Lastly we simply bring the most predicted class to rank the entry $\vec{x} \in \mathbb{R}^p$:

$$\hat{f}^{\text{bag}}(x) = \underset{k=1,\ldots K}{\text{argmax}} \sum_{b=1}^{B} 1\left\{\hat{f}^{\text{tree},b}(x) = k\right\} \quad \textbf{(2.13)}$$

Breiman [45] demonstrated the effectiveness of the bagging of "unstable" learning algorithms where tiny modifications of the training set lead in significant predictive modifications. Breiman stated to be cases of unstable learning algorithms in neural nets and decision trees.

2.3.3 Extremely randomized trees

In such an Extremely Randomized Trees ensemble, every tree is built by considering a random number of split applicants at each node. More specifically, a random group of features is chosen, and a random splitting point is selected for each feature. The applicant with the highest value for the criterion split is selected from these applicants. ERT has shown that the more populous Random Forests have a stronger predictive performance.

Any hierarchical clustering technique could produce a similar encoding at this stage. But, the use of ERTs proves useful. Firstly, ERT is a technique of tree ensemble and is thus resistant to tiny data disturbances. Due to the automatic choice system, it is also resistant to non-informative or noisy features. The produced illustration of the feature is thus regarded as less noise version [49].

Furthermore, a further benefit is that, without preprocessing, the tree assemblies can handle numerical and nonnumerical numbers, making the technique more straightforward and stable. Furthermore, it provides a natural manner to address missing values, as opposed to several other approaches, by spreading cases with a missing split value across all branches or by choosing a branch to follow by randomly. Another benefit of the approach proposed is that this is parameters-free and inductive. The model can withstand new data without the need for a training set after training. This makes it possible to use concurrent online systems and systems that process large-scale data.

The ERT algorithm constructs an array of unpruned decision trees or regression trees by the standard top-down operation. His two primary distinctions with other tree-based ensemble techniques is that he splits nodes by selecting a random cut-point, and utilizes the entire training sample (rather than a copy of a bootstrap) to develop the trees.

Algorithm lists the operation for splitting ERT for numeric attributes. It has two parameters: a minimum sample size for splits, K, the number of features chosen at random at each node; and a minimum $n_{\min}$. It is used multiple times to produce an ensemble (we indicate by B the amount of forests of this ensemble) through the (complete) initial training sample. The trees' predictions are summed up to produce the final forecast by majority vote in problems of ranking and arithmetical average in

problems of regression. From the standard of the bias variant point of perspective, the rationale behind the Extremely Randomized Trees technique is to decrease variance more highly than the weaker randomization systems used for the other techniques by specific randomization of the point and the attribute together with average set. To decreasing biases, the use of the entire initial training sample is supported instead of bootstrap replicas [50].

K, $n_{\min}$ and B parameters have various impacts: K determines the power of the choice of the features, $n_{\min}$ the power of the average noise yield and $n_{\min}$ the force of the bias reduction of the aggregation of the ensemble model. These parameters may be adjusted manually or automatically (e.g., through cross-validation) for particular problems. However, we tend to use default configurations to maximize the computing benefits and method independence.

2.3.4 AdaBoost

Boosting originates from Kearns and Valiant's established theoretical framework for the study of machine learning called "PAC" ("Probably Approximately Correct") [51]. They were the first to question whether a "weak" learning algorithm, mildly stronger than random guessing, could be a building block for a generally precise "strong" learning algorithm.

AdaBoost is a machine learning algorithm first developed by Freund and Schapire [52]. It is adaptive in the way that classifiers coming in next for execution are adjusted according to those cases that were incorrectly classified by prior classifiers. It is susceptible to non-required set noisy data and information. However, in some circumstances, this algorithm may be less sensitive to fixed memory input compared to many other algorithms. AdaBoost constantly calls the weak classifiers, performing a sequence of $k = 1,\ldots,K$ classifiers. In each execution, "weight" calculated by incorrectly classified instances increases (or, alternately, weights of each implementation). In each execution, "weight" computed by incorrectly categorized instances rises (or weights decreases from each properly classified example). New classifiers will only focus on the instances incorrectly classified by prior classifiers. This meta-algorithm can be used together to enhance efficiency by using a variety of other algorithms [53].

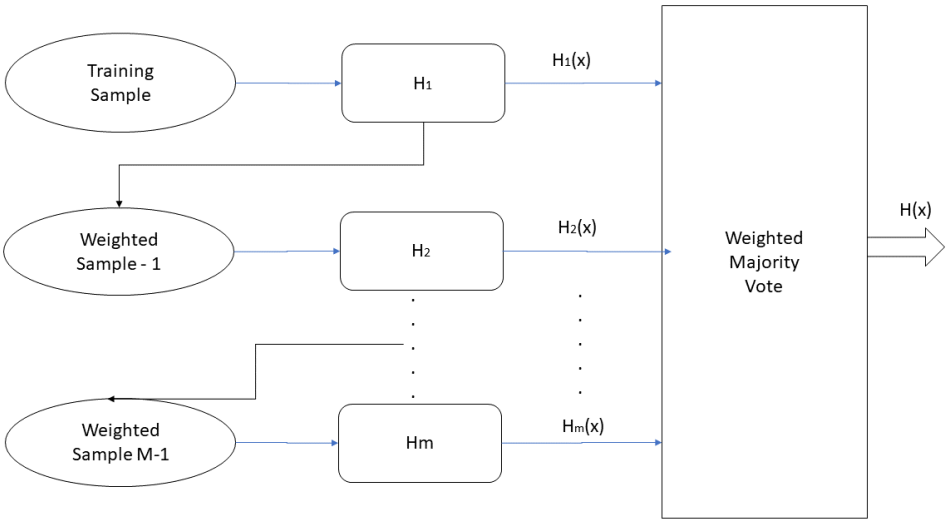

Figure 2.6 : AdaBoost Algorithm.

In our setting, we get N training points $(\vec{x}_i, y_i)$ where$\vec{x}_i \in \vec{X}$ and $y_i \in \{-1, +1\}$. On round m, where $m = 1, 2 \ldots, M$, we fit a weak classifier $H_m(\vec{x})$ to a data set variant reweighted by some vector $\mathbf{W}_m$. We then calculate our selected learner's weighted misclassification rate and update our next round weighting metric $\mathbf{w}_{m+1}$. In reality, one sometimes limits the number of boosting round as a form of regularization. The last classifier is the weighted linear combination of classifiers generated at each stage of the algorithm [54].

AdaBoost has many benefits in practice. It is quick, simple, and programmable. AdaBoost has no parameters to be individually adapted (except steps number K). In contrast, real boosting performance in a specific issue depends mainly on data and a weak learning algorithm. Boosting can lead in accordance with the theory to incorrect outcomes if the data for training are insufficient, the weak hypothesis is very costly, or the weak hypothesis is too weak.

2.3.5 Gradient boosting

A gradient-based formula of boosting techniques was derived from linking up the statistical structure [52, 55, 56]. The gradient boosters were referred to in this formulation of boosting techniques and the matching model. The fundamental justification of the model hyperparameters was given by this framework as well as the methodological foundation for further improvement of the model.

The training process adapts consecutively to new models for the accurate estimation of the response variable in gradient boosting systems or simply in GBMs. The main concept behind this algorithm is for the new base learners to be linked as closely as the whole ensemble to the adverse gradient of the loss function. The loss features may be random, but the training method will lead to successive error-fitting if an error function is the standard squared-error loss. In particular, researchers are responsible for choosing the loss function, with so far a wide range of loss functions and the ability to implement a custom-made loss [57].

This strong flexibility makes the GBMs extremely adaptable to every task driven by data. It gives a ton of liberty to model design and makes it a problem of test and error to choose the most suitable loss function. Although it is comparatively easy to introduce boosting algorithms that enable you to experiment with distinct model design in the practical apps, GBMs were successful in numerous problems related to machine learning and data mining [58, 59].

2.4 Decision Fusion Algorithms

The aim of ensemble methods/decision fusion is to combine different classifiers into one meta-classifier, which performs better than every single classifier. For example, if we have collected predictions from 5 oracles, ensemble/fusion approaches will permit us to combine the five oracles strategically with a prediction that is more accurate and robust than each oracle's predictions [60].

The strategy of combining classification systems is one of the key components of an ensemble system. In our study, we call that classifier combination as decision fusion. Combination rules are often divided into two groups: (1) trainable vs. non-trainable (2) rules apply to class labels to class-specific continuous outputs [42]. The following is discussed, the weighted majority voting, as the parameters are instantly accessible when classifying, is an example of such untrained rules. However, ADF algorithm classifiers weights change in training and test phase.

2.4.1 Majority voting

There are three versions of majority voting, where the ensemble choose the class:

- Class on which all classifiers agree (unanimous voting).

- Class predicted by at least one more than half the number of classifiers (simple majority).
- Class that receives the highest number of votes, whether or not the sum of those votes exceeds 50% (plurality voting or just majority voting).

We are assuming that from classifier outputs, only class labels are available. Let's define decision of the m^{th} classifier as $C_{m,k} \in \{0,1\}$, $m = 1,2,\ldots,M$ and $k = 1,2,\ldots,K$, where M is the number of classifier and K is the number of classes. If m^{th} classifier choose class ω_k, then $C_{m,k} = 1$, and 0, otherwise. The decision of the ensemble for plurality voting could be defined as follows; choose class ω_k, if

$$\sum_{m=1}^{M} C_{m,K} = \max_{k=1}^{K} \sum_{m=1}^{M} C_{m,k} \tag{2.14}$$

It is easy to show that majority voting is an ideal combination rule according to minor assumptions:

- We have an odd number of classifications for a two-class issue.
- For every example $\vec{x}$, p is the probability for every classifier selecting the right class.
- The classifiers' classes are independent.

Then plurality and majority vote are identical, and an ensemble makes the right decision when the right label is chosen at least by $M/2+1$, and the floor function returns the largest integer less or equal to its argument [42].

2.4.2 Weighted average based decision fusion

If we have proof that some specialists are more skilled than others, it may enhance the general efficiency by weighing up the choices of those skilled professionals more than plurality votes can achieve it. Let' denote the decision of hypothesis h_m on class ω_k as $C_{m,k}$, such that $C_{m,k}$ is 1, if h_t selects ω_k and 0, otherwise. Also, suppose that we have a way to estimate each classifier's future performance, and assign weight w_m to classifier h_m in proportion to its estimated performance. According to this note, classes whose decisions are combined by weighted majority vote will be selected K, if:

$$\sum_{m=1}^{M} w_m C_{m,K} = \max_{k=1}^{K} \sum_{m=1}^{M} w_m C_{m,k} \tag{2.15}$$

This means, when the total weighted vote that ω_k receives is greater than any other class's total vote. In order to simplify interpretation, the weights can be standardized so that the total 1; however, the result of weighted majority voting does not change standardization.

The predicted class probabilities could be advantageous if the classifiers in our fusion are good-adjusted instead of the class labels for majority voting [60]. The modified weighted majority voting for class labels can be written as follows:

$$\sum_{m=1}^{M} w_m C_{m,K} = \max_{k=1}^{K} \sum_{m=1}^{M} w_m p_{m,k} C_{m,k} \tag{2.16}$$

Here the $p_{m,k}$ is the m^{th} classifier for class k is a predicted probability. Note that, the classifier's output must give the class probabilities.

2.4.3 Adaptive decision fusion algorithm

The online Adaptive Decision Fusion algorithm [20] was developed based on the POCS [61] algorithm. This algorithm is an online adaptive way to update each sample according to the amount of error that aims to purify the performance of the decision. Toreyin and et. al. developed the POCS algorithm to reveal the EADF and ADF algorithms. The EADF algorithm is an entropy-based online adaptive decision fusion algorithm. These algorithms were used to increase the performance of smoke detection in forest fires and achieved higher performance than similar algorithms. He used it for anomaly in his study. We will develop this algorithm for multi-class multi-classifier [20].

Assume that the combined algorithm consists of M multiple classifiers: $C_1, C_2 \ldots, C_M$ $C_i(\vec{x}, n)$ is the decision value for $\vec{x}$ input in any n step. $C_i(\vec{x}, n)$ shows which class has been selected for i^{th} classifier. For intrusion detection case, that is the type of attack. The $\vec{x}$ input varies according to the algorithm and the field to be used. For our field that input will be a traffic flow that occurs as a result of an action on the network. Four sub classifiers ($M = 4$) used to determine the attack interference in Section 4. The type of attack determined by each classifier is expressed in $C_i(\vec{x}, n)$ of one of the classifiers.

The decision vector is $\vec{C}(x,n) = [C_1(\vec{x}, n), C_2(\vec{x}, n) \ldots, C_M(\vec{x}, n)]^T$, and the elements represent class selection in step n for the $\vec{x}$ input. Classifier weights $\vec{w}(x,n) = [w_1(\vec{x}, n), w_2(\vec{x}, n), \ldots, w_M(\vec{x}, n)]^T$ shows weights at n step. For clarity, we remove x

from $\vec{w}(x,n)$. With this decision fusion, the result of class will be:

$$\hat{y}(\vec{x},n) = \vec{C}^{\mathbf{T}}(\vec{x},n)\vec{w}(n) = \sum_i w_i(n)C_i(\vec{x},n) \tag{2.17}$$

We can state that $y(\vec{x},n)$ as true or correct result. Error for $\vec{x}$ input in n step can be expressed as $e(\vec{x},n) = y(\vec{x},n) - \hat{y}(\vec{x},n)$. The advantage of this method compared to other methods is that weights can be updated with feedback based error. The weights of the sub-algorithms that make the wrong decisions are reduced in every step. Another advantage of this method is that it does not make any assumptions about how data is distributed. The distribution of data does not affect the result [20].

For finding weights based on projections we will first review the weight update scheme based on orthogonal projection. In ideal case, weighted sub-algorithm decision values/classes should be equal to the decision value of the oracle/real value:

$$y(\vec{x},n) = \vec{C}^{T}(\vec{x},n)\vec{w} \tag{2.18}$$

That represents a M-dimensional space hyper-plane, $\mathbb{R}^M$. Hyper-planes in $\mathbb{R}^M$ are closed and convex. $\vec{C}^{T}(\vec{x},n)\vec{w}(n)$ may not be equal to $y(\vec{x},n)$ at instant n. In Toreyin approach, projection of the current weight vector $\vec{w}(n)$ onto the hyperplane represented by Equation **2.18** determines the next set of weights. The orthogonal weight vector $\vec{w}(n+1)$ projection onto the $y(\vec{x},n) = \vec{C}^{T}(\vec{x},n)\vec{w}$ hyperplane is the closest vector to the $\vec{w}(n)$ vector on the hyperplane. Let's formulate the problem as a problem of minimization:

$$\min_{\vec{w}^*} \|\vec{w}^* - \vec{w}(n)\|_2 \text{ subject to } y(\vec{x},n) = \vec{C}^{T}(\vec{x},n)\vec{w}^* \tag{2.19}$$

Using Lagrange multipliers, the solution can be obtained. The solution is called the mapping solution for metric projection. However, we use the term orthogonal projection because it is orthogonal to the hyperplane that the line passes through $\vec{W}^*$ and $\vec{w}(n)$. If the next set of weights is identified as $\vec{w}(n+1) = \mathbf{w}^*$, the following iteration can be acquired:

$$\vec{w}(n+1) = \vec{w}(n) + \frac{e(\vec{x},n)}{\|\vec{C}(\vec{x},n)\|_2^2}\vec{C}(\vec{x},n) \tag{2.20}$$

Therefore the vector of the projection is calculated by Equation **2.20**. Note the Equation **2.20** is the same as the normalized least mean square (NLMS) algorithm

with $\mu = 1$ update parameter. Convergence should be satisfied in the NLMS algorithm $0 < \mu < 2$ [62]. According to the theory of projection on convex sets (POCS), when there is a finite number of convex sets, repeated cyclic projections converge on these sets to a vector in the intersection set [20] there are an innumerable number of convex modules. They propose the use of an online adaptive algorithm convex combination of projections in the latest q sets. The next section provides the algorithm's block projection version that covers the case when an infinite number of convex sets exist. Another hyperplane based on the new decision values $\vec{C}(\vec{x},n)$ of sub-algorithms is defined in $\mathbb{R}^M$ when a new data entry comes.

$$y(\vec{x},n+1) = \vec{C}^T(\vec{x},n+1)\vec{w}^* \tag{2.21}$$

This hyperplane is not typically the same as the hyperplane $y(\vec{x},n) = \vec{C}^T(\vec{x},n)\vec{w}$. The following set of weights, $w(n+2)$, is determined with the $w(n+1)$ projected onto Equation **2.21**. Where a limited number of hyperplanes exist, iterate weights that converge on the hyperplanes by cyclic projections. Figure 2.7 summarizes the pseudo-code of the orthogonal projections into the algorithm based on the hyperplane [20].

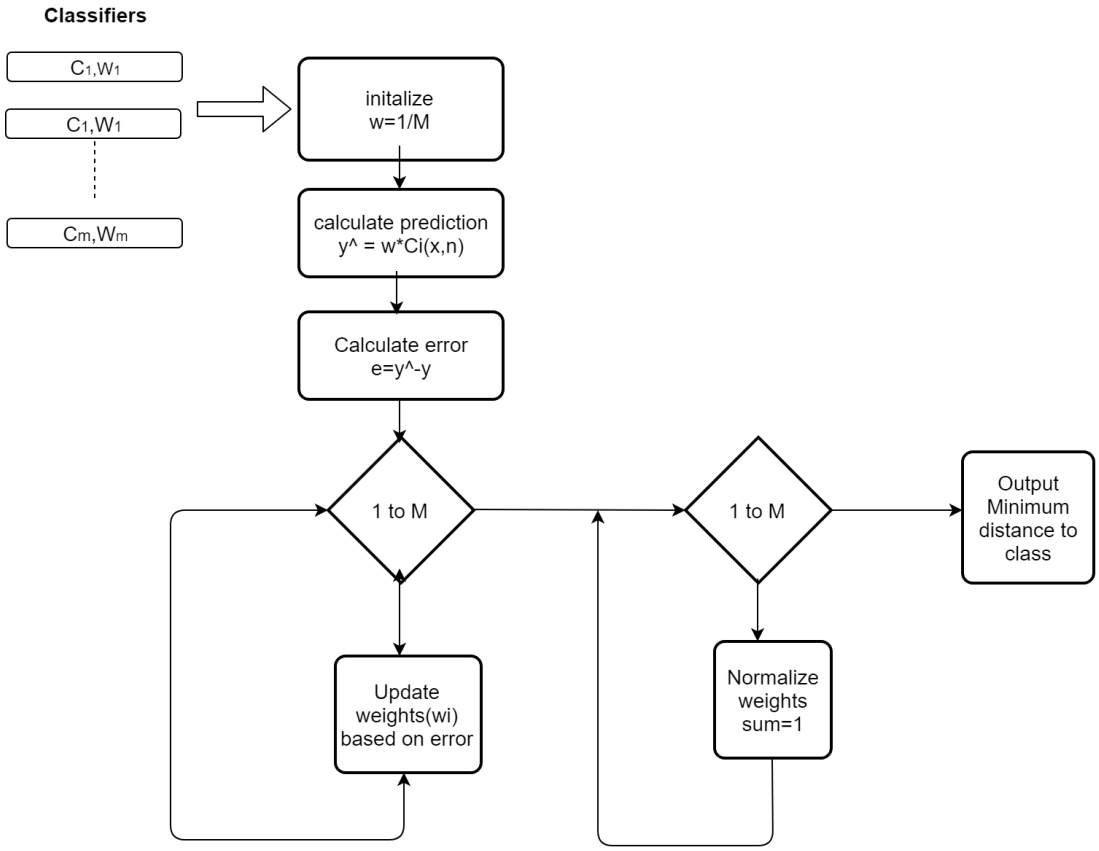

Figure 2.7 : Online Adaptive Decision Fusion Algorithm Diagram.

In our problem we will need to develop the algorithm specified in proposed method, since multi-class classifier will be used. Figure 2.7 shows the flow diagram of the ADF algorithm. The current algorithm is a two-class algorithm, and the output is calculated by giving if $\hat{y}(\vec{x},n) \geq 0$ 1 if not -1. In our approach, the algorithm output must be the closest class to $\hat{y}(\vec{x},n)$. Assuming that the classifier $C_i(\vec{x},n)$ have K classes and it must be a value of $y(\vec{x},n) = \{\omega_1, \omega_2, \ldots, \omega_K\}$ and every $C_i(\vec{x},n)$ have $\omega_1, \omega_2, \ldots, \omega_K$ classes to find the minimum distance and predicted class for $C_i(\vec{x},n)$ is ω_i. For finding predicted class by decision fusion:

$$y(\vec{x},n) = \arg\min(\mid C_1(\vec{x},n) - \omega_i \mid, \mid C_2(\vec{x},n) - \omega_i \mid \ldots, \mid C_M(\vec{x},n) - \omega_i \mid) \qquad \textbf{(2.22)}$$

3. PROPOSED METHOD

3.1 Proposed Method

Firstly CICIDS2017 data set was made available for model training with data preprocessing techniques which referred in section 3.3. At this step, deleting the wrong and missing data, removing unnecessary columns, correcting the wrong data, grouping categorical data, and filling missing values operations were performed. The detailed procedures are specified in the relevant section.The general flow diagrams of the proposed method are shown in Figures 3.1 and 1.1.

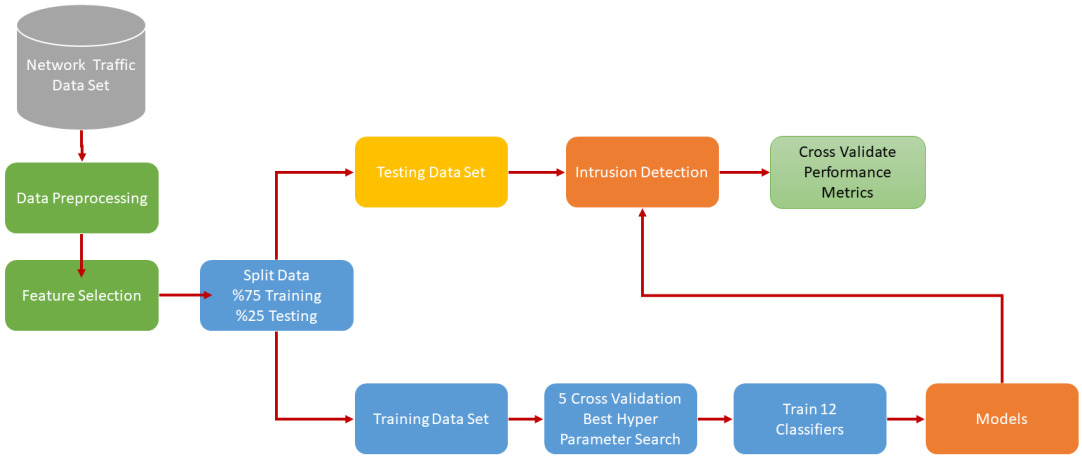

Figure 3.1 : Flow Chart of the Proposed Method.

3.1.1 Step 1: anomaly detection

For the first step anomaly detection, all attack types were classified in a single class and re-labeled as abnormal (-1). Normal/benign data is labeled as (1). Table 3.1 shows regrouped data set. This data set was divided into 25 % test and 75% training sets while maintaining class weights. The test set was in no way used for training, validation, and finding the best hyperparameter. The results indicated are the studies on the test set. Step 1 is divided into two sub-steps. In the first step, various feature selection methods

Table 3.1 : CICIDS2017 Data Set Reclassified for Normal/Attack Classes.

Labels	Number of Instances	% of in Total Instances.
Normal	2271320	80.319
Attack	556556	19.681
Sum	2827876	100.000

performed on the data set. Moreover, the best features were selected on F1-score as the performance metrics. In the second sub-step, experiments were performed out with different machine learning algorithms for detecting anomaly/attacks.

For the first step, 12 supervised learning classifiers were tested. Firstly, a list of possible hyperparameters was created for each classifier, and the best hyperparameters were determined from the scikit-learn library using randomizedsearchcv. Cross-validation was performed using $cv = 5$. That is, 15% of the data set was used as validation/development set. Since class weights were different (imbalanced), the best performance was determined according to F1-score. HPC (High-Performance Computing) was used because the data set is large (2.8 million rows, up to 70 features) and the parameter set is large. "dask" and "ipyparallel" "joblib" libraries were used for parallel programming. After finding the best hyperparameters for each classifier, the models were trained and performed on the test data set. The above operations were performed for each classifier at step 1 and step 2. As a result of step 1, the classifier that provides the best performance was selected and used for anomaly detection. This classifier was determined whether there is intrusion/attack on the network. The reason for using only one classifier in this step is to achieve high performance with a single classifier already and to prevent possible delays in using multiple classifiers in heavy network traffic.

3.1.2 Step 2: attack type detection

In the second step, we aim to determine the type of attack/intrusion. For this purpose, only the intrusion/attack data was used. Normal/benign traffic is continued without entering this step. Abnormal traffic was getting into this step. For this purpose, only the intrusion/attack data was taken from the data set. The class labels are the labels specified in the data set description. Table 3.2 shows the classes, number of instances, and class weights. In this step, 12 classifiers were used for the test process. Data sets were distributed in 25% test and 75% training considering class weights — the best

Table 3.2 : CICIDS2017 Data Set Classes for Step-2.

Labels	Number of Instances	% of in Total Rec.
DoS Hulk	230124	41.348
PortScan	158804	28.533
DDoS	128025	23.003
DoS GoldenEye	10293	1.849
FTP-Patator	7935	1.426
SSH-Patator	5897	1.060
DoS slowloris	5796	1.041
DoS Slowhttptest	5499	0.988
Bot	1956	0.351
Web Attack - Brute Force	1507	0.271
Web Attack - XSS	652	0.117
Infiltration	36	0.006
Web Attack - Sql Injection	21	0.004
Heartbleed	11	0.002
Sum	556556	100.000

Table 3.3 : CICIDS2017 Data Set Attack Classes Reclassified for Step-2.

New Labels	Old Labels	Number of Instances	% of in Total Rec.
Normal	Benign	2271320	80.319
Botnet	Bot	1956	0.069
BruteForce	FTP-Patator, SSH-Patator	13832	0.489
DosDDos	DDoS, DoSGoldenEye, DoS Hulk, DoS Slow- httptest, DoS slowloris, Heartbleed	379748	13.429
Infiltration	Infiltration	36	0.001
PortScan	PortScan	158804	5.616
Web Attack	Web Attack – Brute Force, Web Attack – Sql Injection, Web Attack – XSS	2180	0.077

hyperparameter determination, cross-validation operations also performed in this step. The output of this step is to determine the type of attack for each classifier. Besides, this data set is imbalanced and contains very few examples of some classes/attacks. Also, some types of attack have similar characteristics. Therefore, as shown in Table 3.3, some classes/attacks are reclassified. The purpose of this operation, achieving a higher attack detection rate. In this step, four classifiers giving the best F1-score is selected for step 3 to be used in decision fusion.

3.1.3 Step 3: decision fusion

In this step, the decision fusion process was applied at the output of step-2 selected classifiers. In this step, three different methods were used for comparison. In the first method, the result decided by the majority voting that the class with the most votes from four classifiers. In the second method, the class with the highest weighted average probabilities is selected. The Adaptive Decision Fusion method, which update classifiers weights dynamically, was used as the last method.

3.2 Performance Metrics

Our model performance evaluation is based on the following measurements. Confusion Matrix is shown in Figure 3.2 and definition of them:

- TP: True Positive (Correct Detection) The attack/anomaly data classified as attack/anomaly
- FP: False Positive (Type-1 Error) The benign/normal data classified as attack/anomaly.
- FN: False Negative (Type-2 Error) The attack/anomaly data classified as benign/normal.
- TN: True Negative (Correct Rejection) The benign/normal data classified as benign/normal.

Accuracy is a metric that measures the overall detection and false alerts percentage of IDS model that represents and is calculated according to the overall success rate of any IDS.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

The detection rate (DR) is the proportion of properly classified malicious instances, which are also called the true positive rate (TPR) or sensitivity.

$$DetectionRate(DR) = \frac{TP}{TP + FN} \tag{3.2}$$

The True Negative Rate (TNR), also called the specificity, is the ratio of normal instances to the total number of normal instances properly classified and shall be

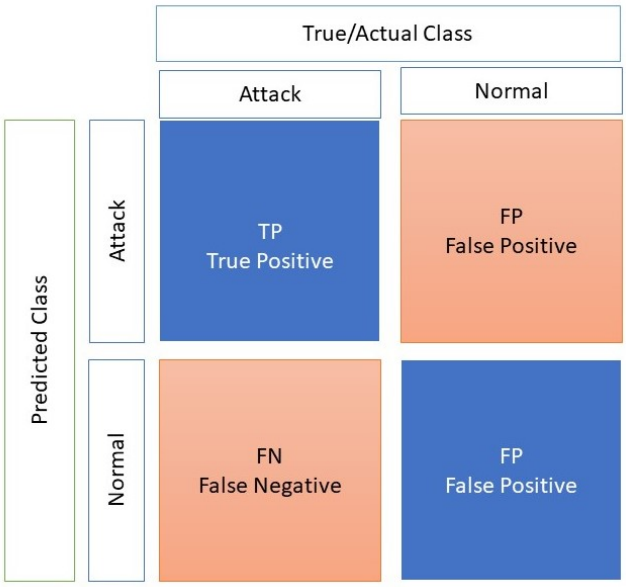

Figure 3.2 : Confusion Matrix.

calculated as:

$$TNR = \frac{TN}{TN + FP} \tag{3.3}$$

The False Positive Rate (FPR) is the proportion of normal instances that are classified as attacks in the total number of normal instances and is calculated in the following:

$$FPR = \frac{FP}{FP + TN} \tag{3.4}$$

The False Negative Rate (FNR) is a ratio of the total number of attack instances classified as:

$$FNR = \frac{FN}{FN + TP} \tag{3.5}$$

In order to estimate to what extent they are acert in the detection of malicious activities, IDS approaches are assessed using TPR-FPR or specificity-sensitivity measure. A perfect IDS approach could have 100% DR, while 0% FPR reflects the absence of any error on all attack instances. This is however very difficult and shows how best to achieve performance in a real environment [10].

Ultimately a preferred measure for assessing IDS approaches is the F-measure criterion. It is a harmonic reminder and accurate mean, i.e., a statistical role for the accuracy assessment of a system through its accuracy and recall given as:

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.6}$$

where precise are a fraction of the actual positive values which, as given in the equations **3.7** and **3.8** respectively, recall the actual number of positive values that

are detected correctly.

$$Precision = \frac{TP}{TP+FP} \tag{3.7}$$

$$Recall = \frac{TP}{TP+FN} \tag{3.8}$$

Similar to the True Positive Rate- False Positive Rate measure, when the precision and recall of an IDS approach to 100%, as the F-measure is the maximum, a 0% FAR and 100% DR are produced.

3.3 Data Preprocessing

The CICIDS2017 data set is a new generation data set that is well prepared and presented. However, this data set contains missing or incorrect values. There are also some problems was occurred when converting pcap files to csv files. Various data preprocessing has been carried out for training and testing operations.

1. Remove socket information: The IP and Port numbers of the source and target hosts are included in the network as an original data set. In order to provide unbiased identification, it is essential to delete such data if such data can result in exaggerated practice for this socket data. However, it is essential that the classifier learns about the features of the packet itself so that a host with comparable packet data is filtered out regardless of its socket details.

2. Remove White spaces: The data set contains some class labels which contain white space. Such white spaces result in different classes, since the actual value is different from other labels in the same class of tuples.

3. Remove bad characters: The character "–" (Unicode Code 8211) used to identify subtypes of web attacks should be replaced by the "- " (Unicode Code 45) since utf-8, the default codec in the pandas library, doesn't recognize this by the label feature.

4. Fill "Flow Bytes/s" feature n.a values with class mean: Flow Byte/s feature have n.a (not available) values. NA values of this feature are filled with the average value of each class.

5. NaN and infinity values: There are 2,867 infinite, NaN value records in the data set. The label of all of these records is BENIGN, which is normal. Therefore, these records have been deleted.

6. Data Normalization: Numerical values in the data set exist in very different value ranges. Too much of these intervals may cause machine learning algorithms not to work correctly and do not work well. Data normalization has been performed to make machine learning algorithms work better, and even some of them can work. Standard scaler is used for this. This data ensures that the average of each feature (each column) is 0, and the standard deviation is 1. Note that same scaler was used for training and testing data set.

3.4 Feature Selection

Feature Selection is a crucial component in the evaluation of workflow data and machine-based learning models. When elevated dimensionality of the data is presented, models generally shock because with the amount of features the training duration rises exponentially. The risk of overfitting with more features is higher than less features. Feature selection techniques help with these issues by decreasing the size of information without much loss. It helps to understand the features and their significance. In this research, we use ANOVA F-score based univariate feature selection. F-test is a statistical test for comparing models and for examining whether the difference between models is noticeable. F-test is helpful when choosing features as we become aware of the importance of each element in model improvement. How ANOVA F-score is calculated is shown in equations **3.10** and **3.11**.

$$F = \frac{\text{between-group variability}}{\text{within-group variability}} \tag{3.9}$$

$$between-group-variability = \sum_{i=1}^{K} n_i \left(\overline{Y}_{i.} - \overline{Y}\right)^2 / (K-1) \tag{3.10}$$

$$within-group-variability = \sum_{i=1}^{K} \sum_{j=1}^{n_i} \left(Y_{ij} - \overline{Y}_i\right)^2 / (N-K) \tag{3.11}$$

Apart from this, the best features can be selected after the model is created with "coef" and "featureimportances" of liner models that are penalized with $L1$ error or classifiers using decision trees. For decision tree-based features importance random forest classifier is used when importance weights of features are calculated. This

algorithm creates a decision-forest. In this decision forest, each feature is given a weight of importance as to how useful they are in the construction of the decision-tree. When the process is finished, these importance weights of features are compared and sorted [63]. The sum of the importance weights of all the properties gives the total importance weight of the decision tree. The comparison of the score of any feature to the score of the whole tree gives information about the importance of that feature in the decision tree.

4. DATA SET

4.1 CICIDS2017 Data Set

In late 2017 the CICIDS2017 data set [64] was released by the Canadian Institute for Cybersecurity (CIC), containing benign and commonly occurring up-to-date attacks. The B-Profiled system, which is used for the abstract behavior and generation of benign natural background traffic in human interactions and CICFlowmeter used [65], for extracting features from the created data set. The most common attack on the 2016 McAfee report (Dos, DDoS, Web Attacks (BruteForce, SQL, XSS), Heart Bleed, Infiltration and PortScan) contains more than 80 characteristics derived from the network traffic generated.

The data set is used to create the complete network topology that includes various operating systems (Windows Server, Ubuntu, Windows 8.1/10, MAC OSX) and network devices (Modem, Switch, Firewall, Router). The distribution and number of record per attack in the CICIDS2017 data set shows in Table 4.1. It includes normal traffic (BENIGN) which representing 80% of the size of the data set (2,273,097 records), and 14 types of attacks (557,646 records) representing 20% of data set. The data set offers a range of recent and common attacks which are helpful for making an efficient and realistic evaluation [66]. CICIDS2017 data set contains 2,830,743 rows separated on 8 files. Each record has 79 features. Each record of CICIDS2017 is labeled as Benign or attack types (14 attack types). Table 4.1 shows the percentage of different attack type and Benign [9]. The CICIDS2017 data set includes benign/normal activity data, and the most-recent recent attacks commonly encountered data that resemble a real network environment. Table 4.2 shows the date of creation of this traffic and what type of attacks were carried out. This data set also includes network analysis results by using the CICFlowMeter tool, with labeled traffic flows based on timestamps, source IP's, target IP's, source/target ports, protocols and attacks (CSV files). Also, Sharafaldin [64] aims to build realistic background traffic. In order to profile the simulated behavior of human interacting and generate naturalistic

Table 4.1 : CICIDS2017 Data Set Classes and Number of Records.

No	Category	Number of Records	% of in Total Rec.
1	BENIGN	2273097	80,3004
2	DoS Hulk	231073	8,1630
3	DoS GoldenEye	10293	0,3636
4	DoS slowloris	5796	0,2048
5	DoS Slowhttptest	5499	0,1943
6	DDoS	128027	4,5227
7	PortScan	158930	5,6144
8	FTP-Patator	7938	0,2804
9	SSH-Patator	5897	0,2083
10	Botnet	1966	0,0695
11	Infiltration	36	0,0013
12	Heartbleed	11	0,0004
13	Web Attack Brute Force	1507	0,0532
14	Web Attack XSS	652	0,0230
15	Web Attack Sql Injection	21	0,0007

good background traffic, proposed B-profile system [5] has been used. For data set generation, 25 users abstract behaviors based on HTTP, HTTPS, POP3, SSH was built. In [67] shows the list of features of the data set.

This data set consists of a 5 day (3rd July - 7th July 2017) data stream on a network created by computers using conventional operating systems such as Windows Vista / 7 / 8.1 / 10, Mac, Ubuntu 12/16 and Kali. CICIDS2017 claims that it has to satisfy the required data set for valid IDS data set.

- Anonymity,
- Attack Diversity,
- Complete Capture,
- Complete Interaction,
- Complete Network Environment (Network must include Firewall, Router, Servers, Different O.S Clients, Modem, Switch etc.)
- Available Protocols (HTTP, HTTPS, POP3, SSL etc.)
- Complete Traffic,
- Feature Set,

Table 4.2 : CICIDS2017 Data Set Capture Schedule and Attack Types.

Flow Rec.Day (WH)	Duration	CSV Size	Attack Name	Flow Count
Monday	All Day	257 MB	No Attack	529918
Tuesday	All Day	166 MB	FTP-Patator, SSH-Patator	445909
Wednesday	All Day	272 MB	DoS-Hulk, DoS-GoldenEye, DoS-slowloris, DoS-Slowhttptest, Heartbleed	692703
Thursday	Morning	87.7 MB	Web-Attacks (BruteForce, XSS, Sql-Injection)	170366
Thursday	Afternoon	103 MB	Infiltration	288602
Friday	Morning	71.8 MB	Bot	192033
Friday	Afternoon	92.7 MB	DDoS	225745
Friday	Afternoon	97.1 MB	PortScan	286467

- Metadata,
- Heterogeneity,
- Labelling

Description of classes:

- **BENIGN**: Normal Activity
- **DoS Hulk**: Generate high volume network traffic to a web server. By bypassing caching engines usage of server resources hit the max and server could not serve.
- **DoS GoldenEye**: This is Layer 7 attack type. Known as HTTP Flood. It consumes all available socket with put/get requests, and try to keep them an alive connection.
- **DoS Slowloris**: Works by sending several incomplete requests that connect to the server and keep these connections open as long as possible, resulting in maximum communication being utilized and thus normal communications being denied.
- **DoS Slowhhtptest**: This attack is designed to run out the server's memory and CPU by sending incomplete requests on the HTTP server to maintain such requests until they are completed and later denied the server.

- **DDoS**: Distributed Denial of Service(DDoS) attacks tries, by flooding the target with traffic flows that operate in a group of hacked/zombie computers as a source of attack traffic, to disrupt the normal traffic of the target server or network.
- **PortScan**: This attack is carried out using malicious techniques to scan port and exploit detected vulnerabilities in network and server access.
- **FTP-Patator**: It is a kind of brute force attack which attempts to crack FTP passwords (login information) with two methods, a dictionary attack or generated passwords.
- **SSH-Patator**: It is a kind of brute force attack which attempts to crack ssh passwords (login information) with two methods, a dictionary attack or generated passwords.
- **Botnet**: The word Botnet is derived from the combination of the words "robot" and "network." Cybercriminals use Malicious proprietary software to infect a large number of users' computer security, take control of each computer, and organize all infected machines into a "bot" network that the criminal can manage remotely for later attacks.
- **Infiltration**: The first step is to exploit security vulnerabilities in applications or to send malicious files into network resources when attempting to attack.
- **Heartbleed**: The attack is directed at a software vulnerability in the popular OpenSSL encryption software library that allows an attacker to directly obtain a server memory block from the vulnerable server via a malicious heartbeat.
- **Web Attack Brute Force**: It uses several automated, consecutive programs to create a wide range of passwords or PINs for getting access to the system.
- **Web Attack XSS**: Scripting Cross-site Attacks Refer to a website or web application to inject malicious code into an attacker for access to a victim's device and therefore to direct the victim to malicious sites or to access his/her private details.
- **Web Attack Sql Injection**: Refer to malicious SQL code for the web site/application form manipulation of injection for access to secret information.

5. EXPERIMENTAL RESULTS

The experiments are done in several computer environments. For developing the models and proof-of-concept Windows 10 i5 64 bit 2.5 GHz, 16 GB RAM, SSD disk is used. For model training, finding the best hyper-parameters and cross-validation, UHEM (National Centre of High Computing) is used. Each node has 28 or 40 cores Xeon CPU (Intel Xeon Gold 6148 v5, Intel Xeon E5-2680 v4) with 128 GB RAM or 512 GB RAM and up to 10 nodes are used (280 cores, 1.2 TB RAM). In our experiment, python (3.6), scikit-learn (20.03), numpy, pandas are used for building the models.

5.1 Step 1 - Anomaly Detection Results

In this step, feature selection is made first. After the feature selection, 12 different machine learning methods used for building models. These models tested separately. For training, these models segmented data set used, which referred in the proposed method. As shown in Table 5.1, accuracy and F1-scores increase when the number of features increases. However, after 70 features, performance no longer increases, and performance decrease. When we look at the confusion matrix in Table 5.3 for feature selection, it is seen that FN (False Negative) samples are very high even if the results are satisfactory. In IDS, FN (False Negative) means that traffic is an attack, but the system accepts as normal traffic.The choice is made by taking the value of $k = 70$. All other experiments similarly are used the same properties. The same feature was not selected for each attack in the other steps. Even if practically done, it will not be suitable for daily life. We can only use it to understand the characteristics of the attack.

Data set tested with two classifiers with select from model for feature selection. Tests results for random forest and SVM are shown in Table 5.2. SVM results better, but note that the number of features is higher than Random Forest Classifier. Also confusion matrices of these tests are shown in Table 5.3 and 5.4.

Table 5.1 : Anova F-Score Based Feature Selection.

Num.of Features	Accuracy	Precision	Recall	F1-score	Train(s)	Test(s)
10	93.5606	90.8034	90.8034	91.3772	705.56	9.40
20	98.3367	98.2530	98.2530	98.2727	918.21	11.61
30	99.7332	99.7324	99.7324	99.7326	932.57	12.41
40	99.7512	99.7505	99.7505	99.7505	1045.88	12.26
50	99.7511	99.7503	99.7503	99.7506	1191.48	13.06
60	99.8838	99.8837	99.8837	99.8837	1258.42	14.11
70	99.8948	99.8948	99.8948	99.8948	1258.42	14.11
77(All features)	99.8925	99.8925	99.8925	99.8925	1205.41	14.17

Table 5.2 : Select from Model Based Feature Selection.

Num.of Features	Algorithm	Accuracy	Precision	Recall	F1-Score
23	Random Forest	99.6953	99.6993	99.6953	99.6961
34	SVM	99.7107	99.7145	99.7107	99.7114

Table 5.3 : Confusion Matrix of 23 features with Random Forest Classifier.

Actual Class	Predicted Class	
	Attack	Normal
Attack	139035	104
Normal	2050	565780

Table 5.4 : Confusion Matrix of 70 features with Random Forest Classifier.

Actual Class	Predicted Class	
	Attack	Normal
Attack	138820	319
Normal	425	567405

As in results of anomaly detection in Table 5.5 shows that Extremely Randomized Tress gives the best result on F1-score. For Accuracy score again, ERT gives the best scores. However, ERT training and testing times are significantly higher than other Decision Tree-based models. For training time, QDA far better than other models, but the accuracy score is too low. As Table 5.5 shows that ensemble methods and decision tree-based models give better performance to SVM, KNN, and LR. kNN algorithm scores are good, but training and testing time is too high for real-world application. As a result, ERT, DT, RF, Bagging, GB give reasonable performance for IDS.

Table 5.5 : Step 1 Anomaly Detection 12 Classifiers Scores and Times.

Algorithm	Accuracy	Precision	Recall	F1-Score	Train(s)	Test(s)
LR	89.885	92.3596	89.885	90.4708	680.65	**0.06**
DT	99.9098	99.9099	99.9098	99.9098	54.71	0.21
RF	99.9062	99.9064	99.9062	99.9063	198.93	11.21
MLP	99.508	99.5112	99.508	99.509	772.82	1.88
ERT	**99.9215**	**99.9216**	**99.9215**	**99.9215**	236.57	13.16
Bagging	99.9042	99.9043	99.9042	99.9043	1000.56	6.44
AdaBoost	99.4881	99.4878	99.4881	99.4879	1526.35	34.52
K-NN	99.6467	99.6482	99.6467	99.6471	1106.44	1564.13
SVM	97.2184	97.4793	97.2184	97.2753	79675.18	8773.27
LDA	89.8766	89.5742	89.8766	89.1028	18.05	**0.06**
QDA	68.0743	87.5688	68.0743	71.2726	**3.88**	0.68
GB	99.7965	99.7964	99.7965	99.7964	472.19	5.47

5.2 Step 2 - Attack Category Detection Results

In this step, the data sets are primarily prepared as specified in the proposed method. Then, models were created and tested with 12 different machine learning algorithms. The objective at this stage is to determine the category or class of the incoming attack. First, the model was created and tested with the classes in the original data set. Then, similar attacks were grouped, and the results were obtained on the data set.

Experiments and results performed as the first option are shown with averaged results in Table 5.6. It would be healthier to look at the F1-score because the data set is imbalanced. Besides, because the class weights are different, weighted and macro results are shown (Table 5.7). If the weighted score is high, and the macro F1-score is low, it indicates that it is good for detecting high weighted classes, but is not successful in detecting individual classes. Table 5.8 shows these results comparatively. Table 5.6 shows that Bagging classifier have highest Accuracy (99.8440%) and F1-score (99.8374%). In Table 5.9 detailed class classification report for Bagging Classifier is given. According to these results DT, RF, ERT and Bagging Classifiers selected for decision fusion.

Table 5.6 : Attack Class Detection Weighted Average Scores.

Algorithm	Accuracy	Precision	Recall	F1-Score	Train(s)	Test(s)
LR	94.7319	98.2852	94.7319	96.2347	182.21	**0.02**
DT	99.8095	99.8269	99.8095	**99.8156**	14.48	0.09
RF	99.8311	99.8198	99.8311	**99.8231**	73.09	12.28
MLP	99.8088	**99.8381**	99.8088	99.761	182.82	0.27
ERT	99.8246	99.821	99.8246	**99.8221**	51.078	12.09
Bagging	**99.8440**	99.8365	**99.8440**	**99.8374**	75.26	1.73
AdaBoost	72.1322	55.2646	72.1322	61.7464	406.54	9.58
K-NN	99.8103	99.7962	99.8103	99.8014	32.66	154.48
SVM	99.7226	99.7577	99.7226	99.7269	1266.73	146.56
LDA	97.9474	98.2864	97.9474	97.8497	4.52	0.08
QDA	99.5997	99.7466	99.5997	99.5793	**0.49**	0.82
GB	99.1907	99.1785	99.1907	99.0325	6011.51	17.14

Table 5.7 : Attack Class Detection Macro Scores.

Algorithm / Macro Score	Precision	Recall	F1-Score
LR	61.2363	60.5521	54.2758
DT	91.6053	**91.6309**	**91.4197**
RF	91.8608	90.585	**91.0553**
MLP	90.3598	82.7806	82.7395
ERT	92.2983	90.7008	**91.3736**
Bagging	92.6829	89.6916	**90.9009**
AdaBoost	47.2312	41.5055	41.5103
K-NN	86.1794	82.8597	84.0011
SVM	79.9088	81.1365	79.7488
LDA	77.8189	70.5913	67.4186
QDA	**94.3058**	90.9648	**89.1765**
GB	79.9713	66.2431	67.1962

Table 5.8 : Attack Class Detection Comparing Weighted Average and Macro F1-Scores.

Algorithm	Average Weight F1-Score	Macro F1 Score
LR	94.7319	54.2758
DT	**99.8095**	**91.4197**
RF	**99.8311**	**91.0553**
MLP	99.8088	82.7395
ERT	**99.8246**	**91.3736**
Bagging	**99.8440**	**90.9009**
AdaBoost	72.1322	41.5103
K-NN	99.8103	84.0011
SVM	99.7226	79.7488
LDA	97.9474	67.4186
QDA	99.5997	89.1765
GB	99.1907	67.1962

Table 5.9 : Bagging Classifier Classification Report.

Attack Type	F1-score	Precision	Recall	Support
Bot	99.7951	100.0000	99.5910	489
DDoS	99.9969	99.9969	99.9969	32006
DoS GoldenEye	99.6503	99.6117	99.6891	2573
DoS Hulk	99.9722	99.9635	99.9809	57531
DoS Slowhttptest	99.3802	99.6345	99.1273	1375
DoS slowloris	99.5169	99.5169	99.5169	1449
FTP-Patator	100.0000	100.0000	100.0000	1984
Heartbleed	100.0000	100.0000	100.0000	3
Infiltration	87.5000	100.0000	77.7778	9
PortScan	99.9710	99.9798	99.9622	39701
SSH-Patator	100.0000	100.0000	100.0000	1474
Web Attack-Brute Force	78.8030	74.3529	83.8196	377
Web Attack-Sql Injection	66.6667	75.0000	60.0000	5
Web Attack XSS	40.0000	46.7213	34.9693	163
Macro avg	90.8037	92.4841	89.6022	139139
Weighted avg	99.8314	99.8298	99.8369	139139
Accuracy				99.83685

Table 5.10 : ERT Classifier Classification Report.

Attack Type	F1-score	Precision	Recall	Support
Bot	99.7951	100.0000	99.5910	489
DDoS	99.9953	99.9906	100.0000	32006
DoS GoldenEye	99.6509	99.4580	99.8445	2573
DoS Hulk	99.9739	99.9774	99.9705	57531
DoS Slowhttptest	99.3445	99.4894	99.2000	1375
DoS slowloris	99.5166	99.5853	99.4479	1449
FTP-Patator	100.0000	100.0000	100.0000	1984
Heartbleed	100.0000	100.0000	100.0000	3
Infiltration	87.5000	100.0000	77.7778	9
PortScan	99.9748	99.9849	99.9647	39701
SSH-Patator	100.0000	100.0000	100.0000	1474
Web Attack - Brute Force	75.8003	73.2673	78.5146	377
Web Attack - Sql Injection	80.0000	80.0000	80.0000	5
Web Attack - XSS	39.6104	42.0690	37.4233	163
Macro avg	91.5116	92.4159	90.8382	139139
Weighted avg	99.8244	99.8238	99.8261	139139
Accuracy				99.826

Table 5.11 : ERT Classifier with Regrouped Classes Classification Report.

Attack Type	F1-score	Precision	Recall	Support
Botnet	99.7950	100	99.5910	489
BruteForce	100	100	100	3458
DosDDos	99.9910	99.9852	99.9968	94937
Infiltration	100	100	100	9
PortScan	99.9760	99.9974	99.9546	39701
WebAttack	99.0892	98.3725	99.8165	545
macro avg	99.8085	99.7258	99.8931	139139
weighted avg	99.9827	99.9828	99.9827	139139
Accuracy				99.9827

The second step is to create new classes by grouping similar attacks. After the data set was prepared, the models were trained and tested, as indicated in the proposed method. Table 5.11 shows ERT classification report which most successful classifier in this stage. When we compare this result to Table 5.10, it is seen that the rate of detecting web attacks with a detection rate of 37-80% in normal classification reached 99.72% with reclassification. Moreover for comparing, as indicated step-1, weighted means (Table 5.12) and macro scores (Table 5.13) are shown.

Table 5.12 : Attack Class Detection with Regrouped Classes Weighted Average Scores.

Algorithms	Accuracy	Precision	Recall	F1-Score	Train(s)	Test(s)
LR	98.6927	98.8566	98.6927	98.7459	164.22	**0.02**
DT	99.9626	99.963	99.9626	**99.9627**	8.93	0.08
RF	99.9792	99.9792	99.9792	**99.9792**	61.06	6.33
MLP	99.9483	99.9481	99.9483	99.9479	123.29	0.27
ERT	**99.9828**	**99.9828**	**99.9828**	**99.9828**	48.00	6.07
Bagging	99.9705	99.9707	99.9705	**99.9706**	68.03	1.29
AdaBoost	93.4864	95.5736	93.4864	93.7991	161.47	3.83
K-NN	99.9633	99.9633	99.9633	**99.9631**	35.53	197.08
SVM	99.7945	99.8057	99.7945	99.7981	1491.78	86.65
LDA	98.7135	98.8018	98.7135	98.713	4.49	0.08
QDA	96.2728	99.5131	96.2728	97.7061	**1.42**	0.42
GB	99.9281	99.9235	99.9281	99.9256	2756.98	5.85

Table 5.13 : Attack Class Detection with Regrouped Classes Macro Scores.

Algorithms	Precision	Recall	F1-Score
LR	68.3322	76.7877	70.7905
DT	99.3559	99.7002	**99.5258**
RF	**99.8120**	99.7398	**99.7758**
MLP	97.5332	95.1791	96.2986
ERT	99.7259	**99.8932**	**99.8086**
Bagging	96.3245	97.6989	**96.9726**
AdaBoost	73.1629	68.6165	61.567
K-NN	99.6215	95.5542	**97.3536**
SVM	86.2123	91.48	88.3593
LDA	86.6893	74.3048	74.9026
QDA	68.1343	81.8537	68.947
GB	82.2209	81.5446	81.8723

The comparative table of weighted and macro scores is shown in Table 5.14. The goal here is that both values are highest. ERT, DT, RF classifiers meet these requirements. Both F1-scores are above 99.50%. Bagging was chosen as the classifier for the next stage. kNN has the highest test and training time. As a result, ERT, Bagging, DT, RF classifiers will be used for decision fusion.

Table 5.14 : Attack Class Detection with Regrouped Classes Comparing Weighted Average and Macro F1-Scores.

Algorithm	Average Weighted F1 Score	Macro F1 Score
LR	98.7459	70.7905
DT	**99.9627**	**99.5258**
RF	**99.9792**	**99.7758**
MLP	99.9479	96.2986
ERT	**99.9828**	**99.8086**
Bagging	**99.9706**	**96.9726**
AdaBoost	93.7991	61.567
K-NN	**99.9631**	**97.3536**
SVM	99.7981	88.3593
LDA	98.713	74.9026
QDA	97.7061	68.947
GB	99.9256	81.8723

5.3 Step 3 - Decision Fusion Results

At the end of the second step, the best four classifiers (ERT, Bagging, Decision Tree, Random Forest) used for the decision fusion process. In this step, three different methods were used for comparison. Majority Voting, the class with the most votes is

selected. Note that the same class weights were used in the second method prediction based on class prediction probabilities. The last method is the ADF method which performs online adaptive class weight update.

Table 5.15 : Attack Class Detection Using Decision Fusion Methods.

Algorithm	Accuracy	Precision	Recall	F1-Score	Macro-F1	Train(s)	Test(s)
MV	99.8412	99.8313	99.8412	99.832	91.0535	126.91	28.42
MV Prob.	99.8404	**99.8416**	99.8404	**99.841**	**92.6199**	195.57	43.96
ADF	99.8361	99.8291	99.8361	99.8302	90.7886	169.69	43
S2-Baging	**99.844**	99.8365	**99.844**	99.8374	90.9009	**75.26**	**1.73**

In Table 5.15, the best results of the experiments performed with normal class labels are given by Majority Voting-probability based decision-fusion with 99.84% F1 score and 99.84% accuracy rate and 99.83% detection rate. Bagging classifier, which was developed in the second stage, gives very close results. However, there is a 2% difference in the F1-macro result. This result shows that the method can better detect different attacks. Among the decision-fusion methods, the lowest F1-score, albeit with a slight difference, has the ADF algorithm. Since the classifiers have very high performance, the ADF algorithm has not been able to achieve enough performance enhancements.

Table 5.16 : Attack Class Detection with Grouped Classes Using Decision Fusion Methods.

Algorithm	Accuracy	Precision	Recall	F1-Score	Macro-F1	Train(s)	Test(s)
MV	99.9792	99.9794	99.99792	99.9792	98.0943	113.54	17.05
MV Prob.	**99.9806**	**99.9807**	**99.9806**	**99.9806**	**99.7774**	214.94	22.9
ADF	99.9777	99.9779	99.9777	99.9778	97.1141	204.85	27.72
S2-ERT	**99.9828**	**99.9828**	**99.9828**	**99.9828**	**99.8086**	**48**	**6.07**

The results for class regrouping are shown in Table 5.16. Here, the decision fusion methods failed, and the second step ERT classifier was more successful than decision fusion methods. With very little performance difference, probability-based majority voting ranks second and the most successful of decision-making methods. The highest performance in regrouping classes is the ERT classifier with a 99.98% F1-score.

5.4 Comparison to Other Studies

In the literature studies, the results of the experiments on the same data set are shown. In these experiments, sometimes recall, and F1-score were used as accuracy metrics. Different machine learning algorithms are combined with different feature selection methods in these studies. Besides, deep learning methods have been widely used. In these studies, the class weights usually found in the data set were changed, and training and test sets were reduced. Normal traffic weight has been reduced to eliminate imbalance between classes. There is no standard in data segmentation. In some studies, only certain classes of the data set were taken into two categories for detecting specific attack type. Models were created with binary classification, and test operations were performed.

These problems occur because the CICIDS2017 data set is not distributed separately as a test and training set. Also, since the sample set of some classes in the data set is very small, the probability of detection is reduced. In the second step, we tried to solve this problem by regrouping the classes. However, experiments were performed while maintaining the class weights in the original data set. Table 5.17 shows both results.

According to the studies examined, the proposed method has the highest success. For such imbalanced data sets, the F1-score is a better metric as a success metric. An ideal intrusion detection system should have as few undetected intrusions as possible.

Table 5.17 : Comparison of the Proposed Method to Other Studies.

Study	Year	Score	Method
R. Vinayakumar et.al. [68]	2018	Binary Classification: 93.9% (F1) Multi-Class: 96.5% (F1)	Deep Neural Network (DNN)
Piotr Gaj et.al. [69]	2019	SNN:98.4% (Accuracy), DNN: 98.40% (Accuracy)	SNN, DNN
Iman Sharafaldin et.al. [5]	2018	ID3: 98% (F1), RF: 97% (F1)	ID3, RF
Ahmed Ahmim et.al. [13]	2018	94.475% (Recall-DR) 96.665%(Accuracy)	Hierarchical
Watson, Gavin [70]	2018	%94.8(F1)	MLP, Payload Related Features
Huiqiang Wang [71]	2018	92.1% (F1)	SVM,DBN
Aksu Dogukan et.al. [72]	2018	%99.7(F1)	KNN, Fischer Scoring
Bansal, Ashua et. al. [73]	2018	%99.5 (F1)	XgBoost for Dos Attack
Dogukan Aksu et. al. [74]	2018	97.80% (Accuracy)	Deep Learning for PortScan Attacks
R. Abdulhammed et.al. [8]	2019	%99.6% (F1)	Random Forest, PCA, UDBB
	2019	Binary Classification: 99.92% (F1) Multi-Class: 99.84% (F1)	Proposed Method
	2019	Binary Classification:99.92% (F1) Multi-Class: 99.98% (F1)	Proposed Method-Regroup Class

6. CONCLUSION

One of the most critical emerging issues today is cybersecurity. In parallel with the development and spread of information technologies, cyber attacks are increasing. One of the essential parts of cybersecurity is intrusion detection systems. In parallel with machine learning, AI, and increasing the computing power of computers, intrusion detection systems have developed, and anomaly-based intrusion detection systems have started to become widespread instead of signature-based. In order to develop these systems and to create suitable models, data sets were created from synthetic and real network environment. In this study, the most recent CICIDS2017 data set was used. This data set contains up-to-date attacks that are synthetically created and suitable for today's network environment.

In order to establish a successful intrusion detection system, a multi-stage hybrid method was developed in this study. It was determined whether there was an anomaly/attack in the first stage. In the following stages, the type/category of the attack was tried to be predicted. Machine learning methods were used for this purpose. In this study, 12 different machine learning methods were used for the first two stages. While training and testing these models, it is aimed to increase performance by using preliminary data processing and feature selection methods. Besides, experiments on regrouping classes/attacks were conducted. In the third stage, decision fusion methods were used for more accurate attack type detection.

The system developed by the proposed method has high intrusion detection rate and accuracy rate. A system with a 99.98% F1-score with a 99.98% detection rate was developed by regrouping the classes. 99.84% accuracy score and 99.83% F1-score were obtained in the experiments conducted considering the original classes in the data set. According to the literature review, the highest achievement system has been developed. These performance scores are on a previously unseen test set. In the proposed method, the data were first separated into training, and test clusters, and then the test set was used only for the final experiment. The best hyperparameters were found for this performance, and cross-validation was performed.

6.1 Future Work

The developed system is planned to be tested on different data sets. It is planned to use UNSW-NB15, CICIDS2018 data sets, which are also up-to-date. Also, it is planned to use NSL-KDD data set, which is old but used as a reference. It is also planned to use Deep Learning methods to make comparisons and develop better models. The data sets are too large because it consists of too many records and properties. For that, it is aimed to use Hadoop and Spark extensive data processing methods for faster data processing in these data sets. Besides, it is aimed to use the EADF algorithm mentioned in Toreyin's [20] article in multiple classification and to use it in decision fusion. Finally, it is planned to develop a component that classifies and learns unknown attacks. Thus, it was aimed to be used as a commercial product.

REFERENCES

[1] **Ring, M.**, **Wunderlich, S.**, **Scheuring, D.**, **Landes, D. and Hotho, A.** (2019). A survey of network-based intrusion detection data sets, *Computers & Security*.

[2] Logistic Regression: A Simplified Approach Using Python, `https://cutt.ly/mwZHtm4`, accessed: 24.08.2019.

[3] **statista.com**, `https://www.statista.com/statistics/617136/digital-population-worldwide/`, access Date: 25.04.2019.

[4] **statista.com**, `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`, access Date: 25.04.2019.

[5] **Sharafaldin, I.**, **Gharib, A.**, **Habibi Lashkari, A. and Ghorbani, A.** (2017). Towards a Reliable Intrusion Detection Benchmark Dataset, *Software Networking*, *2017*, 177–200.

[6] KDD Cup 1999 Data, `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, access Date: 14.03.2019.

[7] NSL-KDD Data Set, `https://www.unb.ca/cic/datasets/nsl.html`, access Date: 12.03.2019.

[8] **Abdulhammed, R.**, **Musafer, H.**, **Alessa, A.**, **Faezipour, M. and Abuzneid, A.** (2019). Features Dimensionality Reduction Approaches for Machine Learning Based Network Intrusion Detection, *Electronics*, *8*, 322.

[9] **Ahmim, A.**, **Maglaras, L.**, **Ferrag, M.A.**, **Derdour, M. and Janicke, H.** (2018). A Novel Hierarchical Intrusion Detection System based on Decision Tree and Rules-based Models.

[10] **M., N.M.A.** (2017). Designing an online and reliable statistical anomaly detection framework for dealing with large high-speed network traffic, *Ph.D. thesis*.

[11] **Li, L.**, **Yu, Y.**, **Bai, S.**, **Hou, Y. and Chen, X.** (2018). An Effective Two-Step Intrusion Detection Approach Based on Binary Classification and k -NN, *IEEE Access*, *6*, 12060–12073.

[12] **Shone, N.**, **Ngoc, T.N.**, **Phai, V.D. and Shi, Q.** (2018). A Deep Learning Approach to Network Intrusion Detection, *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(1), 41–50.

[13] **Ahmim, A.**, **Maglaras, L.A.**, **Ferrag, M.A.**, **Derdour, M. and Janicke, H.** (2018). A Novel Hierarchical Intrusion Detection System based on Decision Tree and Rules-based Models, *CoRR*, *abs/1812.09059*, `http://arxiv.org/abs/1812.09059`, `1812.09059`.

[14] **Moustafa, N.**, The UNSW-NB15 Dataset Description, `https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/`, access Date: 20.05.2019.

[15] Intrusion Detection Evaluation Dataset (CICIDS2017), `https://www.unb.ca/cic/datasets/ids-2017.html`, access Date: 12.03.2019.

[16] CSE-CIC-IDS2018 on AWS, `https://www.unb.ca/cic/datasets/ids-2018.html`, access Date: 12.03.2019.

[17] **Timčenko, V. and Gajin, S.** (2017). Ensemble classifiers for supervised anomaly based network intrusion detection, *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp.13–19.

[18] **Salman, T.**, **Bhamare, D.**, **Erbad, A.**, **Jain, R. and Samaka, M.** (2017). Machine Learning for Anomaly Detection and Categorization in Multi-Cloud Environments, *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pp.97–103.

[19] **Zhang, Y.**, **Chen, X.**, **Jin, L.**, **Wang, X. and Guo, D.** (2019). Network Intrusion Detection: Based on Deep Hierarchical Network and Original Flow Data, *IEEE Access*, *7*, 37004–37016.

[20] **Gunay, O.**, **Toreyin, B.U.**, **Kose, K. and Cetin, A.E.** (2012). Entropy-Functional-Based Online Adaptive Decision Fusion Framework With Application to Wildfire Detection in Video, *IEEE Transactions on Image Processing*, *21*(5), 2853–2865.

[21] **D., A.**, (2018). Performance Analysis of Log-Based Intrusion Detection Systems, Master's thesis.

[22] **C., K.**, (2016). Use of Machine Learning Techniques in Intrusion Detection Systems:Comparative Analysis of Performance, Master's thesis.

[23] Sentinel One, `https://www.sentinelone.com/`, accessed: 24.04.2019.

[24] Carbon Black, `https://www.carbonblack.com/`, accessed: 24.04.2019.

[25] **Chen, Y.**, **Garcia, E.K.**, **Gupta, M.R.**, **Rahimi, A. and Cazzanti, L.** (2009). Similarity-based Classification: Concepts and Algorithms, *Journal of Machine Learning Research*, *10*, 747–776.

[26] **Alpaydin, E.** (2014). *Introduction to Machine Learning*, The MIT Press.

[27] **Ramanan, D. and Baker, S.** (2011). Local Distance Functions: A Taxonomy, New Algorithms, and an Evaluation, *IEEE Trans. Pattern Anal. Mach. Intell.*, *33*(4), 794–806.

[28] **Mitchell, T.M.** (1997). *Machine Learning*, McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[29] **Gupta, B.**, **Rawat, A.**, **Jain, A.**, **Arora, A. and Dhami, N.** (2017). Analysis of Various Decision Tree Algorithms for Classification in Data Mining, *International Journal of Computer Applications*, *163*, 15–19.

[30] **Breiman, L.**, **Friedman, J.H.**, **Olshen, R.A. and Stone, C.J.** (1984). *Classification and Regression Trees*, Wadsworth and Brooks, Monterey, CA.

[31] **Quinlan, J.R.** (1986). Induction of Decision Trees, *Mach. Learn.*, *1*(1), 81–106.

[32] **Quinlan, J.R.** (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[33] scikit-learn Machine Learning in Python, `https://scikit-learn.org/stable/`, accessed: 24.08.2019.

[34] **Singh, S. and Gupta, P.** (2014). Comparative Study Id3, Cart And C4.5 Decision Tree Algorithm: A Survey, *International Journal of Advanced Information Science and Technology (IJAIST)*, *27*(27).

[35] **Bahgat, A.**, **Elseddawy, A.**, **Sultan, T. and Khedr, A.** (2013). Applying Classification Technique using DID3 Algorithm to improve Decision Support System under Uncertain Situations, *International Journal of Engineering Research*, *3*, 2249–6645.

[36] **Christopher D. Manning, P.R. and Schütze, H.**, (2008). Introduction to Information Retrieval, Cambridge University Press, California,USA, 1. edition.

[37] **Gaurang, P.**, **Ganatra, A.**, **Kosta, Y. and Panchal, D.** (2011). Behaviour Analysis of Multilayer Perceptronswith Multiple Hidden Neurons and Hidden Layers, *International Journal of Computer Theory and Engineering*, *3*, 332–337.

[38] Multi-Layer Neural Networks, `https://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn12-notes-nup.pdf`, accessed: 2019-08-15.

[39] **Murphy, K.P.** (2013). *Machine learning : a probabilistic perspective*, MIT Press.

[40] **Döring, M.**, Linear, Quadratic, and Regularized Discriminant Analysis, `https://www.datascienceblog.net/post/machine-learning/linear-discriminant-analysis/`, accessed: 2019-06-12.

[41] **Martin., D.J..J.H.** (2019). *Speech and Language Processing*, Standford University, third edition draft edition.

[42] **Polikar, R.**, (2012). Ensemble Learning In: Zhang C., Ma Y. (eds) Ensemble Machine Learning., Springer, Boston, MA, pp.1–34.

[43] **Breiman, L.** (2001). Random Forests, *Mach. Learn.*, *45*(1), 5–32.

[44] **Hastie, T.**, **Tibshirani, R. and Friedman, J.** (2009). *The elements of statistical learning: data mining, inference and prediction*, Springer, 2 edition.

[45] **Breiman, L.** (1996). Bagging Predictors, *Machine Learning*, *24*(2), 123–140, https://doi.org/10.1023/A:1018054314350.

[46] **Efron, B. and Tibshirani, R.J.** (1993). *An Introduction to the Bootstrap*, number 57 in Monographs on Statistics and Applied Probability, Chapman & Hall/CRC, Boca Raton, Florida, USA.

[47] **Opitz, D. and Maclin, R.** (1999). Popular Ensemble Methods: An Empirical Study, *Journal of Artificial Intelligence Research*, *11*, 169–198.

[48] Bagging, https://www.stat.cmu.edu/~ryantibs/datamining/lectures/24-bag.pdf, accessed: 20.08.2019.

[49] **Pliakos, K. and Vens, C.** (2016). Feature Induction based on Extremely Randomized Tree Paths.

[50] **Geurts, P.**, **Ernst, D. and Wehenkel, L.** (2006). Extremely randomized trees, *Machine Learning*, *63*(1), 3–42, https://doi.org/10.1007/s10994-006-6226-1.

[51] **Kearns, M. and Valiant, L.** (1994). Cryptographic Limitations on Learning Boolean Formulae and Finite Automata, *J. ACM*, *41*(1), 67–95, http://doi.acm.org/10.1145/174644.174647.

[52] **Freund, Y. and Schapire, R.E.** (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, *J. Comput. Syst. Sci.*, *55*(1), 119–139, http://dx.doi.org/10.1006/jcss.1997.1504.

[53] **Ivankovic, Z.**, **Markoski, B.**, **Ivkovic, M.**, **Radosav, D. and Pecev, P.** (2012). AdaBoost in basketball player identification, 151–156.

[54] **Wyner, A.J.**, **Olson, M.**, **Bleich, J. and Mease, D.** (2017). Explaining the Success of AdaBoost and Random Forests as Interpolating Classifiers, *Journal of Machine Learning Research*, *18*(48), 1–33, http://jmlr.org/papers/v18/15-240.html.

[55] **Friedman, J.**, **Hastie, T. and Tibshirani, R.** (2000). Additive Logistic Regression: A Statistical View of Boosting, *The Annals of Statistics*, *28*, 337–407.

[56] **Friedman, J.** (2000). Greedy Function Approximation: A Gradient Boosting Machine, *The Annals of Statistics*, *29*.

[57] **Natekin, A. and Knoll, A.** (2013). Gradient boosting machines, a tutorial, *Frontiers in Neurorobotics*, *7*, 21.

[58] **Pittman, S. and Brown, K.** (2011). Multi-Scale Approach for Predicting Fish Species Distributions across Coral Reef Seascapes, *PloS one*, *6*, e20583.

[59] **Bissacco, A.**, **Yang, M.H. and Soatto, S.** (2007). Detecting humans via their pose, *Advances in Neural Information Processing Systems*, pp.169–176.

[60] **M., S.R.V.**, (2017). Python Machine Learning Second Edition, Packt Publishing, UK, 2. edition.

[61] **Gunay, O.**, **Tasdemir, K.**, **Toreyin, B.U. and Cetin, A.E.** (2009). Video Based Wildfire Detection at Night, *Fire Safety J.*, *44*(6), 860–868.

[62] **Niesen, U.**, **Shah, D. and Wornell, G.W.** (2009). Adaptive Alternating Minimization Algorithms, *IEEE Transactions on Information Theory*, *55*(3), 1423–1429.

[63] Feature Importance With XGBoost in Python, `https://cutt.ly/HwZHr43`, accessed: 20.07.2019.

[64] **Sharafaldin., I.**, **Lashkari., A.H. and Ghorbani., A.A.** (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization, *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, INSTICC, SciTePress, pp.108–116.

[65] **CICFlowmeter**, `http://netflowmeter.ca/`, access Date: 20.04.2019.

[66] **Faker, O.M.F.**, (2019). Intrusion Detection Using Big Data and Deep Learning Techniques, Master's thesis.

[67] Netflowmeter, `http://www.netflowmeter.ca/netflowmeter.html`, accessed: 24.08.2019.

[68] **Vinayakumar, R.**, **Alazab, M.**, **Soman, K.P.**, **Poornachandran, P.**, **Al-Nemrat, A. and Venkatraman, S.** (2019). Deep Learning Approach for Intelligent Intrusion Detection System, *IEEE Access*, *7*, 41525–41550.

[69] **Gaj, P.**, **Sawicki, M. and Kwiecień, A.** (2019). *Computer Networks: 26th International Conference, CN 2019, Kamień Śląski, Poland, June 25-27, 2019, Proceedings*, Communications in Computer and Information Science, Springer International Publishing, `https://books.google.com.tr/books?id=stqdDwAAQBAJ`.

[70] **Watson, G.M.** (2018). A Comparison of Header and Deep Packet Features when Detecting Network Intrusions.

[71] **Marir, N.**, **Wang, H.**, **Feng, G.**, **Li, B. and Jia, M.** (2018). Distributed Abnormal Behavior Detection Approach Based on Deep Belief Network and Ensemble SVM Using Spark, *IEEE Access*, *6*, 59657–59671.

[72] **Aksu, D.**, **Üstebay, S.**, **Aydin, M.A. and Atmaca, T.** (2018). Intrusion Detection with Comparative Analysis of Supervised Learning Techniques and Fisher Score Feature Selection Algorithm, ***T. Czachórski**, **E. Gelenbe**, **K. Grochla and R. Lent**, editors, Computer and Information Sciences*, Springer International Publishing, Cham, pp.141–149.

[73] **Bansal, A. and Kaur, S.** (2018). Extreme Gradient Boosting Based Tuning for Classification in Intrusion Detection Systems, ***M. Singh**, **P.K. Gupta**, **V. Tyagi**, **J. Flusser and T. Ören**, editors, Advances in Computing and Data Sciences*, Springer Singapore, Singapore, pp.372–380.

[74] **Aksu, D. and Aydin, M.A.** (2018). Detecting Port Scan Attempts with Comparative Analysis of Deep Learning and Support Vector Machine Algorithms, *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, 77–80.

CURRICULUM VITAE

Name Surname: Erkan AS

Place and Date of Birth: Midyat, 04.02.1985

E-Mail: erkan.as@outlook.com

EDUCATION:

- **B.Sc.:** 2014, Yildiz Technical University, Faculty of Electrical and Electronic Engineering , Electronics and Communications Engineering

PROFESSIONAL EXPERIENCE:

- 2015 -, Smartme Technology Ltd., Co-Founder Project Manager.
- 2011-2014, N-value, E-commerce Project Manager.
- 2009-2011, Bulan Software, Project Manager
- 2009 , Dogan Online A.S,System Admin Specialist