

# **Network Denial of Service Detection**

## **A Capstone Project Report**

*Submitted in partial fulfillment of the  
requirement for the award of the  
Degree of*

**BACHELOR OF TECHNOLOGY  
IN  
COMPUTER SCIENCE AND ENGINEERING**

*by*

**Pathange Naga Ajay (20BCD7056)  
E. Varun Sai (20BCD7179)  
Shaik Mohammad Waseem Akram (20BCD7141)  
Yenuganti Varun Sai (20BCI7112)**

*Under the Guidance of*

**Dr. Sheela Jayachandran**




**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING  
VIT-AP UNIVERSITY  
AMARAVATI- 522237**


*DECEMEBR 2023*

### Certificate

This is to certify that the Capstone Project work titled "**Network Denial of Service Detection**" that is being submitted by **Shaik Mohammad Waseem Akram (20BCD7141)** is in partial fulfilment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

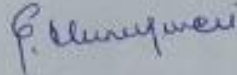
  
Dr. Sheela Jayachandran  
Guide

The thesis is satisfactory / unsatisfactory

  
Dr. Deepthi Godavarthi  
Internal Examiner 1

  
Dr. Y. Narasimha Rao  
Internal Examiner 2

Approved by

  
Dr. G. Muneeswari

HOD, Dept. of Data Science and Engineering  
School of Computer Science and Engineering

## **Acknowledgements**

We are grateful to all those with whom we have had the pleasure to work during this project. Our guide has provided us with extensive personal and professional guidance and taught us a great deal about both scientific research and life in general. We would like to thank Dr. Sheela Jayachandran, our professor-in-charge, for their support and guidance in completing our project on the above-given topic. It was a great learning experience. The project would not have been successful without our cooperation and input.

## Abstract

As IoT-enabled devices and mobile technology improvements became more common in our everyday lives, significant increases in wireless network traffic resulted in the generation of a large volume of high-dimensional network log data. Because of this, there are challenges facing the security of Wi-Fi network systems that must analyze such complex, large amounts of data for intrusion detection. Many Wi-Fi network systems employ machine learning-based intrusion detection systems (IDS). Such IDS usually employ supervised algorithms that extensively depend on human experts' observations to extract, select, and label features from training data for classification.

In order to improve prediction accuracy, we propose a supervised and unsupervised approach in this study that replaces human intervention and the manual labeling process with an automatic feature extraction and selection process. We do this by comparing different models of Classification (Binary and multiclass) and Regression of Machine Learning and to detect the three most common types of network attacks: Injection, Flooding, and Impersonate attacks. The NSL-KDD Dataset, which was recently gathered and includes actual evidence of several network attack types, is used in this study. The results of the trial showed how effective our feature extraction and selection strategy were. evaluating and comparing the performance of the selected characteristics and the accuracy of the intrusion detection of the three distinct assault techniques.

# Contents

Abstract.....	3
Chapter – 1 .....	6
Introduction .....	6
1.1 Objectives .....	8
1.2 Background and Literature Survey .....	8
1.2.1 Feature Extraction.....	8
1.2.2 Feature Selection .....	9
1.2.3 Logistic Regression .....	9
1.2.4 SVM (Support Vector Machine) .....	9
1.2.5 K-Means .....	9
1.2.6 Hierarchical Algorithm.....	9
1.3 Organization of Report .....	10
Chapter – 2 .....	11
A Machine Learning Based Intrusion Detection System for Automatic Feature Extraction and Selection .....	11
2.1 Proposed System .....	11
2.1.1 Working Methodology .....	11
2.1.1(a) Feature Extraction .....	11
2.1.1(b) Feature Selection.....	11
2.2 Standards .....	11
2.2.1 NSL-KDD Dataset.....	11
2.2.2 Support Vector Machine (SVM) .....	12
2.3 Software Details .....	14
2.3.1 Kaggle.....	14
2.3.2 GitHub .....	14
2.3.3 Collab .....	15
2.3.4 Libraries Used .....	16
Chapter – 3 .....	18
Results And Discussions .....	18
3.1 Data Exploration.....	18
3.2 Accuracy .....	21
Chapter – 4 .....	27
Conclusion And Future Work .....	27
Conclusion .....	27
Future Works .....	27

Chapter – 5 .....	29
Appendix .....	29
5.1 Logistic Regression .....	29
5.2 Support Vector Machine SVM.....	46
5.3 K-Means .....	53
5.4 Hierarchical Algorithm.....	55
Chapter – 6 .....	58
References .....	58

# Chapter – 1

## Introduction

People who are mobile and flexible often utilize wireless networks to connect to other locations quickly. However, in recent years, the adoption of IoT-based gadgets and other cutting-edge mobile technologies has increased wireless traffic. According to several sources, within the past four years, there has been a remarkable 13-fold rise in mobile data traffic. With the increasing prevalence of IOT and its integration into daily life, more active and passive assaults against wireless network systems have been conducted than in the past. As a result, there were issues with wireless network systems' security and privacy.

Three categories have been identified by Kollias et al. for the intrusion assaults against WEP (wired equivalent protection).

- **Injectable attacks:** generate a flood of smaller-sized, legitimately encrypted data frames, Injection attacks, which include SQL injections, were ranked as the third most dangerous online application security risk in 2021 by the Open online Application Security Project. In the applications they examined, injection occurred 274,000 times.

It is crucial to comprehend SQL injection attacks and their mechanisms in order to test for vulnerabilities, adhere to best practices, and think about purchasing software that actively prevents assaults.

- **Flooding attacks:** Flooding attacks are a type of denial-of-service (DoS) attack that attempts to overwhelm a target system with a large volume of traffic, making it unavailable to legitimate users. There are many different types of flooding attacks, but they all share the same goal of disrupting normal service.
- **Impersonation attacks:** add another AP to the local broadcasting system. Beacon frames promoting a legitimately established network beforehand. When unusual behavior is detected in network traffic, an intrusion detection system (IDS) informs users. It is software that looks for nefarious activity or policy infractions on a system or network. Any unlawful behavior or violation is frequently reported to an administrator or centrally documented utilizing a security information and event management (SIEM) system. A wireless network's security infrastructure is incomplete without an intrusion detection system.

IDSs are categorized into three categories based on the detection algorithms they utilize to find intrusions and attacks: rule-based, signature-based, and anomaly detection-based. In order to

characterize common behaviors, the rule-based IDS manually compiles a set of rules and constraints. According to Mitchell et al., any violation of the guidelines while there is network activity is recognized and reported as an attack. A signature-based intrusion detection system (IDS) may recognize any attack if any network activity corresponds to any known attack types, whose patterns or attributes have been previously observed and recorded. The signature-based intrusion detection system is good at identifying known attacks, but it is not very good at identifying new or unknown ones. An anomaly detection-based intrusion detection system (IDS) may identify malicious assaults and intrusions by first analyzing usual network traffic behavior and then identifying any outlier by detecting a departure from that trend. The advantage of this detection type is that it can recognize assaults that are known and unknown. Big data is produced not only on a massive scale but also in very complicated ways due to the broad variety of IoT devices that use Wi-Fi networks and their fast advancements. In order to determine the guidelines and limitations needed to capture attack patterns, processing such large amounts of data for intrusion detection necessitates the lengthy and agonizing involvement of humans. Scalability has become an issue in a machine learning-based intrusion detection system using supervised learning techniques when the amount of data grows quickly.

There are two main scalability challenges in the machine learning-based IDS with large amounts of high-dimensional data. There are two main scalability challenges in the machine learning-based IDS with large amounts of high-dimensional data. First off, most machine learning-based detection systems rely on supervised learning, in which training data is labeled by hand for various attack patterns and profiles of typical network traffic behavior. Secondly, relying solely on human intelligence is problematic since human errors are inevitable when reading vast amounts of complex, high-dimensional data. As data becomes more complex and highly dimensional, feature extraction and selection become more expensive, unpleasant, and usually wrong.

An unsupervised approach is used to feature extraction and selection in order to study the three primary wireless attacks: injection, flooding, and impersonation. The primary idea is to pick four machine learning algorithms for automated feature selection based on each algorithm's ability to provide the significance rate of each input feature as an output at the end of classification. We use an auto-encoder with two hidden layers for the automated feature extraction in our automatic feature extraction and selection (AFES) component, following Tanuwidjaja et al.'s methodology to extract the most important features.



## **1.1 Objectives**

Numerous Wi-Fi network systems include machine learning-based IDs, which mostly need human oversight throughout the feature extraction, selection, and labeling stages of training data preparation for classification. Our goal is to replace human supervision with an unsupervised method that uses an automated feature extraction and selection procedure.

## **1.2 Background and Literature Survey**

The challenges facing intrusion detection systems (IDS) primarily arise from the prevalent use of a supervised approach in many companies. The critical issue lies in the manual labeling of data by human operators, introducing vulnerabilities that can compromise the system's effectiveness in detecting attacks. The problems include subjectivity and inconsistency in human labeling, limitations in annotators' expertise, and the struggle to adapt to an evolving threat landscape. Incomplete and inaccurate labels, along with scalability challenges due to data volume and resource constraints, further contribute to the issues. Adversarial attacks targeting the weaknesses in manual labeling pose a significant threat. Continuous maintenance is essential to address the dynamic nature of networks and cyber threats. Resolving these challenges requires a holistic approach, incorporating automation, machine learning, and continuous learning mechanisms to enhance the accuracy and efficiency of intrusion detection systems.

The following are the four areas in which our work and theirs differ.

Improving impersonation attack detection is the paper's main objective. Ours The study focuses on identifying all attack classes rather than just impersonation assaults. Second, the article uses as one feature selection model. Since it manages outliers more effectively. Then, four weighted feature selection methods using popular machine learning classifiers— K-Means, Hierarchical Algorithm, Logistic Regression (LR) and support vector machine (SVM)—are used to the combined feature pool to choose features.

### **1.2.1 Feature Extraction**

We adopt a neural network-based stacked auto encoder method to extract features. An auto encoder is an unsupervised neural network model that applies backpropagation, setting the target values to be equal to the inputs.

### **1.2.2 Feature Selection**

We employ four feature selection methods, two of them are well-known supervised machine learning and two of them are un-supervised Algorithm classifier with an ability to builds a model by the weight of each feature, which is used to select features.

### **1.2.3 Logistic Regression**

Logistic regression is a statistical method used to predict the probability of a binary outcome, such as yes or no, pass or fail, or spam or not spam. It is a supervised learning algorithm, meaning that it learns from a training set of data that contains both the input variables and the corresponding output labels. The algorithm then uses this training data to fit a logistic regression model, which can be used to predict the probability of the binary outcome for new input data.

### **1.2.4 SVM (Support Vector Machine)**

- Support Vector Machine (SVM) is a supervised machine learning technique that is applied to regression and classification. We also use the word "regression concerns," although "categorization" is more suitable. The SVM approach aims to find a hyperplane in an N-dimensional space that unambiguously classifies the data points.
- The size of the hyperplane is determined by the number of features. When input characteristics are limited to two, the hyperplane can be thought of as a line. If three input characteristics are present, the hyperplane transforms into a 2-D plane.

### **1.2.5 K-Means**

- K-means clustering is a popular unsupervised machine learning algorithm that partitions a set of data points into K clusters. It is an iterative algorithm that aims to minimize the within-cluster variance, assigning data points to the cluster with the nearest mean (centroid).

### **1.2.6 Hierarchical Algorithm**

- Hierarchical clustering, also known as hierarchical cluster analysis or HCA, is a type of unsupervised machine learning algorithm that groups a set of data points into a hierarchy of clusters. It is an iterative algorithm that builds a dendrogram.

### **1.3 Organization of Report**

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, hardware and software details.
- Chapter 3 discusses the results obtained after the project was implemented.
- Chapter 4 concludes the report.
- Chapter 5 consists of codes.
- Chapter 6 gives references.

## **Chapter – 2**

### **A Machine Learning Based Intrusion Detection System for Automatic Feature Extraction and Selection**

This Chapter describes the proposed system, working methodology, software and hardware details.

#### **2.1 Proposed System**

Building many models, comparing their accuracy with the NSL-KDD dataset, determining if deep learning can be implemented, and determining which model is suitable for the Intrusion Detection System comprise the next project.

##### **2.1.1 Working Methodology**

###### **2.1.1(a) Feature Extraction**

To extract features, we use a stacked auto encoder approach based on neural networks. An unsupervised neural network model called an auto encoder uses backpropagation to adjust the target values to equal the inputs. For instance, this technique will be able to identify certain correlations if there are any between any of the input characteristics. The original features can be represented using this structure in a compressed manner that is more responsive to feature selection techniques in the future.

###### **2.1.1(b) Feature Selection**

We employ four feature selection methods, two of them are well-known supervised machine learning and two of them are un-supervised Algorithm classifier with an ability to build a model by the weight of each feature, which is used to select features.

#### **2.2 Standards**

Various standards used in this project are:

##### **2.2.1 NSL-KDD Dataset**

The KDD'99 data set has been updated with the NSL-KDD data set. An efficient benchmark data set for comparing various intrusion detection techniques is available to researchers. The NSL-KDD train and test sets have a respectable number of records. This benefit eliminates the requirement to

choose a small sample at random and makes it feasible to conduct the tests on the entire set at a reasonable cost. As a result, assessment findings from various research projects will be uniform and equivalent.

- **KDDTrain+.ARFF**: The full NSL-KDD train set with binary labels in ARFF format
- **KDDTrain+.TXT**: The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
- **KDDTrain+\_20Percent.ARFF**: A 20% subset of the KDDTrain+.arff file
- **KDDTrain+\_20Percent.TXT**: A 20% subset of the KDDTrain+.txt file
- **KDDTest+.ARFF**: The full NSL-KDD test set with binary labels in ARFF format
- **KDDTest+.TXT**: The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
- **KDDTest-21. ARFF**: A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
- **KDDTest-21.TXT**: A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

F#	Feature name	F#	Feature name	F#	Feature name
F1	Duration	F15	Su attempted	F29	Same srv rate
F2	Protocol type	F16	Num root	F30	Diff srv rate
F3	Service	F17	Num file creations	F31	Srv diff host rate
F4	Flag	F18	Num shells	F32	Dst host count
F5	Source bytes	F19	Num access files	F33	Dst host srv count
F6	Destination bytes	F20	Num outbound cmds	F34	Dst host same srv rate
F7	Land	F21	Is host login	F35	Dst host diff srv rate
F8	Wrong fragment	F22	Is guest login	F36	Dst host same src port rate
F9	Urgent	F23	Count	F37	Dst host srv diff host rate
F10	Hot	F24	Srv count	F38	Dst host serror rate
F11	Number failed logins	F25	Serror rate	F39	Dst host srv serror rate
F12	Logged in	F26	Srv serror rate	F40	Dst host rerror rate
F13	Num compromised	F27	Rerror rate	F41	Dst host srv rerror rate
F14	Root shell	F28	Srv rerror rate	F42	Class label

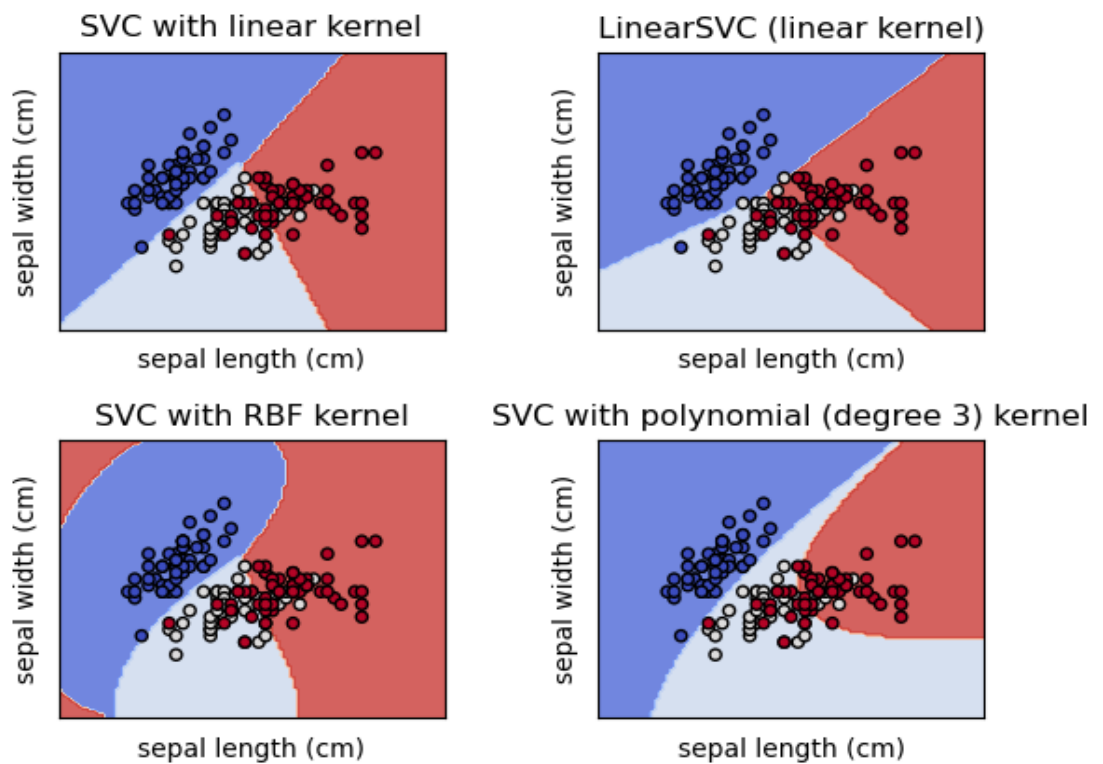
#### NSL-KDD dataset features

### 2.2.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning technique that is applied to regression and classification. We also use the word "regression concerns," although "categorization" is more suitable. The SVM approach aims to find a hyperplane in an N-

dimensional space that unambiguously classifies the data points. The size of the hyperplane is determined by the number of features. When input characteristics are limited to two, the hyperplane can be thought of as a line. If three input characteristics are present, the hyperplane transforms into a 2-D plane. It becomes difficult to imagine something having more than three characteristics. Given that it employs support vectors, a subset of training points in the decision function, it is memory-efficient and effective in high-dimensional situations.

Different kernel functions can be specified for the decision functions and its possible to specify custom kernels some of the used kernels are Linear kernel , RBF kernel , Polynomial kernel



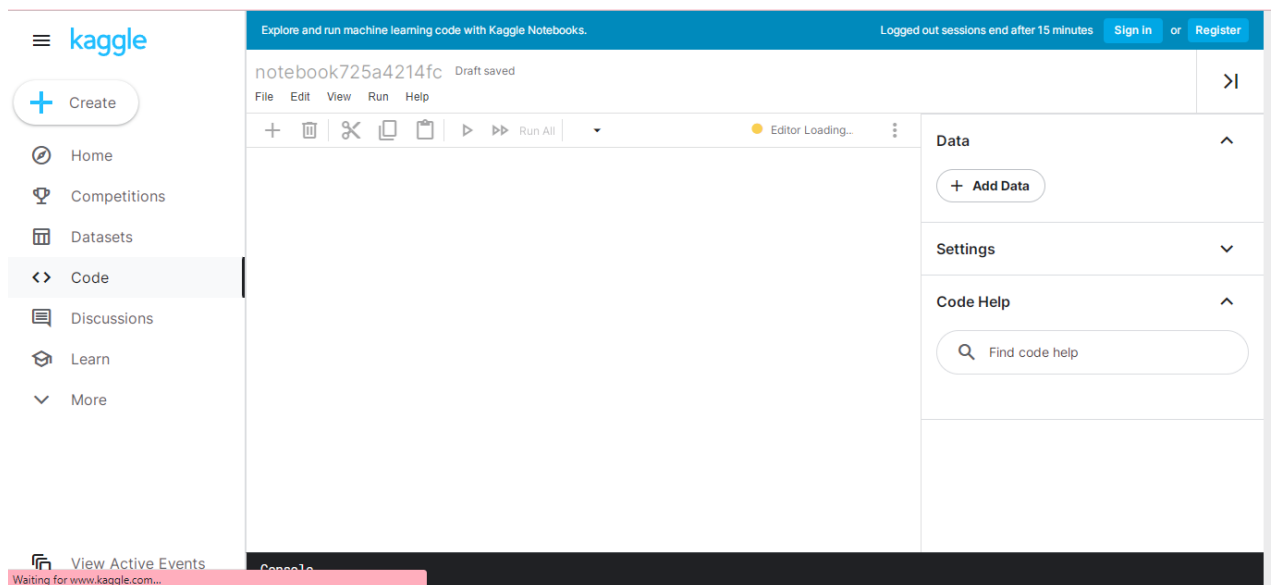
**SVM with kernel's**

## 2.3 Software Details

### 2.3.1 Kaggle

For those interested in machine learning and data science, Kaggle is an online community platform. Users may use GPU integrated notebooks, search and publish datasets, communicate with other users, and compete with other data scientists to solve data science challenges on Kaggle. With its strong tools and resources, this online platform—which was developed in 2010 by Anthony Goldbloom and Jeremy Howard and purchased by Google in 2017—aims to assist professionals and students in achieving their objectives in the field of data science. On Kaggle as of right now (2021), there are more than 8 million registered users.

With Kaggle, you may utilize up to 30 hours of GPU and 20 hours of TPU every week, along with robust cloud resources. We have the ability to download and submit datasets to Kaggle, including yours. You may also initiate conversation topics and examine other people's datasets and notebooks. Every action you do on the site earns you points, which rises when you lend a hand to others and distribute helpful content.

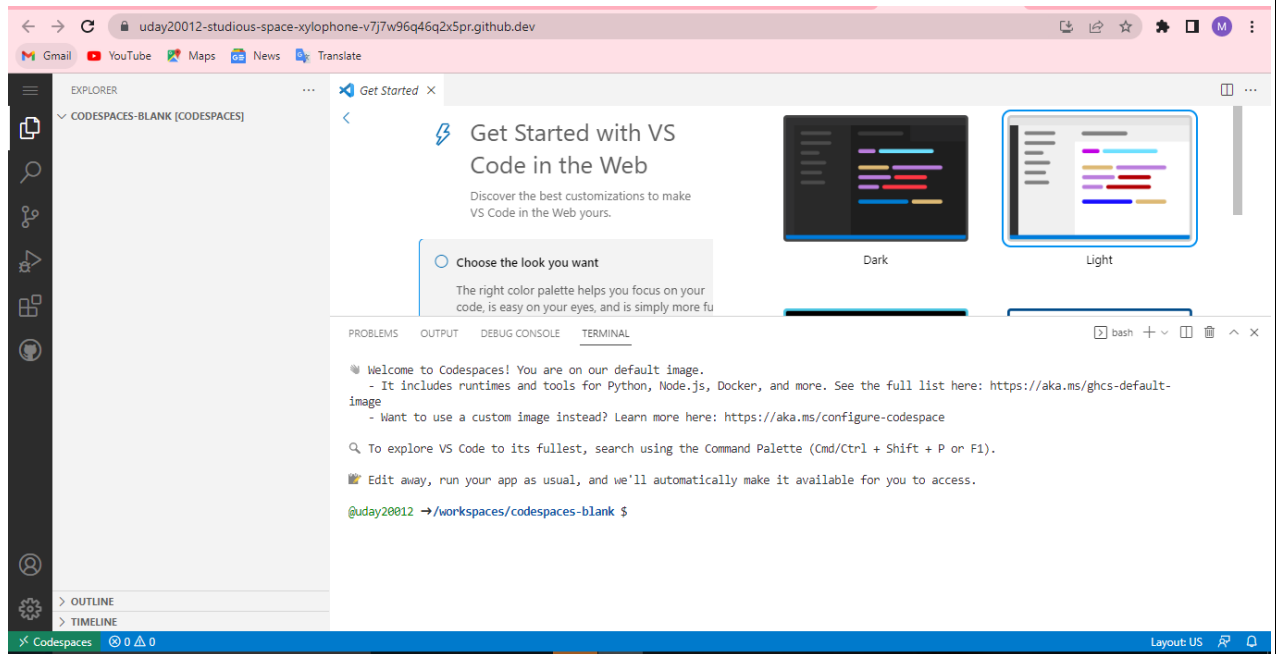


### Kaggle Website

### 2.3.2 GitHub

GitHub is a site that hosts code for version control and teamwork. It enables remote collaboration on projects between you and other people.

You will learn the fundamentals of GitHub, including repositories, branches, commits, and pull requests, in this lesson. You will learn how to generate and evaluate code using GitHub's pull request protocol, as well as establish your own Hello World repository.

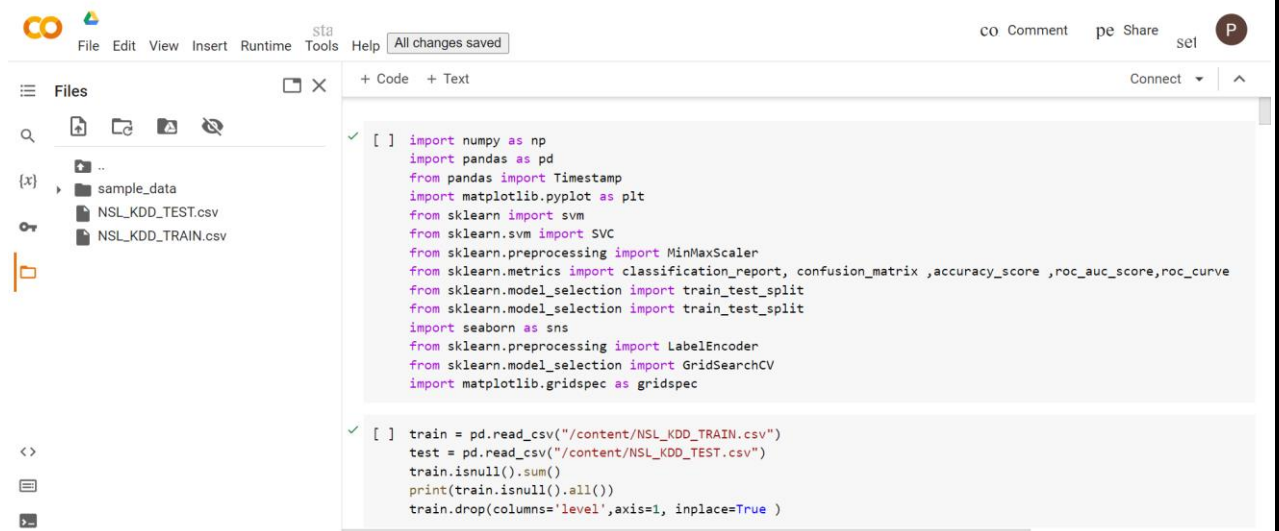
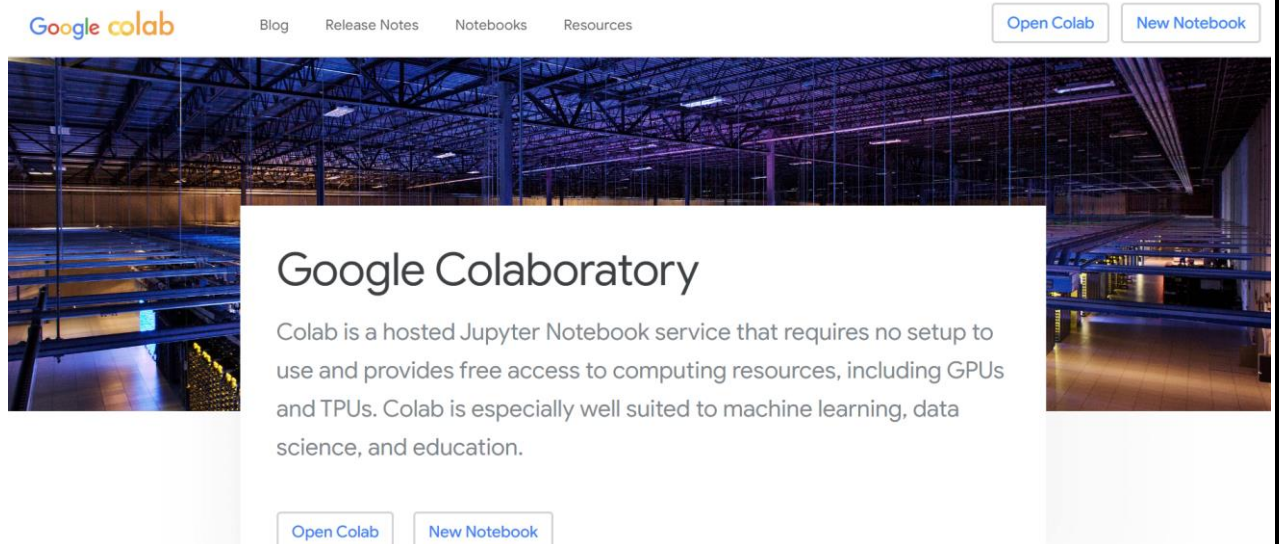


## GitHub Page

### 2.3.3 Collab

Google Research produces a product called Collaboratory, or simply "Collab." With Collab, anybody can create and run any Python code through a browser, making it particularly useful for data analysis, machine learning, and teaching. In technical terms, Collab is a hosted Jupyter notebook service that offers free access to GPU-intensive computing capabilities without requiring any setup.

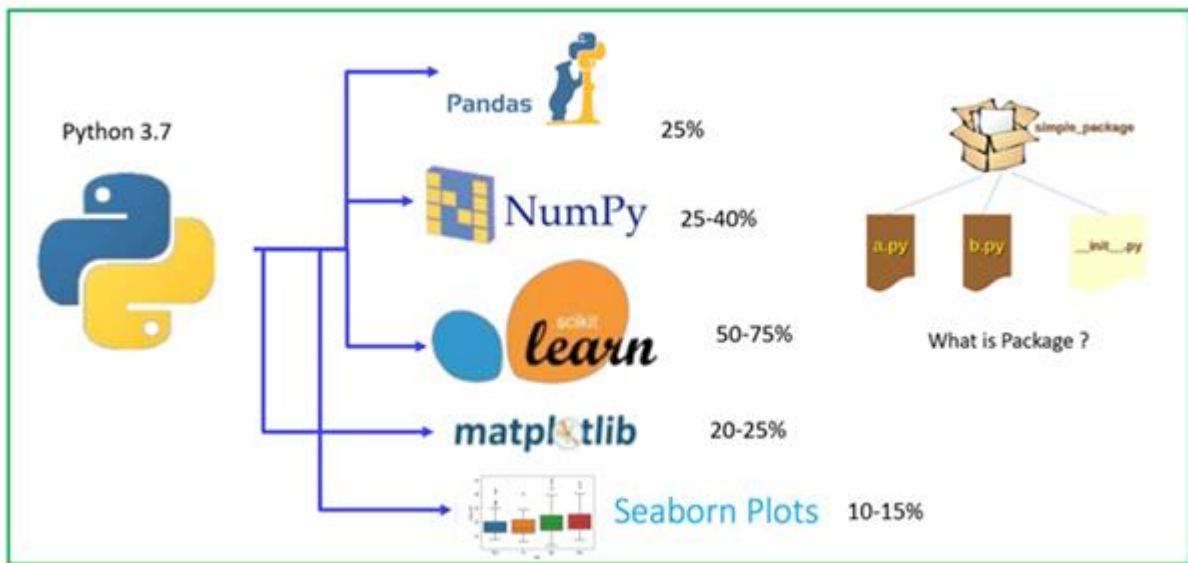




**Collab website**

### 2.3.4 Libraries Used

Libraries used are NumPy, Pandas, Matplotlib, Sklearn, seaborn, TensorFlow etc. for all the models



**Modules used for model building.**

## Chapter – 3

### Results And Discussions

#### 3.1 Data Exploration

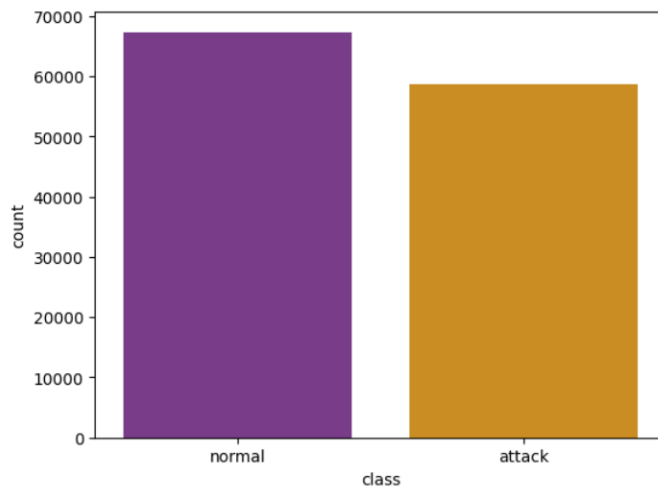
We have performed standard data investigations after obtaining the NSL-KDD dataset, such as data checking, binary value discovery, converting the data type from string to float or int, null and NA value removal, filling in missing values, etc.

```
[ ] train.columns
```

```
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',  
      'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',  
      'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',  
      'su_attempted', 'num_root', 'num_file_creations', 'num_shells',  
      'num_access_files', 'num_outbound_cmds', 'is_host_login',  
      'is_guest_login', 'count', 'srv_count', 'serror_rate',  
      'srv_serror_rate', 'error_rate', 'srv_error_rate', 'same_srv_rate',  
      'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',  
      'dst_host_srv_count', 'dst_host_same_srv_rate',  
      'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',  
      'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',  
      'dst_host_srv_serror_rate', 'dst_host_rerror_rate',  
      'dst_host_srv_rerror_rate', 'class', 'level'],  
      dtype='object')
```

#### Columns in NSL-KDD train dataset

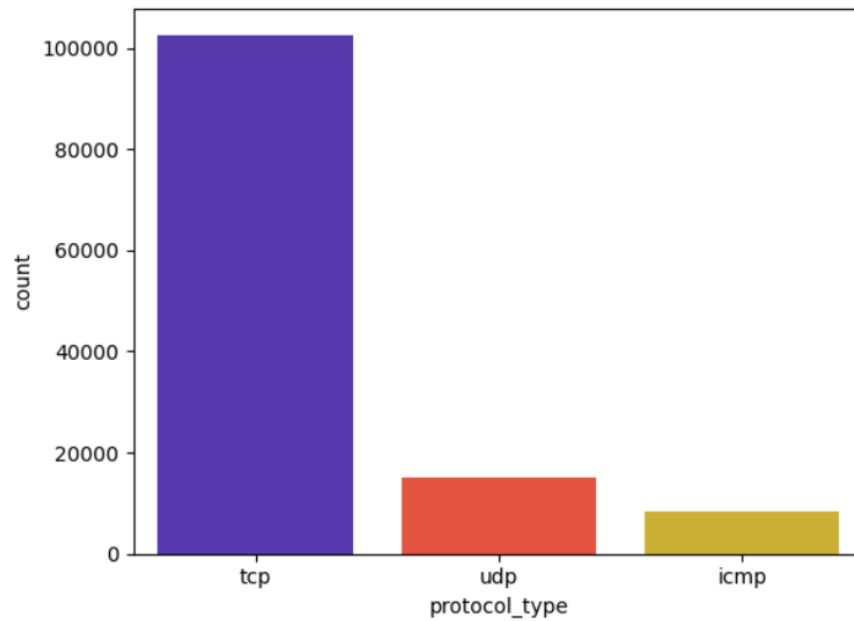
```
✓ [ ] import seaborn as sns  
import matplotlib.pyplot as plt  
  
sns.countplot(x='class', data=train, palette='CMRmap')  
plt.show()
```



Plot of attacks column

```
✓ [ ] import seaborn as sns
import matplotlib.pyplot as plt

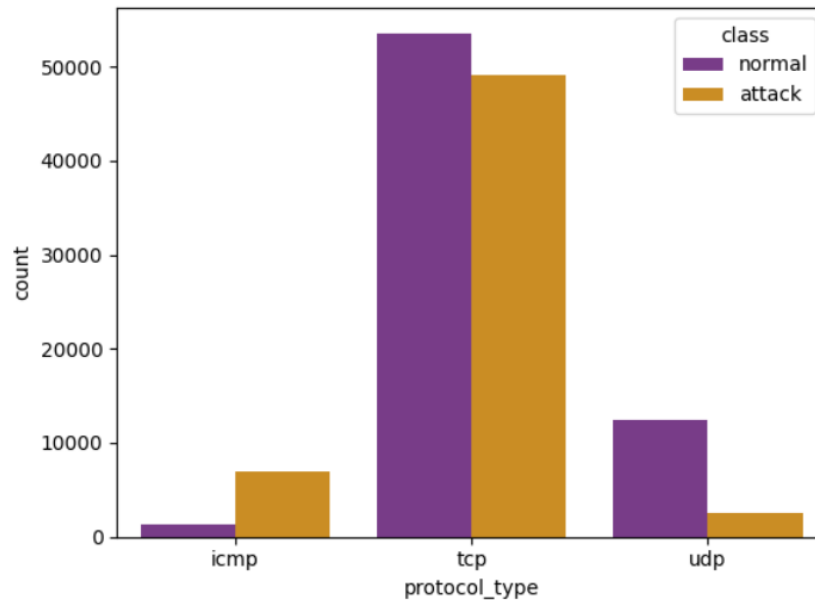
sns.countplot(x='protocol_type', data=train, palette='CMRmap')
plt.show()
```



**Plot of protocol type columns**

```
✓ [ ] import seaborn as sns
import matplotlib.pyplot as plt

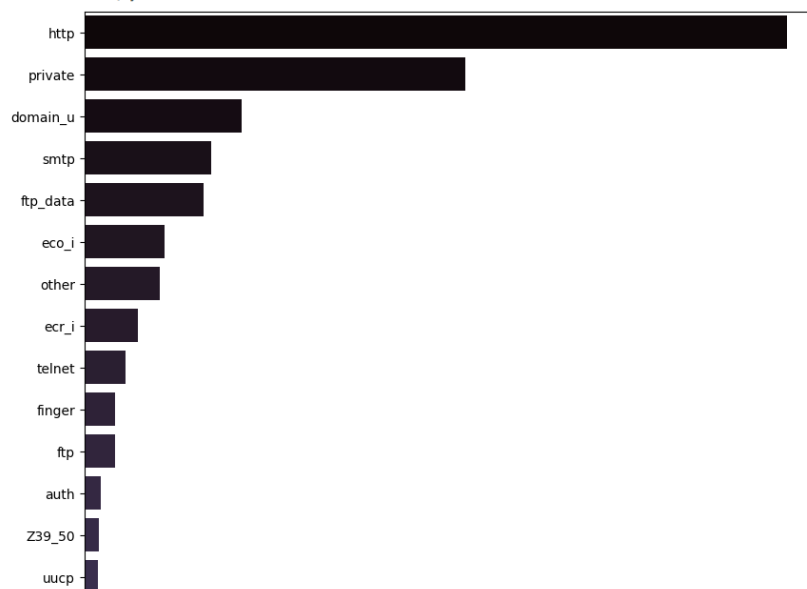
# Create countplot using 'protocol_type' as the x-axis and 'class' as the hue
sns.countplot(x='protocol_type', hue='class', data=train, palette='CMRmap')
plt.show()
```



**Plot between attack and protocol\_type columns**

```
✓ [ ] plt.figure(figsize=(10,40))
sns.countplot(palette='mako', y='service', data=train, order = train['service'].value_counts().index)
```

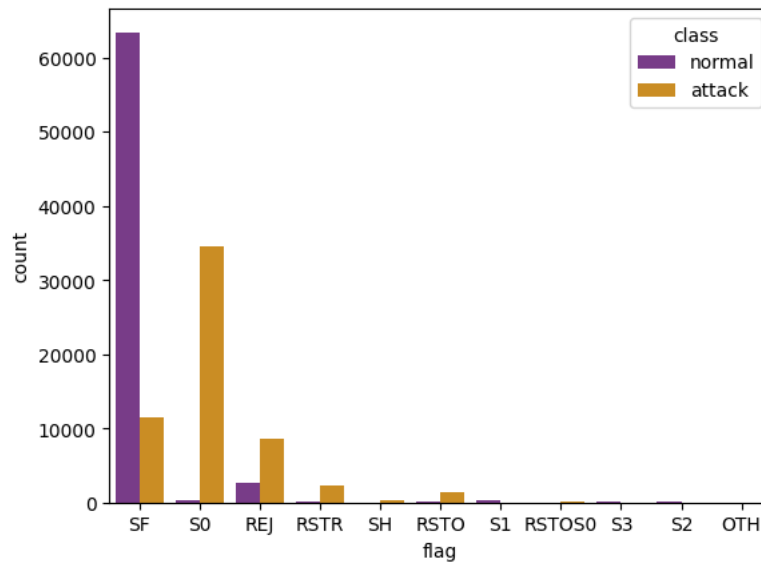
<Axes: xlabel='count', ylabel='service'>



**Plot of service column**

```
✓ [ ] sns.countplot(x='flag',hue='class' , data = train , palette = 'CMRmap')
```

<Axes: xlabel='flag', ylabel='count'>



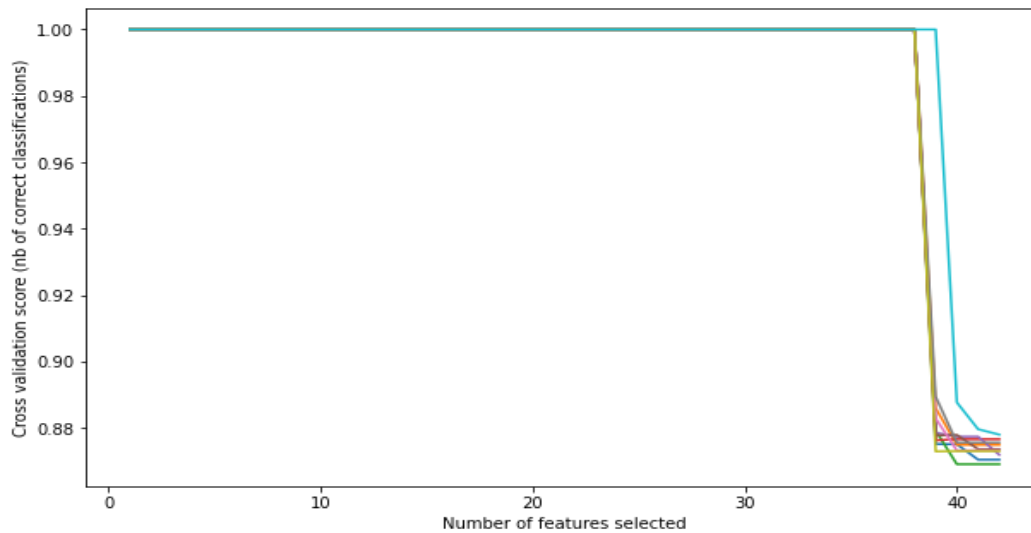
**Plot between service and protocol\_type columns**

### 3.2 Accuracy

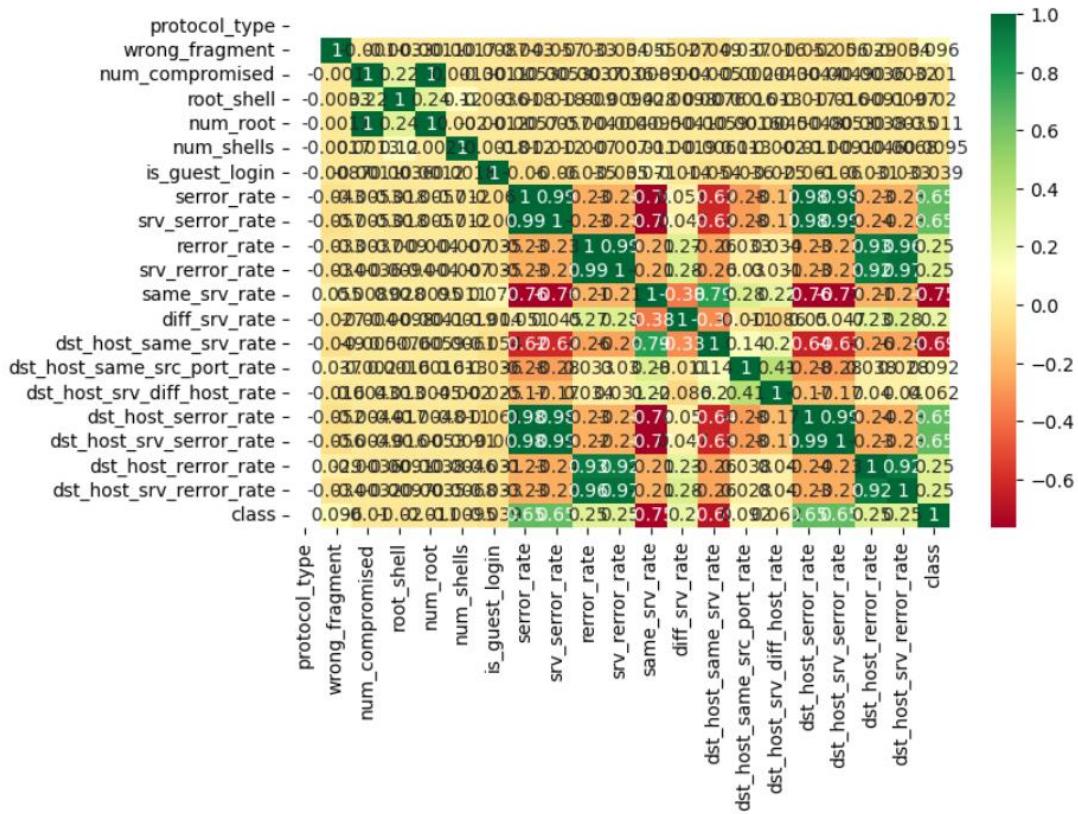
The accuracy was measured and compared with all the models which are been created. Logistic Regression, SVM (in SVM we have tried various kernels like linear, RBF, Polynomial etc.). In different methods we have tried the inclusion of test and train datasets and in some we have mixed them and tried our analysis.

```
Selected features: ['protocol_type', 'wrong_fragment', 'num_compromised', 'root_shell', 'num_root', 'num_shells', 'is_guest_login', 'serror_rate', 'srv_error_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate', 'dst_host_same_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'dst_host_serror_rate', 'dst_host_srv_error_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'class']
```

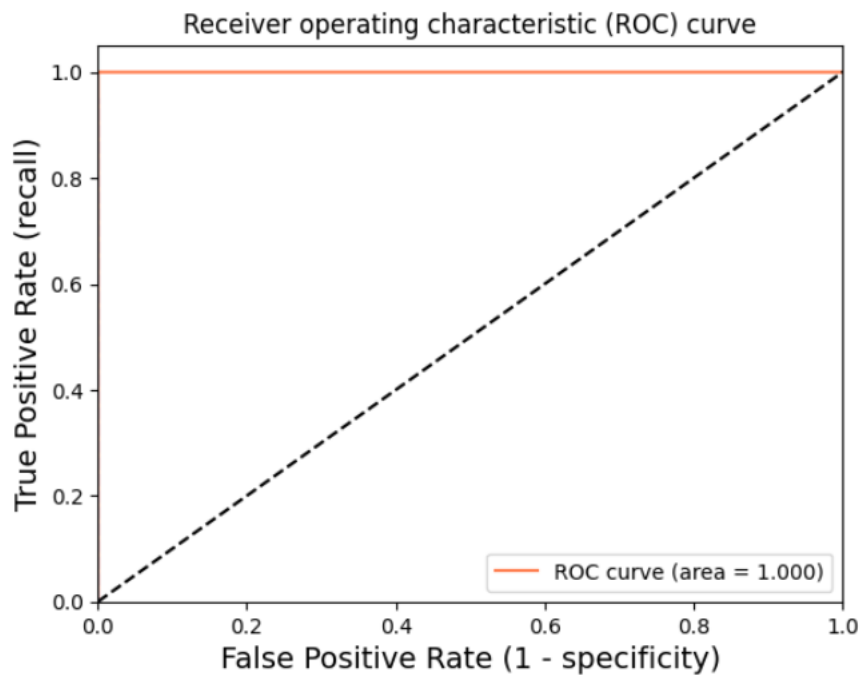
**Selected features(21 out of 43)**



Plot of features and relation



Co-relation plot



**True positive Vs False Positive Plot**

---

```
n_iter_1 = _check_optimize_result(  
K-fold cross-validation results:  
LogisticRegression average accuracy is 1.000  
LogisticRegression average log_loss is 0.000  
LogisticRegression average auc is 1.000
```

**Result of Logistic Regression**





## Liner SVC Accuracy

	precision	recall	f1-score	support
attack	1.00	1.00	1.00	17725
normal	1.00	1.00	1.00	20067
accuracy			1.00	37792
macro avg	1.00	1.00	1.00	37792
weighted avg	1.00	1.00	1.00	37792

## Liner SVC Result

```
> The Training accuracy = 0.9999659790657851
The Testing accuracy = 0.9993914055884844
```

```
-----
SVM (kernel: 'rbf') accuracy : 0.999
```

## RBF kernel Accuracy

	precision	recall	f1-score	support
attack	1.00	1.00	1.00	17725
normal	1.00	1.00	1.00	20067
accuracy			1.00	37792
macro avg	1.00	1.00	1.00	37792
weighted avg	1.00	1.00	1.00	37792

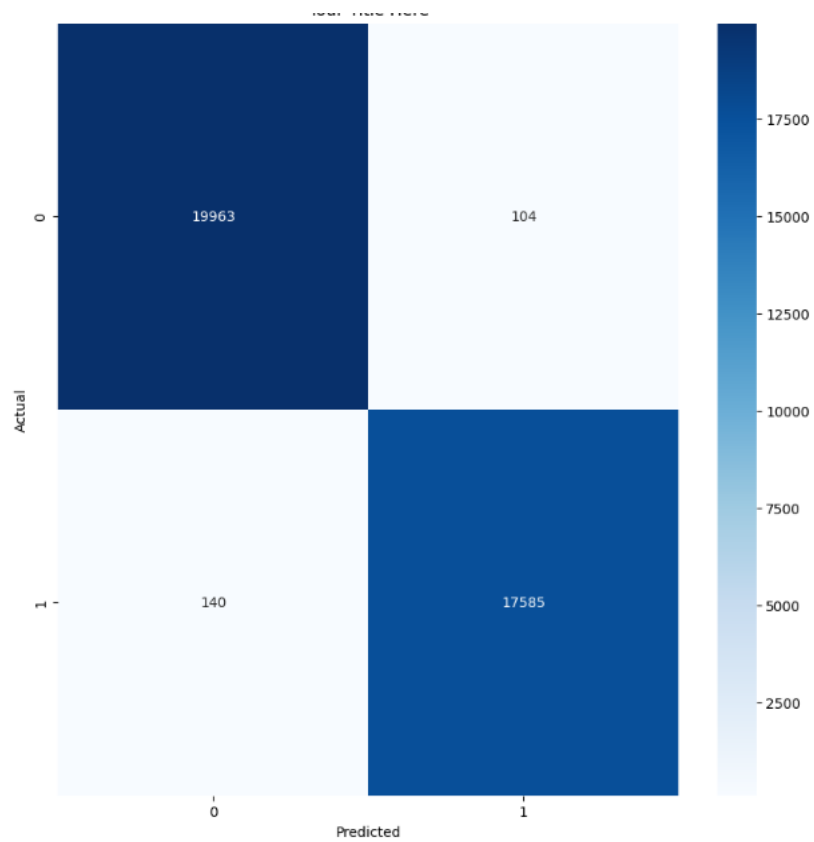
## RBF kernel Result

```
☞ The Training accuracy = 0.9999659790657851
The Testing accuracy = 0.9999206181202371
```

```
-----
SVM (kernel: 'poly') accuracy : 1.0
```

	precision	recall	f1-score	support
attack	1.00	1.00	1.00	17725
normal	1.00	1.00	1.00	20067
accuracy			1.00	37792
macro avg	1.00	1.00	1.00	37792
weighted avg	1.00	1.00	1.00	37792

## Polynomial Kernal Accuracy and Result



**Confusion Matrix of SVM**

## Chapter – 4

### Conclusion And Future Work

#### Conclusion

Following the analysis's conclusion, we identified the critical frequencies and tested them;

We obtained 0.99 accuracy using SVM.

We obtained 1.0 accuracy using logistic regression.

We obtained 0.920 accuracy using K-Means.

We obtained 0.754 accuracy using Hierarchical Algorithm.

Supervised learning algorithms typically have higher accuracy than unsupervised learning algorithms. This is because supervised learning algorithms are trained on labeled data, which means that the correct output is known for each input.

Unsupervised learning algorithms, on the other hand, are trained on unlabeled data, which means that the correct output is not known. This makes it more difficult for the algorithm to learn the underlying structure of the data, and as a result, unsupervised learning algorithms typically have lower accuracy than supervised learning algorithms.

#### Future Works

- The future works for network denial of service (DDoS) detection are likely to be more sophisticated and efficient. These systems will be able to detect and prevent DDoS attacks more effectively in real time, and they will be more resilient to new and emerging attack techniques.
- **Machine learning and artificial intelligence (AI):** Machine learning and AI can be used to develop more intelligent and effective DDoS detection systems. These systems can learn from historical data to identify patterns and anomalies that are indicative of DDoS attacks. They can also be used to adapt to new and emerging attack techniques.
- **Network programmability:** Software-defined networking (SDN) and other network programmability technologies can be used to develop more flexible and agile DDoS detection systems. These systems can be programmed to quickly respond to changes in the network topology and traffic patterns.

- **Distributed detection:** Distributed DDoS detection systems can be used to detect and mitigate attacks more effectively. These systems collect data from multiple sources, such as routers, switches, and firewalls, and use this data to identify and block DDoS attacks.
- **Hybrid Approaches:** Combine signature-based detection with anomaly-based methods for a more robust and comprehensive DoS detection system. Hybrid approaches can leverage the strengths of multiple techniques to improve accuracy and reduce false positives.
- **Edge Computing for DoS Mitigation:**  
Investigate the use of edge computing to implement distributed DoS detection and mitigation at the network's edge. Distributing detection mechanisms closer to the source can reduce latency and improve response times.
- **Behavioural Analysis:**  
Implement behavioural analysis techniques to identify deviations from normal network behaviour.

## Chapter – 5

### Appendix

#### 5.1 Logistic Regression

```
import numpy as np
import pandas as pd
from pandas import Timestamp
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix ,accuracy_score
,roc_auc_score,roc_curve
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
import matplotlib.gridspec as gridspec
train = pd.read_csv("/content/NSL_KDD_TRAIN.csv")
test = pd.read_csv("/content/NSL_KDD_TEST.csv")
train.isnull().sum()
print(train.isnull().all())
train.drop(columns='level',axis=1, inplace=True )
```

```
attack_n = []
for i in train['class']:
    if i == 'normal':
        attack_n.append("normal")
    else:
```

```
    attack_n.append("attack")
train['class'] = attack_n

print(train.head())
```

```
protocol_n = []
for i in train.protocol_type :
    if i == 'tcp':
        protocol_n.append(0)
    elif i == 'udp':
        protocol_n.append(1)
    else:
        protocol_n.append(2)
train['protocol_type'] = protocol_n
protocol_n = []
for i in train.protocol_type :
    if i == 'tcp':
        protocol_n.append(0)
    elif i == 'udp':
        protocol_n.append(1)
    else:
        protocol_n.append(2)
train['protocol_type'] = protocol_n

flag_n = []
for i in train.flag:
    if i == 'SF':
        flag_n.append(0)
    elif i == 'S0':
        flag_n.append(1)
    elif i == 'REJ':
        flag_n.append(2)
```

```

elif i == 'RSTR':
    flag_n.append(3)
elif i == 'SH':
    flag_n.append(4)
elif i == 'RSTO':
    flag_n.append(5)
elif i == 'S1':
    flag_n.append(6)
elif i == 'RSTOS0':
    flag_n.append(7)
elif i == 'S3':
    flag_n.append(8)
elif i == 'S2':
    flag_n.append(9)
elif i == 'OTH':
    flag_n.append(10)
else:
    flag_n.append(11)

```

```

train['flag'] = flag_n

```

```

service_n = []

```

```

for i in train.service:
    if i == 'ftp_data':
        service_n.append(0)
    elif i == 'other':
        service_n.append(1)
    elif i == 'private':
        service_n.append(2)
    elif i == 'http':
        service_n.append(3)
    elif i == 'remote_job':
        service_n.append(4)

```



```
elif i == 'name':
    service_n.append(5)
elif i == 'netbios_ns':
    service_n.append(6)
elif i == 'eco_i':
    service_n.append(7)
elif i == 'mtp':
    service_n.append(8)
elif i == 'telnet':
    service_n.append(9)
elif i == 'finger':
    service_n.append(10)
elif i == 'domain_u':
    service_n.append(11)
elif i == 'supdup':
    service_n.append(12)
elif i == 'uucp_path':
    service_n.append(13)
elif i == 'Z39_50':
    service_n.append(14)
elif i == 'smtp':
    service_n.append(15)
elif i == 'csnet_ns':
    service_n.append(16)
elif i == 'uucp':
    service_n.append(17)
elif i == 'netbios_dgm':
    service_n.append(18)
elif i == 'urp_i':
    service_n.append(19)
elif i == 'auth':
    service_n.append(20)
elif i == 'domain':
```

```
service_n.append(21)
elif i == 'ftp':
    service_n.append(22)
elif i == 'bgp':
    service_n.append(23)
elif i == 'ldap':
    service_n.append(24)
elif i == 'ecr_i':
    service_n.append(25)
elif i == 'gopher':
    service_n.append(26)
elif i == 'vmnet':
    service_n.append(27)
elif i == 'systat':
    service_n.append(28)
elif i == 'http_443':
    service_n.append(29)
elif i == 'efs':
    service_n.append(30)
elif i == 'whois':
    service_n.append(31)
elif i == 'imap4':
    service_n.append(32)
elif i == 'iso_tsap':
    service_n.append(33)
elif i == 'echo':
    service_n.append(34)
elif i == 'klogin':
    service_n.append(35)
elif i == 'link':
    service_n.append(36)
elif i == 'sunrpc':
    service_n.append(37)
```

```
elif i == 'login':
    service_n.append(38)
elif i == 'kshell':
    service_n.append(39)
elif i == 'sql_net':
    service_n.append(40)
elif i == 'time':
    service_n.append(41)
elif i == 'hostnames':
    service_n.append(42)
elif i == 'exec':
    service_n.append(43)
elif i == 'ntp_u':
    service_n.append(44)
elif i == 'discard':
    service_n.append(45)
elif i == 'nntp':
    service_n.append(46)
elif i == 'courier':
    service_n.append(47)
elif i == 'ctf':
    service_n.append(48)
elif i == 'ssh':
    service_n.append(49)
elif i == 'daytime':
    service_n.append(50)
elif i == 'shell':
    service_n.append(51)
elif i == 'netstat':
    service_n.append(52)
elif i == 'pop_3':
    service_n.append(53)
elif i == 'nnsp':
```

```
service_n.append(54)
elif i == 'IRC':
    service_n.append(55)
elif i == 'pop_2':
    service_n.append(56)
elif i == 'printer':
    service_n.append(57)
elif i == 'tim_i':
    service_n.append(58)
elif i == 'pm_dump':
    service_n.append(59)
elif i == 'red_i':
    service_n.append(60)
elif i == 'netbios_ssn':
    service_n.append(61)
elif i == 'rje':
    service_n.append(62)
elif i == 'X11':
    service_n.append(63)
elif i == 'urh_i':
    service_n.append(64)
elif i == 'http_8001':
    service_n.append(65)
elif i == 'aol':
    service_n.append(66)
elif i == 'http_2784':
    service_n.append(67)
elif i == 'tftp_u':
    service_n.append(68)
elif i == 'harvest':
    service_n.append(69)
else:
    service_n.append(70)
```

```

train['service'] = service_n

attack_n = []
for i in train['class']:
    if i == 'normal':
        attack_n.append(0)
    else:
        attack_n.append(1)
train['class'] = attack_n

from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

cols = train.columns
X = train[cols]
y = train['class']
# Build a logreg and compute the feature importances
model=LogisticRegression(max_iter=13000)

# create the RFE model and select 8 attributes
rfe = RFE(model)

rfe = rfe.fit(X, y)
# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))

from sklearn.feature_selection import RFECV
# Create the RFE object and compute a cross-validated score.
# The "accuracy" scoring is proportional to the number of correct classifications
rfecv = RFECV(estimator=LogisticRegression(max_iter=13000), step=1, cv=10,
scoring='accuracy')

```

```

rfecv.fit(X,y)

print("Optimal number of features: %d" % rfecv.n_features_)
print('Selected features: %s' % list(X.columns[rfecv.support_]))

# Plot number of features VS. cross-validation scores
import numpy as np
import matplotlib.pyplot as plt

# Your existing code
plt.figure(figsize=(10, 6))
plt.xlabel("Number of features selected")
plt.ylabel("Cross-validation score (mean test score)")

# Use rfecv.cv_results_ instead of rfecv.grid_scores_
plt.plot(range(1, len(rfecv.cv_results_['mean_test_score']) + 1),
rfecv.cv_results_['mean_test_score'])

# Additional suggestions
plt.title('Number of Features vs. Cross-Validation Scores with RFECV')

max_score_index = np.argmax(rfecv.cv_results_['mean_test_score'])
plt.annotate(f'Max Score ({max_score_index+1} features)',
            xy=(max_score_index+1, rfecv.cv_results_['mean_test_score'][max_score_index]),
            xytext=(max_score_index+5, rfecv.cv_results_['mean_test_score'][max_score_index] +
0.01),
            arrowprops=dict(facecolor='red', shrink=0.05),
            )

plt.show()

Selected_features = ['protocol_type', 'wrong_fragment', 'num_compromised', 'root_shell',
'num_root', 'num_shells', 'is_guest_login', 'serror_rate', 'srv_error_rate', 'error_rate',
'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate', 'dst_host_same_srv_rate',

```

```
'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
'dst_host_srv_serror_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'class']
X = train[Selected_features]
```

```
plt.subplots(figsize=(8, 5))
sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")
plt.show()
```

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_curve, auc, log_loss

# create X (features) and y (response)
X = train[Selected_features]
y = train['class']
```

```
# use train/test split with different random_state values
# we can change the random_state values that changes the accuracy scores
# the scores change a lot, this is why testing scores is a high-variance estimate
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

```
# check classification scores of logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_pred_proba = logreg.predict_proba(X_test)[:, 1]
[fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
print("Train/Test split results:")
print(logreg.__class__.__name__ + " accuracy is %2.3f" % accuracy_score(y_test, y_pred))
```

```

print(logreg.__class__.__name__+" log_loss is %2.3f" % log_loss(y_test, y_pred_proba))
print(logreg.__class__.__name__+" auc is %2.3f" % auc(fpr, tpr))

idx = np.min(np.where(tpr > 0.95)) # index of the first threshold for which the sensibility > 0.95

plt.figure()
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot([0, fpr[idx]], [tpr[idx], tpr[idx]], 'k--', color='blue')
plt.plot([fpr[idx], fpr[idx]], [0, tpr[idx]], 'k--', color='blue')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)
plt.ylabel('True Positive Rate (recall)', fontsize=14)
plt.title('Receiver operating characteristic (ROC) curve')
plt.legend(loc="lower right")
plt.show()

```

```

print("Using a threshold of %.3f " % thr[idx] + "guarantees a sensitivity of %.3f " % tpr[idx] +
      "and a specificity of %.3f" % (1-fpr[idx]) + ", i.e. a false positive rate of %.2f%%." %
      (np.array(fpr[idx])*100))

```

```

# 10-fold cross-validation logistic regression
logreg = LogisticRegression()
# Use cross_val_score function
# We are passing the entirety of X and y, not X_train or y_train, it takes care of splitting the data
# cv=10 for 10 folds
# scoring = {'accuracy', 'neg_log_loss', 'roc_auc'} for evaluation metric - although they are many
scores_accuracy = cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
scores_log_loss = cross_val_score(logreg, X, y, cv=10, scoring='neg_log_loss')
scores_auc = cross_val_score(logreg, X, y, cv=10, scoring='roc_auc')

```



```

print('K-fold cross-validation results:')
print(logreg.__class__.__name__+" average accuracy is %2.3f" % scores_accuracy.mean())
print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss.mean())
print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())

```

```

from sklearn.model_selection import cross_validate

scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelCV = LogisticRegression()

results = cross_validate(modelCV, X, y, cv=10, scoring=list(scoring.values()),
                        return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc],
    -results['test_%s' % list(scoring.values())[sc]].mean()
        if list(scoring.values())[sc]=='neg_log_loss'
        else results['test_%s' % list(scoring.values())[sc]].mean(),
        results['test_%s' % list(scoring.values())[sc]].std()))

from sklearn.model_selection import GridSearchCV

X = train[Selected_features]

```

```

param_grid = {'C': np.arange(1e-05, 3, 0.1)}
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}

gs = GridSearchCV(LogisticRegression(), return_train_score=True,
                  param_grid=param_grid, scoring=scoring, cv=10, refit='Accuracy')

```

```
gs.fit(X, y)
```

```
results = gs.cv_results_  
print('='*20)  
print("best params: " + str(gs.best_estimator_))  
print("best params: " + str(gs.best_params_))  
print('best score:', gs.best_score_)  
print('='*20)
```

```
plt.figure(figsize=(10, 10))  
plt.title("GridSearchCV evaluating using multiple scorers simultaneously", fontsize=16)
```

```
plt.xlabel("Inverse of regularization strength: C")  
plt.ylabel("Score")  
plt.grid()
```

```
ax = plt.axes()  
ax.set_xlim(0, param_grid['C'].max())  
ax.set_ylim(0.35, 0.95)
```

```
# Get the regular numpy array from the MaskedArray  
X_axis = np.array(results['param_C'].data, dtype=float)  
  
for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):  
    for sample, style in (('train', '--'), ('test', '-')):  
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer] ==  
'neg_log_loss' else results['mean_%s_%s' % (sample, scorer)]  
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
```

```

label = "%s (%s)" % (scorer, sample)
ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                sample_score_mean + sample_score_std,
                alpha=0.1 if sample == 'test' else 0, color=color)
ax.plot(X_axis, sample_score_mean, style, color=color,
        alpha=1 if sample == 'test' else 0.7,
        label=label)

best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer] == 'neg_log_loss'
else results['mean_test_%s' % scorer][best_index]

# Plot a dotted vertical line at the best score for that scorer marked by x
ax.plot([X_axis[best_index], ] * 2, [0, best_score], linestyle='-.', color=color, marker='x',
        markeredgewidth=3, ms=8)

# Annotate the best score for that scorer
ax.annotate("%0.2f" % best_score,
            (X_axis[best_index], best_score + 0.005))

# Add labels and title
plt.xlabel("C (Regularization Parameter)")
plt.ylabel("Score")
plt.title("Hyperparameter Tuning Results")

# Add legend and grid lines
plt.legend(loc="upper right")
plt.grid(True, linestyle='--', alpha=0.7)

plt.show()

```

```

from sklearn.preprocessing import StandardScaler

```

```

from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline

#Define simple model
#####
#
C = np.arange(1e-05, 5.5, 0.1)
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}
log_reg = LogisticRegression()

#Simple pre-processing estimators
#####
#
std_scale = StandardScaler(with_mean=False, with_std=False)
#std_scale = StandardScaler()

#Defining the CV method: Using the Repeated Stratified K Fold
#####
#

n_folds=5
n_repeats=5

rskfold = RepeatedStratifiedKFold(n_splits=n_folds, n_repeats=n_repeats, random_state=2)

#Creating simple pipeline and defining the gridsearch
#####
#

log_clf_pipe = Pipeline(steps=[('scale',std_scale), ('clf',log_reg)])

log_clf = GridSearchCV(estimator=log_clf_pipe, cv=rskfold,

```

```
scoring=scoring, return_train_score=True,  
param_grid=dict(clf__C=C), refit='Accuracy')
```

```
log_clf.fit(X, y)
```

```
results = log_clf.cv_results_
```

```
print('='*20)  
print("best params: " + str(log_clf.best_estimator_))  
print("best params: " + str(log_clf.best_params_))  
print('best score:', log_clf.best_score_)  
print('='*20)
```

```
plt.figure(figsize=(10, 10))  
plt.title("GridSearchCV evaluating using multiple scorers simultaneously", fontsize=16)  
  
plt.xlabel("Inverse of regularization strength: C")  
plt.ylabel("Score")  
plt.grid()  
  
ax = plt.axes()  
ax.set_xlim(0, C.max())  
ax.set_ylim(0.35, 0.95)
```

```
# Get the regular numpy array from the MaskedArray  
X_axis = np.array(results['param_clf__C'].data, dtype=float)  
for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):  
    for sample, style in (('train', '--'), ('test', '-')):  
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if  
scoring[scorer]=='neg_log_loss' else results['mean_%s_%s' % (sample, scorer)]  
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
```

```

ax.fill_between(X_axis, sample_score_mean - sample_score_std,
               sample_score_mean + sample_score_std,
               alpha=0.1 if sample == 'test' else 0, color=color)
ax.plot(X_axis, sample_score_mean, style, color=color,
        alpha=1 if sample == 'test' else 0.7,
        label="%s (%s)" % (scorer, sample))

best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss'
else results['mean_test_%s' % scorer][best_index]

# Plot a dotted vertical line at the best score for that scorer marked by x
ax.plot([X_axis[best_index], ] * 2, [0, best_score],
        linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

# Annotate the best score for that scorer
ax.annotate("%0.2f" % best_score,
           (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")
plt.grid('off')
plt.show()

```

## 5.2 Support Vector Machine SVM

```
import numpy as np
import pandas as pd
from pandas import Timestamp
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix ,accuracy_score
,roc_auc_score,roc_curve
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
import matplotlib.gridspec as gridspec
```

```
train = pd.read_csv("/content/NSL_KDD_TRAIN.csv")
```

```
train.columns
```

```
train.isnull().sum()
```

```
train.drop(columns='level',axis=1, inplace=True )
```

```
attack_n = []
```

```
for i in train['class'] :
```

```
    if i == 'normal':
```

```
        attack_n.append("normal")
```

```
    else:
```

```
attack_n.append("attack")  
train['class'] = attack_n  
train['class'].value_counts()
```

```
print(train.columns)
```

```
print(train.head())
```

```
print(train.dtypes)
```

```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
sns.countplot(x='class', data=train, palette='CMRmap')  
plt.show()
```

```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
sns.countplot(x='protocol_type', data=train, palette='CMRmap')  
plt.show()
```

```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Convert 'protocol_type' to categorical
```



```
train['protocol_type'] = train['protocol_type'].astype('category')
```

```
# Create countplot
```

```
sns.countplot(x='protocol_type', data=train, palette='CMRmap')  
plt.show()
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
# Check column names
```

```
print(train.columns)
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
# Create countplot using 'protocol_type' as the x-axis and 'class' as the hue
```

```
sns.countplot(x='protocol_type', hue='class', data=train, palette='CMRmap')  
plt.show()
```

```
plt.figure(figsize=(10,40))
```

```
sns.countplot(palette='mako', y='service', data=train, order = train['service'].value_counts().index)
```

```
sns.countplot(x='flag', hue='class', data = train , palette = 'CMRmap')
```

```
data_obj = train.select_dtypes(['object']).columns
```

```
train["class"].value_counts()
```

```
data_obj
```

```
from sklearn.preprocessing import LabelEncoder
```

```
protocol_type_le = LabelEncoder()
service_le = LabelEncoder()
flag_le = LabelEncoder()
train['protocol_type'] = protocol_type_le.fit_transform(train['protocol_type'])
train['service'] = service_le.fit_transform(train['service'])
train['flag'] = flag_le.fit_transform(train['flag'])
```

```
attack_n = []
for i in train['class']:
    if i == 'normal':
        attack_n.append(0)
    else:
        attack_n.append(1)

train['class'] = attack_n
train['class'].value_counts()
```

```
plt.figure(figsize=(30,30))
sns.heatmap(train.corr(), annot= True,cmap='mako')
```

```
y = train['class'].copy()
x = train.drop(['class'], axis=1)
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y , test_size=0.3, random_state=40)
```

```
from sklearn.preprocessing import StandardScaler  
scalar=StandardScaler()  
x_train=scalar.fit_transform(x_train)  
x_test = scalar.fit_transform(x_test)
```

```
lin_svc = svm.LinearSVC().fit(x_train, y_train)
```

```
Y_pred =lin_svc.predict(x_test)
```

```
print("The Training accuracy = ',lin_svc.score(x_train, y_train))  
print("The Testing accuracy = ',lin_svc.score(x_test, y_test))  
print("-----")  
print( "linearSVC accuracy : " + str(np.round(accuracy_score(y_test,Y_pred),3)))  
  
print(classification_report(y_test,Y_pred))
```

```
rbf_svc = svm.SVC(kernel='rbf').fit(x_train, y_train)  
Y_pred_rbf =rbf_svc.predict(x_test)
```

```
print("The Training accuracy = ',rbf_svc.score(x_train, y_train))  
print("The Testing accuracy = ',rbf_svc.score(x_test, y_test))  
print("-----")  
print( "SVM (kernel: 'rbf') accuracy : " + str(np.round(accuracy_score(y_test,Y_pred_rbf),3)))  
print(classification_report(y_test,Y_pred_rbf))
```

```
svclassifier_poly = SVC(kernel='poly')a
```

```

poly = svcclassifier_poly.fit(x_train,y_train)
Y_pred_poly =svcclassifier_poly.predict(x_test)

print('The Training accuracy = ',poly.score(x_train, y_train))
print('The Testing accuracy = ',poly.score(x_test, y_test))
print("-----")
print( "SVM (kernel: 'poly') accuracy : " +
str(np.round(accuracy_score(y_test,Y_pred_poly),3)))
print(classification_report(y_test,Y_pred_poly))
print(classification_report(y_test,Y_pred_poly))

param_grid = { 'C': [0.2,0.5,1], 'gamma': [0.5],'kernel': ['rbf']}
grid = GridSearchCV(SVC(),param_grid ,verbose=2, cv= 3,refit=False)

grid.fit(x_train,y_train)

```

```

print(grid.best_params_)

```

```

rbf_svc = svm.SVC(kernel=grid.best_params_['kernel'], gamma=grid.best_params_['gamma'],
C=grid.best_params_['C']).fit(x_train, y_train)

```

```

Y_pred_rbf =rbf_svc.predict(x_test)

```

```

print(rbf_svc.score(x_train, y_train))
print(rbf_svc.score(x_test, y_test))
print("-----")
print( "SVM (kernel: 'rbf') accuracy : " + str(np.round(accuracy_score(y_test,Y_pred_rbf),3)))

print(classification_report(y_test,Y_pred_rbf))

```

```
def plot_cm(y_true, y_pred, title):
    figsize=(10,10)
    cm = confusion_matrix(y_true, y_pred, labels=np.unique(y_true))
    cm_sum = np

plot_cm(y_test,Y_pred_rbf, 'Confusion matrix for predictions on the testing using SVC(kernel :
\'rbf\'))
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def plot_cm(y_true, y_pred, title):
    figsize = (10, 10)
    cm = confusion_matrix(y_true, y_pred, labels=np.unique(y_true))

    plt.figure(figsize=figsize)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_true),
yticklabels=np.unique(y_true))
    plt.title(title)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

# Assuming you have y_test and Y_pred_rbf defined
# Replace 'Your Title Here' with an appropriate title
plot_cm(y_test, Y_pred_rbf, 'Your Title Here')
```

## 5.3 K-Means

```
import pandas as pd

# Read data from two CSV files
test = pd.read_csv('/content/sample_data/NSL_KDD_TEST.csv')
train = pd.read_csv('/content/sample_data/NSL_KDD_TRAIN.csv')

print(train.columns)
print(test.columns)

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Read data from two CSV files
train = pd.read_csv('/content/sample_data/NSL_KDD_TRAIN.csv')
test = pd.read_csv('/content/sample_data/NSL_KDD_TEST.csv')

# Concatenate the datasets vertically (assuming they have the same columns)
merged_data = pd.concat([train, test], ignore_index=True)

# Select relevant features for clustering (you may need to adjust this based on your specific use case)
features = merged_data[['duration', 'src_bytes', 'dst_bytes']]

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Determine the optimal number of clusters using the Elbow Method
inertias = []
K_range = range(1, 11)
```

```

for k in K_range:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(scaled_features)
    inertias.append(kmeans.inertia_)

# Plotting the Elbow Method graph
plt.plot(K_range, inertias, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method for Optimal k')
plt.show()

# Choose the optimal k based on the Elbow Method (e.g., k=3)

# Apply K-Means with the chosen k
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k)
merged_data['cluster'] = kmeans.fit_predict(scaled_features)

# Display the first few rows of the merged_data with cluster information
print(merged_data.head())

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Assuming 'class' is the target column
X = merged_data[['duration', 'src_bytes', 'dst_bytes']]
y = merged_data['class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```

# Create a decision tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Fit the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

```

## 5.4 Hierarchical Algorithm

```

import pandas as pd
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import pdist, squareform
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

# Load data from CSV files
test = pd.read_csv('/content/sample_data/NSL_KDD_TEST.csv')
train = pd.read_csv('/content/sample_data/NSL_KDD_TRAIN.csv')

# Concatenate the data from both files
merged_data = pd.concat([train, test], ignore_index=True)

# Select only numeric columns
numeric_columns = merged_data.select_dtypes(include=['float64', 'int64']).columns
merged_data_subset = merged_data[numeric_columns].sample(frac=0.1, random_state=42)

```



```

# Calculate the pairwise distance matrix
distance_matrix = pdist(merged_data_subset)

# Perform hierarchical clustering
linkage_matrix = linkage(distance_matrix, method='complete')

# Plot the dendrogram
dendrogram(linkage_matrix, labels=merged_data_subset.index, orientation='top')
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Assuming you have already loaded your datasets into train and test DataFrames

# Identify target variable in the training set
y_train = train['class']

# Identify categorical columns in your training dataset
categorical_columns_train = train.select_dtypes(include=['object']).columns

# Apply one-hot encoding to the categorical columns in the training set
X_train_encoded = pd.get_dummies(train, columns=categorical_columns_train)

# Drop the target variable from X_train_encoded if it exists
if 'class' in X_train_encoded.columns:
    X_train_encoded = X_train_encoded.drop('class', axis=1)

# Train the model

```

```
model = RandomForestClassifier()
model.fit(X_train_encoded, y_train)

# Apply the same transformations to the test set
# Identify categorical columns in your test dataset
categorical_columns_test = test.select_dtypes(include=['object']).columns

# Apply one-hot encoding to the categorical columns in the test set
X_test_encoded = pd.get_dummies(test, columns=categorical_columns_test)

# Reorder the columns to match the order in X_train_encoded
missing_columns = set(X_train_encoded.columns) - set(X_test_encoded.columns)
for column in missing_columns:
    X_test_encoded[column] = 0

X_test_encoded = X_test_encoded[X_train_encoded.columns]

# Make predictions on the test set
predictions = model.predict(X_test_encoded)

# Evaluate accuracy
accuracy = accuracy_score(test['class'], predictions)
print(f"Accuracy: {accuracy}")
```

## Chapter – 6

### References

- [1] J. Liu and S. S. Chung, "Automatic Feature Extraction and Selection For Machine Learning Based Intrusion Detection," *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCCom/IOP/SCI)*, 2019, pp. 1400-1405, doi:10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00254.
- [2] M. Ishaque and L. Hudec, "Feature extraction using Deep Learning for Intrusion Detection System," *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, 2019, pp. 1-5, doi: 10.1109/CAIS.2019.8769473.
- [3] Towards a Reliable Comparison and Evaluation of Network Intrusion Detection Systems Based on Machine Learning Approaches:-[Roberto Magán-Carrión](#),[Daniel Urda](#),[Ignacio Díaz-Cano](#),[Bernabé Dorronsoro Díaz](#) oai:rodin.uca.es:10498/22871
- [4] Training an Intrusion Detection System with Keras and KDD99 (14.4)[YOUTUBE]  
<https://www.youtube.com/watch?v=VgyKQ5MTDFc&t=2s>
- [5] [Network intrusion detection using deep learning techniques](#) [YOUTUBE]  
<https://www.youtube.com/watch?v=wUMObYAhQ4I>
- [6] C. Koliás, G. Kambourakis, and M. Maragoudakis, "Swarm intelligence in intrusion detection: A survey," *Computers & Security*, vol. 30, no. 8, pp. 625–642, 2011.
- [7] H. Motoda and H. Liu, "Feature selection, extraction and construction, Communication of IICM (Institute of Information and Computing Machinery), Taiwan, vol. 5, pp. 67–72, 2002.
- [8] Tanuwidjaja, P. D. Yoo and K. Kim, "Deep Abstraction and Weighted Feature Selection for Wi-Fi Impersonation Detection", in *IEEE Transactions on Information Forensics and Security* doi:10.1109/TIFS.2017.2762828
- [9] J. Zhang, X. Hu, P. Li, W. He, Y. Zhang, and H. Li, "A hybrid feature selection approach by correlation-based filters and svm-rfe," in *Proc. Pattern Recognition (ICPR)*, Stockholm, Sweden. IEEE, 2014, pp. 3684–3689.

