

**JULY 2019**

**M.Sc. in Electronics and Computer Engineering**

**KURBAN KOTAN**

**T.C.**

**HASAN KALYONCU UNIVERSITY**

**GRADUATE SCHOOL OF**

**NATURAL & APPLIED SCIENCES**

**COMPARATIVE ANALYSIS OF CLASSIFICATION  
TECHNIQUES FOR NETWORK ANOMALIES  
MANAGEMENT**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENT FOR MASTER DEGREE OF**

**M. Sc. THESIS**

**IN**

**ELECTRONICS AND COMPUTER ENGINEERING**

**KURBAN KOTAN**

**TEMMUZ 2019**

**Comparative Analysis of Classification Techniques for  
Network Anomalies Management**

**A Thesis Submitted In Partial Fulfilment of the  
Requirement for Master Degree Of**

**M. Sc. THESIS  
IN  
ELECTRONICS AND COMPUTER ENGINEERING  
HASAN KALYONCU UNIVERSITY**

**Supervisor**

**Assist. Prof. Dr. Mohammed K. M. MADI**

**Kurban KOTAN**

**July 2019**



©2019 [Kurban KOTAN].



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

Electronics-Computer Engineering M.Sc. (Master Of Science) programme student **Kurban KOTAN** prepared and submitted the thesis titled "**Comparative Analysis of Classification Techniques For Network Anomalies Management**" defended successfully on the date of **02/07/2019** and accepted by the jury as an M.Sc.thesis.

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

Kurban KOTAN

Signature

**ABSTRACT**  
**COMPARATIVE ANALYSIS OF CLASSIFICATION**  
**TECHNIQUES FOR NETWORK ANOMALIES MANAGEMENT**  
**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF**  
**THE REQUIREMENT FOR MASTER DEGREE OF**

KOTAN, KURBAN  
M.Sc. in Electronics and Computer Engineering  
Supervisor: Assist. Prof. Dr. Mohammed K. M. MADI  
July 2019  
116 pages

Today, the rapid development in technology is enabling billions of devices to communicate with each other. This development requires new network technologies to allow all these devices to connect to network easily. In recent years, cyber-attacks have been a serious threat to governments, businesses and individuals. Many Intrusion Detection Systems, which were designed to prevent these cyber-attacks failed. Intrusion Detections Systems (IDS) could not sufficiently recognize the attacks and the cunning ways the attackers used, resulting in inefficient IDS solution and vulnerable networks. It would be a much smarter solution to counteract attacks by using machine learning based systems that is the result of data mining and statistics. This approach will provide a more efficient IDS solution than a conventional IDS solution based on attack recognition techniques. The purpose of this thesis is to propose a method for Network Anomaly Detection System (NADS) using machine learning algorithms with the aim of enhancing the processes of the network troubleshooting, and raising the efficiency of the maintenance processes. This study compares the performance of four selected machine learning classifiers with each other. The selected algorithms are: K-Nearest Neighbors (KNN), K-means, Naïve Bayes and Random Forest. This comparison is conducted to detect the network anomaly and analyze the performance of the classification framework. This comparison is conducted to provide recommendations related to the framework selection. The above mentioned algorithms are implemented and tested on KDD CUP99 intrusion detection dataset that is widely used to evaluate intrusion detection prototypes. The experimental outcomes demonstrate that KNN algorithm perform well in terms of accuracy and computation time. Furthermore the results show that KNN has a successful detection of potential threat of 98.0379 % of all known attacks.

**Keywords:** Network Security, Machine Learning, Artificial Intelligence, Anomaly Detection, K-Nearest Neighbors Algorithm, Principal Component Analysis Algorithm, KDD CUP99 dataset.



**ÖZET**  
**COMPARATIVE ANALYSIS OF CLASSIFICATION**  
**TECHNIQUES FOR NETWORK ANOMALIES MANAGEMENT**  
**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF**  
**THE REQUIREMENT FOR MASTER DEGREE OF**

KOTAN, KURBAN  
Yüksek Lisans Tezi, Elektronik Bilgisayar Müh. Bölümü  
Tez Yöneticisi: Dr. Öğr. Üyesi Mohammed K. M. MADI  
Temmuz 2019  
116 sayfa

Bugün, teknolojiadaki hızlı gelişme milyarlarca cihazın birbiriyle iletişim kurmasını sağlıyor. Bu gelişme, tüm bu cihazların ağa kolayca bağlanabilmesi için yeni ağ teknolojilerini gerektirir. Son yıllarda, siber saldırılar hükümetler, işletmeler ve bireyler için ciddi bir tehdit oluşturuyor. Bu siber saldırıları önlemek için tasarlanan birçok saldırı tespit sistemi başarısız oldu. Saldırı Tespit Sistemleri (IDS) saldırıları ve saldırganların kullandığı kurnazca yollarını yeterince tanıyamadığından yetersiz IDS çözümü ve savunmasız ağlarla sonuçlandı. Veri madenciliği ve istatistiğin bir sonucu olan makine öğrenmesi tabanlı sistemler kullanmak saldırıları önlemek için çok daha akıllıca bir çözüm olacaktır. Bu yaklaşım, saldırı tanıma tekniklerine dayanan klasik IDS çözümüne kıyasla daha verimli bir IDS çözümü getirecektir. Bu tezin amacı, ağ sorun giderme işlemlerini geliştirmek ve bakım işlemlerinin verimliliğini artırmak amacıyla makine öğrenmesini kullanarak Ağ Tabanlı Anomali Tespit Sistemi (NADS) için bir yöntem önermektir. Bu çalışma, seçilen dört makine öğrenme sınıflandırma algoritmasının performansını birbiriyle karşılaştırmaktadır. Seçilen algoritmalar şunlardır: K-En Yakın Komşular (KNN), K-Means, Naïve Bayes ve Random Forest. Bu karşılaştırma ağ anomalisini tespit etmek ve sınıflandırma çerçevesinin performansını analiz etmek içindir. Bu karşılaştırma, çerçeve seçimi ile ilgili öneriler sunmak için yapılmıştır. Yukarıda belirtilen algoritmalar, izinsiz giriş tespit prototiplerini değerlendirmek için yaygın olarak kullanılan KDD CUP99 izinsiz giriş tespit veri setinde uygulanır ve test edilir. Deneysel sonuçlar KNN algoritmasının doğruluk ve hesaplama süresi açısından iyi çalıştığını göstermektedir. Ayrıca, KNN'nin bilinen tüm saldırıların % 98.0379'luk potansiyel tehdidin başarılı bir şekilde tespit ettiğini göstermiştir.



**Anahtar Kelimeler:** Ağ Güvenliđi, Makine Öğrenimi, Yapay Zeka, Anomali Tespiti, K-En Yakın Komşular Algoritması, Temel Bileşen Analizi Algoritması, KDD CUP99 veri seti.



This research work is dedicated to my family, my parents and teachers for strengthening me, and giving me resources and wisdom to finish this project.



## **ACKNOWLEDGEMENTS**

I want to express my gratitude and thanks to my major professor, Assistant Professor Mohammed K. M. MADI for his kindness and patience during this study, for his commitment and perseverance with me that enabled me to complete this work.



## TABLE OF CONTENTS

|                             |      |
|-----------------------------|------|
| ABSTRACT .....              | v    |
| ÖZET.....                   | vii  |
| ACKNOWLEDGEMENTS.....       | x    |
| TABLE OF CONTENTS.....      | xi   |
| LIST OF TABLES .....        | xiii |
| LIST OF FIGURES .....       | xiv  |
| LIST OF ABBREVIATIONS ..... | xvii |

### CHAPTER 1

#### INTRODUCTION

|                                     |   |
|-------------------------------------|---|
| 1.1 Problem Statement.....          | 2 |
| 1.2 Research Objectives.....        | 3 |
| 1.3 Significance of the Study ..... | 3 |
| 1.4 Background of the Study .....   | 3 |

### CHAPTER 2

#### BACKGROUND AND RELATED WORKS

|                                                                |    |
|----------------------------------------------------------------|----|
| 2.1 Introduction.....                                          | 5  |
| 2.2 Misuse Detection .....                                     | 5  |
| 2.3 Anomaly Detection .....                                    | 6  |
| 2.4 KDD CUP 99 Data Set .....                                  | 6  |
| 2.5 Literature Review .....                                    | 7  |
| 2.5.1 Statistical Analysis for Network Anomaly Detection ..... | 9  |
| 2.5.2 Classification-based NAD .....                           | 10 |
| 2.5.3 Clustering and outlier-based NAD .....                   | 10 |
| 2.5.4 Soft Computing .....                                     | 11 |
| 2.5.5 Knowledge-based NAD.....                                 | 12 |
| 2.5.6 Combination learners.....                                | 13 |
| 2.6 K-Nearest Neighbor (KNN) Algorithm.....                    | 13 |
| 2.7 K-Means Classification Algorithm.....                      | 15 |
| 2.8 Naïve Bayes Classification Algorithm .....                 | 16 |
| 2.9 Random Forest Classification Algorithm .....               | 17 |
| 2.10 Principal Component Analysis (PCA) Algorithm .....        | 18 |
| 2.11 NADS Algorithm.....                                       | 18 |

### CHAPTER 3

#### RESEARCH METHODOLOGY

|                                                |    |
|------------------------------------------------|----|
| 3.1 Methodology (Or Process) of CRISP-DM ..... | 20 |
| 3.1.1 Business Understanding .....             | 21 |
| 3.1.2 Data Understanding .....                 | 21 |

|                                   |    |
|-----------------------------------|----|
| 3.1.3 Data Preparation .....      | 24 |
| 3.1.3.1 Integrating data .....    | 24 |
| 3.1.3.2 Data Pre-processing ..... | 26 |
| 3.1.4 Modeling .....              | 28 |
| 3.1.4.1 Classification.....       | 28 |
| 3.1.4.2 Prediction .....          | 29 |
| 3.1.5 Evaluation.....             | 29 |
| 3.1.6 Deployment .....            | 29 |
| 3.2 System Structure .....        | 29 |
| 3.3 The Proposed System .....     | 30 |

## **CHAPTER 4**

### **NADS SYSTEM IMPLEMENTATION AND RESULTS**

|                                                                          |    |
|--------------------------------------------------------------------------|----|
| 4.1 Introduction.....                                                    | 33 |
| 4.2 System Structure .....                                               | 33 |
| 4.3 Performance Metrics.....                                             | 34 |
| 4.3.1 Confusion Matrix .....                                             | 34 |
| 4.4 Experiments and Results.....                                         | 36 |
| 4.4.1 System Accuracy Results .....                                      | 71 |
| 4.4.2 Classification Speed Results Based On Number of Feature Used ..... | 71 |
| 4.4.3 Memory Allocation Results.....                                     | 71 |

## **CHAPTER 5**

### **CONCLUSION**

|                         |           |
|-------------------------|-----------|
| <b>REFERENCES.....</b>  | <b>75</b> |
| <b>APPENDICES .....</b> | <b>78</b> |
| Appendix: A.....        | 78        |
| Appendix: B.....        | 79        |
| Appendix: C.....        | 81        |
| Appendix: D.....        | 82        |
| Appendix: E .....       | 82        |
| Appendix: F .....       | 90        |
| Appendix: G.....        | 94        |

## LIST OF TABLES

|                                                                          |    |
|--------------------------------------------------------------------------|----|
| Table 2.1: Distributions of attack classes in 1999 KDD cup dataset ..... | 7  |
| Table 2.2: The NADS Algorithm.....                                       | 18 |
| Table 3.1: Training Dataset of NADS.....                                 | 22 |
| Table 3.2: Test Dataset of NADS .....                                    | 23 |
| Table 3.3: OSI Layers .....                                              | 26 |
| Table 4.1: Confusion Matrix for Binary Case.....                         | 34 |
| Table 4.2: Example Confusion Matrix of NADS Which Is Multiclass .....    | 36 |
| Table 4.3: Experiment 1 .....                                            | 37 |
| Table 4.4: Experiment 2 .....                                            | 41 |
| Table 4.5: Experiment 3 .....                                            | 44 |
| Table 4.6: Experiment 4 .....                                            | 47 |
| Table 4.7: Experiment 5 .....                                            | 50 |
| Table 4.8: Scalability Experiments for KNN Algorithm .....               | 54 |
| Table 4.9: Scalability Experiments for KM Algorithm .....                | 56 |
| Table 4.10: Scalability Experiments for NB Algorithm .....               | 58 |
| Table 4.11: Scalability Experiments for RF Algorithm .....               | 60 |
| Table 4.12: Feature Scaling Experiments for KNN .....                    | 62 |
| Table 4.13: Feature Scaling Experiments for KM Algorithm.....            | 64 |
| Table 4.14: Feature Scaling Experiments for NB .....                     | 66 |
| Table 4.15: Feature Scaling Experiments for RF Algorithm .....           | 68 |

## LIST OF FIGURES

|                                                                                                 |    |
|-------------------------------------------------------------------------------------------------|----|
| Figure 1.1: Network Anomaly Detection Methods .....                                             | 1  |
| Figure 2.1: Classification-based NAD.....                                                       | 10 |
| Figure 2.2: Clustering and outlier-based NAD.....                                               | 11 |
| Figure 2.3: An Example of KNN Algorithm.....                                                    | 14 |
| Figure 2.4: Different boundaries separating the two classes with different values of k<br>..... | 15 |
| Figure 2.5: An Example of K-Means Classification Algorithm .....                                | 16 |
| Figure 2.6: An Example Question of Naïve Bayes Classification Algorithm and Its<br>Answer ..... | 17 |
| Figure 2.7: Illustration of Random Forest Classification Algorithm .....                        | 17 |
| Figure 2.8: An Example of PCA Algorithm.....                                                    | 18 |
| Figure 3.1: CRISP-DM .....                                                                      | 20 |
| Figure 3.2: Sample vectors of NADS's KDD dataset .....                                          | 23 |
| Figure 3.3: Captured and converted network packets by Wireshark .....                           | 24 |
| Figure 3.4: The application that deal with real packet capturing .....                          | 25 |
| Figure 3.5: Stages of packet decoder .....                                                      | 26 |
| Figure 3.6: The stages of data pre-processing data .....                                        | 28 |
| Figure 3.7: A Generic structure of NADS .....                                                   | 30 |
| Figure 3.8: Block Diagram of Training Phase.....                                                | 31 |
| Figure 3.9: Block Diagram of Testing Phase. ....                                                | 32 |
| Figure 4.1: System Architecture of NADS.....                                                    | 34 |
| Figure 4.2: Comparison of DR & ACC of First Experiment .....                                    | 38 |
| Figure 4.3: Comparison of Training and Testing Time of First Experiment.....                    | 38 |
| Figure 4.4: Comparison of Memory Consume of First Experiment .....                              | 39 |
| Figure 4.5: Experiment 1 of KNN .....                                                           | 39 |

|                                                                                |    |
|--------------------------------------------------------------------------------|----|
| Figure 4.6: Experiment 1 of KNN .....                                          | 40 |
| Figure 4.7: Comparison of DR & ACC of Second Experiment.....                   | 41 |
| Figure 4.8: Comparison of Training and Testing Time of Second Experiment ..... | 42 |
| Figure 4.9: Comparison of Memory Consume of Second Experiment.....             | 42 |
| Figure 4.10: Experiment 2 of KNN .....                                         | 43 |
| Figure 4.11: Comparison of DR & ACC of Third Experiment.....                   | 44 |
| Figure 4.12: Comparison of Training and Testing Time of Third Experiment ..... | 45 |
| Figure 4.13: Comparison of Memory Consume of Third Experiment.....             | 45 |
| Figure 4.14: Experiment 3 of KNN .....                                         | 46 |
| Figure 4.15: Comparison of DR & ACC of Fourth Experiment.....                  | 47 |
| Figure 4.16: Comparison of Training and Testing Time of Fourth Experiment..... | 48 |
| Figure 4.17: Comparison of Memory Consume of Fourth Experiment.....            | 48 |
| Figure 4.18: Experiment 4 of KNN .....                                         | 49 |
| Figure 4.19: Comparison of DR & ACC of Fifth Experiment.....                   | 50 |
| Figure 4.20: Comparison of Training and Testing Time of Fifth Experiment ..... | 51 |
| Figure 4.21: Comparison of Memory Consume of Fifth Experiment.....             | 51 |
| Figure 4.22: Experiment 5 of KNN .....                                         | 52 |
| Figure 4.23: Comparison of DR & ACC for KNN Algorithm.....                     | 54 |
| Figure 4.24: Comparison of Training and Testing Time for KNN Algorithm .....   | 55 |
| Figure 4.25: Comparison of Memory Consume KNN Algorithm .....                  | 55 |
| Figure 4.26: Comparison of DR & ACC for KM Algorithm.....                      | 56 |
| Figure 4.27: Comparison of Training and Testing Time for KM Algorithm .....    | 57 |
| Figure 4.28: Comparison of Memory Consume KM Algorithm .....                   | 57 |
| Figure 4.29: Comparison of DR & ACC for NB Algorithm.....                      | 58 |
| Figure 4.30: Comparison of Training and Testing Time for NB Algorithm .....    | 59 |
| Figure 4.31: Comparison of Memory Consume NB Algorithm .....                   | 59 |



|                                                                              |    |
|------------------------------------------------------------------------------|----|
| Figure 4.32: Comparison of DR & ACC for RF Algorithm .....                   | 60 |
| Figure 4.33: Comparison of Training and Testing Time for RF Algorithm .....  | 61 |
| Figure 4.34: Comparison of Memory Consume RF Algorithm .....                 | 61 |
| Figure 4.35: Comparison of DR & ACC for KNN Algorithm.....                   | 63 |
| Figure 4.36: Comparison of Training and Testing Time for KNN Algorithm ..... | 63 |
| Figure 4.37: Comparison of Memory Consume KNN Algorithm .....                | 64 |
| Figure 4.38: Comparison of DR & ACC for KM Algorithm.....                    | 65 |
| Figure 4.39: Comparison of Training and Testing Time for KM Algorithm .....  | 65 |
| Figure 4.40: Comparison of Training and Testing Time for KM Algorithm .....  | 66 |
| Figure 4.41: Comparison of DR & ACC for NB Algorithm.....                    | 67 |
| Figure 4.42: Comparison of Training and Testing Time for NB Algorithm .....  | 67 |
| Figure 4.43: Comparison of Memory Consume NB Algorithm .....                 | 68 |
| Figure 4.44: Comparison of DR & ACC for RF Algorithm .....                   | 69 |
| Figure 4.45: Comparison of Training and Testing Time for RF Algorithm .....  | 69 |
| Figure 4.46: Comparison of Memory Consume RF Algorithm.....                  | 70 |
| Figure 4.47: K-Fold Cross Validation .....                                   | 71 |
| Figure 4.48: Experiment of KM Algorithm.....                                 | 72 |
| Figure 4.49: Experiment of NB Algorithm .....                                | 73 |
| Figure 4.50: Experiment of RF Algorithm .....                                | 73 |

## LIST OF ABBREVIATIONS

|                  |                                                   |
|------------------|---------------------------------------------------|
| <b>ACC</b>       | : Accuracy Rate                                   |
| <b>ADS</b>       | : Anomaly Detection System                        |
| <b>AI</b>        | : Artificial Intelligence                         |
| <b>CM</b>        | : Confusion Matrix                                |
| <b>CRISP-DM</b>  | : Cross-Industry Standard Process for Data Mining |
| <b>DARPA</b>     | : Defense Advanced Research Project Agency        |
| <b>DoS</b>       | : Denial of Service                               |
| <b>DR</b>        | : Detection Rate                                  |
| <b>FN</b>        | : False Negative                                  |
| <b>FP</b>        | : False Positive                                  |
| <b>ICMP</b>      | : Internet Control Message Protocol               |
| <b>ID</b>        | : Intrusion Detection                             |
| <b>IDS</b>       | : Intrusion Detection System                      |
| <b>IP</b>        | : Internet Protocol                               |
| <b>KDD</b>       | : Knowledge Discovery and Data                    |
| <b>KDD CUP99</b> | : Knowledge Discovery and Data Mining CUP1999     |
| <b>KM</b>        | : K-Means                                         |
| <b>KNN</b>       | : K-Nearest Neighbors                             |
| <b>LAN</b>       | : Local Area Network                              |
| <b>NADS</b>      | : Network-based Anomaly Detection System          |
| <b>NAD</b>       | : Network-based Anomaly Detection                 |
| <b>NB</b>        | : Naïve Bayes                                     |
| <b>PCA</b>       | : Principal Component Analysis                    |
| <b>PSP</b>       | : Percentage of Successful Prediction             |

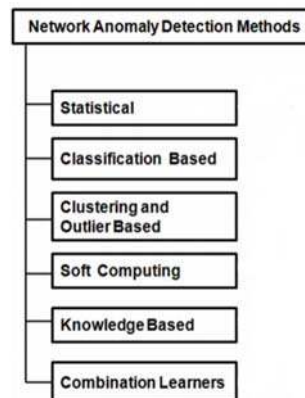
|                |                                                   |
|----------------|---------------------------------------------------|
| <b>RF</b>      | : Random Forests                                  |
| <b>R2L</b>     | : Remote to Local                                 |
| <b>SVM</b>     | : Support Vector Machine                          |
| <b>TCP</b>     | : Transmission Control Protocol                   |
| <b>WAN</b>     | : Wide Area Network                               |
| <b>WinPcap</b> | : Windows Packet Capturing                        |
| <b>TN</b>      | : True Negative                                   |
| <b>TP</b>      | : True Positive                                   |
| <b>TPR</b>     | : True Positive Rate                              |
| <b>U2R</b>     | : User to Root                                    |
| <b>UDP</b>     | : User Datagram Protocol                          |
| <b>TCP</b>     | : Transmission Control Protocol                   |
| <b>TCP/IP</b>  | : Transmission Control Protocol/Internet Protocol |

# CHAPTER 1

## INTRODUCTION

Network anomalies (network intrusions, network overload conditions, denial of service attacks and malicious/hostile activities) can cause network failures for both private and public entities. Malicious individuals and groups routinely employ cyber-attacks that target businesses and governments in what is now known as cyber-terrorism (Gable 2010). The goal is to cause fear and changes in behavior to affect political or ideological ends. Internet and computer networks are increasing day by day and the number of computers connected to the internet and computer networks are increasing and diversifying. On the other hand, cybersecurity (Cybersecurity is the protection of internet connected systems, hardware, software and data from cyber-attacks) threats are growing every day. Because of these factors, cybersecurity is becoming more complex and costly. Developing flexible and adaptable security-oriented approaches along with new types of attacks that are constantly emerging is a very difficult task. Therefore, anomaly intrusion detection techniques and systems for networks are very innovative to achieve the necessary protection.

The Network Anomaly Detection System (NADS) monitors computer networks and identifies any deviations from the normal profile to detect new attacks. Thus, it takes appropriate action. In this context, it is indispensable and very important for the network. Network anomaly detection (NAD) can be achieved through Statistical-based, Clustering and outlier-based, Classification-based, Knowledge-based, Soft computing and Combination learner based (Baliga, Kamat et al. 2007). These are shown in Figure 1.1.



**Figure 1.1:** Network Anomaly Detection Methods (Baliga, Kamat et al. 2007)

For statistical based NAD an anomaly is an observation. Observation is suspected of being partially or completely unrelated because observation is not generated by the assumed model (Bhuyan, Bhattacharyya et al. 2013). Therefore, any generated traffic with a low probability of occurrence are considered as anomalies. Based on training datasets, classification based NAD tries to assign new data samples to categories (Bhuyan, Bhattacharyya et al. 2013). Any object can be defined by its properties (or features). Clustering is to categorize new sets of objects into clusters (groups) by using a measurement distance or a specific correlation for it. The objects which are in the same set are more related to each when compare to object in the other sets (Garcia-Teodoro, Diaz-Verdejo et al. 2009). Soft computing is sufficient for NADS because it is impossible to find certain solutions sometimes. Soft computing is usually thought of as encompassing methods (Patcha and Park 2007). Knowledge based methods use host or network events first, then check these against predefined rule sets and known attack patterns (Baliga, Kamat et al. 2007). Combination learners combine multiple methods then partition them into three different categories: Fusion-based methods, Hybrid methods and Ensemble-based methods (Garcia-Teodoro, Diaz-Verdejo et al. 2009). The machine learning approach as developed in this thesis will be Statistical-based.

## **1.1 Problem Statement**

After firewall, ADS is protecting the network. Nowadays researchers focus on ADSs but however ADSs have low detection rate, low accuracy rate and high false alarm rate (Kathareios, Anghel et al. 2017).

Many studies have been carried out on the topic of network traffic modeling using machine learning (Mahoney and Chan 2002, Williams, Zander et al. 2006, Lippi, Bertini et al. 2013). K-Nearest Neighbors (KNN) classifier can achieve superior performance similar to Support Vector Machine (SVM) and Neural Networks, which are parametric classifiers. KNN, SVM and Neural Networks are in top of evaluated machine learning algorithms. In contrast to the parametric classifiers, KNN classifier has several important advantages (Duda, Hart et al. 2001). For KNN Classifier, memory requirements are high and KNN Classifier is susceptible to curse of dimensionality (Liao, Vemuri et al. 2007). We will focus on the KNN classifier, which can reduce the memory requirements and sensitivity to curse of dimensionality by using

less number of the input features instead of using hundreds or thousands of features. However, we will compare it other algorithms, like K-Means Classifier, Naïve Bayes Classifier and Random Forest Classifier, to compare its performance.

Using machine learning for anomaly detection enhances the speed of detection of structural errors, defects or frauds. Detecting anomalies across an entire network system is a very broad proposition. Here we focus on local area network (LAN) anomalies detection using machine learning.

## **1.2 Research Objectives**

The aim of this research is to use the machine learning techniques for network anomalies detection in LAN. In order to do this, the following objectives are identified:

- a) To research the available methods for the network anomalies detection using machine learning.
- b) To outline the identified methods taxonomy and identify the advantages, disadvantages and weaknesses of the identified method.
- c) To evaluate the performance of the identified method.

## **1.3 Significance of the Study**

This thesis's aim is establishing a network based anomaly detection system with machine learning to detect abnormal behavior of network traffic. This can be achieved by using fast machine learning algorithms that can process and analyze network traffic.

In a brief summary, Detection Rate (DR) will be increased to maximum and NADS will uncover and precisely identify new attacks. The system will use the known attack pattern in phase of training in order to increase detection rate which is actually machine learning.

## **1.4 Background of the Study**

There are five chapters in this thesis. Theoretical background of generic NADS, a brief explanation of methods of NADS, Knowledge Discovery Data mining (KDD CUP99 data set) description, K Nearest Neighbor algorithms, K-Means algorithm (KM), Naïve Bayes algorithm (NB), Random Forest algorithm (RF) and Principal Component Analysis algorithm (PCA) are presented in chapter two. Chapter three presents the methodology and system structure. Implementation and experimental

results from the proposed system and comparison of the system with other classification algorithms were presented in chapter four. Conclusions, limitations and suggestions for future work are presented at chapter five.



## **CHAPTER 2**

### **BACKGROUND AND RELATED WORKS**

#### **2.1 Introduction**

Network security is getting more and more importance than ever because computer networks are growing enormous with sensitive information and network based services on them. And internet and networks are exposed to increasing number of security threats. There is a big problem on detection of the growing new intrusion types: Labeling of the network data instances by human is usually troublesome, take too much time and expensive. Developing adaptive security oriented and flexible methods are very hard by new types of attacks appearing incessantly. Because of this, against malicious activities protecting target networks and systems, NADS are important technology.

The different methods of NAD are represented under this chapter and it is divided into six main categories (Bhuyan, Bhattacharyya et al. 2013):

- Soft computing
- Clustering and outlier-based
- Classification-based
- Statistical-based
- Combination learner
- Knowledge-based

In consideration of their advantages and drawbacks under this chapter, these six categories will be briefly explained.

#### **2.2 Misuse Detection**

In a misuse detection, which is an approach of detecting network attacks, first abnormal system behaviors are defined then all other behaviors are defined as normal. Anomaly detection approach defines normal behaviors first and then defines all other behaviors as abnormal. Therefore, it stands against this approach. In misuse detection everything unknown is normal. Using attack signatures in an intrusion detection



system is an example for misuse detection. Generally, term of misuse detection is used to all kinds of computer misuse (Helman, Liepins et al. 1992).

### 2.3 Anomaly Detection

In anomaly detection, it supposes that intrusions are anomalies that differ from normal behaviors. Generally, anomaly detection creates a profile for normal behaviors and marks them, which deviate largely from the profile, as attacks. Main advantage of anomaly detection is that it can detect unknown attacks. Disadvantage of anomaly detection is that it has high false positive rate because anomalies are not necessarily intrusive in practice. In addition, attacks that do not clearly deviate from normal activities cannot be detected by anomaly detection (Gaber and Discovery 2012).

### 2.4 KDD CUP 99 Data Set

In KDD-99 the Fifth International Conference on Knowledge Discovery and Data Mining, The Third International Knowledge Discovery and Data Mining Tools Competition was held and a data set was used for network intrusion detector. That data set is KDD Cup 99. Network intrusion detector was built as which predicts intrusions (or attacks) and label them bad connections and predicts normal connections and label them as normal.

There are standard set of data to be audited in this dataset, which simulated in a military network environment and contains a wide range of intrusions. Since 1999, it has been used wildly for the anomaly detection methods. This data set was built based on the data captured in DARPA'98 evaluation program. DARPA'98 had been obtained from 7 weeks of network traffic. And it contains about 5 million connection records. Each connections is about 100 bytes. Dataset is raw binary tcp dump data and it is about 4GB.

KDD training data set includes about 5 million connection vectors. Every vector includes 41 features, which is labeled normal or an attack (Tavallae, Bagheri et al. 2009). 41 features of vectors are shown in Appendix B.

Attacks fall into one of the following four categories:

***Denial of Service Attack (DoS):*** DoS is any type of attack that attackers prevent users from accessing the service. In this attack type, attacker usually sends excessive messages and asks the network (or server) to authenticate requests, which

have void return addresses. When network (or server) sends the authentication approval, network (or server) will not be able to find attacker's return address and before closing the connection, it will cause the server to wait. When the server ends the connection, attacker sends same type of messages. Therefore, the process of authentication and server waiting process will begin again and it will keep the network (or server) busy.

**User to Root Attack (U2R):** In this type of attacks, attacker gets on the system with normal user account and then attacker abuses security vulnerabilities to obtain super user privileges.

**Remote to Local Attack (R2L):** This is type of attacks that attacker gets access to a local user's computer on system over the internet by sending packets, in order to expose the machines security vulnerabilities and exploit privileges.

**Probing Attack (probe):** This is type of attacks that attacker scans a computer or a networking device, in order to expose the machines vulnerabilities and exploit privileges for later use.

Percentages of these attacks in KDD dataset are shown in Table 2.1.

**Table 2.1:** Distributions of attack classes in 1999 KDD cup dataset (Ahmed and Mahmood 2015, Aljawarneh, Aldwairi et al. 2018)

| Attack | Training Dataset | 10% of Training Dataset | Test Dataset |
|--------|------------------|-------------------------|--------------|
|        | (%)              | (%)                     | (%)          |
| Dos    | 79.28            | 79.24                   | 73.90        |
| Probe  | 0.84             | 0.83                    | 1.34         |
| U2R    | 0.001            | 0.01                    | 0.02         |
| R2L    | 0.023            | 0.23                    | 5.26         |

## 2.5 Literature Review

In networking, the ADSs are distinctively used to help in detecting anomalies in data of a network. This detection is made possible because the anomalous always occurs in the form of patterns. Nevertheless, other studies have depicted modeling of data in a sequential fashion in the process of detecting subsequences which are anomalous (Parmar and Patel 2017).

A review as well as a survey which has been conducted by (Lazarevic, Ertoz et al. 2003), on anomaly detection. In his review, it is established that the emergence of anomaly-based systems for detecting intrusion have been possible to develop numerous systems which can be utilized to effectively track novel attacks which have been waged on a given system. This has been possible through utilization of techniques such as maintaining a high DR of about 98% as well as a low rate of alarm of 1%. In his analysis, it is clear that he postulates that despite the fact that despite the efficiency level of the anomaly-based approaches in attack detections, the signature-based detections seem to be preferable in the event that there is a need for mainstream implementation of intrusion in a detection system.

In the same perspective, (Dasgupta, Ji et al. 2003) , projects that in enhancing the effectiveness of the anomaly detection systems, it is important to pay distinctive focus on the immunity-based techniques. The reason is that the technique does not focus on offering a remedy to anomalies in the system but it helps the network to be able to remain immune to any form of intrusion. Their analysis is relevant to this study because it offers an alternative to ensuring that the anomaly detection system is developed in a way that the focus is not only on detection of breach but also in enhancing protection of the system from any potential breach. In this way, the application of the immunity-based techniques is an emerging branch of the Artificial Intelligence (AI) and this is very important in the application of security.

The problem of anomaly detection system has been a concern of many intellectuals and this has led to the development of further mechanisms that can be effectively diploid to enhance not only detection of an anomaly but also protection of the system from any form of intrusion. Dorothy E. Denning (Denning 1987) in her analysis presents the detection of anomaly intrusion in a system should be detected in real time. In her opinion, this is very important because it helps in instituting measures that can help reverse the intrusion and block any further intrusion that can be incurred in the system. In this perspective in her analysis, she developed a system that enhanced real-time detection of system anomaly intrusions. The system developed by Dorothy E. Denning (Denning 1987) has the capacity of detecting a form of intrusion whether it is a break-in, a penetration as well as other forms of intrusion. This helps in ensuring that the system is under constant surveillance and protection. In her system, the foundation of the detection is on the hypothesis that through monitoring a system's

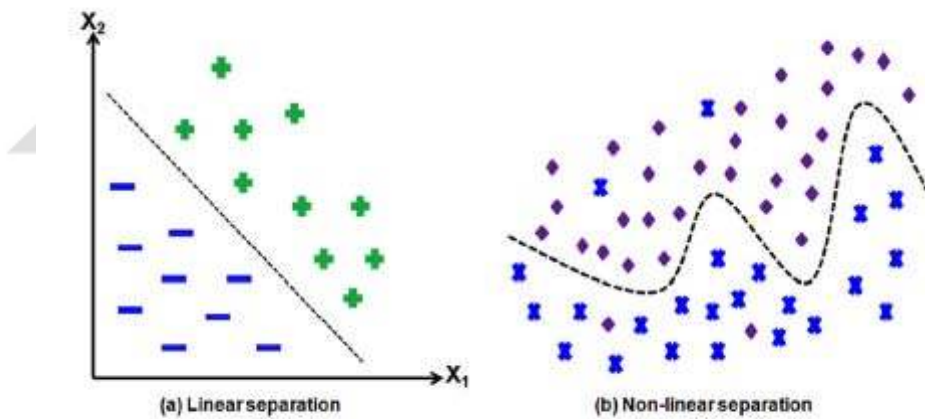
audit records for any abnormal patterns of use in the system, it is possible to detect a security violation in the system. In her system, she mentions that to make this possible there is every need to ensure that a profile that is utilized to represent the subject's behavior has to be maintained. This should be done with respect to the object in terms of metrics as well as statistical models. In the same way, in order to be able to enhance the real-time detection of the anomalies, a rule that enables the system to acquire knowledge about the behavior of the intruder from the audit records will enhance effectiveness in anomalous behavior detection. The important of this model presented by Dorothy E. Denning (Denning 1987) is that it operates independently of any other system. It also does not depend on the application environment of the system in question or under threat of attack, it has to be independent of the system vulnerability as well as the type of intrusion. In this way, it is possible to have a framework for the general-purpose intrusion-detection expert system.

### **2.5.1 Statistical-based NAD**

For statistical-based NAD, because of not generating from the stochastic model assumed an anomaly is an observation, which is suspected of being partially or completely irrelevant (Bhuyan, Bhattacharyya et al. 2013). Therefore, samples are anomalies because they have low probability of being generated. This method has two types: (a) parametric and (b) non-parametric. Parametric methods learn knowledge of distribution and they predict the parameters from data that is given, while nonparametric methods do not (Bhuyan, Bhattacharyya et al. 2013). Namely, parametric methods suppose that network data has certain distribution while non-parametric methods do not. Parametric methods make assumptions about statistical characteristics of the given data. There is no need prior knowledge about normal activity for this method and this is advantage of this it and these methods give accurate alert of malicious activities (Garcia-Teodoro, Diaz-Verdejo et al. 2009, Bhuyan, Bhattacharyya et al. 2013). However, until the network traffic composed during the attack is considered normal, they are vulnerable to be used to by attackers. Setting values for different parameters and metrics is hard, especially balancing between false positives and negatives (Bhuyan, Bhattacharyya et al. 2013). Those are disadvantages of statistical based NAD.

### 2.5.2 Classification-based NAD

In classification-based NAD, any object can be defined using properties or features. Based on training datasets, classification-based NAD tries to appoint new data samples to categories (Bhuyan, Bhattacharyya et al. 2013). In linear classification, it tries to draw a line between classes but the boundary might be nonlinear (Bhuyan, Bhattacharyya et al. 2013) as in Figure 2.1. By integrating new data, they are able to improve their execution that is why these classification methods are proper for training and testing (Bhuyan, Bhattacharyya et al. 2013). These are advantages of classification-based NAD. In addition, for known anomalies subject to appropriate thresholds these methods have very high detection rate (Bhuyan, Bhattacharyya et al. 2013). They are very sensitive to classifying hypotheses, which is the main disadvantage of classification based NAD. In addition, they are unable to detect unidentified anomalies until appropriate training datasets are given (Bhuyan, Bhattacharyya et al. 2013).

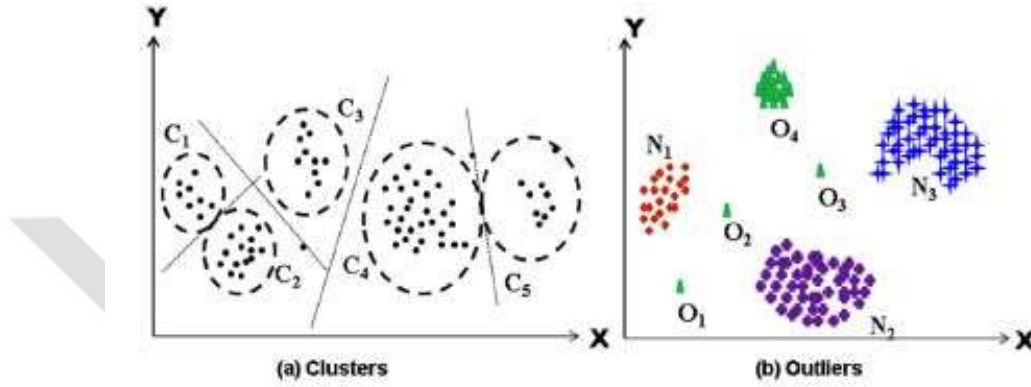


**Figure 2.1:** Classification-based NAD (Bhuyan, Bhattacharyya et al. 2013)

### 2.5.3 Clustering and outlier-based NAD

In clustering, it categorizes new sets of objects into groups and these groups are called clusters. It use a specific correlation or measurement distance during clustering. Observations are more related in same set to each other (Garcia-Teodoro, Diaz-Verdejo et al. 2009). As it is shown in Figure 2.2(a), in the most common application of clustering it consists of choosing representative points for each cluster. This is shown in Figure 2.2(a) as two dimensions. There are series of unidentified observations and by drawing ellipses around them; they are grouped into five clusters

(Bhuyan, Bhattacharyya et al. 2013). In Figure 2.2(b), we see that the outliers (abnormal data points) are separated from the normal clusters, which are non-existent clusters (Bhuyan, Bhattacharyya et al. 2013). In small-scale datasets, one of the main advantage for these methods it is able to find outliers easily. On the other hand, its computational complexity might be higher as compared to other NAD methods.



**Figure 2.2:** Clustering and outlier-based NAD (Bhuyan, Bhattacharyya et al. 2013)

#### 2.5.4 Soft Computing

For NAD, this method is sufficient because finding precise solutions is impossible sometimes (Patcha and Park 2007, Hamamoto, Carvalho et al. 2018). Methods of soft computing are shown below:

- *Artificial Neural Networks*: Artificial Neural Networks has been motivated from its inception by the recognition that the human brain computes in an entirely different way from the conventional digital computer. Artificial Neural Networks are established tools for various applications such as data clustering, feature extraction and anomalous pattern identification in a network.
- *Genetic algorithm (GA)*: Genetic algorithms (GAs) represent a computational model-based on principles of evolution and natural selection.
- *Artificial immune systems*: Artificial Immune Systems represent a computational method inspired by the principles of the human immune

system. The human immune system is adept at performing anomaly detection. The anomaly detection in the human immune system classifies certain external objects that enter the body as undesirable antigens, i.e., objects that may cause illness.

- *Colony algorithms*: Colony algorithms are probabilistic techniques for solving computational problems that can be reformulated to find optimal paths through graphs.
- *Rough Sets*: Rough sets have been effectively used in classification systems, where complete knowledge of the system is not available. A classifier aims to form various classes where members of a class are not noticeably different. These indiscernible or indistinguishable objects are viewed as basic building blocks (concepts) used to build a knowledge base about the real world. This kind of uncertainty is referred to as rough uncertainty.
- *Fuzzy Sets*: The concept of fuzzy logic provides a language with syntax and local semantics for translating qualitative knowledge about a problem to be solved.

These methods have high flexibility and adaptability. This is advantages of soft computing-based anomaly detection methods. Consuming high resource are their disadvantages. So in lack of normal traffic data, the training of the systems becomes very hard.

### **2.5.5 Knowledge-based NAD**

These methods check network or host events against known attack patterns and predefined rule sets. Knowledge based contains methods as shown down:

- Rule-based and expert system approaches
- Ontology and logic-based approaches

Advantages of these methods are scalability flexibility and robustness. If there are available training datasets for normal and anomalies both, these methods have high detection rate. Drawbacks of those methods are the costs and time consumption. Detecting unknown anomalies is very hard for knowledge based NAD (Garcia-Teodoro, Diaz-Verdejo et al. 2009).

### 2.5.6 Combination learners

This method combines multiple methods first. Then it divides these combined methods into three different categories, which are shown below:

- *Fusion-based methods*: Some methods of Fusion-based methods work in high dimensional feature spaces to extract and concatenate different semantic meanings. Others of Fusion-based methods attempt to combine classifiers trained on different features divided on the basis of hierarchical abstraction levels or the types of information contained.
- *Hybrid methods*: To overcome the limitations of the high false positive rate of anomaly detection and unknown intrusions of misuse detection, hybrid methods make use of features from approaches and get high accuracy.
- *Ensemble-based methods*: Ensemble-based methods are to weigh individual classifiers first and then combine them to get an overall classifier that outperforms all of them.

These methods cost a lot (Garcia-Teodoro, Diaz-Verdejo et al. 2009).

### 2.6 K-Nearest Neighbor (KNN) Algorithm

K-Nearest Neighbors algorithm is a very simple technique that is used in data mining. All available points are stored first and then KNN classifies new points according to similarity criteria using distance functions as follow:

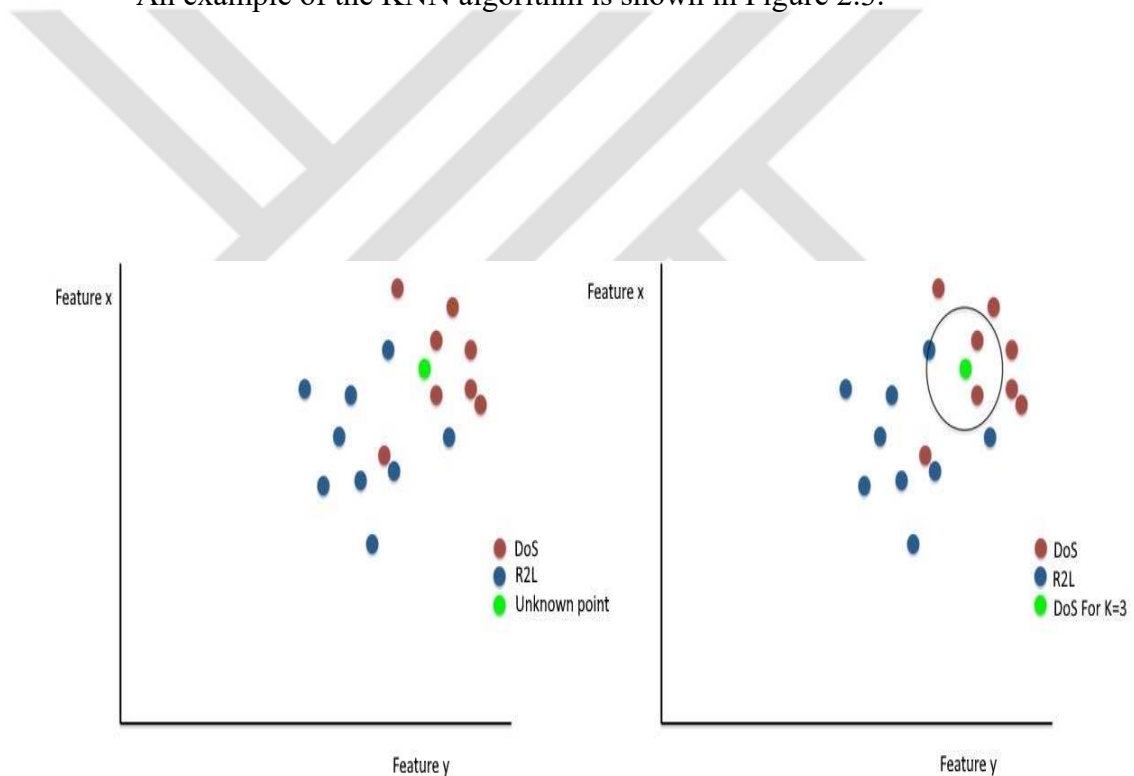
- *Euclidean* :  $\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$
- *Manhattan* :  $\sum_{i=1}^k |x_i - y_i|$
- *Minkowski* :  $(\sum_{i=1}^k (|x_i - y_i|^q))^{\frac{1}{q}}$



Pseudo Code of K-Nearest Neighbors algorithm is

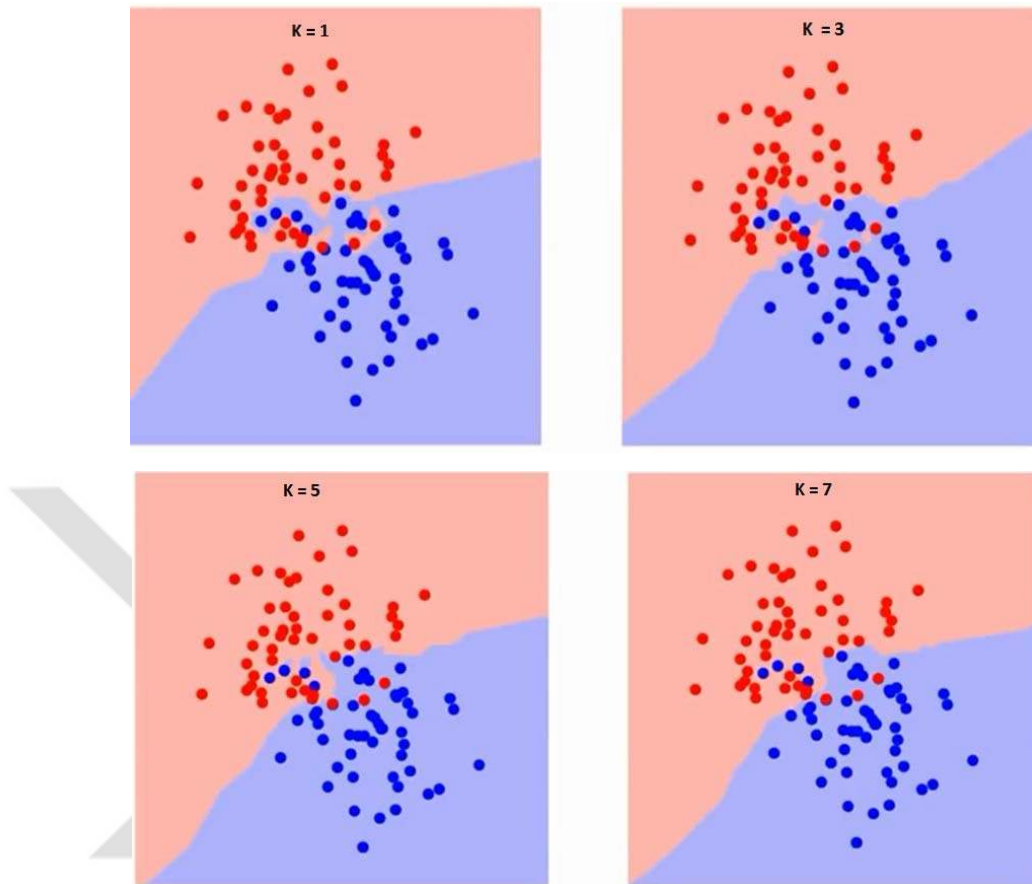
- 1-Load the data
- 2-Assign k value (In our example k value will be 1)
- 3-Iterate steps for all training data points to get predicted class
  - i- Calculate the distance between each points by using any distance function.
  - ii- Sort distances values in ascending order.
  - iii- From the sorted array, get the first top k points.
  - iv- Specify the class that contains most of these points.
  - v- Assign the point as specified class member.

An example of the KNN algorithm is shown in Figure 2.3.



**Figure 2.3:** An Example of KNN Algorithm

Boundaries that separates two classes according to different k values are shown at Figure 2.4.



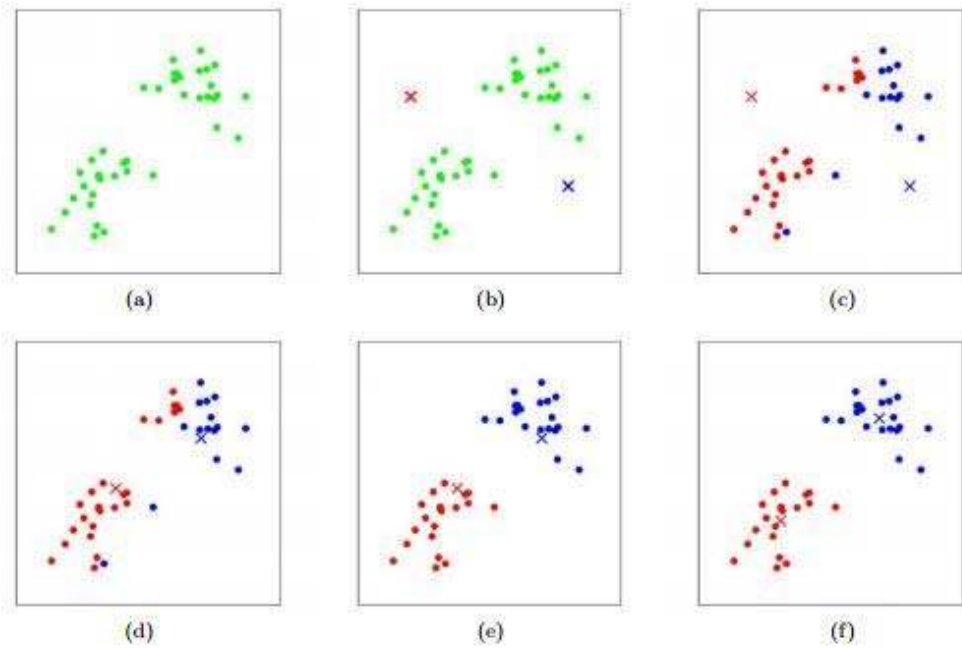
**Figure 2.4:** Different boundaries separating the two classes with different values of k

KNN is easy to be understood when there are few predictors .It is useful and easy to build models which are non-standard types. For example, text is non-standard type and if model contains text it is useful to use KNN (Omar, Ngadi et al. 2013).

Some other data classification algorithms that briefly described after this section will be compared with the KNN algorithm.

## 2.7 K-Means Classification Algorithm

K-Means Classification Algorithm is unsupervised. At beginning, it calculates initial class means by equally distributing them in the data space. Then by using a minimum distance measurement method, iteratively it groups data points into the nearest class. In every iteration, it recalculates every class means and according to these new means, it regroups observations. Until the maximum number of iterations is reached or the number of observations in each class changes by less than the selected observations change threshold this iteration continues.



**Figure 2.5:** An Example of K-Means Classification Algorithm

KM has low complexity but it is obligation to specify k value because it is unsupervised. Disadvantages of KM are that it is sensitive to outlier data points and noise and initial assignment of centroids may change the result too much.

## 2.8 Naïve Bayes Classification Algorithm

This Classifier is based on the Bayesian Theorem,  $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ , and it is especially suited to use if features dimensions are high. It can often perform better than complex classification methods despite its simplicity.

| Name          | Give Birth | Can Fly | Live in Water | Have Legs | Class       |
|---------------|------------|---------|---------------|-----------|-------------|
| human         | yes        | no      | no            | yes       | mammals     |
| python        | no         | no      | no            | no        | non-mammals |
| salmon        | no         | no      | yes           | no        | non-mammals |
| whale         | yes        | no      | yes           | no        | mammals     |
| frog          | no         | no      | sometimes     | yes       | non-mammals |
| komodo        | no         | no      | no            | yes       | non-mammals |
| bat           | yes        | yes     | no            | yes       | mammals     |
| pigeon        | no         | yes     | no            | yes       | non-mammals |
| cat           | yes        | no      | no            | yes       | mammals     |
| leopard shark | yes        | no      | yes           | no        | non-mammals |
| turtle        | no         | no      | sometimes     | yes       | non-mammals |
| penguin       | no         | no      | sometimes     | yes       | non-mammals |
| porcupine     | yes        | no      | no            | yes       | mammals     |
| eel           | no         | no      | yes           | no        | non-mammals |
| salamander    | no         | no      | sometimes     | yes       | non-mammals |
| gila monster  | no         | no      | no            | yes       | non-mammals |
| platypus      | no         | no      | no            | yes       | mammals     |
| owl           | no         | yes     | no            | yes       | non-mammals |
| dolphin       | yes        | no      | yes           | no        | mammals     |
| eagle         | no         | yes     | no            | yes       | non-mammals |

**A: attributes**

**M: mammals**

**N: non-mammals**

$$P(A|M) = \frac{6}{7} \times \frac{6}{7} \times \frac{2}{7} \times \frac{2}{7} = 0.06$$

$$P(A|N) = \frac{1}{13} \times \frac{10}{13} \times \frac{3}{13} \times \frac{4}{13} = 0.0042$$

$$P(A|M)P(M) = 0.06 \times \frac{7}{20} = 0.021$$

$$P(A|N)P(N) = 0.004 \times \frac{13}{20} = 0.0027$$

| Give Birth | Can Fly | Live in Water | Have Legs | Class |
|------------|---------|---------------|-----------|-------|
| yes        | no      | yes           | no        | ?     |

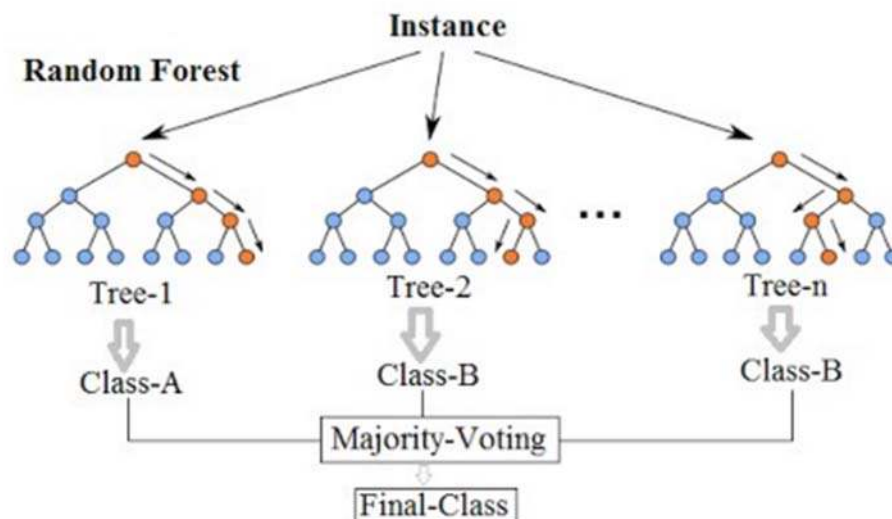
**P(A|M)P(M) > P(A|N)P(N)**

**=> Mammals**

**Figure 2.6:** An Example Question of Naïve Bayes Classification Algorithm and Its Answer (Han, Pei et al. 2011)

## 2.9 Random Forest Classification Algorithm

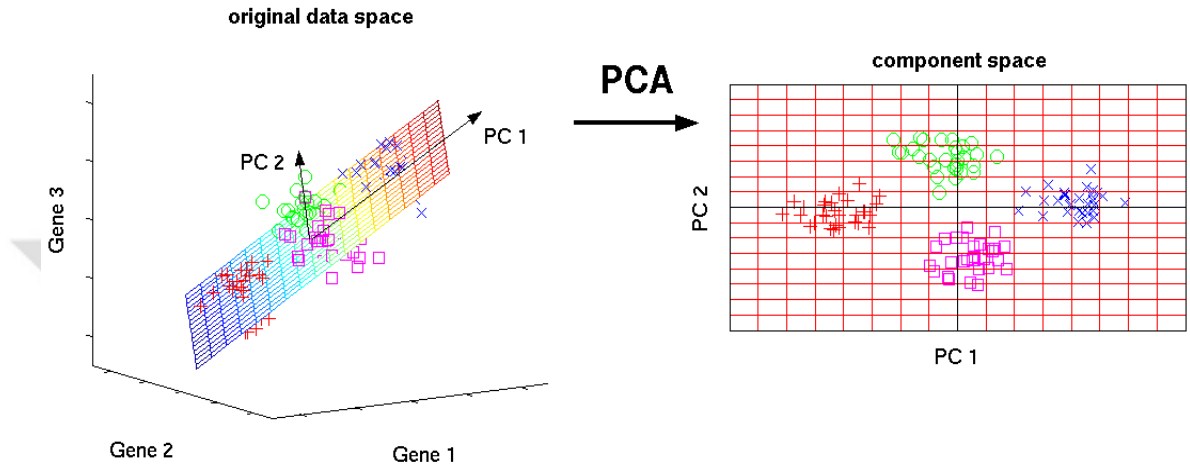
Random Forest is a supervised, most flexible and easy to use algorithm. A forest that it has more trees is more robust. RF create decision trees on randomly selected data first. Then it gets prediction from each tree. At last, by means of voting, it selects the best solution.



**Figure 2.7:** Illustration of Random Forest Classification Algorithm

## 2.10 Principal Component Analysis (PCA) Algorithm

PCA (Jolliffe 1986) is a most used method in data processing and dimension (size) reduction with many applications in social sciences, engineering and biology. Some examples include the handwritten, zip code classification and human face recognition. Nowadays it is used widely in data mining and machine learning.



**Figure 2.8:** An Example of PCA Algorithm

PCA Algorithm will be used for reduction features before using K-Nearest Neighbors classifier in proposed system, which is in Chapter 3 and Chapter 4.

## 2.11 NADS Algorithm

The pseudo code of the NADS is shown below. It shows the general functions and working steps of the NADS system.

**Table 2.2:** The NADS Algorithm

- |    |                                                                                |
|----|--------------------------------------------------------------------------------|
| 1. | Start                                                                          |
| 2. | Read training file.                                                            |
| 3. | Train the System                                                               |
| 4. | Capture network packets by Wireshark application and extract features vectors. |
| 5. | If vector is normal Than                                                       |
| 6. | Mark the vector is normal.                                                     |
| 7. | Else                                                                           |
| 8. | Mark the vector as abnormal.                                                   |
| 9. | End                                                                            |

In this thesis, one of the efficient and easiest data-mining algorithm called KNN Algorithm is applied. Experimental results on the KDDCUP99 data set show that our approach is very effective in detecting network intrusion. Especially detection rate is 98.0379 %. It is observed that the proposed NADS works much better in terms of detection rate and accuracy rate when it is applied to KDD99 dataset compared with other algorithms (Farid, Harbi et al. 2010, Rao, Srinivas et al. 2011, Shanmugavadivu, Nagarajan et al. 2011).



## CHAPTER 3

### RESEARCH METHODOLOGY

#### 3.1 Methodology

This chapter presents the NADS, which is machine-learning system for anomalies detection system by using K-Nearest Neighbors Algorithm for classification and Principal Component Analysis Algorithm for dimension reduction in frame of CRISP-DM (Cross-Industry Standard Process for Data Mining) Methodology. It describes typical phases of a project, tasks in every phase and relationships between these tasks as a methodology. It provides a conspectus of the data-mining life cycle as a process model. It contains six steps as follow:

1. **Business Understanding:** What is the problem we are dealing with?
2. **Data Understanding:** What is the data we are working with?
3. **Data Preparation:** What are the transformations and extractions to be done on the data?
4. **Modeling:** What is the data model we should use?
5. **Evaluation:** Does the model meet the project goals?
6. **Deployment:** How should we use the model we developed

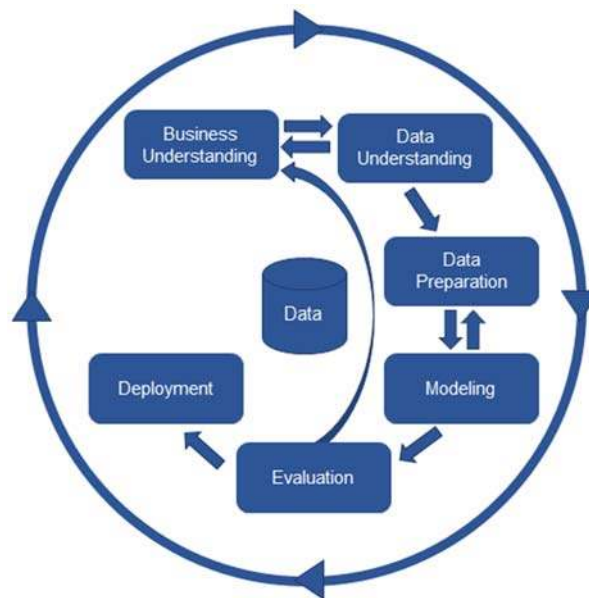


Figure 3.1: CRISP-DM

We will follow the steps of the CRISP-DM methodology in our machine learning system.

### **3.1.1 Business Understanding**

This is first step of the Methodology (Process) is about description of the problem, defining situation, determining goals and success criteria and determining project plan. Chapter one of this thesis is “business understanding” step of methodology. Business understanding of this thesis briefly is:

Network anomalies can cause network failures for both private and public entities. Internet and computer networks are getting larger day by day and by the number of devices connected to it is increasing and getting various. Because of these factors, cybersecurity is becoming more complex and costly. Developing flexible and adaptable security-oriented approaches along with new types of attacks that are constantly emerging is a very difficult task. Therefore, anomaly intrusion detection techniques and systems for networks are very innovative to achieve the necessary protection.

### **3.1.2 Data Understanding**

This step of the Methodology (Process) is about understanding of data. Those are gathering data, identify data, investigating data and verifying data quality. Subchapter 2.4 is “data understanding” step of methodology. The following table shows the part of the KDD Cup dataset which is used in proposed system NADS. Data understanding of this thesis briefly is:

KDD Cup 99 data set was used for network intrusion detector. Network intrusion detector was built as which predicts intrusions (or attacks) and label them bad connections and predicts normal connections and label them as normal. KDD training data set includes about 5 million connection vectors. Every vector includes 41 features, which is labeled normal or an attack. Attacks are generally as follow:

DoS, U2R, R2L and Probe.

Percentages of these attacks in KDD dataset which are used for training and testing phases are shown in Table 3.1 and Table 3.2.



**Table 3.1:** Training Dataset of NADS

|             |                 |         |
|-------------|-----------------|---------|
| normal      | normal          | 67.343  |
| dos         | back            | 956     |
|             | land            | 18      |
|             | neptune         | 41.214  |
|             | pod             | 201     |
|             | smurf           | 2.646   |
|             | teardrop        | 892     |
| probe       | ipsweep         | 3.599   |
|             | nmap            | 1.493   |
|             | portsweep       | 2.931   |
|             | satan           | 3.633   |
| u2r         | buffer_overflow | 30      |
|             | loadmodule      | 9       |
|             | perl            | 3       |
|             | rootkit         | 10      |
| r2l         | ftp_write       | 8       |
|             | guess_passwd    | 53      |
|             | imap            | 11      |
|             | multihop        | 7       |
|             | phf             | 4       |
|             | spy             | 2       |
|             | warezclient     | 890     |
|             | warezmaster     | 20      |
| Grand Total |                 | 125.973 |

**Table 3.2: Test Dataset of NADS**

|             |                 |        |
|-------------|-----------------|--------|
| normal      | normal          | 13.449 |
| dos         | back            | 196    |
|             | land            | 1      |
|             | neptune         | 8.282  |
|             | pod             | 38     |
|             | smurf           | 529    |
|             | teardrop        | 188    |
| probe       | ipsweep         | 710    |
|             | nmap            | 301    |
|             | portsweep       | 587    |
|             | satan           | 691    |
| u2r         | buffer_overflow | 6      |
|             | loadmodule      | 1      |
|             | rootkit         | 4      |
| r2l         | ftp_write       | 1      |
|             | guess_passwd    | 10     |
|             | imap            | 5      |
|             | multihop        | 2      |
|             | phf             | 2      |
|             | spy             | 1      |
|             | warezclient     | 181    |
|             | warezmaster     | 7      |
| Grand Total |                 | 25.192 |

Figure 3.2 represents some sample vectors of NADS; these vectors were collected from the KDD CUP99 data set, which consists of normal and abnormal behavior for network traffic.

[illegible]

**Figure 3.2:** Sample vectors of NADS’s KDD dataset

### 3.1.3 Data Preparation

This step of the Methodology (Process) is about integrating data from multi sources, formatting data, feature extraction, cleaning data, constructing data (derive attributes-transformation, filling in missing values) and feature selection.

#### 3.1.3.1 Integrating data

In this part, NADS catches all packets in network (or it will input training file or testing file). All packets in the network is captured. In captured packets, data and time fields are displayed which are data and packet captured time. Typical packets showed in Figure 3.3. All captured packets will be monitored and can be saved for analyzing.

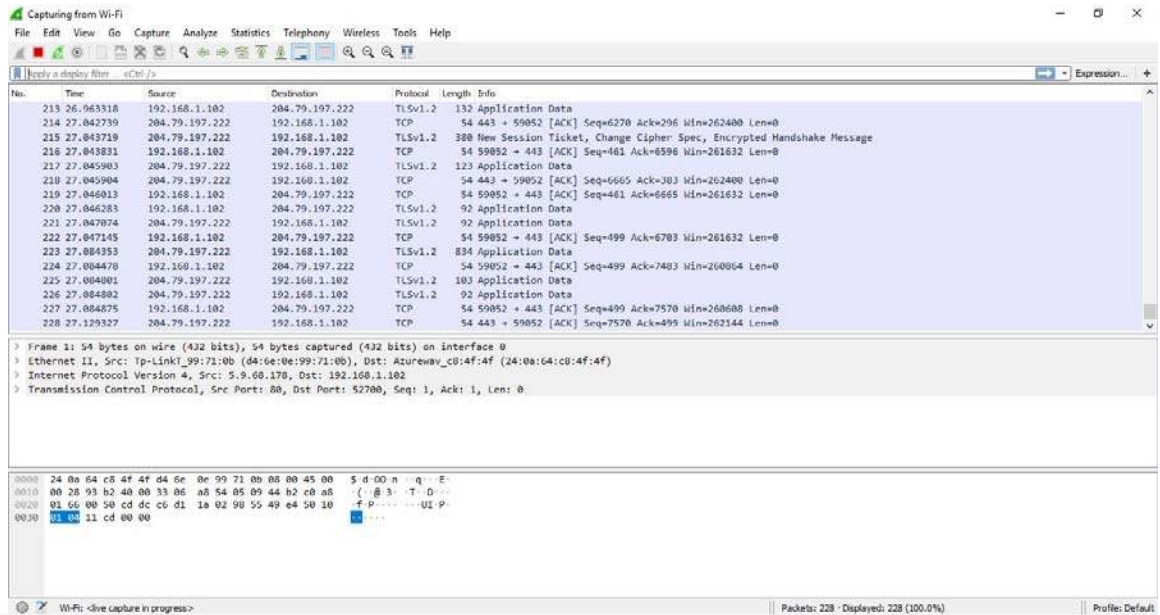


| No. | Time     | Source                    | Destination     | Protocol | Length | Info                                         |
|-----|----------|---------------------------|-----------------|----------|--------|----------------------------------------------|
| 1   | 0.000000 | HewlettP_9b:cb:b1         | Broadcast       | ARP      | 60     | Who has 192.168.104.36? Tell 192.168.104.60  |
| 2   | 0.019654 | 192.168.104.219           | 224.188.188.188 | UDP      | 298    | 39048 > 34952 Len=256                        |
| 3   | 0.020685 | 192.168.104.219           | 224.188.188.188 | UDP      | 298    | 39048 > 30583 Len=256                        |
| 4   | 0.027025 | fe80::11ba:51c7:ad61:a89b | ff02::1:3       | LLMNR    | 86     | Standard query 0x57a4 A isatap               |
| 5   | 0.028068 | 192.168.104.68            | 224.0.0.252     | LLMNR    | 66     | Standard query 0x57a4 A isatap               |
| 6   | 0.039366 | Dell_86:3e:ad             | Broadcast       | ARP      | 60     | Who has 192.168.104.36? Tell 192.168.104.41  |
| 7   | 0.039367 | Dell_86:3e:ad             | Broadcast       | ARP      | 60     | Who has 192.168.104.18? Tell 192.168.104.41  |
| 8   | 0.039367 | Dell_86:3e:ad             | Broadcast       | ARP      | 60     | Who has 192.168.104.30? Tell 192.168.104.41  |
| 9   | 0.086602 | 192.168.104.113           | 192.168.105.255 | NBNS     | 92     | Name query NB QIBRASOB.RU<00>                |
| 10  | 0.125713 | fe80::11ba:51c7:ad61:a89b | ff02::1:3       | LLMNR    | 86     | Standard query 0x57a4 A isatap               |
| 11  | 0.126806 | 192.168.104.68            | 224.0.0.252     | LLMNR    | 66     | Standard query 0x57a4 A isatap               |
| 12  | 0.144095 | Shuttle_40:7e:56          | Broadcast       | ARP      | 60     | Who has 169.254.109.152? Tell 192.168.105.2  |
| 13  | 0.161825 | 00:b9:7d:86:84:16         | Broadcast       | ARP      | 60     | Who has 87.118.124.104? Tell 192.168.104.214 |
| 14  | 0.209501 | Shuttle_40:7e:56          | Broadcast       | ARP      | 60     | Who has 169.254.110.152? Tell 192.168.105.2  |
| 15  | 0.234857 | Shuttle_40:7e:56          | Broadcast       | ARP      | 60     | Who has 169.254.89.152? Tell 192.168.105.2   |
| 16  | 0.234858 | Shuttle_40:7e:56          | Broadcast       | ARP      | 60     | Who has 169.254.90.152? Tell 192.168.105.2   |

**Figure 3.3:** Captured and converted network packets by Wireshark

First step of proposed system is packet capture. All packets of the real network traffic is captured in this step. This process runs in promiscuous mode. It captures all packets and then stores them as a set of traffic flows in data storage file.

In Figure 3.4 the user can capture packets by Wireshark and convert them csv file which contains features vectors in every line of the file showed in Figure 3.2.



**Figure 3.4:** The application that deal with real packet capturing

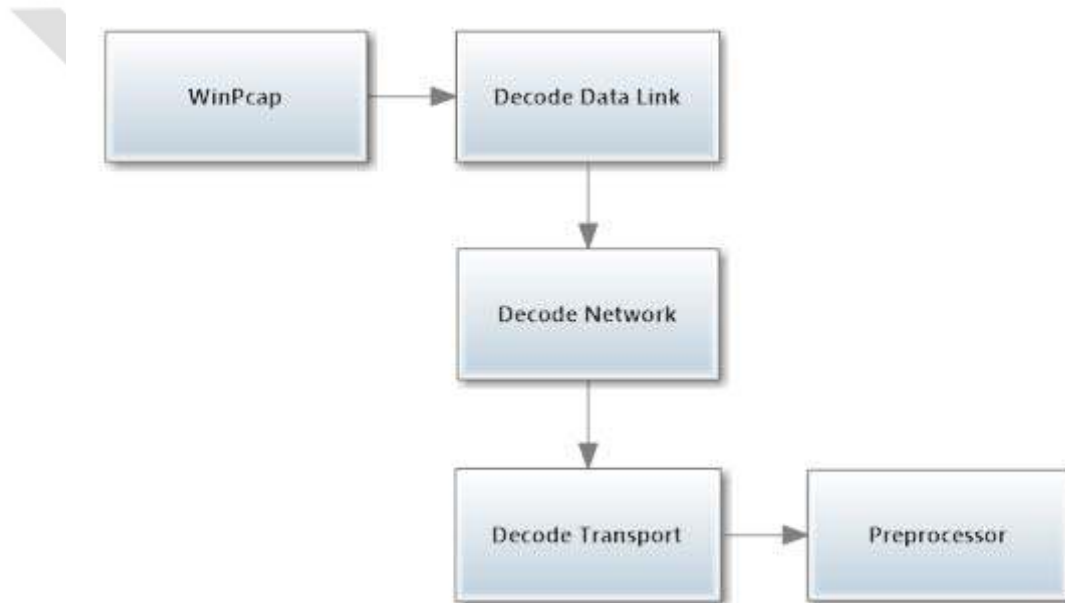
Packet decoder are shown in Figure 3.5. The packet decoder grabs packets from Data Link layer via WinPcap library, and defines which protocol is in use for any packet captured. WinPcap is a packet capture library that runs under the Windows operating system and captures packets from the network via the data link layer, the second layer of the OSI.

Wireshark is a multiplatform. It is open source application interface. It is used for monitoring network packets and convert them into other file types. In our thesis, we will use Wireshark which use WinPcap library to capture packets and convert their format for data pre-processing step.

Then, the packet stored in data structure is sent for data pre-processing stage (Rao, Srinivas et al. 2011).

**Table 3.3:** 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.<br>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 and error handling                      | TCP, UDP                  |
| 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 3.5:** Stages of packet decoder (Rao, Srinivas et al. 2011)

### 3.1.3.2 Data Pre-processing

Data pre-processing steps are shown in Figure 3.6. Data pre-processing means extracting information about the packet connection from its header and create new statistical features from data.

Standard data pre-processing steps contain dataset creation, data cleaning, integration, feature construction to derive new higher-level features, feature selection to choose the optimal subset of relevant features, reduction, optimization and normalization.

Most appropriate steps for NADS are now described below (Shanmugavadivu, Nagarajan et al. 2011):

**Dataset Creation:** This step contains identifying representative network traffic, which are packets information and some statistical information gathered from network, for the training and the testing phases. These dataset features of proposed system were created from several normal network sessions through weeks of normal work on the network, like dataset KDD CUP99 that we use for NADS. The features have been processed to get the values of the basic and statistical features that are considered normal and abnormal values for the network traffic.

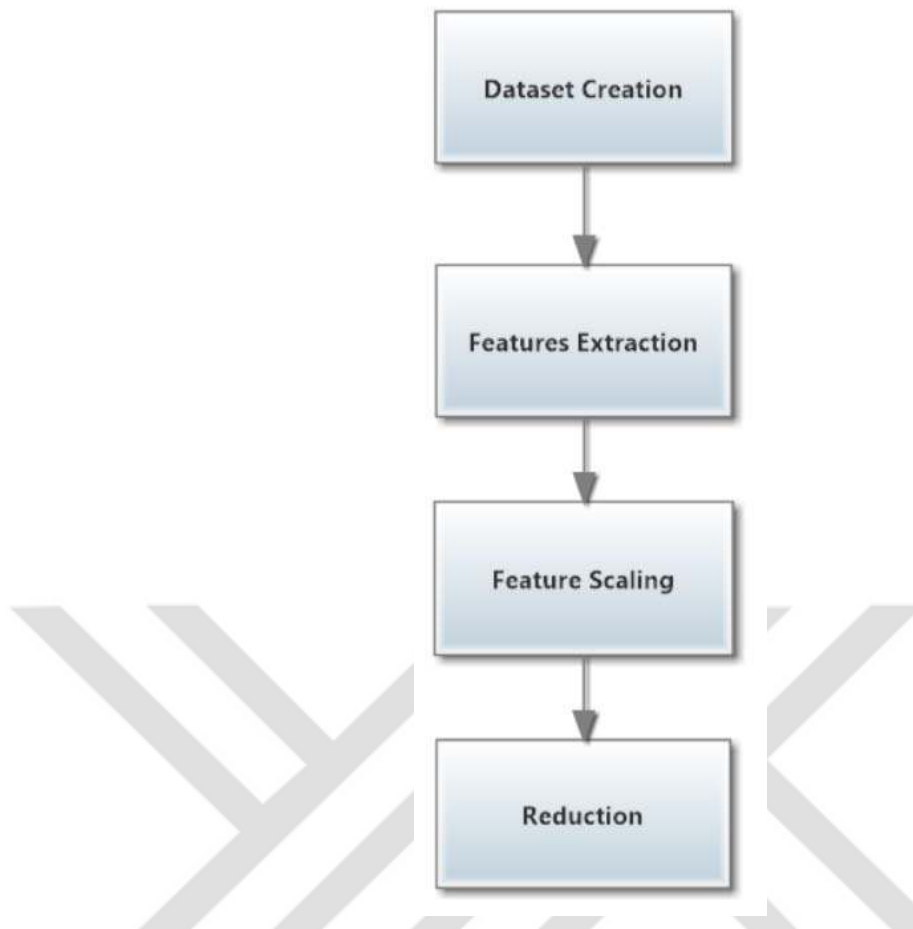
**Features Extraction:** Detecting anomalies depend on the values of features gathered packets from network. In this step system extracts basic features from packets header (such as protocol type, service, flag etc.) or extracts content features from payload of packets (such as logged in, etc.) or computes statistical values in order to create new features (like count, srv\_count, etc.).

**Feature Scaling:** This step is a technique for standardizing range of features of data or independent variables. Feature scaling is also known as data normalization. In data processing, it is generally performed during the data preprocessing step. In some machine learning algorithms, if the range of values of raw data varies widely without normalization objective functions will not work properly. For instance, most of classifiers calculate the distance between two points by using Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Because of this, the range of all features should be normalized first so that each feature contributes approximately proportionately to the final distance (Ioffe and Szegedy 2015):

- Standardization
- Normalization

**Reduction:** Reduction is used to determine whether multiple dimensions are related to each other and it is used to decrease the dimensions of the dataset by discarding any unnecessary or unrelated features.

After data pre-processing these feature vectors are suitable as input to machine learning algorithms.



**Figure 3.6:** The stages of data pre-processing data (Shanmugavadivu, Nagarajan et al. 2011)

### **3.1.4 Modeling**

This step of the Methodology (Process) is about selecting machine learning model (consider computer resources, computation time, number of features, business needs), generating test design (train/test split, cross validation, simulation [chronological order]), building model and assessing model. The most relevant steps of Modeling of Methodology for NADS are described below.

#### **3.1.4.1 Classification**

Classification is a method of determining what group a certain observation belongs to by classifier algorithm. For example, categorizing plants, animals, and other life forms into different taxonomies by biologists are classification. Classification is one of the primary uses of machine learning and data mining. In order to determine the correct category for a given observation, following are done by machine learning:



- Applies a classification algorithm to identify shared characteristics of certain classes.
- Compares those characteristics to the data it is trying to classify.
- Uses that information to estimate how likely it is that observation belongs to a particular class.

Then the train data is being classified by algorithms and the system is trained with the train data. In the prediction step, the trained system estimates vectors to determine whether the data is normal or not.

#### **3.1.4.2 Prediction**

In this stage, value of machine-learning is realized. Finally, trained system is used to predict the outcome and it labels packets as normal or as attack types. This stage is responsible with deciding whether an event or set of events are intrusion or not.

#### **3.1.5 Evaluation**

This step of the Methodology (Process) is about evaluating results in terms of business needs, reviewing process and determining next step. “Evaluation” step will be discussed in chapter four of this thesis.

#### **3.1.6 Deployment**

This step of the Methodology (Process) is about planning and deploying the model and planning monitoring and maintenance process. “Deployment” step will be discussed in chapter five of this thesis.

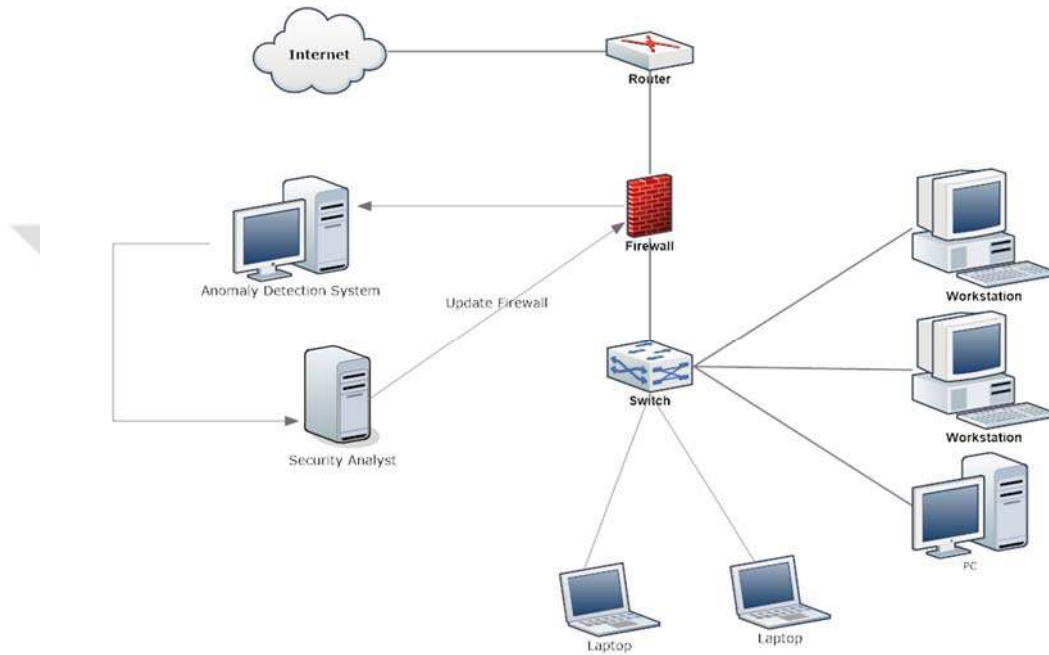
### **3.2 System Structure**

NADS classifies and identifies each connection vector to normal or intrusion types. Then results are displayed as alert. The NADS contains data aggregation, data pre-processing, classification, prediction and response stages. In “Data aggregation” stage, data is captured from network traffic by Wireshark network monitoring application and then data is used to train and to test the NADS in “Classification” and “Prediction” stages. In NADS, data for train is provided by KDD CUP99 dataset and “Pre-processing” stage regulates the data to ensure an efficient configuration of the classification system. KNN classification algorithm is used to build the proposed



NADS and classify the intrusion attacks in offline mode and online mode by cooperation of PCA and Wireshark network monitoring application. The last stage of NADS, "Response" displays the important information and informs the security analyst. Security analyst analyzes the risks, and then takes proper actions.

The system structure is shown in Figure 3.7. It presents a generic structure of NADS.



**Figure 3.7:** A Generic structure of NADS

The most important part of the NADS is to detect abnormal behavior and classify them, and then inform the Security Analyst. Whereupon, Security Analyst takes proper actions, through updating the database of the protection systems (Firewall, Antivirus Servers etc.) on the network.

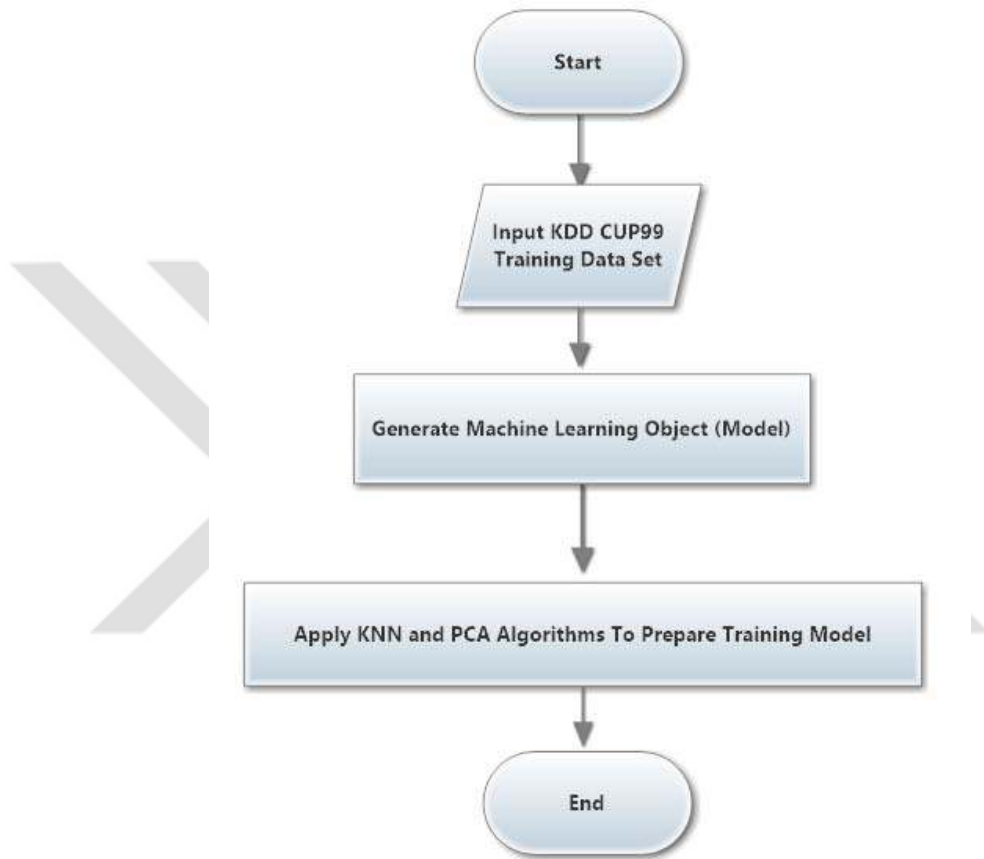
### 3.3 The Proposed System

NADS consists of two phases:

- Training phase
- Testing phase.

In Figure 3.8 training phase is shown and there are three steps in training phase, which are summarized as follows:

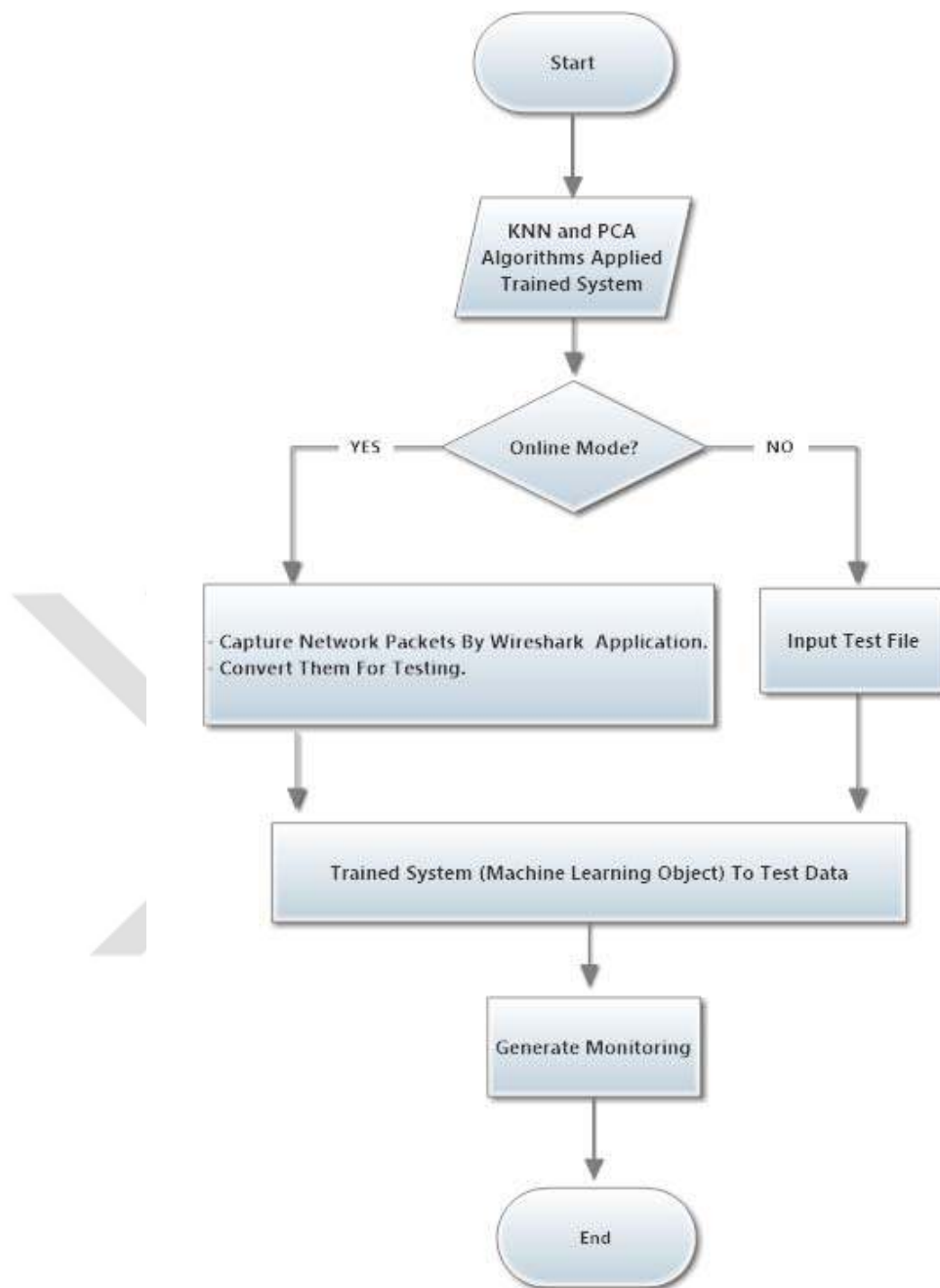
- i. *Input:* Input KDD CUP99 dataset.
- ii. *Process:* Train the system.
- iii. *Output:* NADS uses the KNN algorithm for classification, PCA algorithm for reduction and as a result, it classifies connections as normal or as attack and it gives attack type. Output of this phase is Trained System.



**Figure 3.8:** Block Diagram of Training Phase.

Testing phase is shown in Figure 3.9 and there are three steps in testing phase, which are summarized as follows:

- i. *Input:* Input Trained System is the first step in the testing phase.
- ii. *Process:* Trained System (Machine Learning Object) classifies connections as normal or attack types instantly (or testing file in offline mode).
- iii. *Output:* System generates monitoring.



**Figure 3.9:** Block Diagram of Testing Phase.

## **CHAPTER 4**

### **NADS SYSTEM IMPLEMENTATION AND RESULTS**

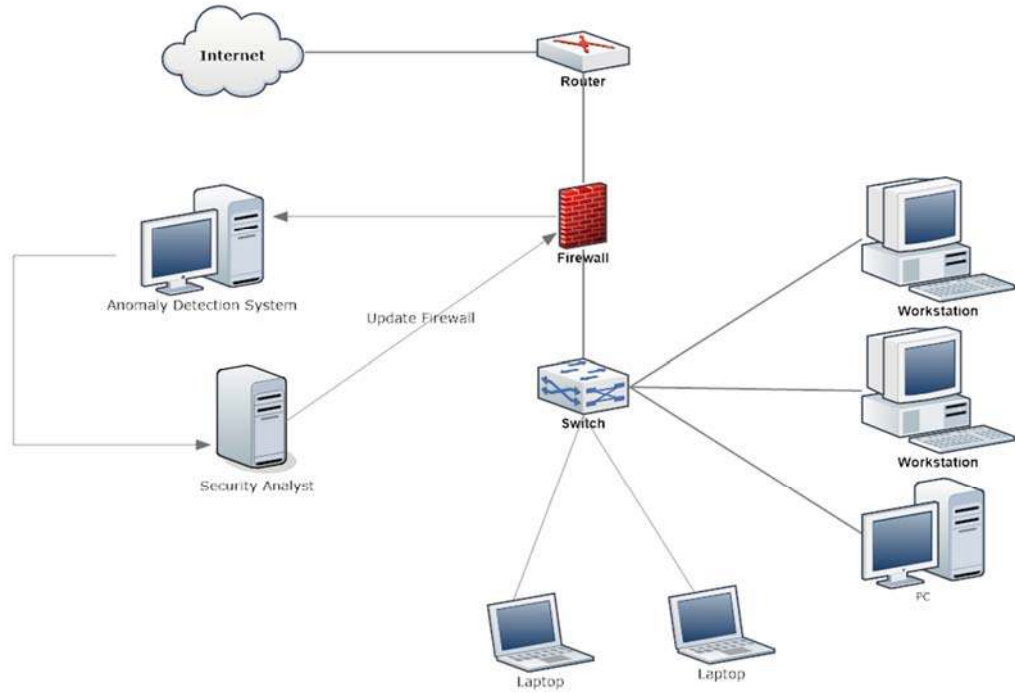
#### **4.1 Introduction**

Chapter four presents implementation of the proposed system by using KNN and PCA algorithms that are described in chapters two with their main points. This chapter also compares KNN algorithm with K-Means, Naïve Bayes and Random Forest algorithms which were explained in chapter two. All algorithms were compared with each other using them in NADS. The system is implemented on a computer which is connected to the LAN and this LAN has WAN connection. NADS were built by using machine-learning which was coded in Python programming language and it can be applied on most networks.

#### **4.2 System Structure**

NADS main components are listed below and it is shown in Figure 4.1.

1. Router, it is used for packets routing in network.
2. Switch which is configured in promiscuous mode. It is used for network packets switching and sniffing network packets.
3. NADS, captures packets, pre-processing, classifying attacks or normal connections and warning.
4. Security analyst, takes an appropriate prevention according to warnings.
5. LAN, it contains seven clients, one switch, one router, one firewall.



**Figure 4.1:** System Architecture of NADS

### 4.3 Performance Metrics

#### 4.3.1 Confusion Matrix

Confusion matrix (CM) is a performance measurement. It is used for machine learning classification problems which has two or more classes as output. It is a table, which contains four different combinations. Those are predicted values and actual values and these values are explained below. Confusion matrix is used to measure the performances of NADS in this thesis. If this measurement is adapted to NADS, following result will be gotten.

**Table 4.1:** Confusion Matrix for Binary Case (Stallings 2003)

|               |         | PREDICTED VALUES |        |
|---------------|---------|------------------|--------|
|               |         | Anomaly          | Normal |
| ACTUAL VALUES | Anomaly | TP               | FP     |
|               | Normal  | FN               | TN     |

1. True Positive (TP): Number of correctly predicted packets as attacks by system (Stallings 2003).
2. True Negative (TN): Number of correctly predicted packets as normal by system (Stallings 2003).
3. False Positive (FP): Number of normal packets, which are predicted as attacks (Stallings 2003).
4. False Negative (FN): Number of attack packets which are predicted as normal (Stallings 2003).

There are standard metrics for evaluating network anomalies detections which are shown below. Percentage of Successful Prediction (PSP), namely Detection Rate (DR) and Accuracy Rate (ACC) are most satisfactory metrics ratios. DR is the ratio between the number of correctly detected attacks and the total number of attacks as in Equation (4.1). Accuracy is a ratio of the total number of correctly classified attacks and normal connections divide to the total number of connections as in Equation (4.2) (Stallings 2003).

#### **4.3.1 Sensitivity-Detection Rate (DR) or True Positive Rate (TPR):**

Sensitivity is a ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High DR indicates the class is correctly recognized (Stallings 2003).

$$DR = TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \quad (4.1)$$

#### **4.3.2 Accuracy:**

Accuracy is a ratio of the total number of correctly classified positive and negative examples divide to the total number of examples (Stallings 2003).

$$Accuracy = \frac{TP + TN}{Total} \quad (4.2)$$

Example Confusion Matrix of NADS which is for multiclass is shown in Table 4.2.

**Table 4.2:** Example Confusion Matrix of NADS Which Is Multiclass

|               |        | PREDICTED VALUES |       |      |     |     |
|---------------|--------|------------------|-------|------|-----|-----|
|               |        | Normal           | Probe | DoS  | R2L | U2R |
| ACTUAL VALUES | Normal | 9223             | 10    | 0    | 1   | 0   |
|               | Probe  | 0                | 13432 | 9    | 8   | 0   |
|               | DoS    | 2                | 12    | 2275 | 0   | 0   |
|               | R2L    | 1                | 9     | 0    | 199 | 0   |
|               | U2R    | 0                | 1     | 0    | 3   | 7   |

#### 4.4 Experiments

In this section, we apply four scenarios, where each scenario consists of several experiments, and we summarize experimental results to detect the network anomaly over KDDCUP99 data set. These scenarios are explained as below:

**Scenario 1:** In order to see effect of the PCA algorithm five experiments were carried out on KDD CUP99 dataset by using KNN Algorithm. The conditions of these experiments are as follow:

*Experiment 1:* 125,973 lines of KDD CUP99 dataset were used for training and 25,192 lines for testing, but 41 features were selected.

*Experiment 2:* 125,973 lines of KDD CUP99 dataset were used for training and 25,192 lines for testing, but 30 features were selected.

*Experiment 3:* 125,973 lines of KDD CUP99 dataset were used for training and 25,192 lines for testing, but 20 features were selected.

*Experiment 4:* 125,973 lines of KDD CUP99 dataset were used for training and 25,192 lines for testing, but 10 features were selected.

*Experiment 5:* 125,973 lines of KDD CUP99 dataset were used for training and 25,192 lines for testing, but 5 features were selected.

### ***Scenario 1 Implementation:***

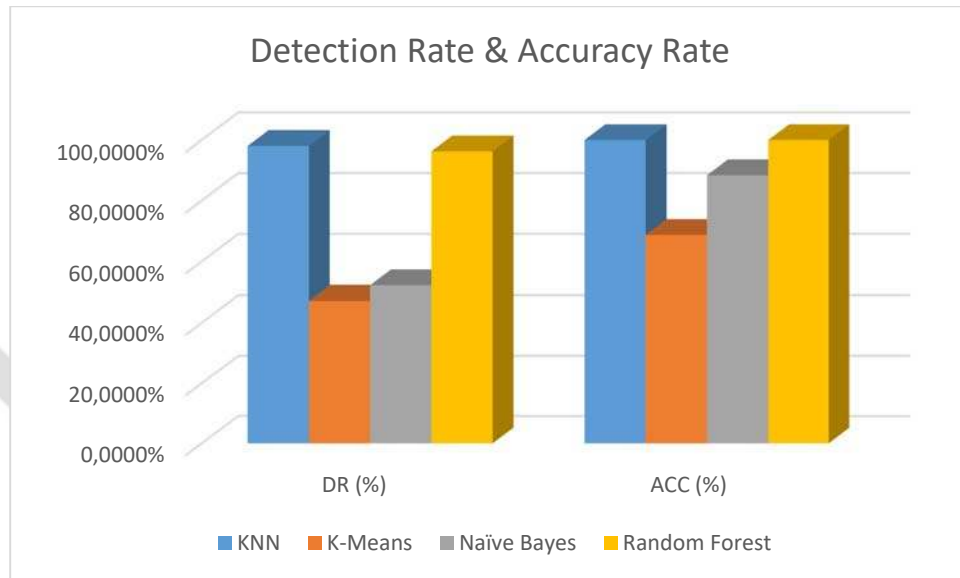
The results of first experiment can be seen in Table 4.3. The first experimental result indicated that KNN algorithm achieved a DR percent of 98.0379 % as highest compared with other algorithms. Highest ACC Rate is 99.9603 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by RF algorithm, highest DR percent for Probe achieved by RF algorithm, highest DR percent for DoS achieved by KNN and RF algorithms, highest DR percent for R2L achieved by KNN algorithm, and highest DR percent for U2R achieved by KNN algorithms. Lowest time taken to train is 0.128126 second that achieved by NB algorithm and lowest time taken to test is 0.08962 second achieved by NB algorithm. In addition, lowest memory usage is 0.307209 GB that achieved by KM algorithm.

**Table 4.3:** Experiment 1

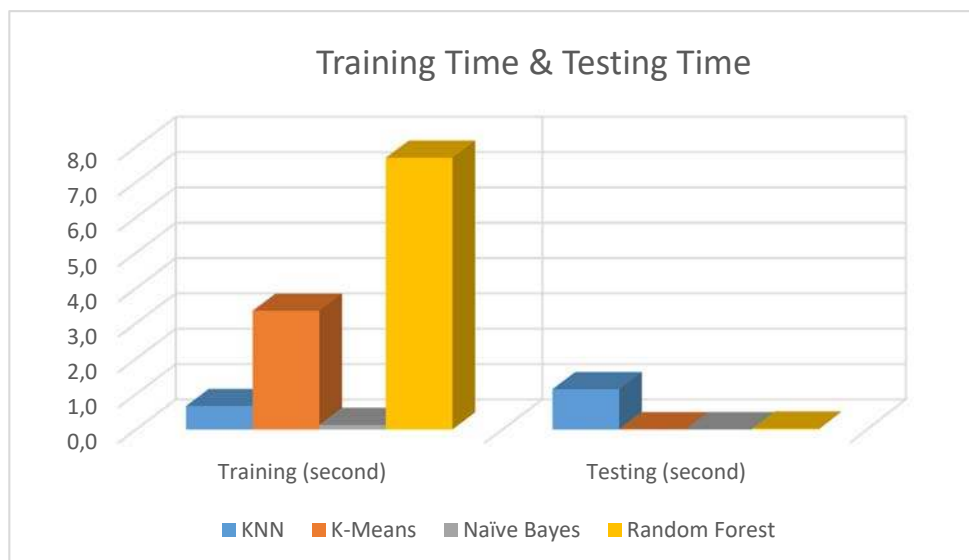
|                         | KNN      | KM       | NB       | RF       |
|-------------------------|----------|----------|----------|----------|
| DR (%)                  | 98.0379  | 47.1663  | 52.2071  | 96.2549  |
| ACC (%)                 | 99.9603  | 68.756   | 88.3773  | 99.9563  |
| DR For Normal (%)       | 99.9776  | 69.797   | 90.4825  | 99.9926  |
| DR For Probe (%)        | 99.7815  | 44.9104  | 80.166   | 99.9563  |
| DR For DoS (%)          | 100      | 74.713   | 89.4303  | 100      |
| DR For R2L (%)          | 99.5215  | 0.9569   | 0.9569   | 99.5074  |
| DR For U2R (%)          | 90.909   | 45.4545  | 0        | 81.8182  |
| Mean Of Cross V. (%)    | 99.6539  | 1.6965   | 88.4578  | 99.8396  |
| Time For Train (second) | 0.66159  | 3.872822 | 0.128126 | 7.694642 |
| Time For Test (second)  | 1.139292 | 0.009377 | 0.006212 | 0.026766 |
| Memory (GB)             | 0.304344 | 0.300694 | 0.505257 | 0.308975 |



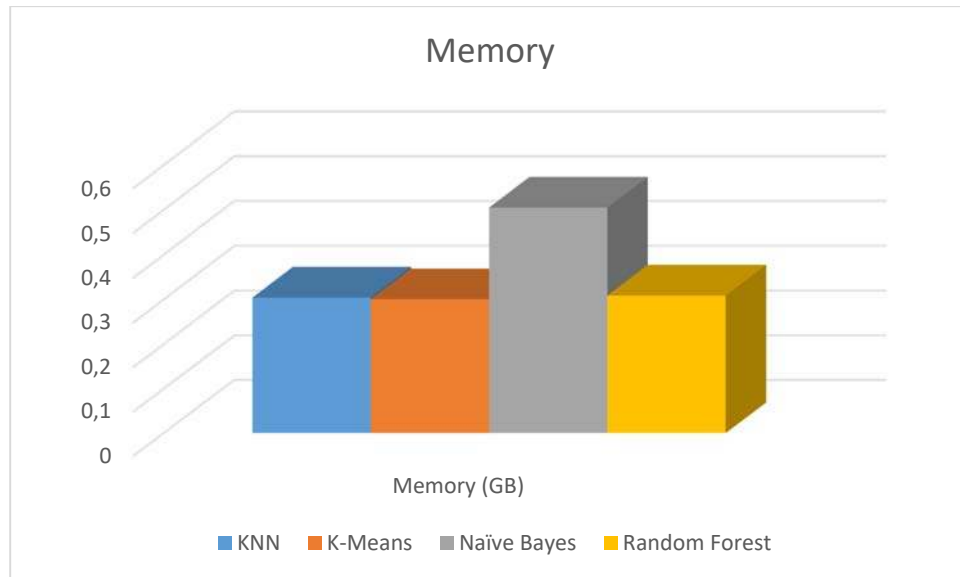
The comparison of DR and ACC ratios of all algorithms with each other is shown in Figure 4.2. The comparison of training and testing times of all algorithms with each other is shown in Figure 4.3. The comparison of memory consume of all algorithms with each other is shown in Figure 4.4.



**Figure 4.2:** Comparison of DR & ACC of First Experiment

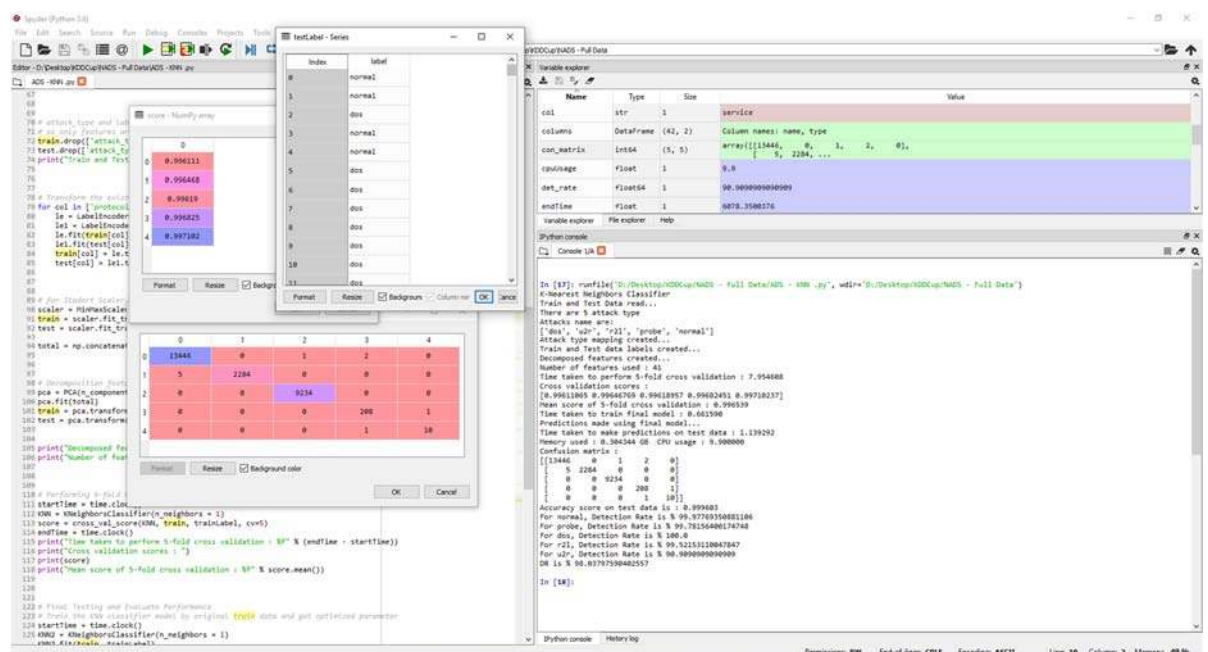


**Figure 4.3:** Comparison of Training and Testing Time of First Experiment



**Figure 4.4:** Comparison of Memory Consume of First Experiment

Screen shot of first experiment is shown in Figure 4.5 and screen shot of the program code for first experiment is shown in Figure 4.6.



**Figure 4.5:** Experiment 1 of KNN

```

K-Nearest Neighbors Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack 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 : 8.204882
Cross validation scores :
[0.99611065 0.99646769 0.99618957 0.99682451 0.99710237]
Mean score of 5-fold cross validation : 0.996539
Time taken to train final model : 0.669883
Predictions made using final model...
Time taken to make predictions on test data : 1.222388
Memory used : 0.318832 GB CPU usage : 30.400000
Confusion matrix :
[[13446    0     1     2     0]
 [    5  2284     0     0     0]
 [    0     0  9234     0     0]
 [    0     0     0   208     1]
 [    0     0     0     1    10]]
Accuracy score on test data is : 0.999603
For normal, Detection Rate is % 99.97769350881106
For probe, Detection Rate is % 99.78156400174748
For dos, Detection Rate is % 100.0
For r2l, Detection Rate is % 99.52153110047847
For u2r, Detection Rate is % 90.9090909090909
DR is % 98.03797590402557

```

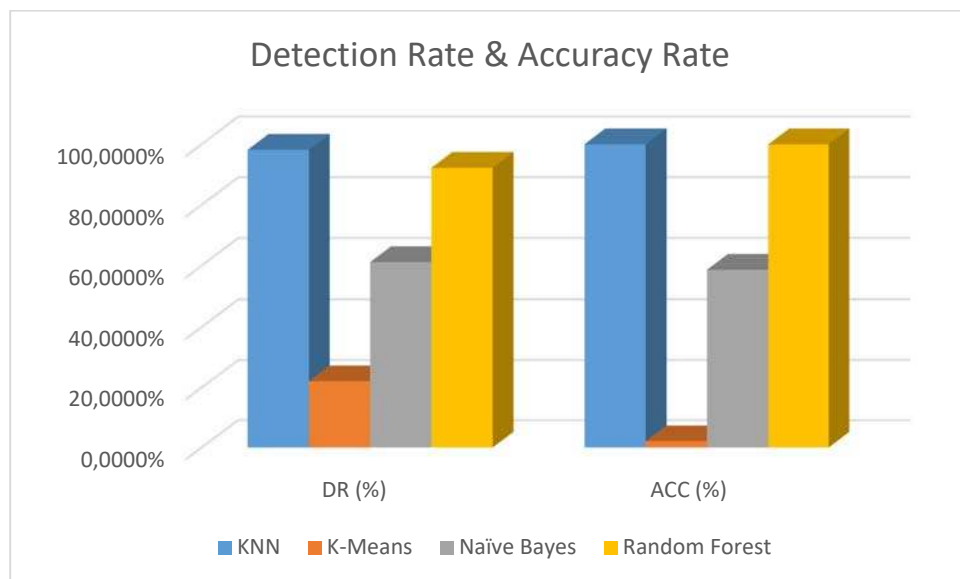
**Figure 4.6:** Experiment 1 of KNN

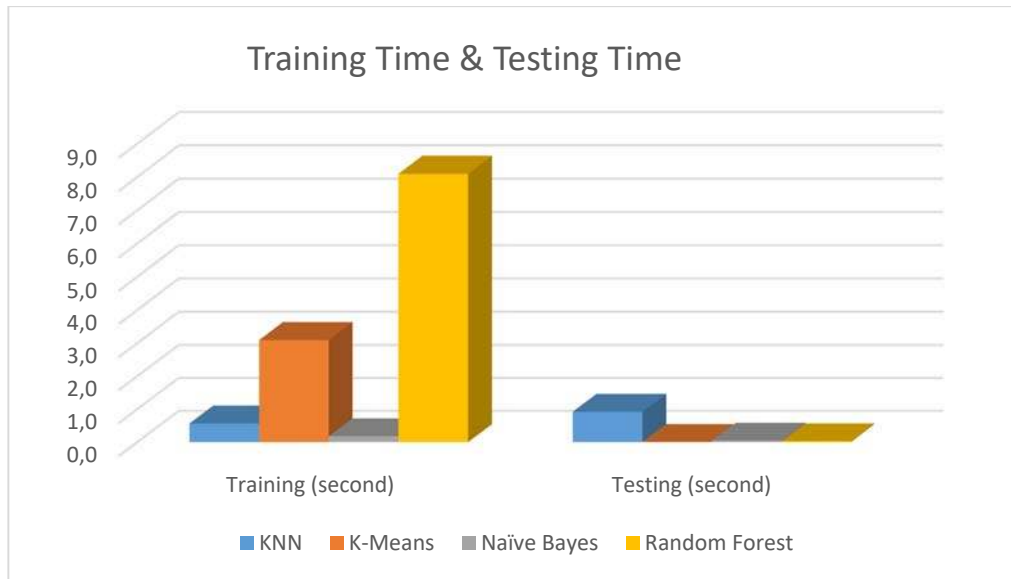
The results of second experiment can be seen in Table 4.4. The second experimental result indicated that KNN algorithm achieved a DR percent of 98.1387 % as highest compared with other algorithms. Highest ACC Rate is 99.9603 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by KNN algorithm, highest DR percent for Probe achieved by KNN algorithm, highest DR percent for DoS achieved by KNN algorithm, highest DR percent for R2L achieved by KNN algorithm, and highest DR percent for U2R achieved by KM algorithm. Lowest time taken to train is 0.179758 second that achieved by NB algorithm and lowest time taken to test is 0.007657 second achieved by KM algorithm. In addition, lowest memory usage is 0.300423 GB that achieved by NB algorithm.

**Table 4.4:** Experiment 2

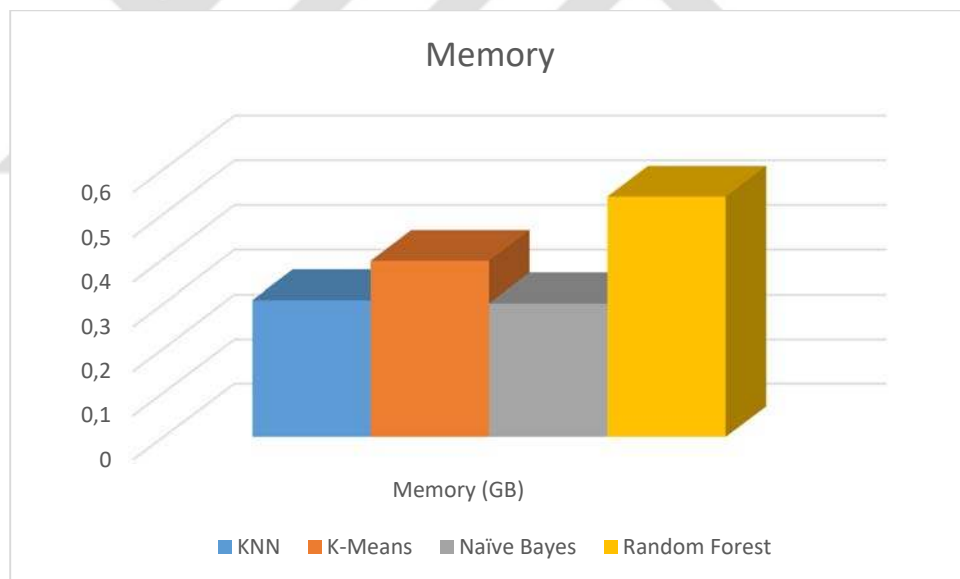
|                                | KNN      | KM       | NB       | RF       |
|--------------------------------|----------|----------|----------|----------|
| <b>DR (%)</b>                  | 98.1387  | 22.0586  | 61.1676  | 92.2005  |
| <b>ACC (%)</b>                 | 99.9603  | 2.0999   | 58.6377  | 99.9166  |
| <b>DR For Normal (%)</b>       | 99.9702  | 0.2007   | 30.374   | 99.9553  |
| <b>DR For Probe (%)</b>        | 99.8252  | 0.131    | 95.8934  | 99.8252  |
| <b>DR For DoS (%)</b>          | 99.9891  | 5.1765   | 90.0152  | 99.9783  |
| <b>DR For R2L (%)</b>          | 100      | 4.7846   | 35.0101  | 97.6076  |
| <b>DR For U2R (%)</b>          | 90.909   | 100      | 54.5455  | 63.6363  |
| <b>Mean Of Cross V. (%)</b>    | 99.6348  | 1.6953   | 58.6594  | 99.6483  |
| <b>Time For Train (second)</b> | 0.566747 | 3.098525 | 0.179758 | 8.095981 |
| <b>Time For Test (second)</b>  | 0.923757 | 0.007657 | 0.047077 | 0.028039 |
| <b>Memory (GB)</b>             | 0.307774 | 0.395073 | 0.300423 | 0.538746 |

The comparison of DR and ACC ratios of all algorithms with each other is shown in Figure 4.7. The comparison of training and testing times of all algorithms with each other is shown in Figure 4.8. The comparison of memory consume of all algorithms with each other is shown in Figure 4.9.

**Figure 4.7:** Comparison of DR & ACC of Second Experiment



**Figure 4.8:** Comparison of Training and Testing Time of Second Experiment



**Figure 4.9:** Comparison of Memory Consume of Second Experiment

Screen shot of the program code for second experiment is shown in Figure 4.10.

```
K-Nearest Neighbors Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack type mapping created...
Train and Test data labels created...
Decomposed features created...
Number of features used : 30
Time taken to perform 5-fold cross validation : 6.673927
Cross validation scores :
[0.9959519 0.99634863 0.99591173 0.99662605 0.9969039 ]
Mean score of 5-fold cross validation : 0.996348
Time taken to train final model : 0.566747
Predictions made using final model...
Time taken to make predictions on test data : 0.923757
Memory used : 0.307774 GB CPU usage : 36.900000
Confusion matrix :
[[13445    1    1    2    0]
 [    4  2285    0    0    0]
 [    1    0  9233    0    0]
 [    0    0    0   209    0]
 [    0    0    0    1   10]]
Accuracy score on test data is : 0.999603
For normal, Detection Rate is % 99.97025801174809
For probe, Detection Rate is % 99.82525120139799
For dos, Detection Rate is % 99.98917045700671
For r2l, Detection Rate is % 100.0
For u2r, Detection Rate is % 90.9090909090909
DR is % 98.13875411584874
```

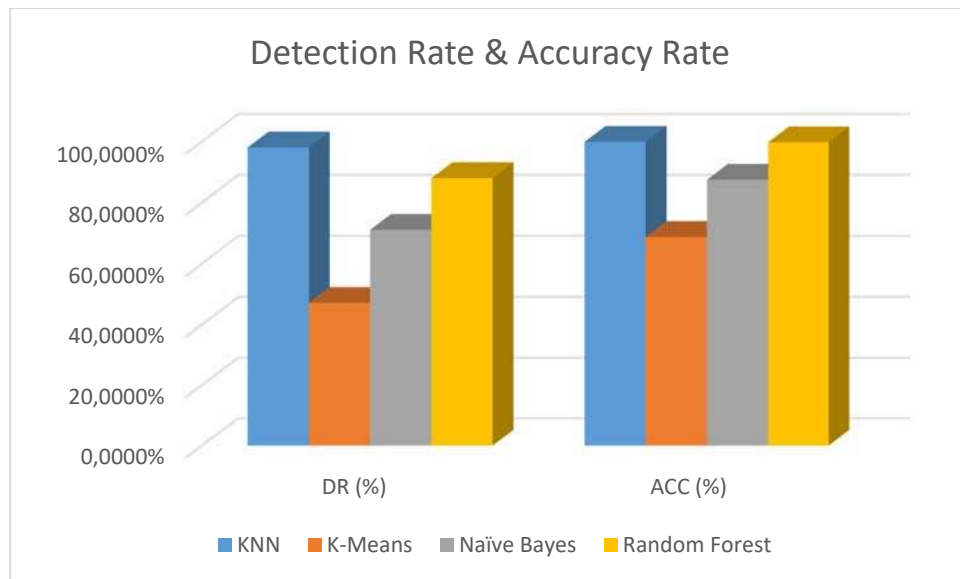
**Figure 4.10:** Experiment 2 of KNN

The results of third experiment can be seen in Table 4.5. The third experimental result indicated that KNN algorithm achieved a DR percent of 98.1249 % as highest compared with other algorithms. Highest ACC Rate is 99.9603 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by KNN algorithm, highest DR percent for Probe achieved by RF algorithm, highest DR percent for DoS achieved by KNN algorithm, highest DR percent for R2L achieved by KNN algorithm, and highest DR percent for U2R achieved by KNN algorithm. Lowest time taken to train is 0.159209 second that achieved by NB algorithm and lowest time taken to test is 0.006390 second achieved by KM algorithm. In addition, lowest memory usage is 0.289413 GB that achieved by RF algorithm.

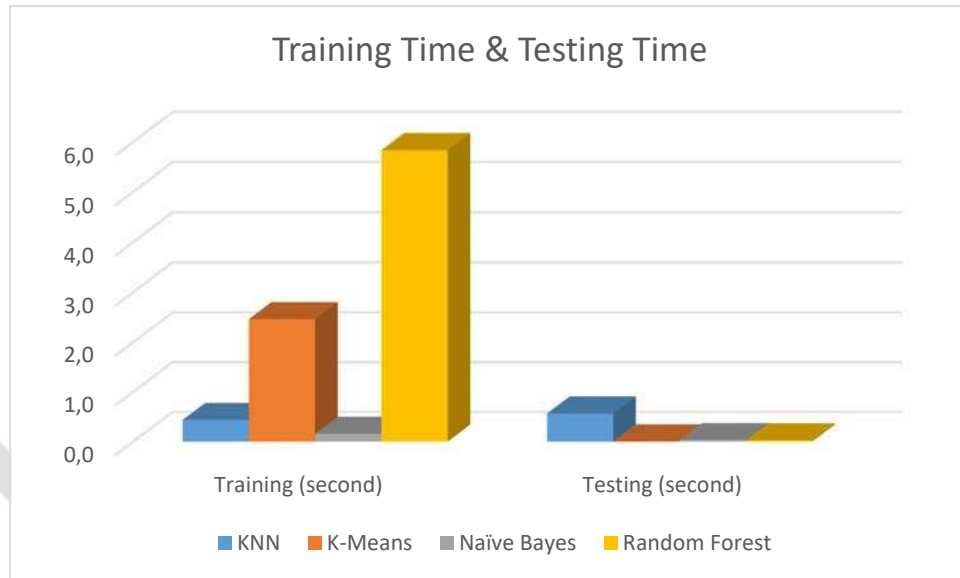
**Table 4.5:** Experiment 3

|                                | <b>KNN</b> | <b>KM</b> | <b>NB</b> | <b>RF</b> |
|--------------------------------|------------|-----------|-----------|-----------|
| <b>DR (%)</b>                  | 98.1249    | 47.1649   | 71.1073   | 88.0633   |
| <b>ACC (%)</b>                 | 99.9603    | 68.752    | 87.5318   | 99.8492   |
| <b>DR For Normal (%)</b>       | 99.9776    | 69.7895   | 86.6979   | 99.9108   |
| <b>DR For Probe (%)</b>        | 99.7378    | 44.9104   | 89.2093   | 99.8253   |
| <b>DR For DoS (%)</b>          | 100        | 74.713    | 88.4882   | 99.9567   |
| <b>DR For R2L (%)</b>          | 100        | 0.9569    | 36.5957   | 95.1691   |
| <b>DR For U2R (%)</b>          | 90.909     | 45.4545   | 54.5455   | 45.4545   |
| <b>Mean Of Cross V. (%)</b>    | 99.5793    | 1.6572    | 87.5497   | 99.5428   |
| <b>Time For Train (second)</b> | 0.438048   | 2.465096  | 0.159209  | 5.836239  |
| <b>Time For Test (second)</b>  | 0.568895   | 0.00639   | 0.033312  | 0.026006  |
| <b>Memory (GB)</b>             | 0.295273   | 0.383064  | 0.289391  | 0.289413  |

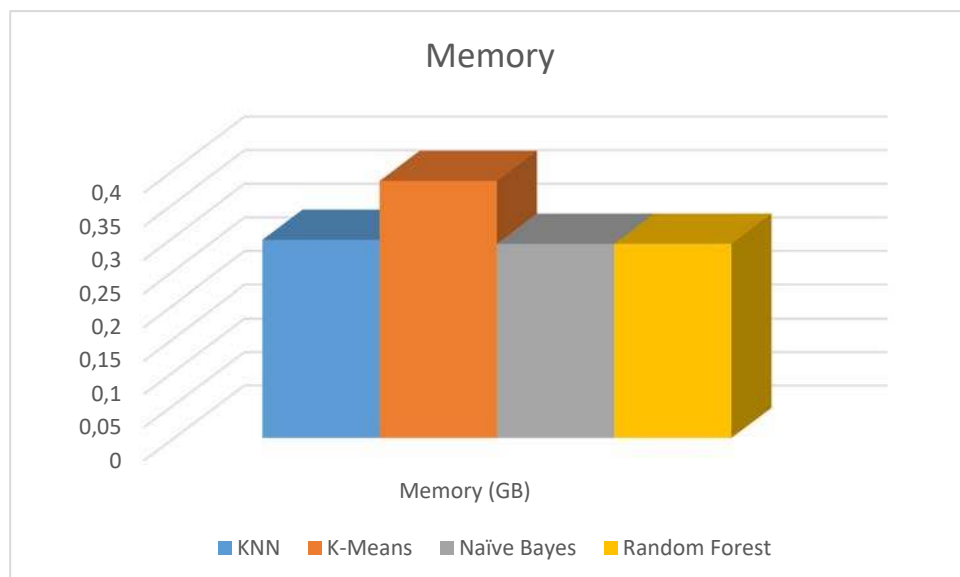
The comparison of DR and ACC ratios of all algorithms with each other is shown in Figure 4.11. The comparison of training and testing times of all algorithms with each other is shown in Figure 4.12. The comparison of memory consume of all algorithms with each other is shown in Figure 4.13.

**Figure 4.11:** Comparison of DR & ACC of Third Experiment





**Figure 4.12:** Comparison of Training and Testing Time of Third Experiment



**Figure 4.13:** Comparison of Memory Consume of Third Experiment



Screen shot of the program code for third experiment is shown in Figure 4.14.

```
K-Nearest Neighbors Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack type mapping created...
Train and Test data labels created...
Decomposed features created...
Number of features used : 20
Time taken to perform 5-fold cross validation : 4.706389
Cross validation scores :
[0.99555503 0.99567392 0.99531635 0.99595126 0.99646727]
Mean score of 5-fold cross validation : 0.995793
Time taken to train final model : 0.438048
Predictions made using final model...
Time taken to make predictions on test data : 0.568895
Memory used : 0.295273 GB CPU usage : 33.200000
Confusion matrix :
[[13446    0     2     1     0]
 [    5  2283     1     0     0]
 [    0     0  9234     0     0]
 [    0     0     0   209     0]
 [    0     0     0     1    10]]
Accuracy score on test data is : 0.999603
For normal, Detection Rate is % 99.97769350881106
For probe, Detection Rate is % 99.73787680209699
For dos, Detection Rate is % 100.0
For r2l, Detection Rate is % 100.0
For u2r, Detection Rate is % 90.9090909090909
DR is % 98.1249322439998
```

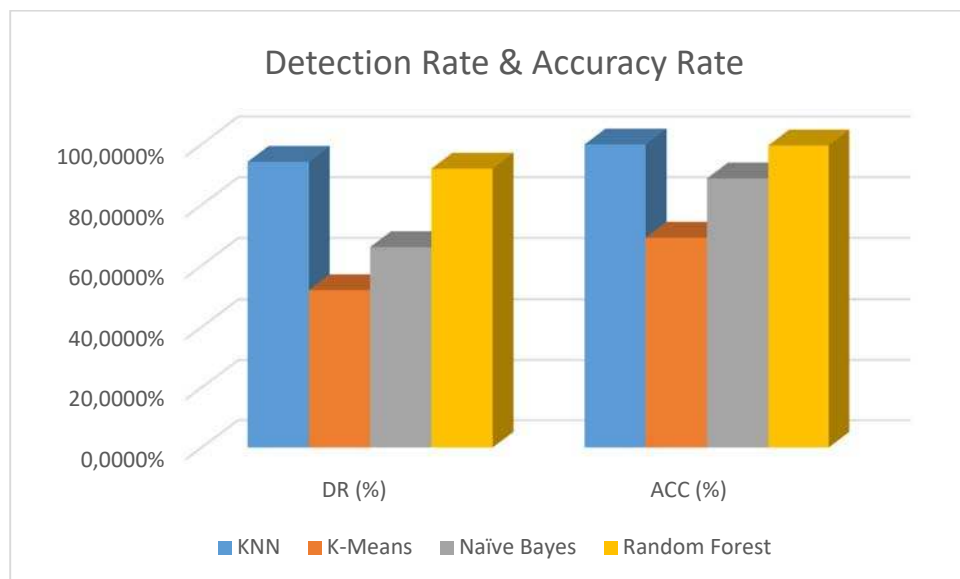
**Figure 4.14:** Experiment 3 of KNN

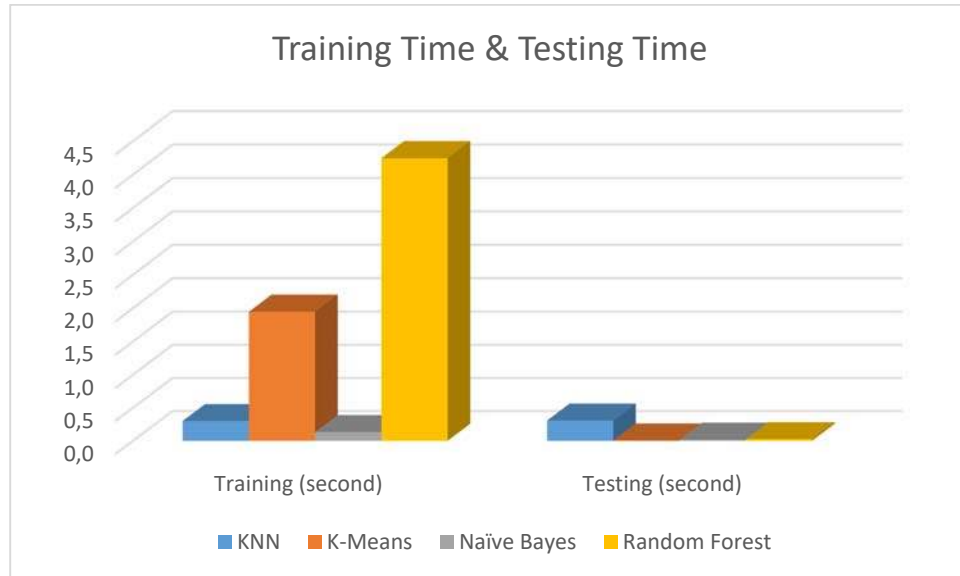
The results of fourth experiment can be seen in Table 4.6. The fourth experimental result indicated that KNN achieved a DR percent of 94.1806 % as highest compared with other algorithms. Highest ACC Rate is 99.8849 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by KNN algorithm, highest DR percent for Probe achieved by RF algorithm, highest DR percent for DoS achieved by KNN algorithm, highest DR percent for R2L achieved by KNN algorithm, and highest DR percent for U2R achieved by KNN algorithm. Lowest time taken to train is 0.132482 second that achieved by NB algorithm and lowest time taken to test is 0.005167 second achieved by KM algorithm. In addition, lowest memory usage is 0.277580 GB that achieved by NB algorithm.

**Table 4.6:** Experiment 4

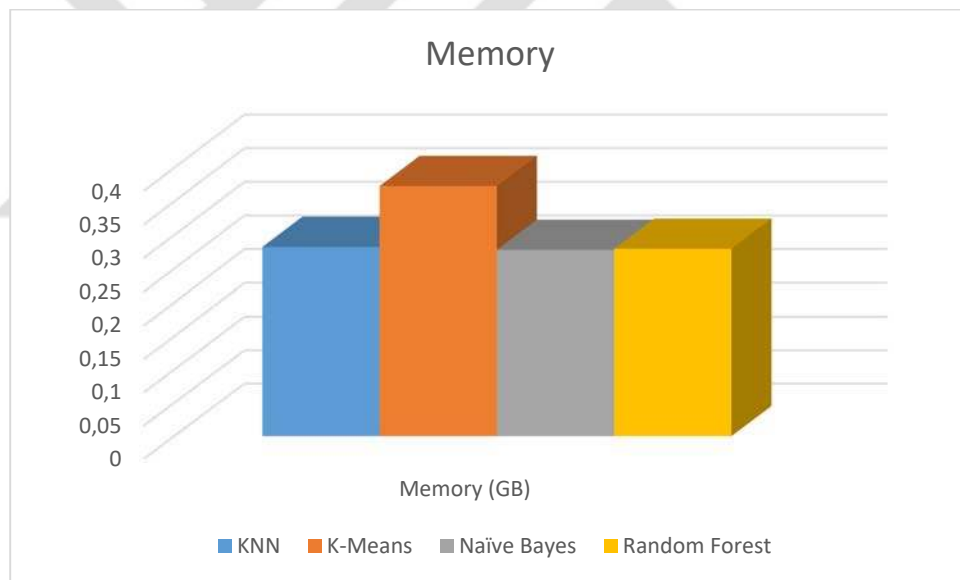
|                                | KNN      | KM       | NB       | RF       |
|--------------------------------|----------|----------|----------|----------|
| <b>DR (%)</b>                  | 94.1806  | 51.9481  | 66.115   | 91.9289  |
| <b>ACC (%)</b>                 | 99.8849  | 69.2323  | 88.7028  | 99.5475  |
| <b>DR For Normal (%)</b>       | 99.8735  | 69.7821  | 89.3895  | 99.487   |
| <b>DR For Probe (%)</b>        | 99.7378  | 44.9104  | 84.7532  | 99.7816  |
| <b>DR For DoS (%)</b>          | 100      | 74.713   | 89.3546  | 99.8484  |
| <b>DR For R2L (%)</b>          | 98.5645  | 61.244   | 30.7143  | 96.8912  |
| <b>DR For U2R (%)</b>          | 72.7272  | 9.0909   | 36.3636  | 63.6364  |
| <b>Mean Of Cross V. (%)</b>    | 99.4721  | 1.3816   | 88.9175  | 99.3816  |
| <b>Time For Train (second)</b> | 0.300025 | 1.947047 | 0.132482 | 4.24552  |
| <b>Time For Test (second)</b>  | 0.306337 | 0.005167 | 0.016904 | 0.024888 |
| <b>Memory (GB)</b>             | 0.282036 | 0.372643 | 0.27758  | 0.279289 |

The comparison of DR and ACC ratios of all algorithms with each other is shown in Figure 4.15. The comparison of training and testing times of all algorithms with each other is shown in Figure 4.16. The comparison of memory consume of all algorithms with each other is shown in Figure 4.17.

**Figure 4.15:** Comparison of DR & ACC of Fourth Experiment



**Figure 4.16:** Comparison of Training and Testing Time of Fourth Experiment



**Figure 4.17:** Comparison of Memory Consume of Fourth Experiment

Screen shot of the program code for fourth experiment is shown in Figure 4.18.

```
K-Nearest Neighbors Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack type mapping created...
Train and Test data labels created...
Decomposed features created...
Number of features used : 10
Time taken to perform 5-fold cross validation : 2.616509
Cross validation scores :
[0.99452316 0.99464201 0.99464158 0.99476045 0.9950383 ]
Mean score of 5-fold cross validation : 0.994721
Time taken to train final model : 0.300025
Predictions made using final model...
Time taken to make predictions on test data : 0.306337
Memory used : 0.282036 GB CPU usage : 16.100000
Confusion matrix :
[[13432    0    15    1    1]
 [    5  2283    1    0    0]
 [    0    0  9234    0    0]
 [    3    0    0   206    0]
 [    2    0    0    1    8]]
Accuracy score on test data is : 0.998849
For normal, Detection Rate is % 99.87359654992936
For probe, Detection Rate is % 99.73787680209699
For dos, Detection Rate is % 100.0
For r2l, Detection Rate is % 98.56459330143541
For u2r, Detection Rate is % 72.72727272727273
DR is % 94.1806678761469
```

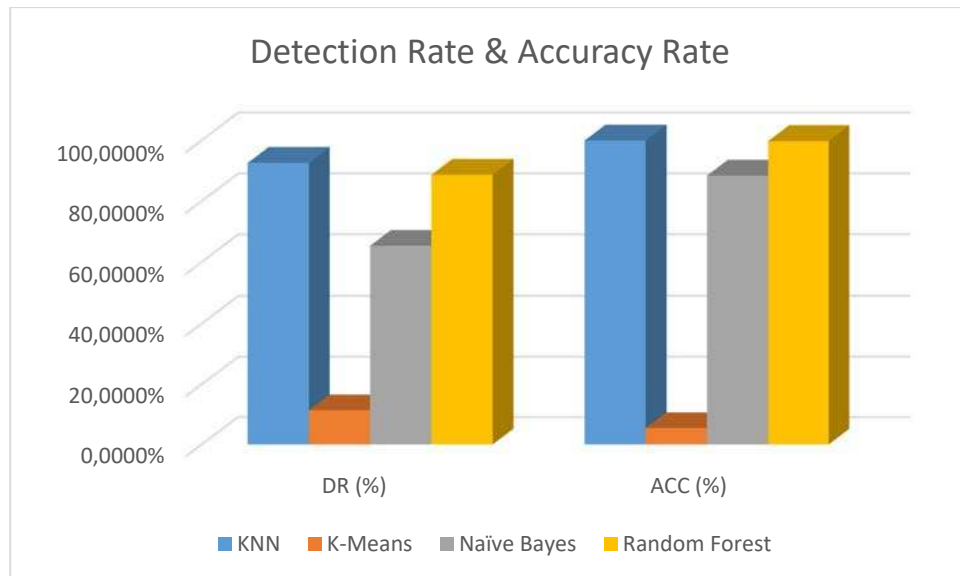
**Figure 4.18:** Experiment 4 of KNN

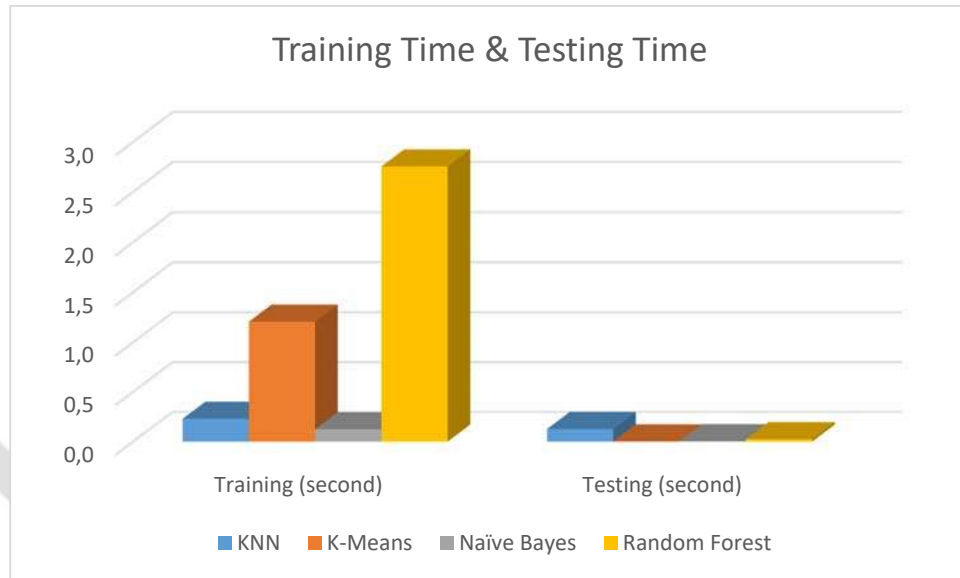
The results of fifth experiment can be seen in Table 4.7. The fifth experimental result indicated that KNN achieved a DR percent of 92.5403 % as highest compared with other algorithms. Highest ACC Rate is 99.8690 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by KNN algorithm, highest DR percent for Probe achieved by KNN algorithm, highest DR percent for DoS achieved by KNN algorithm, highest DR percent for R2L achieved by KNN algorithm, and highest DR percent for U2R achieved by KNN algorithm. Lowest time taken to train is 0.124872 second that achieved by NB algorithm and lowest time taken to test is 0.004403 second achieved by KM algorithm. In addition, lowest memory usage is 0.272610 GB that achieved by NB algorithm.

**Table 4.7:** Experiment 5

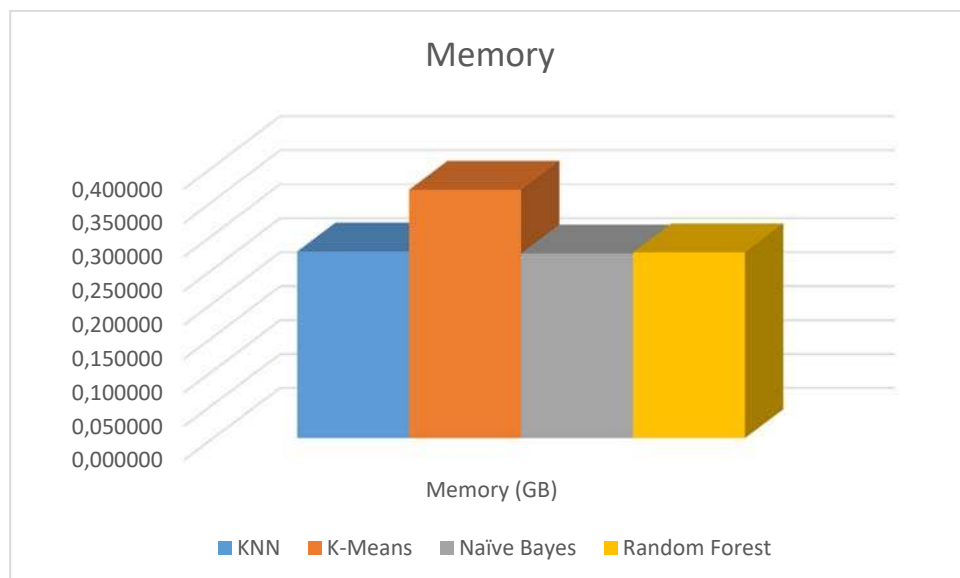
|                                | KNN      | KM       | NB       | RF       |
|--------------------------------|----------|----------|----------|----------|
| <b>DR (%)</b>                  | 92.5403  | 11.3342  | 65.3491  | 88.644   |
| <b>ACC (%)</b>                 | 99.869   | 5.4224   | 88.3773  | 99.742   |
| <b>DR For Normal (%)</b>       | 99.9033  | 0.2007   | 90.4826  | 99.8736  |
| <b>DR For Probe (%)</b>        | 99.7378  | 49.5849  | 80.166   | 99.6068  |
| <b>DR For DoS (%)</b>          | 99.9025  | 2.1009   | 89.4304  | 99.8159  |
| <b>DR For R2L (%)</b>          | 99.5215  | 4.7846   | 66.6667  | 98.4694  |
| <b>DR For U2R (%)</b>          | 63.6363  | 0        | 0        | 45.4545  |
| <b>Mean Of Cross V. (%)</b>    | 99.3499  | 0.8269   | 88.4578  | 99.2133  |
| <b>Time For Train (second)</b> | 0.231077 | 1.207119 | 0.124872 | 2.755741 |
| <b>Time For Test (second)</b>  | 0.127503 | 0.004403 | 0.005636 | 0.024517 |
| <b>Memory (GB)</b>             | 0.27552  | 0.366207 | 0.27261  | 0.274162 |

The comparison of DR and ACC ratios of all algorithms with each other is shown in Figure 4.19. The comparison of training and testing times of all algorithms with each other is shown in Figure 4.20. The comparison of memory consume of all algorithms with each other is shown in Figure 4.21.

**Figure 4.19:** Comparison of DR & ACC of Fifth Experiment



**Figure 4.20:** Comparison of Training and Testing Time of Fifth Experiment



**Figure 4.21:** Comparison of Memory Consume of Fifth Experiment



Screen shot of the program code for fifth experiment is shown in Figure 4.22.

```
K-Nearest Neighbors Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack type mapping created...
Train and Test data labels created...
Decomposed features created...
Number of features used : 5
Time taken to perform 5-fold cross validation : 1.753162
Cross validation scores :
[0.99297535 0.99364979 0.99345082 0.99368872 0.99372842]
Mean score of 5-fold cross validation : 0.993499
Time taken to train final model : 0.231077
Predictions made using final model...
Time taken to make predictions on test data : 0.127503
Memory used : 0.275520 GB CPU usage : 15.300000
Confusion matrix :
[[13436    1    3    4    5]
 [    5  2283    1    0    0]
 [    9    0  9225    0    0]
 [    1    0    0   208    0]
 [    2    0    0    2    7]]
Accuracy score on test data is : 0.998690
For normal, Detection Rate is % 99.90333853818127
For probe, Detection Rate is % 99.73787680209699
For dos, Detection Rate is % 99.90253411306043
For r2l, Detection Rate is % 99.52153110047847
For u2r, Detection Rate is % 63.63636363636363
DR is % 92.54032883803616
```

**Figure 4.22:** Experiment 5 of KNN

**Scenario 2:** Testing the scalability of NADS by using different size of KDD CUP data set.

**Scenario 2 Implementation:** In order to see the scalability of NADS 3 different size of KDD CUP data set were used as fallow:

- 1- 125,793 vector of KDD dataset were used to train NADS and 25,192 lines for testing.
- 2- 494,021 vector of KDD dataset were used to train NADS and 25,192 lines for testing.
- 3- 1,000,000 vector of KDD dataset were used to train NADS and 25,192 lines for testing.

As a result, the first experimental indicated that KNN achieved a DR percent of 98.0379 % as highest compared with other algorithms. Highest ACC Rate is 99.9603 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by RF algorithm, highest DR percent for Probe achieved by RF algorithm, highest DR percent for DoS achieved by KNN and RF algorithms, highest DR percent for R2L achieved by KNN algorithm, and highest DR percent for U2R achieved by KNN algorithm. Lowest time taken to train is 0.128126 second that achieved by NB algorithm and lowest time taken to test is 0.08962 second achieved by NB algorithm. In addition, lowest memory usage is 0.307209 GB that achieved by KM algorithm.

The second experimental result indicated that RF algorithm achieved a DR percent of 94,8460 % as highest compared with other algorithms. Highest ACC Rate is 99.903 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by RF algorithm, Highest DR percent for Probe achieved by KNN algorithm, highest DR percent for DoS achieved by KNN algorithm, highest DR percent for R2L achieved RF algorithm, and highest DR percent for U2R achieved by KNN and KM algorithms. Lowest time taken to train is 0.517239 second that achieved by NB algorithm and lowest time taken to test is 0.006631 second achieved by NB algorithm. In addition, lowest memory usage is 0.246151 GB that achieved by KNN algorithm.

The third experimental result indicated that RF algorithm achieved a DR percent of 77.4958 % as highest compared with other algorithms. Highest ACC Rate is 91.5092 % that achieved by KNN algorithm. Highest DR percent for Normal achieved by RF algorithm, highest DR percent for Probe achieved by KNN algorithm, highest DR percent for DoS achieved by KNN algorithm, highest DR percent for R2L achieved by RF algorithm, and highest DR percent for U2R achieved by KM algorithm. Lowest time taken to train is 1.079136 second that achieved by NB algorithm and lowest time taken to test is 0.006515 second achieved by NB algorithm. In addition, lowest memory usage is 0.456940 GB that achieved by KNN algorithm.

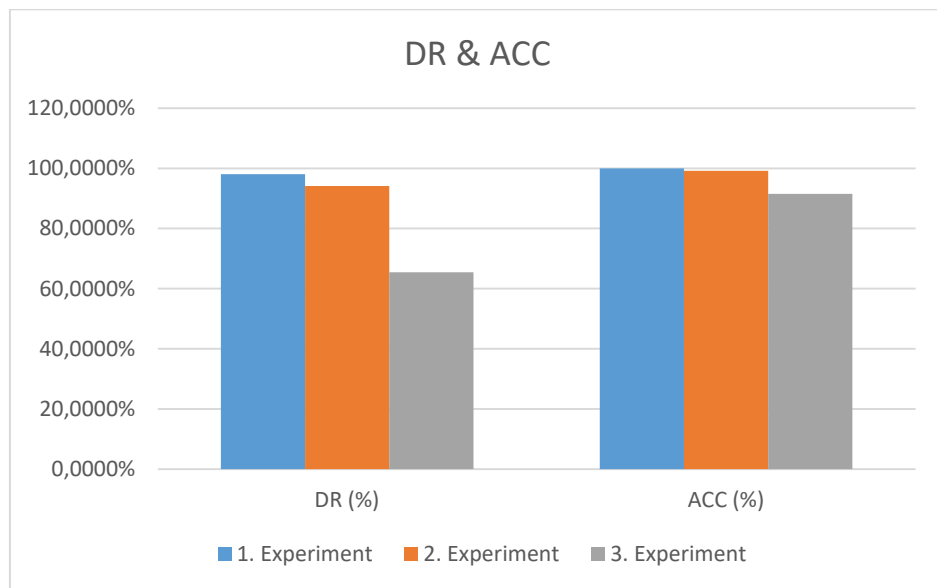


The results of experiments for KNN can be seen in Table 4.8.

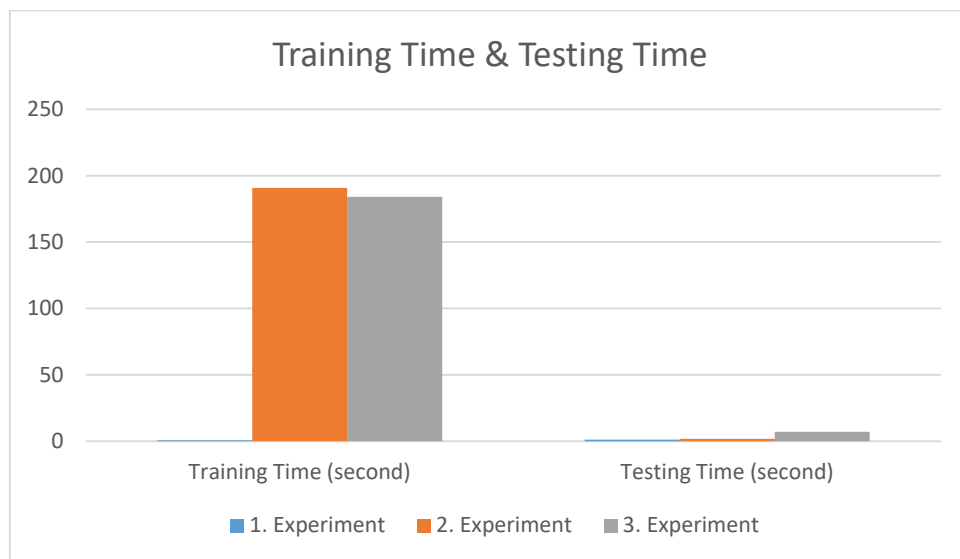
**Table 4.8:** Scalability Experiments for KNN Algorithm

|                         | 1. Exp.  | 2. Exp.    | 3. Exp.    |
|-------------------------|----------|------------|------------|
| DR (%)                  | 98.0379  | 94.1442    | 65.4249    |
| ACC (%)                 | 99.9603  | 99.1862    | 91.5092    |
| DR For Normal (%)       | 99.9776  | 99.5538    | 96.9737    |
| DR For Probe (%)        | 99.7815  | 98.2088    | 96.8108    |
| DR For DoS (%)          | 100      | 99.2744    | 84.0805    |
| DR For R2L (%)          | 99.5215  | 82.7751    | 12.9186    |
| DR For U2R (%)          | 90.909   | 90.909     | 36.3636    |
| Mean Of Cross V. (%)    | 99.6539  | 95.2493    | 99.2649    |
| Time For Train (second) | 0.66159  | 190.729492 | 184.077617 |
| Time For Test (second)  | 1.139292 | 1.826953   | 7.139832   |
| Memory (GB)             | 0.304344 | 0.246151   | 0.45694    |

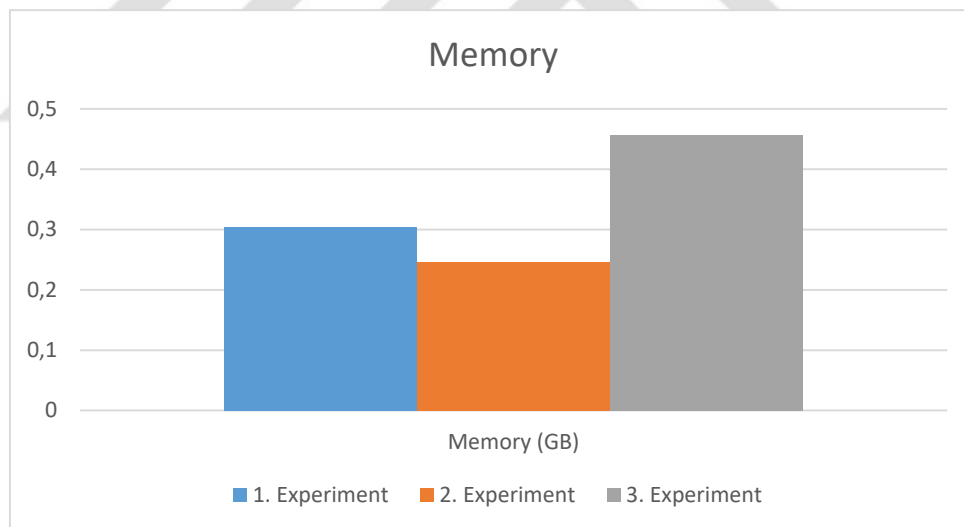
For KNN, the comparison of DR and ACC ratios of all experiments with each other is shown in Figure 4.23. The comparison of training and testing times of all experiments with each other is shown in Figure 4.24. The comparison of memory consume of all experiments with each other is shown in Figure 4.25.



**Figure 4.23:** Comparison of DR & ACC for KNN Algorithm



**Figure 4.24:** Comparison of Training and Testing Time for KNN Algorithm



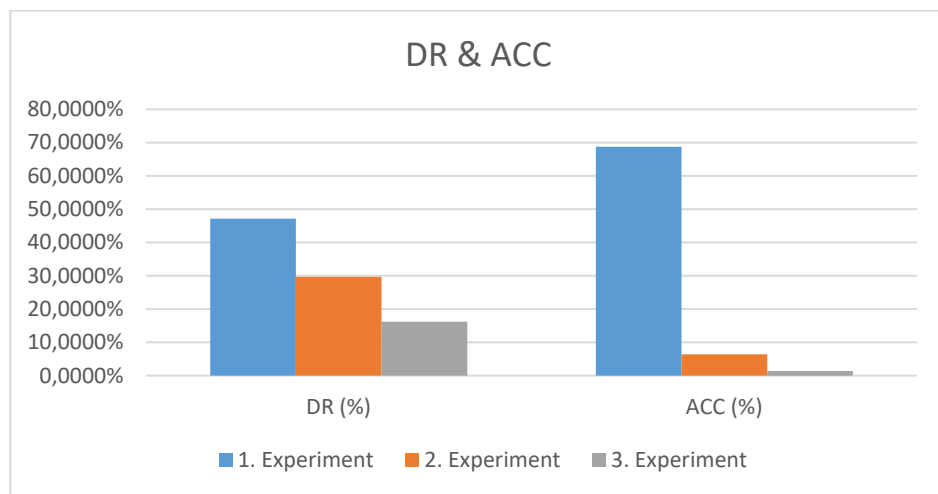
**Figure 4.25:** Comparison of Memory Consume KNN Algorithm

The results of experiments for KM can be seen in Table 4.9.

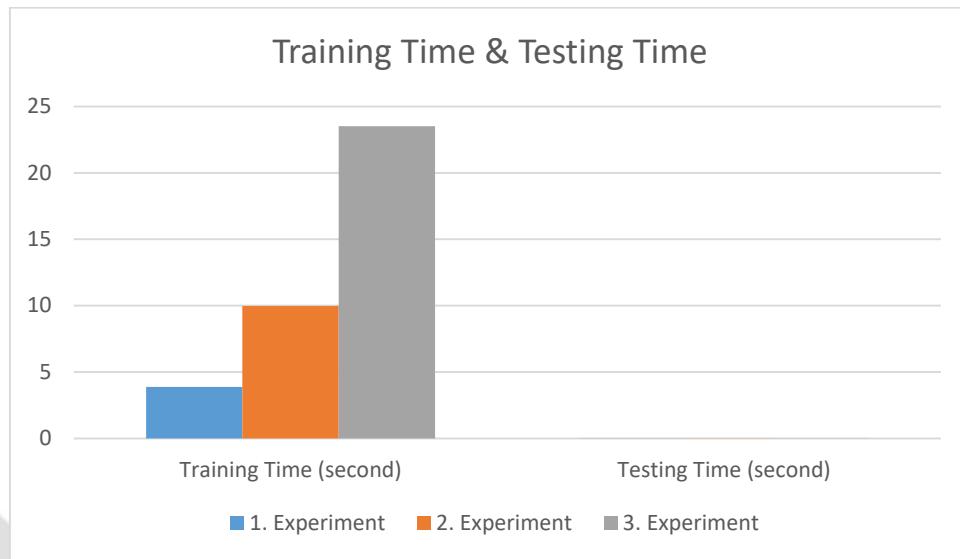
**Table 4.9:** Scalability Experiments for KM Algorithm

|                         | 1. Exp.  | 2. Exp.  | 3. Exp.   |
|-------------------------|----------|----------|-----------|
| DR (%)                  | 47.1663  | 29.6809  | 16.1885   |
| ACC (%)                 | 68.756   | 6.375    | 1.3774    |
| DR For Normal (%)       | 69.797   | 0.258    | 0.2899    |
| DR For Probe (%)        | 44.9104  | 47.4006  | 0         |
| DR For DoS (%)          | 74.713   | 5.0573   | 3.1405    |
| DR For R2L (%)          | 0.9569   | 4.7846   | 4.7846    |
| DR For U2R (%)          | 45.4545  | 90.909   | 72.7272   |
| Mean Of Cross V. (%)    | 1.6965   | 2.8465   | 6.2429    |
| Time For Train (second) | 3.872822 | 9.982122 | 23.515026 |
| Time For Test (second)  | 0.009377 | 0.009005 | 0.008977  |
| Memory (GB)             | 0.300694 | 0.82386  | 1.203556  |

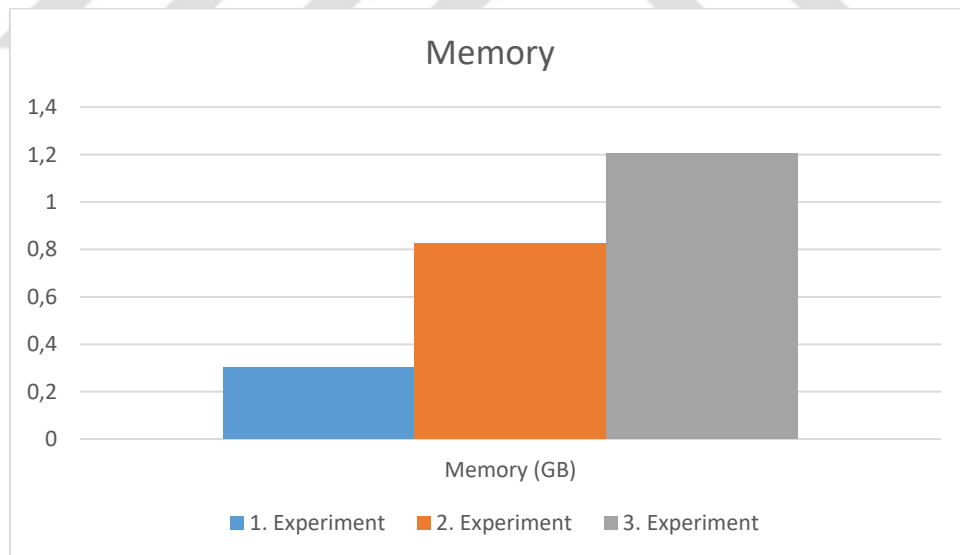
For KM, the comparison of DR and ACC ratios of all experiments with each other is shown in Figure 4.26. The comparison of training and testing times of all experiments with each other is shown in Figure 4.27. The comparison of memory consume of all experiments with each other is shown in Figure 4.28.



**Figure 4.26:** Comparison of DR & ACC for KM Algorithm



**Figure 4.27:** Comparison of Training and Testing Time for KM Algorithm



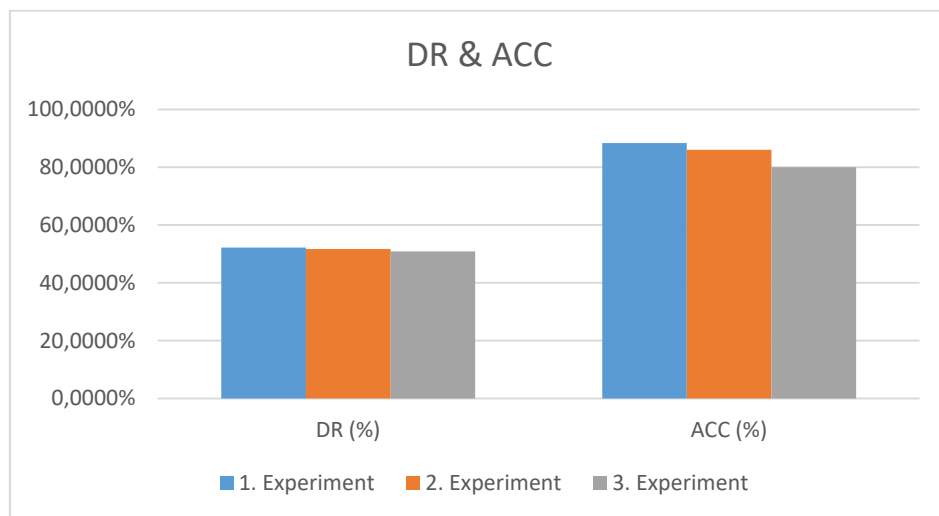
**Figure 4.28:** Comparison of Memory Consume KM Algorithm

The results of experiments for NB can be seen in Table 4.10.

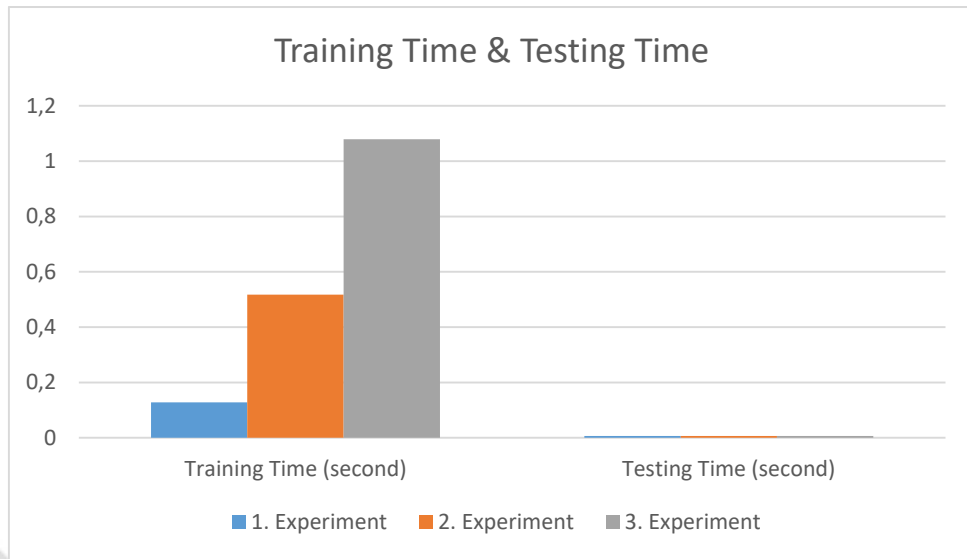
**Table 4.10:** Scalability Experiments for NB Algorithm

|                                | 1. Exp.  | 2. Exp.  | 3. Exp.  |
|--------------------------------|----------|----------|----------|
| <b>DR (%)</b>                  | 52.2071  | 51.6936  | 50.8574  |
| <b>ACC (%)</b>                 | 88.3773  | 86.0432  | 79.9817  |
| <b>DR For Normal (%)</b>       | 90.4825  | 94.5052  | 82.9727  |
| <b>DR For Probe (%)</b>        | 80.166   | 88.9035  | 96.1992  |
| <b>DR For DoS (%)</b>          | 89.4303  | 75.0596  | 73.381   |
| <b>DR For R2L (%)</b>          | 0.9569   | 0        | 1.7341   |
| <b>DR For U2R (%)</b>          | 0        | 0        | 0        |
| <b>Mean Of Cross V. (%)</b>    | 88.4578  | 93.2765  | 95.5095  |
| <b>Time For Train (second)</b> | 0.128126 | 0.517239 | 1.079136 |
| <b>Time For Test (second)</b>  | 0.006212 | 0.006631 | 0.006515 |
| <b>Memory (GB)</b>             | 0.505257 | 0.975502 | 1.12112  |

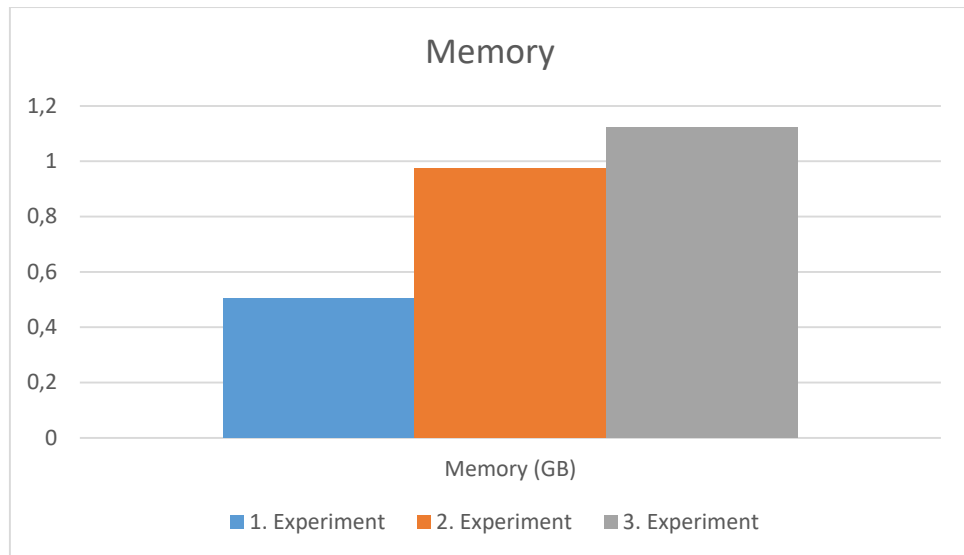
For NB, the comparison of DR and ACC ratios of all experiments with each other is shown in Figure 4.29. The comparison of training and testing times of all experiments with each other is shown in Figure 4.30. The comparison of memory consume of all experiments with each other is shown in Figure 4.31.



**Figure 4.29:** Comparison of DR & ACC for NB Algorithm



**Figure 4.30:** Comparison of Training and Testing Time for NB Algorithm



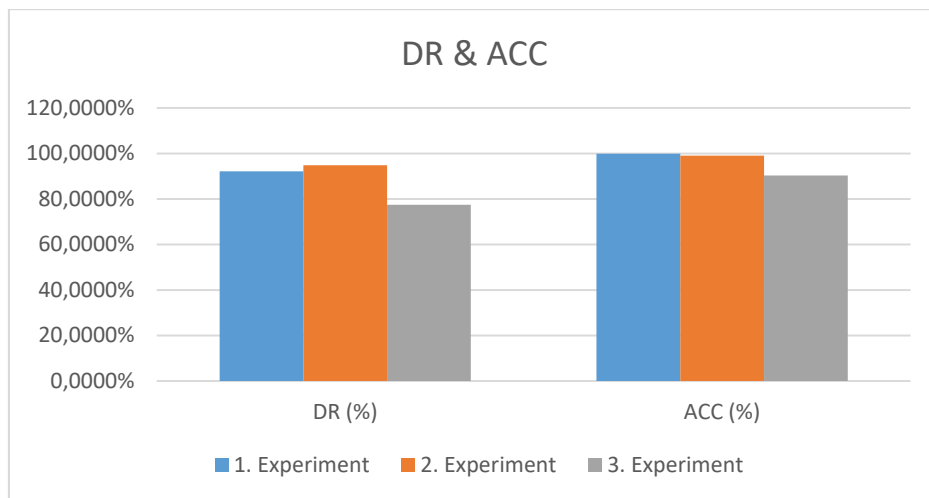
**Figure 4.31:** Comparison of Memory Consume NB Algorithm

The results of experiments for RF can be seen in Table 4.11.

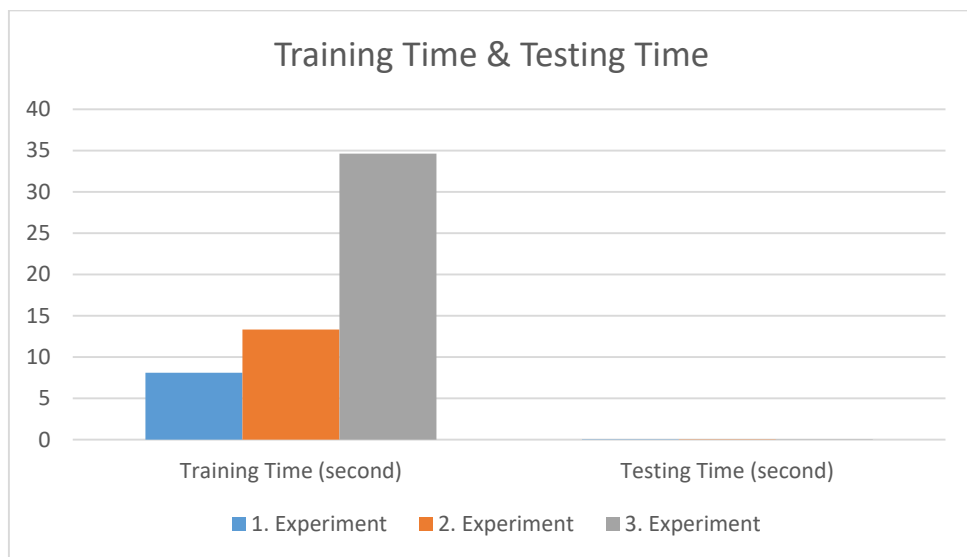
**Table 4.11:** Scalability Experiments for RF Algorithm

|                         | 1. Exp.  | 2. Exp.   | 3. Exp.   |
|-------------------------|----------|-----------|-----------|
| DR (%)                  | 92.2005  | 94.846    | 77.4958   |
| ACC (%)                 | 99.9166  | 99.087    | 90.3541   |
| DR For Normal (%)       | 99.9553  | 99.7769   | 97.6801   |
| DR For Probe (%)        | 99.8252  | 97.204    | 91.5684   |
| DR For DoS (%)          | 99.9783  | 99.112    | 81.2216   |
| DR For R2L (%)          | 97.6076  | 96.319    | 80.6452   |
| DR For U2R (%)          | 63.6363  | 81.8182   | 36.3636   |
| Mean Of Cross V. (%)    | 99.6483  | 95.606    | 99.4851   |
| Time For Train (second) | 8.095981 | 13.321269 | 34.635815 |
| Time For Test (second)  | 0.028039 | 0.029124  | 0.029161  |
| Memory (GB)             | 0.538746 | 0.969166  | 1.249306  |

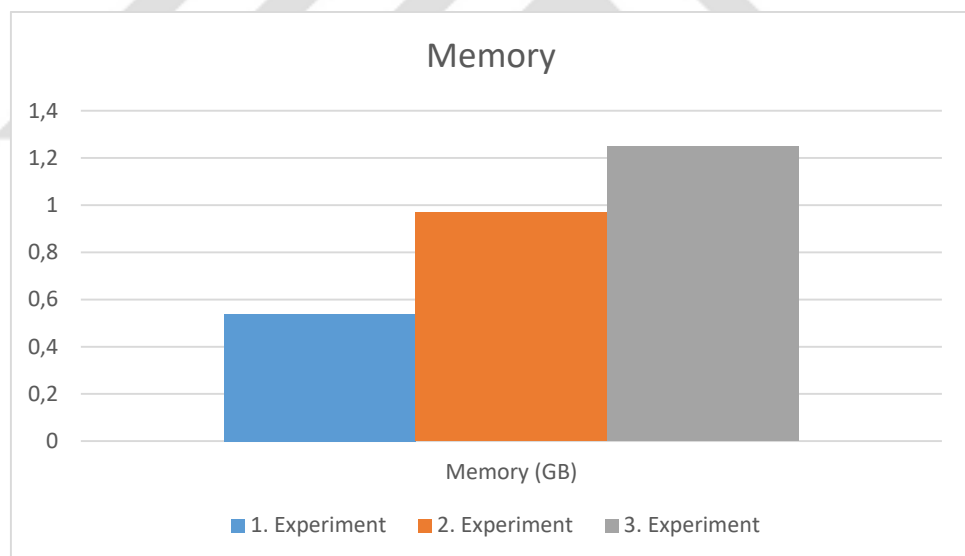
For RF, the comparison of DR and ACC ratios of all experiments with each other is shown in Figure 4.32. The comparison of training and testing times of all experiments with each other is shown in Figure 4.33. The comparison of memory consume of all experiments with each other is shown in Figure 4.34.



**Figure 4.32:** Comparison of DR & ACC for RF Algorithm



**Figure 4.33:** Comparison of Training and Testing Time for RF Algorithm



**Figure 4.34:** Comparison of Memory Consume RF Algorithm

As the size of the dataset used increases, the memory usage size of the algorithm will increase.



**Scenario 3:** Testing the effect of feature scaling on classification.

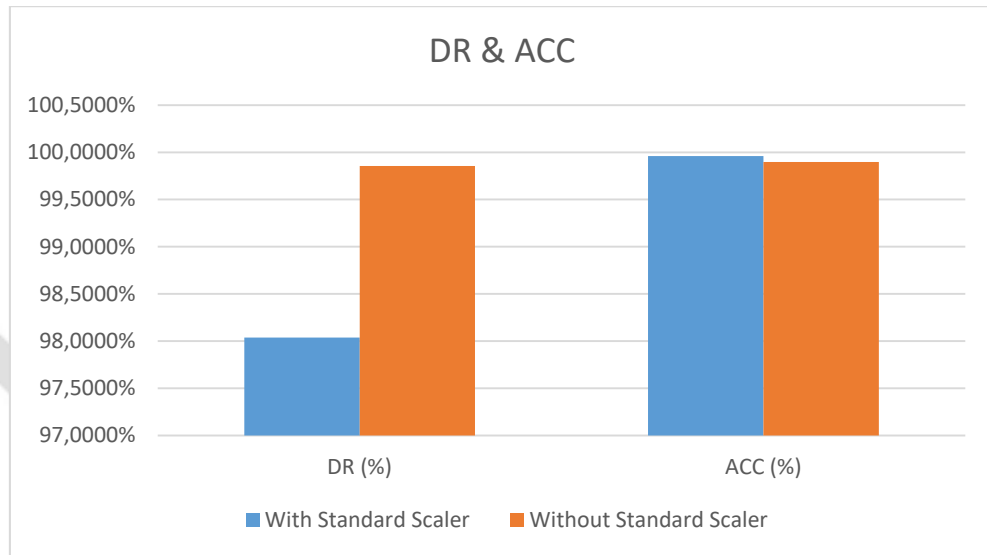
**Scenario 3 Implementation:**

In order to see the effect of feature scaling on classification, codes without feature scaling (Standard Scaler) are compared with the codes with feature scaling. The results of experiments for KNN can be seen in Table 4.12.

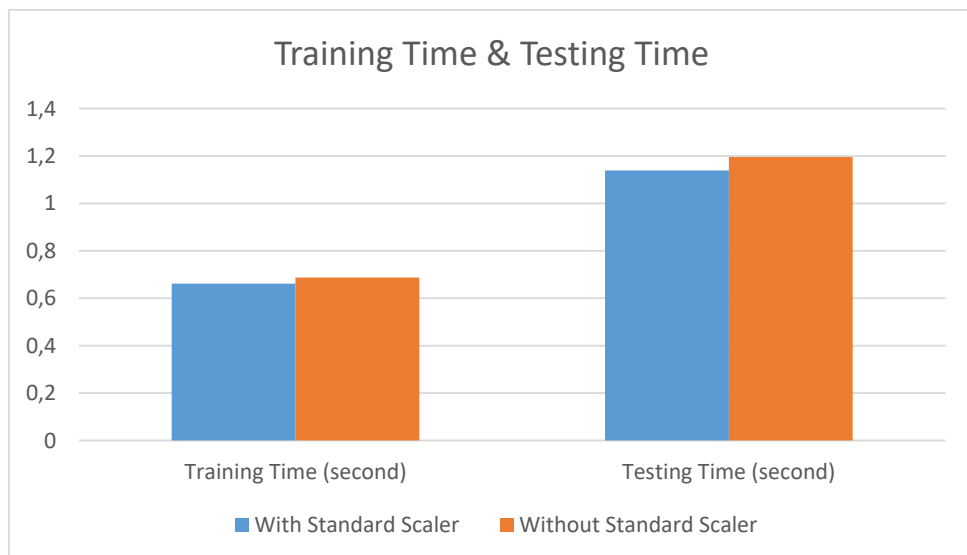
**Table 4.12:** Feature Scaling Experiments for KNN

|                         | With Standard Scale | Without Standard Scale |
|-------------------------|---------------------|------------------------|
| DR (%)                  | 98.0379             | 99.8544                |
| ACC (%)                 | 99.9603             | 99.8968                |
| DR For Normal (%)       | 99.9776             | 99.9703                |
| DR For Probe (%)        | 99.7815             | 99.3884                |
| DR For DoS (%)          | 100                 | 99.9134                |
| DR For R2L (%)          | 99.5215             | 100                    |
| DR For U2R (%)          | 90.909              | 100                    |
| Mean Of Cross V. (%)    | 99.6539             | 99.5467                |
| Time For Train (second) | 0.66159             | 0.687662               |
| Time For Test (second)  | 1.139292            | 1.195943               |
| Memory (GB)             | 0.304344            | 0.29446                |

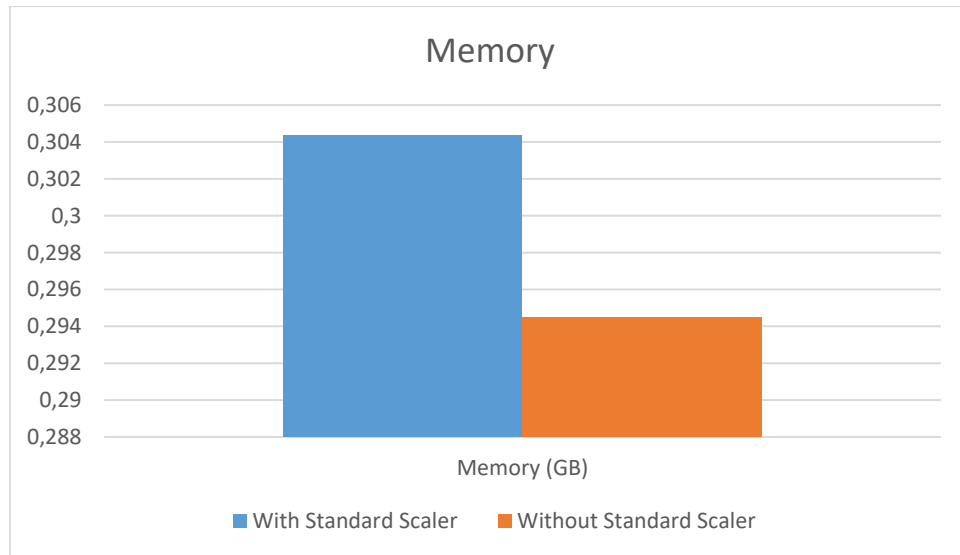
For KNN, the comparison of DR and ACC ratios of experiments with each other is shown in Figure 4.35. The comparison of training and testing times of experiments with each other is shown in Figure 4.36. The comparison of memory consume of experiments with each other is shown in Figure 4.37.



**Figure 4.35:** Comparison of DR & ACC for KNN Algorithm



**Figure 4.36:** Comparison of Training and Testing Time for KNN Algorithm



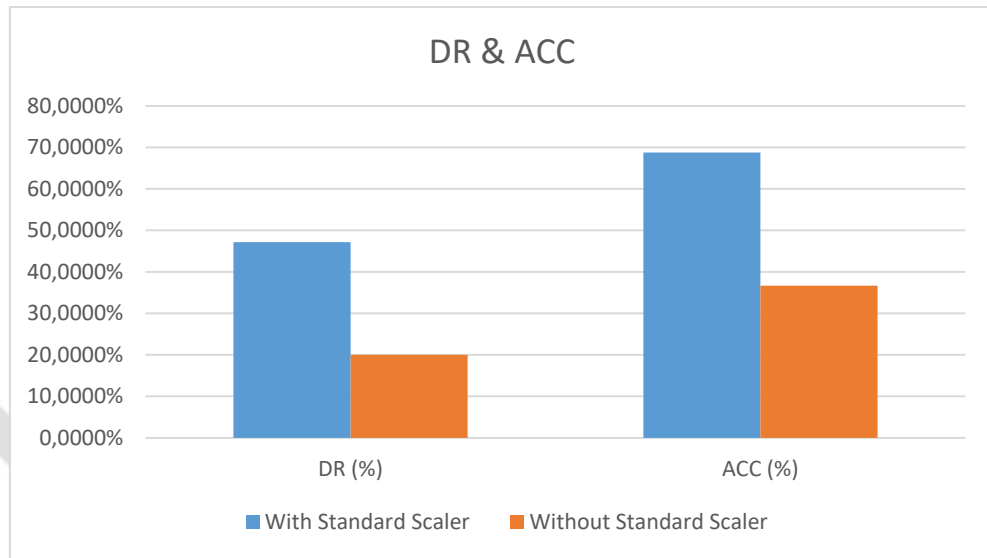
**Figure 4.37:** Comparison of Memory Consume KNN Algorithm

Without a standard scaler for KNN, the experiment gives a better result because numerical data is not being used in any mathematical process in KNN algorithm. The results of experiments for KM can be seen in Table 4.13.

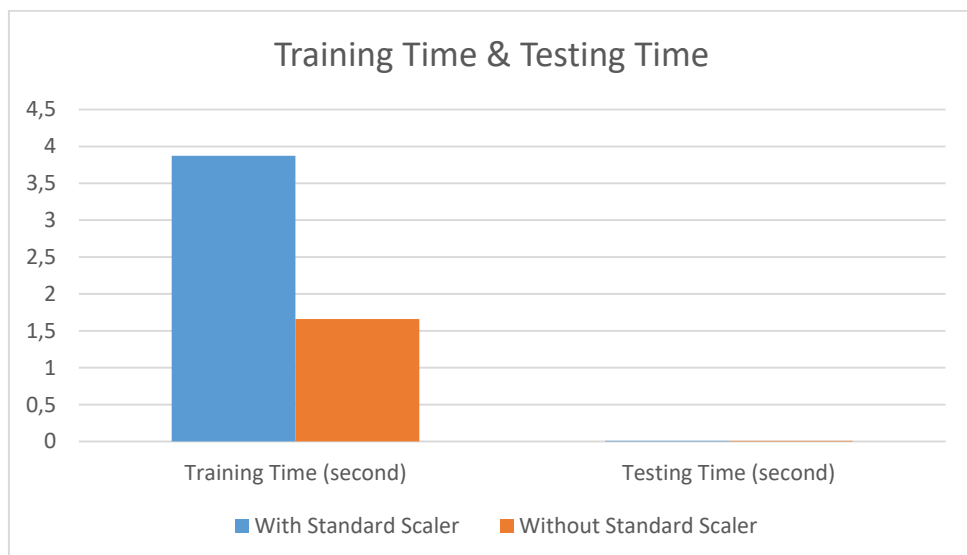
**Table 4.13:** Feature Scaling Experiments for KM Algorithm

|                         | With Standard Scale | Without Standard Scale |
|-------------------------|---------------------|------------------------|
| DR (%)                  | 47.1663             | 20                     |
| ACC (%)                 | 68.756              | 36.6545                |
| DR For Normal (%)       | 69.797              | 0                      |
| DR For Probe (%)        | 44.9104             | 0                      |
| DR For DoS (%)          | 74.713              | 100                    |
| DR For R2L (%)          | 0.9569              | 0                      |
| DR For U2R (%)          | 45.4545             | 0                      |
| Mean Of Cross V. (%)    | 1.6965              | 0.000000000000003      |
| Time For Train (second) | 3.872822            | 1.659936               |
| Time For Test (second)  | 0.009377            | 0.008598               |
| Memory(GB)              | 0.300694            | 0.413994               |

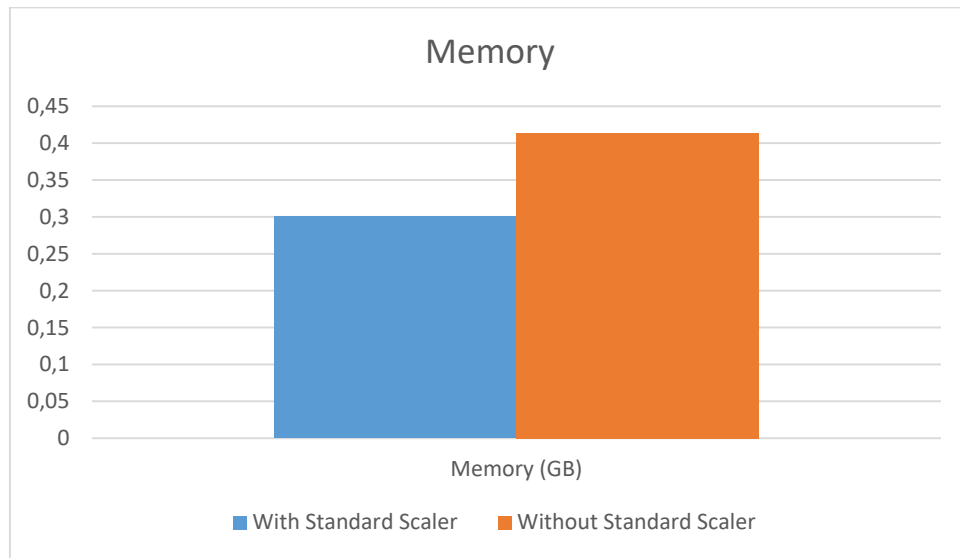
For KM, the comparison of DR and ACC ratios of experiments with each other is shown in Figure 4.38. The comparison of training and testing times of experiments with each other is shown in Figure 4.39. The comparison of memory consume of experiments with each other is shown in Figure 4.40.



**Figure 4.38:** Comparison of DR & ACC for KM Algorithm



**Figure 4.39:** Comparison of Training and Testing Time for KM Algorithm



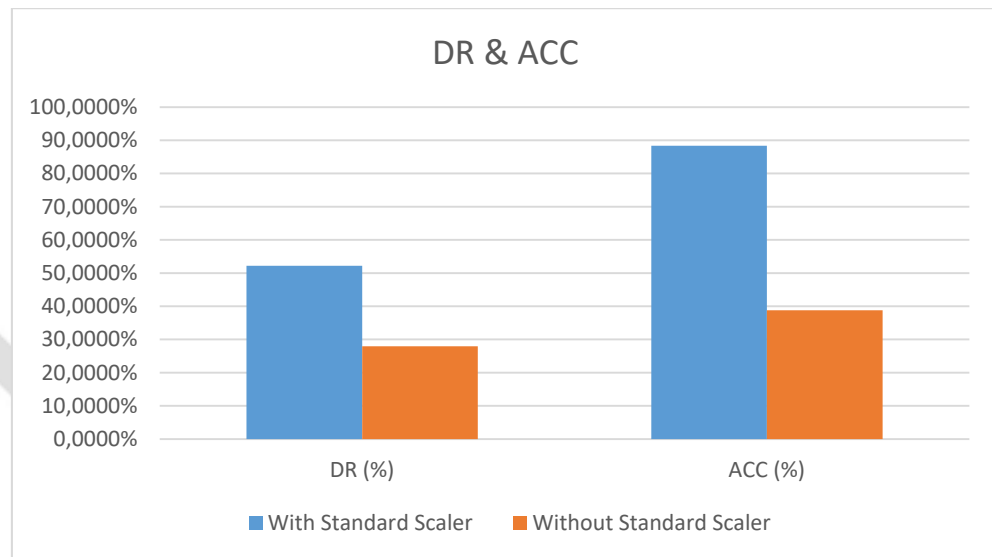
**Figure 4.40:** Comparison of Training and Testing Time for KM Algorithm

The results of experiments for NB can be seen in Table 4.14.

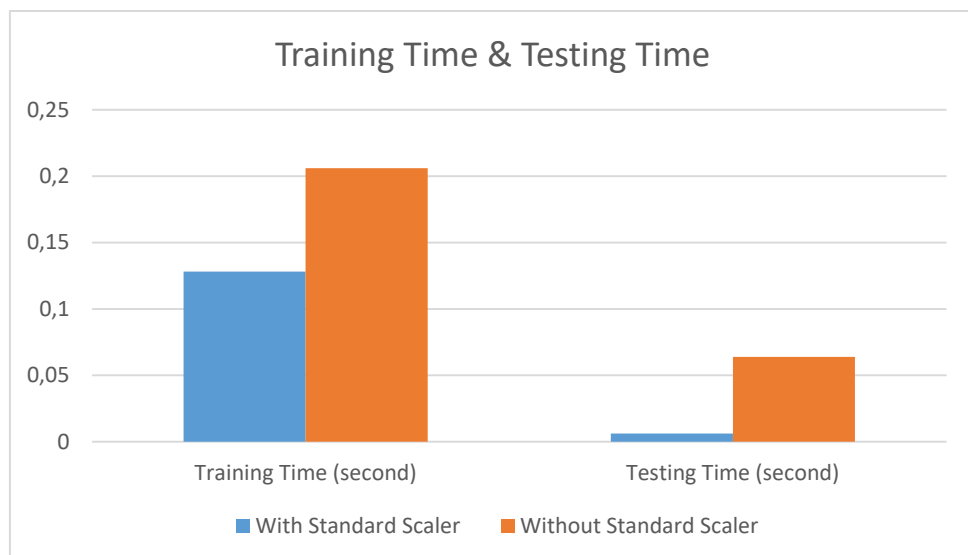
**Table 4.14:** Feature Scaling Experiments for NB

|                         | With Standard Scale | Without Standard Scale |
|-------------------------|---------------------|------------------------|
| DR (%)                  | 52.2071             | 27.9698                |
| ACC (%)                 | 88.3773             | 38.7861                |
| DR For Normal (%)       | 90.4825             | 3.9854                 |
| DR For Probe (%)        | 80.166              | 8.3879                 |
| DR For DoS (%)          | 89.4303             | 97.8774                |
| DR For R2L (%)          | 0.9569              | 2.3256                 |
| DR For U2R (%)          | 0                   | 27.2727                |
| Mean Of Cross V. (%)    | 88.4578             | 38.8377                |
| Time For Train (second) | 0.128126            | 0.206092               |
| Time For Test (second)  | 0.006212            | 0.063926               |
| Memory (GB)             | 0.505257            | 0.312431               |

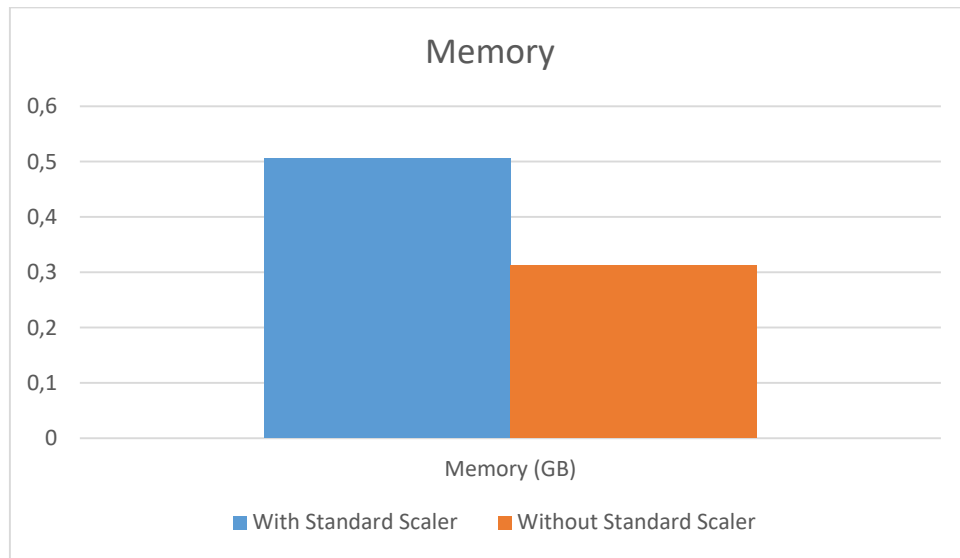
For NB, the comparison of DR and ACC ratios of experiments with each other is shown in Figure 4.41. The comparison of training and testing times of experiments with each other is shown in Figure 4.42. The comparison of memory consume of experiments with each other is shown in Figure 4.43.



**Figure 4.41:** Comparison of DR & ACC for NB Algorithm



**Figure 4.42:** Comparison of Training and Testing Time for NB Algorithm



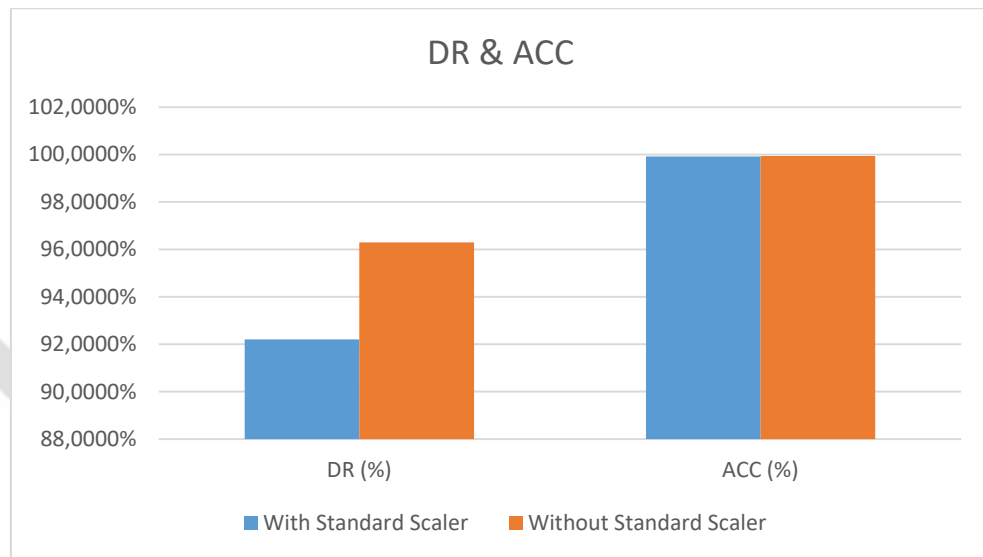
**Figure 4.43:** Comparison of Memory Consume NB Algorithm

The results of experiments for RF can be seen in Table 4.15.

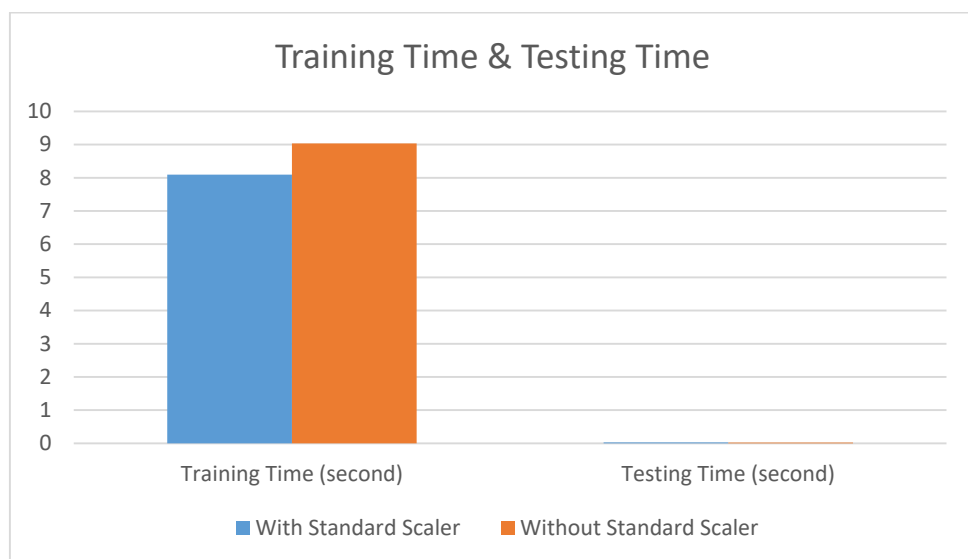
**Table 4.15:** Feature Scaling Experiments for RF Algorithm

|                                | With Standard Scale | Without Standard Scale |
|--------------------------------|---------------------|------------------------|
| <b>DR (%)</b>                  | 92.2005             | 96.2967                |
| <b>ACC (%)</b>                 | 99.9166             | 99.9365                |
| <b>DR For Normal (%)</b>       | 99.9553             | 99.9926                |
| <b>DR For Probe (%)</b>        | 99.8252             | 99.6942                |
| <b>DR For DoS (%)</b>          | 99.9783             | 99.9783                |
| <b>DR For R2L (%)</b>          | 97.6076             | 100                    |
| <b>DR For U2R (%)</b>          | 63.6363             | 81.8182                |
| <b>Mean Of Cross V. (%)</b>    | 99.6483             | 99.7904                |
| <b>Time For Train (second)</b> | 8.095981            | 9.033393               |
| <b>Time For Test (second)</b>  | 0.028039            | 0.025188               |
| <b>Memory (GB)</b>             | 0.538746            | 0.312523               |

For RF, the comparison of DR and ACC ratios of experiments with each other is shown in Figure 4.44. The comparison of training and testing times of experiments with each other is shown in Figure 4.45. The comparison of memory consume of experiments with each other is shown in Figure 4.46.

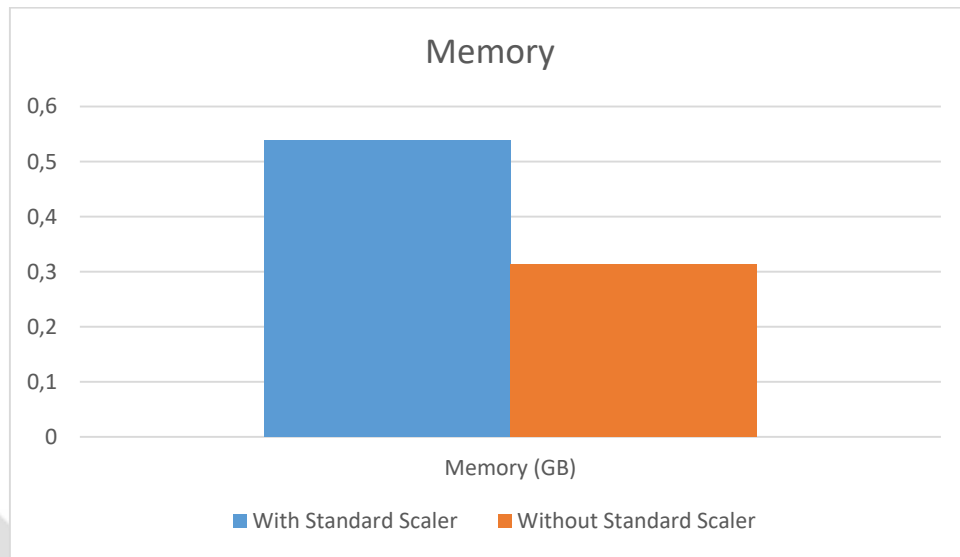


**Figure 4.44:** Comparison of DR & ACC for RF Algorithm



**Figure 4.45:** Comparison of Training and Testing Time for RF Algorithm





**Figure 4.46:** Comparison of Memory Consume RF Algorithm

**Scenario 4:** Testing the effect of imbalanced data on classification.

**Scenario 4 Implementation:** In order to see the effect of data imbalance, K-fold cross validation is applied to algorithm and every class accuracy are calculated in every experiment.

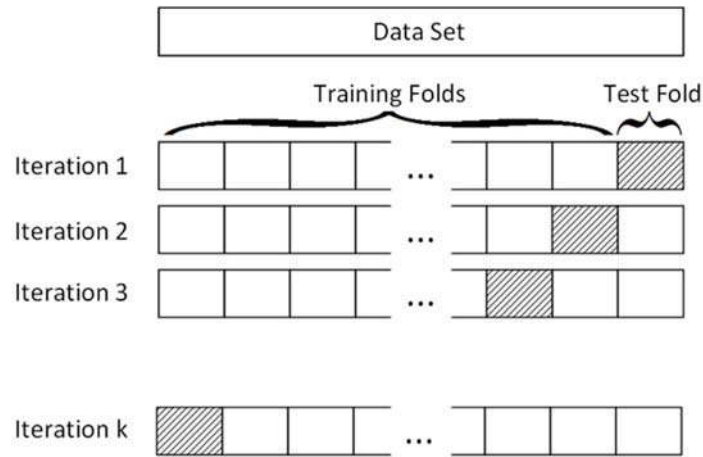
K-fold cross validation is a statistical method, in order to reduce variability and to avoid bias that involves partitioning the dataset into subsets, training the dataset on a subset and use the other subset to evaluate the model's performance (Kohavi 1995).

Steps of K-fold cross validation is:

1. Divide the dataset into k equal parts.
2. Use 1 part for testing and k-1 parts for training.
3. Repeat the procedure k times, rotating the test dataset.
4. Determine an expected performance metric based on the results across the iterations.

the iterations.

Representation of K-fold cross validation is shown in Figure 4.47.



**Figure 4.47: K-Fold Cross Validation**

## 4.5 Results

Detection rate and accuracy rate, classification speed and memory allocation are discussed to clarify results.

### 4.5.1 Detection Rate and Accuracy Rate Results

The KNN algorithm has accuracy performance of 99.9603 %. This also shows that KNN has better accuracy performance compared to other classification algorithms. The aim of our research is to detect many attacks and classify them while maximizing the generation of DR and minimizing test time. Experiments show that NADS which use KNN algorithm is able to detect most of the attacks for the KDD CUP99 data set at a high DR rate of 98.0379 %

### 4.5.2 Classification Speed Results

KNN uses 0.661590s for training and 1.139292s for testing phases. In terms of training time, KNN has better speed performance compared to other classification algorithms. In terms of testing time, KNN has not much difference than others algorithm because testing time takes very small value for each algorithms.

### 4.5.3 Memory Allocation Results

Results shows that memory allocation for KNN is very small. The KNN algorithm uses 0.304344 GB. Using less memory will cause the system to perform more efficiently because it is dealing with less data.

#### 4.5.4 Overall Discussion

The number of examples is an influential factor on the percentage of the classification accuracy and the training and testing time. The proposed NADS using the NB algorithm outperforms all algorithms in terms of training time since it has the lowest time. NADS using KM algorithm has lowest testing according to experiments.

With respect to DR 98.0379% performance, the proposed NADS using the KNN algorithm outperforms all other algorithms. But it has the second lowest speed in the comparison list.

With respect to Accuracy 99.9603% performance, KNN Algorithm outperforms all other algorithms. KNN achieves better performance for the anomaly detection at second highest speed.

In Figure 4.48, Figure 4.49 and Figure 4.50 first experiment of first scenario for KM, NB and RF are shown. Confusion matrix of RF algorithm is the one which is most similar with confusion matrix of KNN algorithm.

```
K-Means Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack 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 : 19.089063
Cross validation scores :
[16.93520674 17.27679116 16.92365334 16.92099478 16.76692781]
Mean score of 5-fold cross validation : 0.016965
Time taken to train final model : 3.872822
Predictions made using final model...
Time taken to make predictions on test data : 0.009377
Memory used : 0.300694 GB CPU usage : 14.100000
Confusion matrix :
[[9387  558   33 3205  266]
 [   8 1028   89  339  825]
 [ 195 1382 6899  701   57]
 [   68   10    1    2  128]
 [    5    0    0    1    5]]
Accuracy score on test data is : 0.687560
For normal, Detection Rate is % 69.79701093018068
For probe, Detection Rate is % 44.91044124071647
For dos, Detection Rate is % 74.71301711067792
For r2l, Detection Rate is % 0.9569377990430622
For u2r, Detection Rate is % 45.45454545454545
DR is % 47.166390507032716
```

**Figure 4.48:** Experiment of KM Algorithm

```

Naive Bayes Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack type mapping created...
Train and Test data labels created...
Decomposed features created...
Number of features used : 5
Time taken to perform 5-fold cross validation : 0.827913
Cross validation scores :
[0.88395444 0.88724401 0.88342462 0.88250705 0.88576192]
Mean score of 5-fold cross validation : 0.884578
Time taken to train final model : 0.128126
Predictions made using final model...
Time taken to make predictions on test data : 0.006212
Memory used : 0.505257 GB CPU usage : 9.900000
Confusion matrix :
[[12169 1229 50 1 0]
 [ 147 1835 307 0 0]
 [ 883 93 8258 0 0]
 [ 187 20 0 2 0]
 [ 11 0 0 0 0]]
Accuracy score on test data is : 0.883773
For normal, Detection Rate is % 90.48256375938732
For probe, Detection Rate is % 80.16601135867191
For dos, Detection Rate is % 89.43036603855317
For r2l, Detection Rate is % 0.9569377990430622
For u2r, Detection Rate is % 0.0
DR is % 52.2071757911311

```

**Figure 4.49:** Experiment of NB Algorithm

```

Random Forest Classifier
Train and Test Data read...
There are 5 attack type
Attacks name are:
['dos', 'u2r', 'r2l', 'probe', 'normal']
Attack 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 : 32.351665
Cross validation scores :
[0.99575346 0.99662645 0.99626895 0.99638789 0.99738022]
Mean score of 5-fold cross validation : 0.996483
Time taken to train final model : 8.095981
Predictions made using final model...
Time taken to make predictions on test data : 0.028038
Memory used : 0.538746 GB CPU usage : 13.500000
Confusion matrix :
[[13443 2 1 3 0]
 [ 4 2285 0 0 0]
 [ 2 0 9232 0 0]
 [ 3 0 1 204 1]
 [ 4 0 0 0 7]]
Accuracy score on test data is : 0.999166
For normal, Detection Rate is % 99.95538701762213
For probe, Detection Rate is % 99.82525120139799
For dos, Detection Rate is % 99.97834091401343
For r2l, Detection Rate is % 97.60765550239235
For u2r, Detection Rate is % 63.63636363636363
DR is % 92.2005996543579

```

**Figure 4.50:** Experiment of RF Algorithm

## **CHAPTER 5**

### **CONCLUSION**

The main aim of this thesis is to propose a Network Anomaly Detection System using machine learning, which helps to detect anomalies and respond with appropriate actions. The purpose of the anomaly detection system is to reveal detectable and undetected anomalies. The proposed NADS system uses the KNN Algorithm for classification, PCA Algorithm for reduction and it classifies connections as normal or abnormal.

Results show that the proposed NADS achieved highest classification ACC of 99.9603 % by using the KNN and PCA algorithms and second highest speed after NB which has the highest speed. KNN performance is very high in terms of training time since it has the second lowest time. But it was not possible to verify how the system will behave on larger networks and using bigger dataset and it was not possible to simulate and recreate all possible intrusions and attacks.

Many studies on NADS using machine learning have been recently carried out and new technologies introduced. As a future study, this study can be expanded in different ways such as using other packet sniffing technologies in NADS. In addition, different data sets can be used to test the proposed NADS system.

## REFERENCES

Ahmed, M. and A. N. J. A. o. D. S. Mahmood (2015). "Novel approach for network traffic pattern analysis using clustering-based collective anomaly detection." 2(1): 111-130.

Aljawarneh, S., et al. (2018). "Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model." 25: 152-160.

Baliga, A., et al. (2007). Lurking in the shadows: Identifying systemic threats to kernel data. 2007 IEEE Symposium on Security and Privacy (SP'07), IEEE.

Bhuyan, M. H., et al. (2013). "Network anomaly detection: methods, systems and tools." 16(1): 303-336.

Dasgupta, D., et al. (2003). Artificial immune system (AIS) research in the last five years. The 2003 Congress on Evolutionary Computation, 2003. CEC'03., IEEE.

Denning, D. E. J. I. T. o. s. e. (1987). "An intrusion-detection model." (2): 222-232.

Duda, R. O., et al. (2001). "Pattern classification." 1: 335-339.

Farid, D. M., et al. (2010). "Combining naive bayes and decision tree for adaptive intrusion detection."

Gaber, M. M. J. W. I. R. D. M. and K. Discovery (2012). "Advances in data stream mining." 2(1): 79-85.

Gable, K. A. J. V. J. T. I. L. (2010). "Cyber-Apocalypse Now: Securing the Internet Against Cyberterrorism and Using Universal Jurisdiction as a Deterrent." 43: 57.

Garcia-Teodoro, P., et al. (2009). "Anomaly-based network intrusion detection: Techniques, systems and challenges." 28(1-2): 18-28.

Hamamoto, A. H., et al. (2018). "Network anomaly detection system using genetic algorithm and fuzzy logic." 92: 390-402.

Han, J., et al. (2011). Data mining: concepts and techniques, Elsevier.

Helman, P., et al. (1992). Foundations of intrusion detection (computer security). [1992] Proceedings The Computer Security Foundations Workshop V, IEEE.

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

Kathareios, G., et al. (2017). Catch it if you can: Real-time network anomaly detection with low false alarm rates. 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. Ijcai, Montreal, Canada.

Lazarevic, A., et al. (2003). A comparative study of anomaly detection schemes in network intrusion detection. Proceedings of the 2003 SIAM International Conference on Data Mining, SIAM.

Liao, Y., et al. (2007). "Adaptive anomaly detection with evolving connectionist systems." 30(1): 60-80.

Lippi, M., et al. (2013). "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning." 14(2): 871-882.

Mahoney, M. V. and P. K. Chan (2002). Learning nonstationary models of normal network traffic for detecting novel attacks. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM.

Omar, S., et al. (2013). "Machine learning techniques for anomaly detection: an overview." 79(2).

Parmar, J. D. and J. T. J. I. J. Patel (2017). "Anomaly Detection in Data Mining: A Review." 7(4).

Patcha, A. and J.-M. J. C. n. Park (2007). "An overview of anomaly detection techniques: Existing solutions and latest technological trends." 51(12): 3448-3470.

Rao, K. H., et al. (2011). "Implementation of anomaly detection technique using machine learning algorithms." 2(3): 25-31.

Shanmugavadivu, R., et al. (2011). "Network intrusion detection system using fuzzy logic." 2(1): 101-111.

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

Tavallaee, M., et al. (2009). A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, IEEE.

Williams, N., et al. (2006). "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification." 36(5): 5-16.



## APPENDICES

### Appendix: A

#### Machine Learning and Machine Learning Python Libraries

Machine learning is the data analytical method, basically based on algorithms, math and statistics, that applies the ability of learning to machines through data which gathered from the natural experience of humans and animals. We have used some Python Libraries in building our NADS.

##### 1. Numpy (Numerical Python)

Numpy that stands for Numerical Python contains some arrays that operates rapid mathematical operations. Random numbers can be generated with the Numpy Library. Many of mathematical operations can be performed from matrix multiplication to linear algebra operations and Fourier transforms.

##### 2. Pandas

Pandas is an easy-to-use, high-performance data configuration and data analysis library. With this library data can be read and written from many different sources such as excel, json, text (csv) and database. It contains table structures which are one-dimensional as named Serie, 2-dimensional named as DataFrame. Pandas tables can keep many different type of variables (digital, categorical, date etc.). Important data processing steps, such as data conversion, filtering, can be easily performed with this library.

##### 3. Scikit-Learn

This widely used library has many machine learning algorithms. In addition to these algorithms, this library also includes dimensional reduction, data processing and model selection methods.

## Appendix: B

### 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 |
| 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 | Error_rate                  | No. of connections that have "SYN" errors                                                | Continuous |
| 26 | Srv_Error_rate              | No. of connections that have "SYN" errors                                                | Continuous |
| 27 | Error_rate                  | No. of connections that have "REJ" errors                                                | Continuous |
| 28 | Srv_error_rate              | No. of connections that have "REJ" errors                                                | Continuous |
| 29 | Same_error_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 |
| 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_error_rate         | No. of connections to the current host that have an S0 error                             | Continuous |
| 39 | Dst_host_Srv_error_rate     | No. of connections to the current host and specified service that have an S0 error       | Continuous |
| 40 | Dst_host_error_rate         | No. of connections to the current host that have an RST error                            | Continuous |
| 41 | Dst_host_error_rate         | No. of connections to the current host and specified service that have an RST error      | Continuous |

## Appendix: C

‘Fields Name File’ shows types of attacks. ‘Attack Types File’ maps its columns to the attack column of main file.

### Fields Name File

|    | A                  | B          | C |
|----|--------------------|------------|---|
| 1  | duration           | continuous |   |
| 2  | protocol_type      | symbolic   |   |
| 3  | service            | symbolic   |   |
| 4  | flag               | symbolic   |   |
| 5  | src_bytes          | continuous |   |
| 6  | dst_bytes          | continuous |   |
| 7  | land               | continuous |   |
| 8  | wrong_fragment     | continuous |   |
| 9  | urgent             | continuous |   |
| 10 | hot                | continuous |   |
| 11 | num_failed_logins  | continuous |   |
| 12 | logged_in          | continuous |   |
| 13 | num_compromised    | continuous |   |
| 14 | root_shell         | continuous |   |
| 15 | su_attempted       | continuous |   |
| 16 | num_root           | continuous |   |
| 17 | num_file_creations | continuous |   |

### Attack Types File

|    | A               | B     | C |
|----|-----------------|-------|---|
| 1  | back            | dos   |   |
| 2  | buffer_overflow | u2r   |   |
| 3  | ftp_write       | r2l   |   |
| 4  | guess_passwd    | r2l   |   |
| 5  | imap            | r2l   |   |
| 6  | ipsweep         | probe |   |
| 7  | land            | dos   |   |
| 8  | loadmodule      | u2r   |   |
| 9  | multihop        | r2l   |   |
| 10 | neptune         | dos   |   |
| 11 | nmap            | probe |   |
| 12 | perl            | u2r   |   |
| 13 | phf             | r2l   |   |
| 14 | pod             | dos   |   |
| 15 | portsweep       | probe |   |
| 16 | rootkit         | u2r   |   |
| 17 | satan           | probe |   |
| 18 | smurf           | dos   |   |

## Appendix: D

### Python Code for K-Nearest Neighbor Classification Algorithm

```
1 import pandas as pd
2 import numpy as np
3 import time
4 from sklearn.metrics import accuracy_score
5 from sklearn.metrics import confusion_matrix
6 from sklearn.preprocessing import LabelEncoder
7 from sklearn.preprocessing import MinMaxScaler
8 from sklearn.decomposition import PCA
9 from sklearn.neighbors import KNeighborsClassifier
10 from sklearn.model_selection import cross_val_score
11 import psutil
12 import os
13
14
15 print("K-Nearest Neighbors Classifier")
16 # Reads the train and test data
17
18 #train = pd.read_csv('1Million.csv', header=None)
19 #train = pd.read_csv('500K.csv', header=None)
20 train = pd.read_csv('./KDDTrain+.csv', header=None)
21 test = pd.read_csv('./KDDTest+.csv', header=None)
22 print("Train and Test Data read...")
23
24
25
26 # Reads "Field Names.csv". Use this to set the names of train and test data columns
27 columns = pd.read_csv('Field Names.csv', header=None)
28 columns.columns = ['name', 'type']
29 train.columns = columns['name']
30 test.columns = columns['name']
31
32
33
34 # Read Attack Types.csv
35 # Use this to create a mapping from attack types to final labels (Normal, Dos, R2L, Prob, U2R)
36 attackType = pd.read_csv('Attack Types.csv', header=None)
37 attackType.columns = ['Name', 'Type']
38 attackMap = {}
39
40
41
42 #find count of type from attackType
43 attackTypeCount = len(attackType['Type'].drop_duplicates())
44 attackNames = attackType['Type'].drop_duplicates().values.tolist()
45 print('There are ' + str(attackTypeCount) + ' attack type')
46 print('Attacks name are:')
47 print(attackNames)
```

```

51 # Creates attackMap map which contains a mapping between attack type and the final label
52 for i in range(len(attackType)):
53     attackMap[attackType['Name'][i]] = attackType['Type'][i]
54 print("Attack type mapping created...")
55
56
57
58 # Add a new variable called 'label' which contains the final label
59 train['label'] = train['attack_type'].map(attackMap)
60 test['label'] = test['attack_type'].map(attackMap)
61
62
63 # The variable 'label' is stored in different variables
64 # This is required to keep the dependent variable separate from the independent variable
65 trainLabel = train['label']
66 testLabel = test['label']
67
68
69
70 # attack_type and label variables are removed from the train and test data
71 # so only features are remained in train and test data
72 train.drop(['attack_type', 'label'], axis=1, inplace=True)
73 test.drop(['attack_type', 'label'], axis=1, inplace=True)
74 print("Train and Test data labels created...")
75
76
77
78 # Transform the existing nominal variables into the integer coded variables using the LabelEncoder
79 for col in ['protocol_type', 'flag', 'service']:
80     le = LabelEncoder()
81     le1 = LabelEncoder()
82     le.fit(train[col])
83     le1.fit(test[col])
84     train[col] = le.transform(train[col])
85     test[col] = le1.transform(test[col])
86
87
88
89 # for Stadart Scaler
90 scaler = MinMaxScaler() #scale between 0 and 1
91 train = scaler.fit_transform(train)
92 test = scaler.fit_transform(test)
93
94 total = np.concatenate([train, test] )
95
96
97
98 # Decomposition features are generated for both train and test data
99 pca = PCA(n_components=41, random_state=100)
100 pca.fit(total)
101 train = pca.transform(train)
102 test = pca.transform(test)

```

```

105 print("Decomposed features created...")
106 print("Number of features used : %d" % train.shape[1])
107
108
109
110 # Performing k-fold Cross validation
111 startTime = time.clock()
112 KNN = KNeighborsClassifier(n_neighbors = 1)
113 score = cross_val_score(KNN, train, trainLabel, cv=5)
114 endTime = time.clock()
115 print("Time taken to perform 5-fold cross validation : %f" % (endTime - startTime))
116 print("Cross validation scores : ")
117 print(score)
118 print("Mean score of 5-fold cross validation : %f" % score.mean())
119
120
121
122 # Final Testing and Evaluate Performance
123 # Train the KNN classifier model by original train data and got optimized parameter
124 startTime = time.clock()
125 KNN2 = KNeighborsClassifier(n_neighbors = 1)
126 KNN2.fit(train, trainLabel)
127 endTime = time.clock()
128 print("Time taken to train final model : %f" % (endTime - startTime))
129 print("Predictions made using final model...")
130
131
132
133 # Predictions for test data and evaluate its performance
134 startTime = time.clock()
135 pred = KNN2.predict(test)
136 endTime = time.clock()
137 cpuUsage = psutil.cpu_percent()
138 pid = os.getpid()
139 py = psutil.Process(pid)
140 memoryUse = py.memory_info()[0] / 2. ** 30
141 print("Time taken to make predictions on test data : %f" % (endTime - startTime))
142 print("Memory used : %f GB CPU usage : %f" % (memoryUse, cpuUsage))
143
144
145 #calculate scores and confusion_matrix
146 Classes = ['normal', 'probe', 'dos', 'r2l', 'u2r' ]
147 acc = accuracy_score(y_pred=pred, y_true=testLabel)
148 con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels=Classes)
149 print("Confusion matrix : ")
150 print(con_matrix)
151
152
153 # Print accuracy and detection rate
154 acc = accuracy_score(y_pred=pred, y_true=testLabel)
155 print("Accuracy score on test data is : %f" % acc)

```

```

157     sumDr = 0
158     for i in range(con_matrix.shape[0]):
159         det_rate = 0
160         for j in range(con_matrix.shape[1]):
161             if i != j :
162                 det_rate += con_matrix[i][j]
163             if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i]) != 0:
164                 det_rate = 100 * con_matrix[i][i] / (det_rate + con_matrix[i][i])
165                 sumDr += det_rate
166             print("For " + Classes[i] + ", Detection Rate is % " + str(det_rate))
167         else:
168             print("For " + Classes[i] + ", Detection Rate is % 0")
169
170     DR = sumDr/attackTypeCount
171     print("DR is % " + str(DR))

```





## Appendix: E

### Python Code for K-Means Classification Algorithm

```
1 import pandas as pd
2 import numpy as np
3 import time
4 from sklearn.metrics import accuracy_score
5 from sklearn.metrics import confusion_matrix
6 from sklearn.preprocessing import LabelEncoder
7 from sklearn.preprocessing import MinMaxScaler
8 from sklearn.decomposition import PCA
9 from sklearn.cluster import KMeans
10 from sklearn.model_selection import cross_val_score
11 import psutil
12 import os
13
14
15
16 print("K-Means Classifier")
17 # Reads the train and test data
18
19 #train = pd.read_csv('1Million.csv', header=None)
20 train = pd.read_csv('500K.csv', header=None)
21 #train = pd.read_csv('./KDDTrain+.csv', header=None)
22 test = pd.read_csv('./KDDTest+.csv', header=None)
23 print("Train and Test Data read...")
24
25
26
27 # Reads "Field Names.csv". Use this to set the names of train and test data columns
28 columns = pd.read_csv('Field Names.csv', header=None)
29 columns.columns = ['name', 'type']
30 train.columns = columns['name']
31 test.columns = columns['name']
32
33
34
35 # Read Attack Types.csv
36 # Use this to create a mapping from attack types to final labels (Normal, Dos, R2L, Prob, U2R)
37 attackType = pd.read_csv('Attack Types.csv', header=None)
38 attackType.columns = ['Name', 'Type']
39 attackMap = {}
40
41
42
43 #find count of type from attackType
44 attackTypeCount = len(attackType['Type'].drop_duplicates())
45 attackNames = attackType['Type'].drop_duplicates().values.tolist()
46 print('There are ' + str(attackTypeCount) + ' attack type')
47 print('Attacks name are:')
48 print(attackNames)
```

```

52 # Creates attackMap map which contains a mapping between attack type and the final label
53 for i in range(len(attackType)):
54     attackMap[attackType['Name'][i]] = attackType['Type'][i]
55 print("Attack type mapping created...")
56
57
58
59 # Add a new variable called 'label' which contains the final label
60 train['label'] = train['attack_type'].map(attackMap)
61 test['label'] = test['attack_type'].map(attackMap)
62
63
64
65
66 # Transform the existing nominal variables into the integer coded variables using the LabelEncoder
67 for col in ['protocol_type', 'flag', 'service', 'label']:
68     le = LabelEncoder()
69     le1 = LabelEncoder()
70     le.fit(train[col])
71     le1.fit(test[col])
72     train[col] = le.transform(train[col])
73     test[col] = le1.transform(test[col])
74
75
76 # The variable 'label' is stored in different variables
77 # This is required to keep the dependent variable separate from the independent variable
78 trainLabel = train['label']
79 testLabel = test['label']
80
81
82
83 # attack_type and label variables are removed from the train and test data
84 # so only features are remained in train and test data
85 train.drop(['attack_type', 'label'], axis=1, inplace=True)
86 test.drop(['attack_type', 'label'], axis=1, inplace=True)
87 print("Train and Test data labels created...")
88
89
90 # for Stadart Scaler
91 scaler = MinMaxScaler() #scale between 0 and 1
92 train = scaler.fit_transform(train)
93 test = scaler.fit_transform(test)
94 total = np.concatenate([train, test] )
95
96
97
98 # Decomposition features are generated for both train and test data
99 pca = PCA(n_components=41, random_state=100)
100 pca.fit(total)
101 train = pca.transform(train)
102 test = pca.transform(test)

```

```

105 print("Decomposed features created...")
106 print("Number of features used : %d" % train.shape[1])
107
108
109
110 # Performing k-fold Cross validation
111 startTime = time.clock()
112 KM1 = KMeans(n_clusters = 5, init = 'random', random_state=100)
113 KM1.fit(train, trainLabel)
114 score = abs(cross_val_score(KM1, train, trainLabel, cv=5))/1000
115 endTime = time.clock()
116 print("Time taken to perform 5-fold cross validation : %f" % (endTime - startTime))
117 print("Cross validation scores : ")
118 print(score)
119 print("Mean score of 5-fold cross validation : %f" % abs(score.mean()/1000))
120
121
122
123 # Final Testing and Evaluate Performance
124 # Train the KNN classifier model by original train data and got optimized parameter
125 startTime = time.clock()
126 KM2 = KMeans(n_clusters = 5, init = 'k-means++')
127 KM2.fit(train, trainLabel)
128 endTime = time.clock()
129 print("Time taken to train final model : %f" % (endTime - startTime))
130 print("Predictions made using final model...")
131
132
133
134 # Predictions for test data and evaluate its performance
135 startTime = time.clock()
136 pred = KM2.predict(test)
137 endTime = time.clock()
138 cpuUsage = psutil.cpu_percent()
139 pid = os.getpid()
140 py = psutil.Process(pid)
141 memoryUse = py.memory_info()[0] / 2. ** 30
142 print("Time taken to make predictions on test data : %f" % (endTime - startTime))
143 print("Memory used : %f GB CPU usage : %f" % (memoryUse, cpuUsage))
144
145
146 #calculate scores and confusion_matrix
147 Classes = ['normal', 'probe', 'dos', 'r2l', 'u2r']
148 acc = accuracy_score(y_pred=pred, y_true=testLabel)
149 con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels= list(1e.transform(Classes))) #, labels=Classes
150 print("Confusion matrix : ")
151 print(con_matrix)

```

```

158     sumDr = 0
159     for i in range(con_matrix.shape[0]):
160         det_rate = 0
161         for j in range(con_matrix.shape[1]):
162             if i != j :
163                 det_rate += con_matrix[i][j]
164             if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i]) != 0:
165                 det_rate = 100 * con_matrix[i][i] / (det_rate + con_matrix[i][i])
166                 sumDr += det_rate
167                 print("For " + Classes[i] + ", Detection Rate is % " + str(det_rate))
168             else:
169                 print("For " + Classes[i] + ", Detection Rate is % 0")
170
171 DR = sumDr/attackTypeCount
172 print("DR is % " + str(DR))

```



## Appendix: F

### Python Code for Naïve Bayes Classification Algorithm

```
1  import pandas as pd
2  import numpy as np
3  import time
4  from sklearn.metrics import accuracy_score
5  from sklearn.metrics import confusion_matrix
6  from sklearn.preprocessing import LabelEncoder
7  from sklearn.preprocessing import MinMaxScaler
8  from sklearn.decomposition import PCA
9  from sklearn.naive_bayes import GaussianNB
10 from sklearn.model_selection import cross_val_score
11 import psutil
12 import os
13
14
15
16 print("Naive Bayes Classifier")
17 # Reads the train and test data
18
19 #train = pd.read_csv('1Million.csv', header=None)
20 #train = pd.read_csv('500K.csv', header=None)
21 train = pd.read_csv('./KDDTrain+.csv', header=None)
22 test = pd.read_csv('./KDDTest+.csv', header=None)
23 print("Train and Test Data read...")
24
25
26
27 # Reads "Field Names.csv". Use this to set the names of train and test data columns
28 columns = pd.read_csv('Field Names.csv', header=None)
29 columns.columns = ['name', 'type']
30 train.columns = columns['name']
31 test.columns = columns['name']
32
33
34
35 # Read Attack Types.csv
36 # Use this to create a mapping from attack types to final labels (Normal, Dos, R2L, Prob, U2R)
37 attackType = pd.read_csv('Attack Types.csv', header=None)
38 attackType.columns = ['Name', 'Type']
39 attackMap = {}
40
41
42
43 #find count of type from attackType
44 attackTypeCount = len(attackType['Type'].drop_duplicates())
45 attackNames = attackType['Type'].drop_duplicates().values.tolist()
46 print('There are ' + str(attackTypeCount) + ' attack type')
47 print('Attacks name are:')
48 print(attackNames)
```

```

52 # Creates attackMap map which contains a mapping between attack type and the final label
53 for i in range(len(attackType)):
54     attackMap[attackType['Name'][i]] = attackType['Type'][i]
55 print("Attack type mapping created...")
56
57
58
59 # Add a new variable called 'label' which contains the final label
60 train['label'] = train['attack_type'].map(attackMap)
61 test['label'] = test['attack_type'].map(attackMap)
62
63
64 # The variable 'label' is stored in different variables
65 # This is required to keep the dependent variable separate from the independent variable
66 trainLabel = train['label']
67 testLabel = test['label']
68
69
70
71 # attack_type and label variables are removed from the train and test data
72 # so only features are remained in train and test data
73 train.drop(['attack_type', 'label'], axis=1, inplace=True)
74 test.drop(['attack_type', 'label'], axis=1, inplace=True)
75 print("Train and Test data labels created...")
76
77
78
79 # Transform the existing nominal variables into the integer coded variables using the LabelEncoder
80 for col in ['protocol_type', 'flag', 'service']:
81     le = LabelEncoder()
82     le1 = LabelEncoder()
83     le.fit(train[col])
84     le1.fit(test[col])
85     train[col] = le.transform(train[col])
86     test[col] = le1.transform(test[col])
87
88
89
90 # for Stadart Scaler
91 scaler = MinMaxScaler() #scale between 0 and 1
92 train = scaler.fit_transform(train)
93 test = scaler.fit_transform(test)
94 total = np.concatenate([train, test] )
95
96
97
98 # Decomposition features are generated for both train and test data
99 pca = PCA(n_components=5, random_state=100)
100 pca.fit(total)
101 train = pca.transform(train)
102 test = pca.transform(test)

```



```

105 print("Decomposed features created...")
106 print("Number of features used : %d" % train.shape[1])
107
108
109
110 # Performing k-fold Cross validation
111 startTime = time.clock()
112 GNB1 = GaussianNB()
113 score = cross_val_score(GNB1, train, trainLabel, cv=5)
114 endTime = time.clock()
115 print("Time taken to perform 5-fold cross validation : %f" % (endTime - startTime))
116 print("Cross validation scores : ")
117 print(score)
118 print("Mean score of 5-fold cross validation : %f" % score.mean())
119
120
121
122 # Final Testing and Evaluate Performance
123 # Train the KNN classifier model by original train data and got optimized parameter
124 startTime = time.clock()
125 GNB2 = GaussianNB()
126 GNB2.fit(train, trainLabel)
127 endTime = time.clock()
128 print("Time taken to train final model : %f" % (endTime - startTime))
129 print("Predictions made using final model...")
130
131
132
133 # Predictions for test data and evaluate its performance
134 startTime = time.clock()
135 pred = GNB2.predict(test)
136 endTime = time.clock()
137 cpuUsage = psutil.cpu_percent()
138 pid = os.getpid()
139 py = psutil.Process(pid)
140 memoryUse = py.memory_info()[0] / 2. ** 30
141 print("Time taken to make predictions on test data : %f" % (endTime - startTime))
142 print("Memory used : %f GB CPU usage : %f" % (memoryUse, cpuUsage))
143
144
145 #calculate scores and confusion_matrix
146 Classes = ['normal', 'probe', 'dos', 'r2l', 'u2r']
147 acc = accuracy_score(y_pred=pred, y_true=testLabel)
148 con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels=Classes)
149 print("Confusion matrix : ")
150 print(con_matrix)
151
152
153 # Print accuracy and detection rate
154 acc = accuracy_score(y_pred=pred, y_true=testLabel)
155 print("Accuracy score on test data is : %f" % acc)

```

```

157 sumDr = 0
158 for i in range(con_matrix.shape[0]):
159     det_rate = 0
160     for j in range(con_matrix.shape[1]):
161         if i != j :
162             det_rate += con_matrix[i][j]
163     if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i]) != 0:
164         det_rate = 100 * con_matrix[i][i] / (det_rate + con_matrix[i][i])
165         sumDr += det_rate
166         print("For " + Classes[i] + ", Detection Rate is % " + str(det_rate))
167     else:
168         print("For " + Classes[i] + ", Detection Rate is % 0")
169
170 DR = sumDr/attackTypeCount
171 print("DR is % " + str(DR))

```





## Appendix: G

### Python Code for Random Forest Classification Algorithm

```
1  import pandas as pd
2  import numpy as np
3  import time
4  from sklearn.metrics import accuracy_score
5  from sklearn.metrics import confusion_matrix
6  from sklearn.preprocessing import LabelEncoder
7  from sklearn.preprocessing import MinMaxScaler
8  from sklearn.decomposition import PCA
9  from sklearn.ensemble import RandomForestClassifier
10 from sklearn.model_selection import cross_val_score
11 import psutil
12 import os
13
14
15
16 print("Random Forest Classifier")
17 # Reads the train and test data
18
19 #train = pd.read_csv('1Million.csv', header=None)
20 #train = pd.read_csv('500K.csv', header=None)
21 train = pd.read_csv('./KDDTrain+.csv', header=None)
22 test = pd.read_csv('./KDDTest+.csv', header=None)
23 print("Train and Test Data read...")
24
25
26
27 # Reads "Field Names.csv". Use this to set the names of train and test data columns
28 columns = pd.read_csv('Field Names.csv', header=None)
29 columns.columns = ['name', 'type']
30 train.columns = columns['name']
31 test.columns = columns['name']
32
33
34
35 # Read Attack Types.csv
36 # Use this to create a mapping from attack types to final labels (Normal, Dos, R2L, Prob, U2R)
37 attackType = pd.read_csv('Attack Types.csv', header=None)
38 attackType.columns = ['Name', 'Type']
39 attackMap = {}
40
41
42
43 #find count of type from attackType
44 attackTypeCount = len(attackType['Type'].drop_duplicates())
45 attackNames = attackType['Type'].drop_duplicates().values.tolist()
46 print('There are ' + str(attackTypeCount) + ' attack type')
47 print('Attacks name are:')
48 print(attackNames)
```

```

52 # Creates attackMap map which contains a mapping between attack type and the final label
53 for i in range(len(attackType)):
54     attackMap[attackType['Name'][i]] = attackType['Type'][i]
55     print("Attack type mapping created...")
56
57
58
59 # Add a new variable called 'label' which contains the final label
60 train['label'] = train['attack_type'].map(attackMap)
61 test['label'] = test['attack_type'].map(attackMap)
62
63
64 # The variable 'label' is stored in different variables
65 # This is required to keep the dependent variable separate from the independent variable
66 trainLabel = train['label']
67 testLabel = test['label']
68
69
70
71 # attack_type and label variables are removed from the train and test data
72 # so only features are remained in train and test data
73 train.drop(['attack_type', 'label'], axis=1, inplace=True)
74 test.drop(['attack_type', 'label'], axis=1, inplace=True)
75 print("Train and Test data labels created...")
76
77
78
79 # Transform the existing nominal variables into the integer coded variables using the LabelEncoder
80 for col in ['protocol_type', 'flag', 'service']:
81     le = LabelEncoder()
82     le1 = LabelEncoder()
83     le.fit(train[col])
84     le1.fit(test[col])
85     train[col] = le.transform(train[col])
86     test[col] = le1.transform(test[col])
87
88
89
90
91
92 # for Stadart Scaler
93 scaler = MinMaxScaler() #scale between 0 and 1
94 train = scaler.fit_transform(train)
95 test = scaler.fit_transform(test)
96 total = np.concatenate([train, test] )
97
98
99
100 # Decomposition features are generated for both train and test data
101 pca = PCA(n_components=41, random_state=100)
102 pca.fit(total)
103 train = pca.transform(train)
104 test = pca.transform(test)

```

```

107 print("Decomposed features created...")
108 print("Number of features used : %d" % train.shape[1])
109
110
111
112 # Performing k-fold Cross validation
113 startTime = time.clock()
114 RFC1 = RandomForestClassifier(n_estimators = 10, criterion = 'entropy' )
115 score = cross_val_score(RFC1, train, trainLabel, cv=5)
116 endTime = time.clock()
117 print("Time taken to perform 5-fold cross validation : %f" % (endTime - startTime))
118 print("Cross validation scores : ")
119 print(score)
120 print("Mean score of 5-fold cross validation : %f" % score.mean())
121
122
123
124 # Final Testing and Evaluate Performance
125 # Train the KNN classifier model by original train data and got optimized parameter
126 startTime = time.clock()
127 RFC2 = RandomForestClassifier(n_estimators = 10, criterion = 'entropy' )
128 RFC2.fit(train, trainLabel)
129 endTime = time.clock()
130 print("Time taken to train final model : %f" % (endTime - startTime))
131 print("Predictions made using final model...")
132
133
134
135 # Predictions for test data and evaluate its performance
136 startTime = time.clock()
137 pred = RFC2.predict(test)
138 endTime = time.clock()
139 cpuUsage = psutil.cpu_percent()
140 pid = os.getpid()
141 py = psutil.Process(pid)
142 memoryUse = py.memory_info()[0] / 2. ** 30
143 print("Time taken to make predictions on test data : %f" % (endTime - startTime))
144 print("Memory used : %f GB CPU usage : %f" % (memoryUse, cpuUsage))
145
146
147 #calculate scores and confusion_matrix
148 Classes = ['normal', 'probe', 'dos', 'r2l', 'u2r' ]
149 acc = accuracy_score(y_pred=pred, y_true=testLabel)
150 con_matrix = confusion_matrix(y_pred=pred, y_true=testLabel, labels=Classes)
151 print("Confusion matrix : ")
152 print(con_matrix)
153
154
155 # Print accuracy and detection rate
156 acc = accuracy_score(y_pred=pred, y_true=testLabel)
157 print("Accuracy score on test data is : %f" % acc)

```

```

159     sumDr = 0
160     for i in range(con_matrix.shape[0]):
161         det_rate = 0
162         for j in range(con_matrix.shape[1]):
163             if i != j :
164                 det_rate += con_matrix[i][j]
165             if con_matrix[i][i] != 0 or (det_rate + con_matrix[i][i]) != 0:
166                 det_rate = 100 * con_matrix[i][i] / (det_rate + con_matrix[i][i])
167                 sumDr += det_rate
168                 print("For " + Classes[i] + ", Detection Rate is % " + str(det_rate))
169             else:
170                 print("For " + Classes[i] + ", Detection Rate is % 0")
171
172     DR = sumDr/attackTypeCount
173     print("DR is % " + str(DR))

```

