SEPTEMBER 2019 M.Sc. in Electronics and Computer Engineering BAYRAM KOTAN

T.C.

HASAN KALYONCU UNIVERSITY

GRADUATE SCHOOL OF

NATURAL & APPLIED SCIENCES

NETWORK MONITORING SYSTEM USING MACHINE LEARNING

COMPARATIVE ANALYSIS OF CLASSIFICATION TECHNIQUES FOR NETWORK TRAFFIC MONITORING

M. Sc. THESIS

IN

ELECTRONICS AND COMPUTER ENGINEERING

BAYRAM KOTAN

SEPTEMBER 2019

Network Monitoring System Using Machine Learning

Comparative Analysis of Classification Techniques for Network Traffic Monitoring

M. Sc. THESIS

IN

ELECTRONICS AND COMPUTER ENGINEERING

HASAN KALYONCU UNIVERSITY

Supervisor

Assistant Professor Mohammed K. M. MADI

Bayram KOTAN

SEPTEMBER 2019

	GRADUATE SCHOOL OF NATURAL & APPLIED SCIENCES INSTITUTE **M.Sc. ACCEPTANCE AND APPROVAL FORM**

Electronics-Computer Engineering M.Sc. (Master Of Science) programme student **Bayram KOTAN** prepared and submitted the thesis titled **"Network Monitoring System Using Machine Learning Comparative Analysis Of Classification Techniques For Network Traffic Monitoring"** defendant successfully on the date of …./…./…. and accepted by the jury as an M.Sc. Thesis.

Position	**Title. Name and Surname** **Department/University**	**Signature:**
Jury Head	Assist. Prof. Dr. Tolgay KARA Electrical and Electronics Engineering Department Gaziantep University	
M.Sc. Supervisor **Jury Member**	Assist. Professor Mohammed K. M. MADI Computer Engineering Department Hasan Kalyoncu University	
Jury Member	Assist. Prof. Dr. Saed AL-QARALEH Computer Engineering Department Hasan Kalyoncu University	

This thesis is accepted by the jury members selected by the institute management board and approved by the institute management board.

Prof. Dr. Mehmet KARPUZCU

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Bayram KOTAN

Signature

ABSTRACT

NETWORK MONITORING SYSTEM USING MACHINE LEARNING COMPARATIVE ANALYSIS OF CLASSIFICATION TECHNIQUES FOR NETWORK TRAFFIC MONITORING

KOTAN, BAYRAM

M.Sc. in Electronics and Computer Engineering

Supervisor: Assistant Professor Mohammed K. M. MADI

September 2019

124 pages

Online network traffic classification continues to be the focus of long-term interest. Network traffic monitoring and analysis can be done for many different reasons. Generally, it provides raw data input for network monitoring, Quality of Service (QoS) and intrusion detection. Specifically, network traffic monitoring enables the network analyst to understand network resources use and identify network performance. With this information, network analyst may adjust QoS policies to control and manage network resources. This aim is achieved by setting priorities for specific types of data in the network and logging the traffic to comply with the regulations. Network traffic monitoring can be used to create models for academic research. In this thesis, a machine-learning approach that accurately classifies network traffic using Decision Tree Algorithm (DT) is presented and implementing the Principal Component Analysis (PCA) Algorithm for reduction, side by side, to reach the best optimization. Machine learning technology will generate better solutions to monitor and classify network traffic as a result of highly accurate data mining technics and advanced statistics. The purpose of this thesis is to build a Network Monitoring System (NMS) using modern machine learning technologies that works in both online and offline modes. DT algorithm; one of the available data mining algorithms; is used to build the classifier of network. The experiment's results showed that NMS based system has 97.7486 % accuracy (ACC) in successfully classifying the network traffic.

Keywords: Machine Learning, Artificial Intelligence, Traffic Classification, Decision Tree Algorithm, Principal Component Analysis Algorithm, KDD CUP99 dataset.

ÖZET

NETWORK MONITORING SYSTEM USING MACHINE LEARNING COMPARATIVE ANALYSIS OF CLASSIFICATION TECHNIQUES FOR NETWORK TRAFFIC MONITORING

KOTAN, BAYRAM
Yüksek Lisans Tezi, Elektronik Bilgisayar Müh. Bölümü
Tez Yöneticisi: Assistant Professor Mohammed K. M. MADI
Eylül 2019
124 sayfa

Çevrimiçi ağ trafiği sınıflandırması, uzun vadeli ilginin odak noktası olmaya devam ediyor. Ağ trafiğini izleme ve Ağ trafiği analizi birçok farklı yoldan yapılabilir. Genellikle, ağ trafiğini izleme, hizmet kalitesi (QoS) ve izinsiz giriş tespiti için ham veri girişi sağla. Özellikle, ağ trafiğini izleme, ağ analistine ağ kaynaklarını nasıl kullandığını anlama ve ağ performansını belirleme olanağı sağlar. Bu bilgi ile ağ analisti, ağ kaynaklarını kontrol etmek ve yönetmek için QoS politikalarını ayarlayabilir. Bu amaca, ağdaki belirli veri tipleri için önceliklerin ayarlanması ve trafiğin yönetmeliklere uyması için günlüğe kaydedilmesi ile ulaşılmaktadır. Ağ trafiğinin izlenmesi akademik araştırma için modeller oluşturmak için kullanılabilir. Bu tezde, en yakın optimizasyona ulaşmak için Karar Ağacı Algoritmasını (DT) kullanarak ve Temel Bileşen Analizi (PCA) Algoritmasını kullanarak ağ trafiğini doğru şekilde sınıflandıran bir makine öğrenme yaklaşımı sunulmaktadır. Makine öğrenimi teknolojisi, yüksek doğrulukta veri madenciliği teknikleri ve ileri istatistiklerin bir sonucu olarak ağ trafiğini izlemek ve sınıflandırmak için daha iyi çözümler üretecektir. Bu tezin amacı, hem çevrimiçi hem de çevrimdışı olarak çalışan modern makine öğrenme teknolojilerini kullanarak bir Ağ İzleme Sistemi (NMS) inşa etmektir. DT algoritması (mevcut veri madenciliği algoritmalarından biri) ağın sınıflandırıcısını oluşturmak için kullanılır. Deney sonuçları, NMS tabanlı sistemin ağ trafiğini başarılı bir şekilde sınıflandırmada %97,7486 doğruluğa (ACC) sahip olduğunu göstermiştir.

Anahtar Kelimeler: Makine Öğrenmesi, Yapay Zekâ, Trafik Sınıflandırması, K-En Yakın Komşular Algoritması, Temel Bileşen Analizi Algoritması, KDD CUP99 veri seti.

I dedicate this research to my family, my parents and teachers for their endless support and encouragement in completing this thesis.

I would like to express my sincere gratitude and deep appreciations to my major professor, Assistant Professor Mohammed K. M. MADI, who showed an extreme engagement with my work, guided me through the uphill of research, and always pointing the right direction to reach my objectives.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACC	: Accuracy Rate
CM	: Confusion Matrix
CPU	: Central Processing Unit
DARPA	: Defense Advanced Research Project Agency
DL	: Deep Learning
DPI	: Deep Packet Inspection
DR	: Detection Rate
DT	: Decision Tree
FN	: False Negative
FP	: False Positive
FTP	: File Transferring Protocol
GNB	: Gaussian Naïve Bayes
IANA	: Internet Assigned Numbers Authority
IP	: Internet Protocol
IDS	: Intrusion Detection System
KDD	: Knowledge Discovery and Data
KDD CUP99	: Knowledge Discovery and Data Mining CUP1999
LAN	: Local Area Network
LR	: Logistic Regression
MAC	: Media Address Control
NAT	: Network Address Translation
NMS	: Network Monitoring System
PCA	: Principal Component Analysis
P2P	: Peer to peer
PSP	: Percentage of Successful Prediction

QoS	: Quality of Service
RBF	: Radial Basis Function
SVM	: Support Vector Machine
SYN	: Synchronize
TN	: True Negative
TP	: True Positive
TPR	: True Positive Rate
TCP	: Transmission Control Protocol
TCP/IP	: Transmission Control Protocol/Internet Protocol
UDP	: User Datagram Protocol
Npcap	: Packet Capture Library for Windows

CHAPTER 1

INTRODUCTION

1.1 Introduction

Internet is evolving to a tremendous and ubiquitous network of networks, containing increasingly huge data and digital media communication, and generating enormous revenues every day to all businesses worldwide. Data transmission is managed by simple protocols; Transmission Control Protocol (TCP) and User Datagram Protocol (UDP); without monitoring, inspection and intelligent control over the traffic built in functionality (Cerf & Kahn, 1974). Businesses and governments need applications to classify and monitor network traffic, manage its resources and detect possible anomalies to protect their investments and interests. In general, the Internet traffic is the product of a complex system containing diverse of networks, hosts, applications and different clients interacting with each other.

Network traffic classification (monitoring) has attracted great attention nowadays (Karagiannis, Papagiannaki, & Faloutsos, 2005; Kim et al., 2008; Lim et al., 2010; Nguyen, Armitage, & Tutorials, 2008; Wu, Min, Li, & Javadi, 2009). Classification of traffic flows according to production applications has very important part in security and network management, like QoS control, intrusion detection and lawful interception (Xiang, Zhou, Guo, & Systems, 2008). Our days, billions of devices use Internet resources. Every device sends requests for connection to other devices and exchange data over the Internet. As a result, huge amount of traffic will be generated, so classification is necessary; not only for QoS or for maintaining availability of resources; but also, for efficient processing of information.

Manual labeling of data samples is mostly tiring, time wasting and costly. This complexity is continuously increasing by wide range of network applications are produced every day. Therefore, we need a system that can learn and apply. In this context, it will be more useful to apply machine learning.

Network monitoring can be succeeded through port-based traffic classification methods (Ioffe & Szegedy, 2015), payload-based classification methods (Deep packet inspection) (Ioffe & Szegedy, 2015) and flow features-based classification methods (Machine learning and statistical feature) (Ioffe & Szegedy, 2015). Many classification methods have been suggested (Auld, Moore, & Gull, 2007; Crotti, Gringoli, Pelosato, & Salgarelli, 2006), as interest in traffic classification increases. Port based method is known as one of the best techniques for network traffic classification (Namdev, Agrawal, & Silkari, 2015). This method uses network ports that are firstly registered in Internet Assigned Numbers Authority (IANA). However, this method has failed in correctly classifying Point to Point (P2P) applications, which use unregistered network port number and uses dynamic port numbering (Karagiannis, Broido, & Faloutsos, 2004). Payload based methods gives better classification results (Karagiannis et al., 2004). Yet, this method fails to classify encrypted traffic. Note that many network applications use encryption to protect data from detection (Haffner, Sen, Spatscheck, & Wang, 2005; Sen, Spatscheck, & Wang, 2004). Many network classification methods have been proposed using machine learning to monitor network traffic. We will propose a technique for classifying network traffic based on ML. Machine Learning Method gives very accurate results in traffic classification (Namdev et al., 2015). This Method uses training and testing data sets to classify unknown traffic classes.

1.2 Statement of the Problem

Network specialists work day in and day out attempting to sift through incredible amounts of data from network (server logs, network packets and network controllers). Nowadays billions of devices use internet resources. Every device sends requests for connection to other devices and exchange data over the internet. As a result, huge amount of traffic will be generated in network, so classification is necessary for network management (Park, Tyan, & Kuo, 2006).

In addition, to monitor all the packets traffic simultaneously on a network will be not easy. (A. Moore, Hall, Kreibich, Harris, & Pratt, 2003). Protocols overlapping or protocol layering complicate the fast monitoring and extraction of the features. To overcome these challenges, machine learning technology is one of the best solutions.

1.3 Specific objectives

The primary purpose of this study is to apply the machine learning methods in the network traffic classification and to evaluate the results. To achieve this goal, the following goals should be considered:

- To examine the methods available for classifying network traffic using ML.
- To draft the methods taxonomy identified and provide the advantage, the disadvantages and weakness.
- To evaluate the performance of the identified method and compare it with other methods.

1.4 Significance of the Study

This thesis aims at building a network monitoring system that use Machine Learning for classifications of network packets. This can be achieved by using fast machine learning algorithms that can process and analyze network traffic. In a short description, we will accurately define traffic classes by maximizing the Detection Rate (DR), determining the class of any packet recorded on the basis of recognized classification patterns. During the training stage, these classification patterns are produced to raise the detection rate. The significance of this study lies in describing and analyzing the best method of Network Monitoring that can be used for Machine Learning.

1.5 Organization of Thesis

The thesis is comprised of five chapters. Chapter two provides theoretical background consisting of a general NMS, a short overview of NMS methods, the definition of Knowledge Discovery Data Mining (KDD CUP99 data set), DT and PCA algorithms. Chapter 3 provides the methodology, architecture of the system and the system suggested. Chapter 4 provides the implementation and outcomes of the suggested system. Chapter 5 provides findings for conclusion and suggestions for the next researchers.

CHAPTER 2

BACKGROUND AND RELATED WORKS

2.1 Introduction

Before the implementation of the proposed system, an adequate research has been conducted on the published literature on this subject. In this section, a summary of the mentioned research and investigation will be discussed.

2.2 Literature Review

From security monitoring to QoS measurements, in network management traffic classification (monitoring) has extensive applications. Researchers mostly apply machine learning techniques to flow statistical feature-based classification methods recently.

Network Monitoring can be achieved through the following methods (Ioffe & Szegedy, 2015):

- Port based traffic classification
- Payload based classification (Deep packet inspection)
- Flow features-based (Machine learning and statistical feature)

Before discussing those classifications methods, we have to know what these classes are. Table 2.1 shows network classes.

Table 2.1: Network Classes (Auld et al., 2007; Li & Moore, 2007)

Network Classes	Example Applications
BULK	ftp, ftp_data
DATABASE	postgres, sqlnet oracle, ingres
INTERACTIVE	ssh, klogin, rlogin, telnet
MAIL	imap, pop3, smtp
SERVICES	X11, dns, ident, ldap, ntp
WWW	http
P2P	BitTorrent
ATTACK	DoS, Probe
GAMES	Half-Life
MULTIMEDIA	Windows Media Player, Real Time

2.2.1 Port based classification

This is the oldest way of performing traffic classification. Its assumption is application servers use well known ports for client to initiate communication. Such ports are registered in the IANA list of registered ports (Schneider, 1996):

80: HTTP

22: SSH

20, 21: FTP

25: SMTP

53: DNS

143: IMAP

161, 162: SNMP

It is enough to intercept the TCP/UDP packet header to infer the server-side application. For TCP flows, the SYN (synchronize) packet is enough.

Port based classification is very simple and fast to implement. There is no need to inspect payload but checking the packet headers will be sufficient. It is often used on firewalls and access control lists. Nevertheless, many applications have not ports registered with IANA. Even if they have well known ports, they may use others like they may hide behind port 80. Ports are randomly/dynamically allocated in some cases, and port-based classification fails on NAT (Network Address Translator) and IP (Internet Protocol) tunnels.

TCP Segment and UDP Datagram Header Format (Degermark, 1999) is shown Figure 2.1.

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset / Res	Flags	Window Size	
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Figure 2.1: TCP Segment and UDP Datagram Header Format (Degermark, 1999)

For network traffic classification port-based method is a perfect. This method implicates ports that are firstly registered in IANA. However, this method has failed owing to increase of P2P applications, that use unregistered number (dynamic port numbers) with IANA (Karagiannis, Broido, Brownlee, Claffy, & Faloutsos, 2003; A. W. Moore & Papagiannaki, 2005).

2.2.2 Payload based classification

It is such methods that inspects the TCP or UDP payloads of captured packets looking for:

- Known protocol behaviors (protocol decoding)
- Specific - application data (pattern matching)

They are also called as Deep Packet Inspection (DPI) methods as they inspect the content of the payload (Porter, 2005). Payload based classification can identify many protocols that port-based classification cannot do, and it has higher accuracy rate. In payload-based classification first eight packets will be sufficient for the process. Real-time application is possible as it can classify traffic in short time. As this method inspects payload, it fails to classify encrypted communication. This method generates high processing loads on CPU (Central Processing Unit). Protocol decoding is a very complex operation as it requires deep information of the all protocols. It is used for only given popular protocol types and it is hard to keep such decoders up to date.

Payload Based methods give definitive results in classification. However, many network applications, called encrypted data network applications, use encryption to protect their data from detection so payload-based methods cannot classify them, and fail (Karagiannis et al., 2004).

2.2.3 Flow feature-based classification

Flow feature based classification methods are capable of overcoming the problems of payload-based and port-based classification techniques. It uses statistical properties of the characteristics (features) of each flow to judge the protocol/application type. That is why, those methods are also known as statistical methods or machine learning methods. In general, there are two machine learning methods.

1-Supervized classification: In supervised methods, the machine is trained by using data which is well "labeled." It indicates that with the correct answer some data is tagged. It can be contrasted with learning in the presence of a supervisor or teacher. A supervised learning algorithm draws on labeled training data and helps guess unexpected results. It requires time and technical knowledge from a team of extremely qualified information researchers to successfully build, scale, and deploy precise monitored machine learning models. In addition, data scientists need to reconstruct models to ensure that their insights remain true until their data modifications. Supervised classifications make traffic recognition (one versus all classification) especially attractive. Training on all classes that are expected to be seen is important for multi-classification (Kotsiantis, Zaharakis, & Pintelas, 2007).

2- Unsupervised classification: Unsupervised methods such as clustering may reveal naturally different classes or even new applications. Clusters need to be labeled, for example they can be labeled directly by human. Clusters may not map to applications one to one. One application may dominate multiple clusters, or vice versa. It may be very hard to map back from a single cluster to a source application(Hinton, Sejnowski, & Poggio, 1999).

Preparing data for machine learning initiatives can be accomplished by following the six critical phases below:

1-Data (flows) acquisition (input): This aspect concerns the capture of packets passing through the entire network.

2-Feature extraction: After data capture and (possibly) sampling, both supervised and unsupervised techniques extract flow features. Some works use up to 250 features per flows. Example features:

- Flow duration in seconds

- Data volume per flow

- Number of packets per flow

- TCP port

- Packet inter arrival time (mean, variance, etc.)

- Payload size (mean, variance)

3-Feature selection: Feature Selection is a process of selection a subset of relevant features from all features, which is used to make model building.

4-Training: Training is the most crucial phase, so how well the system performs on the data provided to the system depends on the algorithms used. At this phase, the system is trained with previously provided training data to ensure that it recognizes the patterns in the data.

5-Validation: In this phase of validation, the algorithm used to train the machine is better accurate and efficient.

6-Testing (output): The test data is used in this phase to see how well the machine can predict on the basis of its training new answers.

2.3 Related Works

In this thesis, machine learning and statistical feature method which is flow feature-based classification will be used to implement NMS.

For the authors in (Zhou, Li, & Yang, 2007), in generic, four stages of traffic classification with ML algorithms are shown in Figure 2.2. Network packets that captured online by packet sniffing are first inputs, but these inputs can be prepared offline too. Then packets are classified into vectors according to protocols, source port, source IP address, destination port and destination IP address. In the second stage vector features are calculated which is features extraction. When dataset is huge, for decreasing search space of machine learning algorithm, in order to get a subset of the vector features (decreasing vector dimensions) data sampling can be performed. These features are used in features selection (filtering) stage. In this stage, unnecessary features are filtered, and important and necessary features are selected. Finally, on the last step machine learning algorithm is done (Zhou et al., 2007).

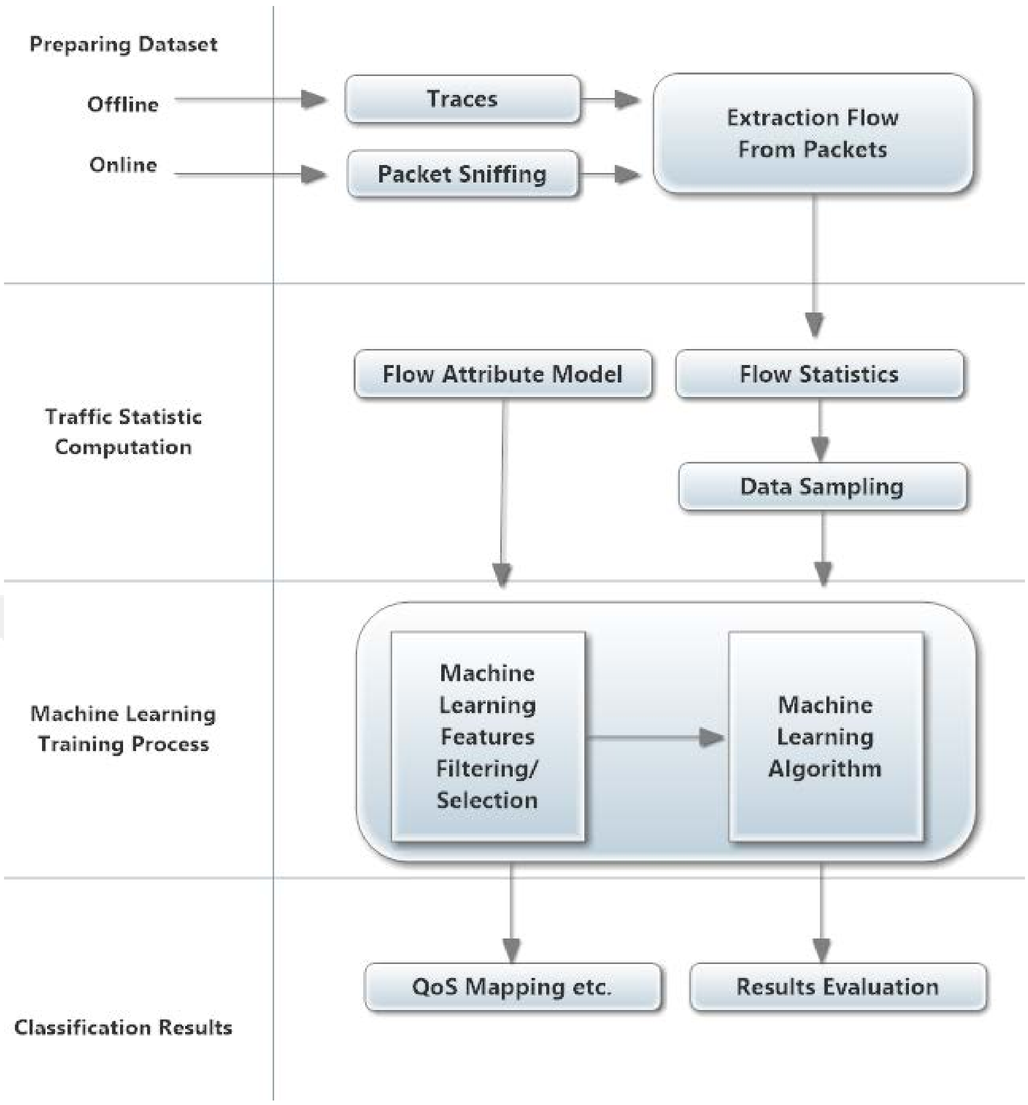

Figure 2.2: Traffic Classification Process by Machine Learning (Zhou et al., 2007)

The most important part of machine learning based network monitoring is algorithm which is used for classifications.

2.4 Support Vector Machine Classification Algorithm

SVM algorithm is similar with Logistic Regression (LR). The both algorithms try to find the best line separating the two classes. The algorithm allows the line to be drawn from the most distant places in two classes of the line(Cortes & Vapnik, 1995). It is a classifier that is non-parametric. SVM may also classify linear and nonlinear information, but typically attempt to classify information as linear. There are numerous of kernels that can be used in SVM classification algorithm like sigmoid, polynomial, linear and radial basis function (RBF) (Scholkopf et al., 1997).

Pros of SVM is:

- With clear margin of separation, it operates really well.
- It is efficient in spaces of high dimensions.
- It is efficient if the number of samples exceeds the number of dimensions.
- It uses in the decision function a subset of training points (called support vectors), so it is also efficient in memory.

Cons of SVM is:

- • If the data set is big, it does not perform well because the necessary training time in this case is greater.
- When the data set has more noise, it does not perform very well because the target classes overlap.
- SVM does not provide direct probability estimates, they are computed using a costly five-fold cross-validation.

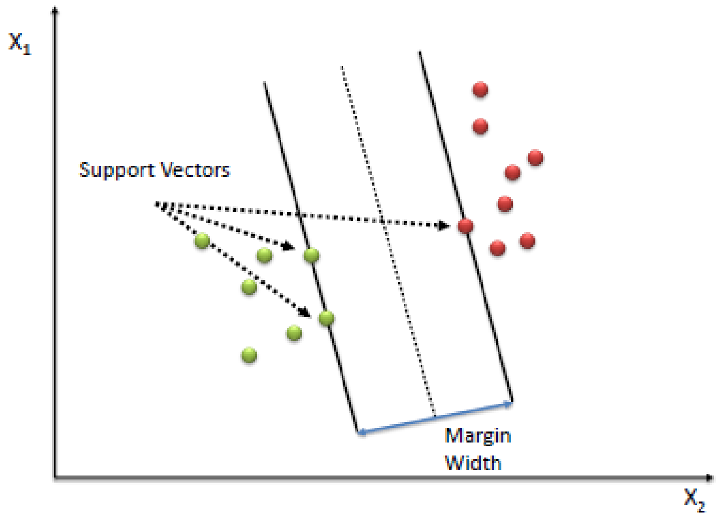

Figure 2.3: Support Vector Machine Classification

- If $\alpha_i > 0$ then the distance of $\underline{x}_i$ from the separating hyperplane is M
  - *Support vectors* - points with associated $\alpha_i > 0$

- The decision function f($\underline{x}$) is computed from support vectors as

$$f(x) = \sum_{i=1}^{n} y_i \alpha_i x^T x_i$$

  => prediction can be fast if α_i are sparse (i.e., most are zero)

- Non-linearly-separable case: can generalize to allow "slack" constraints

- Non-linear SVMs: replace original $\underline{x}$ vector with non-linear functions of $\underline{x}$
  - "kernel trick" : can solve high-d problem without working directly in high d

- Computational speedups: can reduce training time to near- linear
  - e.g Platt's SMO algorithm, Joachim's SVMLight

Figure 2.4: Pseudo code of Support Vector Machine Algorithm

2.5 Decision Tree Classification Algorithm

Decision Tree (DT) Classifier is widely used classification method. It poses a series of definite answered questions about the attributes of the test data. After it receive an answer each time, another question is asked until a decision about the class label of each data is reached (Shalev-Shwartz & Ben-David, 2014).

Pros of DT is:

- It is easy to understand and interpret, perfect for visual representation.
- It can work with numerical and categorical features.
- It requires little data preprocessing.
- It is fast for inference.

Cons of DT is:

- It tends to over fit.

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

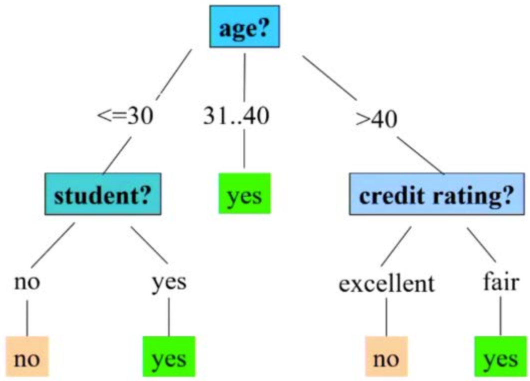

Figure 2.5: A Decision Tree Classification Algorithm Example

2.6 Logistic Regression Classification Algorithm

LR classifier assigns a discrete set of classes to observations. By using the logistic sigmoid function, $p(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$, LR transforms its output into a probability value that can be mapped to two or more separate classes (Hosmer & Lemeshow, 2000).

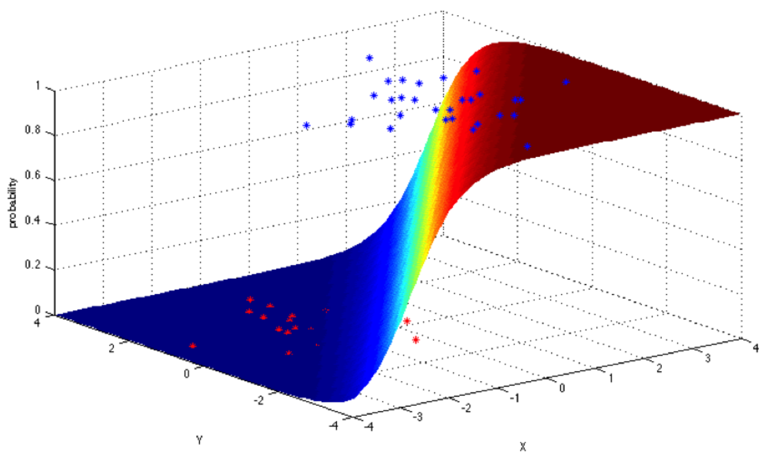

Figure 2.6: A Logistic Regression Classification Algorithm Example

Pros of LR is:

- It is very efficient and highly interpretable.
- LR doesn't require too many computational resources.
- It doesn't require input features to be scaled.

Cons of LR is:

- It doesn't perform well when feature space is too large
- It doesn't handle large number of categorical features/variables well
- Non-linear problems cannot be solved by LR so it needs transformations for non-linear features.
- It relies on entire data.
- LR can only predict a categorical outcome.
- It is vulnerable to overfitting.

2.7 Deep Learning

Deep learning (DL) is primarily neural networks and is usually tailored to machine learning. Most deep learning methods use neural network architectures. That is why they are often referred to as deep neural networks. It teaches computers to do what comes naturally to humans and it learns from example. DL classifier contains 3 type of layers: input, output and hidden. Each layer contains at least one interconnected node. In data set, classifier detects complex structure, and it changes its internal parameters to calculate the prior layers (LeCun, Bengio, & Hinton, 2015; Schmidhuber, 2015).

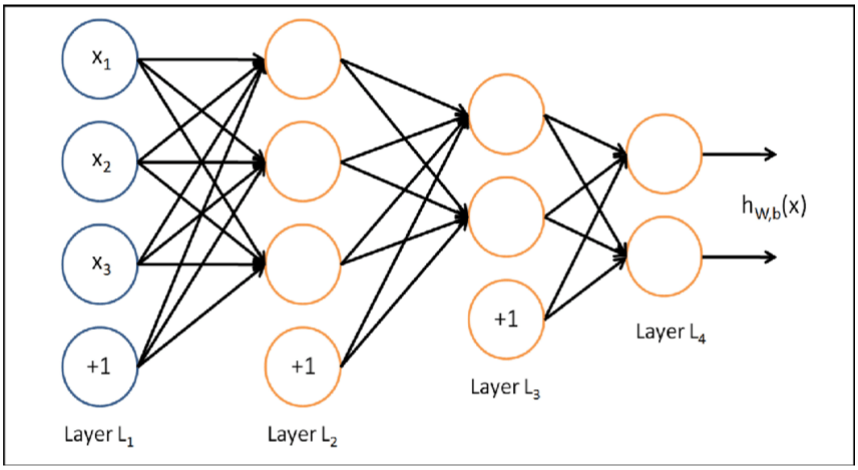

Figure 2.7: Neural Network Illustration

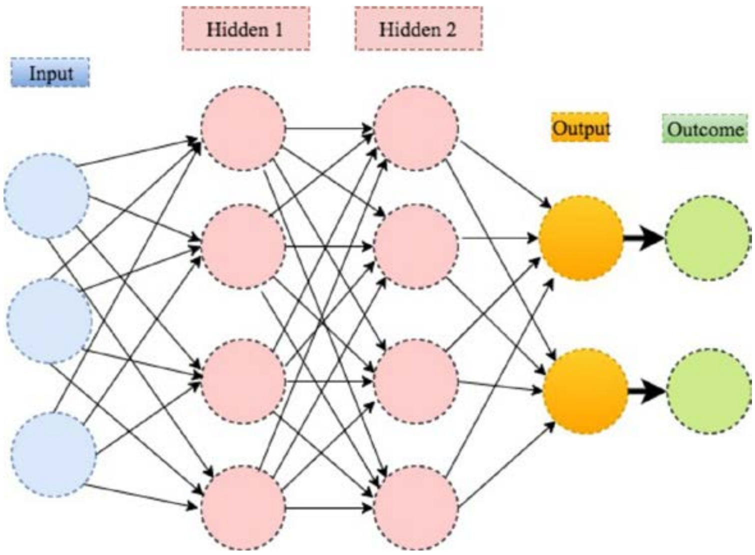

Figure 2.8: Deep Learning Illustration

Pros of DL is:

- DL has very high performance.
- DL lowers the need for engineering features.
- DL is an architecture that can comparatively readily tailored to fresh issues.

Cons of DL is:

- Large amounts of data are required
- DL is extremely costly to train computationally.
- It has little to do with a powerful theoretical basis.

2.8 Gaussian Naïve Bayes (GNB)

GNB Classifier is modeled on the Bayesian Theorem (Webb, Boughton, & Wang, 2005). Bayesian theorem allows us to use the naïve independence assumption to indicate the conditional probability as follows:

$$P(y\backslash X) = \frac{P(y)P(X\backslash y)}{P(X)} = \frac{P(y)\prod_{i=1}^{n} P(X_i\backslash y)}{P(X)}$$

The following rule is used to classify the sample since $P(X)$ is continuous for a specified example:

$$\breve{y} = \arg max_y \; P(y) \prod_{i=1}^{n} P(X_i \backslash y)$$

Estimation of maximum a posteriori (MAP) is generally utilized to estimate the parameters in the naïve Bayes model, inclusive of *P*(*y*) and *P*(*xi*|*y*); the preceding is the frequency of samples in the training set with class *y*. In addition, Gaussian naïve Bayes uses the classification by assuming the probability of Gaussian characteristics:

$$P(X_i \backslash y) = \frac{1}{\sqrt{2\pi\sigma_y{}^2}} \exp(-\frac{(x_i - \mu_y)^2}{2\pi\sigma_y{}^2})$$

where the maximum probability is estimated for the parameters σy and μy. Because of its simplicity and extreme speed compared to more advanced methods (Lou et al., 2014). Illustration of GNB classifier is show in Figure 2.9.

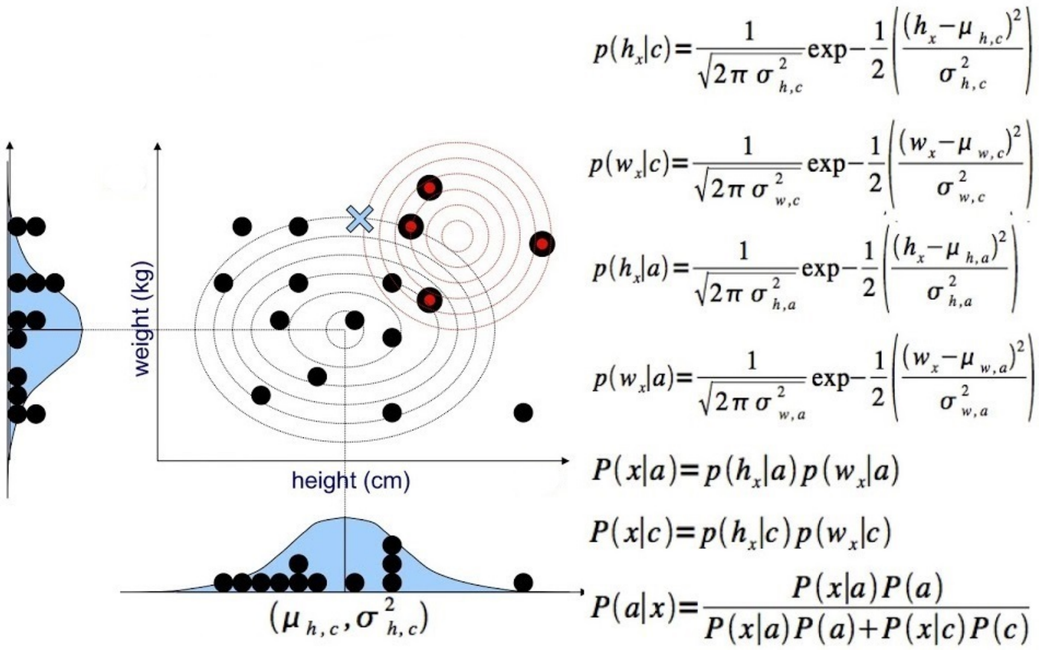

Figure 2.9: Illustration of GNB Classification Algorithm

Pros of GNB is:

- GNB is simple and quick to predict class of test data set.
- GNB operates well in multi class prediction.

Cons of GNB is:

- If categorical variable has a category that was not observed in training data set, then GNB cannot make a prediction.
- It is bad estimator.

2.9 Principal Component Analysis (PCA) Algorithm

PCA is reducing the dimensionality of a data set consisting of many variables correlated with each other while keeping the variation present in the dataset, up to the maximum extent.

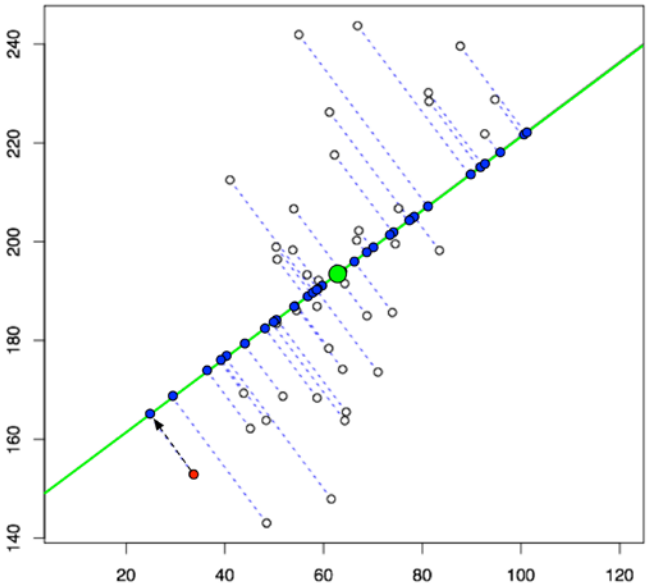

Figure 2.10: A PCA Algorithm Example

The proposed system will use PCA algorithm to reduce features.

2.10 KDD (Knowledge Discovery Data mining) CUP 99 Data set Description

KDD CUP99 is based on data captured by Stolfo et al in the DARPA'98 IDS evaluation program. Since 1999, it has used anomaly detection techniques for assessment. It has been using widely popular. DARPA'98 is about 4 gigabytes data. This data is 7 weeks of network traffic and it is compressed raw “binary” TCP dump. There are 5 million connection records in this data. As a row, every connection of this data called a vector. Each vector is about 100 bytes. Test data is about fortnight of network traffic and includes about 2 million vectors. Training data includes about 5 million vectors. Each vector includes 41 features.

CHAPTER 3

METHODOLOGY

3.1 Machine Learning Workflow

In this chapter, the theoretical description of the system will be presented following the Machine Learning Workflow methodology. PCA algorithm will be used in Data pre-processing step and DT algorithm will be used for classification and regression predictive problems in the Train Model step and Test Data step.

Machine Learning Workflow consists of five main stages of a project and it defines tasks in every stages and relationships between them. These steps are illustrated below:

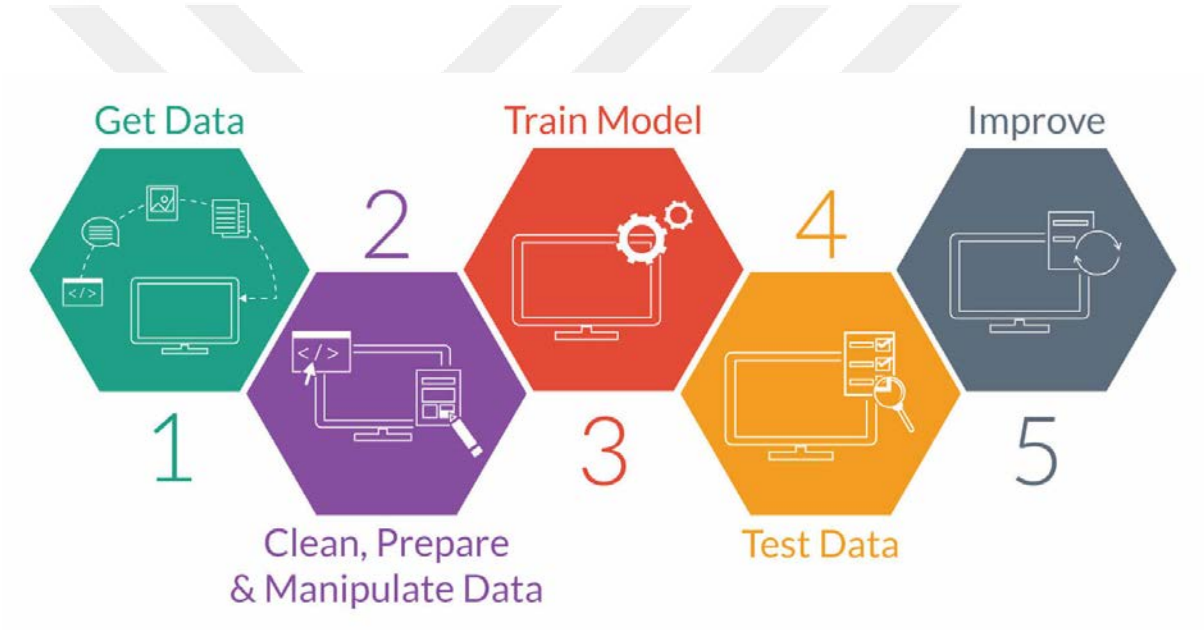

Figure 3.1: The Machine Learning Workflow

When we construct NMS, we will apply the steps of the machine learning workflow in our machine learning system.

3.1.1 Gathering Data

In the first stage of NMS, packets that passed through the entire network are being grabbed and any packet that targeted to any node of the network can be grabbed by Wireshark network application which contains packet decoder. Wireshark is a free and open-source packet analyzer. Wireshark can convert network packets into other file types. In addition to capturing all features, Wireshark also catches data and time fields and display them in this stage. Typical packets information gathered by Wireshark illustrated in Figure 3.2. All packets that have been captured will be processed in the next stage for analyzing.

```
"No.","Time","Source","Destination","Protocol","Length","Info"
"1","0.000000","Dell_3c:38:d1","Broadcast","ARP","60","Who has 192.168.105.137? Tell 192.168.104.112"
"2","0.022264","192.168.104.200","239.255.255.250","SSDP","382","NOTIFY * HTTP/1.1 "
"3","0.151932","192.168.104.200","239.255.255.250","SSDP","431","NOTIFY * HTTP/1.1 "
"4","0.282104","192.168.104.200","239.255.255.250","SSDP","447","NOTIFY * HTTP/1.1 "
"5","0.292393","192.168.104.200","239.255.255.250","UDP","637","34455  >  3702 Len=595"
"6","0.295455","192.168.104.113","192.168.105.255","NBNS","92","Name query NB DIFFERENTIA.RU<00>"
"7","0.375827","Dell_86:2e:8d","Broadcast","ARP","60","Who has 192.168.104.36? Tell 192.168.104.157"
"8","0.420669","192.168.104.159","239.255.255.250","UDP","637","34455  >  3702 Len=595"
"9","0.620017","Dell_3c:38:d1","Broadcast","ARP","60","Who has 192.168.105.137? Tell 192.168.104.112"
```

Figure 3.2: Captured and converted network packets by Wireshark

As shown in the Figure 3.3, the client can capture Wireshark packets and convert them to a csv file containing vectors of features in each row of the file shown in Figure 3.6.

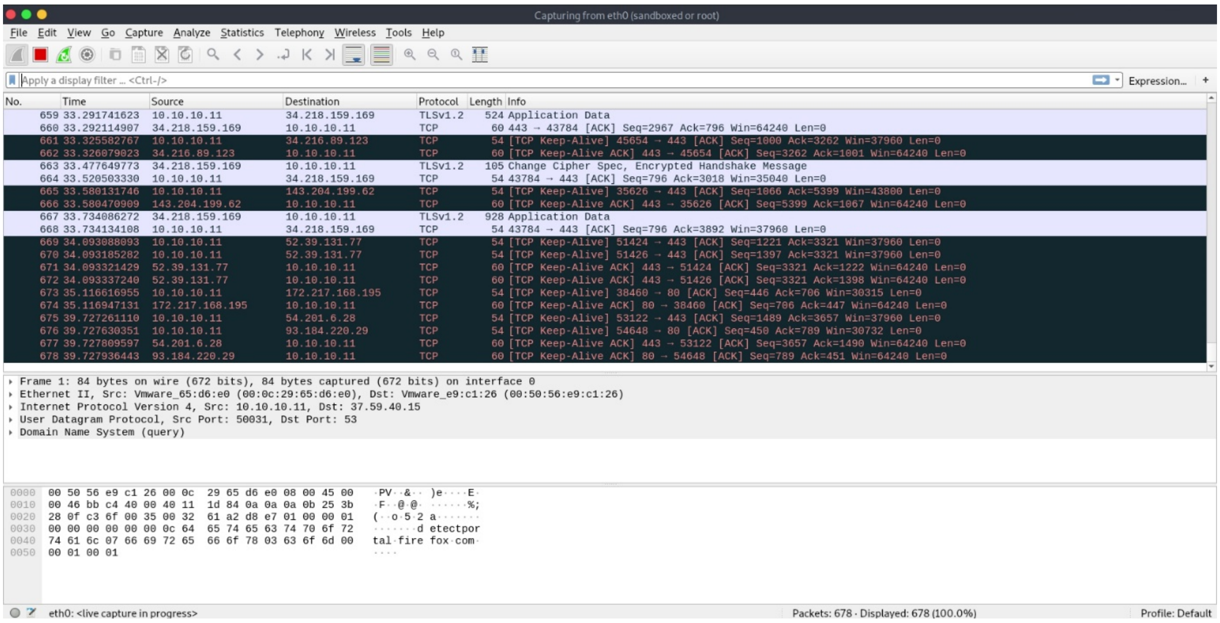

Figure 3.3: Wireshark is capturing the packets in real time

As shown in Figure 3.4, the packet decoder takes packets from the network interface via the Npcap library, and identifies which protocol is in use for a grabbed packet. Npcap is used to grab packets from the network, it is library, which works under windows operating system. Data acquisition is also referred to as data acquisition using the Npcap library. Wireshark will be used in the NMS to capture packets using the Npcap library and for processing step their format will be converted.

The packets stored in the dataset will then be sent for pre-processing (Wolpert & Macready, 1997).

Table 3.1: Illustration of OSI Layers

Layer	Function	Example
Application – Layer 7	Services that are used with end user applications.	SMTP
Presentation – Layer 6	Formats the data so that it can be viewed by the user. Encrypt and decrypt.	JPG, GIF, HTTPS, SSL, TLS
Session –Layer 5	Establishes/ends connections between two hosts.	NetBIOS, PPTP
Transport – Layer 4	Responsible for the transport protocol end error handling	TCP, UNDP
Network – Layer 3	Reads the IP address from the packet.	Router, Layer 3 Switches
Data Link – Layer 2	Reads the MAC address from the data packet.	Switches
Physical – Layer 1	Send data on the physical wire.	Hubs, NICS, Cable

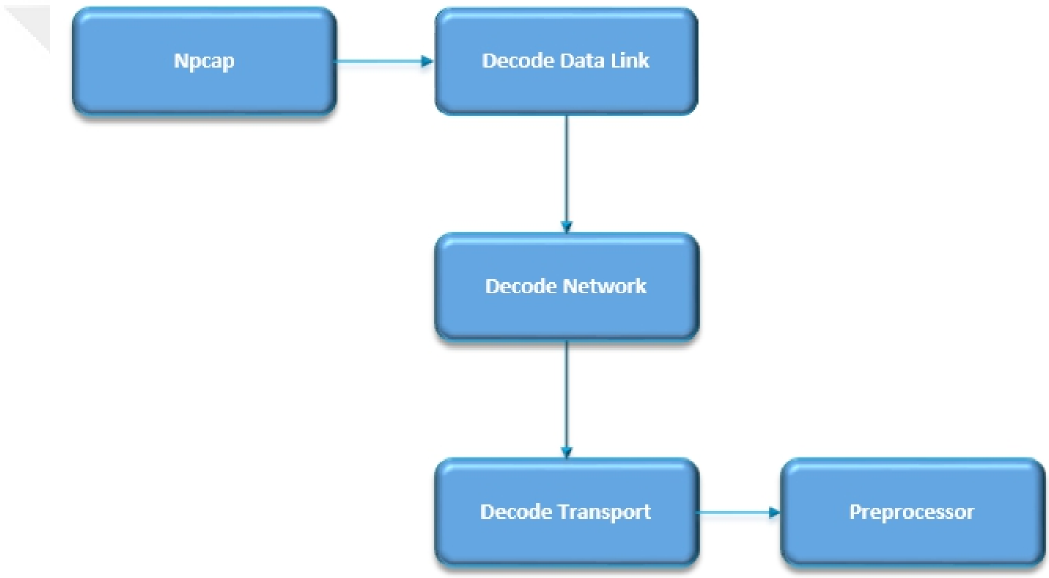

Figure 3.4: Phases for package decoding (Wolpert & Macready, 1997)

3.1.2 Data Preparation

Figure 3.5 demonstrates the pre-processing steps of the data before the Train model, which includes the training process. Generally, pre-processing data is necessary for all tasks of machine learning. Pre-processing of data is based on extracting information from header and load for the packages. Then new statistical features will be created from the header and the load. Pre-processing generally consists of dataset creation, data cleaning, integration, and feature construction, feature selection, reduction, and normalization. The most related steps for NMS are briefly explained below (Tavallaee, Bagheri, Lu, & Ghorbani, 2009).

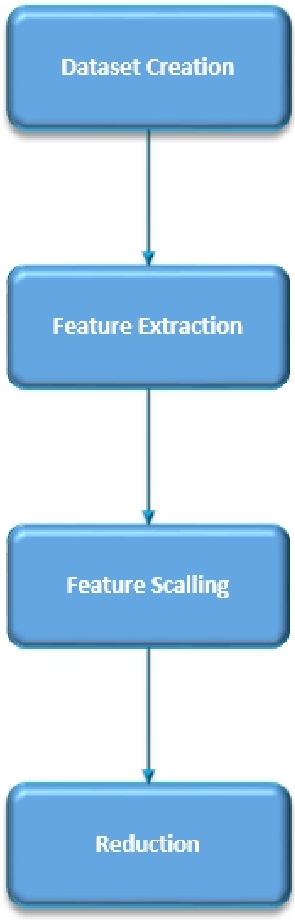

Figure 3.5: The phase of pre-processing data (Wolpert & Macready, 1997)

***Dataset Creation*:** For classification and prediction, it is representative network traffic. The KDD dataset that is used for NMS, were composed from some normal network logs through weeks. The dataset is compressed raw binary TCP dump. There are 5 million connection records in this data. As a row, every connection of this data called a vector. Each vector is about 100 bytes. Each vector contains 41 features.

The table below shows the part of the data set of the KDD Cup utilized in the suggested NMS.

Table 3.2: NMS Training Dataset

BULK	ftp	373
	ftp_data	3.798
	tftp_u	1
INTERACTIVE	shell	1
	ssh	1
	telnet	219
MAIL	pop_3	79
	smtp	9.598
MEDIA	X11	9
OTHER	auth	220
	domain	3
	domain_u	5.862
	eco_i	389
	finger	468
	other	5.632
	private	7.366
	red_i	1
	tim_i	2
	urh_i	14
	urp_i	537
SERVICE	ecr_i	345
	ntp_u	380
	time	52
WWW	http	61.886
	IRC	42
Grand Total		97.278

Table 3.3: NMS Test Dataset

BULK	ftp	2.381
	ftp_data	110.020
INTERACTIVE	shell	2
	ssh	6
	telnet	1.313
MAIL	imap4	3
	pop_3	531
	smtp	55.095
MEDIA	X11	60
OTHER	auth	1.905
	domain	2.202
	domain_u	1.226.133
	eco_i	2.587
	finger	2.906
	other	3.193
	private	12
	urh_i	156
	urp_i	2.905
SERVICE	ecr_i	2.099
	ntp_u	3.061
	time	200
WWW	http	4.526.600
	IRC	236
Grand Total		5.943.606


```
1,tcp,smtp,SF,2383,332,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,0.00,0.00,0.00,0.00,1.00,0.00,1.00,46,120,0.83,0.09,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,1030,327,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,2,4,0.00,0.00,0.00,0.00,1.00,0.00,0.75,47,121,0.83,0.09,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,udp,domain_u,SF,30,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,4,0.00,0.00,0.00,0.00,1.00,0.00,0.75,48,43,0.15,0.08,0.15,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,606,329,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,49,122,0.82,0.08,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,1153,329,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,50,123,0.82,0.08,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,926,328,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,51,124,0.82,0.08,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,1309,326,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,52,125,0.83,0.08,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,udp,domain_u,SF,30,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,53,44,0.15,0.08,0.15,0.05,0.00,0.00,0.00,0.00,normal.
0,udp,domain_u,SF,30,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,3,0.00,0.00,0.00,0.00,1.00,0.00,1.00,54,45,0.17,0.07,0.17,0.04,0.00,0.00,0.00,0.00,normal.
0,udp,domain_u,SF,30,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,0.00,0.00,0.00,0.00,1.00,0.00,1.00,55,46,0.18,0.07,0.18,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,1491,327,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,1,0.00,0.00,0.00,0.00,0.50,1.00,0.00,56,126,0.79,0.07,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,807,278,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,57,127,0.79,0.07,0.02,0.02,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,1014,332,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,58,128,0.79,0.07,0.02,0.02,0.00,0.00,0.00,0.00,normal.
```


Figure 3.6: Sample vectors of NMS's KDD dataset

***Features Extraction*:** Classification of packets relies on the network connection feature values. Extraction of features requires the captured network packets as an input and extracts features from these packets as output. In port-based classification, basic features are extracted from the header of packets (protocol type, service, flag etc.). In payload classification, content features are extracted from payload of packets (logged in, etc.). Statistical features (count, srv_count, etc.) manually are computed.

***Feature Scaling*:** This stage is a method for standardizing the range of independent variables or dataset features. Since the range of values of raw data varies widely, objective functions will not work properly without normalization in some machine learning algorithms. There are two types of feature scaling (Ioffe & Szegedy, 2015):

Standardization: Data standardization is the process of rescaling one or more features so that features have 0 mean value and a standard deviation of 1. Standardization assumes that data has a Gaussian distribution. This does not strictly have to be true, but the technique is more effective if features distribution is Gaussian.

Normalization: Normalization is the process of rescaling one or more features to the range of 0 to 1. This means that the largest value for each feature is 1 and the smallest value is 0.

After feature scaling, vectors are suitable as input to machine learning algorithms.

Reduction: This is utilized to reduce the dimensions (count) of features by dismissing any excessive or irrelevant features.

3.1.3 Train Model (Classification)

Classification is a method by which classifier specifies which group belongs to a particular observation, such as when biologists categorize crops, animals and other life forms into separate taxonomies. It is one of the main uses of information science and machine learning.

Then algorithms classify the train data and train data train the system. In the final stage, the trained system estimates vectors to determine if the data is normal or not.

3.1.4 Test Data (Prediction)

The machine learning value is realized in this step. The trained model is used in this phase to forecast the result and it labels packets with the class name it belongs to. This step is the phase of deciding which class the package belongs to.

3.1.5 Improve

It covers the evaluation of the results from the technical point of view according to the test values, arranging and sending of the observing and support model and arranging process. Selecting the most efficient model by looking at the test results enters this stage. Improve step of methodology will be discussed in chapter five which is the last chapter of this thesis.

3.2 System Architecture

The NMS includes data gathering, data pre-processing, classification, prediction and response stages. In data gathering stage, Wireshark captures data from network and then data is used in the classification and prediction stages to train and test the NMS. In NMS, KDD CUP99 provides train data and pre-processing data controls the data to assure an effective configuration of the classification system. DT classification algorithm is used by PCA algorithm and Wireshark network monitoring application to build the suggested NMS and classify network traffic in online and offline mode. The last NMS phase response shows important data and tells the network administrator to take appropriate action.

In Figure 3.7 a general NMS architecture is illustrated. It shows a general architecture of an NMS.

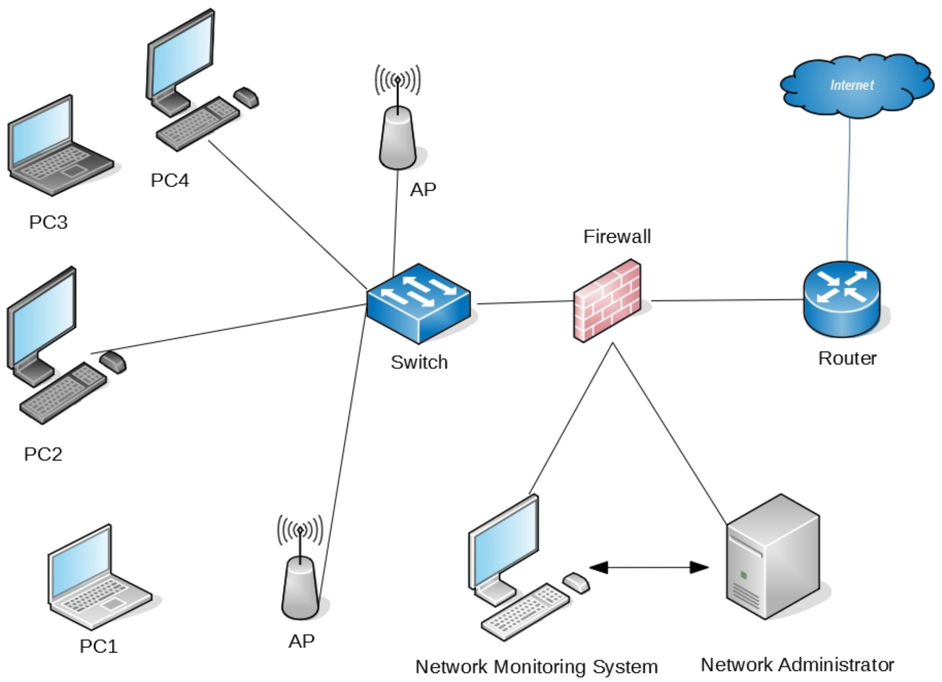

Figure 3.7: A General architecture of an NMS

The most considerable part of the NMS is to classify network packets by service types, then dispatch a copy of the record to the network administrator. As a result, the network administrator takes proper action by updating the monitoring systems on the network.

3.3 The Proposed System

NMS has two phases as follows:

I. Training.

II. Testing.

Training phase which is shown in Figure 3.8, consists of three steps and those steps are listed below:

- Input dataset.
- Train the system by dataset.
- DT algorithm classifies and PCA algorithm deducts dimensions, and NMS classifies the packets by classes that determined in feature extraction phase. Output of this phase is trained system.

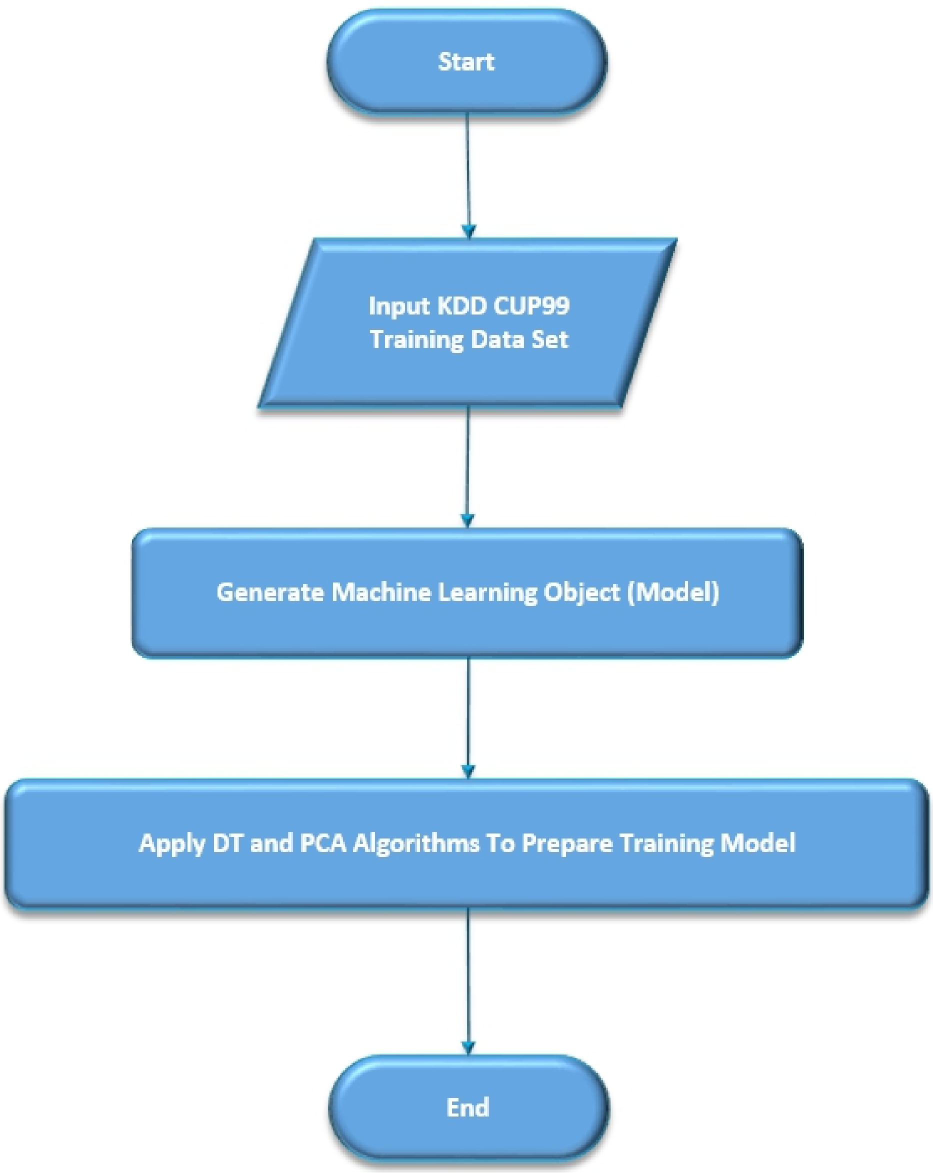

Figure 3.8: Training Phase's Block Diagram

In Figure 3.9 the testing stage is illustrated. It consists of 3 steps and those steps can be described as bellow:

- Input trained system.
- Learning machine (trained system) classifies network traffic (testing file in offline mode).
- Generate monitoring report as output. It is the last phase in the suggested system NMS. Important information is displayed, and system informs the network administrator for further actions

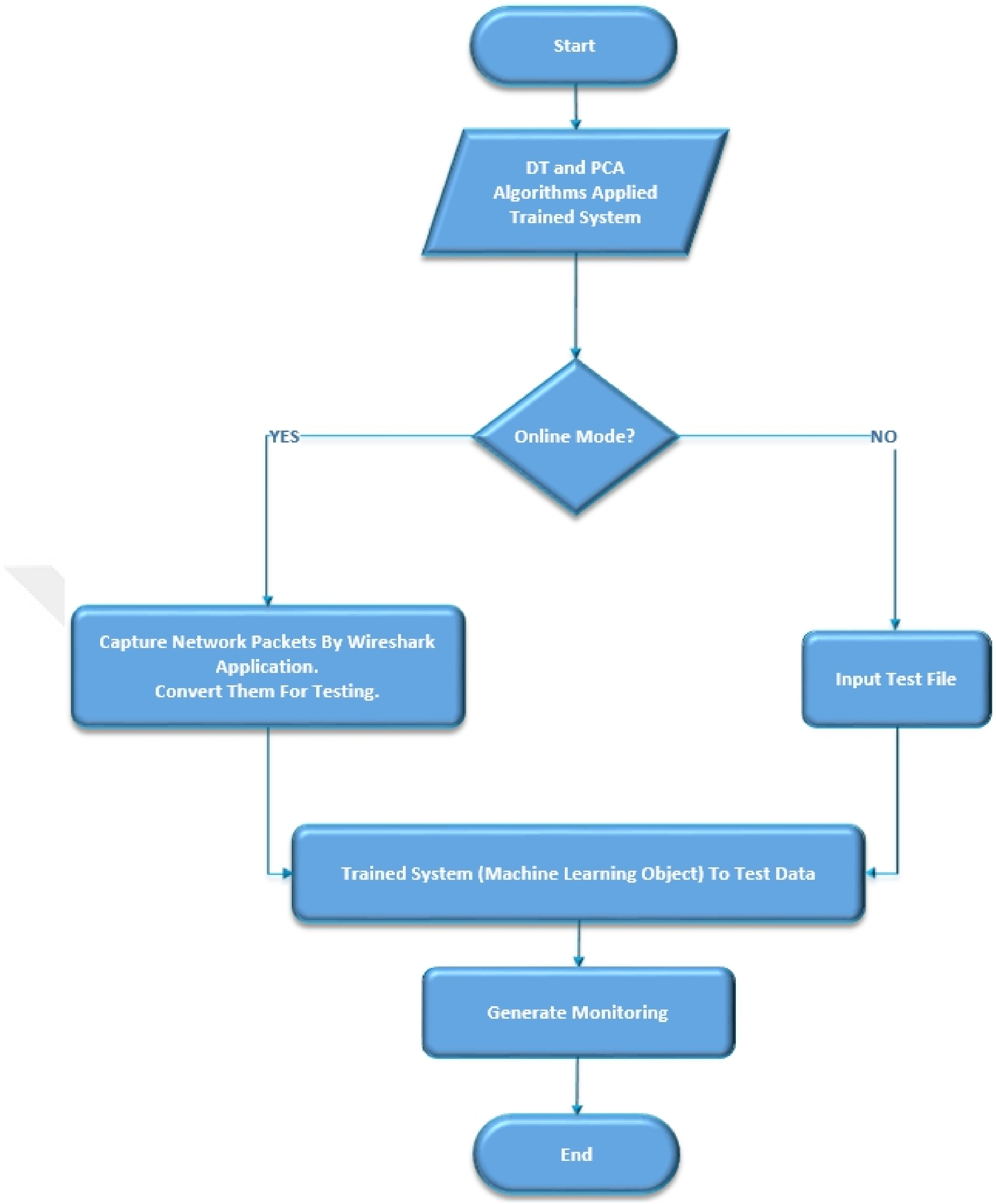

Figure 3.9: Testing Phase’s Block Diagram

CHAPTER 4

NMS SYSTEM IMPLEMENTATION AND RESULTS

4.1 Introduction

The application of the suggested NMS will be presented in this section by using the DT and PCA algorithms. Then we will compare DT algorithm of the NMS with other algorithms. This chapter provides a comparison of the test results of NMS with the different codes given in the Appendix. NMS is designed as it is in a LAN network with a WAN connection. It is programmed in Python language and implementation of the code is highly suitable for the most of networks types.

4.2 System Architecture

In Figure 4.1 the suggested NMS system is illustrated. It is comprised of the below components:

- Router which will be used for routing network packages.
- Switch for receiving, processing and forwarding the packets.
- PC, which runs NMS.
- Network Administrator is to monitor the network.
- LAN with five clients, one firewall, one switch, one router and two access point.

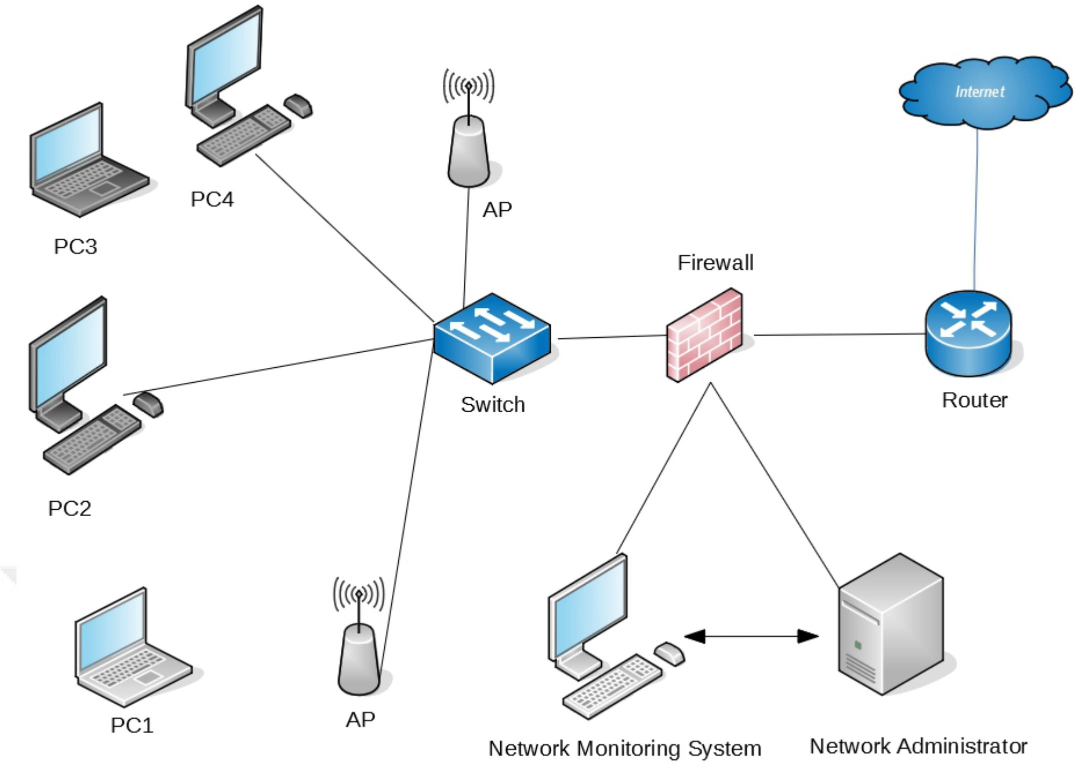

Figure 4.1: NMS system architecture

4.3 Performance Metrics

4.3.1 Confusion Matrix (CM)

CM is a measure of efficiency. It is utilized for problems where there can be at least two output classes. It is shown as a table with four cells. Columns are actual values and rows are predicted values, and it is showed in Table 4.1.

True Positive (TP): Actual value is positive, and prediction is positive

True Negative (TN): Actual value is positive, and prediction is negative

False Positive (FP): Actual value is negative, and prediction is positive

False Negative (FN): Actual value is negative, and prediction is negative

Table 4.1: Confusion Matrix (CM) (Tavallaee et al., 2009)

		PREDICTED VALUES	
		Positive	**Negative**
ACTUAL VALUES	**Positive**	TP	FN
	Negative	FP	TN

A confusion matrix for binary class problem is show in Table 4.1. and a confusion matrix for multiclass problem is shown in Table 4.2.

Table 4.2: An Example CM for NMS (Stallings, 2003)

		PREDICTED VALUES						
		WWW	**MAIL**	**OTHER**	**INTERACTIVE**	**BULK**	**SERVICE**	**MEDIA**
ACTUAL VALUES	**WWW**	475767	1036	59	22	35	10	0
	MAIL	804	49996	204	110	119	3	0
	OTHER	133	205	38163	66	111	1312	9
	INTERACTIVE	67	250	146	691	34	0	1
	BULK	24	116	107	22	21423	1	0
	SERVICE	0	1	939	0	3	3780	0
	MEDIA	2	1	5	12	3	0	6

Standard metrics are shown below for evaluating network monitoring. The most commonly used evaluation metrics are the detection rate (DR) and the accuracy rate. As shown in Equation (4.1), DR is calculated as the ratio of the number of correctly classified vectors to the total number of class vectors and as shown in as in Equation (4.2), Accuracy is the ratio of number of correct predictions to the total number of input samples (Tavallaee et al., 2009).

Sensitivity-Detection Rate (DR) or True Positive Rate (TPR): It is a ratio of the total number of positive examples that are correctly classified divide into the total number of positive examples. High recall shows that the class is identified correctly (Tavallaee et al., 2009).

$$DR = TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{4.1}$$

Accuracy: It is the ratio of the correctly labeled examples to the whole pool of example (Tavallaee et al., 2009).

$$Accuracy = \frac{TP + TN}{Total} \tag{4.2}$$

4.4 Experiments and Results

There will be four scenarios that contains experiments to classify network traffic over KDDCUP99 data set by using DT, LG, SVM, DL and GNB algorithms in NMS in this section.

Scenario 1: 5 experiments were performed on KDD CUP99 dataset by using SVM, DT, LR, GNB and DL classification algorithms individually. These experiments' requirements described as below:

Experiment 1: 97,278 rows of KDD CUP99 dataset are chosen for the training and 595,798 rows for testing, however all forty-one features are chosen.

Experiment 2: 97,278 rows of KDD CUP99 dataset are chosen for the training and 595,798 rows for testing, however only thirty features are chosen.

Experiment 3: 97,278 rows of KDD CUP99 dataset are chosen for the training and 595,798 rows for testing, however only twenty features are chosen.

Experiment 4: 97,278 rows of KDD CUP99 dataset are chosen for the training and 595,798 rows for testing, however only ten features are chosen.

Experiment 5: 97,278 rows of KDD CUP99 dataset are chosen for the training and 595,798 rows for testing, however only five features are chosen.

Scenario 2: To test scalability of NMS, there will tree different size of KDD data set.

Scenario 3: In this scenario standard non-scalar algorithms were applied and compared to each other to see the outcome of feature scaling on the classification.

Scenario 4: To avoid impact of unbalanced data on classification.

Scenario 1:

The 1st experimental outcome showed that the DT classifier reached the largest DR rate of 79.8813% compared to other classifiers. The maximum ACC rate is 98.9923 % that reached by DL classifier. The maximum value of DR rate for network classes was reached by DT classifier. Although the training time is long, DL classifier reached by a high DR rate. The minimum time taken to train is 0.156394 seconds that reached by GNB classifier and minimum time taken to test is 0.12333 seconds achieved by DT classifier. Moreover, the minimum memory usage is 0.741154 GB that reached by DT classifier.

Table 4.3: First Scenario of 1st Experiment

	EXPERIMETNS				
	SVM	**DT**	**LR**	**DL**	**GNB**
DR (%)	51.1479	79.8813	56.0539	78.8496	51.4787
ACC (%)	96.2793	97.7486	96.1568	98.9923	84.0907
DR for WWW (%)	99.7406	99.8135	99.6165	99.6575	95.4771
DR for MAIL (%)	82.7367	83.6521	84.9207	90.6064	49.8243
DR for OTHER (%)	96.7149	94.6473	92.5823	96.3380	26.0731
DR for INTERACTIVE (%)	1.1774	63.9192	11.5222	0	46.7619
DR for BULK (%)	77.6656	96.5887	72.0186	97.4803	20.9560
DR for SERVICE (%)	0	82.6169	31.7171	68.0890	97.1204
DR for MEDIA (%)	0	37.9310	0	99.7761	24.1379
MEAN OF CROSS V. (%)	96.0249	97.9379	95.7114	98.9900	75.5659
TRAINING TIME(SECOND)	45.529160	5.581684	7.594568	122.242404	0.156394
TESTING TIME(SECOND)	335.489777	0.12333	0.102367	32.621607	2.362087
MEMORY(GB)	0.743065	0.741154	0.813011	0.977589	0.758854

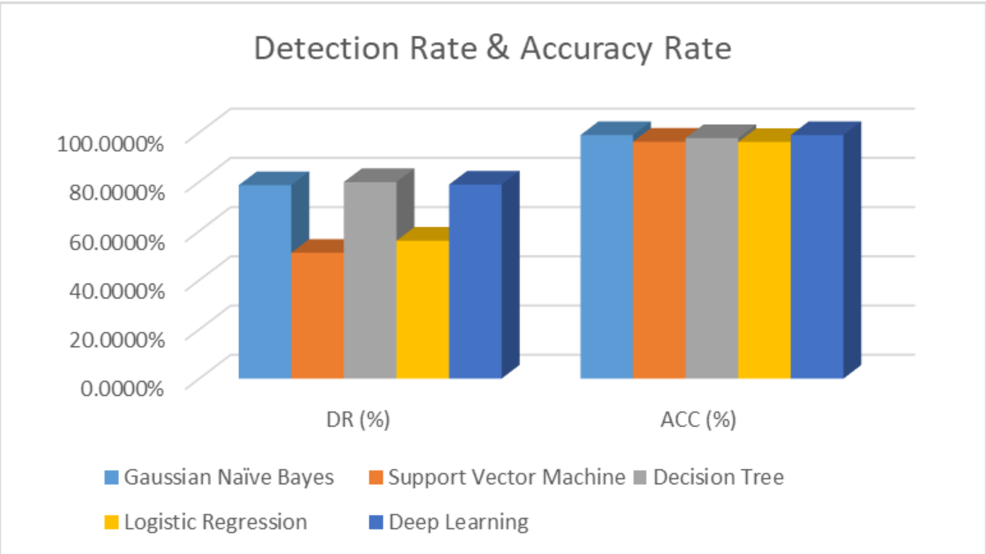

Figure 4.2: DR & ACC Comparison for the 1st Experiment

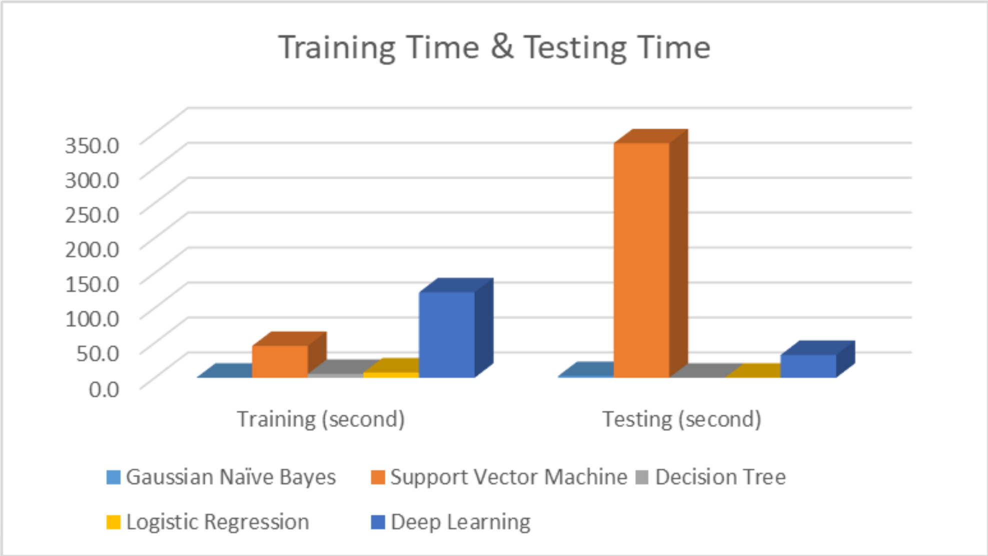

Figure 4.3: Training and Testing Time Comparison for the 1st Experiment

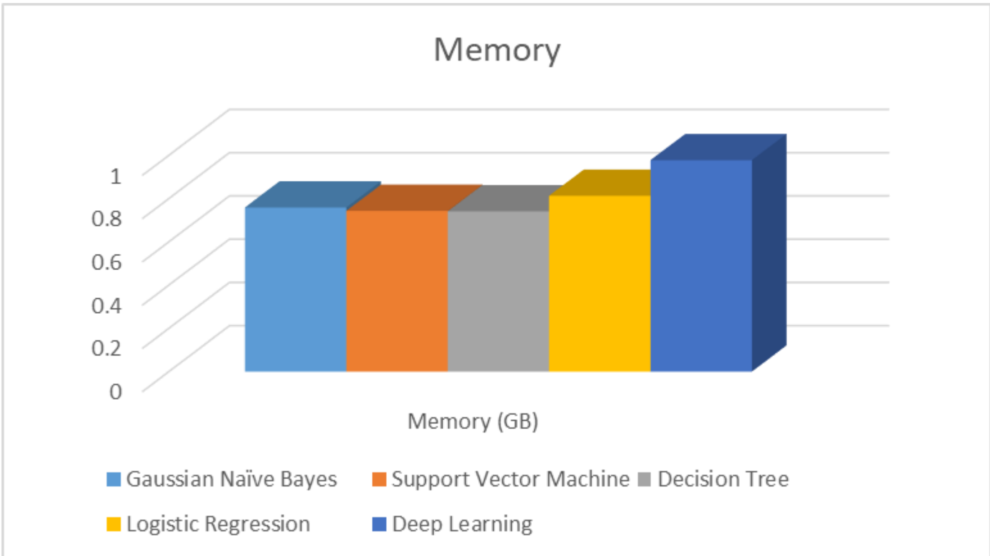

Figure 4.4: Memory Consume Comparison for the 1st Experiment

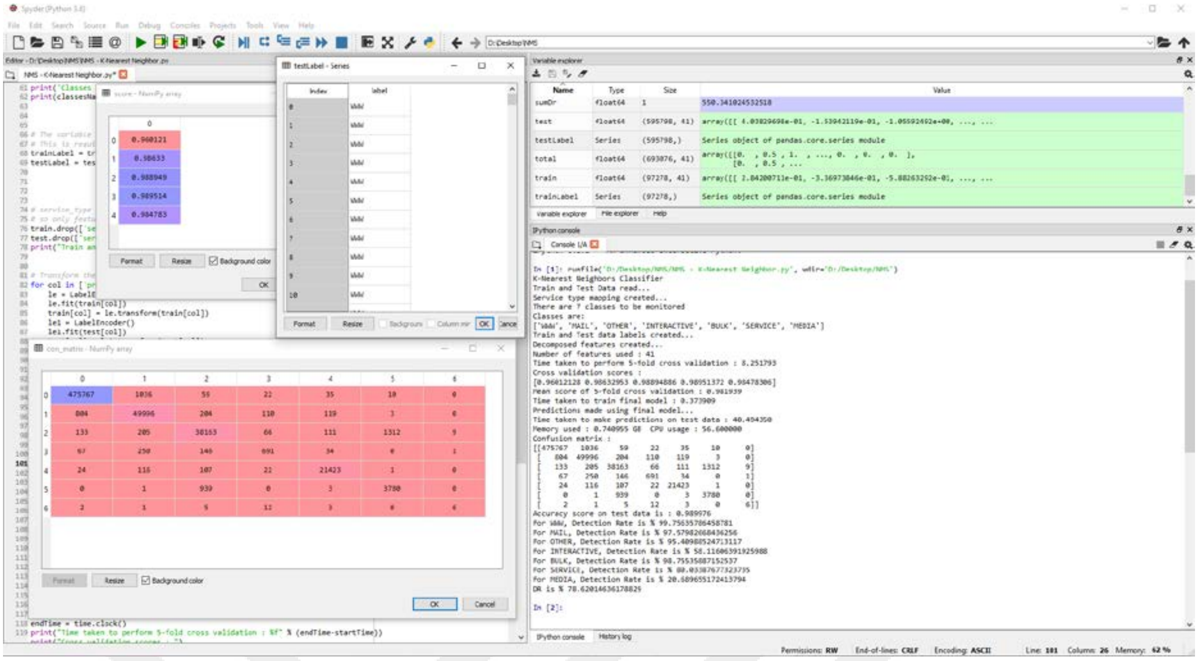

Figure 4.5: Screenshot of 1st Experiment

```
Decision Tree Classifier
Train and Test Data read...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Service type mapping created...
Train and Test data labels created...
Decomposed features created...
Number of features used : 41
Time taken to perform 5-fold cross validation : 20.761818
Cross validation scores :
[0.96099491 0.98406825 0.984991   0.98678935 0.98005346]
Mean score of  5-fold cross validation : 0.979379
Time taken to train final model : 5.637815
Predictions made using final model...
Time taken to make predictions on test data : 0.128305
Memory used : 0.983337 GB  CPU usage : 19.200000
Confusion matrix :
[[476040    654     45     70    110      8      2]
 [  6332  42860    390     63   1589      0      2]
 [   117    197  37858     30    362   1401     34]
 [    74    172    110    760     67      0      6]
 [    63    305    343     21  20953      7      1]
 [     0      1    810      0     10   3902      0]
 [     1      1      3      8      5      0     11]]
Accuracy score on test data is : 0.977486
For WWW Class, Detection Rate is % 98.63517789100071
For MAIL Class, Detection Rate is % 96.99026929169496
For OTHER Class, Detection Rate is % 95.70009353118127
For INTERACTIVE Class, Detection Rate is % 79.83193277310924
For BULK Class, Detection Rate is % 90.72133702805681
For SERVICE Class, Detection Rate is % 73.37344866491162
For MEDIA Class, Detection Rate is % 19.642857142857142
```

Figure 4.6: Outputs for the 1st Experiment

The 2nd result showed that DT classifier reached a DR rate of 75.8523 % as maximum value compared with other classifiers. The maximum ACC rate is 98.8568 % that reached by DL classifier. The maximum DR rate for network classes was reached by DT classifier. Although the training time is long, DL classifier reached a high DR rate. The minimum time taken to train is 0.13955 seconds that reached by GNB classifier and minimum time taken to test is 0.088197 seconds reached by LR classifier. Moreover, the minimum memory usage is 0.685406 GB that reached by Support Vector Machine classifier.

Table 4.4: 2nd Experiment of First Scenario

	EXPERIMETNS				
	SVM	DT	LR	DL	GNB
DR (%)	51.6828	75.8523	56.0486	74.3756	55.0105
ACC (%)	96.3691	97.5658	96.1539	98.8568	86.6332
DR for WWW (%)	99.7381	99.8775	996160	98.5109	95.1472
DR for MAIL (%)	83.6306	80.7244	84.8973	60.0671	51.4228
DR for OTHER (%)	96.7249	94.8773	92.5823	96.6456	71.4717
DR for INTERACTIVE (%)	2.8595	57.6955	11.5222	0	38.2674
DR for BULK (%)	77.7255	96.7685	72.0047	97.1703	11.9946
DR for SERVICE (%)	1.1009	83.7814	31.7171	68.5300	92.6318
DR for MEDIA (%)	0	17.2413	0	99.7057	24.1379
MEAN OF CROSS V. (%)	96.1174	97.5987	95.7083	98.9100	82.9959
TRAINING TIME(SECOND)	34.305190	3.192578	6.628949	120.981310	0.139555
TESTING TIME(SECOND)	257.928970	0.097885	0.088197	30.250127	1.819185
MEMORY(GB)	0.685406	0.698105	0.799507	0.985912	0.740261

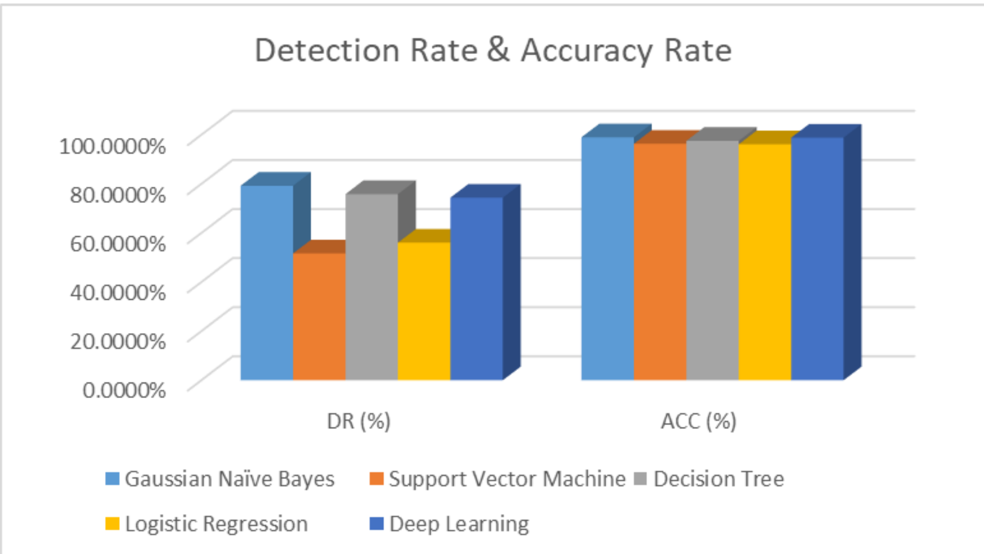

Figure 4.7: DR & ACC Comparison for the 2nd Experiment

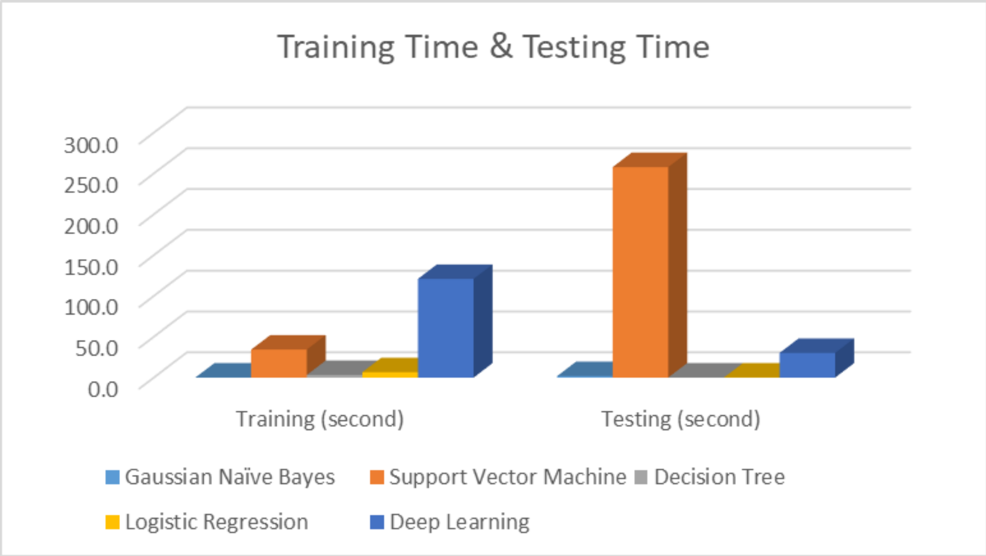

Figure 4.8: Training and Testing Time Comparison for the 2nd Experiment

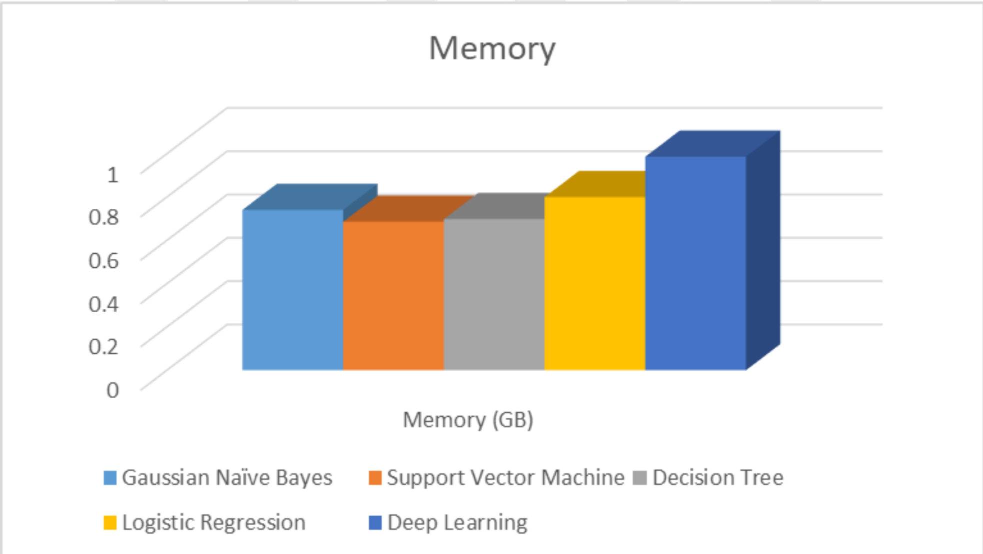

Figure 4.9: Memory Consume Comparison for the 2nd Experiment

```
Decision Tree Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 30
Time taken to perform 5-fold cross validation : 13.394001
Cross validation scores :
[0.95554756 0.98067633 0.9842714  0.987509   0.97193091]
Mean score of  5-fold cross validation : 0.975987
Time taken to train final model : 3.435026
Predictions made using final model...
Time taken to make predictions on test data : 0.099846
Memory used : 0.602497 GB  CPU usage : 18.700000
Confusion matrix :
[[476345    391     43     28    115      7      0]
 [  7675  41360    366    174   1661      0      0]
 [   116    206  37950    114    262   1343      8]
 [    61    200    144    686     98      0      0]
 [    65    320    247     56  20992     10      3]
 [     1      0    756      0      9   3957      0]
 [     1      1      6     11      5      0      5]]
Accuracy score on test data is : 0.975658
For WWW Class, Detection Rate is % 98.36473493796773
For MAIL Class, Detection Rate is % 97.36804934318941
For OTHER Class, Detection Rate is % 96.0467706013363
For INTERACTIVE Class, Detection Rate is % 64.17212347988774
For BULK Class, Detection Rate is % 90.7095324518192
For SERVICE Class, Detection Rate is % 74.42166635320669
For MEDIA Class, Detection Rate is % 31.25
```

Figure 4.10: Outputs for 2nd Experiment

The 3rd experimental result showed that DT classifier reached a DR rate of 76.6766 % as maximum value compared with other classifiers. The maximum ACC rate is 98.8694 % that reached by DL classifier. The maximum DR rate for network classes was reached by DT classifier. Although the training time is long, DL classifier reached a high DR rate. The minimum time taken to train is 0.121520 seconds that reached by GNB classifier and the minimum time taken to test is 0.074805 second reached by LR classifier. Moreover, the minimum memory usage is 0.513020 GB that reached by SVM classifier.

Table 4.5: 3rd Experiment of First Scenario

	EXPERIMETNS				
	SVM	**DT**	**LR**	**DL**	**GNB**
DR (%)	56.8923	76.6766	55.5178	73.9691	60.4801
ACC (%)	96.6302	98.7969	96.1393	98.8694	90.2403
DR for WWW (%)	99.7343	99.7605	99.6135	97.7406	95.1244
DR for MAIL (%)	85.1881	96.6683	84.8368	58.2278	61.3143
DR for OTHER (%)	93.0723	94.4548	92.5798	96.1108	83.9420
DR for INTERACTIVE (%)	3.3641	53.4903	7.9058	0	30.3616
DR for BULK (%)	80.2793	96.9898	72.0140	97.3861	72.0278
DR for SERVICE (%)	36.6080	81.5795	31.6747	68.5345	63.3495
DR for MEDIA (%)	0	13.7931	0	99.7839	17.2413
MEAN OF CROSS V. (%)	96.2161	97.6234	95.6980	98.8300	87.7698
TRAINING TIME(SECOND)	64.876303	2.557625	5.147132	118.425041	0.121520
TESTING TIME(SECOND)	181.408424	0.085482	0.074805	34.845944	1.351757
MEMORY(GB)	0.513020	0.646824	0.628727	0.877766	0.712097

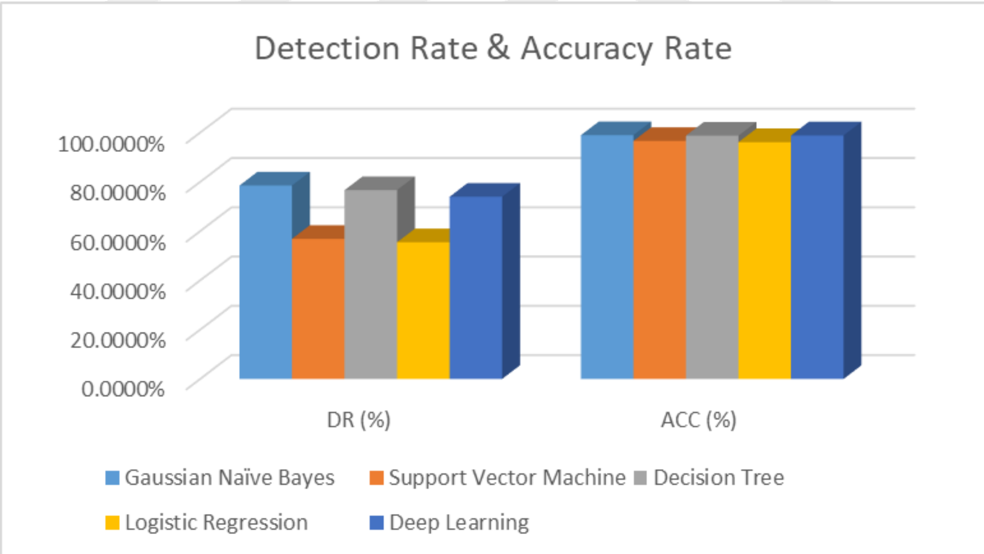

Figure 4.11: DR & ACC Comparison for the 3rd Experiment

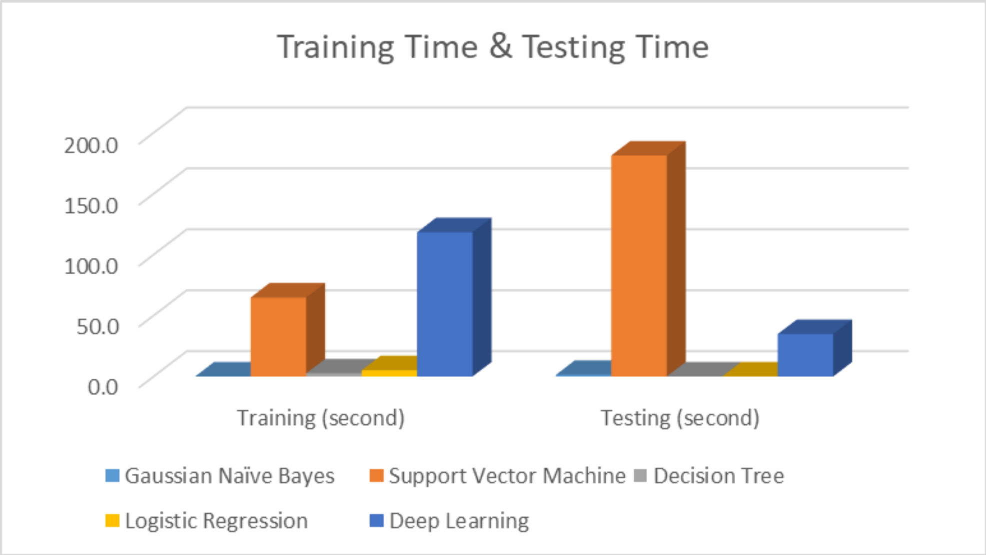

Figure 4.12: Training and Testing Time Comparison for the 3rd Experiment

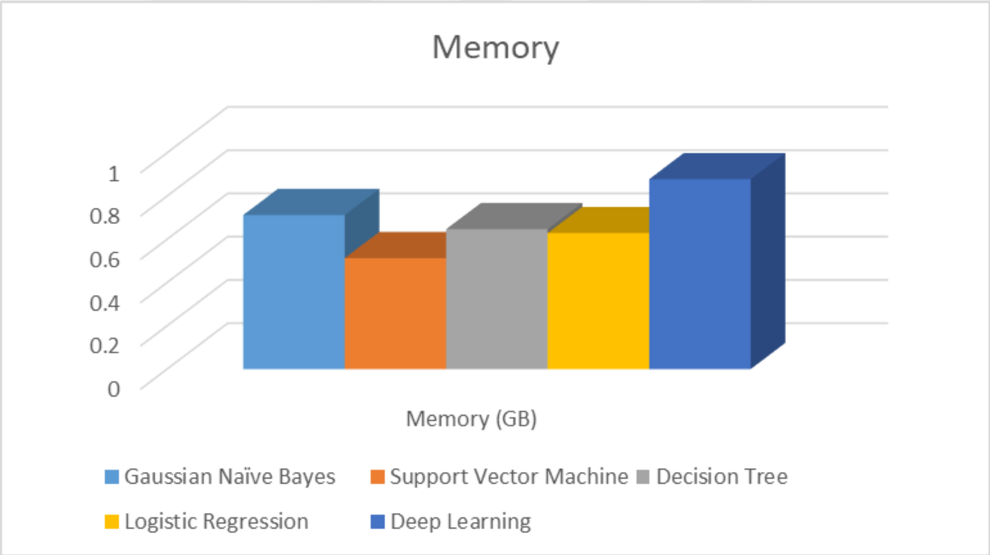

Figure 4.13: Memory Consume Comparison for the 3rd Experiment

```
Decision Tree Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 20
Time taken to perform 5-fold cross validation : 9.152325
Cross validation scores :
[0.95164191 0.97990544 0.98391159 0.98478462 0.98092741]
Mean score of  5-fold cross validation : 0.976234
Time taken to train final model : 2.557285
Predictions made using final model...
Time taken to make predictions on test data : 0.087730
Memory used : 0.662579 GB  CPU usage : 20.000000
Confusion matrix :
[[475787    947     55     74     47     12      7]
 [   905  49529    232    204    360      1      5]
 [   116    221  37781    110    452   1294     25]
 [    62    216    141    636    134      0      0]
 [    62    257    255     78  21040      1      0]
 [     0      1    868      0      1   3853      0]
 [     0      4      7      9      5      0      4]]
Accuracy score on test data is : 0.987969
For WWW Class, Detection Rate is % 99.75992384658609
For MAIL Class, Detection Rate is % 96.78358573522227
For OTHER Class, Detection Rate is % 96.0395536236305
For INTERACTIVE Class, Detection Rate is % 57.24572457245725
For BULK Class, Detection Rate is % 95.46712645764327
For SERVICE Class, Detection Rate is % 74.65607440418523
For MEDIA Class, Detection Rate is % 9.75609756097561
```

Figure 4.14: Outputs for 3rd Experiment

The 4th experimental result showed that DT classifier reached by a DR rate of 98.6031 % as maximum compared with other classifiers. The maximum ACC rate is 98.5930 % that reached by DL classifier. The maximum DR rate for network classes was reached by DT classifier. The minimum time taken to train is 0.099856 seconds that reached by GNB classifier and minimum time taken to test is 0.063542 second reached by LR classifier. Moreover, minimum memory usage is 0.595535 GB that reached by DT classifier.

Table 4.6: 4th Experiment of First Scenario

	EXPERIMETNS				
	SVM	**DT**	**LR**	**DL**	**GNB**
DR (%)	58.9431	98.6031	53.9965	73.8632	64.4817
ACC (%)	97.3076	76.3247	95.9443	98.5930	91.6638
DR for WWW (%)	99.6966	99.6125	99.5991	96.6327	95.4873
DR for MAIL (%)	87.0130	95.6105	83.4081	60.0000	72.5544
DR for OTHER (%)	93.2948	946473	92.9448	97.1305	86.8871
DR for INTERACTIVE (%)	0	52.6492	0.2523	0	54.3313
DR for BULK (%)	94.9522	96.3124	70.0963	94.7365	72.1707
DR for SERVICE (%)	37.6455	80.6478	31.6747	69.0048	52.6995
DR for MEDIA (%)	0	13.7931	0	99.5379	17.2413
MEAN OF CROSS V. (%)	97.1300	97.3664	95.4338	98.5400	90.4417
TRAINING TIME(SECOND)	13.486251	1.229930	2.607169	122.421115	0.099856
TESTING TIME(SECOND)	121.162376	0.070930	0.063542	40.892389	0.720488
MEMORY(GB)	0.736279	0.595535	0.699520	0.845486	0.692577

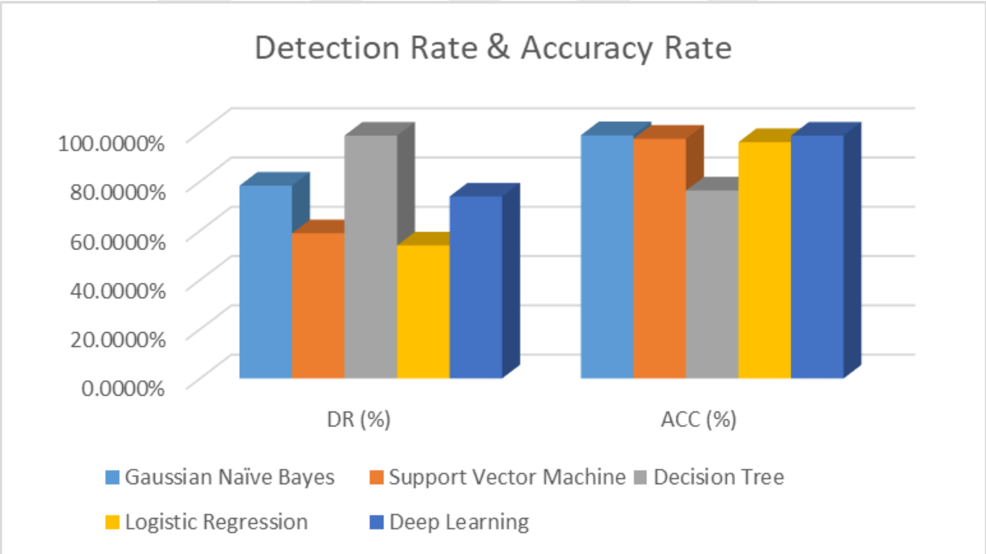

Figure 4.15: DR & ACC Comparison for the 4th Experiment

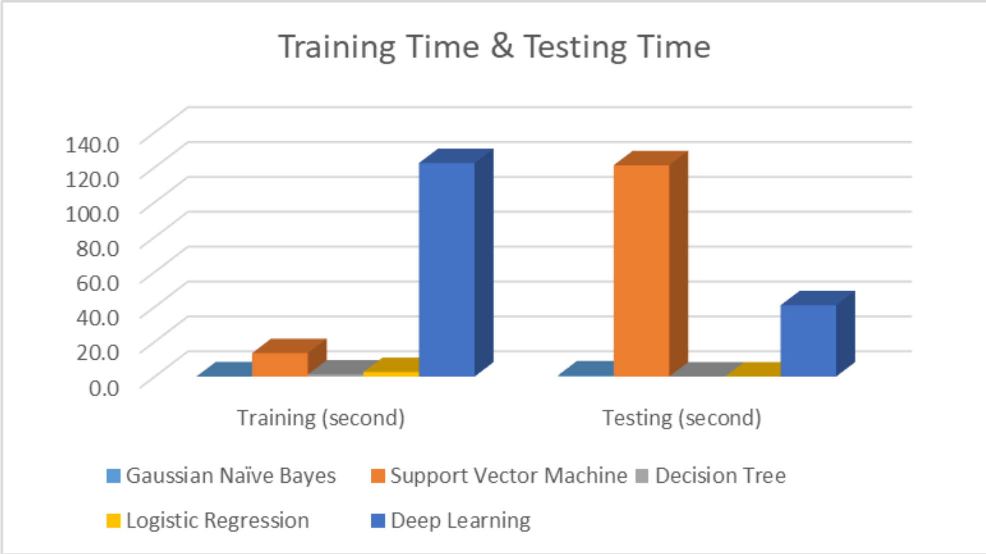

Figure 4.16: Training and Testing Time Comparison for the 4th Experiment

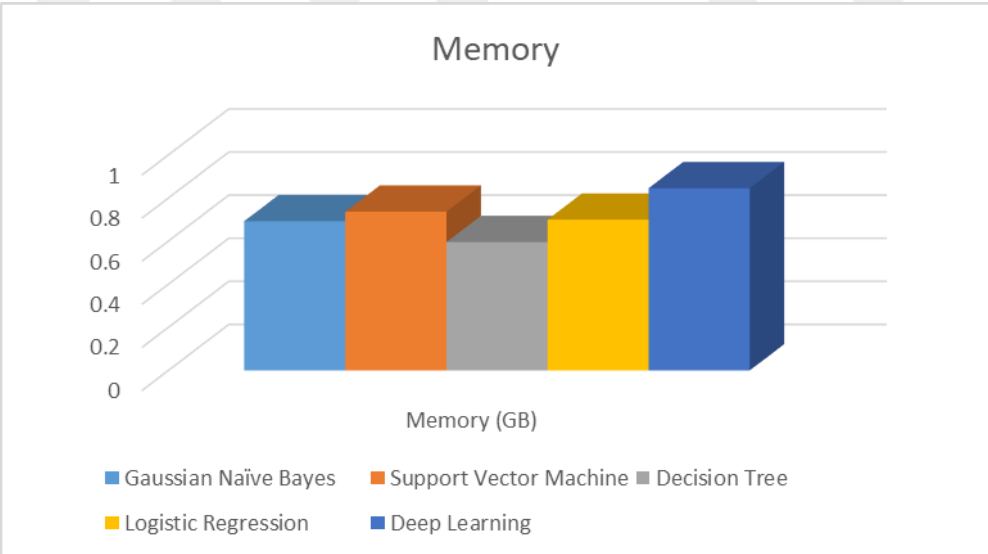

Figure 4.17: Memory Consume Comparison for the 4th Experiment

```
Decision Tree Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 10
Time taken to perform 5-fold cross validation : 4.855016
Cross validation scores :
[0.94958631 0.98077911 0.98175276 0.98283129 0.97337035]
Mean score of  5-fold cross validation : 0.973664
Time taken to train final model : 1.242173
Predictions made using final model...
Time taken to make predictions on test data : 0.070874
Memory used : 0.614639 GB  CPU usage : 17.100000
Confusion matrix :
[[475081   1702     49     49     34     13      1]
 [  1504  48987    225    193    308      1     18]
 [   136    224  37858    139    287   1332     23]
 [    69    227    170    626     96      0      1]
 [    52    254    214     60  21110      0      3]
 [     0      2    904      8      0   3809      0]
 [     2      2     10      9      2      0      4]]
Accuracy score on test data is : 0.986031
For WWW Class, Detection Rate is % 99.63027740728624
For MAIL Class, Detection Rate is % 95.30915599828788
For OTHER Class, Detection Rate is % 96.01318792797362
For INTERACTIVE Class, Detection Rate is % 57.74907749077491
For BULK Class, Detection Rate is % 96.67078811192013
For SERVICE Class, Detection Rate is % 73.8894277400582
For MEDIA Class, Detection Rate is % 8.0
```

Figure 4.18: Output for 4^{th} Experiment

The fifth experimental result showed that DT classifier reached a DR rate of 74.2333 % as maximum value compared with other classifiers. The maximum ACC rate is 97.6861 % that reached by DT classifier. The maximum DR rate for network classes was reached by DT classifier. The minimum time taken to train is 0.090653 seconds that reached by GNB classifier and the minimum time taken to test is 0.056931 seconds reached by LR classifier. Moreover, the minimum memory usage is 0.543766 GB that reached by SVM classifier.

Table 4.7: 5th Experiment of First Scenario

	EXPERIMETNS				
	SVM	**DT**	**LR**	**DL**	**GNB**
DR (%)	51.0742	74.2333	46.0962	65.5014	60.2825
ACC (%)	94.9800	97.6861	93.1589	97.5127	92.1700
DR for WWW (%)	98.1749	99.1493	98.5924	91.5607	96.5625
DR for MAIL (%)	83.8746	91.9626	68.1649	13.3333	70.4582
DR for OTHER (%)	96.1374	93.5248	95.4598	90.5986	88.1997
DR for INTERACTIVE (%)	0.8410	38.0151	0.1682	0	0.5046
DR for BULK (%)	73.7703	94.2285	52.2426	95.3255	66.6897
DR for SERVICE (%)	4.7215	78.6152	8.0457	68.6417	58.1833
DR for MEDIA (%)	0	24.1379	0	99.0499	41.3793
MEAN OF CROSS V. (%)	94.7029	96.5276	92.5555	97.2300	90.9701
TRAINING TIME(SECOND)	14.668287	0.515767	1.083124	121.528057	0.090653
TESTING TIME(SECOND)	136.944998	0.062312	0.056931	37.436296	0.464589
MEMORY(GB)	0.543766	0.569656	0.683208	0.838245	0.689236

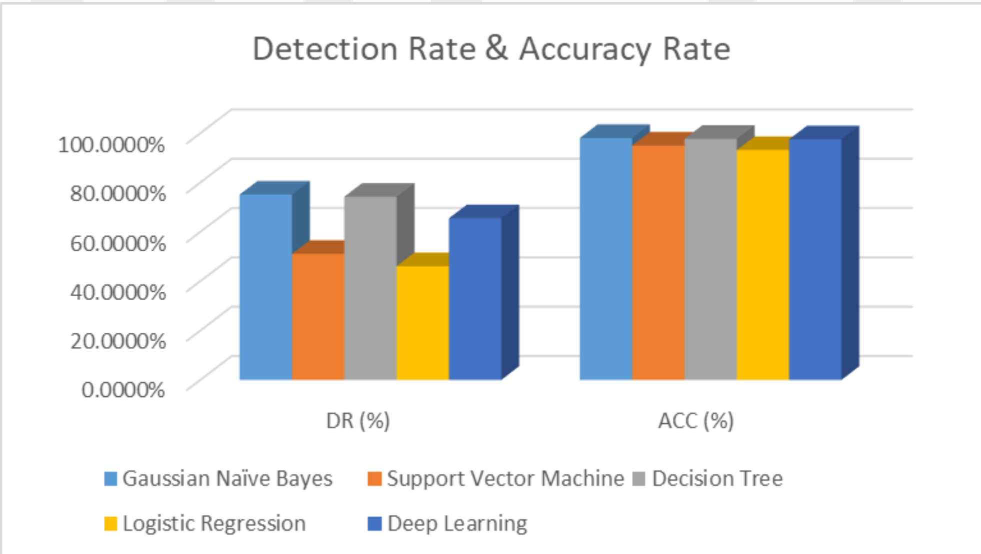

Figure 4.19: DR & ACC Comparison for the 5th Experiment

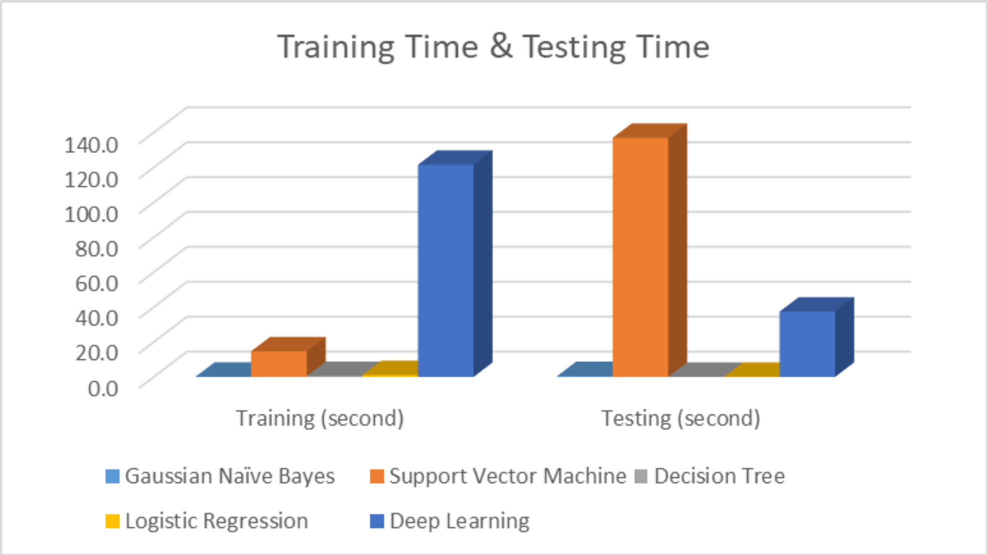

Figure 4.20: Training and Testing Time Comparison for the 5th Experiment

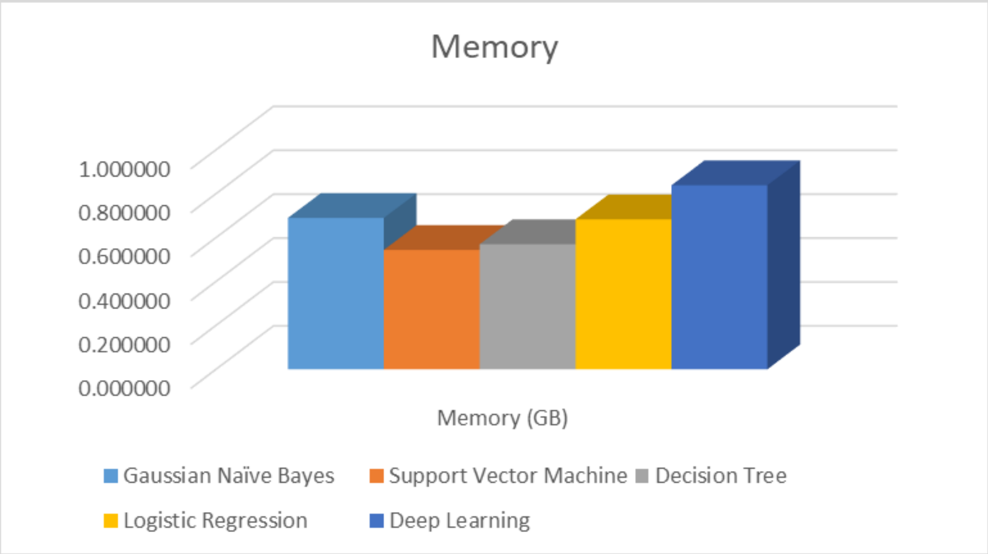

Figure 4.21: Memory Consume Comparison for the 5th Experiment

```
Decision Tree Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 5
Time taken to perform 5-fold cross validation : 2.205563
Cross validation scores :
[0.93473457 0.97224792 0.97481367 0.97388712 0.9706971 ]
Mean score of  5-fold cross validation : 0.965276
Time taken to train final model : 0.508035
Predictions made using final model...
Time taken to make predictions on test data : 0.057651
Memory used : 0.591026 GB  CPU usage : 15.100000
Confusion matrix :
[[472872   3830     60     31    117     15      4]
 [  3111  47118    212    198    585      5      7]
 [   143    242  37409    159    466   1562     18]
 [    51    228    182    452    271      2      3]
 [   109    555    321    229  20441     30      8]
 [     0      1    991      8     10   3713      0]
 [     4      1      5      8      4      0      7]]
Accuracy score on test data is : 0.976861
For WWW Class, Detection Rate is % 99.28236998467321
For MAIL Class, Detection Rate is % 90.65512265512265
For OTHER Class, Detection Rate is % 95.47983665135273
For INTERACTIVE Class, Detection Rate is % 41.65898617511521
For BULK Class, Detection Rate is % 93.363478578606
For SERVICE Class, Detection Rate is % 69.70152055565984
For MEDIA Class, Detection Rate is % 14.893617021276595
```

Figure 4.22: Output for 5th Experiment

Scenario 2: Three experiments were carried out to see the scalability of NMS as fallow:

1- We have used 125,793 vector of KDD dataset to train NMS and 595,798 rows for testing.

2- We have used 494,021 vector of KDD dataset to train NMS and 595,798 rows for testing.

3- We have used 1,000,000 vector of KDD dataset to train NMS and 595,798 rows for testing.

Outcomes are illustrated as follows.

Experimental result showed that SVM classifier reached highest DR rate in first and second experiments and DT classifier achieved highest DR rate in third experiment compared with other classifiers. SVM classifier achieved highest ACC rate in all experiments compared with other classifiers. Highest DR rate for network classes was achieved by SVM classifier.

Table 4.8: Scalability Experiments for Support Vector Machine Algorithm

	1.Experiment	2.Experiment	3.Experiment
DR (%)	42.2910	40.1904	43.8707
ACC (%)	96.1984	96.1737	97.0856
DR for WWW (%)	99.7536	98.7353	99.7483
DR for MAIL (%)	81.2358	82.2468	85.4379
DR for OTHER (%)	96.9249	96.7174	93.1648
DR for INTERACTIVE (%)	25.7359	7.3170	17.5777
DR for BULK (%)	76.9695	75.6972	98.9074
DR for SERVICE (%)	0	0	0
DR for REMOTE (%)	0	0	0
DR for DATABASE (%)	0	0	0
DR for MEDIA (%)	0	0	0
MEAN OF CROSS V. (%)	80.5173	98.2594	95.1001
TRAINING TIME(SECOND)	941.402687	3509.292585	18861.513270
TESTING TIME(SECOND)	1504.380082	686.375096	2062.129607
MEMORY(GB)	0.605934	0.366829	0.970428

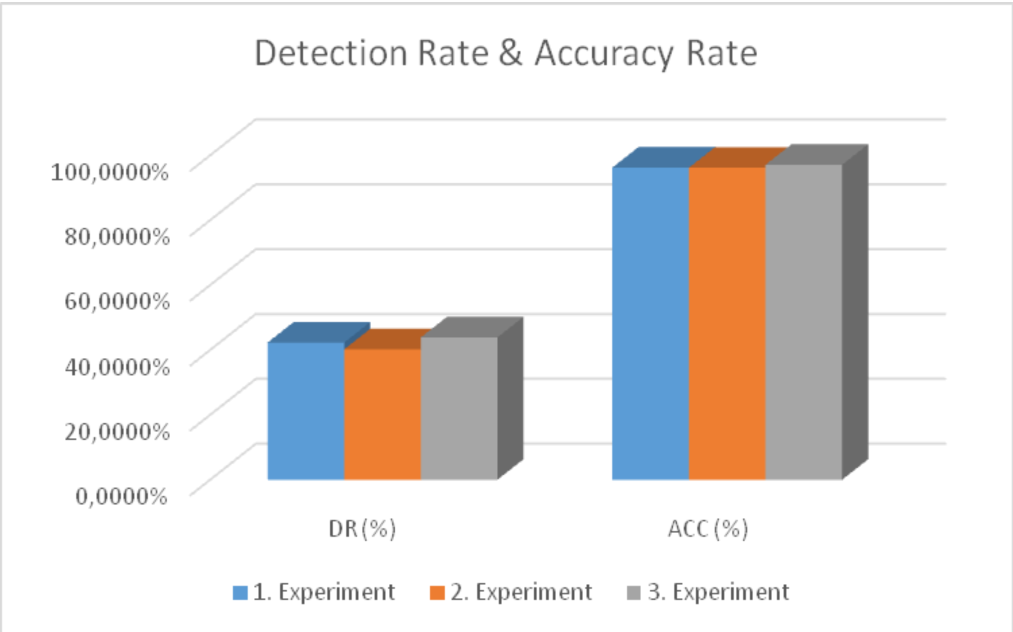

Figure 4.23: DR & ACC Comparison for SVM Algorithm

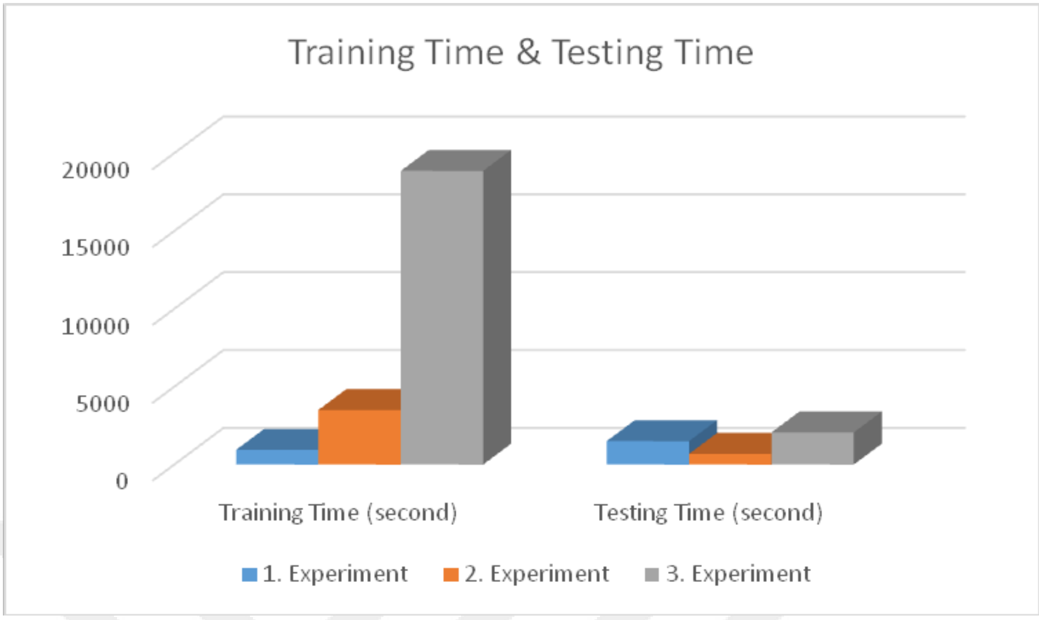

Figure 4.24: Training and Testing Time Comparison for SVM Algorithm

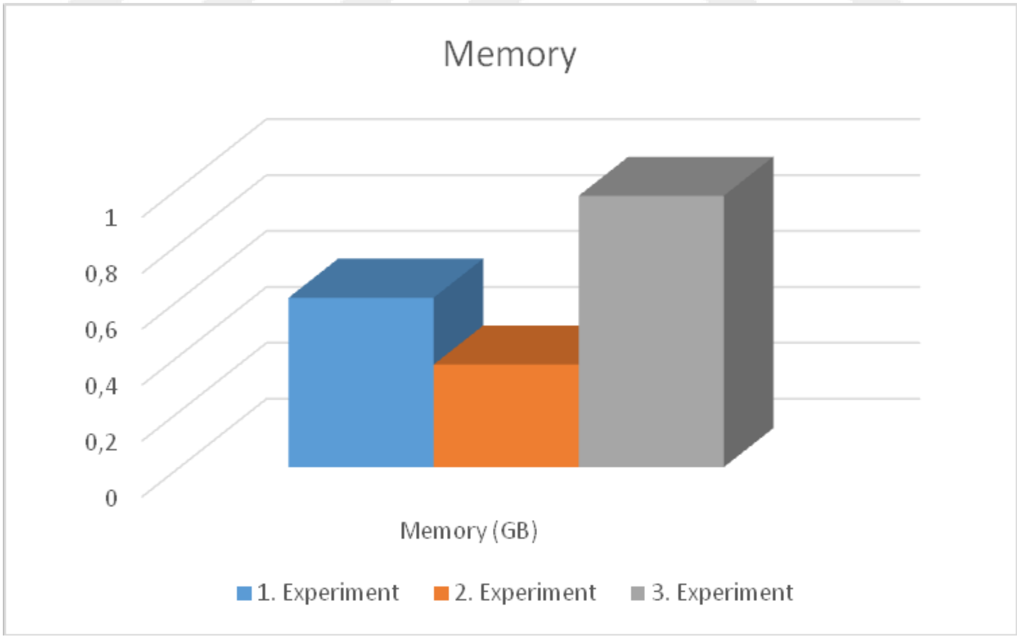

Figure 4.25: Memory Consume Comparison for SVM Algorithm

Table 4.9: Scalability Experiments for Decision Tree Algorithm

	1.Experiment	2.Experiment	3.Experiment
DR (%)	34.0963	32.7188	52.4999
ACC (%)	60.2649	89.8068	92.6321
DR for WWW (%)	63.0867	98.6618	97.8930
DR for MAIL (%)	25.4274	69.1993	79.3933
DR for OTHER (%)	77.8519	54.8838	82.0395
DR for INTERACTIVE (%)	11.9428	23.4650	74.0958
DR for BULK (%)	56.4329	27.7877	90.1673
DR for SERVICE (%)	34.1943	17.0230	12.9790
DR for REMOTE (%)	0	0	0
DR for DATABASE (%)	0	0	0
DR for MEDIA (%)	37.9310	3.4482	37.9310
MEAN OF CROSS V. (%)	80.1712	95.4338	95.2322
TRAINING TIME(SECOND)	8.857297	20.935107	55.359079
TESTING TIME(SECOND)	0.115061	0.139380	0.137375
MEMORY(GB)	0.870491	1.029617	1.368744

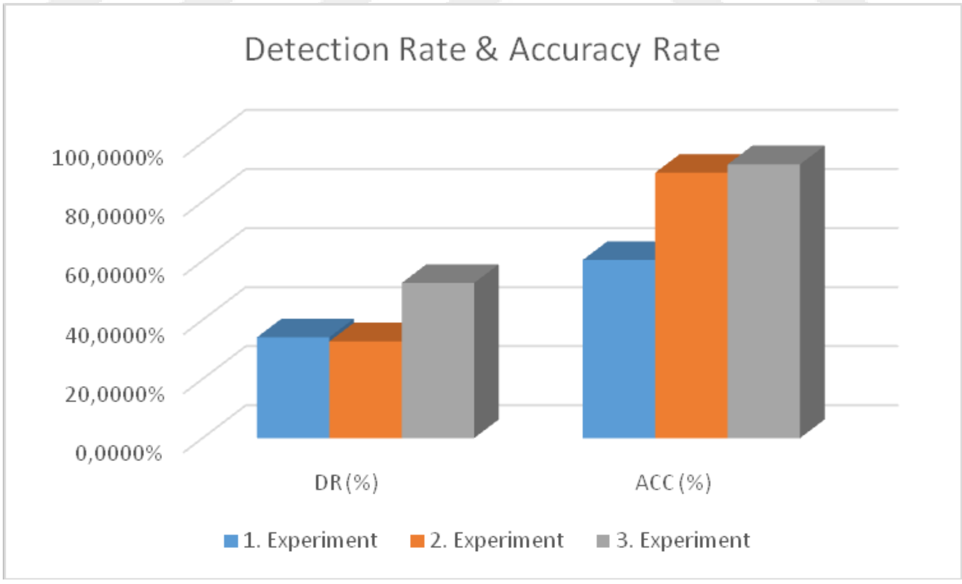

Figure 4.26: DR & ACC Comparison for DT Algorithm

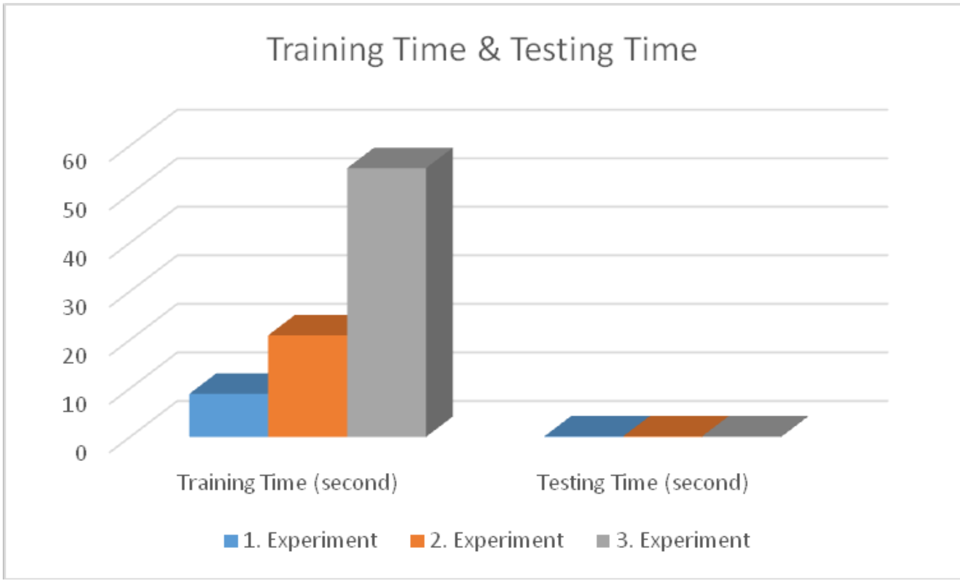

Figure 4.27: Training and Testing Time Comparison for DT Algorithm

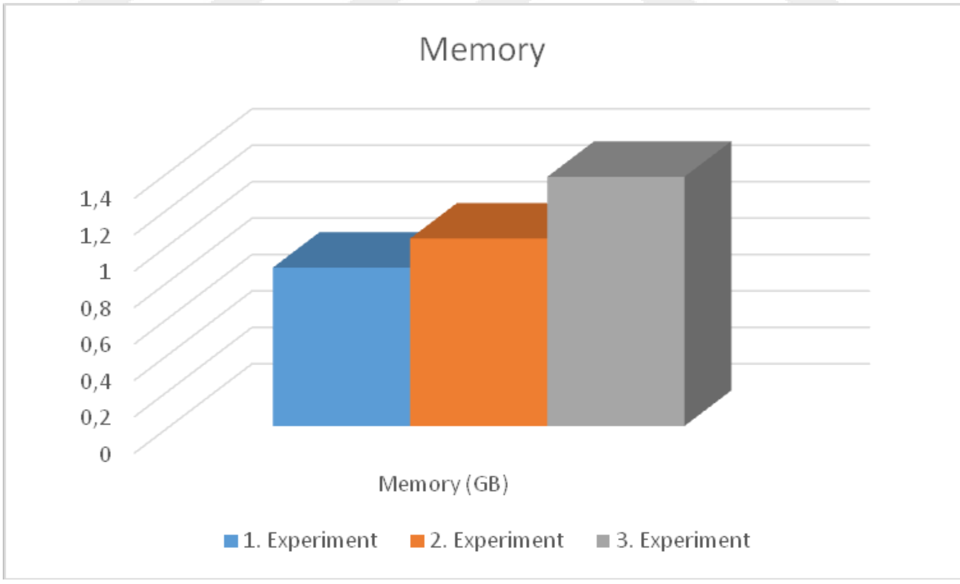

Figure 4.28: Memory Consume Comparison for DT Algorithm

Table 4.10: Scalability Experiments for Logistic Regression Algorithm

	1.Experiment	2.Experiment	3.Experiment
DR (%)	28.3212	33.0188	31.3164
ACC (%)	90.4110	91.2449	77.6468
DR for WWW (%)	99.4093	98.0661	78.0004
DR for MAIL (%)	44.4218	77.8300	97.0899
DR for OTHER (%)	96.7049	81.9445	96.7874
DR for INTERACTIVE (%)	0	0	0
DR for BULK (%)	14.3548	8.3529	9.9064
DR for SERVICE (%)	0	30.9760	0.0635
DR for REMOTE (%)	0	0	0
DR for DATABASE (%)	0	0	0
DR for MEDIA (%)	0	0	0
MEAN OF CROSS V. (%)	79.1297	98.0784	95.2067
TRAINING TIME(SECOND)	3.236786	10.060068	19.838131
TESTING TIME(SECOND)	0.067650	0.071333	0.066887
MEMORY(GB)	0.722469	0.867374	1.019993

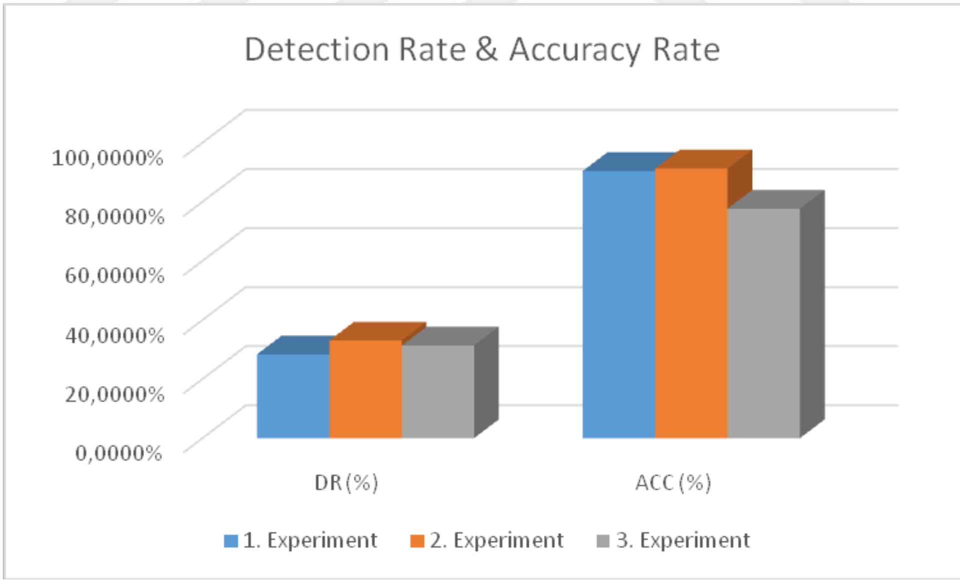

Figure 4.29: DR & ACC Comparison for LR Algorithm

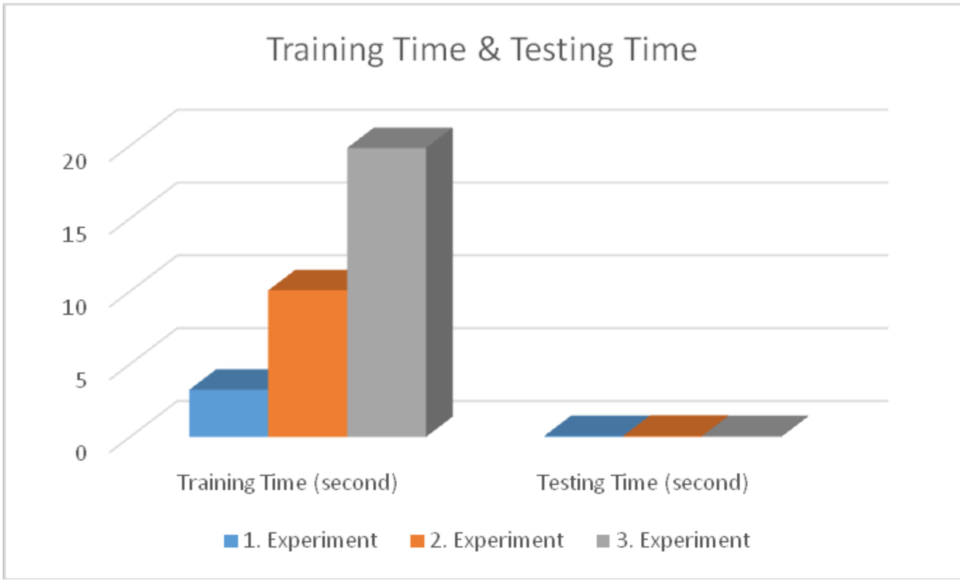

Figure 4.30: Training and Testing Time Comparison for LR Algorithm

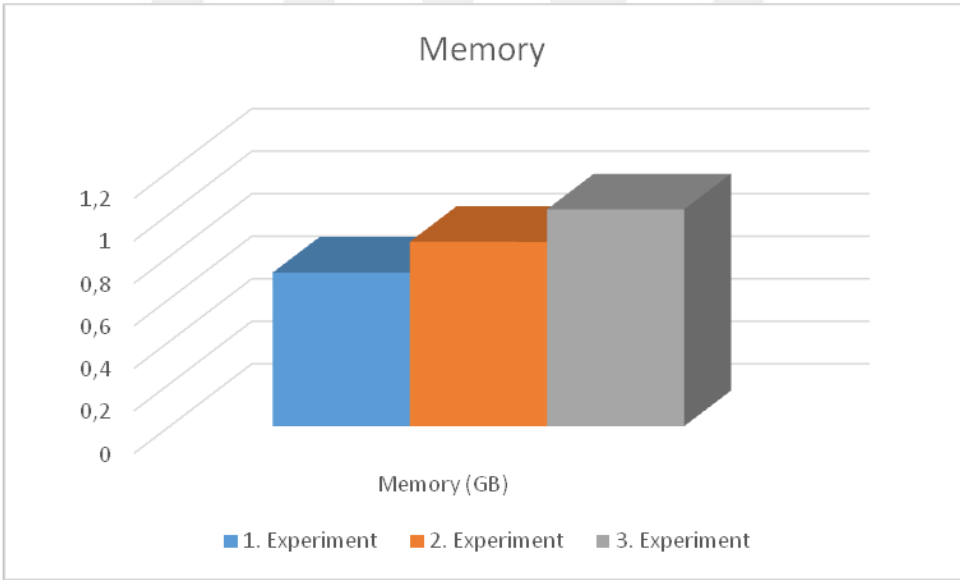

Figure 4.31: Memory Consume Comparison for LR Algorithm

Table 4.11: Scalability Experiments for DL Algorithm

	1.Experiment	2.Experiment	3.Experiment
DR (%)	20.5481	17.7597	12.6341
ACC (%)	2.28902	3.6543	3.3543
DR for WWW (%)	47.4525	99.5948	99.9396
DR for MAIL (%)	40.0000	0	0
DR for OTHER (%)	0	54.5995	7.5688
DR for INTERACTIVE (%)	6.7200	0	0.0049
DR for BULK (%)	90.7609	0	0
DR for SERVICE (%)	0	5.6533	6.1942
DR for REMOTE (%)	0	0	0
DR for DATABASE (%)	0	0	0
DR for MEDIA (%)	0	0	0
MEAN OF CROSS V. (%)	83.8800	98.7800	98.5800
TRAINING TIME(SECOND)	164.890413	657.273461	1354.568521
TESTING TIME(SECOND)	46.122825	43.937499	44.832119
MEMORY(GB)	1.136715	1.256710	1.273716

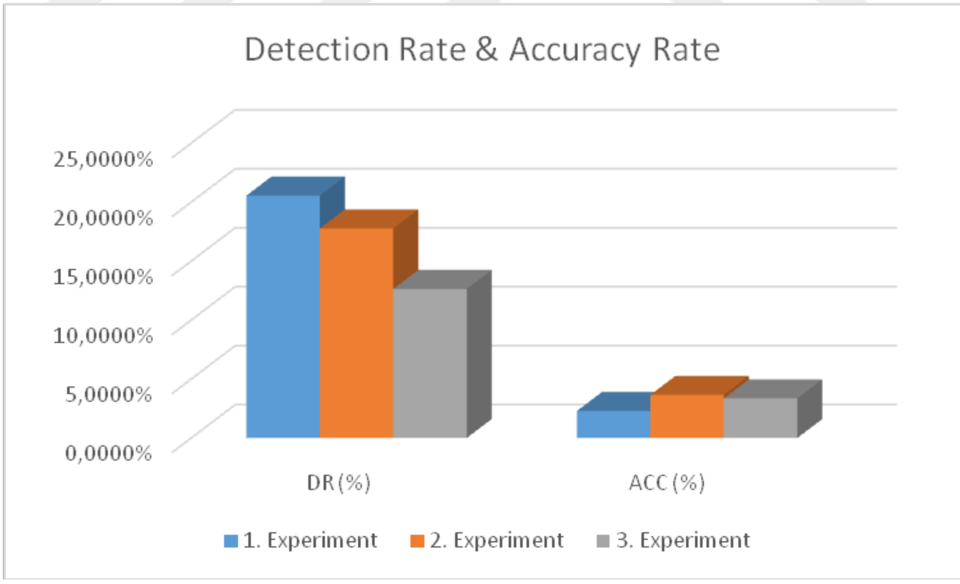

Figure 4.32: DR & ACC Comparison for DL Algorithm

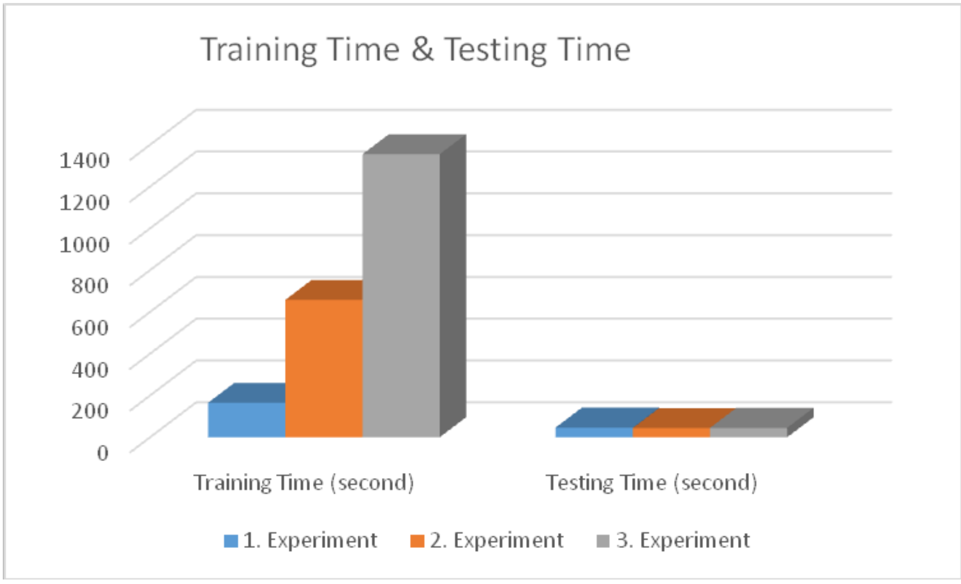

Figure 4.33: Training and Testing Time Comparison for DL Algorithm

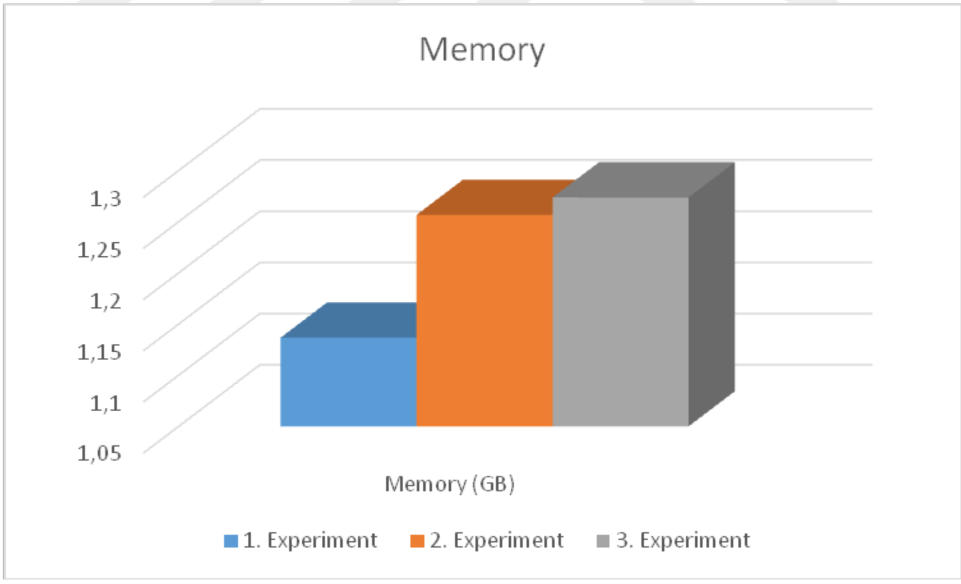

Figure 4.34: Memory Consume Comparison for DL Algorithm

Table 4.12: Scalability Experiments for GNB Algorithm

	1.Experiment	2.Experiment	3.Experiment
DR (%)	30.0925	37.2135	37.9236
ACC (%)	57.7956	85.0720	61.5967
DR for WWW (%)	68.5697	92.3539	61.5412
DR for MAIL (%)	14.9933	44.6834	60.9805
DR for OTHER (%)	2.8275	94.6448	89.7972
DR for INTERACTIVE (%)	27.5862	30.6980	36.4171
DR for BULK (%)	15.9636	24.2658	27.0594
DR for SERVICE (%)	99.5130	0	0
DR for REMOTE (%)	0	0	0
DR for DATABASE (%)	0	0	0
DR for MEDIA (%)	41.3793	48.2758	65.5172
MEAN OF CROSS V. (%)	39.8331	75.6497	69.2628
TRAINING TIME(SECOND)	0.220978	0.865924	1.784725
TESTING TIME(SECOND)	3.015205	3.018600	3.043701
MEMORY(GB)	1.014290	1.455036	1.349602

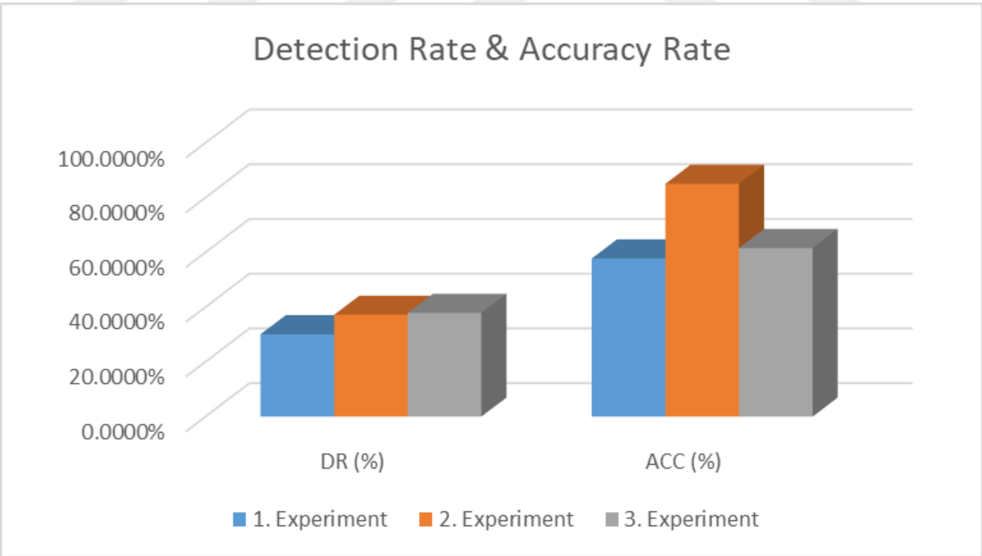

Figure 4.35: DR & ACC Comparison for GNB Algorithm

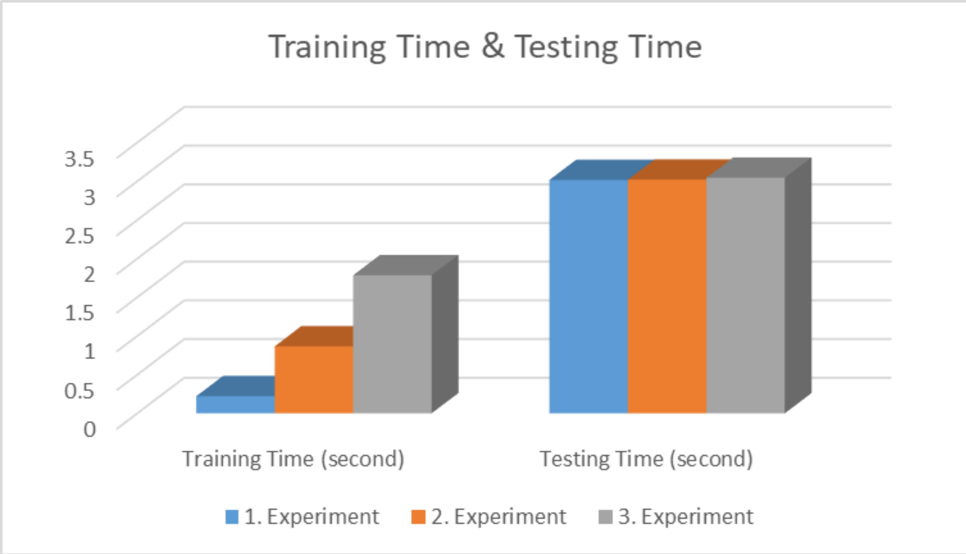

Figure 4.36: Training and Testing Time Comparison for GNB Algorithm

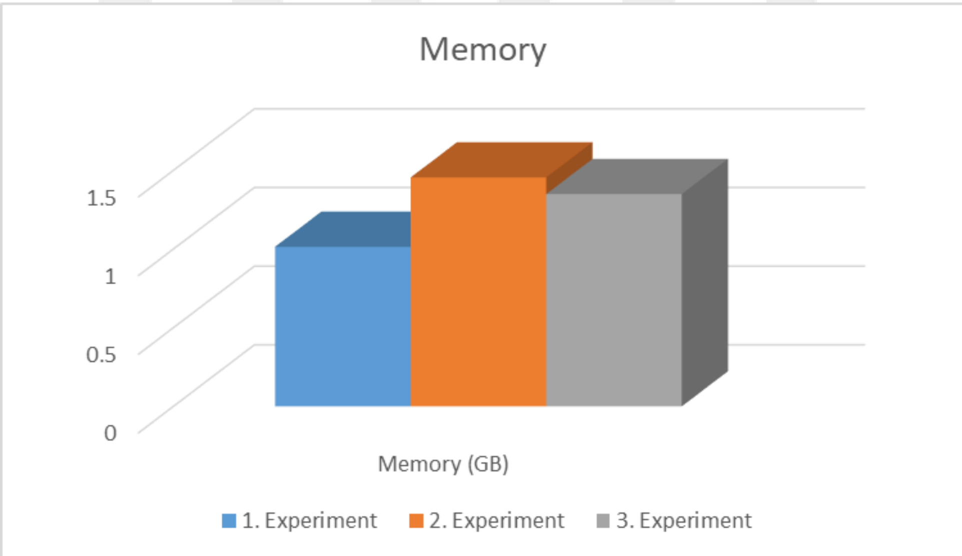

Figure 4.37: Memory Consume Comparison for GNB Algorithm

Scenario 3:

In order to see the impact of feature scaling, an experiment without feature scaling (standard scaler) was implemented to all classifiers and the experimental outcome showed that DT classifier reached the largest DR percentage of 84,9629 percent compared to other algorithms. Maximum ACC rate is 99.4449 % that reached by DT classifier. Maximum DR rate for network classes was reached by DT classifier. The minimum time taken to train is

6.371368 seconds that reached by DT classifier and the minimum time taken to test is 0.059934 second reached by LR classifier. Moreover, the minimum memory usage is 0.285355 GB that reached by SVM classifier.

Table 4.13: Standard Scaler Experiments for SVM Algorithm

	With Standard Scaler	**Without Standard Scaler**
DR (%)	51.1479	43.2610
ACC (%)	96.2793	87.0701
DR for WWW (%)	99.7406	99.9997
DR for MAIL (%)	82.7367	11.2889
DR for OTHER (%)	96.7149	70.2692
DR for INTERACTIVE (%)	1.1774	9.5878
DR for BULK (%)	77.6656	16.9593
DR for SERVICE (%)	0	87.8255
DR for MEDIA (%)	0	6.8965
MEAN OF CROSS V. (%)	96.0249	
TRAINING TIME(SECOND)	45.529160	10286.497473
TESTING TIME(SECOND)	335.489777	4208.674068
MEMORY(GB)	0.743065	0.285355

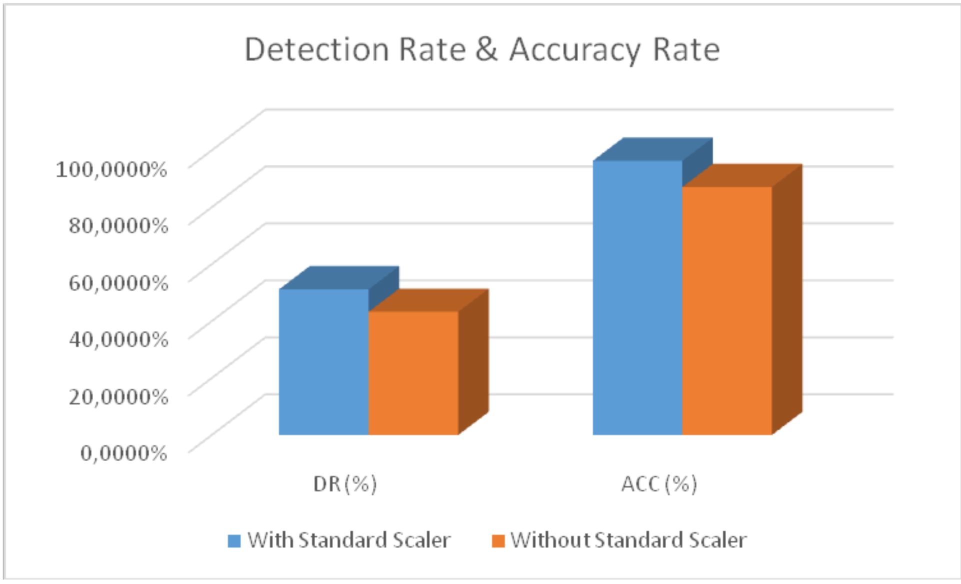

Figure 4.38: DR & ACC Comparison for SVM Algorithm

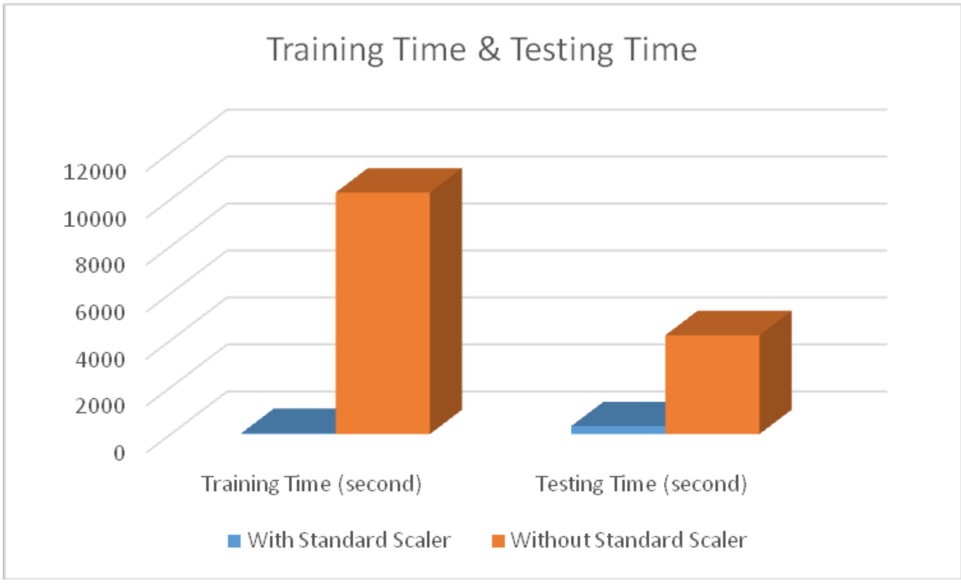

Figure 4.39: Training and Testing Time Comparison for SVM Algorithm

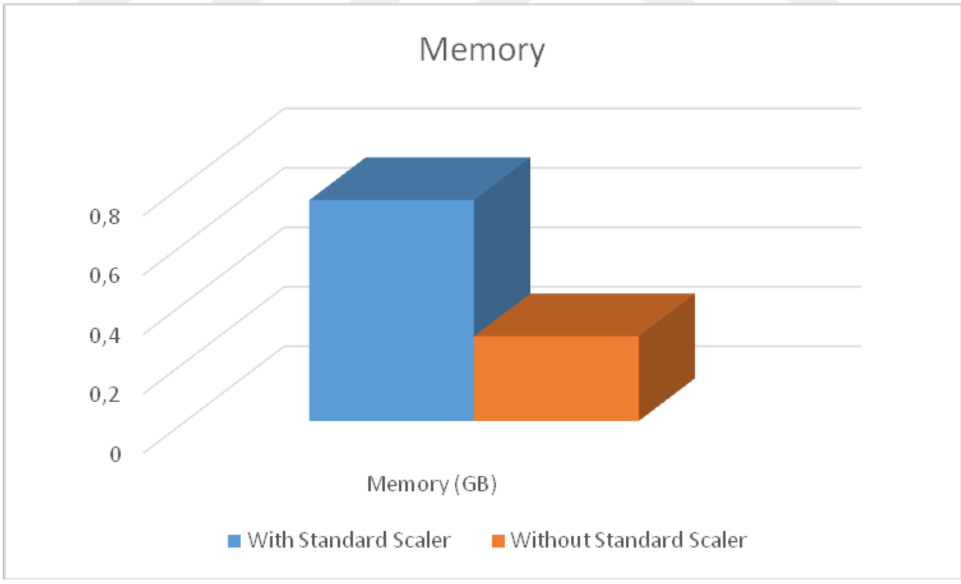

Figure 4.40: Memory Consume Comparison for SVM Algorithm

Table 4.14: Standard Scaler Experiments for Decision Tree Algorithm

	With Standard Scaler	Without Standard Scaler
DR (%)	79.8813	84.9649
ACC (%)	97.7486	99.4449
DR for WWW (%)	99.8135	99.9201
DR for MAIL (%)	83.6521	99.1607
DR for OTHER (%)	94.6473	96.6574
DR for INTERACTIVE (%)	63.9192	83.2632
DR for BULK (%)	96.5887	98.7138
DR for SERVICE (%)	82.6169	86.0046
DR for MEDIA (%)	37.9310	31.0344
MEAN OF CROSS V. (%)	97.9379	98.4581
TRAINING TIME(SECOND)	5.581684	6.371368
TESTING TIME(SECOND)	0.12333	0.132308
MEMORY(GB)	0.741154	0.761307

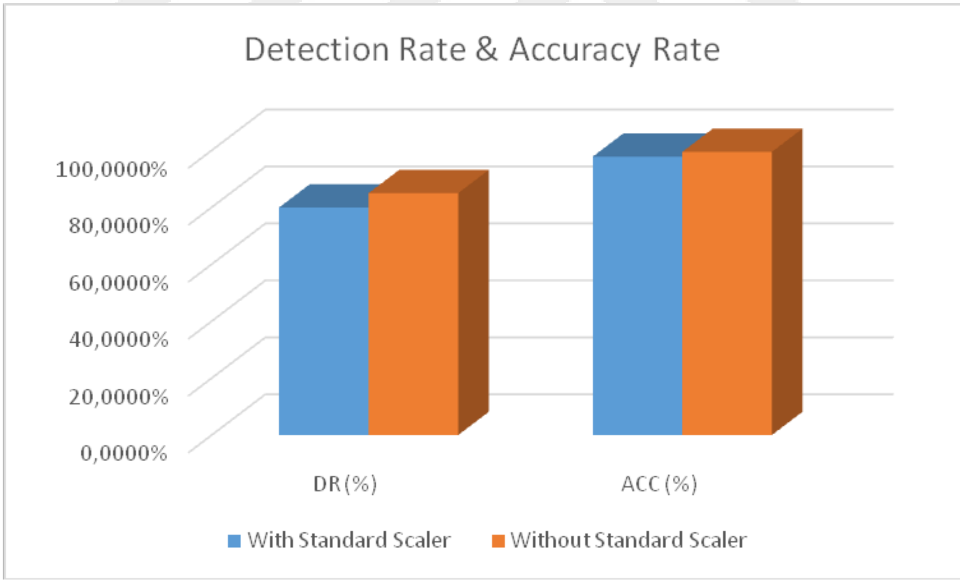

Figure 4.41: DR & ACC Comparison for DT Algorithm

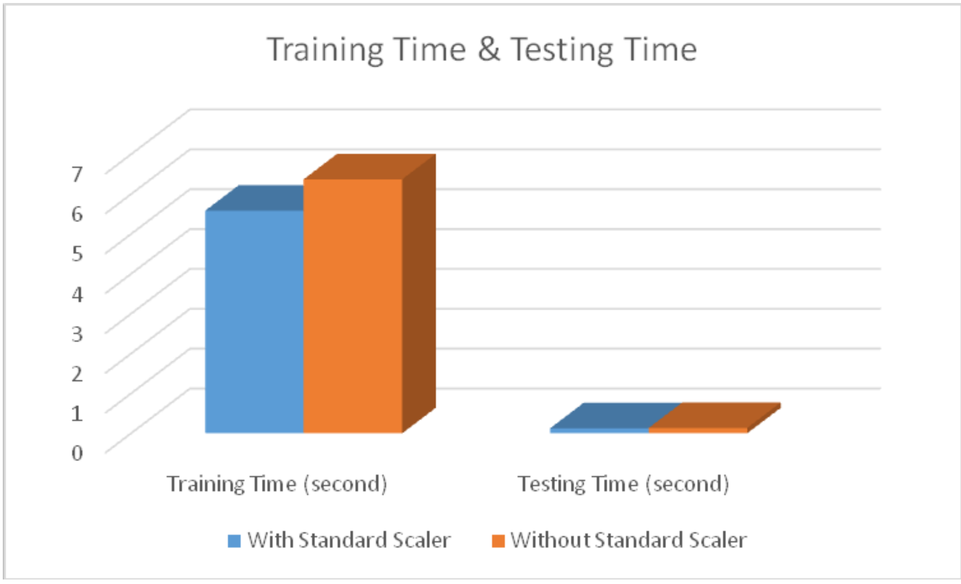

Figure 4.42: Training and Testing Time Comparison for DT Algorithm

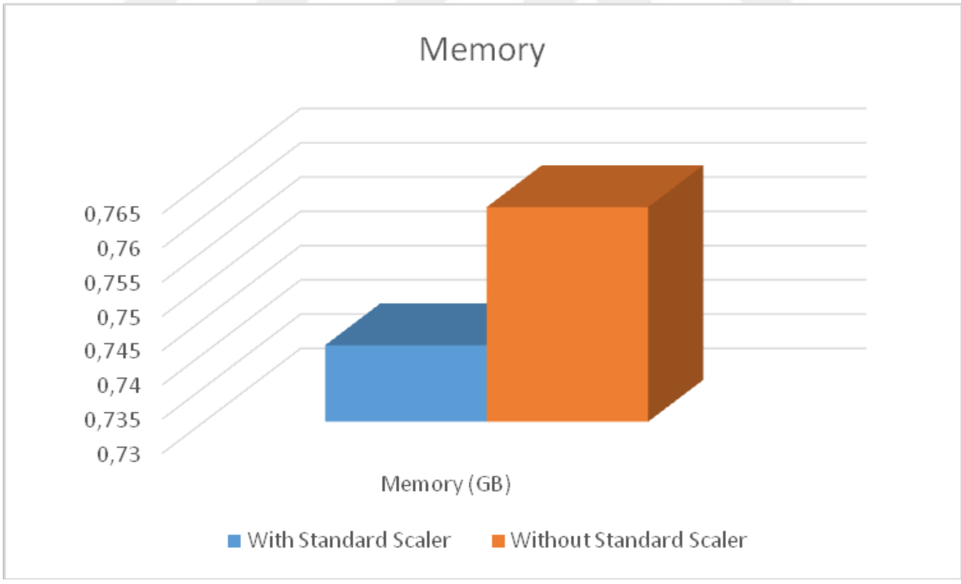

Figure 4.43: Memory Consume Comparison for DT Algorithm

Table 4.15: Standard Scaler Experiments for Logistic Regression Algorithm

	With Standard Scaler	**Without Standard Scaler**
DR (%)	56.0539	26.6159
ACC (%)	96.1568	82.0567
DR for WWW (%)	99.6165	97.7371
DR for MAIL (%)	84.9207	10.8751
DR for OTHER (%)	92.5823	28.6382
DR for INTERACTIVE (%)	11.5222	23.9697
DR for BULK (%)	72.0186	25.0910
DR for SERVICE (%)	31.7171	0
DR for MEDIA (%)	0	0
MEAN OF CROSS V. (%)	95.7114	75.0696
TRAINING TIME(SECOND)	7.594568	9.285133
TESTING TIME(SECOND)	0.102367	0.059934
MEMORY(GB)	0.813011	0.683086

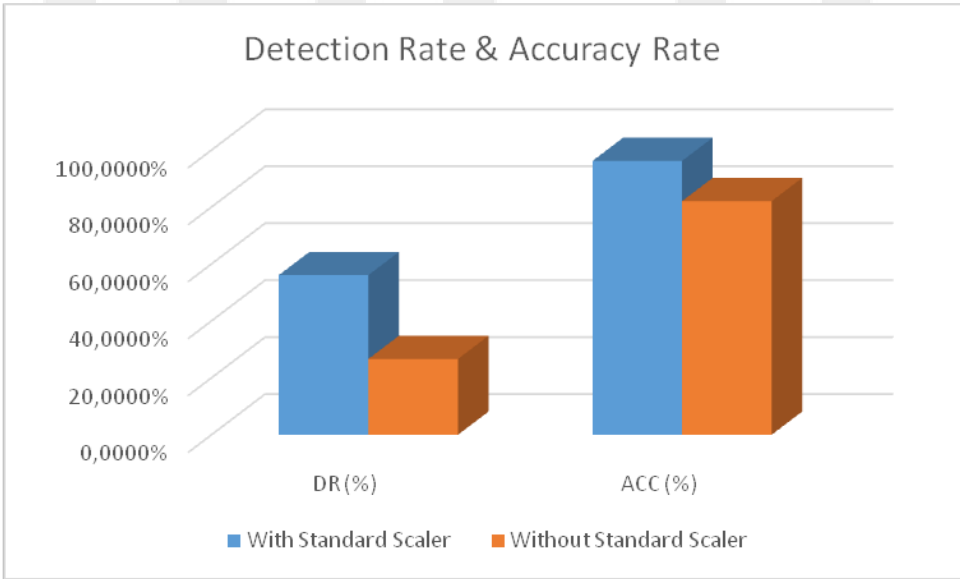

Figure 4.44: DR & ACC Comparison for LR Algorithm

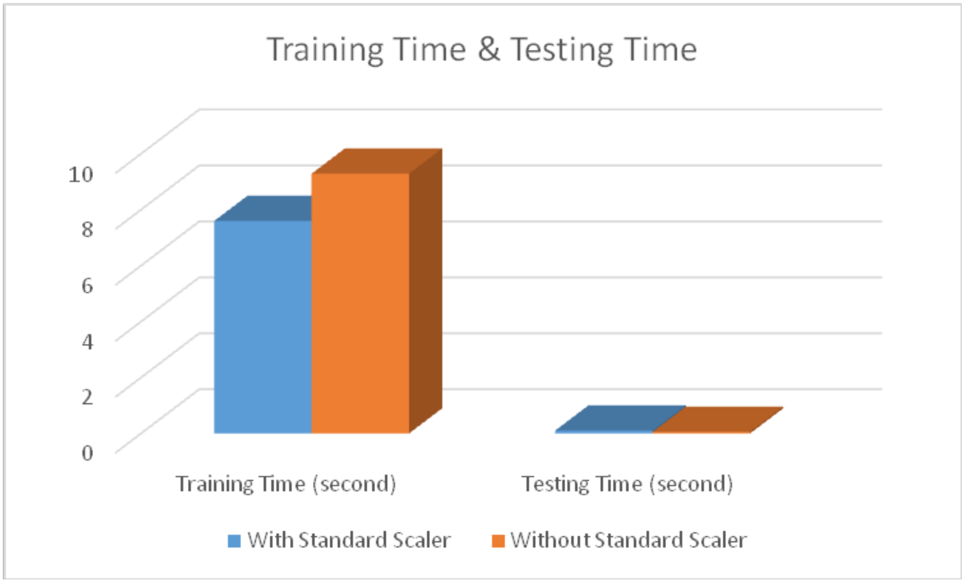

Figure 4.45: Training and Testing Time Comparison for LR Algorithm

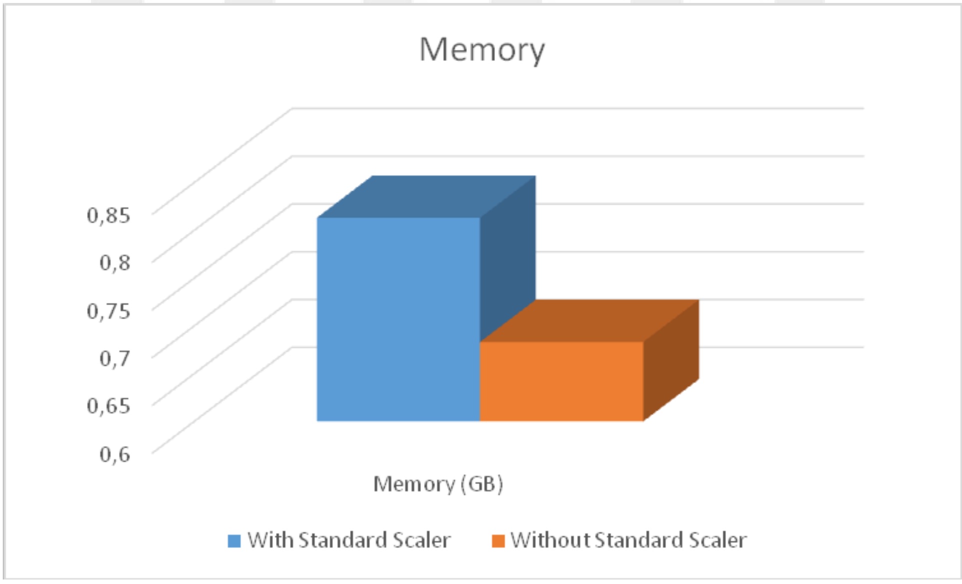

Figure 4.46: Memory Consume Comparison for LR Algorithm

Table 4.16: Standard Scaler Experiments for DL Algorithm

	With Standard Scaler	Without Standard Scaler
DR (%)	78.8496	0.5201
ACC (%)	98.9923	0.036410
DR for WWW (%)	99.6575	3.6409
DR for MAIL (%)	90.6064	0
DR for OTHER (%)	96.3380	0
DR for INTERACTIVE (%)	0	0
DR for BULK (%)	97.4803	0
DR for SERVICE (%)	68.0890	0
DR for MEDIA (%)	99.7761	0
MEAN OF CROSS V. (%)	98.9900	31.0300
TRAINING TIME(SECOND)	122.242404	127.661013
TESTING TIME(SECOND)	32.621607	37.999987
MEMORY(GB)	0.977589	1.094322

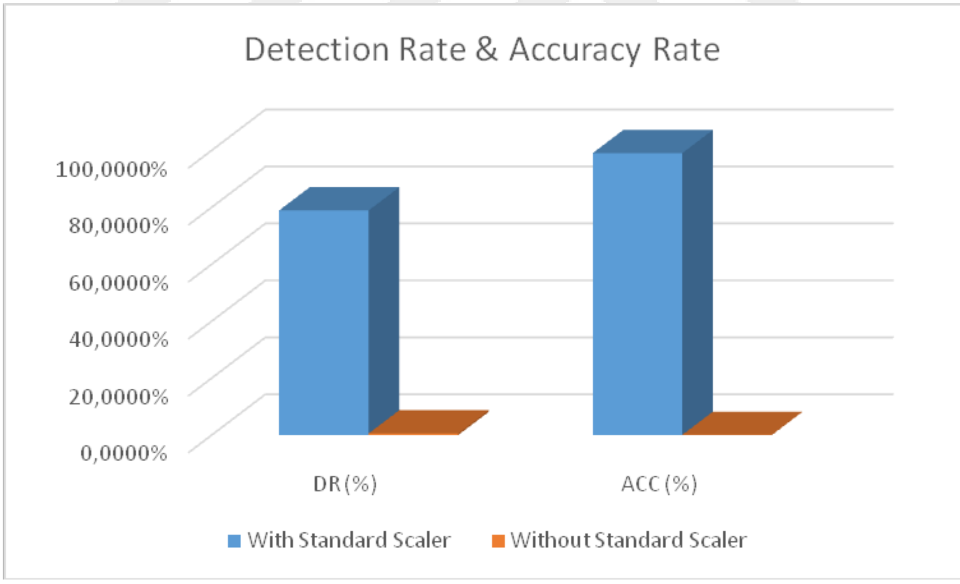

Figure 4.47: DR & ACC Comparison for DL Algorithm

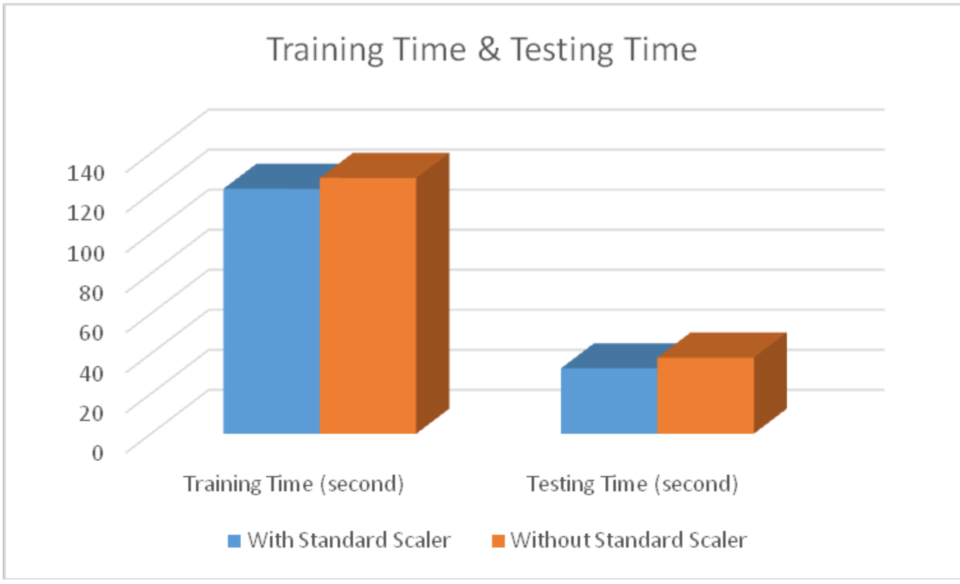

Figure 4.48: Training and Testing Time Comparison for DL Algorithm

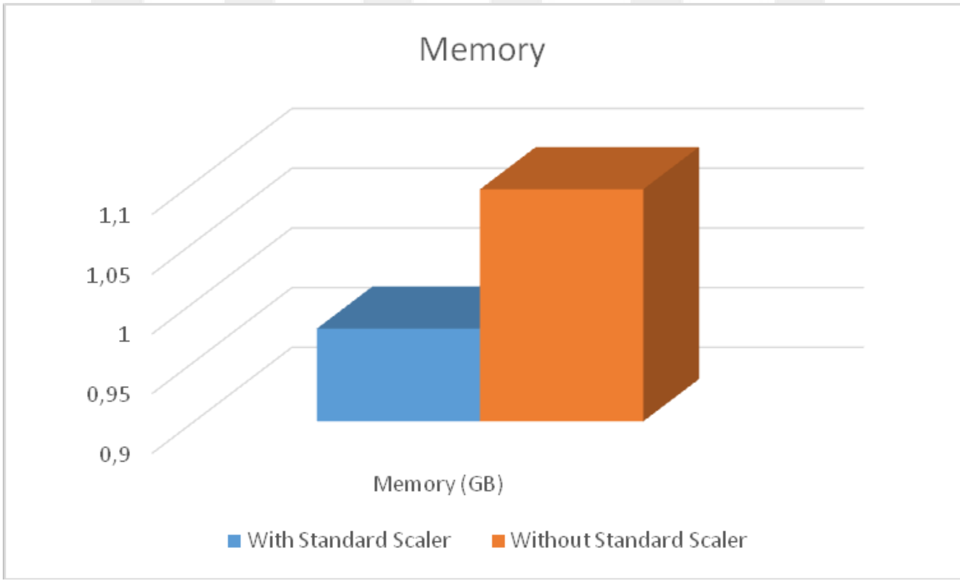

Figure 4.49: Memory Consume Comparison for DL Algorithm

Table 4.17: Standard Scaler Experiments for GNB Algorithm

	With Standard Scaler	Without Standard Scaler
DR (%)	51.4787	51.6904
ACC (%)	84.0907	67.7983
DR for WWW (%)	95.4771	69.1700
DR for MAIL (%)	49.8243	98.8250
DR for OTHER (%)	26.0731	38.4234
DR for INTERACTIVE (%)	46.7619	33.7258
DR for BULK (%)	20.9560	13.8846
DR for SERVICE (%)	97.1204	97.4592
DR for MEDIA (%)	24.1379	10.3448
MEAN OF CROSS V. (%)	75.5659	69.3296
TRAINING TIME(SECOND)	0.156394	0.156402
TESTING TIME(SECOND)	2.362087	2.348045
MEMORY(GB)	0.758854	0.846127

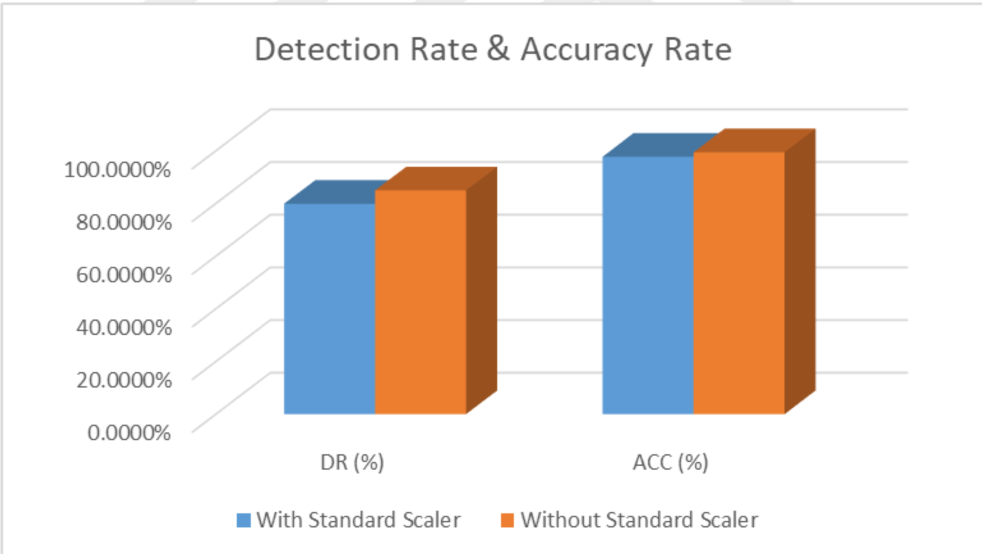

Figure 4.50: DR & ACC Comparison for GNB Algorithm

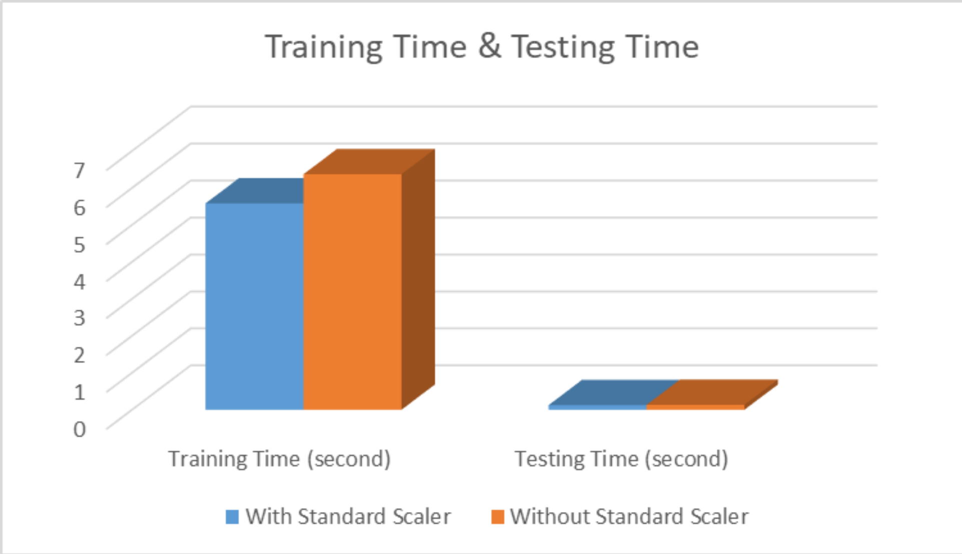

Figure 4.51: Training and Testing Time Comparison for GNB Algorithm

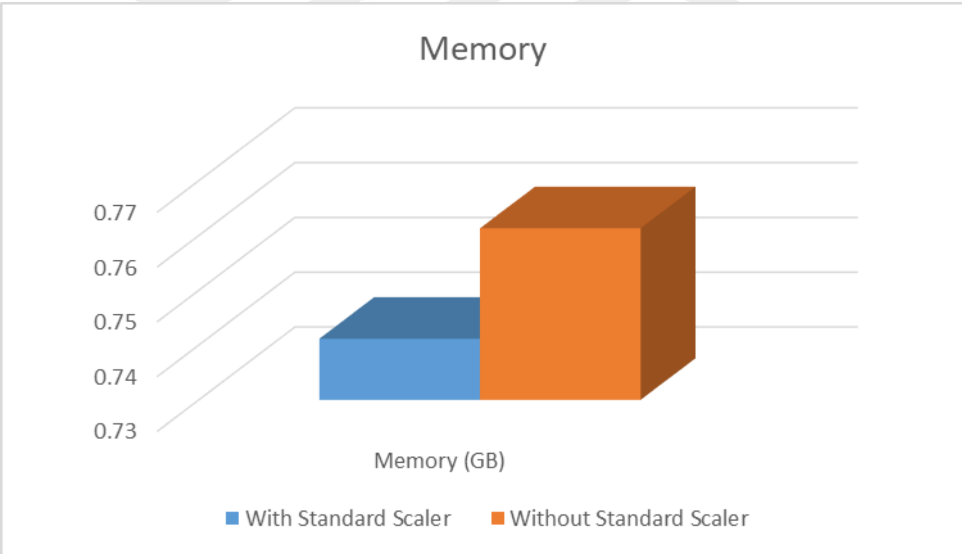

Figure 4.52: Memory Consume Comparison for GNB Algorithm

Scenario 4:

K-fold cross validation is applied to the algorithm to understand the impact of data unbalance and each class accuracy is calculated in each experiment.

Model performance is evaluated in machine learning based on an error metric to determine the model's accuracy. This evaluation is not very accurate since the accuracy acquired for one test set may vary greatly from the accuracy acquired for another test set. Kfold cross validation solves evaluation problem by dividing the dataset into folds and making sure that each fold is used as a testing set at some point. Figure 4.53 shows the K-fold cross validation algorithm:

1. Divide the dataset into k equal parts.
2. Use k-1 parts for training and 1 part for testing.
3. Repeat the procedure k times, rotating the test dataset.
4. Determine a performance metric for all iterations.

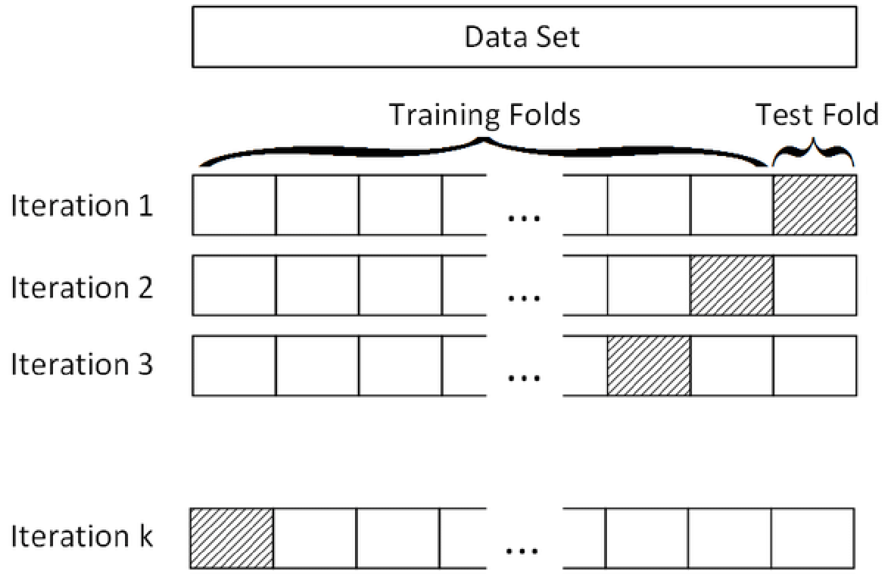

Figure 4.53: K-Fold Cross Validation

4.5 Discussion

Specifically, the following parameters are discussed in terms of memory allocation, classification speed and system accuracy to explain the outcomes of the experiment.

4.5.1 System Accuracy Results

The DT algorithm has greater efficiency compared to other algorithms, where its classification accuracy is 97.6861 %. The aim of our research is to classify network packets while enhancing the generation of DR rate and ACC rate. Moreover, Figure 4.54 shows that our system can classify data set vectors at an elevated average DR rate of 74.2333%.

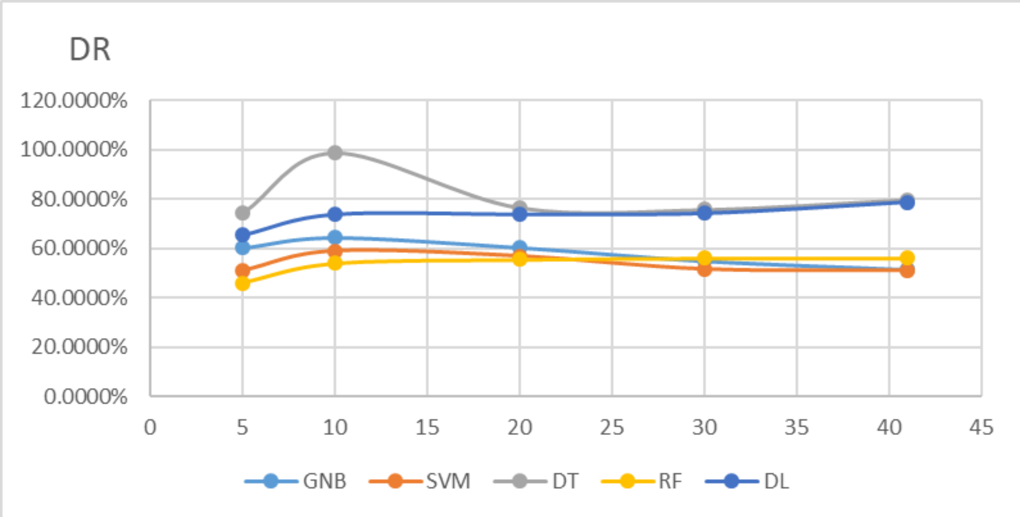

Figure 4.54: Number of Features versus Detection Rate

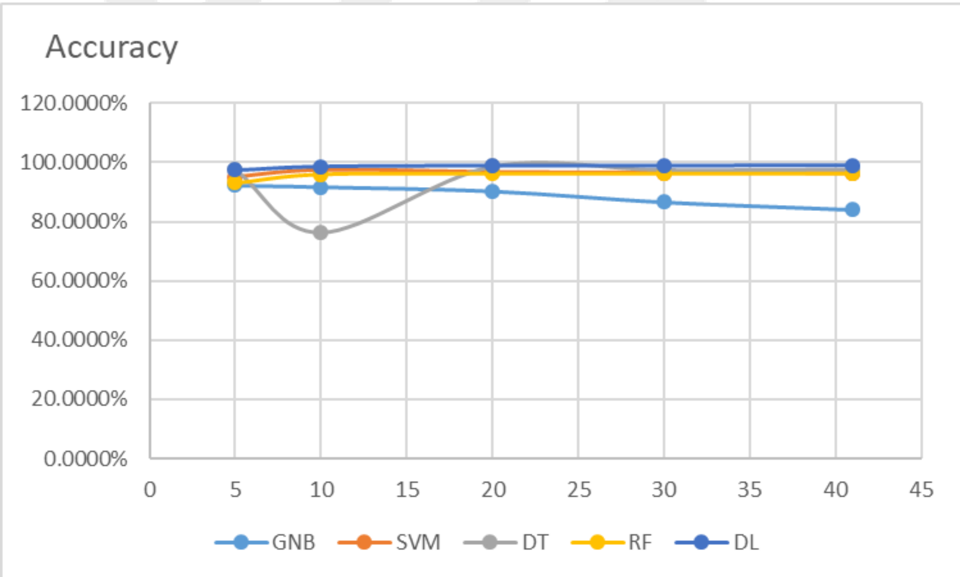

Figure 4.55: Number of Features versus Accuracy Rate

4.5.2 Classification Speed Results

Figure 4.56 indicates the duration of the training versus the number of features. DT algorithm second lowest training duration and DT is one of the algorithms that has lowest testing duration that was shown in Figure 4.57.

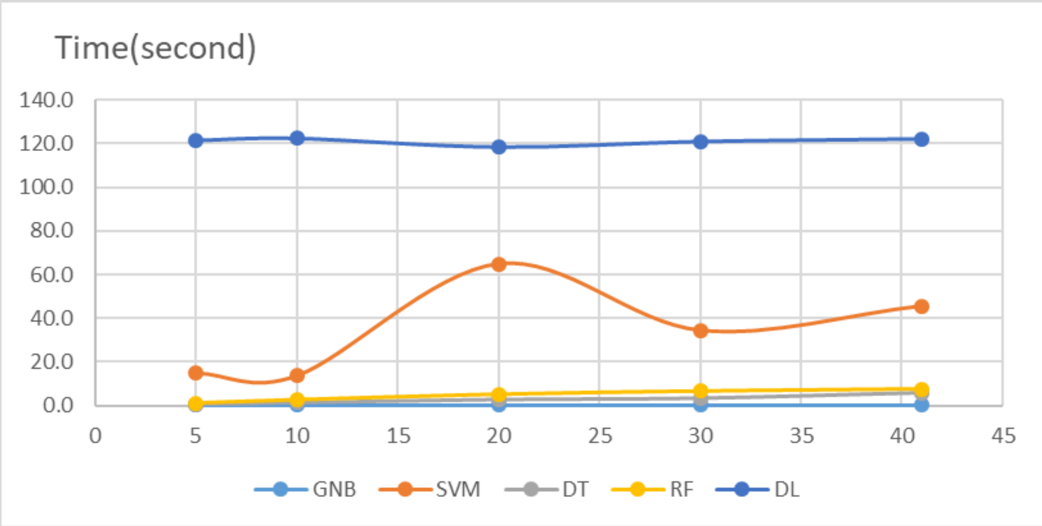

Figure 4.56: Number of Features versus Training Time

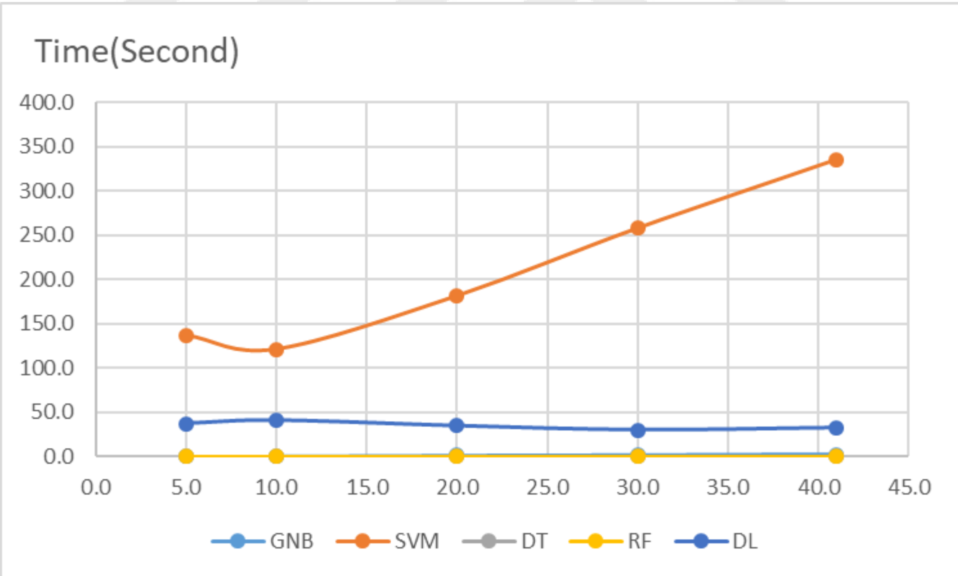

Figure 4.57: Number of Features versus Testing Time

4.5.3 Memory Allocation Results

The memory usage versus number of features is indicated in Figure 4.52. The DT algorithm utilizes 5 features that uses 0.569656 GB, while it uses 41 features using time 0.741154.

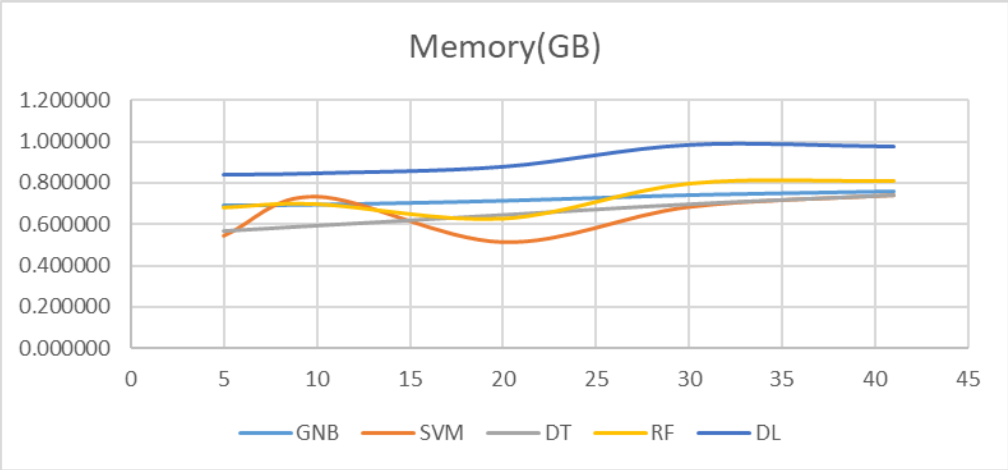

Figure 4.58: Features versus Memory Allocation

The number of instances is an effective factor on the percentage of the classification accuracy and the training and testing time. In terms of training time, the suggested NMS using the DT algorithm outperforms all algorithms as it has the minimum time.

The suggested NMS using the DT algorithm outperforms all other algorithms with regard to DR 79.8813 percent results and it has the largest speed in the comparison lists.

The proposed NMS using the DT algorithm has the second highest ACC rate of 97.7486 %. The proposed NMS achieves best performance in terms of ACC rate, DR rate and highest speed.

4.5.3 Other Algorithms Results

Screenshots of experiments for SVM, LR, DL and GNB are shown Figure 4.59, Figure 4.60, Figure 4.61 and Figure 4.62. It can be seen that DL gives closest results to the DT algorithm.

```
Support Vector Machine Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 41
Time taken to perform 5-fold cross validation : 203.223855
Cross validation scores :
[0.9407986  0.96073594 0.95985608 0.97835921 0.96149496]
Mean score of 5-fold cross validation : 0.960249
Time taken to train final model : 45.529160
Predictions made using final model...
Time taken to make predictions on test data : 335.489777
Memory used : 0.743065 GB  CPU usage : 25.500000
Confusion matrix :
[[475692    684    153      0    400      0      0]
 [  7825  42391     46      0    974      0      0]
 [   140     20  38685      0   1154      0      0]
 [    14    370     79     14    712      0      0]
 [    20    196   4629      0  16848      0      0]
 [     5      0   4718      0      0      0      0]
 [     0      0     18      0     11      0      0]]
Accuracy score on test data is : 0.962793
For WWW, Detection Rate is % 99.74063225343814
For MAIL, Detection Rate is % 82.73674759934421
For OTHER, Detection Rate is % 96.71491787294683
For INTERACTIVE, Detection Rate is % 1.1774600504625736
For BULK, Detection Rate is % 77.66560641681649
For SERVICE, Detection Rate is % 0.0
For MEDIA, Detection Rate is % 0.0
DR is % 51.14790917042975
```

Figure 4.59: Experiment of Support Vector Machine Classification Algorithm

```
Logistic Regression Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 41
Time taken to perform 5-fold cross validation : 30.019858
Cross validation scores :
[0.93190811 0.96212355 0.95970188 0.97090573 0.96092947]
Mean score of 5-fold cross validation : 0.957114
Time taken to train final model : 7.594568
Predictions made using final model...
Time taken to make predictions on test data : 0.102367
Memory used : 0.813011 GB  CPU usage : 17.200000
Confusion matrix :
[[475100   1530    154     40    103      2      0]
 [  7313  43510     98     51    263      1      0]
 [   150    788  37032     39    568   1422      0]
 [    35    873     75    137     66      3      0]
 [    82   1426   4539     12  15623     11      0]
 [     1      0   3200      0     24   1498      0]
 [     0      9      6     12      2      0      0]]
Accuracy score on test data is : 0.961568
For WWW, Detection Rate is % 99.61650476276343
For MAIL, Detection Rate is % 84.9207588414396
For OTHER, Detection Rate is % 92.58231455786395
For INTERACTIVE, Detection Rate is % 11.52228763666947
For BULK, Detection Rate is % 72.01862351910755
For SERVICE, Detection Rate is % 31.717128943468136
For MEDIA, Detection Rate is % 0.0
DR is % 56.05394546590174
```

Figure 4.60: Experiment of Logistic Regression Classification Algorithm

```
Deep Learning Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 41
Time taken to perform 5-fold cross validation : 474.372798
Mean score of 5-fold cross validation: 98.99%
Time taken to train final model : 122.242404
Predictions made using final model...
Time taken to make predictions on test data : 32.621607
Memory used : 0.977589 GB  CPU usage : 59.500000
Confusion matrix :
[[ 21242      6     19      7     23      8     10]
 [    23    762     15      0     23      0     18]
 [   230    301  50300      8    569      2    802]
 [     0      0      0      0      0      0      0]
 [   121     66    129      2  37334    644      3]
 [     1      0      0      0   1897   4069      9]
 [    76     54    773     12    153      0 476087]]
Accuracy score on test data is : 0.989923
For WWW, Detection Rate is % 99.65751817968567
For MAIL, Detection Rate is % 90.6064209274673
For OTHER, Detection Rate is % 96.33800658852371
For INTERACTIVE, Detection Rate is % 0
For BULK, Detection Rate is % 97.4803519674143
For SERVICE, Detection Rate is % 68.08902275769745
For MEDIA, Detection Rate is % 99.77617336085758
DR is % 78.84964196880658
```

Figure 4.61: Experiment of Deep Learning Algorithm

```
Gaussian Naive Bayes Classifier
Train and Test Data read...
Service type mapping created...
There are 7 classes to be monitored
Classes are:
['WWW', 'MAIL', 'OTHER', 'INTERACTIVE', 'BULK', 'SERVICE', 'MEDIA']
Train and Test data labels created...
Decomposed features created...
Number of features used : 41
Time taken to perform 5-fold cross validation : 1.283931
Cross validation scores :
[0.73513541 0.74134032 0.83726548 0.77110106 0.69345054]
Mean score of 5-fold cross validation : 0.755659
Time taken to train final model : 0.156394
Predictions made using final model...
Time taken to make predictions on test data : 2.362087
Memory used : 0.758854 GB  CPU usage : 36.400000
Confusion matrix :
[[455358   4787    443   2466    141  11925   1809]
 [ 21977  25528    985    664     76   1509    497]
 [ 13031    644  10429    180     28  14899    788]
 [     0    167     14    556    218     31    203]
 [     3   7528    294    266   4546   8514    542]
 [     0      0    101      5     12   4587     18]
 [     0      5      1      7      6      3      7]]
Accuracy score on test data is : 0.840907
For WWW, Detection Rate is % 95.47710455854016
For MAIL, Detection Rate is % 49.824342259348896
For OTHER, Detection Rate is % 26.07315182879572
For INTERACTIVE, Detection Rate is % 46.76198486122792
For BULK, Detection Rate is % 20.956068777946804
For SERVICE, Detection Rate is % 97.12047427482533
For MEDIA, Detection Rate is % 24.137931034482758
DR is % 51.478722513595365
```

Figure 4.62: Experiment of Gaussian Naïve Bayes Algorithm

CHAPTER 5

CONCLUSION

5.1 Result

Network monitoring addresses all level of network operation from basic connectivity to application throughput. The aim of this paper is to suggest a Network Monitoring System using Machine Learning that assists to classify network packages. The proposed NMS utilizes the DT algorithm for classification and PCA algorithm for dimension (feature) reduction, and it classifies connections by network classes.

Experimental results suggested that using the DT and PCA algorithms, the suggested NMS system reached a high classification ACC rate of 97,7486 %. Compared to all other algorithms, it is the best performance.

The DT algorithm exceeds all other training algorithms as it has the minimum time for execution. But it was not possible to check how will the data behaves on a larger network and some service types are classified as 'other' due to they were belong to many classes. As a future study, the suggested NMS system can also be analyzed using other classification algorithms or proposed NMS system can be modified to capture network packets without cooperation of Wireshark and Npcap Library or other data sets can be experimented with the suggested NMS system.

REFERENCES

Auld, T., Moore, A. W., & Gull, S. F. J. I. T. o. n. n. (2007). Bayesian neural networks for internet traffic classification. 18(1), 223-239.

Cerf, V., & Kahn, R. J. I. T. o. c. (1974). A protocol for packet network intercommunication. 22(5), 637-648.

Cortes, C., & Vapnik, V. J. M. l. (1995). Support-vector networks. 20(3), 273-297.

Crotti, M., Gringoli, F., Pelosato, P., & Salgarelli, L. (2006). A statistical approach to IP-level classification of network traffic. Paper presented at the 2006 IEEE International Conference on Communications.

Degermark, M. (1999). IP header compression.

Haffner, P., Sen, S., Spatscheck, O., & Wang, D. (2005). ACAS: automated construction of application signatures. Paper presented at the Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data.

Hinton, G. E., Sejnowski, T. J., & Poggio, T. A. (1999). Unsupervised learning: foundations of neural computation: MIT press.

Hosmer, D., & Lemeshow, S. J. N. Y., NY, US. (2000). Applied logistic regression. 2nd edWiley.

Ioffe, S., & Szegedy, C. J. a. p. a. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., & Faloutsos, M. J. U. o. C., Riverside, USA, Tech. Rep. (2003). File-sharing in the Internet: A characterization of P2P traffic in the backbone.

Karagiannis, T., Broido, A., & Faloutsos, M. (2004). Transport layer identification of P2P traffic. Paper presented at the Proceedings of the 4th ACM SIGCOMM conference on Internet measurement.

Karagiannis, T., Papagiannaki, K., & Faloutsos, M. (2005). BLINC: multilevel traffic classification in the dark. Paper presented at the ACM SIGCOMM computer communication review.

Kim, H., Claffy, K. C., Fomenkov, M., Barman, D., Faloutsos, M., & Lee, K. (2008). Internet traffic classification demystified: myths, caveats, and the best practices. Paper presented at the Proceedings of the 2008 ACM CoNEXT conference.

Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. J. E. a. i. a. i. c. e. (2007). Supervised machine learning: A review of classification techniques. 160, 3-24.

LeCun, Y., Bengio, Y., & Hinton, G. J. n. (2015). Deep learning. 521(7553), 436.

Li, W., & Moore, A. W. (2007). A machine learning approach for efficient traffic classification. Paper presented at the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems.

Lim, Y.-s., Kim, H.-c., Jeong, J., Kim, C.-k., Kwon, T. T., & Choi, Y. (2010). Internet traffic classification demystified: on the sources of the discriminative power. Paper presented at the Proceedings of the 6th International COnference.

Lou, W., Wang, X., Chen, F., Chen, Y., Jiang, B., & Zhang, H. J. P. o. (2014). Sequence based prediction of DNA-binding proteins based on hybrid feature selection using random forest and Gaussian naive Bayes. 9(1), e86703.

Moore, A., Hall, J., Kreibich, C., Harris, E., & Pratt, I. (2003). Architecture of a network monitor. Paper presented at the Passive & Active Measurement Workshop.

Moore, A. W., & Papagiannaki, K. (2005). Toward the accurate identification of network applications. Paper presented at the International Workshop on Passive and Active Network Measurement.

Namdev, N., Agrawal, S., & Silkari, S. J. P. C. S. (2015). Recent advancement in machine learning based internet traffic classification. 60, 784-791.

Nguyen, T. T., Armitage, G. J. J. I. C. S., & Tutorials. (2008). A survey of techniques for internet traffic classification using machine learning. 10(1-4), 56-76.

Park, J., Tyan, H.-R., & Kuo, C.-C. J. (2006). Internet traffic classification for scalable qos provision. Paper presented at the 2006 IEEE International Conference on Multimedia and Expo.

Porter, T. J. S. F. (2005). The perils of deep packet inspection. 6.

Schmidhuber, J. J. N. n. (2015). Deep learning in neural networks: An overview. 61, 85-117.

Schneider, P. J. D. O. A. S., Cambridge, MA. (1996). TCP/IP traffic Classification Based on port numbers. 2138(5).

Scholkopf, B., Sung, K.-K., Burges, C. J., Girosi, F., Niyogi, P., Poggio, T., & Vapnik, V. J. I. t. o. S. P. (1997). Comparing support vector machines with Gaussian kernels to radial basis function classifiers. 45(11), 2758-2765.

Sen, S., Spatscheck, O., & Wang, D. (2004). Accurate, scalable in-network identification of p2p traffic using application signatures. Paper presented at the Proceedings of the 13th international conference on World Wide Web.

Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms: Cambridge university press.

Stallings, W. J. U. S. R., New Jersey, USA. (2003). Network Security Essentials-Applications and Standards Pearson Education.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. Paper presented at the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications.

Webb, G. I., Boughton, J. R., & Wang, Z. J. M. l. (2005). Not so naive Bayes: aggregating one-dependence estimators. 58(1), 5-24.

Wolpert, D. H., & Macready, W. G. J. I. t. o. e. c. (1997). No free lunch theorems for optimization. 1(1), 67-82.

Wu, Y., Min, G., Li, K., & Javadi, B. (2009). Performance analysis of communication networks in multi-cluster systems under bursty traffic with communication locality. Paper presented at the GLOBECOM 2009-2009 IEEE Global Telecommunications Conference.

Xiang, Y., Zhou, W., Guo, M. J. I. T. o. P., & Systems, D. (2008). Flexible deterministic packet marking: An ip traceback system to find the real source of attacks. 20(4), 567-580.

Zhou, Z.-H., Li, H., & Yang, Q. (2007). Advances in Knowledge Discovery and Data Mining: 11th Pacific-Asia Conference, PAKDD 2007, Nanjing, China, May 22-25, 2007, Proceedings (Vol. 4426): Springer.

APPENDICES

Appendix: A

Attributes description of KDD CUP 99 dataset.

No.	Network attributes	Description	Type
1	Duration	Duration of the connection in second.	Continuous
2	Protocol type	Connection protocol (e.g. TCP, UDP, ICMP).	Discrete
3	Service	Destination service (e.g. telnet, ftp, http, pop3…)	Discrete
4	Flag	Status flag of the connection (e.g. REJ, SF, S0…)	Discrete
5	Src bytes	Bytes sent from source to destination	Continuous
6	dst bytes	Bytes sent from destination to source	Continuous
7	Land	1 if connection is from /to the same host/port; 0 otherwise	Discrete
8	Wrong fragment	Number of wrong fragments	Continuous
9	Urgent	Number of urgent packets	Continuous
10	Hot	Number of" hot" indicators	Continuous
11	Num failed logins	Number of failed logins	Continuous
12	Logged in	1 if successfully logged in; 0 otherwise	Discrete
13	Num compromised	Number of "compromised" conditions	Continuous
14	Root shell	1 if root shell is obtained; 0 otherwise	Continuous
15	Su attempted	1 if "as root" command attempted; 0 otherwise	Continuous

No.	Network attributes	Description	Type
16	Num_root	Number of root accesses	Continuous
17	Num_file_creations	Number of file creation operations	Continuous
18	Num_shells	Number of shell prompts	Continuous
19	Num_access_files	Number of operations on access control files	Continuous
20	Num_outbound_cmds	Number of outbound commands in an ftp session	Continuous
21	Is_host_login	1 if the login belongs to the "hot" list; 0 otherwise	Discrete
22	Is_guest_login	1 if the login is a " guest" login; 0 otherwise	Discrete
23	count	Number of connections to the same host as the current connection in the past two seconds	Continuous
24	Srv_ count	Number of connections to the same service as current connection in the past two seconds	Continuous
25	Serror_rate	No. of connections that have " SYN" errors	Continuous
26	Srv_ Serror_rate	No. of connections that have " SYN" errors	Continuous
27	Rerror_rate	No. of connections that have " REJ" errors	Continuous
28	Srv_ rerror_rate	No. of connections that have " REJ" errors	Continuous
29	Same_ rerror_rate	No. of connections to the same service	Continuous
30	Diff_ Srv_ rate	No. of connections to different services	Continuous
31	Srv_ Diff_host_ rate	No. of connections to different hosts	Continuous
32	Dst_ host_count	Count of connections having the same destination host	Continuous
33	Dst_ host_ Srv_count	Count of connections having the same destination host and using the same service	Continuous

No.	Network attributes	Description	Type
34	Dst host Same rate	Count of connections having the same destination host and using the same service	Continuous
35	Dst_ host_Diff_ Srv _ rate	No. of different services on the current host	Continuous
36	Dst_ host_ Same_src_port_ rate	No. of connections to current host having the same src port	Continuous
37	Dst_ host_ Srv _ Diff_ host_ rate	No. of connections to same service coming from different hosts	Continuous
38	Dst host serror rate	No. of connections to the current host that have an S0 error	Continuous
39	Dst_ host_ Srv _serror_ rate	No. of connections to the current host and specified service that have an S0 error	Continuous
40	Dst host rerror rate	No. of connections to the current host that have an RST error	Continuous
41	Dst host rerror rate	No. of connections to the current host and specified service that have an RST error	Continuous

Appendix: B

Content of Field Names.csv file.

ftp_data	BULK
other	OTHER
private	OTHER
http	WWW
remote_job	REMOTE
name	OTHER
netbios_ns	WWW
eco_i	OTHER
mtp	MAIL
telnet	INTERACTIVE
finger	OTHER
domain_u	OTHER
supdup	OTHER
uucp_path	BULK
Z39_50	DATABASE
smtp	MAIL
csnet_ns	OTHER
uucp	BULK
netbios_dgm	OTHER
urp_i	OTHER
auth	OTHER
domain	OTHER
ftp	BULK
bgp	SERVICE
ldap	SERVICE
ecr_i	SERVICE
gopher	BULK
vmnet	OTHER
systat	INTERACTIVE
http_443	WWW
efs	INTERACTIVE
whois	INTERACTIVE
imap4	MAIL
iso_tsap	DATABASE
echo	INTERACTIVE
klogin	INTERACTIVE
link	SERVICE
sunrpc	REMOTE
login	INTERACTIVE
kshell	REMOTE
sql_net	DATABASE

time	SERVICE
hostnames	SERVICE
exec	MAIL
ntp_u	SERVICE
discard	OTHER
nntp	WWW
courier	OTHER
ctf	INTERACTIVE
ssh	INTERACTIVE
daytime	SERVICE
shell	INTERACTIVE
netstat	INTERACTIVE
pop_3	MAIL
nnsp	OTHER
IRC	WWW
pop_2	MAIL
printer	INTERACTIVE
tim_i	OTHER
pm_dump	DATABASE
red_i	OTHER
netbios_ssn	OTHER
rje	REMOTE
X11	MEDIA
urh_i	OTHER
http_8001	WWW
aol	WWW
http_2784	WWW
tftp_u	BULK
harvest	OTHER

Appendix: C

Content of Service Types.csv file.

ftp_data,BULK
other,OTHER
private,OTHER
http,WWW
remote_job,REMOTE
name,OTHER
netbios_ns,WWW
eco_i,OTHER
mtp,MAIL
telnet,INTERACTIVE
finger,OTHER
domain_u,OTHER
supdup,OTHER
uucp_path,BULK
Z39_50,DATABASE
smtp,MAIL
csnet_ns,OTHER
uucp,BULK
netbios_dgm,OTHER
urp_i,OTHER
auth,OTHER
domain,OTHER
ftp,BULK
bgp,SERVICE
ldap,SERVICE
ecr_i,SERVICE
gopher,BULK
vmnet,OTHER
systat,INTERACTIVE
http_443,WWW
efs,INTERACTIVE
whois,INTERACTIVE

imap4,MAIL
iso_tsap,DATABASE
echo,INTERACTIVE
klogin,INTERACTIVE
link,SERVICE
sunrpc,REMOTE
login,INTERACTIVE
kshell,REMOTE
sql_net,DATABASE
time,SERVICE
hostnames,SERVICE
exec,MAIL
ntp_u,SERVICE
discard,OTHER
nntp,WWW
courier,OTHER
ctf,INTERACTIVE
ssh,INTERACTIVE
daytime,SERVICE
shell,INTERACTIVE
netstat,INTERACTIVE
pop_3,MAIL
nnsp,OTHER
IRC,WWW
pop_2,MAIL
printer,INTERACTIVE
tim_i,OTHER
pm_dump,DATABASE
red_i,OTHER
netbios_ssn,OTHER
rje,REMOTE
X11,MEDIA
urh_i,OTHER
http_8001,WWW
aol,WWW

http_2784,WWW
tftp_u,BULK
harvest,OTHER

Appendix: D

Python Code of Support Vector Machine Algorithm

```python
import pandas as pd
import numpy as np
import time
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import psutil
import os


print("Support Vector Machine Classifier")
train = pd.read_csv('./KDDTrain.csv', header=None)
test = pd.read_csv('./KDDTest.csv', header=None)
print("Train and Test Data read...")


# Reads "Field Names.csv". Use this to set the names of train and test data columns
columns = pd.read_csv('Field Names.csv', header=None)
columns.columns = ['name', 'type']
train.columns = columns['name']
test.columns = columns['name']


# Read Service Types.csv
# Use this to create a mapping from service types to final labels (MAIL, WWW, FTP, DATABASE, v.b.)
serviceType = pd.read_csv('Service Types.csv', header=None)
serviceType.columns = ['Name', 'Type']
serviceMap={}



# Creates attackMap map which contains a mapping between service type and the final label
for i in range(len(serviceType)):
    serviceMap[serviceType['Name'][i]] = serviceType['Type'][i]


print("Service type mapping created...")





# Add a new variable called 'label' which contains the final label
train['label'] = train['service'].map(serviceMap)
test['label'] = test['service'].map(serviceMap)
```

```
#find count of classes from serviceType
classesCount = len(train['label'].drop_duplicates())
classesName = train['label'].drop_duplicates().values.tolist()
print('There are ' +  str(classesCount) + ' classes to be monitored')
print('Classes are:')
print(classesName)



# The variable 'label' is stored in different variables
# This is required to keep the dependent variable separate from the independent variable
trainLabel = train['label']
testLabel = test['label']




# service_type and label variables are removed from the train and test data
# so only features are remained in train and test data
train.drop(['service', 'label'], axis=1, inplace=True)
test.drop(['service', 'label'], axis=1, inplace=True)
print("Train and Test data labels created...")


# Transform the existing nominal variables into the integer coded variables using the LabelEncoder
for col in ['protocol_type', 'flag', 'attack_type']:
    le = LabelEncoder()
    le.fit(train[col])
    train[col] = le.transform(train[col])
    le1 = LabelEncoder()
    le1.fit(test[col])
    test[col] = le1.transform(test[col])



# for Stadart Scaler
scaler = MinMaxScaler()   #scale between 0 and 1
train = scaler.fit_transform(train)
test = scaler.fit_transform(test)
total = np.concatenate([train, test] )




# Decomposition features are generated for both train and test data
pca = PCA(n_components=41, random_state=100)
pca.fit(total)
train = pca.transform(train)
test = pca.transform(test)
```

```python
print("Decomposed features created...")
print("Number of features used : %d" % train.shape[1])






# Performing 5-fold Cross validation
startTime = time.clock()
SVC1 = SVC(kernel = 'rbf')
score = cross_val_score(SVC1, train, trainLabel, cv=5)
endTime = time.clock()
print("Time taken to perform 5-fold cross validation : %f" % (endTime-startTime))
print("Cross validation scores : ")
print(score)
print("Mean score of 5-fold cross validation : %f" % score.mean())



# Final Testing and Evaluate Performance
# Train the KNN classifier model by original train data and got optimized parameter
startTime = time.clock()
SVC2 = SVC(kernel = 'rbf')
SVC2.fit(train, trainLabel)
endTime = time.clock()
print("Time taken to train final model : %f" % (endTime-startTime))
print("Predictions made using final model...")


# Predictions for test data and evaluate its performance
startTime = time.clock()
pred = SVC2.predict(test)
endTime = time.clock()
cpuUsage = psutil.cpu_percent()
pid = os.getpid()
py = psutil.Process(pid)
memoryUse = py.memory_info()[0] / 2. ** 30
print("Time taken to make predictions on test data : %f" % (endTime-startTime))
print("Memory used : %f GB  CPU usage : %f" % (memoryUse, cpuUsage))



con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels = classesName)
print("Confusion matrix : ")
print(con_matrix)
```

```
# Print accuracy and detection rate
acc = accuracy_score(y_pred=pred, y_true=testLabel)
print("Accuracy score on test data is : %f" % acc)

sumDr = 0
for i in range(con_matrix.shape[0]):
    det_rate = 0
    for j in range(con_matrix.shape[1]):
        if i != j :
            det_rate += con_matrix[i][j]
    if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i])  != 0:
        det_rate =100* con_matrix[i][i]/(det_rate + con_matrix[i][i])
        sumDr += det_rate
        print("For " + classesName[i] + ", Detection Rate is % " + str(det_rate))
    else:
        print("For " + classesName[i] + ", Detection Rate is % 0")

DR = sumDr/classesCount
print("DR is % " + str(DR))
```

Appendix: E

Python Code of Decision Tree Algorithm

```python
import pandas as pd
import numpy as np
import time
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn import tree
from sklearn.model_selection import cross_val_score
import psutil
import os

print("Decision Tree Classifier")
train = pd.read_csv('./KDDTrain.csv', header=None)
test = pd.read_csv('./KDDTest.csv', header=None)
print("Train and Test Data read...")



# Reads "Field Names.csv". Use this to set the names of train and test data columns
columns = pd.read_csv('Field Names.csv', header=None)
columns.columns = ['name', 'type']
train.columns = columns['name']
test.columns = columns['name']


# Read Service Types.csv
# Use this to create a mapping from service types to final labels (MAIL, WWW, FTP, DATABASE, v.b.)
serviceType = pd.read_csv('Service Types.csv', header=None)
serviceType.columns = ['Name', 'Type']
serviceMap={}



# Creates attackMap map which contains a mapping between service type and the final label
for i in range(len(serviceType)):
    serviceMap[serviceType['Name'][i]] = serviceType['Type'][i]


print("Service type mapping created...")



# Add a new variable called 'label' which contains the final label
train['label'] = train['service'].map(serviceMap)
test['label'] = test['service'].map(serviceMap)
```

```
#find count of classes from serviceType
classesCount = len(train['label'].drop_duplicates())
classesName = train['label'].drop_duplicates().values.tolist()
print('There are ' +  str(classesCount) + ' classes to be monitored')
print('Classes are:')
print(classesName)



# The variable 'label' is stored in different variables
# This is required to keep the dependent variable separate from the independent variable
trainLabel = train['label']
testLabel = test['label']




# service_type and label variables are removed from the train and test data
# so only features are remained in train and test data
train.drop(['service', 'label'], axis=1, inplace=True)
test.drop(['service', 'label'], axis=1, inplace=True)
print("Train and Test data labels created...")


# Transform the existing nominal variables into the integer coded variables using the LabelEncoder
for col in ['protocol_type', 'flag', 'attack_type']:
    le = LabelEncoder()
    le.fit(train[col])
    train[col] = le.transform(train[col])
    le1 = LabelEncoder()
    le1.fit(test[col])
    test[col] = le1.transform(test[col])


# for Stadart Scaler
scaler = MinMaxScaler()   #scale between 0 and 1
train = scaler.fit_transform(train)
test = scaler.fit_transform(test)
total = np.concatenate([train, test] )

```

```python
# Decomposition features are generated for both train and test data
pca = PCA(n_components=41, random_state=100)
pca.fit(total)
train = pca.transform(train)
test = pca.transform(test)



print("Decomposed features created...")
print("Number of features used : %d" % train.shape[1])






# Performing 5-fold Cross validation
startTime = time.clock()
DT1 = tree.DecisionTreeClassifier(random_state = 0)
score = cross_val_score(DT1, train, trainLabel, cv=5)
endTime = time.clock()
print("Time taken to perform 5-fold cross validation : %f" % (endTime-startTime))
print("Cross validation scores : ")
print(score)
print("Mean score of 5-fold cross validation : %f" % score.mean())


# Final Testing and Evaluate Performance
# Train the KNN classifier model by original train data and got optimized parameter
startTime = time.clock()
DT2 = tree.DecisionTreeClassifier(random_state = 0)
DT2.fit(train, trainLabel)
endTime = time.clock()
print("Time taken to train final model : %f" % (endTime-startTime))
print("Predictions made using final model...")


# Predictions for test data and evaluate its performance
startTime = time.clock()
pred = DT2.predict(test)
endTime = time.clock()
cpuUsage = psutil.cpu_percent()
pid = os.getpid()
py = psutil.Process(pid)
memoryUse = py.memory_info()[0] / 2. ** 30
print("Time taken to make predictions on test data : %f" % (endTime-startTime))
print("Memory used : %f GB  CPU usage : %f" % (memoryUse, cpuUsage))

```

```python
con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels = classesName)
print("Confusion matrix : ")
print(con_matrix)


# Print accuracy and detection rate
acc = accuracy_score(y_pred=pred, y_true=testLabel)
print("Accuracy score on test data is : %f" % acc)

sumDr = 0
for i in range(con_matrix.shape[0]):
    det_rate = 0
    for j in range(con_matrix.shape[1]):
        if i != j :
            det_rate += con_matrix[i][j]
    if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i])  != 0:
        det_rate =100* con_matrix[i][i]/(det_rate + con_matrix[i][i])
        sumDr += det_rate
        print("For " + classesName[i] + ", Detection Rate is % " + str(det_rate))
    else:
        print("For " + classesName[i] + ", Detection Rate is % 0")

DR = sumDr/classesCount
print("DR is % " + str(DR))

```

Appendix: F

Python Code of Logistic Regression Algorithm

```
import pandas as pd
import numpy as np
import time
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
import psutil
import os


print("Logistic Regression Classifier")
train = pd.read_csv('./KDDTrain.csv', header=None)
test = pd.read_csv('./KDDTest.csv', header=None)
print("Train and Test Data read...")



# Reads "Field Names.csv". Use this to set the names of train and test data columns
columns = pd.read_csv('Field Names.csv', header=None)
columns.columns = ['name', 'type']
train.columns = columns['name']
test.columns = columns['name']


# Read Service Types.csv
# Use this to create a mapping from service types to final labels (MAIL, WWW, FTP, DATABASE, v.b.)
serviceType = pd.read_csv('Service Types.csv', header=None)
serviceType.columns = ['Name', 'Type']
serviceMap={}




# Creates attackMap map which contains a mapping between service type and the final label
for i in range(len(serviceType)):
    serviceMap[serviceType['Name'][i]] = serviceType['Type'][i]




print("Service type mapping created...")
```

```python
# Add a new variable called 'label' which contains the final label
train['label'] = train['service'].map(serviceMap)
test['label'] = test['service'].map(serviceMap)



#find count of classes from serviceType
classesCount = len(train['label'].drop_duplicates())
classesName = train['label'].drop_duplicates().values.tolist()
print('There are ' +  str(classesCount) + ' classes to be monitored')
print('Classes are:')
print(classesName)



# The variable 'label' is stored in different variables
# This is required to keep the dependent variable separate from the independent variable
trainLabel = train['label']
testLabel = test['label']




# service_type and label variables are removed from the train and test data
# so only features are remained in train and test data
train.drop(['service', 'label'], axis=1, inplace=True)
test.drop(['service', 'label'], axis=1, inplace=True)
print("Train and Test data labels created...")


# Transform the existing nominal variables into the integer coded variables using the LabelEncoder
for col in ['protocol_type', 'flag', 'attack_type']:
    le = LabelEncoder()
    le.fit(train[col])
    train[col] = le.transform(train[col])
    le1 = LabelEncoder()
    le1.fit(test[col])
    test[col] = le1.transform(test[col])


# for Stadart Scaler
scaler = MinMaxScaler()   #scale between 0 and 1
train = scaler.fit_transform(train)
test = scaler.fit_transform(test)
total = np.concatenate([train, test] )
```

```python
# Decomposition features are generated for both train and test data
pca = PCA(n_components=5, random_state=100)
pca.fit(total)
train = pca.transform(train)
test = pca.transform(test)



print("Decomposed features created...")
print("Number of features used : %d" % train.shape[1])






# Performing 5-fold Cross validation
startTime = time.clock()
LR1 = LogisticRegression(random_state = 0)
score = cross_val_score(LR1, train, trainLabel, cv=5)
endTime = time.clock()
print("Time taken to perform 5-fold cross validation : %f" % (endTime-startTime))
print("Cross validation scores : ")
print(score)
print("Mean score of 5-fold cross validation : %f" % score.mean())


# Final Testing and Evaluate Performance
# Train the KNN classifier model by original train data and got optimized parameter
startTime = time.clock()
LR2 = LogisticRegression(random_state = 0)
LR2.fit(train, trainLabel)
endTime = time.clock()
print("Time taken to train final model : %f" % (endTime-startTime))
print("Predictions made using final model...")


# Predictions for test data and evaluate its performance
startTime = time.clock()
pred = LR2.predict(test)
endTime = time.clock()
cpuUsage = psutil.cpu_percent()
pid = os.getpid()
py = psutil.Process(pid)
memoryUse = py.memory_info()[0] / 2. ** 30
print("Time taken to make predictions on test data : %f" % (endTime-startTime))
print("Memory used : %f GB  CPU usage : %f" % (memoryUse, cpuUsage))

```

```
con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels = classesName)
print("Confusion matrix : ")
print(con_matrix)


# Print accuracy and detection rate
acc = accuracy_score(y_pred=pred, y_true=testLabel)
print("Accuracy score on test data is : %f" % acc)

sumDr = 0
for i in range(con_matrix.shape[0]):
    det_rate = 0
    for j in range(con_matrix.shape[1]):
        if i != j :
            det_rate += con_matrix[i][j]
    if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i])  != 0:
        det_rate =100* con_matrix[i][i]/(det_rate + con_matrix[i][i])
        sumDr += det_rate
        print("For " + classesName[i] + ", Detection Rate is % " + str(det_rate))
    else:
        print("For " + classesName[i] + ", Detection Rate is % 0")

DR = sumDr/classesCount
print("DR is % " + str(DR))
```

Appendix: G

Python Code of Deep Learning Algorithm

```python
import pandas as pd
import numpy as np
import time
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
import psutil
import os
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import to_categorical


print("Deep Learning Classifier")
# Reads the train and test data
train = pd.read_csv('./KDDTrain.csv', header=None)
test = pd.read_csv('./KDDTest.csv', header=None)
print("Train and Test Data read...")


# Reads "Field Names.csv". Use this to set the names of train and test data columns
columns = pd.read_csv('Field Names.csv', header=None)
columns.columns = ['name', 'type']
train.columns = columns['name']
test.columns = columns['name']


# Read Service Types.csv
# Use this to create a mapping from service types to final labels (MAIL, WWW, FTP, DATABASE, v.b.)
serviceType = pd.read_csv('Service Types.csv', header=None)
serviceType.columns = ['Name', 'Type']
serviceMap={}



# Creates attackMap map which contains a mapping between service type and the final label
for i in range(len(serviceType)):
    serviceMap[serviceType['Name'][i]] = serviceType['Type'][i]
print("Service type mapping created...")


```

```python
# Add a new variable called 'label' which contains the final label
train['label'] = train['service'].map(serviceMap)
test['label'] = test['service'].map(serviceMap)



#find count of classes from serviceType
classesCount = len(train['label'].drop_duplicates())
classesName = train['label'].drop_duplicates().values.tolist()
print('There are ' +  str(classesCount) + ' classes to be monitored')
print('Classes are:')
print(classesName)



# Transform the existing nominal variables into the integer coded variables using the LabelEncoder
for col in ['protocol_type', 'flag', 'attack_type', 'label']:
    le = LabelEncoder()
    le.fit(train[col])
    train[col] = le.transform(train[col])
    le1 = LabelEncoder()
    le1.fit(test[col])
    test[col] = le1.transform(test[col])



# The variable 'label' is stored in different variables
# This is required to keep the dependent variable separate from the independent variable
trainLabel = train['label']
testLabel = test['label']



#Transform dependent variable into categorical columns as much as number of classes
trainLabel = to_categorical(trainLabel, classesCount)
testLabel = to_categorical(testLabel, classesCount)



# service_type and label variables are removed from the train and test data
# so only features are remained in train and test data
train.drop(['service', 'label'], axis=1, inplace=True)
test.drop(['service', 'label'], axis=1, inplace=True)
print("Train and Test data labels created...")
```

```python
# for Stadart Scaler
scaler = MinMaxScaler()   #scale between 0 and 1
train = scaler.fit_transform(train)
test = scaler.fit_transform(test)
total = np.concatenate([train, test] )



# Decomposition features are generated for both train and test data
pca = PCA(n_components=41, random_state=100)
pca.fit(total)
train = pca.transform(train)
test = pca.transform(test)



print("Decomposed features created...")
print("Number of features used : %d" % train.shape[1])


# fix random seed for reproducibility
seed = 7
np.random.seed(seed)



# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(110, input_dim=41, activation='relu'))
    model.add(Dense(110, activation='relu'))
    model.add(Dense(classesCount, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model


DLC = KerasClassifier(build_fn=baseline_model, epochs=10, batch_size=5, verbose=0)

'''
# Performing k-fold Cross validation
startTime = time.clock()
kfold = KFold(n_splits=5, shuffle=True, random_state=seed)
results = cross_val_score(DLC, train, trainLabel, cv=kfold)
endTime = time.clock()
print("Time taken to perform 5-fold cross validation : %f" % (endTime - startTime))
print("Mean score of 5-fold cross validation: %.2f%% " % (results.mean()*100))
'''
```

```
# Final Testing and Evaluate Performance
# Train the KNN classifier model by original train data and got optimized parameter
startTime = time.clock()
DLC.fit(train, trainLabel, epochs =10)
endTime = time.clock()
print("Time taken to train final model : %f" % (endTime - startTime))
print("Predictions made using final model...")


# Predictions for test data and evaluate its performance
startTime = time.clock()
pred = DLC.predict(test)
endTime = time.clock()
cpuUsage = psutil.cpu_percent()
pid = os.getpid()
py = psutil.Process(pid)
memoryUse = py.memory_info()[0] / 2. ** 30
print("Time taken to make predictions on test data : %f" % (endTime - startTime))
print("Memory used : %f GB  CPU usage : %f" % (memoryUse, cpuUsage))



pred = to_categorical(pred,classesCount)
pred = np.array(pred)
testLabel = np.array(testLabel)


#calculate confusion matrix
con_matrix = confusion_matrix(pred.argmax(axis=1), testLabel.argmax(axis=1)) #, labels=Classes
print("Confusion matrix : ")
print(con_matrix)


# Print accuracy and detection rate
#acc = accuracy_score(y_pred=pred, y_true=testLabel)
acc = accuracy_score(pred.argmax(axis=1), testLabel.argmax(axis=1))
print("Accuracy score on test data is : %f" % acc)

sumDr = 0
for i in range(con_matrix.shape[0]):
    det_rate = 0
    for j in range(con_matrix.shape[1]):
        if i != j :
            det_rate += con_matrix[i][j]
    if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i])  != 0:
        det_rate =100* con_matrix[i][i]/(det_rate + con_matrix[i][i])
        sumDr += det_rate
        print("For " + classesName[i] + ", Detection Rate is % " + str(det_rate))
    else:
```

```python
sumDr = 0
for i in range(con_matrix.shape[0]):
    det_rate = 0
    for j in range(con_matrix.shape[1]):
        if i != j :
            det_rate += con_matrix[i][j]
    if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i])  != 0:
        det_rate =100* con_matrix[i][i]/(det_rate + con_matrix[i][i])
        sumDr += det_rate
        print("For " + classesName[i] + ", Detection Rate is % " + str(det_rate))
    else:
        print("For " + classesName[i] + ", Detection Rate is % 0")

DR = sumDr/classesCount
print("DR is % " + str(DR))
```

Appendix: H

Gaussian Naïve Bayes

NMS - Gaussian Naive Bayes.py

```python
import pandas as pd
import numpy as np
import time
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
import psutil
import os

print("Gaussian Naive Bayes Classifier")
# Reads the train and test data
#train = pd.read_csv('1Million.csv', header=None)
#train = pd.read_csv('500K.csv', header=None)
#train = pd.read_csv('./125K.csv', header=None)
#train = train.iloc[:,0:42]
train = pd.read_csv('./KDDTrain+.csv', header=None)
test = pd.read_csv('./KDDTest+.csv', header=None)
print("Train and Test Data read...")



# Reads "Field Names.csv". Use this to set the names of train and test data columns
columns = pd.read_csv('Field Names.csv', header=None)
columns.columns = ['name', 'type']
train.columns = columns['name']
test.columns = columns['name']


# Read Service Types.csv
# Use this to create a mapping from service types to final labels (MAIL, WWW, FTP, DATABASE, v.b.)
serviceType = pd.read_csv('Service Types.csv', header=None)
serviceType.columns = ['Name', 'Type']
serviceMap={}




# Creates attackMap map which contains a mapping between service type and the final label
for i in range(len(serviceType)):
    serviceMap[serviceType['Name'][i]] = serviceType['Type'][i]


print("Service type mapping created...")

```

```
# Add a new variable called 'label' which contains the final label
train['label'] = train['service'].map(serviceMap)
test['label'] = test['service'].map(serviceMap)



#find count of classes from serviceType
classesCount = len(train['label'].drop_duplicates())
classesName = train['label'].drop_duplicates().values.tolist()
print('There are ' +  str(classesCount) + ' classes to be monitored')
print('Classes are:')
print(classesName)



# The variable 'label' is stored in different variables
# This is required to keep the dependent variable separate from the independent variable
trainLabel = train['label']
testLabel = test['label']




# service_type and label variables are removed from the train and test data
# so only features are remained in train and test data
train.drop(['service', 'label'], axis=1, inplace=True)
test.drop(['service', 'label'], axis=1, inplace=True)
print("Train and Test data labels created...")


# Transform the existing nominal variables into the integer coded variables using the LabelEncoder
for col in ['protocol_type', 'flag', 'attack_type']:
    le = LabelEncoder()
    le.fit(train[col])
    train[col] = le.transform(train[col])
    le1 = LabelEncoder()
    le1.fit(test[col])
    test[col] = le1.transform(test[col])


# for Stadart Scaler
scaler = MinMaxScaler()   #scale between 0 and 1
train = scaler.fit_transform(train)
test = scaler.fit_transform(test)
total = np.concatenate([train, test] )
```

```python

# Decomposition features are generated for both train and test data
pca = PCA(n_components=41, random_state=100)
pca.fit(total)
train = pca.transform(train)
test = pca.transform(test)



print("Decomposed features created...")
print("Number of features used : %d" % train.shape[1])




# Performing 5-fold Cross validation
startTime = time.clock()
GNB = GaussianNB()
score = cross_val_score(GNB, train, trainLabel, cv=5)
endTime = time.clock()
print("Time taken to perform 5-fold cross validation : %f" % (endTime-startTime))
print("Cross validation scores : ")
print(score)
print("Mean score of 5-fold cross validation : %f" % score.mean())


# Final Testing and Evaluate Performance
# Train the KNN classifier model by original train data and got optimized parameter
startTime = time.clock()
GNB2 = GaussianNB()
GNB2.fit(train, trainLabel)
endTime = time.clock()
print("Time taken to train final model : %f" % (endTime-startTime))
print("Predictions made using final model...")


# Predictions for test data and evaluate its performance
startTime = time.clock()
pred = GNB2.predict(test)
endTime = time.clock()
cpuUsage = psutil.cpu_percent()
pid = os.getpid()
py = psutil.Process(pid)
memoryUse = py.memory_info()[0] / 2. ** 30

print("Time taken to make predictions on test data : %f" % (endTime-startTime))
print("Memory used : %f GB  CPU usage : %f" % (memoryUse, cpuUsage))

```

```python

con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels = classesName)
print("Confusion matrix : ")
print(con_matrix)


# Print accuracy and detection rate
acc = accuracy_score(y_pred=pred, y_true=testLabel)
print("Accuracy score on test data is : %f" % acc)


sumDr = 0
for i in range(con_matrix.shape[0]):
    det_rate = 0
    for j in range(con_matrix.shape[1]):
        if i != j :
            det_rate += con_matrix[i][j]
    if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i])  != 0:
        det_rate =100* con_matrix[i][i]/(det_rate + con_matrix[i][i])
        sumDr += det_rate
        print("For " + classesName[i] + ", Detection Rate is % " + str(det_rate))
    else:
        print("For " + classesName[i] + ", Detection Rate is % 0")

DR = sumDr/classesCount
print("DR is % " + str(DR))
```