*A project report on*

*Information Security Management*

# IOT NETWORK INTRUSION SYSTEM

*Submitted in partial fulfillment for the award of the degree of*

## BTECH COMPUTER SCIENCE WITH AI & ML

*By*


AVANTHIKA RAJESH – 20BAI1144

JAMAL MOHOMMAD – 20BAI1166

KHUSHI V – 20BAI1285

SCOPE


NOVEMBER, 2022

## Abstract

One of the goals of smart environments is to improve people's quality of life in terms of comfort and efficiency. The Internet of Things (IoT) paradigm has recently evolved into a technology for building intelligent environments.

Everything from our mobile devices to our cars, cameras and appliances. We are moving toward a fully automated world. We are creating “smart homes,” wearables that track our every move, and devices that will keep us inter-connected with everyone in the world.

Having a smart home is a signal that you have made it. A smart home has everything connected to the Internet, allowing for better energy efficiency, greater security, and happier pets. Did you ask Siri or Alexa a question today that used their voice assistance connected to the IoT? Whether it is a Nest thermometer, Ring home security system or a Kenmore connected refrigerator, our home can know what we need and when we need it before we even realize it.

The Internet of Things helps to improve our energy efficiency and helps us live a more connected life. Security and privacy are considered key issues in real-world intelligent environments based on IoT models. Vulnerabilities in IoT-based systems create security threats affecting smart environment applications. Therefore, an intrusion detection system (IDS) designed for IoT environments is highly needed to

mitigate IoT-related security attacks that exploit some of these vulnerabilities.

## Scope

The Internet of Things (loT) is a network of connected physical objects embedded with electronics, sensors, and software. These connected objects (also called nodes) exchange data over the internet. There will be 22 billion connected devices in the world by 2025.

loT technology can be used in sectors like Manufacturing, Agriculture, Healthcare, Transportation, Media/Advertising, Retail, Water and Waste Management, Power Distribution, etc. loT is used in various applications like smart home, smart parking, smart water level monitoring, smart healthcare, smart traffic lighting, smart waste management, smart solar panels, etc.

It is essential for achieving high productivity, efficiency, a safe working environment, and low carbon emissions in the industry and other contexts. Healthcare and manufacturing are the sectors where several startups are working extensively on smart solutions.

For loT implementation, data analytics, cloud computing and network security are the key enabling technologies. Several companies offer cloud services in India, reducing the cost of smart solutions for smart cities. therefore, data security is a very challenging task in loT.

Because the scope of IoT is so much, the devices will be at a higher risk for intrusion. Hence, our project will be useful to find the intrusions and vulnerabilities in smart systems.

## Literature Review

1) **Intrusion detection systems for IoT-based smart environments: a survey:**

One of the goals of smart environments is to improve the quality of human life in terms of comfort and efficiency. The Internet of Things (IoT) paradigm has recently evolved into a technology for building smart environments. Security and privacy are considered key issues in any real-world smart environment based on the IoT model. The security vulnerabilities in IoT-based systems create security threats that affect smart environment applications. Thus, there is a crucial need for intrusion detection systems (IDSs) designed for IoT environments to mitigate IoT-related security attacks that exploit some of these security vulnerabilities. Due to the limited computing and storage capabilities of IoT devices and the specific protocols used, conventional IDSs may not be an option for IoT environments. This article presents a comprehensive survey of the latest IDSs designed for the IoT model, with a focus on the corresponding methods, features, and mechanisms. This article also provides deep insight into the IoT architecture, emerging security vulnerabilities, and their relation to the layers of the IoT architecture. This work demonstrates that despite previous studies regarding the design and implementation of IDSs for the IoT paradigm, developing efficient, reliable and robust IDSs for IoT-based smart environments is still a crucial task. Key considerations for the development of such IDSs are introduced as a future outlook at the end of this survey.

2) **A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges:**

The Internet of Things (IoT) has been rapidly evolving towards making a greater impact on everyday life to large industrial systems. Unfortunately, this has attracted the attention of cybercriminals who made IoT a target of malicious activities, opening the door to a possible attack on the end nodes. To this end, Numerous IoT intrusion detection Systems (IDS) have been proposed in the literature to tackle attacks on the IoT ecosystem, which can be broadly classified based on detection technique, validation strategy, and deployment strategy. This survey paper presents a comprehensive review of contemporary IoT IDS and an overview of techniques, deployment Strategy, validation strategy and datasets that are commonly applied for building IDS. We also review how existing IoT IDS detect intrusive attacks and secure communications on the IoT. It also presents the classification of IoT attacks and discusses future research challenges to counter such IoT attacks to make IoT more secure. These purposes help IoT security researchers by uniting, contrasting, and compiling scattered research efforts. Consequently, we provide a unique IoT IDS taxonomy, which sheds light on IoT IDS techniques, their advantages and disadvantages, IoT attacks that exploit IoT communication systems, corresponding advanced IDS and detection capabilities to detect IoT attacks.

### 3) Network Intrusion Detection for IoT Security Based on Learning Techniques:

Pervasive growth of Internet of Things (IoT) is visible across the globe. The 2016 Dyn cyberattack exposed the critical fault-lines among smart networks. Security of Internet of Things (IoT) has become a critical concern. The danger exposed by infested Internet-connected things not only affects the security of IoT, but also threatens the complete Internet eco-system which can possibly exploit the vulnerable Things (smart devices) deployed as botnets. Mirai malware compromised the video surveillance devices and paralyzed Internet via distributed denial of service (DDoS) attacks. In the recent past, security attack vectors have evolved bothways, in terms of complexity and diversity. Hence, to identify and prevent or detect novel attacks, it is important to analyze techniques in IoT context. This survey classifies the IoT security threats and challenges for IoT networks by evaluating existing defense techniques. Our main focus is on Network Intrusion Detection Systems (NIDS); hence, this paper reviews existing NIDS implementation tools and datasets as well as free & open-source network sniffing software. Then, it

surveys, analyzes and compares state-of-the-art NIDS proposals in the IoT context in terms of architecture, detection methodologies, validation strategies, treated threats and algorithm deployments. The review deals with both traditional and machine learning (ML) NIDS techniques and discusses future directions. This paper focuses on IoT NIDS deployed via Machine Learning since learning algorithms have a good success rate in security and privacy.

4) **Intrusion detection systems for iot: opportunities and challenges offered by edge computing and machine learning:**

Key components of current cybersecurity methods are the Intrusion Detection Systems (IDSs) were different techniques and architectures are applied to detect intrusions. IDSs can be based either on cross-checking monitored events with a database of known intrusion experiences, known as signature-based, or on learning the normal behavior of the system and reporting whether some anomalous events occur, named anomaly-based. This work is dedicated to survey the application of IDS to the Internet of Things (IoT) networks, where also the edge computing is used to support the IDS implementation. New challenges that arise when deploying an IDS in an edge scenario are identified and remedies are proposed. The paper focuses on anomaly-based IDSs, showing the main techniques that can be leveraged to detect anomalies and we present machine learning techniques and their application in the context of an IDS, describing the expected advantages and disadvantages that a specific technique could cause.

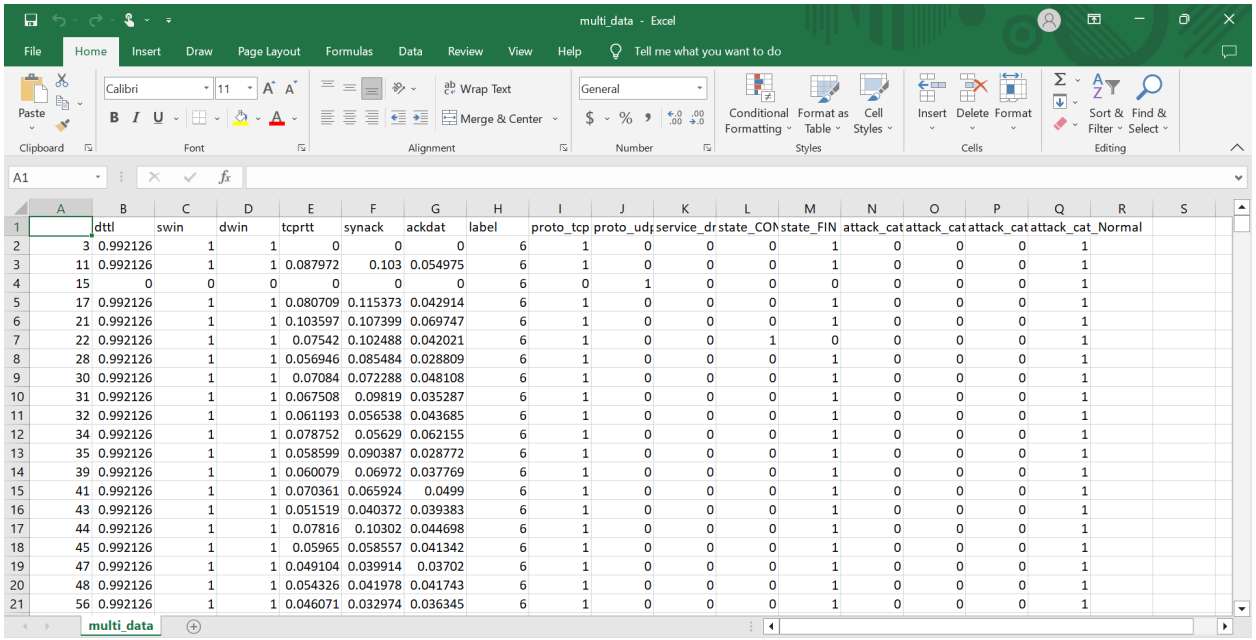
5) **Intrusion Detection in Internet of Things Systems: A Review on Design Approaches Leveraging Multi-Access Edge Computing, Machine Learning, and Datasets:**

The explosive growth of the Internet of Things (IoT) applications has imposed a dramatic increase of network data and placed a high computation complexity across various connected devices. The IoT devices capture valuable information, which allows the industries or individual users to make critical live dependent decisions. Most of these IoT devices have resource constraints such as low CPU, limited memory, and low energy storage. Hence, these devices are vulnerable to cyber-attacks due to the lack of capacity to run existing general-purpose security software. It creates an inherent risk in IoT networks. The multi-access edge computing (MEC) platform has emerged to

mitigate these constraints by relocating complex computing tasks from the IoT devices to the edge. Most of the existing related works are focusing on finding the optimized security solutions to protect the IoT devices. This paper implies that distributed solutions leveraging MEC should draw more attention. This paper presents a comprehensive review of state-of-the-art network intrusion detection systems (NIDS) and security practices for IoT networks. The approaches have been analyzed based on MEC platforms and utilizing machine learning (ML) techniques.

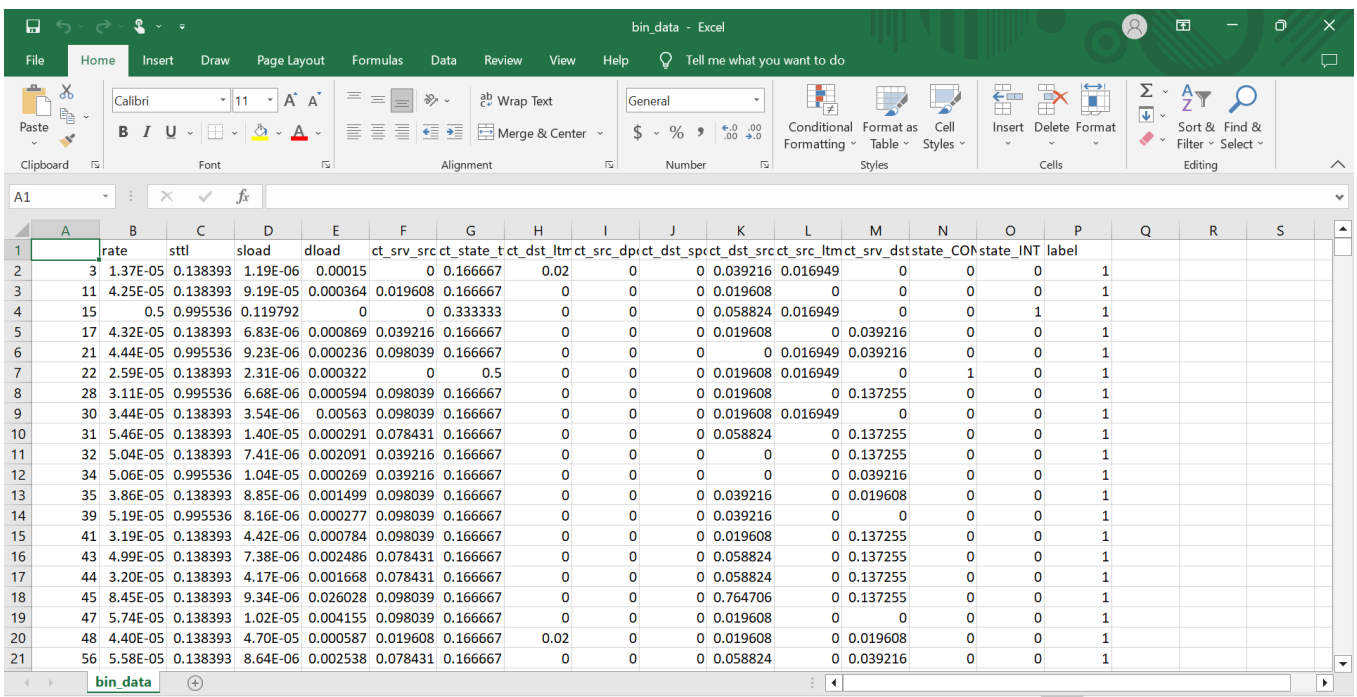
## Dataset

### Multiclass Dataset:



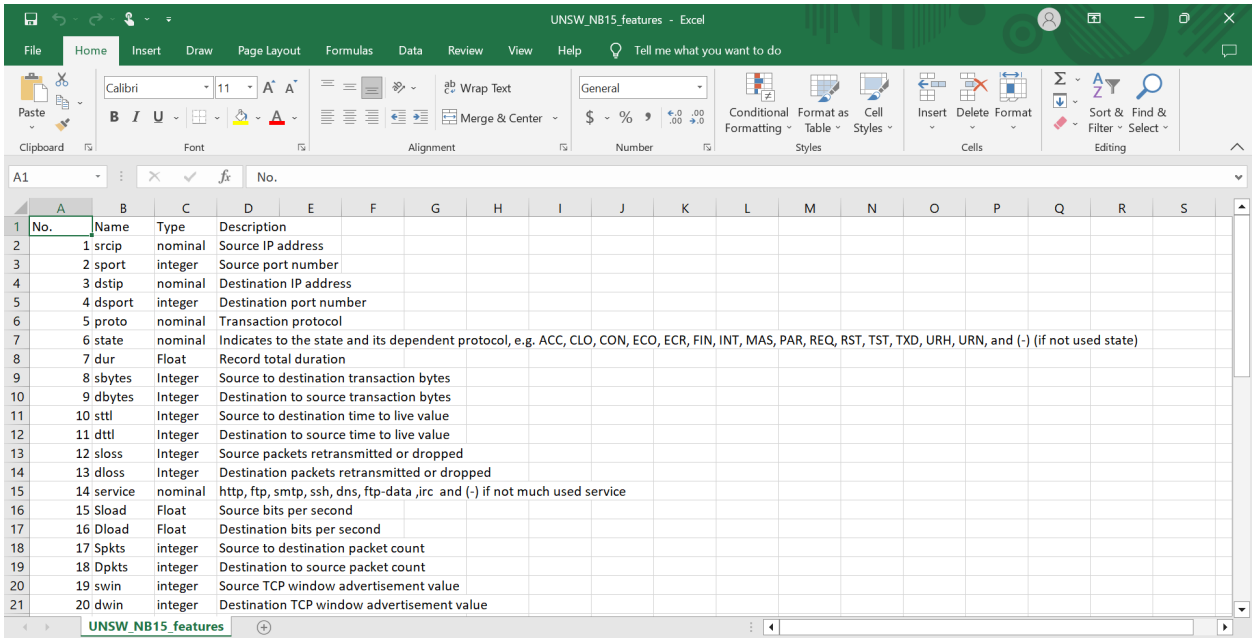
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | dttl | swin | dwin | tcprtt | synack | ackdat | label | proto_tcp | proto_udp | service_dr | state_CON | state_FIN | attack_cat | attack_cat | attack_cat | attack_cat_Normal | |
| 2 | 3 | 0.992126 | 1 | 1 | 0 | 0 | 0 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 3 | 11 | 0.992126 | 1 | 1 | 0.087972 | 0.103 | 0.054975 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 4 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 5 | 17 | 0.992126 | 1 | 1 | 0.080709 | 0.115373 | 0.042914 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 6 | 21 | 0.992126 | 1 | 1 | 0.103597 | 0.107399 | 0.069747 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 7 | 22 | 0.992126 | 1 | 1 | 0.07542 | 0.102488 | 0.042021 | 6 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | |
| 8 | 28 | 0.992126 | 1 | 1 | 0.056946 | 0.085484 | 0.028809 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 9 | 30 | 0.992126 | 1 | 1 | 0.07084 | 0.072288 | 0.048108 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 10 | 31 | 0.992126 | 1 | 1 | 0.067508 | 0.09819 | 0.035287 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 11 | 32 | 0.992126 | 1 | 1 | 0.061193 | 0.056538 | 0.043685 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 12 | 34 | 0.992126 | 1 | 1 | 0.078752 | 0.05629 | 0.062155 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 13 | 35 | 0.992126 | 1 | 1 | 0.058599 | 0.090387 | 0.028772 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 14 | 39 | 0.992126 | 1 | 1 | 0.060079 | 0.06972 | 0.037769 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 15 | 41 | 0.992126 | 1 | 1 | 0.070361 | 0.065924 | 0.0499 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 16 | 43 | 0.992126 | 1 | 1 | 0.051519 | 0.040372 | 0.039383 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 17 | 44 | 0.992126 | 1 | 1 | 0.07816 | 0.10302 | 0.044698 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 18 | 45 | 0.992126 | 1 | 1 | 0.05965 | 0.058557 | 0.041342 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 19 | 47 | 0.992126 | 1 | 1 | 0.049104 | 0.039914 | 0.03702 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 20 | 48 | 0.992126 | 1 | 1 | 0.054326 | 0.041978 | 0.041743 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 21 | 56 | 0.992126 | 1 | 1 | 0.046071 | 0.032974 | 0.036345 | 6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |

# Binary Dataset:



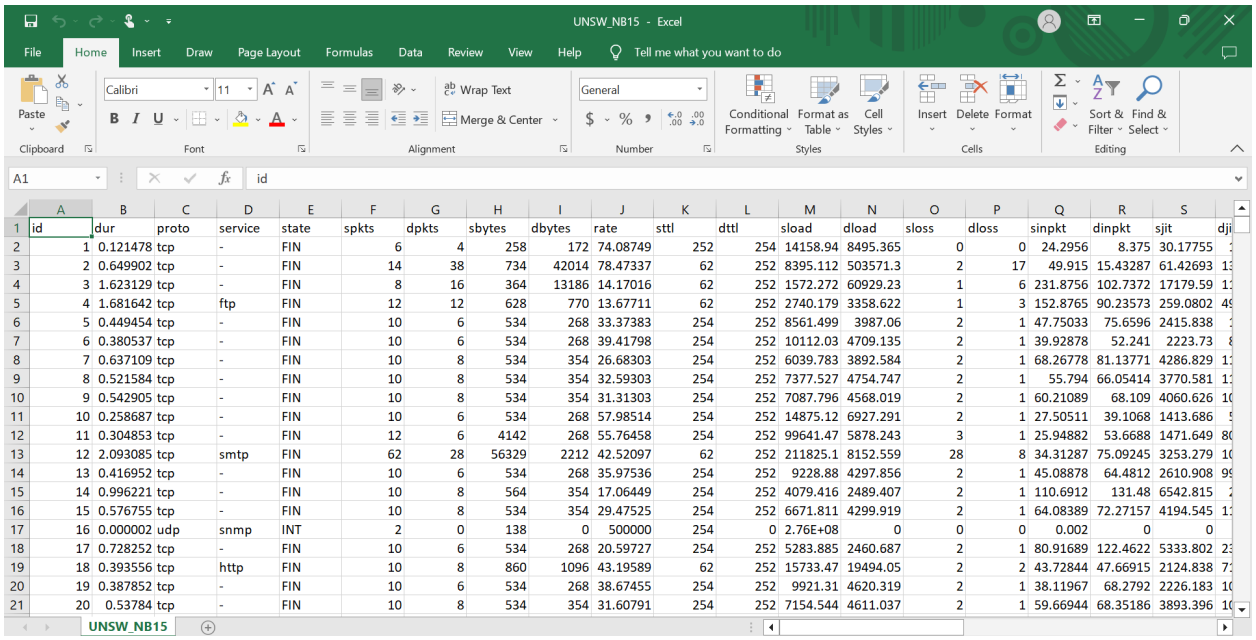
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | rate | sttl | sload | dload | ct_srv_src | ct_state_t | ct_dst_ltm | ct_src_dp | ct_dst_spc | ct_dst_src | ct_src_ltm | ct_srv_dst | state_CON | state_INT | label |
| 2 | 3 | 1.37E-05 | 0.138393 | 1.19E-06 | 0.00015 | 0 | 0.166667 | 0.02 | 0 | 0 | 0.039216 | 0.016949 | 0 | 0 | 0 | 1 |
| 3 | 11 | 4.25E-05 | 0.138393 | 9.19E-05 | 0.000364 | 0.019608 | 0.166667 | 0 | 0 | 0 | 0.019608 | 0 | 0 | 0 | 0 | 1 |
| 4 | 15 | 0.5 | 0.995536 | 0.119792 | 0 | 0 | 0.333333 | 0 | 0 | 0 | 0.058824 | 0.016949 | 0 | 0 | 1 | 1 |
| 5 | 17 | 4.32E-05 | 0.138393 | 6.83E-06 | 0.000869 | 0.039216 | 0.166667 | 0 | 0 | 0 | 0.019608 | 0 | 0.039216 | 0 | 0 | 1 |
| 6 | 21 | 4.44E-05 | 0.995536 | 9.23E-06 | 0.000236 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0 | 0.016949 | 0.039216 | 0 | 0 | 1 |
| 7 | 22 | 2.59E-05 | 0.138393 | 2.31E-06 | 0.000322 | 0 | 0.5 | 0 | 0 | 0 | 0.019608 | 0.016949 | 0 | 1 | 0 | 1 |
| 8 | 28 | 3.11E-05 | 0.995536 | 6.68E-06 | 0.000594 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0.019608 | 0 | 0.137255 | 0 | 0 | 1 |
| 9 | 30 | 3.44E-05 | 0.138393 | 3.54E-06 | 0.00563 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0.019608 | 0.016949 | 0 | 0 | 0 | 1 |
| 10 | 31 | 5.46E-05 | 0.138393 | 1.40E-05 | 0.000291 | 0.078431 | 0.166667 | 0 | 0 | 0 | 0.058824 | 0 | 0.137255 | 0 | 0 | 1 |
| 11 | 32 | 5.04E-05 | 0.138393 | 7.41E-06 | 0.002091 | 0.039216 | 0.166667 | 0 | 0 | 0 | 0 | 0 | 0.137255 | 0 | 0 | 1 |
| 12 | 34 | 5.06E-05 | 0.995536 | 1.04E-05 | 0.000269 | 0.039216 | 0.166667 | 0 | 0 | 0 | 0 | 0 | 0.039216 | 0 | 0 | 1 |
| 13 | 35 | 3.86E-05 | 0.138393 | 8.85E-06 | 0.001499 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0.039216 | 0 | 0.019608 | 0 | 0 | 1 |
| 14 | 39 | 5.19E-05 | 0.995536 | 8.16E-06 | 0.000277 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0.039216 | 0 | 0 | 0 | 0 | 1 |
| 15 | 41 | 3.19E-05 | 0.138393 | 4.42E-06 | 0.000784 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0.019608 | 0 | 0.137255 | 0 | 0 | 1 |
| 16 | 43 | 4.99E-05 | 0.138393 | 7.38E-06 | 0.002486 | 0.078431 | 0.166667 | 0 | 0 | 0 | 0.058824 | 0 | 0.137255 | 0 | 0 | 1 |
| 17 | 44 | 3.20E-05 | 0.138393 | 4.17E-06 | 0.001668 | 0.078431 | 0.166667 | 0 | 0 | 0 | 0.058824 | 0 | 0.137255 | 0 | 0 | 1 |
| 18 | 45 | 8.45E-05 | 0.138393 | 9.34E-06 | 0.026028 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0.764706 | 0 | 0.137255 | 0 | 0 | 1 |
| 19 | 47 | 5.74E-05 | 0.138393 | 1.02E-05 | 0.004155 | 0.098039 | 0.166667 | 0 | 0 | 0 | 0.019608 | 0 | 0 | 0 | 0 | 1 |
| 20 | 48 | 4.40E-05 | 0.138393 | 4.70E-05 | 0.000587 | 0.019608 | 0.166667 | 0.02 | 0 | 0 | 0.019608 | 0 | 0.019608 | 0 | 0 | 1 |
| 21 | 56 | 5.58E-05 | 0.138393 | 8.64E-06 | 0.002538 | 0.078431 | 0.166667 | 0 | 0 | 0 | 0.058824 | 0 | 0.039216 | 0 | 0 | 1 |

# UNSW_NB15_features:



| No. | Name | Type | Description |
|---|---|---|---|
| 1 | srcip | nominal | Source IP address |
| 2 | sport | integer | Source port number |
| 3 | dstip | nominal | Destination IP address |
| 4 | dsport | integer | Destination port number |
| 5 | proto | nominal | Transaction protocol |
| 6 | state | nominal | Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state) |
| 7 | dur | Float | Record total duration |
| 8 | sbytes | Integer | Source to destination transaction bytes |
| 9 | dbytes | Integer | Destination to source transaction bytes |
| 10 | sttl | Integer | Source to destination time to live value |
| 11 | dttl | Integer | Destination to source time to live value |
| 12 | sloss | Integer | Source packets retransmitted or dropped |
| 13 | dloss | Integer | Destination packets retransmitted or dropped |
| 14 | service | nominal | http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service |
| 15 | Sload | Float | Source bits per second |
| 16 | Dload | Float | Destination bits per second |
| 17 | Spkts | integer | Source to destination packet count |
| 18 | Dpkts | integer | Destination to source packet count |
| 19 | swin | integer | Source TCP window advertisement value |
| 20 | dwin | integer | Destination TCP window advertisement value |

# UNSW_NB15:



| id | dur | proto | service | state | spkts | dpkts | sbytes | dbytes | rate | sttl | dttl | sload | dload | sloss | dloss | sinpkt | dinpkt | sjit | dji |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.121478 | tcp | - | FIN | 6 | 4 | 258 | 172 | 74.08749 | 252 | 254 | 14158.94 | 8495.365 | 0 | 0 | 24.2956 | 8.375 | 30.17755 | 1 |
| 2 | 0.649902 | tcp | - | FIN | 14 | 38 | 734 | 42014 | 78.47337 | 62 | 252 | 8395.112 | 503571.3 | 2 | 17 | 49.915 | 15.43287 | 61.42693 | 13 |
| 3 | 1.623129 | tcp | - | FIN | 8 | 16 | 364 | 13186 | 14.17016 | 62 | 252 | 1572.272 | 60929.23 | 1 | 6 | 231.8756 | 102.7372 | 17179.59 | 1 |
| 4 | 1.681642 | tcp | ftp | FIN | 12 | 12 | 628 | 770 | 13.67711 | 62 | 252 | 2740.179 | 3358.622 | 1 | 3 | 152.8765 | 90.23573 | 259.0802 | 49 |
| 5 | 0.449454 | tcp | - | FIN | 10 | 6 | 534 | 268 | 33.37383 | 254 | 252 | 8561.499 | 3987.06 | 2 | 1 | 47.75033 | 75.6596 | 2415.838 | 1 |
| 6 | 0.380537 | tcp | - | FIN | 10 | 6 | 534 | 268 | 39.41798 | 254 | 252 | 10112.03 | 4709.135 | 2 | 1 | 39.92878 | 52.241 | 2223.73 | 8 |
| 7 | 0.637109 | tcp | - | FIN | 10 | 8 | 534 | 354 | 26.68303 | 254 | 252 | 6039.783 | 3892.584 | 2 | 1 | 68.26778 | 81.13771 | 4286.829 | 1 |
| 8 | 0.521584 | tcp | - | FIN | 10 | 8 | 534 | 354 | 32.59303 | 254 | 252 | 7377.527 | 4754.747 | 2 | 1 | 55.794 | 66.05414 | 3770.581 | 1 |
| 9 | 0.542905 | tcp | - | FIN | 10 | 8 | 534 | 354 | 31.31303 | 254 | 252 | 7087.796 | 4568.019 | 2 | 1 | 60.21089 | 68.109 | 4060.626 | 10 |
| 10 | 0.258687 | tcp | - | FIN | 10 | 6 | 534 | 268 | 57.98514 | 254 | 252 | 14875.12 | 6927.291 | 2 | 1 | 27.50511 | 39.1068 | 1413.686 | 5 |
| 11 | 0.304853 | tcp | - | FIN | 12 | 6 | 4142 | 268 | 55.76458 | 254 | 252 | 99641.47 | 5878.243 | 3 | 1 | 25.94882 | 53.6688 | 1471.649 | 80 |
| 12 | 2.093085 | tcp | smtp | FIN | 62 | 28 | 56329 | 2212 | 42.52097 | 62 | 252 | 211825.1 | 8152.559 | 28 | 8 | 34.31287 | 75.09245 | 3253.279 | 10 |
| 13 | 0.416952 | tcp | - | FIN | 10 | 6 | 534 | 268 | 35.97536 | 254 | 252 | 9228.88 | 4297.856 | 2 | 1 | 45.08878 | 64.4812 | 2610.908 | 99 |
| 14 | 0.996221 | tcp | - | FIN | 10 | 8 | 564 | 354 | 17.06449 | 254 | 252 | 4079.416 | 2489.407 | 2 | 1 | 110.6912 | 131.48 | 6542.815 | 2 |
| 15 | 0.576755 | tcp | - | FIN | 10 | 8 | 534 | 354 | 29.47525 | 254 | 252 | 6671.811 | 4299.919 | 2 | 1 | 64.08389 | 72.27157 | 4194.545 | 1 |
| 16 | 0.000002 | udp | snmp | INT | 2 | 0 | 138 | 0 | 500000 | 254 | 0 | 2.76E+08 | 0 | 0 | 0 | 0.002 | 0 | 0 | |
| 17 | 0.728252 | tcp | - | FIN | 10 | 6 | 534 | 268 | 20.59727 | 254 | 252 | 5283.885 | 2460.687 | 2 | 1 | 80.91689 | 122.4622 | 5333.802 | 23 |
| 18 | 0.393556 | tcp | http | FIN | 10 | 8 | 860 | 1096 | 43.19589 | 62 | 252 | 15733.47 | 19494.05 | 2 | 2 | 43.72844 | 47.66915 | 2124.838 | 7 |
| 19 | 0.387852 | tcp | - | FIN | 10 | 6 | 534 | 268 | 38.67455 | 254 | 252 | 9921.31 | 4620.319 | 2 | 1 | 38.11967 | 68.2792 | 2226.183 | 10 |
| 20 | 0.53784 | tcp | - | FIN | 10 | 8 | 534 | 354 | 31.60791 | 254 | 252 | 7154.544 | 4611.037 | 2 | 1 | 59.66944 | 68.35186 | 3893.396 | 10 |

## Expected Result

Our IoT network intrusion system is designed to detect and prevent unauthorized access to an IoT network, which can result in data theft, system damage, and privacy breaches. The expected results of an IoT network intrusion system include:

The primary goal of an IoT network intrusion system is to detect any unauthorized access attempts or suspicious behavior within the network. The system should be able to identify potential threats, such as malware or unauthorized devices, and trigger alerts to the network administrators.

Our IoT network intrusion system will be able to collect and analyze threat intelligence data to identify new and emerging threats. This could include analyzing traffic patterns, device behavior, and user activity to detect any potential threats.

In the event of a security breach, IoT network intrusion system will be able to provide detailed forensic analysis to identify the root cause of the breach. This analysis can be used to improve security measures and prevent similar attacks from occurring in the future.

Our system will be able to scale to accommodate the growing number of devices connected to the network. This includes the ability to manage large volumes of data.

Overall, an IoT network intrusion system is essential for protecting the security and privacy of IoT devices. The expected results of such a system include threat detection and compliance with industry regulations. The system will be scalable, able to handle large volumes of data, and provide detailed forensic analysis in the event of a security breach.

## Working and Code Implementation

# Importing Modules and Libraries

```
# importing required libraries
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

import pickle
from os import path

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

# Importing Datasets

```
#data = pd.read_csv('/content/drive/MyDrive/IoT_ML_Dataset/UNSW/UNSW_NB15.csv')
data = pd.read_csv('datasets/UNSW/UNSW_NB15.csv')
```

```python
data.head(n=5)
```

Python

| | id | dur | proto | service | state | spkts | dpkts | sbytes | dbytes | rate | sttl | dttl | sload | dload | sloss | dloss | sinpkt | dinpkt | sjit | djit | swin | stcpb | dt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.121478 | tcp | - | FIN | 6 | 4 | 258 | 172 | 74.087490 | 252 | 254 | 14158.942380 | 8495.365234 | 0 | 0 | 24.295600 | 8.375000 | 30.177547 | 11.830604 | 255 | 621772692 | 2202533 |
| 1 | 2 | 0.649902 | tcp | - | FIN | 14 | 38 | 734 | 42014 | 78.473372 | 62 | 252 | 8395.112305 | 503571.312500 | 2 | 17 | 49.915000 | 15.432865 | 61.426934 | 1387.778330 | 255 | 1417884146 | 3077387 |
| 2 | 3 | 1.623129 | tcp | - | FIN | 8 | 16 | 364 | 13186 | 14.170161 | 62 | 252 | 1572.271851 | 60929.230470 | 1 | 6 | 231.875571 | 102.737203 | 17179.586860 | 11420.926230 | 255 | 2116150707 | 2963114 |
| 3 | 4 | 1.681642 | tcp | ftp | FIN | 12 | 12 | 628 | 770 | 13.677108 | 62 | 252 | 2740.178955 | 3358.622070 | 1 | 3 | 152.876547 | 90.235726 | 259.080172 | 4991.784669 | 255 | 1107119177 | 1047442 |
| 4 | 5 | 0.449454 | tcp | - | FIN | 10 | 6 | 534 | 268 | 33.373826 | 254 | 252 | 8561.499023 | 3987.059814 | 2 | 1 | 47.750333 | 75.659602 | 2415.837634 | 115.807000 | 255 | 2436137549 | 1977154 |

```python
data.info()
```

Python

```
Output exceeds the size limit. Open the full output data in a text editor
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175341 entries, 0 to 175340
Data columns (total 45 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   id                 175341 non-null  int64
 1   dur                175341 non-null  float64
 2   proto              175341 non-null  object
 3   service            175341 non-null  object
 4   state              175341 non-null  object
 5   spkts              175341 non-null  int64
 6   dpkts              175341 non-null  int64
 7   sbytes             175341 non-null  int64
 8   dbytes             175341 non-null  int64
 9   rate               175341 non-null  float64
 10  sttl               175341 non-null  int64
 11  dttl               175341 non-null  int64
 12  sload              175341 non-null  float64
 13  dload              175341 non-null  float64
 14  sloss              175341 non-null  int64
 15  dloss              175341 non-null  int64
 16  sinpkt             175341 non-null  float64
 17  dinpkt             175341 non-null  float64
 18  sjit               175341 non-null  float64
 19  djit               175341 non-null  float64
```

```
...
 43  attack_cat         175341 non-null  object
 44  label              175341 non-null  int64
dtypes: float64(11), int64(30), object(4)
memory usage: 60.2+ MB
```

```python
data[data['service']=='-']
```

Python

| | id | dur | proto | service | state | spkts | dpkts | sbytes | dbytes | rate | sttl | dttl | sload | dload | sloss | dloss | sinpkt | dinpkt | sjit | djit | swin | stcp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.121478 | tcp | - | FIN | 6 | 4 | 258 | 172 | 74.087490 | 252 | 254 | 14158.942380 | 8495.365234 | 0 | 0 | 24.295600 | 8.375000 | 30.177547 | 11.830604 | 255 | 6217726 |
| 1 | 2 | 0.649902 | tcp | - | FIN | 14 | 38 | 734 | 42014 | 78.473372 | 62 | 252 | 8395.112305 | 503571.312500 | 2 | 17 | 49.915000 | 15.432865 | 61.426934 | 1387.778330 | 255 | 14178841 |
| 2 | 3 | 1.623129 | tcp | - | FIN | 8 | 16 | 364 | 13186 | 14.170161 | 62 | 252 | 1572.271851 | 60929.230470 | 1 | 6 | 231.875571 | 102.737203 | 17179.586860 | 11420.926230 | 255 | 21161507 |
| 4 | 5 | 0.449454 | tcp | - | FIN | 10 | 6 | 534 | 268 | 33.373826 | 254 | 252 | 8561.499023 | 3987.059814 | 2 | 1 | 47.750333 | 75.659602 | 2415.837634 | 115.807000 | 255 | 24361375 |
| 5 | 6 | 0.380537 | tcp | - | FIN | 10 | 6 | 534 | 268 | 39.417980 | 254 | 252 | 10112.025390 | 4709.134766 | 2 | 1 | 39.928778 | 52.241000 | 2223.730342 | 82.550500 | 255 | 39841555 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 175125 | 175126 | 0.653375 | tcp | - | FIN | 10 | 8 | 564 | 354 | 26.018748 | 254 | 252 | 6220.011230 | 3795.676025 | 2 | 1 | 70.986667 | 89.141570 | 5392.277592 | 133.977047 | 255 | 22538638 |
| 175179 | 175180 | 0.695566 | tcp | - | FIN | 10 | 8 | 564 | 354 | 24.440528 | 254 | 252 | 5842.723633 | 3565.441650 | 2 | 1 | 76.977778 | 90.474711 | 5483.590978 | 137.802578 | 255 | 10074441 |
| 175191 | 175192 | 0.962856 | tcp | - | FIN | 24 | 60 | 1256 | 59374 | 86.201883 | 62 | 252 | 10003.573240 | 485098.500000 | 6 | 28 | 41.863305 | 16.133339 | 2031.193057 | 1279.137355 | 255 | 6539110 |
| 175277 | 175278 | 3.719110 | tcp | - | FIN | 66 | 340 | 3086 | 426483 | 108.897021 | 62 | 252 | 6539.198730 | 914689.812500 | 6 | 167 | 57.217077 | 10.914012 | 3358.369752 | 1585.622696 | 255 | 30468701 |
| 175337 | 175338 | 0.505762 | tcp | - | FIN | 10 | 8 | 620 | 354 | 33.612649 | 254 | 252 | 8826.286133 | 4903.492188 | 2 | 1 | 54.400111 | 66.980570 | 3721.068786 | 120.177727 | 255 | 35187762 |

94168 rows × 45 columns

```python
data['service'].replace('-',np.nan,inplace=True)
```

Python

+ Code + Markdown

```python
data.isnull().sum()
```

Python

```
Output exceeds the size limit. Open the full output data in a text editor
id                 0
dur                0
proto              0
service        94168
state              0
spkts              0
```

```
dpkts                0
sbytes               0
dbytes               0
rate                 0
sttl                 0
dttl                 0
sload                0
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
djit                 0
swin                 0
stcpb                0
dtcpb                0
dwin                 0
tcprtt               0
...
ct_srv_dst           0
is_sm_ips_ports      0
attack_cat           0
label                0
dtype: int64
```

```
data.shape
```

```
(175341, 45)
```

```
data.dropna(inplace=True)
```

```
data.shape
```

```
(81173, 45)
```

```
data['attack_cat'].value_counts()
```

```
Generic           39496
Normal            19488
Exploits          16187
DoS                1791
Fuzzers            1731
Reconnaissance     1703
Analysis            564
Worms               114
Backdoor             99
Name: attack_cat, dtype: int64
```

```python
data['state'].value_counts()
```

Python

```
INT    39580
FIN    33640
CON     7711
REQ      228
RST       14
Name: state, dtype: int64
```

```python
data
```

Python

| | id | dur | proto | service | state | spkts | dpkts | sbytes | dbytes | rate | sttl | dttl | sload | dload | sloss | dloss | sinpkt | dinpkt | sjit | djit | swin | stcpl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 1.681642 | tcp | ftp | FIN | 12 | 12 | 628 | 770 | 13.677108 | 62 | 252 | 2.740179e+03 | 3358.622070 | 1 | 3 | 152.876547 | 90.235726 | 259.080172 | 4991.784669 | 255 | 1107119177 |
| 11 | 12 | 2.093085 | tcp | smtp | FIN | 62 | 28 | 56329 | 2212 | 42.520967 | 62 | 252 | 2.118251e+05 | 8152.559082 | 28 | 8 | 34.312868 | 75.092445 | 3253.278833 | 106.113453 | 255 | 1824722662 |
| 15 | 16 | 0.000002 | udp | snmp | INT | 2 | 0 | 138 | 0 | 500000.001300 | 254 | 0 | 2.760000e+08 | 0.000000 | 0 | 0 | 0.002000 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 |
| 17 | 18 | 0.393556 | tcp | http | FIN | 10 | 8 | 860 | 1096 | 43.195886 | 62 | 252 | 1.573347e+04 | 19494.048830 | 2 | 2 | 43.728444 | 47.669145 | 2124.837873 | 71.805867 | 255 | 3882971404 |
| 21 | 22 | 0.338017 | tcp | http | FIN | 10 | 6 | 998 | 268 | 44.376468 | 254 | 252 | 2.127704e+04 | 5301.508789 | 2 | 1 | 35.218111 | 56.579801 | 1928.550710 | 82.022258 | 255 | 2665974075 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 175335 | 175336 | 0.000006 | udp | dns | INT | 2 | 0 | 114 | 0 | 166666.660800 | 254 | 0 | 7.600000e+07 | 0.000000 | 0 | 0 | 0.006000 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 |
| 175336 | 175337 | 0.000009 | udp | dns | INT | 2 | 0 | 114 | 0 | 111111.107200 | 254 | 0 | 5.066666e+07 | 0.000000 | 0 | 0 | 0.009000 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 |
| 175338 | 175339 | 0.000009 | udp | dns | INT | 2 | 0 | 114 | 0 | 111111.107200 | 254 | 0 | 5.066666e+07 | 0.000000 | 0 | 0 | 0.009000 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 |
| 175339 | 175340 | 0.000009 | udp | dns | INT | 2 | 0 | 114 | 0 | 111111.107200 | 254 | 0 | 5.066666e+07 | 0.000000 | 0 | 0 | 0.009000 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 |
| 175340 | 175341 | 0.000009 | udp | dns | INT | 2 | 0 | 114 | 0 | 111111.107200 | 254 | 0 | 5.066666e+07 | 0.000000 | 0 | 0 | 0.009000 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 |

81173 rows × 45 columns

```python
#features = pd.read_csv('/content/drive/MyDrive/IoT_ML_Dataset/UNSW/UNSW_NB15_features.csv')
features = pd.read_csv('datasets/UNSW/UNSW_NB15_features.csv')
```

```python
features.head()
```

| | No. | Name | Type | Description |
|---|---|---|---|---|
| 0 | 1 | srcip | nominal | Source IP address |
| 1 | 2 | sport | integer | Source port number |
| 2 | 3 | dstip | nominal | Destination IP address |
| 3 | 4 | dsport | integer | Destination port number |
| 4 | 5 | proto | nominal | Transaction protocol |

```python
features['Type '] = features['Type '].str.lower()
```

```python
# selecting column names of all data types
nominal_names = features['Name'][features['Type ']=='nominal']
integer_names = features['Name'][features['Type ']=='integer']
binary_names = features['Name'][features['Type ']=='binary']
float_names = features['Name'][features['Type ']=='float']
```

+ Code + Markdown

```python
# selecting common column names from dataset and feature dataset
cols = data.columns
nominal_names = cols.intersection(nominal_names)
integer_names = cols.intersection(integer_names)
binary_names = cols.intersection(binary_names)
float_names = cols.intersection(float_names)
```

```python
# Converting integer columns to numeric
for c in integer_names:
    pd.to_numeric(data[c])
```

```python
# Converting binary columns to numeric
for c in binary_names:
    pd.to_numeric(data[c])
```

```python
# Converting float columns to numeric
for c in float_names:
    pd.to_numeric(data[c])
```

```python
data.info()
```

```
Output exceeds the size limit. Open the full output data in a text editor
<class 'pandas.core.frame.DataFrame'>
Int64Index: 81173 entries, 3 to 175340
Data columns (total 45 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 81173 non-null  int64
 1   dur                81173 non-null  float64
 2   proto              81173 non-null  object
 3   service            81173 non-null  object
 4   state              81173 non-null  object
 5   spkts              81173 non-null  int64
 6   dpkts              81173 non-null  int64
 7   sbytes             81173 non-null  int64
 8   dbytes             81173 non-null  int64
 9   rate               81173 non-null  float64
 10  sttl               81173 non-null  int64
 11  dttl               81173 non-null  int64
 12  sload              81173 non-null  float64
 13  dload              81173 non-null  float64
 14  sloss              81173 non-null  int64
 15  dloss              81173 non-null  int64
 16  sinpkt             81173 non-null  float64
 17  dinpkt             81173 non-null  float64
 18  sjit               81173 non-null  float64
 19  djit               81173 non-null  float64
...
 43  attack_cat         81173 non-null  object
 44  label              81173 non-null  int64
dtypes: float64(11), int64(30), object(4)
memory usage: 28.5+ MB
```

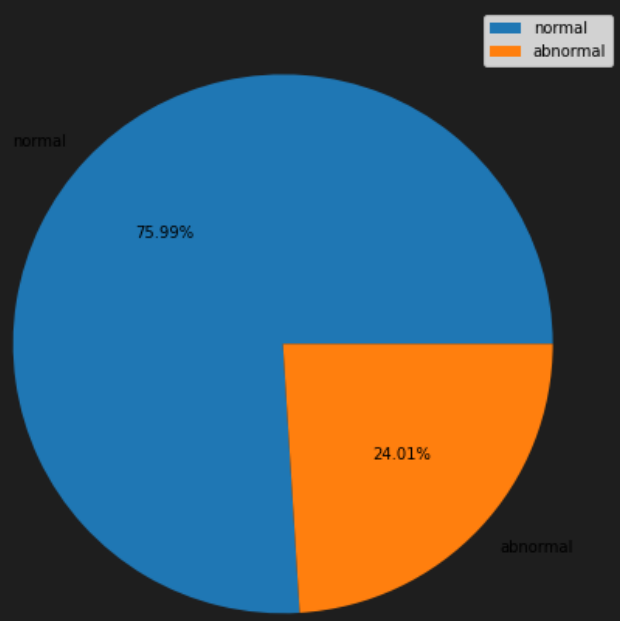
# Data Visualization

## Binary Classification

```
plt.figure(figsize=(8,8))
plt.pie(data.label.value_counts(),labels=['normal','abnormal'],autopct='%0.2f%%')
plt.title("Pie chart distribution of normal and abnormal labels",fontsize=16)
plt.legend()
plt.savefig('plots/Pie_chart_binary.png')
plt.show()
```

...

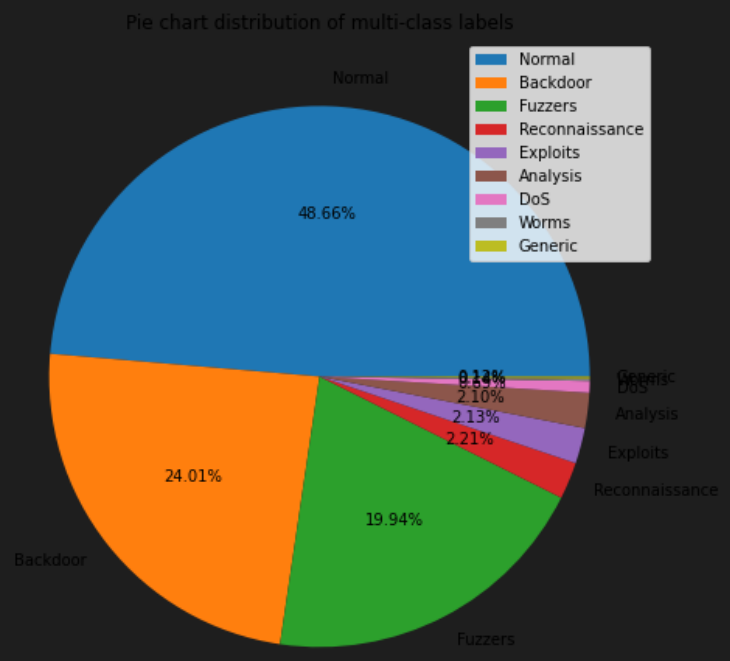
Pie chart distribution of normal and abnormal labels



# Multi-class Classification

```python
plt.figure(figsize=(8,8))
plt.pie(data.attack_cat.value_counts(),labels=data.attack_cat.unique(),autopct='%0.2f%%')
plt.title('Pie chart distribution of multi-class labels')
plt.legend(loc='best')
plt.savefig('plots/Pie_chart_multi.png')
plt.show()
```

...



# One hot encoding

```python
num_col = data.select_dtypes(include='number').columns

# selecting categorical data attributes
cat_col = data.columns.difference(num_col)
cat_col = cat_col[1:]
cat_col
```

```
Index(['proto', 'service', 'state'], dtype='object')
```

```python
# creating a dataframe with only categorical attributes
data_cat = data[cat_col].copy()
data_cat.head()
```

| | proto | service | state |
|---|---|---|---|
| 3 | tcp | ftp | FIN |
| 11 | tcp | smtp | FIN |
| 15 | udp | snmp | INT |
| 17 | tcp | http | FIN |
| 21 | tcp | http | FIN |

```python
# one-hot-encoding categorical attributes using pandas.get_dummies() function
data_cat = pd.get_dummies(data_cat,columns=cat_col)
```

```python
data_cat.head()
```

| | proto_tcp | proto_udp | service_dhcp | service_dns | service_ftp | service_ftp-data | service_http | service_irc | service_pop3 | service_radius | service_smtp | service_snmp | service_ssh | service_ssl | state_CON | state_FIN | state_IN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 11 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | |
| 15 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 17 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 21 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |

```python
data.shape
```

```
(81173, 45)
```

```python
data = pd.concat([data, data_cat],axis=1)
```

```python
data.shape
```

```
(81173, 64)
```

```python
data.drop(columns=cat_col,inplace=True)
```

```python
data.shape
```

```
(81173, 61)
```

# Data Normalization

```python
# selecting numeric attributes columns from data
num_col = list(data.select_dtypes(include='number').columns)
num_col.remove('id')
num_col.remove('label')
print(num_col)
```

Python

```
['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl', 'dttl', 'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit', 'swin', 'stcpb', 'dtcpb', 'dwin', 'tcprtt', 'synack', 'ackd
```

```python
# using minmax scaler for normalizing data
minmax_scale = MinMaxScaler(feature_range=(0, 1))
def normalization(df,col):
  for i in col:
    arr = df[i]
    arr = np.array(arr)
    df[i] = minmax_scale.fit_transform(arr.reshape(len(arr),1))
  return df
```

Python

+ Code + Markdown

```python
# data before normalization
data.head()
```

Python

| | id | dur | spkts | dpkts | sbytes | dbytes | rate | sttl | dttl | sload | dload | sloss | dloss | sinpkt | dinpkt | sjit | djit | swin | stcpb | dtcpb | dwin | tcprtt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 1.681642 | 12 | 12 | 628 | 770 | 13.677108 | 62 | 252 | 2.740179e+03 | 3358.622070 | 1 | 3 | 152.876547 | 90.235726 | 259.080172 | 4991.784669 | 255 | 1107119177 | 1047442890 | 255 | 0.000000 |
| 11 | 12 | 2.093085 | 62 | 28 | 56329 | 2212 | 42.520967 | 62 | 252 | 2.118251e+05 | 8152.559082 | 28 | 8 | 34.312868 | 75.092445 | 3253.278833 | 106.113453 | 255 | 1824722662 | 860716719 | 255 | 0.131140 |
| 15 | 16 | 0.000002 | 2 | 0 | 138 | 0 | 500000.001300 | 254 | 0 | 2.760000e+08 | 0.000000 | 0 | 0 | 0.002000 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 | 0 | 0 | 0.000000 |
| 17 | 18 | 0.393556 | 10 | 8 | 860 | 1096 | 43.195886 | 62 | 252 | 1.573347e+04 | 19494.048830 | 2 | 2 | 43.728444 | 47.669145 | 2124.837873 | 71.805867 | 255 | 3882971404 | 3084071099 | 255 | 0.120313 |
| 21 | 22 | 0.338017 | 10 | 6 | 998 | 268 | 44.376468 | 254 | 252 | 2.127704e+04 | 5301.508789 | 2 | 1 | 35.218111 | 56.579801 | 1928.550710 | 82.022258 | 255 | 2665974075 | 3521361798 | 255 | 0.154433 |

```python
# calling normalization() function
data = normalization(data.copy(),num_col)
```

```python
# data after normalization
data.head()
```

Python

| | id | dur | spkts | dpkts | sbytes | dbytes | rate | sttl | dttl | sload | dload | sloss | dloss | sinpkt | dinpkt | sjit | djit | swin | stcpb | dtcpb | dwin | tcprtt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 2.802780e-02 | 0.001144 | 0.001093 | 0.000041 | 0.000053 | 0.000014 | 0.138393 | 0.992126 | 0.000001 | 0.000150 | 0.000208 | 0.000547 | 1.092583e-02 | 0.006449 | 0.000275 | 0.017249 | 1.0 | 0.257772 | 0.243888 | 1.0 | 0.000000 |
| 11 | 12 | 3.488529e-02 | 0.006344 | 0.002551 | 0.004337 | 0.000151 | 0.000043 | 0.138393 | 0.992126 | 0.000092 | 0.000364 | 0.005830 | 0.001459 | 2.452283e-03 | 0.005367 | 0.003456 | 0.000367 | 1.0 | 0.424852 | 0.200410 | 1.0 | 0.087972 |
| 15 | 16 | 1.666693e-08 | 0.000104 | 0.000000 | 0.000003 | 0.000000 | 0.500000 | 0.995536 | 0.000000 | 0.119792 | 0.000000 | 0.000000 | 0.000000 | 1.429367e-07 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 0.000000 | 0.0 | 0.000000 |
| 17 | 18 | 6.559354e-03 | 0.000936 | 0.000729 | 0.000059 | 0.000075 | 0.000043 | 0.138393 | 0.992126 | 0.000007 | 0.000869 | 0.000416 | 0.000365 | 3.125199e-03 | 0.003407 | 0.002257 | 0.000248 | 1.0 | 0.904076 | 0.718098 | 1.0 | 0.080709 |
| 21 | 22 | 5.633690e-03 | 0.000936 | 0.000547 | 0.000069 | 0.000018 | 0.000044 | 0.995536 | 0.992126 | 0.000009 | 0.000236 | 0.000416 | 0.000182 | 2.516980e-03 | 0.004044 | 0.002049 | 0.000283 | 1.0 | 0.620722 | 0.819917 | 1.0 | 0.103597 |

# Label Encoding

## Binary Labels

```python
# changing attack labels into two categories 'normal' and 'abnormal'
bin_label = pd.DataFrame(data.label.map(lambda x:'normal' if x==0 else 'abnormal'))
```

Python

```python
# creating a dataframe with binary labels (normal,abnormal)
bin_data = data.copy()
bin_data['label'] = bin_label
```

Python

```python
# label encoding (0,1) binary labels
le1 = preprocessing.LabelEncoder()
enc_label = bin_label.apply(le1.fit_transform)
bin_data['label'] = enc_label
```

```
le1.classes_
```

```
array(['abnormal', 'normal'], dtype=object)
```

```
np.save("le1_classes.npy",le1.classes_,allow_pickle=True)
```

# Multi-class Labels

```
# one-hot-encoding attack label
multi_data = data.copy()
multi_label = pd.DataFrame(multi_data.attack_cat)
```

```
multi_data = pd.get_dummies(multi_data,columns=['attack_cat'])
```

```
# label encoding (0,1,2,3,4,5,6,7,8) multi-class labels
le2 = preprocessing.LabelEncoder()
enc_label = multi_label.apply(le2.fit_transform)
multi_data['label'] = enc_label
```

```
le2.classes_
```

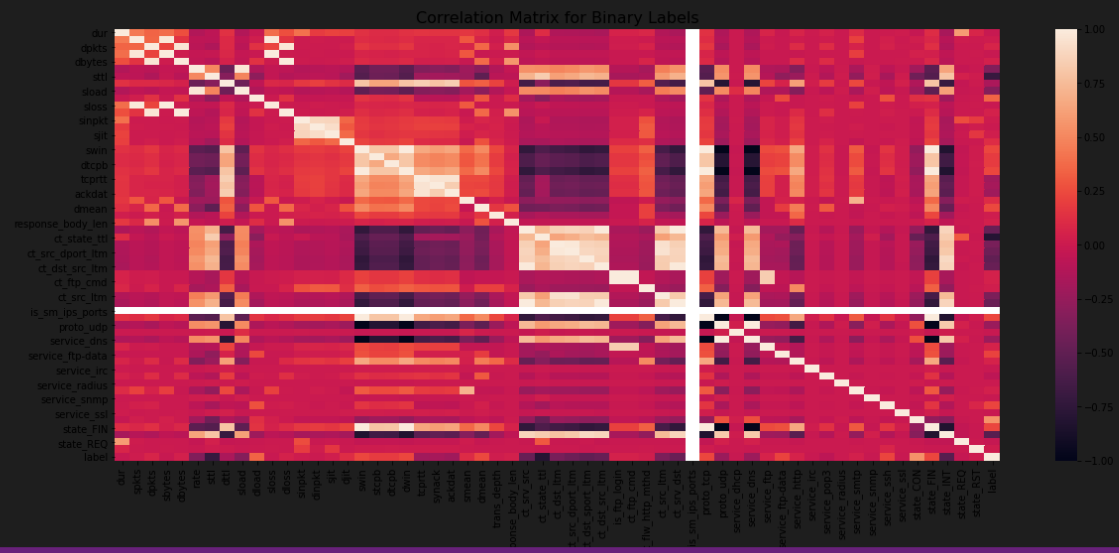
```
array(['Analysis', 'Backdoor', 'DoS', 'Exploits', 'Fuzzers', 'Generic',
       'Normal', 'Reconnaissance', 'Worms'], dtype=object)
```

```
np.save("le2_classes.npy",le2.classes_,allow_pickle=True)
```

```
num_col.append('label')
```

## Correlation Matrix for Binary Labels

```
# Correlation Matrix for Binary Labels
plt.figure(figsize=(20,8))
corr_bin = bin_data[num_col].corr()
sns.heatmap(corr_bin,vmax=1.0,annot=False)
plt.title('Correlation Matrix for Binary Labels',fontsize=16)
plt.savefig('plots/correlation_matrix_bin.png')
plt.show()
```

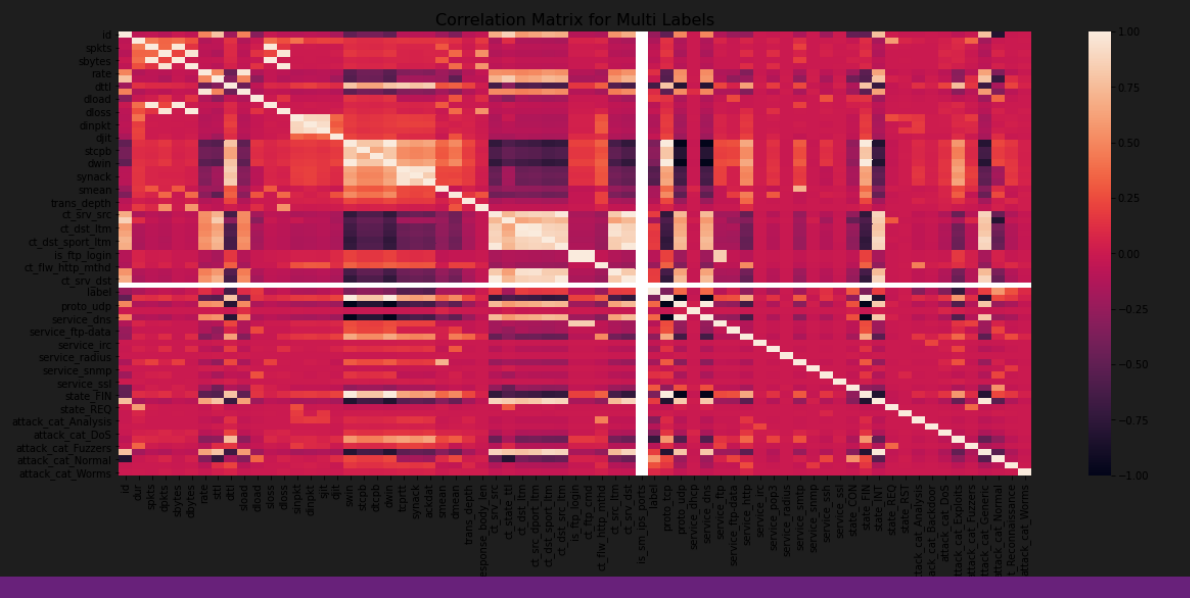


# Correlation Matrix for Multi-class Labels

```
num_col = list(multi_data.select_dtypes(include='number').columns)
```

```
# Correlation Matrix for Multi-class Labels
plt.figure(figsize=(20,8))
corr_multi = multi_data[num_col].corr()
sns.heatmap(corr_multi,vmax=1.0,annot=False)
plt.title('Correlation Matrix for Multi Labels',fontsize=16)
plt.savefig('plots/correlation_matrix_multi.png')
plt.show()
```

...



# Feature Selection

## Binary Labels

```
# finding the attributes which have more than 0.3 correlation with encoded attack label attribute
corr_ybin = abs(corr_bin['label'])
highest_corr_bin = corr_ybin[corr_ybin >0.3]
highest_corr_bin.sort_values(ascending=True)
```

[54]

```
sload               0.334562
dload               0.343910
rate                0.344535
ct_src_ltm          0.368486
ct_dst_ltm          0.387358
ct_src_dport_ltm    0.444874
ct_srv_dst          0.459984
ct_srv_src          0.463153
ct_dst_src_ltm      0.463735
ct_dst_sport_ltm    0.497234
state_INT           0.546631
state_CON           0.552505
sttl                0.707337
ct_state_ttl        0.801403
label               1.000000
Name: label, dtype: float64
```

```
# selecting attributes found by using pearson correlation coefficient
bin_cols = highest_corr_bin.index
bin_cols
```

[55]

```
Index(['rate', 'sttl', 'sload', 'dload', 'ct_srv_src', 'ct_state_ttl',
       'ct_dst_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm',
       'ct_src_ltm', 'ct_srv_dst', 'state_CON', 'state_INT', 'label'],
      dtype='object')
```

```
# Binary labelled Dataset
bin_data = bin_data[bin_cols].copy()
```

| | rate | sttl | sload | dload | ct_srv_src | ct_state_ttl | ct_dst_ltm | ct_src_dport_ltm | ct_dst_sport_ltm | ct_dst_src_ltm | ct_src_ltm | ct_srv_dst | state_CON | state_INT | label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 0.000014 | 0.138393 | 0.000001 | 0.000150 | 0.000000 | 0.166667 | 0.02 | 0.00 | 0.000000 | 0.039216 | 0.016949 | 0.000000 | 0.0 | 0.0 | 1 |
| 11 | 0.000043 | 0.138393 | 0.000092 | 0.000364 | 0.019608 | 0.166667 | 0.00 | 0.00 | 0.000000 | 0.019608 | 0.000000 | 0.000000 | 0.0 | 0.0 | 1 |
| 15 | 0.500000 | 0.995536 | 0.119792 | 0.000000 | 0.000000 | 0.333333 | 0.00 | 0.00 | 0.000000 | 0.058824 | 0.016949 | 0.000000 | 0.0 | 1.0 | 1 |
| 17 | 0.000043 | 0.138393 | 0.000007 | 0.000869 | 0.039216 | 0.166667 | 0.00 | 0.00 | 0.000000 | 0.019608 | 0.000000 | 0.039216 | 0.0 | 0.0 | 1 |
| 21 | 0.000044 | 0.995536 | 0.000009 | 0.000236 | 0.098039 | 0.166667 | 0.00 | 0.00 | 0.000000 | 0.000000 | 0.016949 | 0.039216 | 0.0 | 0.0 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 175335 | 0.166667 | 0.995536 | 0.032986 | 0.000000 | 0.862745 | 0.333333 | 0.64 | 0.64 | 0.533333 | 0.862745 | 0.542373 | 0.862745 | 0.0 | 1.0 | 0 |
| 175336 | 0.111111 | 0.995536 | 0.021991 | 0.000000 | 0.450980 | 0.333333 | 0.46 | 0.46 | 0.400000 | 0.450980 | 0.389831 | 0.450980 | 0.0 | 1.0 | 0 |
| 175338 | 0.111111 | 0.995536 | 0.021991 | 0.000000 | 0.215686 | 0.333333 | 0.04 | 0.04 | 0.066667 | 0.235294 | 0.033898 | 0.215686 | 0.0 | 1.0 | 0 |
| 175339 | 0.111111 | 0.995536 | 0.021991 | 0.000000 | 0.568627 | 0.333333 | 0.58 | 0.58 | 0.433333 | 0.568627 | 0.491525 | 0.568627 | 0.0 | 1.0 | 0 |
| 175340 | 0.111111 | 0.995536 | 0.021991 | 0.000000 | 0.568627 | 0.333333 | 0.58 | 0.58 | 0.500000 | 0.568627 | 0.491525 | 0.568627 | 0.0 | 1.0 | 0 |

81173 rows × 15 columns

### Saving Prepared Dataset to Disk

[57]
```
bin_data.to_csv('./datasets/bin_data.csv')
```

## Multi-class Labels

[58]
```
# finding the attributes which have more than 0.3 correlation with encoded attack label attribute
corr_ymulti = abs(corr_multi['label'])
highest_corr_multi = corr_ymulti[corr_ymulti >0.3]
highest_corr_multi.sort_values(ascending=True)
```

```
state_CON              0.302853
attack_cat_Analysis    0.326209
attack_cat_DoS         0.339669
state_FIN              0.361750
swin                   0.364393
dwin                   0.364393
```

+ Code + Markdown | Outline ···

```
proto_tcp              0.364393
proto_udp              0.364393
service_dns            0.365346
synack                 0.524027
ackdat                 0.570098
tcprtt                 0.570205
attack_cat_Normal      0.570858
dttl                   0.646589
attack_cat_Exploits    0.719733
label                  1.000000
Name: label, dtype: float64
```

```
# selecting attributes found by using pearson correlation coefficient
multi_cols = highest_corr_multi.index
multi_cols
```

[59]

```
Index(['dttl', 'swin', 'dwin', 'tcprtt', 'synack', 'ackdat', 'label',
       'proto_tcp', 'proto_udp', 'service_dns', 'state_CON', 'state_FIN',
       'attack_cat_Analysis', 'attack_cat_DoS', 'attack_cat_Exploits',
       'attack_cat_Normal'],
      dtype='object')
```

```
# Multi-class labelled Dataset
multi_data = multi_data[multi_cols].copy()
```

[60]

## Saving Prepared Dataset to Disk

```
multi_data.to_csv('./datasets/multi_data.csv')
```

[61]

# BINARY CLASSIFICATION

## Data Splitting

```
X = bin_data.drop(columns=['label'],axis=1)
Y = bin_data['label']
```

```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.20, random_state=50)
```

## Linear Regression

```
lr_bin = LinearRegression(normalize=False)
lr_bin.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
y_pred = lr_bin.predict(X_test)
```

```
round = lambda x:1 if x>0.6 else 0
vfunc = np.vectorize(round)
y_pred = vfunc(y_pred)
```

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
```

```
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.021927933477055742
Mean Squared Error -  0.021927933477055742
Root Mean Squared Error -  0.1480808342664767
R2 Score -  88.20923868071647
Accuracy -  97.80720665229443
```

```
cls_report= classification_report(y_true=y_test, y_pred=y_pred,target_names=le1.classes_)
```

```
print(cls_report)
```

```
              precision    recall  f1-score   support

    abnormal       0.97      1.00      0.99     12326
      normal       0.99      0.91      0.95      3909

    accuracy                           0.98     16235
   macro avg       0.98      0.96      0.97     16235
weighted avg       0.98      0.98      0.98     16235
```

## Real and Predicted Data

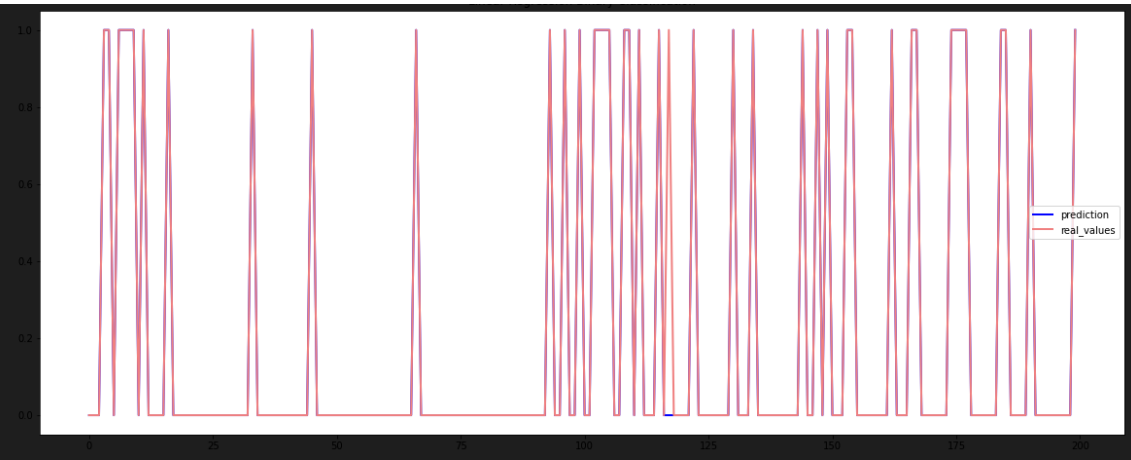
```
lr_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lr_bin_df.to_csv('./predictions/lr_real_pred_bin.csv')
lr_bin_df
```

| | Actual | Predicted |
|---|---|---|
| 159889 | 0 | 0 |
| 125506 | 0 | 0 |
| 158979 | 0 | 0 |
| 8148 | 1 | 1 |
| 39918 | 1 | 1 |
| ... | ... | ... |
| 68470 | 0 | 0 |
| 159674 | 0 | 0 |
| 171321 | 0 | 0 |
| 125777 | 0 | 0 |
| 39178 | 1 | 1 |

16235 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Linear Regression Binary Classification")
plt.savefig('plots/lr_real_pred_bin.png')
plt.show()
```




## Saving Trained Model to Disk

```
pkl_filename = "./models/linear_regressor_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(lr_bin, file)
  print("Saved model to disk")
else:
  print("Previous Model exists on the disk! Please Remove")
```

```
Saved model to disk
```

# Logistic Regression

```python
logr_bin = LogisticRegression(random_state=123, max_iter=5000)
logr_bin
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=5000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=123, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```python
logr_bin.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=5000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=123, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```python
y_pred = logr_bin.predict(X_test)
```

+ Code + Markdown

```python
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.02198952879581152
Mean Squared Error -  0.02198952879581152
Root Mean Squared Error -  0.1482886671186019
R2 Score -  88.17947258428785
Accuracy -  97.80104712041884
```

```python
cls_report= classification_report(y_true=y_test, y_pred=y_pred,target_names=le1.classes_)
print(cls_report)
```

```
              precision    recall  f1-score   support

    abnormal       0.97      1.00      0.99     12326
      normal       0.99      0.91      0.95      3909

    accuracy                           0.98     16235
   macro avg       0.98      0.96      0.97     16235
weighted avg       0.98      0.98      0.98     16235
```

## Real and Predicted Data

```python
logr_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
logr_bin_df.to_csv('./predictions/logr_real_pred_bin.csv')
logr_bin_df
```

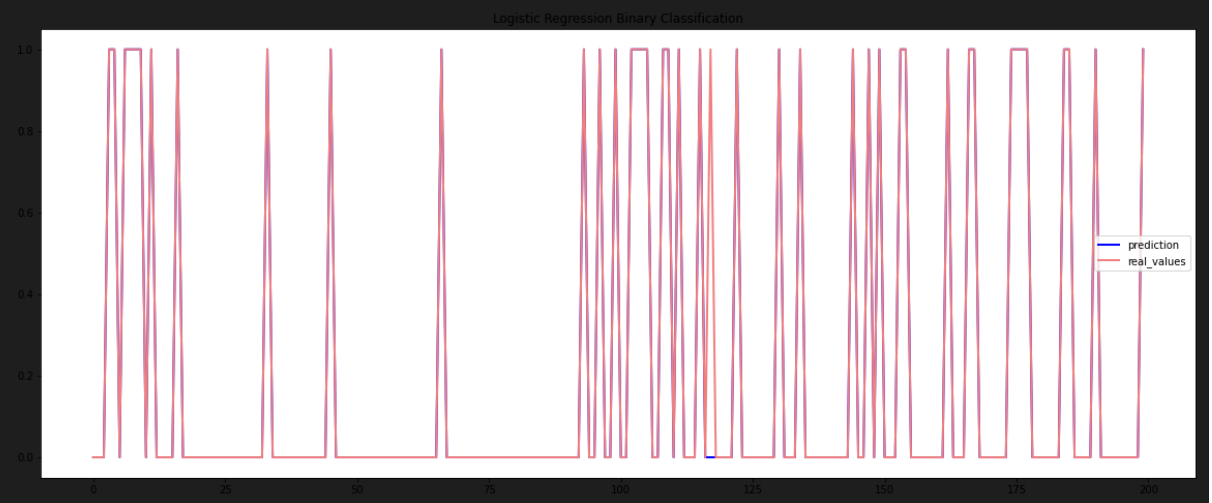
| | Actual | Predicted |
|---|---|---|
| 159889 | 0 | 0 |
| 125506 | 0 | 0 |
| 158979 | 0 | 0 |
| 8148 | 1 | 1 |
| 39918 | 1 | 1 |
| ... | ... | ... |
| 68470 | 0 | 0 |
| 159674 | 0 | 0 |
| 171321 | 0 | 0 |
| 125777 | 0 | 0 |
| 39178 | 1 | 1 |

16235 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Logistic Regression Binary Classification")
plt.savefig('plots/logr_real_pred_bin.png')
plt.show()
```

...



## Saving Trained Model to Disk

```
pkl_filename = "./models/logistic_regressor_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(logr_bin, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

```
Saved model to disk
```

# Linear Support Vector Machine

```
lsvm_bin = SVC(kernel='linear',gamma='auto')
lsvm_bin.fit(X_train,y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred = lsvm_bin.predict(X_test)
```

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.021496766245765322
Mean Squared Error -  0.021496766245765322
Root Mean Squared Error -  0.1466177555610688
R2 Score -  88.45167193436498
Accuracy -  97.85032337542347
```

```python
cls_report= classification_report(y_true=y_test, y_pred=y_pred,target_names=le1.classes_)
print(cls_report)
```

```
              precision    recall  f1-score   support

    abnormal       0.97      1.00      0.99     12326
      normal       1.00      0.91      0.95      3909

    accuracy                           0.98     16235
   macro avg       0.98      0.96      0.97     16235
weighted avg       0.98      0.98      0.98     16235
```

## Real and Predicted Data

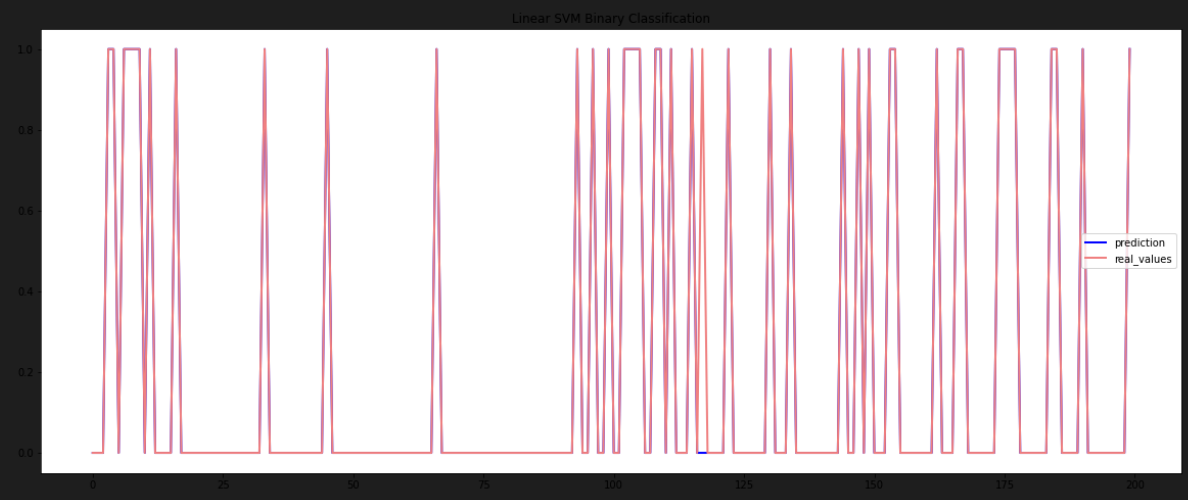
```python
lsvm_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lsvm_bin_df.to_csv('./predictions/lsvm_real_pred_bin.csv')
lsvm_bin_df
```

| | Actual | Predicted |
|---|---|---|
| 159889 | 0 | 0 |
| 125506 | 0 | 0 |
| 158979 | 0 | 0 |
| 8148 | 1 | 1 |
| 39918 | 1 | 1 |
| ... | ... | ... |
| 68470 | 0 | 0 |
| 159674 | 0 | 0 |
| 171321 | 0 | 0 |

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Linear SVM Binary Classification")
plt.savefig('plots/lsvm_real_pred_bin.png')
plt.show()
```

...



## Saving Trained Model to Disk

```
pkl_filename = "./models/lsvm_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(lsvm_bin, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

```
Saved model to disk
```

# K Nearest Neighbor Classifier

```
knn_bin=KNeighborsClassifier(n_neighbors=5)
knn_bin.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
y_pred = knn_bin.predict(X_test)
```

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.016938712657838004
Mean Squared Error -  0.016938712657838004
Root Mean Squared Error -  0.13014880966738807
R2 Score -  90.74435871039374
Accuracy -  98.3061287342162
```

```
cls_report= classification_report(y_true=y_test, y_pred=y_pred,target_names=le1.classes_)
print(cls_report)
```

```
              precision    recall  f1-score   support

    abnormal       0.99      0.99      0.99     12326
      normal       0.97      0.96      0.96      3909

    accuracy                           0.98     16235
   macro avg       0.98      0.97      0.98     16235
weighted avg       0.98      0.98      0.98     16235
```

## Real and Predicted Data

```
knn_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
knn_bin_df.to_csv('./predictions/knn_real_pred_bin.csv')
knn_bin_df
```

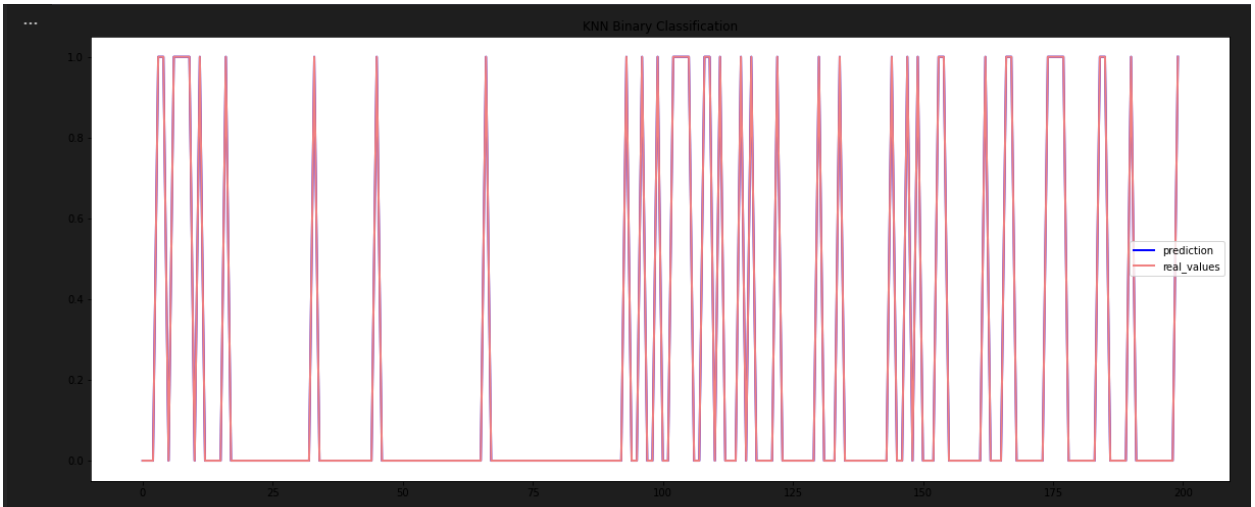
...

| | Actual | Predicted |
|---|---|---|
| 159889 | 0 | 0 |
| 125506 | 0 | 0 |
| 158979 | 0 | 0 |
| 8148 | 1 | 1 |
| 39918 | 1 | 1 |
| ... | ... | ... |
| 68470 | 0 | 0 |
| 159674 | 0 | 0 |
| 171321 | 0 | 0 |
| 125777 | 0 | 0 |
| 39178 | 1 | 1 |

16235 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("KNN Binary Classification")
plt.savefig('plots/knn_real_pred_bin.png')
plt.show()
```




## Saving Trained Model to Disk

```
pkl_filename = "./models/knn_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(knn_bin, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

```
Saved model to disk
```

# Random Forest Classifier

```python
rf_bin = RandomForestClassifier(random_state=123)
rf_bin.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=123,
                       verbose=0, warm_start=False)
```

```python
y_pred = rf_bin.predict(X_test)
```

```python
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.013550970126270403
Mean Squared Error -  0.013550970126270403
Root Mean Squared Error -  0.1164086342427846
R2 Score -  92.59509512345335
Accuracy -  98.64490298737296
```

```python
cls_report= classification_report(y_true=y_test, y_pred=y_pred,target_names=le1.classes_)
print(cls_report)
```

```
...                precision    recall  f1-score   support

      abnormal       0.99      0.99      0.99     12326
        normal       0.98      0.96      0.97      3909

      accuracy                           0.99     16235
     macro avg       0.98      0.98      0.98     16235
  weighted avg       0.99      0.99      0.99     16235
```

## Real and Predicted Data

```python
rf_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
rf_bin_df.to_csv('./predictions/rf_real_pred_bin.csv')
rf_bin_df
```

...

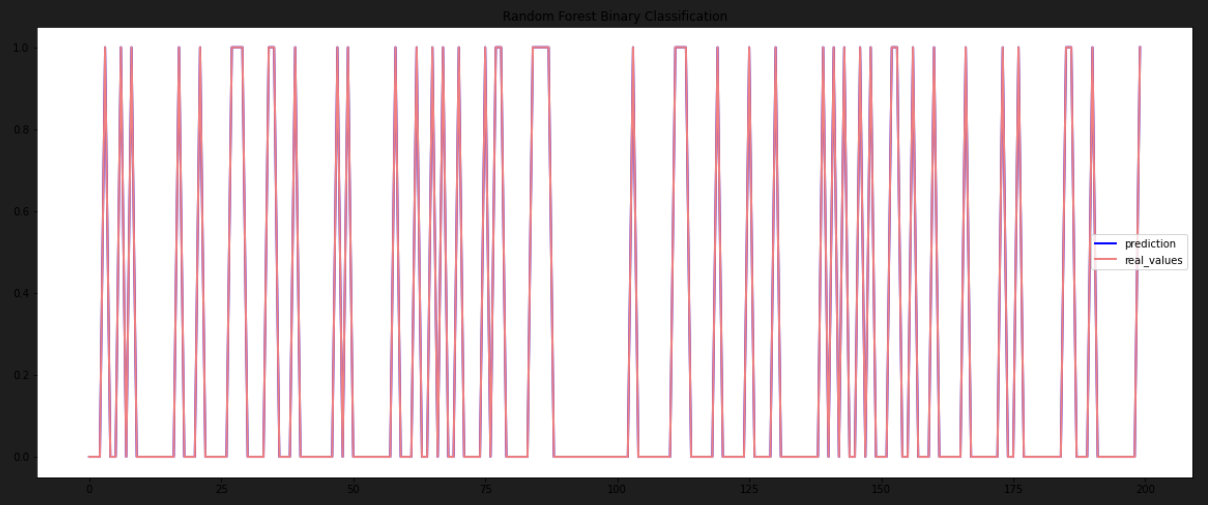
| | Actual | Predicted |
|---|---|---|
| 159889 | 0 | 0 |
| 125506 | 0 | 0 |
| 158979 | 0 | 0 |
| 8148 | 1 | 1 |
| 39918 | 1 | 1 |
| ... | ... | ... |
| 68470 | 0 | 0 |
| 159674 | 0 | 0 |
| 171321 | 0 | 0 |
| 125777 | 0 | 0 |
| 39178 | 1 | 1 |

16235 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[200:400], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[200:400].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Random Forest Binary Classification")
plt.savefig('plots/rf_real_pred_bin.png')
plt.show()
```

...




## Saving Trained Model to Disk

```python
pkl_filename = "./models/random_forest_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(rf_bin, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

```
Saved model to disk
```

# Decision Tree Classifier

```python
dt_bin = DecisionTreeClassifier(random_state=123)
dt_bin.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=123, splitter='best')
```

```python
y_pred = dt_bin.predict(X_test)
```

```python
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.019094548814290114
Mean Squared Error -  0.019094548814290114
Root Mean Squared Error -  0.13818302650575473
R2 Score -  89.55757103838098
Accuracy -  98.09054511857099
```

```python
cls_report= classification_report(y_true=y_test, y_pred=y_pred,target_names=le1.classes_)
print(cls_report)
```

```
              precision    recall  f1-score   support

    abnormal       0.99      0.99      0.99     12326
      normal       0.96      0.97      0.96      3909

    accuracy                           0.98     16235
   macro avg       0.97      0.98      0.97     16235
weighted avg       0.98      0.98      0.98     16235
```

## Real and Predicted Data

```python
dt_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
dt_bin_df.to_csv('./predictions/dt_real_pred_bin.csv')
dt_bin_df
```

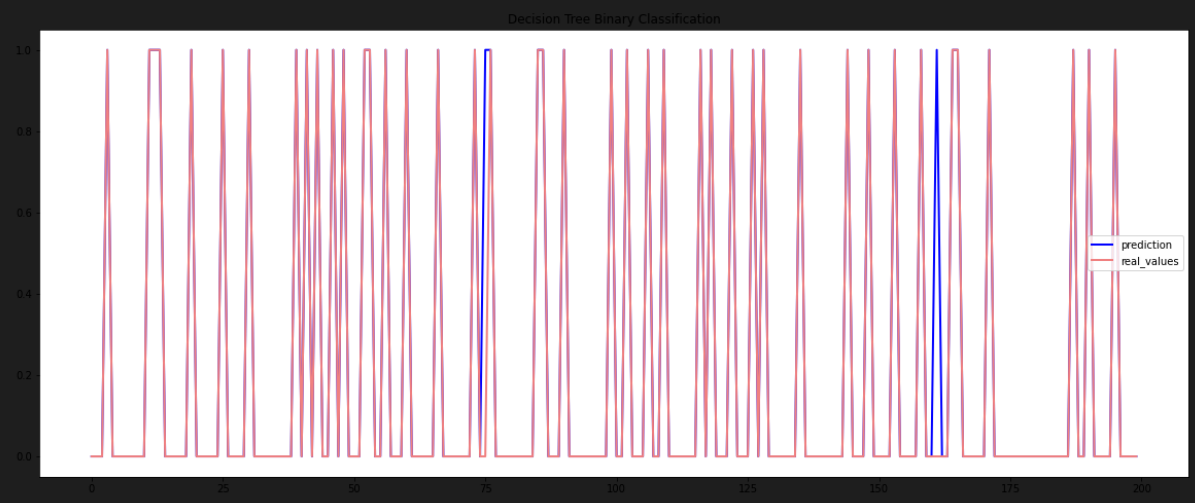
| | Actual | Predicted |
|---|---|---|
| 159889 | 0 | 0 |
| 125506 | 0 | 0 |
| 158979 | 0 | 0 |
| 8148 | 1 | 1 |
| 39918 | 1 | 1 |
| ... | ... | ... |
| 68470 | 0 | 0 |
| 159674 | 0 | 0 |
| 171321 | 0 | 0 |
| 125777 | 0 | 0 |
| 39178 | 1 | 1 |

16235 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[300:500], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[300:500].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Decision Tree Binary Classification")
plt.savefig('plots/dt_real_pred_bin.png')
plt.show()
```

...




## Saving Trained Model to Disk

```python
pkl_filename = "./models/decision_tree_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(dt_bin, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

```
Saved model to disk
```

# Multi Layer Perceptron

```python
mlp_bin = MLPClassifier(random_state=123, solver='adam', max_iter=8000)
```

```python
mlp_bin.fit(X_train,y_train)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=8000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=123, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

```python
y_pred = mlp_bin.predict(X_test)
```

```python
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.01632275947028026
Mean Squared Error -  0.01632275947028026
Root Mean Squared Error -  0.12776055522061674
R2 Score -  91.10646238100463
Accuracy -  98.36772405297197
```

```python
cls_report= classification_report(y_true=y_test, y_pred=y_pred,target_names=le1.classes_)
print(cls_report)
```

```
              precision    recall  f1-score   support

    abnormal       0.98      0.99      0.99     12326
      normal       0.98      0.95      0.97      3909

    accuracy                           0.98     16235
   macro avg       0.98      0.97      0.98     16235
weighted avg       0.98      0.98      0.98     16235
```

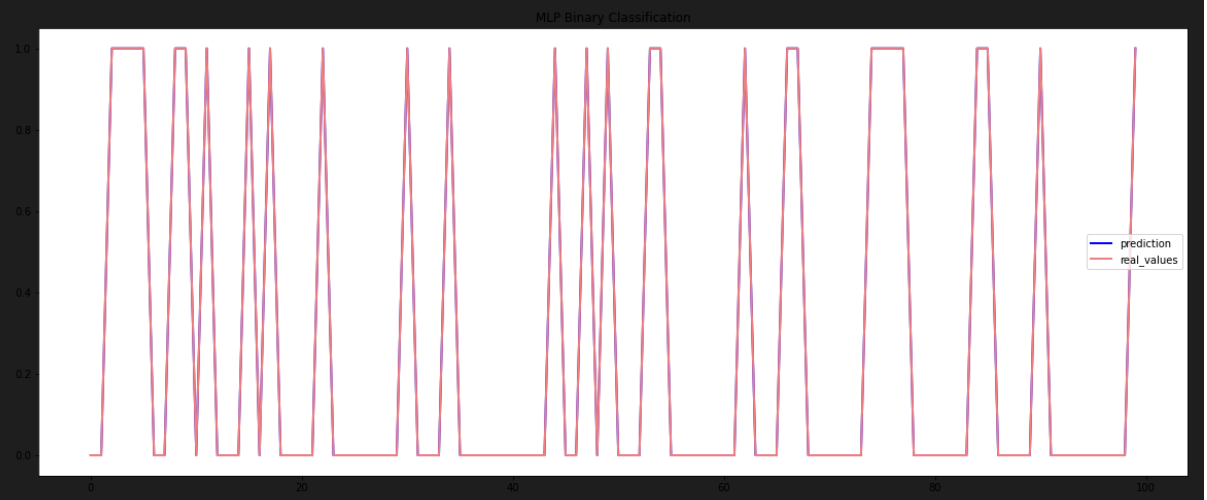
## Real and Predicted Data

```python
mlp_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
mlp_bin_df.to_csv('./predictions/mlp_real_pred_bin.csv')
mlp_bin_df
```

| | Actual | Predicted |
|---|---|---|
| 159889 | 0 | 0 |
| 125506 | 0 | 0 |
| 158979 | 0 | 0 |
| 8148 | 1 | 1 |
| 39918 | 1 | 1 |
| ... | ... | ... |
| 68470 | 0 | 0 |
| 159674 | 0 | 0 |
| 171321 | 0 | 0 |
| 125777 | 0 | 0 |

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[100:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[100:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("MLP Binary Classification")
plt.savefig('plots/mlp_real_pred_bin.png')
plt.show()
```




## Saving Trained Model to Disk

# Saving Trained Model to Disk

```
pkl_filename = "./models/mlp_binary.pkl"
if (not path.isfile(pkl_filename)):
    # saving the trained model to disk
    with open(pkl_filename, 'wb') as file:
        pickle.dump(mlp_bin, file)
    print("Saved model to disk")
else:
    print("Model already saved")
```

```
Saved model to disk
```

# MULTI-CLASS CLASSIFICATION

## Data Splitting

```
X = multi_data.drop(columns=['label'],axis=1)
Y = multi_data['label']
```
[62]

```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.30, random_state=100)
```
[63]

## Linear Regression

```
lr_multi = LinearRegression(normalize=False)
lr_multi.fit(X_train, y_train)
```
[64]

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
y_pred = lr_multi.predict(X_test)
```
[65]

```
for i in range(len(y_pred)):
  y_pred[i] = int(round(y_pred[i]))
```
[66]

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

```
Mean Absolute Error -  0.06824901445466491
Mean Squared Error -  0.12146846254927726
Root Mean Squared Error -  0.3485232596962178
R2 Score -  91.82055676180129
Accuracy -  95.12976346911958
```

```
print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

[68]

```
                  precision    recall  f1-score   support

        Analysis       1.00      1.00      1.00       166
        Backdoor       0.00      0.00      0.00        32
             DoS       0.95      0.96      0.96       521
        Exploits       1.00      0.99      1.00      4900
         Fuzzers       0.00      0.00      0.00       508
         Generic       0.95      0.99      0.97     11839
          Normal       0.91      1.00      0.96      5855
Reconnaissance       0.00      0.00      0.00       502
           Worms       0.00      0.00      0.00        29

        accuracy                           0.95     24352
       macro avg       0.54      0.55      0.54     24352
    weighted avg       0.91      0.95      0.93     24352


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricW
  _warn_prf(average, modifier, msg_start, len(result))
```

## Real and Predicted Data

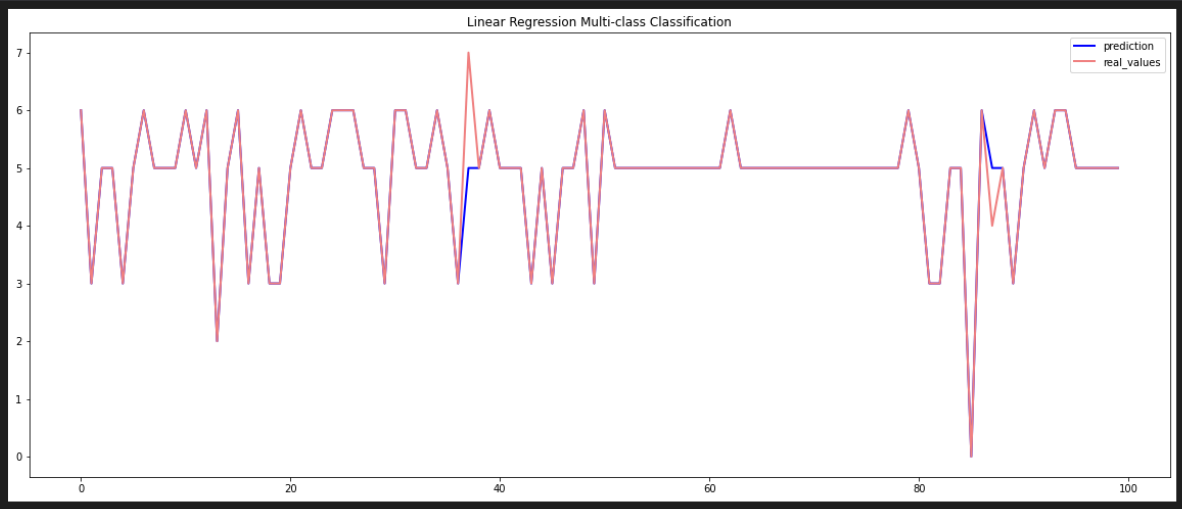
```
lr_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lr_multi_df.to_csv('./predictions/lr_real_pred_multi.csv')
lr_multi_df
```

[69]

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[100:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[100:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Linear Regression Multi-class Classification")
plt.savefig('plots/lr_real_pred_multi.png')
plt.show()
```

[70]



## Saving Trained Model to Disk

+ Code + Markdown

```
pkl_filename = "./models/linear_regressor_multi.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(lr_multi, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

[71]

```
Saved model to disk
```

# Logistic Regression

```
logr_multi = LogisticRegression(random_state=123, max_iter=5000,solver='newton-cg',multi_class='multinomial')
logr_multi.fit(X_train,y_train)
```

[72]

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=5000,
                   multi_class='multinomial', n_jobs=None, penalty='l2',
                   random_state=123, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_pred = logr_multi.predict(X_test)
```

[73]

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

[74]

```
Mean Absolute Error -  0.060077201051248356
Mean Squared Error -  0.18056011826544022
Root Mean Squared Error -  0.42492366169165047
R2 Score -  87.87674567880146
Accuracy -  97.58952036793693
```

```
print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

[75]

```
                 precision    recall  f1-score   support

       Analysis       1.00      1.00      1.00       166
       Backdoor       0.00      0.00      0.00        32
            DoS       1.00      1.00      1.00       521
       Exploits       1.00      1.00      1.00      4900
        Fuzzers       0.56      0.42      0.48       508
        Generic       0.99      0.99      0.99     11839
         Normal       1.00      1.00      1.00      5855
Reconnaissance       0.54      0.76      0.63       502
          Worms       0.00      0.00      0.00        29

       accuracy                           0.98     24352
      macro avg       0.68      0.69      0.68     24352
   weighted avg       0.97      0.98      0.97     24352


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defin
  _warn_prf(average, modifier, msg_start, len(result))
```

## Real and Predicted Data

```
logr_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
logr_multi_df.to_csv('./predictions/logr_real_pred_multi.csv')
logr_multi_df
```

[76]

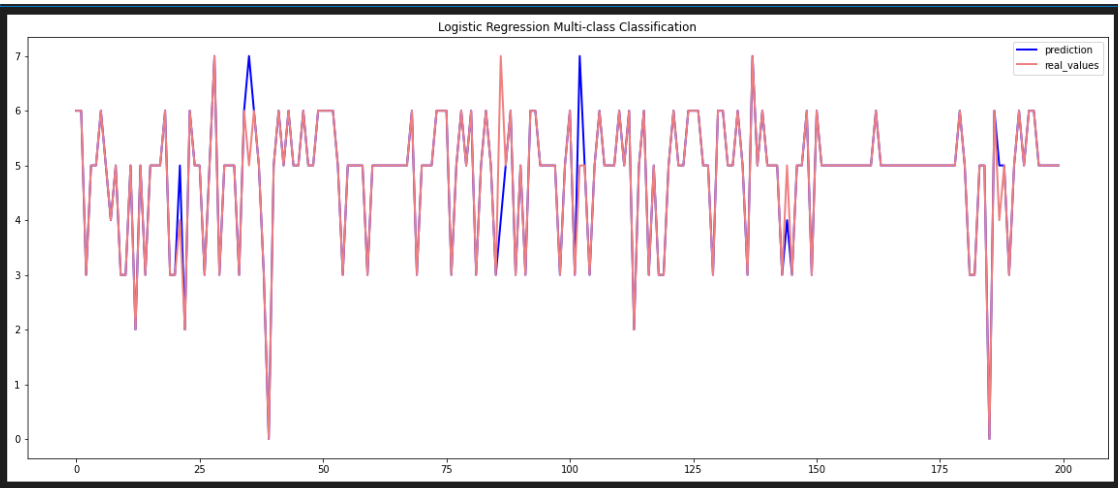
| | Actual | Predicted |
|---|---|---|
| 22650 | 6 | 6 |
| 27815 | 6 | 6 |
| 49033 | 3 | 3 |
| 156811 | 5 | 5 |
| 136878 | 5 | 5 |
| ... | ... | ... |
| 94950 | 4 | 5 |
| 148684 | 5 | 5 |
| 12788 | 6 | 6 |
| 81439 | 3 | 3 |
| 26144 | 6 | 6 |

24352 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Logistic Regression Multi-class Classification")
plt.savefig('plots/logr_real_pred_multi.png')
plt.show()
```

[77]




## Saving Trained Model to Disk

```
pkl_filename = "./models/logistic_regressor_multi.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(logr_multi, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

[78]

```
Saved model to disk
```

# Linear Support Vector Machine

```
lsvm_multi = SVC(kernel='linear',gamma='auto')
lsvm_multi.fit(X_train,y_train)
```

[79]

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred = lsvm_multi.predict(X_test)
```

[80]

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

[81]

```
Mean Absolute Error -  0.059912943495400786
Mean Squared Error -  0.17941031537450722
Root Mean Squared Error -  0.42356854861345317
R2 Score -  87.93449282205455
Accuracy -  97.59362680683311
```

```
print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

[82]

```
                 precision    recall  f1-score   support

       Analysis       1.00      1.00      1.00       166
       Backdoor       0.00      0.00      0.00        32
            DoS       1.00      1.00      1.00       521
       Exploits       1.00      1.00      1.00      4900
        Fuzzers       0.54      0.47      0.50       508
        Generic       0.99      0.99      0.99     11839
         Normal       1.00      1.00      1.00      5855
Reconnaissance       0.56      0.71      0.62       502
          Worms       0.00      0.00      0.00        29
```

```
       accuracy                           0.98     24352
      macro avg       0.68      0.69      0.68     24352
   weighted avg       0.97      0.98      0.98     24352


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetric
  _warn_prf(average, modifier, msg_start, len(result))
```

## Real and Predicted Data

[83]

```
lsvm_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lsvm_multi_df.to_csv('./predictions/lsvm_real_pred_multi.csv')
lsvm_multi_df
```

...

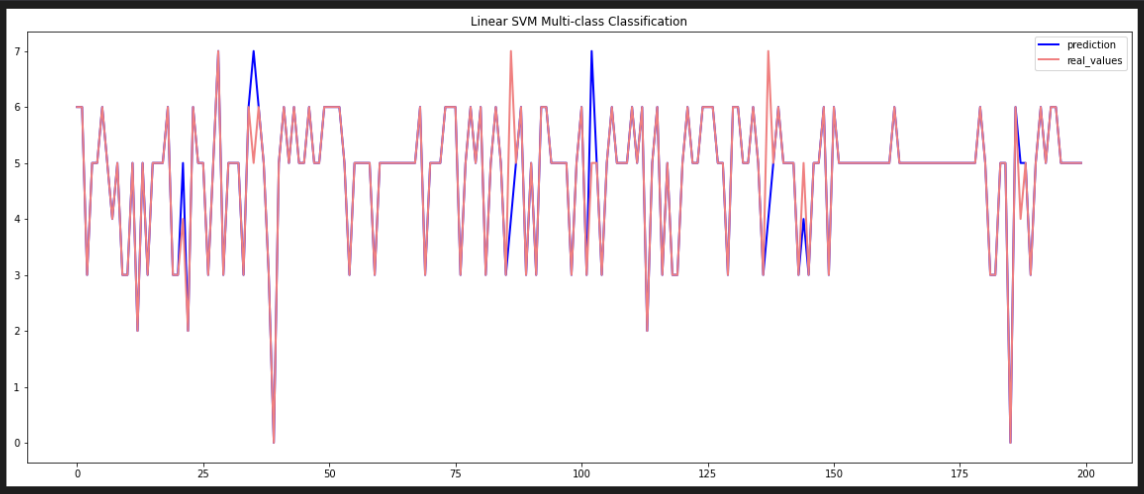
| | Actual | Predicted |
|---|---|---|
| 22650 | 6 | 6 |
| 27815 | 6 | 6 |
| 49033 | 3 | 3 |
| 156811 | 5 | 5 |
| 136878 | 5 | 5 |
| ... | ... | ... |
| 94950 | 4 | 5 |
| 148684 | 5 | 5 |
| 12788 | 6 | 6 |
| 81439 | 3 | 3 |
| 26144 | 6 | 6 |

24352 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Linear SVM Multi-class Classification")
plt.savefig('plots/lsvm_real_pred_multi.png')
plt.show()
```

[84]



## Saving Trained Model to Disk

```
pkl_filename = "./models/lsvm_multi.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(lsvm_multi, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

[85]

```
Saved model to disk
```

# K Nearest Neighbor Classifier

```
knn_multi = KNeighborsClassifier(n_neighbors=5)
knn_multi.fit(X_train,y_train)
```

[86]

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
y_pred = knn_multi.predict(X_test)
```

[87]

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

[88]

```
Mean Absolute Error -  0.06508705650459921
Mean Squared Error -  0.19411136662286466
Root Mean Squared Error -  0.44058071521897624
R2 Score -  86.92848100772136
Accuracy -  97.36777266754271
```

```python
print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

[89]

```
                 precision    recall  f1-score   support

       Analysis       1.00      1.00      1.00       166
       Backdoor       0.00      0.00      0.00        32
            DoS       1.00      1.00      1.00       521
       Exploits       1.00      1.00      1.00      4900
        Fuzzers       0.48      0.52      0.50       508
        Generic       0.99      0.99      0.99     11839
         Normal       1.00      1.00      1.00      5855
Reconnaissance       0.55      0.54      0.55       502
          Worms       0.00      0.00      0.00        29

       accuracy                           0.97     24352
      macro avg       0.67      0.67      0.67     24352
   weighted avg       0.97      0.97      0.97     24352
```

## Real and Predicted Data

```python
knn_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
knn_multi_df.to_csv('./predictions/knn_real_pred_multi.csv')
knn_multi_df
```

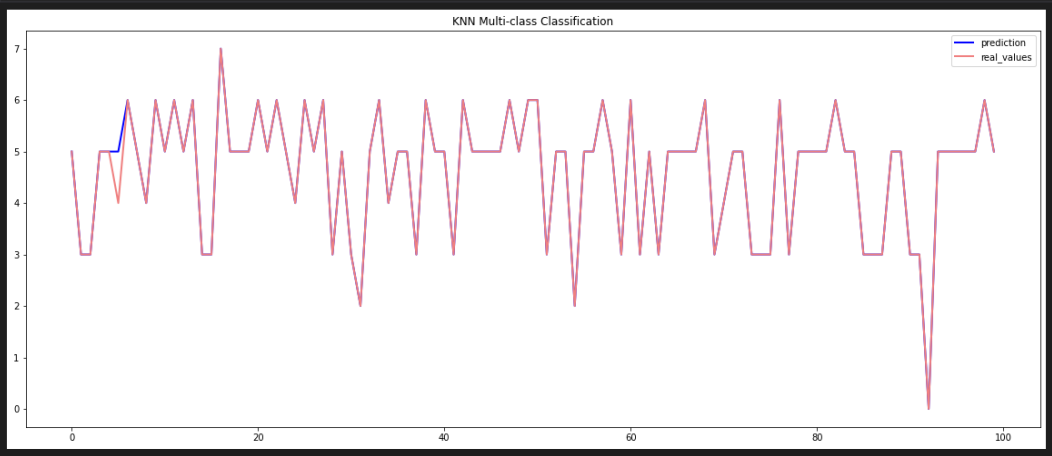
[90]

| | Actual | Predicted |
|---|---|---|
| 22650 | 6 | 6 |
| 27815 | 6 | 6 |
| 49033 | 3 | 3 |
| 156811 | 5 | 5 |
| 136878 | 5 | 5 |
| ... | ... | ... |
| 94950 | 4 | 5 |
| 148684 | 5 | 5 |
| 12788 | 6 | 6 |
| 81439 | 3 | 3 |
| 26144 | 6 | 6 |

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[400:500], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[400:500].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("KNN Multi-class Classification")
plt.savefig('plots/knn_real_pred_multi.png')
plt.show()
```

[93]



## Saving Trained Model to Disk

```
pkl_filename = "./models/knn_multi.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(knn_multi, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

[94]

```
Model already saved
```

# Random Forest Classifier

```
rf_multi = RandomForestClassifier(random_state=50)
rf_multi.fit(X_train,y_train)
```

[95]

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=50, verbose=0,
                       warm_start=False)
```

```
y_pred = rf_multi.predict(X_test)
```

[96]

```
xprint("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

[97]

```
Mean Absolute Error -  0.06611366622864652
Mean Squared Error -  0.1985052562417871
Root Mean Squared Error -  0.4455392869790352
R2 Score -  86.6379909424011
Accuracy -  97.3184954007884 4
```

[99]

```
print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

```
                 precision    recall  f1-score   support

       Analysis       1.00      1.00      1.00       166
       Backdoor       0.08      0.03      0.05        32
            DoS       1.00      1.00      1.00       521
       Exploits       1.00      1.00      1.00      4900
        Fuzzers       0.49      0.43      0.46       508
        Generic       0.99      0.99      0.99     11839
         Normal       1.00      1.00      1.00      5855
Reconnaissance       0.54      0.60      0.57       502
          Worms       0.12      0.07      0.09        29

       accuracy                           0.97     24352
      macro avg       0.69      0.68      0.68     24352
   weighted avg       0.97      0.97      0.97     24352
```

## Real and Predicted Data

[100]

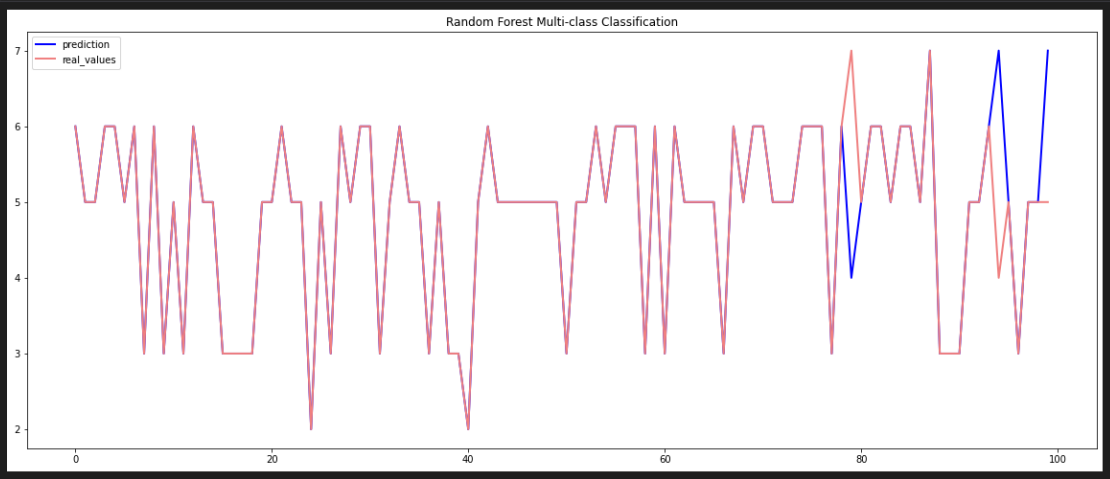
```
rf_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
rf_multi_df.to_csv('./predictions/rf_real_pred_multi.csv')
rf_multi_df
```

| | Actual | Predicted |
|---|---|---|
| 22650 | 6 | 6 |
| 27815 | 6 | 6 |
| 49033 | 3 | 3 |
| 156811 | 5 | 5 |
| 136878 | 5 | 5 |

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[500:600], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[500:600].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Random Forest Multi-class Classification")
plt.savefig('plots/rf_real_pred_multi.png')
plt.show()
```

[101]



## Saving Trained Model to Disk

```
pkl_filename = "./models/random_forest_multi.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(rf_multi, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

[102]

```
Saved model to disk
```

# Decision Tree Classifier

```
dt_multi = DecisionTreeClassifier(random_state=123)
dt_multi.fit(X_train,y_train)
```

[103]

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=123, splitter='best')
```

```
y_pred = dt_multi.predict(X_test)
```

[104]

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

[105]

```
Mean Absolute Error -  0.06800262812089355
Mean Squared Error -  0.20532194480946123
Root Mean Squared Error -  0.4531246459965086
R2 Score -  86.17743099336013
Accuracy -  97.19940867279895
```

```python
print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

[106]

```
                  precision    recall  f1-score   support

        Analysis       1.00      1.00      1.00       166
        Backdoor       0.08      0.06      0.07        32
             DoS       1.00      1.00      1.00       521
        Exploits       1.00      1.00      1.00      4900
         Fuzzers       0.50      0.39      0.44       508
         Generic       0.98      0.99      0.99     11839
          Normal       1.00      1.00      1.00      5855
Reconnaissance       0.54      0.56      0.55       502
          Worms       0.05      0.07      0.06        29

        accuracy                           0.97     24352
       macro avg       0.68      0.67      0.68     24352
    weighted avg       0.97      0.97      0.97     24352
```

## Real and Predicted Data

```python
dt_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
dt_multi_df.to_csv('./predictions/dt_real_pred_multi.csv')
dt_multi_df
```

[107]

## Real and Predicted Data

```
dt_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
dt_multi_df.to_csv('./predictions/dt_real_pred_multi.csv')
dt_multi_df
```

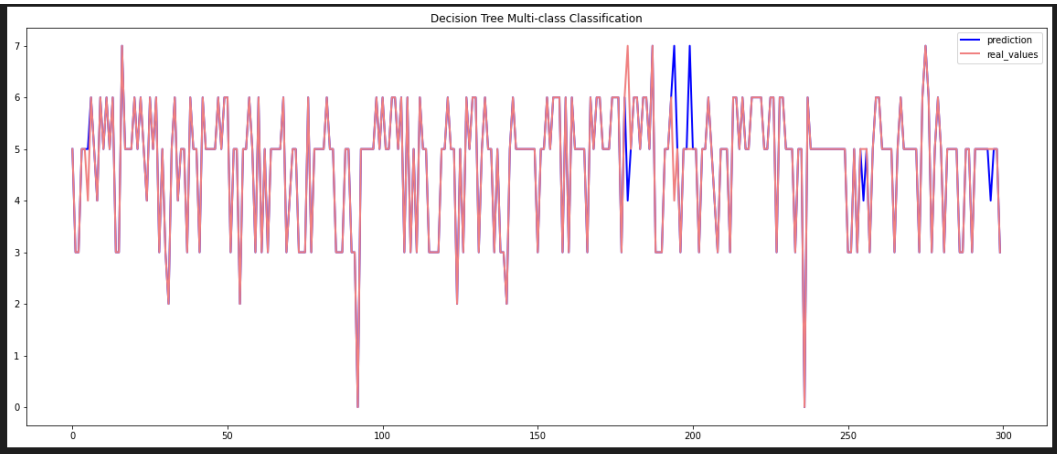
[107]

| | Actual | Predicted |
|---|---|---|
| 22650 | 6 | 6 |
| 27815 | 6 | 6 |
| 49033 | 3 | 3 |
| 156811 | 5 | 5 |
| 136878 | 5 | 5 |
| ... | ... | ... |
| 94950 | 4 | 5 |
| 148684 | 5 | 5 |
| 12788 | 6 | 6 |
| 81439 | 3 | 3 |
| 26144 | 6 | 6 |

24352 rows × 2 columns

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[400:700], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[400:700].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Decision Tree Multi-class Classification")
plt.savefig('plots/dt_real_pred_multi.png')
plt.show()
```




## Saving Trained Model to Disk

```
pkl_filename = "./models/decision_tree_multi.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(dt_multi, file)
  print("Saved model to disk")
else:
  print("Model already saved")
```

[110]

```
Saved model to disk
```

# Multi Layer Perceptron

```
mlp_multi = MLPClassifier(random_state=123, solver='adam', max_iter=8000)
mlp_multi.fit(X_train,y_train)
```

[111]

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=8000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=123, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

```
y_pred = mlp_multi.predict(X_test)
```

[112]

```
print("Mean Absolute Error - " , metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error - " , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error - " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score - " , metrics.explained_variance_score(y_test, y_pred)*100)
print("Accuracy - ",accuracy_score(y_test,y_pred)*100)
```

[113]

```
Mean Absolute Error -  0.06065210249671485
Mean Squared Error -  0.17858902759526937
Root Mean Squared Error -  0.4225979502970517
R2 Score -  87.97913543550516
Accuracy -  97.54434954007884
```

```
print(classification_report(y_test, y_pred,target_names=le2.classes_))
```

[114]

```
...                  precision    recall  f1-score   support

          Analysis       1.00      1.00      1.00       166
          Backdoor       0.00      0.00      0.00        32
               DoS       1.00      1.00      1.00       521
          Exploits       1.00      1.00      1.00      4900
           Fuzzers       0.50      0.63      0.55       508
           Generic       0.99      0.99      0.99     11839
            Normal       1.00      1.00      1.00      5855
    Reconnaissance       0.60      0.52      0.56       502
             Worms       0.00      0.00      0.00        29

          accuracy                           0.98     24352
         macro avg       0.68      0.68      0.68     24352
      weighted avg       0.97      0.98      0.97     24352


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
  _warn_prf(average, modifier, msg_start, len(result))
```

## Real and Predicted Data

```
mlp_multi_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
mlp_multi_df.to_csv('./predictions/mlp_real_pred_multi.csv')
mlp_multi_df
```

[115]

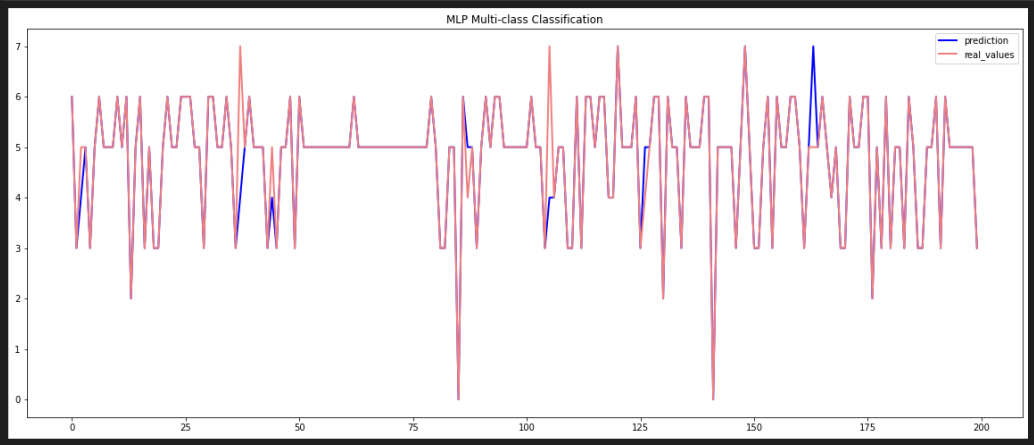
...

| | Actual | Predicted |
|---|---|---|
| 22650 | 6 | 6 |
| 27815 | 6 | 6 |
| 49033 | 3 | 3 |
| 156811 | 5 | 5 |
| 136878 | 5 | 5 |
| ... | ... | ... |
| 94950 | 4 | 5 |
| 148684 | 5 | 5 |
| 12788 | 6 | 6 |
| 81439 | 3 | 3 |
| 26144 | 6 | 6 |

## Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[100:300], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[100:300].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("MLP Multi-class Classification")
plt.savefig('plots/mlp_real_pred_multi.png')
plt.show()
```

[116]




## Saving Trained Model to Disk

# Saving Trained Model to Disk

```
pkl_filename = "./models/mlp_multi.pkl"
if (not path.isfile(pkl_filename)):
    # saving the trained model to disk
    with open(pkl_filename, 'wb') as file:
        pickle.dump(mlp_multi, file)
    print("Saved model to disk")
else:
    print("Model already saved")
```

[117]

```
Saved model to disk
```

**Conclusion**

As the number of IoT users, services, and applications increases, there is an urgent need for robust, lightweight security solutions suitable for use in IoT environments. Moreover, IoT networks are the foundation of intelligent environments. Security flaws in these IoT networks therefore directly affect the intelligent environments in which they are built.

Attacks such as DoS, DDoS, probing, and RPL attacks impact services and applications delivered in IoT-based intelligent environments. Therefore, security in IoT environments is a very serious issue. IDS is a possible solution to this problem. Our study and project provided an overview of an IDS developed for the IoT environment. We deal with recommendations for designing a robust and lightweight IDS.

And on performing various classification algorithms on our dataset we see that for binary class labels random forest gives the best accuracy of 99%, and for multi class labels SVM gives the best accuracy of 98%.

## References

https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-018-0123-6

https://arxiv.org/ftp/arxiv/papers/2012/2012.01174.pdf

https://cybersecurity.springeropen.com/articles/10.1186/s42400-021-00077-7

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9143513/

https://ieeexplore.ieee.org/document/9417815