CLASSIFICATION OF FIREWALL LOG FILES USING SUPERVISED MACHINE LEARNING METHODS

SRI DANALAKSMI. P

(20PCS008)

Project Report Submitted

In partial fulfillment of the requirements for the Award of

Master of Science in Computer Science

DEPARTMENT OF COMPUTER SCIENCE

AVINASHILINGAM INSTITUTE FOR HOME SCIENCE AND

HIGHER EDUCATION FOR WOMEN

COIMBATORE – 641043

MAY – 2022

CLASSIFICATION OF FIREWALL LOG FILES USING SUPERVISED MACHINE LEARNING METHODS

SRI DANALAKSMI. P

(20PCS008)

Project Report Submitted

In partial fulfillment of the requirements for the Award of

Master of Science in Computer Science

DEPARTMENT OF COMPUTER SCIENCE

AVINASHILINGAM INSTITUTE FOR HOME SCIENCE AND HIGHER EDUCATION FOR WOMEN

COIMBATORE – 641043

MAY – 2022

Signature of the Head of the Department **Signature of the Supervisor**

Viva-voce Examination held on _____

Signature of the Examiner

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

I would like to express my sincere thanks to **God** Almighty, for his constant love and grace that he has shown upon me, which kept me in good health, and sound mind without which my project would not have reached a successful end.

I would like to express my deep sense of reverential gratitude and sincere thanks to **Dr.S.P.Thyagarajan, Chancellor,** Avinashilingam Institute of Home Science and Higher Education for Women, Coimbatore, for the opportunity given to me for undertaking this study and for providing all the needed facilities during my study.

I owe my great deal of gratitude to **Dr.V.Bharathi Harishankar,** Ph.D., FRSA **Vice-Chancellor,** Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for extending all resources that facilitated the conduct of the present study.

I express my gratitude to **Dr.S.Kowsalya, Registrar,** M.Sc., M.Phil., Ph.D. Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing all facilities necessary for the study.

I would express my boundless thanks to **Dr. G. Padmavathi,** M.Sc., M.Phil., Ph.D., and **Dean,** School of Physical Sciences & Computational Sciences and Principle Investigator of Center For Cyber Intelligence, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for granting the facility required.

I wish to place on record my deep sense of gratitude to **Dr. Vasantha Kalyani David,** M.Sc., M.Phil(Mathematics)., M.Phil(CS)., Ph.D., **Professor and Head**, Department of Computer Science for support and encouragement to complete the project.

I heartily thank my **project guide Dr. S. N. Geethalakshmi,** MCA., M.Phil., Ph.D, **Professor,** Department of Computer Science, for imparting the tremendous knowledge and well-timed support for the successful completion of my project. Her guidance and constant supervision helped me in completing the project in time.

I express my honorable thanks to my **project coordinator Dr. I. Elizabeth Shanthi,** M.Sc., M.Phil., Ph.D, **Professor,** Department of Computer Science, for her kind advice and knowledgeable suggestions which helped me to complete my project successfully.

I also like to extend my gratitude to **Ms. A. Roshni,** M.Sc., Department of Computer Science, project Guidelines CCI always supported me and nurtured me with valuable advice and profound belief in my work and abilities.

I have great pleasure in expressing my deep sense of gratitude to all the teaching and non-teaching staff members who stood behind the screen for the completion of the project.

Finally, I would like to thank my parents, family members, friends, and all well-wishers for their kind inspiration, blessings, and encouragement during the project time.

ABSTRACT

ABSTRACT

A firewall retains traffic entering and departing the domain it was supposed to protect. Some firewalls may provide information about the source and type of traffic entering the environment. A firewall's policy must be enhanced with a successful logging capability in order to be successful. The logging feature keeps track of how the firewall handles different sorts of traffic. Organizations can use the logs to find out things like Source IP addresses and destination IP addresses, protocols, and port numbers. Monitoring and analyzing log files can assist IT businesses improve the end-user reliability of their systems. Log files may consists of malicious texts, strings that tricks the users to hack the information. In generation of number of firewall logs per day, classifying the log files may help to observe more efficient, the number of unnecessary attributes can be minimized with the help of classification, resulting in a more efficient performance. The project title is 'Classification of firewall log files using supervised machine learning methods', the main intent of this project is to analyze and classify firewall logs which may consists of source port, destination port, bytes sent and received, etc., It checks that each data packet arrives on both sides of the firewall, it then decides whether or not to pass it. Firewalls can improve security even more by allowing quite well control over which system functions and processes have access to networking resources. The process starts with data collection followed by pre-processing techniques and main features to be selected to build a framework using supervised machine learning algorithms. In classification problems, the selection of appropriate and relevant dataset features plays a critical role. The feature selection approaches to improve the accuracy of classification system using Weka tool. Different classification techniques like Support Vector Machine, Naïve Bayes, Logistic Regression and K-Nearest Neighbor were adopted and their performance were analyzed.

CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

A firewall is a security access management point that controls access to computer networks and ensures safe network connectivity. A network firewall is a system or set of systems that uses pre-configured rules or filters to regulate access between different networks, a assured network and an unassured network. The outcomes of firewall rules can be audited, verified, and evaluated via monitoring. The analysing and classifying the firewall, checks and decides the packets to pass it or not. It can improve security purpose even more by allowing based on the required protocols.

1.1 ABOUT THE PROJECT

Firewall rules specifies the different types of network traffic which are permitted or not permitted.

Each firewall profile have different set of firewall rules that can't be changed. Only a few of the profiles allow then, it is compulsory to add new rules. There can also be a profile which allows to create new set of rules. The rules which are allowed to create should be created with pre-defined rules which is important aspect.

A firewall rule can be used to block the network traffic coming from the public Internet to private computer (inbound) or traffic coming from private computer to the public Internet (outbound). A rule can be deployed in both set of traffics at the identical time.

Firewalls are enhanced reliability that prohibit or limit illegitimate access to private networks linked to the Internet, especially intranets. The only traffic allowed on the network is defined by firewall standards; any other traffic intending to connect is blocked. At the network's front end, network firewalls perform as a communication medium between internal and external devices.

When properly configured, a firewall allows users to access any resources they need while keeping out unwanted people, hackers, and viruses. There are hardware and software firewalls available. In addition to limiting access to a protected computer and network, a firewall may track all traffic entering and leaving a network and regulate remote access to a private network using secure authentication certificates and logins.

Hardware firewalls can be purchased as standalone solutions for business use or as integrated components of routers and other networking devices. They are an essential part of any traditional security or network design. Almost all hardware firewalls have at least four network ports, allowing many devices to connect. For larger networks, a more comprehensive networking firewall solution is available.

Software firewalls can be installed on a computer or given by the operating system or network device maker. They can be modified and provide limited control over functionality and security features. A software firewall can protect a system from ordinary control and access attempts, but not from more sophisticated network breaches. Endpoint protection software includes things like firewalls. A firewall can be considered a first line of defence in terms of protecting private data, but it cannot be the only line of defence.

Any network service that is now running on a computer with a publicly visible IP – for example, if it is directly connected via ethernet – could become open to the outside world if firewalls are not in place. Cyber-attacks can affect any computer network that is connected to the internet. If these networks are not protected by a firewall, they become exposed to malicious attacks.

Other malware is designed with the sole intention of destroying data or bringing networks to a standstill. Hackers can easily disrupt a network if the IP address is public. Firewall classification is crucial so that attackers are less likely to infiltrate the network.

The survey's findings show that network engineering teams are devoting more time and effort to firewall maintenance, and that their duties are becoming more difficult. The majority of these chores, according to over 45 percent of respondents, are still done by hand. It's challenging to keep up with everything since most teams are dealing with a multi-vendor environment with inherent complexity.

5 Key Findings and Trends in Firewalls

1. Increased spending on firewalls and network security.

2. The use of several vendors has become the standard.

3. There is a demand for firewall engineers with particular knowledge.

4. Network automation is becoming more popular.

5. Firewall management is still mostly a manual operation.

1.2 MOTIVATION AND JUSTIFICATION

In generation of thousands of firewall logs per day, classifying the log files may help to observe the files and reduce the risk of threats. Thus, this project has its own space and necessity to be developed.

1.3 PROBLEM STATEMENT

To analyse and classify the firewall logs in order to handle the traffic during network observance to check that each data packet arrives and also to decide whether or not to pass it.

1.4 OBJECTIVE

To design a methodology to analyse and classify the firewall logs using different machine learning classifiers based on the action in their activities to apprehend the logs, and the performance of the model is estimated using different metrics.

CHAPTER 2

SYSTEM CONFIGURATION

CHAPTER 2
SYSTEM CONFIGURATION

2.1 HARDWARE SPECIFICATION

DEVICE NAME	:	LENOVO DESKTOP
PROCESSOR	:	Intel® Core™ i3 – 6006U CPU @2.00 GHz 1.99 GHz
RAM	:	4.00 GB
SYSTEM TYPE	:	64 – bit Operating System, *64 - based processor

2.2 SOFTWARE SPECIFICATION

ANACONDA PLATFORM

Anaconda Distribution is a Python and R in data science distribution that provides a package and environment administrator, as well as a collection of over 7,500 open-source tools. Anaconda is platform-independent, so then, may use it on Windows, Mac OS X, or Linux. Anaconda is a free and easy-to-use distribution with free community assistance.

Anaconda Embedded allows to give a smooth Python interface to consumers or utilise Anaconda behind the scenes to support offering as a product or service provider. Anaconda's premium repository, experts, and developers are available to all Embedded partners, as well as extra perks such as SLAs, co-marketing, and bespoke development opportunities.

Anaconda Server is the most recent version of our Anaconda repository. With support for all major operating systems, the repository serves as a central conda, PyPI, and CRAN packaging resource for desktop users, development clusters, CI/CD systems, and production containers.

ANACONDA NAVIGATOR

Anaconda Navigator is a graphical user interface (GUI) for desktops that comes with the Anaconda distribution that allows users to run programmes and manage conda packages, environments, and channels without having to use command-line commands. Navigator may look for packages on Anaconda Cloud or in a local Anaconda Repository, then install, run, and update them in a given environment. Windows, Mac OS X, and Linux are all supported.

Navigator comes with the following applications pre-installed:

• JupyterLab

• Jupyter Notebook

• QtConsole[19]

• Spyder

• Glue

• Orange

• RStudio

• Visual Studio Code

JUPYTER NOTEBOOK

Jupyter is an interactive web platform that allows academics to combine software code, computational results, explanatory prose, and multimedia materials in a single page. Computational notebooks are essentially lab notebooks for scientific computing. Rather of pasting DNA gels alongside lab instructions, researchers incorporate code, data, and text to document their computational methodologies.

Jupyter Notebook is an open-source web tool that lets a user, scientist, scholar, or analyst produce and share a document called a Notebook, which contains live programmes, documentation, graphs, plots, and visualisations.

Jupyter notebook is an excellent tool for analysing and analysing data. Jupyter notebook is used by data scientists to complete their daily tasks of data analysis. Remarkably, in a data science course, the notebook is the first tool that analysts are introduced to.

The main approach to open a Jupyter notebook (.ipynb) file is to use the Jupyter Notebook file browser interface. If then, try to open one in a plain text editor, it will appear as a JSON file instead of interactive code blocks. To use the Jupyter interface to examine a notebook, first open Jupyter Notebook (which will open in a browser window) and then open the file from within Jupyter Notebook. Unfortunately, there is no way to make Jupyter Notebook the default software application for double-clicking.ipynb files. A notebook is completely made up of cells, which are boxes that hold code or text that can be read by humans. Every cell has a kind, which can be chosen from the menu's drop-down selections.

For data scientists, Jupyter Notebook can be a useful tool, especially when it comes to education. Analysts can rely on this tool to write small code snippets and in situations when code production isn't necessary. So, the Jupyter notebook is helped for this project.

PYTHON IN JUPYTER NOTEBOOK

As this notebook is linked to the IPython kernel, it runs Python code. The most common is CPython, which is the official Python version, which can download from their website. It's also assumed 're working with Python 3.

WEKA TOOL

Here, Weka is a software term used to cover visualization tools and algorithms for feature selection and classification, as well as graphical user interfaces enabling quick access to these features. It consists of multiple algorithms and methods for increasing the accuracy level.

Weka's data mining capabilities include data preparation, clustering, classification, regression, visualisation, and feature selection, to name a few. Weka demands files to be named.arff and formatted using the Attribute-Relational File Format (ARFF). Weka's solutions are all based on the assumption that data is available in a single standard file or relation, with each data piece having a set of attributes.

2.2.1 LIBRARIES USED

- **import pandas as pd**

Pandas is usually using the pd alias. For pandas name, pd is used instead.

- **import matplotlib.pyplot as plt**
- **import numpy as np**

Matplotlibis helps in visualizing the data in the graphical format, with numpy.

- **from sklearn.preprocessing import StandardScaler** # feature scaling method

Remove the mean and scale to unit variance to standardise characteristics. By computing the relevant statistics methods on the samples on the train set, each feature are scaled and individually centered.

- **from sklearn.model_selection import train_test_split**

The two different subsets of datasets are training and testing datasets. From sklearn library required library sets are imported for model building.

- **from sklearn.metrics import accuracy_score**

The Categorization accuracy is perfectly calculated using accuracy_score. In multi-label classification, this function computes subset accuracy: the label set predicted for a sample must perfectly match the label set in y true.

- **from sklearn.metrics import classification_report**

A classification summary is a standard evaluation statistic in machine learning. It's used to show the precision, recall, F1 Score, and support of a trained classification model.

- **from sklearn import metrics**

Many loss, score, and utility methods are included in the sklearn. metrics module to assess classification performance.

- **from sklearn import svm**

Support vector machines (SVMs) are a class of supervised learning methods for classification, regression, and detection of outliers.

- **from sklearn.naive_bayes import GaussianNB**

Continuous valued features are accepted by Gaussian Naive Bayes, which models them all as Gaussian (normal) distributions.

- **from sklearn.linear_model import LogisticRegression**

LinearRegression creates a linear model to reduce the residual sum of squares between the dataset's observed targets and the linear approximation's anticipated targets.

- **from sklearn.neighbors import KNeighborsClassifier**

The K in the classifier's name represents the k nearest neighbours, where k is an integer number specified by the user. This classifier, as the name suggests, learns from the k closest neighbours.

CHAPTER 3

LITERATURE REVIEW

CHAPTER 3
LITERATURE REVIEW

Deepanshu Sharma and et. al (2021), examined the performance of machine learning classification models for firewall log data. It also attempted to improve classification by comparing the outcomes of all the models utilising heterogeneous ensemble classification models. The dataset was first put into the workspace, where data pre-processing was carried out. The feature selection was done once the multi labels were encoded into numeric values. The data was then scaled to make the selected features uniform in nature. The dataset was separated afterwards, and the model was built.

FatihErtam, et. al (2018), obtained log data through the firewall. The log logs were obtained from Firat University's Palo Alto 5020 Firewall device. The receiving log record is made up of thousands of records and is the product of approximately 30 seconds of recording. The properties of port, byte, packet, and time information are given priority in the receiving log. The action attribute of the class contains the values "allow", "deny", "drop", and "reset-both". The classifier SVM was chosen, and the activation functions are chosen. To compare the results, the precision, recall, F1 score, and ROC curves were employed.

HajarEsmailAsSuhnbani, et.al (2019), have analysed a firewall log dataset using data mining approaches such as feature selection and redundancy removal. Then they speed up the process by employing machine learning techniques such Naive Bayes, kNN, One R, and J48 in WEKA. In addition, they also compared the accuracy and F-measure of these algorithms' classification performance. And they also performed a 10-fold cross-validation test. In terms of accuracy, the required model has performed admirably.

Robert Winding, et. al (2006), Several analysis methodologies were used to investigate the system anomaly detection. Firewall log data is also subject to this type of statistical analysis. Boxplots were used to choose meaningful features from different data, then after look at all the distribution of selected features, and spot individual data records, and examine outliers for some features. The destination perspective vector was clustered, which led to the creation of a classification model.

ShridharAllagi, RashmiRachh, et. al (2019), used a network log data collection generated by a corporation for research. Several million entries make up the log data set. Only 5% of data appears to be abnormal behaviour. This exposes the system to an issue of class imbalance, in which one sort of cluster is insignificant in comparison to others. They suggest using k-means (k=2) to produce two clusters: normal and abnormal, as well as a self-organizing feature map (SOFM) of an artificial neural network with neurons in the input and output layers. Our focus is on linguistic information in log data, with a particular emphasis on bigram words (for example, "ACCESS DENIED," "401 ERROR," "INVALID PASSCODE," and so on). They used a supervised machine learning has approach and

tested the system against a large dataset of network logs. The findings of the trial indicated that the accuracy of predicting deviant conduct was high and acceptable.

Firewall means

Firewall rules specifies the different types of network traffic which are permitted or not permitted.

Each firewall profile have different set of firewall rules that can't be changed. Only a few of the profiles allow then, it is compulsory to add new rules. There can also be a profile which allows to create new set of rules. The rules which are allowed to create should be created with predefined rules which is important aspect.

A firewall rule can be used to block the network traffic coming from the public Internet to private computer (inbound) or traffic coming from private computer to the public Internet (outbound). A rule can be deployed in both set of traffics at the identical time..

Basics of Machine Learning

Machine learning is a technique for teaching robots to do tasks such as forecasts, suggestions, and guesses based on historical data or previous experience.

Machine Learning trains machines to behave like humans by using past experience and projected data to train them.

The three main aspects of Machine Learning:

- The core problem in which we are interested is defined as a task. This task/problem can be linked to predictions, suggestions, and estimates, among other things.
- Experience is described as the ability to learn from historical or previous data and apply that knowledge to estimate and resolve future tasks.
- Performance is defined as a machine's ability to solve any machine learning job or problem and produce the best possible result. The nature of machine learning problems, however, has an impact on performance.

There are Different types of Machine Learning, they are:

- Supervised Machine learning
- Unsupervised Machine Learning
- Reinforcement Learning

- Semi-Supervised Learning

Supervised Machine Learning

When a machine has sample data, such as input and output data with correct labels, supervised learning can be used. Using some labels and tags, correct labels are utilised to check the model's validity. With the use of prior experience and labelled samples, the supervised learning technique allows us to anticipate future events. It analyses the known training dataset first, then introduces an inferred function that provides output value predictions. Furthermore, it predicts errors throughout the learning process and uses algorithms to fix such faults.

Regression

A strategy for generating a single data point utilizing training sets called regression.

Logistic Regression

The inferential analysis of logistic regression is used to estimate discrete values from a two or more independent variables. By fitting the experimental data to a probit function, it aids in forecasting the probability of a risk. As a result, logistic regression is another name for it. Because it forecasts probability, its output current is between 0 and 1.

Classification

The process of organizing output into diverse factions is referred to as "classification." When an algorithm attempts to characterise data into two discrete categories, it is known as binary classification. Switching between more than two classes is considered to as multiclass classification.

Multiclass Classification

In learning algorithms, multiclass or multinomial classification entails the process of categorising instances into one of three or much more classes (classifying instances into one of two classes is called binary classification).

Naïve Bayes

When dealing with complex datasets, the Naive Bayesian model (NBN) is simple to build and quite useful. This approach use direct acyclic graphs with one parent and multi children.

It suggests that parental nodes are self-contained whenever they are separated from their parents.

Support Vector Machine

SVM machines are strongly tied to kernel functions, which are a significant topic in most learning tasks. The kernel framework and SVM are used in a variety of applications. Pattern recognition, epidemiology, and multimedia information retrieval are all covered.

To learn more about the classification problems based on firewall log data, few publications from reputable websites are studied.

The base papers are listed below:

The paper named, ‘Classification of Firewall Logs using Supervised Machine Learning’ is the base paper for this project. But different algorithms are used based on other papers.

3.1 BACKGROUND STUDY

S. No	Paper Name	Author Name	Algorithms and methods used	Result	Observation
1	Decision Tree for Multiclass Classification of Firewall Access (IJIES) (2015)	AL-Behadili, HayderNaserKhraibet	SVM, ANN, Multi class classifier, OneR and ZeroR.	Classification accuracy: DT -99.839, SVM – 97.474, OneR – 97.326, ANN – 98.712, Multi class classifier – 99.064, ZeroR – 57.437	In comparison to six machine learning algorithms, namely SVM, ANN, Multi class classifier, PSO, OneR, and ZeroR, the efficiency of the DT method determines the best classification accuracy with the highest competence.
2	Analysis of Network log data using Machine Learning(IEEE) (2019)	Allagi, S., & Rachh, R., Jain College of Engineering, Karnataka, India	SVM, K Means, Decision Tree, Naïve Bayes, SOFM	K Means and SOFM, 8:2, achieved 97.5 % accuracy, only text data has been analyzed	Here K means and SOFM obtained better results
3	Classification of Firewall Logs using	As-Suhbani, H. E., &	Naïve Bayes, KNN, One R,	KNN obtained	May be because of

	Supervised Machine Learning(IJCSE) (2019)	Khamitkar, S. S.R.T.M University, India	J48	better results with accuracy 99.8736%, F-measure – 0.999, ROC Area – 0.975	KNN's high variance in real time dataset
4	Classification of Firewall Log Files with Multiclass SVM(IEEE) (2018)	Ertam, F., & Kaya, M. Firat University, Turkey	Multiclass SVM - Linear, Sigmoid and Radial basis Function	SVM RBF worked better with 76.4%, Precision 63.0% and Recall 97.1%	It's a One class SVM, it RBF performed non linear transformation before they fed for classification, over the input vector
5	A Distance-Based Method to detect Anamalous Attributes in Log Files(IEEE) (2012)	Hommes, S., State, R., & Engel, T. University of Luxemburg	Chart tracking algorithm and WECO rules - to detect deviations from a stable process by comparing data to predetermined anomalous patterns on a regular basis	Dataset 1 – data are relevant for stealing by accessing internal database server Dataset 2 – Huge volume of attempted SMTP connections triggered	Identifies most suspicious activities, and human operator confirm the findings and the difference is computed, but labelled dataset will work more

6	The Method of Detecting Malware-Infected Hosts and proxy logs(IEICE) (2021)	Kamiya, K., Aoki, K., Nakata, K., Sato, T., Kurakami, H., &Tanikawa, M. NTT Secure Platform Laboratories	Frequency based technique and sort based method	When compared to standalone Proxy-based detection, the method contributes a 6% improvement in accuracy. As a result, we demonstrate that multi-layer analysis can help to improve malware detection.	The frequency-based technique, which detects malicious traffic if it matches a single value in a harmful list and Sort-based method for detecting whether host traffic matches more than a certain number of elements from a malicious list.
7	Analysis of Log files Intersections for Security Enhancement(IEEE) (2006)	Kowalski, K., & Beheshti, M, CA - 90747	No algorithms, since it's a offline analysis and also its just a design	Out of 19977 IP, 2356 different IP address of hosts tried to access both machines, port 80 once, others are intruders probing different port	They found the existence of same IP address in two different intersection of log files, (desktop firewall and webserver firewall-mainly targets

				numbers	port 80 which is HTTP and HTTPS)
8	Semantic Interpretation of Structured Log Files(IEEE) (2016)	Nimbalkar, P., Mulwad, V., Puranik, N., Joshi, A., &Finin, T., USA	General Splitting algorithm for further process (2 datasets)	Precision - 0.45 and Recall - 0.89 for original dataset Precision – 0.55 and Recall – 0.87 for synthetic dataset	Auto - generated semantic representation of schema and contents and generating a table like structure for log files for each columns and differentiating it
9	Optimized Classification of Firewall Log Data using Heterogenous Ensemble Techniques(IEEE) (2021)	Sharma, D., Wason, V., &Johri, P., Galgotias University, UP, India	KNN, Logistic Regression, SVM, Decision tree classifier and Stochastic gradient descent classifier	Decision tree obtained with 0.87 precision, Recall of 0.86 and F – Score 0.86, Stacking ensemble classifier worked well with precision value 91%	Decision tree helps in automatic feature extraction, so it worked better
10	System Anomaly	Winding, R., Wright,	JRIP - It's a basic	Classification	Clustering performed to

	Detection: Mining Firewall Logs(IEEE) (2006)	T., & Chapple, M	incremental reduced-error pruning algorithm., For building a classification model.	models: Correctly classified instances: 99.9167% Incorrectly classifies instances: 0.0833%	destination perspective vector which lead to classification model using JRIP

CHAPTER 4

METHODOLOGY AND IMPLEMENTATION

CHAPTER 4

METHODOLOGY

PHASE 1

DATA ACQUISITION

PHASE 2

DATA PRE-PROCESSING

Label encoding, Train/Test Split

Standardization, EDA

PHASE 3

FEATURE SELECTION (Filter)

PHASE 4

MODEL BUILDING (SVM, Naïve Bayes, Logistic Regression, KNN)

PHASE 5

COMPARITIVE ANALYSIS OF ALGORITHMS

PHASE 6

RESULTS AND OUTCOME

IMPLEMENTATION

4.1 PHASE 1 - DATA ACQUISITION

The process of acquiring data from relevant sources before it is saved, cleaned, pre-processed, and used in other processes is referred to as "data acquisition." It is the process of acquiring critical business information, converting it into the proper business form, and loading it into the relevant system.

"Data acquisition is the act of sampling data that measure real-world physical parameters and converting the resulting samples into digital numeric values that a computer can control."

DATASET LINK

https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data

SOURCE OF THE DATASET

FatihErtam, fatih.ertam **'@'** firat.edu.tr, Firat University, Turkey.

4.1.1 DATASET DETAILS

This data set was collected from the network traffic records from Firat University's Palo Alto 5020 Firewall device. The dataset is collected from UCI Repositary.

Data Set Information

There are 65533 records and 12 features in total. The Class is ‘Action feature’. So, there are 4 classes in total. These are allow, action, drop and reset-both classes.

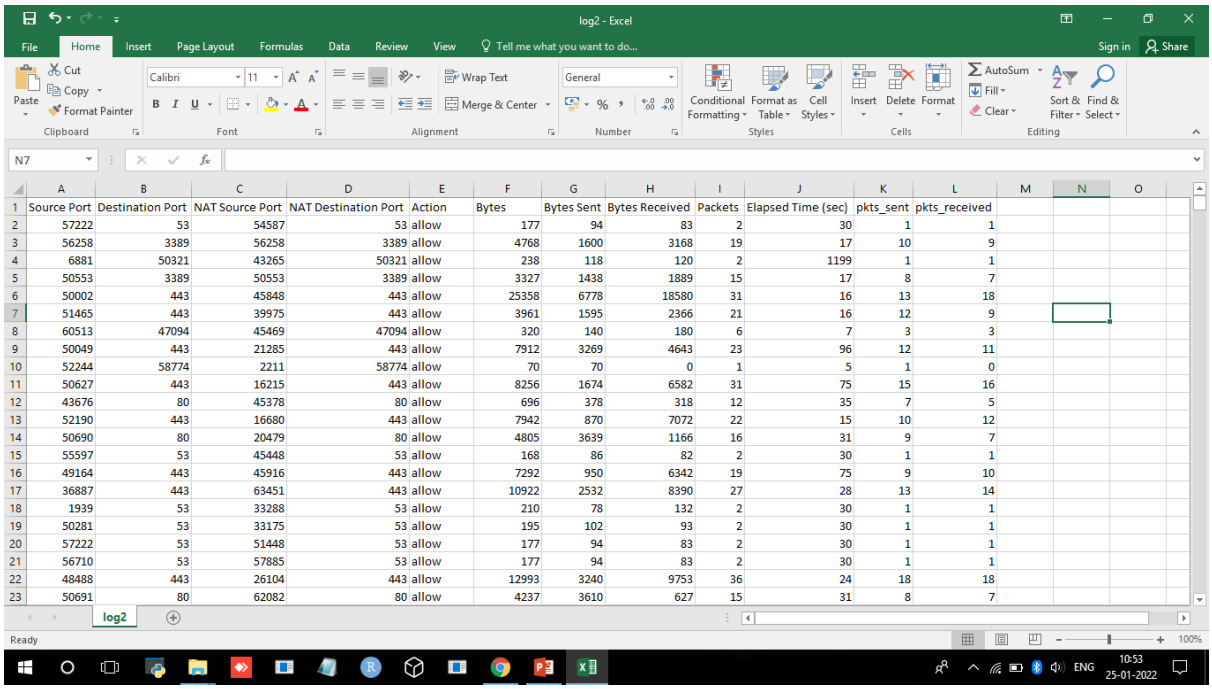
Attribute Information

Source Port, Destination Port, NAT Source Port, NAT Destination Port, Action, Bytes, Bytes Sent, Bytes Received, Packets, Elapsed Time (sec), pkts_sent, pkts_received

Source:

FatihErtam, fatih.ertam '@' firat.edu.tr, Firat University, Turkey.

PREVIEW OF A DATASET



Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received
57222	53	54587	53	allow	177	94	83	2	30	1	1
56258	3389	56258	3389	allow	4768	1600	3168	19	17	10	9
6881	50321	43265	50321	allow	238	118	120	2	1199	1	1
50553	3389	50553	3389	allow	3327	1438	1889	15	17	8	7
50002	443	45848	443	allow	25358	6778	18580	31	16	13	18
51465	443	39975	443	allow	3961	1595	2366	21	16	12	9
60513	47094	45469	47094	allow	320	140	180	6	7	3	3
50049	443	21285	443	allow	7912	3269	4643	23	96	12	11
52244	58774	2211	58774	allow	70	70	0	1	5	1	0
50627	443	16215	443	allow	8256	1674	6582	31	75	15	16
43676	80	45378	80	allow	696	378	318	12	35	7	5
52190	443	16680	443	allow	7942	870	7072	22	15	10	12
50690	80	20479	80	allow	4805	3639	1166	16	31	9	7
55597	53	45448	53	allow	168	86	82	2	30	1	1
49164	443	45916	443	allow	7292	950	6342	19	75	9	10
36887	443	63451	443	allow	10922	2532	8390	27	28	13	14
1939	53	33288	53	allow	210	78	132	2	30	1	1
50281	53	33175	53	allow	195	102	93	2	30	1	1
57222	53	51448	53	allow	177	94	83	2	30	1	1
56710	53	57885	53	allow	177	94	83	2	30	1	1
48488	443	26104	443	allow	12993	3240	9753	36	24	18	18
50691	80	62082	80	allow	4237	3610	627	15	31	8	7

FIG. 4.1 DATASET PREVIEW

DATASET ATTRIBUTES DESCRIPTION

Source Port:

The unregistered port range is used to generate source ports at random.

Destination Port:

The major Internet applications, such as Web and email, may use "well-known ports" (0-1023) as destination ports.

NAT Source Port:

Source NAT modifies the source address in a packet's IP header. In the TCP/UDP headers, it may also modify the source port. For packets leaving network, the most common use is to turn a private Source port into a public Source port.

NAT Destination Port:

Destination NAT modifies the IP header of a packet's destination address. It may also change the target port of TCP/UDP headers. This is typically used to redirect packets with a public address/port destination within a network to a private IP address/port.

Bytes:

Number of bits per second whether it is sent or received.

Bytes Sent:

The amount of data received through the interface is referred to as "Bytes In" (i.eto host from the network).

Bytes Received:

The amount of data delivered through that interface is referred to as "Bytes Out" (from host to the network).

Packets:

A firewall works in a similar manner, examining each packet of data to determine where it came from, where it is going, or both, and deciding whether it should be accepted and permitted to continue on its route, or refused and dropped.

Elapsed Time (sec):

This is the number of seconds since the start of the session (measured by DP).

pkts_sent:

Each packet goes through the network based on a series of hops. Each packet is routed through a local Internet service provider (ISP), a service that supplies network access.

pkts_received:

This implies that each webpage then, receive is delivered in a series of packets, and each email then, send is delivered in a series of packets.

DATASET LABELS DESCRIPTION

Action is the class given in the dataset, as it consists of four classes the descriptions of each class is given below.

ALLOW

The allow action permits specific traffic on the network to pass through the firewall. The source and receiver of traffic's IP addresses, as well as the type of traffic, create the rules**.**

DROP

If the circumstance does not match the restrictions set forth by firewall protocols, the packet is silently dropped without an acknowledgement being sent to the server.

DENY

When a user or group is denied a specific permission in the security settings of a firewall protocol, the acknowledgement is sent to resend the packets.

RESET-BOTH

Drops matched packets, issues a TCP RST to the server as well as client, and adds a threat log record.

So, this dataset is completely based on multi-class classification based on four classes discussed above.

4.2 PHASE 2 - DATA PRE-PROCESSING

Data preparation is a key component in Machine Learning since it improves data integrity and makes it easier to extract useful insights from the data. Data pre-processing in Machine Learning refers to the act of organising and early initiation data in order to prepare it for the creation and training of Machine Learning models. The first stage in generating a Machine Learning model is data preparation, which symbolizes the start of the process. Data from the real world is frequently fragmented, inconsistent, erroneous (owing to errors or outliers), and lacking in precise attribute values and trends. This is where information gathering comes in; it cleans, prepares, and organizes raw data so that Machine Learning models can work with it.

DATA PRE-PROCESSING STEPS

- Label Encoding
- Train and Test Split
- Standardization

4.2.1 Label Encoding

Label encoding is a popular embedding mechanism for numerous data combinations. In this system, each label is assigned a unique integer based on chronological order. The process of converting labels into numeric format so that machineries can read them is known as labelling encoding. Machine learning algorithms understand how to use those labels. It is an important pre-processing step for the organized dataset in supervised methods. If, just deal with structured data and the data gathered is a combination of continuous and categorical variables, most Machine Learning algorithms will not understand or be able to work with categorical variables. Label encoding in Python can be done with the Sklearn Library. Sklearn is an incredible resource for translating categorical feature levels to numerical values. Label Encoder encodes labels using a value between 0 and n classes-1, where n is the number of different labels. When a label is repeated, the same priority is calculated to it as before the train and test split.

4.2.2 Train and Test Split

The dataset is partitioned in the following stage of machine learning data pre-processing. A Machine Learning model's dataset should be split into two parts: training and testing.

The training set is a segment of a dataset that is used to train a machine learning model. A test set, on the other hand, is a subset of the dataset used to evaluate the machine learning model. The ML model uses the test set to predict outcomes.

Sometimes, the dataset is partitioned into 70:30 or 80:20 ratios. This means that either two third or 80% of the data were included to train the model, with the remaining 30 percent or 20 percent left out. Depending on the dataset, the procedure for separating it varies.

- X_train - data features used for training set
- X_test - data features used for testing set
- y _train - training the data, dependent variables
- y_test - testing the data independent variable.

As a result, the train test split() function has four parameters, the first two of which are for data arrays. The test size function determines the size of the test set. The test size parameter could be 5,.3, or.2; it defines the division ratio of the training and test sets. The third option, "random state," gives the seed for a random generator, ensuring that the result is always the same. The ratio is 8:2 in this case.

4.2.3 Standardization

Feature scaling is required by machine learning approaches that determine data distances. Normalization and standardisation are two scaling methods that centre data around the mean and have a single standard deviation. As a result, the attribute's mean is 0 and the underlying distribution has a unit standard deviation. There are no boundaries to how far standardisation can go. As a result, even if the data contains outliers, normalisation has no effect. Data standardisation is the process of putting data into a standard format so that users can analyse and evaluate it. Data is derived from a variety of sources, notably large databases, lakes, cloud services, and databases, by most enterprises. Data from multiple sources, on the other hand, can be problematic if it isn't uniform, leading to future issues (e.g., when then, use that data to produce dashboards and visualizations, etc.).

Data standardisation is necessary for a variety of reasons. It first and foremost assists in the definition of well-defined elements and attributes, resulting in a comprehensive data catalogue. Whatever insights or difficulties you're trying to solve, a good understanding of

data is a must-have first step. We need to put the data into a standard format with logical and consistent definitions in order to get through. Metadata, or labels that explain what, how, why, who, when, and where data is kept, will be created using these guidelines. This is the foundation for the data standardisation technique. Standardizing the way data is labelled will improve access to the most relevant and up-to-date information in terms of correctness. Statistics and reporting will be easier as a result. In terms of security, careful cataloguing serves as the foundation for a sophisticated authentication and authorisation process that applies security constraints to data objects and data users as needed.

4.2.4 Exploratory Data Analysis

Exploratory Data Analysis is a way of evaluating data for ordering to summarize their essential characteristics, which frequently involves the use of statistical graphics and other data visualization techniques.

EDA provides support to data scientists in a variety of ways, including:

- ✓ Improving data comprehension
- ✓ Recognizing different patterns in data
- ✓ Clarifying the problem statement

EDA is divided into four categories:

- **Univariate Non-graphical**
- **Univariate graphical**
- **Non-graphical multivariate**
- **Multivariate graphical**

Univariate Non-graphical

Because, just utilising one variable to explore the data, this is the simplest sort of data analysis. The typical goal of univariate non-graphical EDA is to comprehend sample distribution and data and generate population observations. The analysis also includes the detection of outliers. The following are some of the characteristics of population distribution:

The central tendency, often known as the distribution's position, has to do with typical or middle values. The statistics known as mean, median, and sometimes mode are widely used

to measure central tendency, with mean being the most prevalent. The median may be selected when the distribution is skewed or when outliers are a concern.

Spread is a metric that indicates from the centre we should look for information values. Two relevant measurements of spread are the quality deviation and variance.

The variance is the root of the variance since it is the mean of the squares of the individual deviations. The distribution's skewness and kurtosis are two more important univariate features. Skewness is a measure of asymmetry when compared to a normal distribution, whereas kurtosis is a more subtle measure of peakedness.

Univariate graphical

Unlike the non-graphical method, the graphical method gives a complete picture of the data. The three main methods of analysis for this type of data are histograms, stem and leaf plots, and box plots. The histogram shows the total number of cases for a given set of values. The stem and leaf plot depicts the distribution's shape as well as the data values.

Multivariate non-graphical

The multivariate non-graphical type of EDA uses cross-tabulation or statistics to show the link between various variables of data.

Multivariate graphical

This type of EDA shows how two or more sets of data are related. A bar chart with each group representing a level of one variable and each bar within each group representing levels of other variables.

EDA Tools

Python: When using EDA, Python can be used to find the missing value in a data set. Other procedures that can be conducted include data description, controlling outliers, and getting insights through visualisations. It appeals to EDA because of its built-in high-level data structure, as well as dynamic type and binding. Analyzing a dataset is a time-consuming and labor-intensive process. Python provides a lot of open-source modules that can automate and save time throughout the EDA process**.**

4.3 PHASE 3 - FEATURE SELECTION USING WEKA TOOL (FILTER)

FEATURE SELECTION

The purpose of this framework in machine learning is to find the best set of features for developing useful models of the phenomena being studied. Feature selection strategies in machine learning can be classified into the following categories:

Supervised Approaches: These techniques can be used to determine the important features in supervised models like classification and regression using labelled data.

Unsupervised Approaches: These methods can be utilized with unlabeled data.

FEATURE SELECTION USING WEKA TOOL

When a database has a huge number of attributes, some of them will be irrelevant to the analysis, looking for. As a result, removing undesirable attributes from the dataset has become a necessary stage in creating a good machine learning model. We can visually review the complete dataset and eliminate the irrelevant features. For databases with a large number of properties, this could be a yet over. Fortunately, WEKA has an automated feature selection tool.

Weka is a software concept that describes to feature selection and categorization visualisation tools and algorithms, as well as graphical user interfaces that allow quick access to these features.

Weka's data mining functions include data preparation, clustering, classification, regression, visualisation, and feature selection, to name a few. Weka demands files to be named.arff and formatted using the Attribute-Relational File Format (ARFF). Weka's solutions are all based on the assumption that data is available in a single standard file or relation, with each data piece having a set of attributes.

- Simple CLI is a command-line interface that allows Weka commands to be executed directly.
- Explorer is a data exploration environment.

- Experimenter is a freeware that allows to run experiments and statistical tests on different learning schemes.
- Knowledge Flow is a Programming language interface for configuring and executing machine learning experiments.

After changing the file format, the dataset will be loaded into the environment. The data will be loaded to the environment. Then SELECT ATTRIBUTES tab, which is inside the Explorer section to be preferred for the feature selection process.

The feature selection process is split into two parts:

- Attribute Evaluator
- Search Method

There are various techniques to pick from in each section.

Attribute Evaluator

The attribute evaluator is a tool for evaluating each attribute (also known as a column or feature) in dataset in relation to the output variable (e.g. the class).

Search Method

The search method is a technique for browsing or experimenting with different combinations of attributes in a dataset to arrive at a short list of desired features.

For some Attribute Evaluator techniques, different Search Methods are necessary. For example, the CorrelationAttributeEval method in the next section can only be used with a Ranker Search Method, which ranks the results after evaluating each attribute. When selecting alternative Attribute Evaluators, the interface may prompt them to change the Search Method to one that is compatible with the method they've chosen.

The Attribute Evaluator as well as the Search Method approaches can be customised. To access the technique's setting data, click on the technique's name once it has been selected.

Search Methods in Weka Tool

- **BestFirst**

To search the space of attribute subsets, it employs greedy hillclimbing with a backtracking facility. The number of consecutive non-improving nodes allowed determines how much backtracking is done. Start with an empty set of attributes and search forward, or start with a complete set of attributes and search backward, or start anywhere and search both directions (by considering all possible single attribute additions and deletions at a given point).

The Ranker

Individual evaluations are used to rank attributes. When combined with attribute evaluators, it's a strong option (ReliefF, GainRatio, Entropy etc).

- **GreedyStepwise**

Searches the space of attribute subsets in a greedy forward or backward manner.

Attribute Evaluators in Weka Tool

- **InfoGainAttributeEval**

Measures the information gain with respect to the class to determine the value of a particular feature.

Formula

InfoGain(Class,Attribute) = H(Class) - H(Class | Attribute).

- **GainRatioAttributeEval**

Measures the gain ratio with regard to the class to determine the value of an attribute.

Formula

GainRatio(Class, Attribute) = (H(Class) - H(Class | Attribute)

- **CorrelationAttributeEval**

Measures the correlation (Pearson's) between an attribute and the class to determine the value of best feature.

- **OneRAttributeEval**

Using the OneR classifier, determines the value of an attribute.

- **PrincipalComponents**

Analyzes and transforms data using principle components analysis. When combined with a Ranker search, this is a powerful combination. Dimensionality reduction is performed by picking a relatively large number of eigenvalues to account for a given percentage of the heterogeneity in the original data (default 0.95). (95 percent). Attribute noise can be filtered by translating to the PC space, eliminating some of the worst eigenvectors, and then converting back to the original space.

- **ClassifierAttributeEval**

Using a user-specified classifier, determines the value of an attribute.

- **SymmetricalUncertAttributeEval**

Measures the symmetrical uncertainty with regard to the class to determine the value of an attribute.

- **WrapperSubsetEval**

Using a learning approach, evaluates attribute sets

Search method and Attribute Evaluator used in this project

Under the search method and attribute evaluator, there will be several options in which three search methods and attribute evaluators which comes under filter feature selection method, are selected to evaluate the best features and they are:

- **Ranker +InfoGainAttributeEval**
- **Ranker +CorrelationAttributeEval**

The Ranker

Individual evaluations are used to rank attributes. When combined with attribute evaluators, it's a strong option (ReliefF, GainRatio, Entropy etc).

- **InfoGainAttributeEval**

Measures the information gain with respect to the class to determine the value of an attribute.

Formula

InfoGain(Class,Attribute) = H(Class) - H(Class | Attribute).

- **CorrelationAttributeEval**

Measures the correlation (Pearson's) between an attribute and the class to determine its value to select a best feature.

The best five features were chosen based on ranking using the ranker from the search method and InfoGainAttributeEval, CorrelationAttributeEval from the Attribute Evaluator. The features **Source Port, Destination Port, NAT Source Port, NAT Destination Port, and Bytes** were chosen as they provided the best accuracy when compared to other features. These distinct features were employed to develop a model.

4.4 PHASE 4 - MODEL BUILDING

To achieve its purpose, a machine learning model learns and generalises from training data before applying that knowledge to new data it has never seen before to provide predictions and different classifications.

In machine learning, creating a schematic includes generalising and learning from training data. Using new data, the constructed machine learning model is then utilised to make predictions and obtain results. The model won't be developed if there is no enough data.

The model which is to be created can be a regression or a classification model based on the Y variable, which is the target variable. There should be a regression model if the target variable has a numerical value. If the target variable's data type is qualitative, and should create a classification model.

The accuracy of a classification machine learning model is the most essential metric. The ratio of total number of perfect predictions to total number of data points in the testing dataset is known as accuracy. Accuracy is one approach to evaluate a model's performance, but there are others. When compared to other metrics, accuracy provides the most accurate representation of a model's performance for a particular dataset. The utilized metrics used to assess a model's performance. Model Building is the building block of machine learning to provide reliable accuracy.

There are four algorithms used to build a model, they are:

- SUPPORT VECTOR MACHINE
- NAÏVE BAYES CLASSIFIER
- LOGISTIC REGRESSION CLASSIFIER
- K – NEAREST NEIGHBOUR CLASSIFIER

4.4.1 SUPPORT VECTOR MACHINE

Support vector machines (SVMs, also known as support vector networks) are supervised learning models with related learning algorithms for classification and regression analysis in machine learning. Though we might also argue regression difficulties, categorization is the best fit. The purpose of the SVM methods is to determine a higher dimensional space in an N-dimensional region that group data points clearly. A Support Vector Machine (SVM) is a classifier that has a segregated hyper-plane. To put it another way, the algorithm is used to generate an ideal hyper - plane that classifies new samples based on labelled training data (supervised learning). In addition to linear classification, SVMs may do non-linear classification by implicitly interpreting their inputs into high-dimensional feature spaces. Given a set of training examples that are individually designated as belonging to one of two categories, an SVM training algorithm is used to generate a model that distributes classifier models to one of two categories, making it a non-probabilistic binary linear classifier.

A Support Vector Machine (SVM) is a hyperplane-segregated classifier. In other words, the technique is utilized to create an appropriate hyper-plane that classifies new samples using labelled training data (supervised learning). SVMs may also undertake non-linear classification by implicitly interpreting their inputs into high-dimensional feature spaces, in addition to linear classification. An SVM simulation environment is used to produce a model that redistribute classifier models to one of two categories, making it a non-probabilistic binary linear classifier, given a series of training examples that are individually identified as belonging to one of two categories.

Kernel with a straight line Any of two observations shall be normal dot producted using a linear kernel. The sum of the product of each input values pairs is the multiplication of two vectors.

Polynomial Kernel is a particular kind of polynomial kernel. A more generalised variant of the polynomial kernel is the linear kernel. If the input space is curved or nonlinear, the polynomial kernel can tell. Radial Basis Function Kernel.

The radial basis function kernel is a prominent kernel function in support vector machine classification. RBF has the ability to map an endlessly cartesian coordinates into an input space.

Supporting Characteristics

The data points which are very near to the hyperplanes are called support vectors. These locations will further define the dividing line by computing margins. These considerations are more pertinent to the classifier's design.

Hyperplane

A hyperplane is a decision plane that splits a selection of items into several classes.

Margin

The distance between the two lines on the class points that are closest to each other is known as a margin. The distance between the line and the support vectors or closest points is calculated. A larger gap between the classes is considered a good gap, whereas a smaller gap is considered a bad gap.

In this support vector machine the required libraries are imported and the svc function is created and finally the following metrics of the model are calculated:

- ✓ Train and Test Accuracy
- ✓ Classification metrics
- ✓ Accuracy
- ✓ Precision
- ✓ Recall
- ✓ F1 Score

As, the train and test split is 80:20 ratio, the testing accuracy is taken as the final accuracy level for SVM model. This is a multi-class classification so that, for Precision, Recall and F1 Score we calculated for their averages micro, macro and weighted for their corresponding metrics.

The model is built followed by the steps in the pseudocode to be discussed below.

PSEUDOCODE FOR SVM CLASSIFIER

STEP 1 **:** Start the process.

STEP 2 **:** Import necessary libraries.

STEP 3 **:** Create a function SVC with kernel linear and give necessary parameters.

STEP 4 **:** Fit those function in trained dataset.

STEP 5 **:** Create another function y_predict to store the test dataset predicted accuracy.

STEP 6 **:** Print classification report to display precision, recall, f1-score and support.

STEP 7 **:** Display the train and test accuracy using accuracy_score.

STEP 8 **:** Display precision, recall and f1-score using evaluation metrics such as micro, macro and weighted average.

STEP 9 **:** The output is evaluated.

STEP 10 **:** Stop the process.

4.4.2 NAÏVE BAYES CLASSIFIER

A classifier is a machine learning model that distinguishes objects based on the qualities of particular variables. It's a Bayes theorem-based classification algorithm. Membership probabilities are anticipated for each class, such as the frequency of data points being correlated to that class.

The Bayes' Theorem is a simple technique that can be used to estimate posterior probability. A measure of the possibility of an event occurring if another event has already occurred is called conditional probability (by assumption, presumption, statement, or evidence).

$$P(A|B) = \frac{P(B|A) - P(A)}{P(B)}$$

P(A|B), also known as posterior probability, informs us how likely A is if B occurs. We can write P(B|A) when we know how often B occurs when A occurs, and how often A occurs on its own, written P(A), and how probable B occurs on its own, written P(B) (B).

The essential Nave Bayes assumption is that each feature contributes equally and independently to the outcome.

Predicting the class of the test data set is simple and quick. It can also anticipate many classes. A Naive Bayes classifier outperforms competing models such as logistic regression when the assumption of independence is met, and it takes less training data.

In this Naïve Bayes Classifier the required libraries are imported and the GaussianNB function is created and finally the following metrics of the model are calculated:

- ✓ Train and Test Accuracy
- ✓ Classification metrics
- ✓ Accuracy
- ✓ Precision
- ✓ Recall
- ✓ F1 Score

As, the train and test split is 80:20 ratio, the testing accuracy is taken as the final accuracy level for Naïve Bayes model. This is a multi-class classification so that, for Precision, Recall

and F1 Score we calculated for their averages micro, macro and weighted for their corresponding metrics.

The model is built followed by the steps in the pseudocode to be discussed below.

PSEUDOCODE FOR NAÏVE BAYES CLASSIFIER

STEP 1 : Start the process.

STEP 2 : Import necessary libraries.

STEP 3 : Create a function GaussianNB, give necessary parameters.

STEP 4 : Fit those function in trained dataset.

STEP 5 : Create another function y_nbforecast to store the test dataset predicted accuracy.

STEP 6 : Print classification report to display precision, recall, f1-score and support.

STEP 7 : Display the train and test accuracy using accuracy_score.

STEP 8 : Display precision, recall and f1-score using evaluation metrics such as micro, macro and weighted average.

STEP 9 : The output is evaluated.

STEP 10 : Stop the process.

4.4.3 LOGISTIC REGRESSION CLASSIFIER

A statistical method for creating machine learning models employing a dichotomous (i.e. binary) dependent variable is logistic regression.

One of the most intriguing Supervised Machine Learning Algorithms in Machine Learning is Logistic Regression. Logistic Regression, as well as its related relatives, Multinomial Logistic Regression, allow us to forecast if an observation belongs to a specific class using a simple, easy-to-understand, and follow technique. Data and the relationship between one dependent variable and one or more independent variables is described using logistic regression. Nominal, ordinal, or interval independent variables are all possible. Logistic Algorithms works on this basis:

- The dependant variable must be binary in a binary logistic regression.
- The desired outcome for a binary regression should be represented by the factor level one of the dependent variables.
- Only the most important variables should be included in the analysis.
- Independent variables are those that are not dependent on one another. This implies that the model should be multi-collinear to a minimum.
- The log odds are proportional to the independent variables.

In this Logistic Regression Classifier the required libraries are imported and the LogisticRegressionfunction is created and finally the following metrics of the model are calculated:

- ✓ Train and Test Accuracy
- ✓ Classification metrics
- ✓ Accuracy
- ✓ Precision
- ✓ Recall
- ✓ F1 Score

As, the train and test split is 80:20 ratio, the testing accuracy is taken as the final accuracy level for Logistic Regression model. This is a multi-class classification so that, for Precision, Recall and F1 Score we calculated for their averages micro, macro and weighted for their corresponding metrics.

The model is built followed by the steps in the pseudocode to be discussed below.

PSEUDOCODE FOR LOGISTIC REGRESSION CLASSIFIER

STEP 1 : Start the process.

STEP 2 : Import necessary libraries.

STEP 3 : Create a function LogisticRegression, give necessary parameters.

STEP 4 : Fit those function in trained dataset.

STEP 5 : Create another function y_lrforecast to store the test dataset predicted accuracy.

STEP 6 : Print classification report to display precision, recall, f1-score and support.

STEP 7 : Display the train and test accuracy using accuracy_score.

STEP 8 : Display precision, recall and f1-score using evaluation metrics such as micro, macro and weighted average.

STEP 9 : The output is evaluated.

STEP 10 : Stop the process.

4.4.4 KNN CLASSIFIER

- This algorithm is used to handle the classification model difficulties.
- The K-nearest neighbour or K-NN approach generates an arbitrary boundary to classify data.
- As a result, a larger k value means finer separation curves and simpler models.
- Consequently, smaller k values tend to over-fit the data, culminating in complex systems.

The algorithms evaluate the distances between a given data point in the set and any other K sets of data points in the dataset that are close to the initial point, then vote for the most frequently occurring category. Typically, measurement can be done using Euclidean distance. As a result, the final model is nothing more than labelled data in space. This approach has a long history of use in genetics, meteorology, and other domains. When there are more features, the algorithm performs well, and SVM dominates in this circumstance.

KNN does, in fact, reduce over-fitting. The best value for K, on the other hand, must be chosen. The square root of the number of samples in the dataset is usually K. A system performed must be chosen because a lesser number may lead to over-fitting and a higher value may require a high computational complication in distance. As a result, using an error plot may be advantageous. Another approach is to use the elbow technique. Then maybe can either take root or use the elbow strategy.

In this K nearest neighbour Classifier the required libraries are imported and the KNeighborsClassifier function is created and finally the following metrics of the model are calculated:

- ✓ Train and Test Accuracy
- ✓ Classification metrics
- ✓ Accuracy
- ✓ Precision
- ✓ Recall
- ✓ F1 Score

As, the train and test split is 80:20 ratio, the testing accuracy is taken as the final accuracy level for K Nearest Neighbors model. This is a multi-class classification so that, for

Precision, Recall and F1 Score we calculated for their averages micro, macro and weighted for their corresponding metrics.

The model is built followed by the steps in the pseudocode to be discussed below.

PSEUDOCODE FOR KNN CLASSIFIER

STEP 1	**:**	Start the process.
STEP 2	**:**	Import necessary libraries.
STEP 3	**:**	Create a function KNearestNeighbors, give necessary parameters.
STEP 4	**:**	Fit those function in trained dataset.
STEP 5	**:**	Create another function y_knnforecast to store the test dataset predicted accuracy.
STEP 6	**:**	Print classification report to display precision, recall, f1-score and support.
STEP 7	**:**	Display the train and test accuracy using accuracy_score.
STEP 8	**:**	Display precision, recall and f1-score using evaluation metrics such as micro, macro and weighted average.
STEP 9	**:**	The output is evaluated.
STEP 10	**:**	Stop the process.

4.5 PHASE 5 - COMPARITIVE ANALYSIS OF ALGORITHMS

A Comparitive Analysis of four algorithms namely Support Vector Machine, Naïve Bayes Classifier, Logistic Regression Classifier and K-Nearest Neighbour Classifier gives the best accuracy levels for this Firewall dataset. But with few count different between the accuracy levels of the different algorithms used.

The four different Algorithms are compared based on their accuracy level, and different classification metrics such as precision score, recall score, f1 score and also since this is multi class dataset, Evaluation metrics such as 'micro, macro and weighted' based on Classification metrics these evaluation metrics are also calculated for both train and test datasets. And finally based on the higher level of test accuracy the model is chosen as the best model to fit. One model is chosen to be the best one. Based on their performance the accuracy is improved.

The 5 major features considered are: Action (Allow, Deny, Drop, Reset -Both) Source Port, Destination Port, NAT Source Port, NAT Destination Port, Bytes. The Naive Bayes method performed better than the other classification methods included in the model. With 99.26% accuracy, the Naive Bayes classifier was found to have the highest Accuracy value.

MODEL	ACCURACY
SVM CLASSIFIER	99.03%
NAÏVE BAYES	**99.26%**
LOGISTIC REGRESSION	98.70%
KNN CLASSIFIER	99.10%

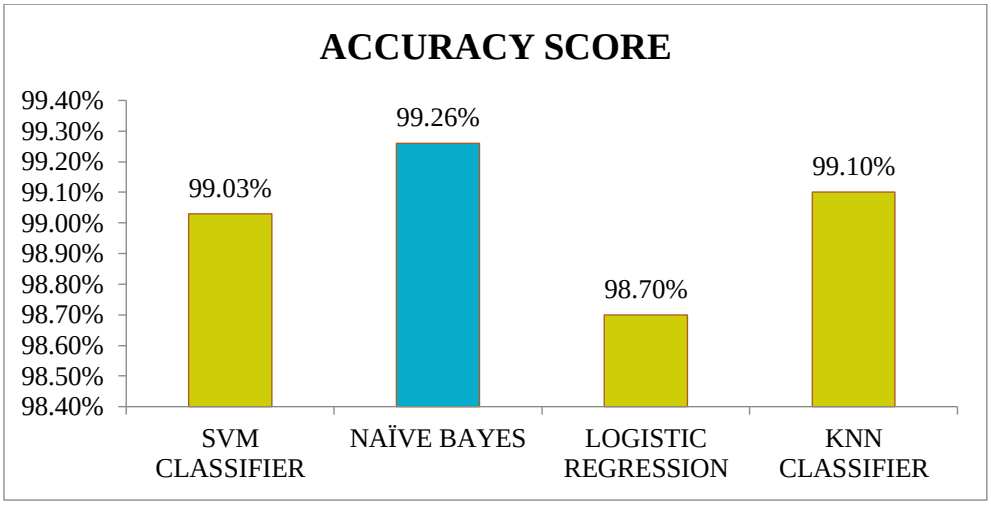


FIG. 4.5 COMPARITIVE ANALYSIS OF ALGORITHMS

4.5.1 PERFORMANCE EVALUATION

TRAIN ACCURACY COMPARISON

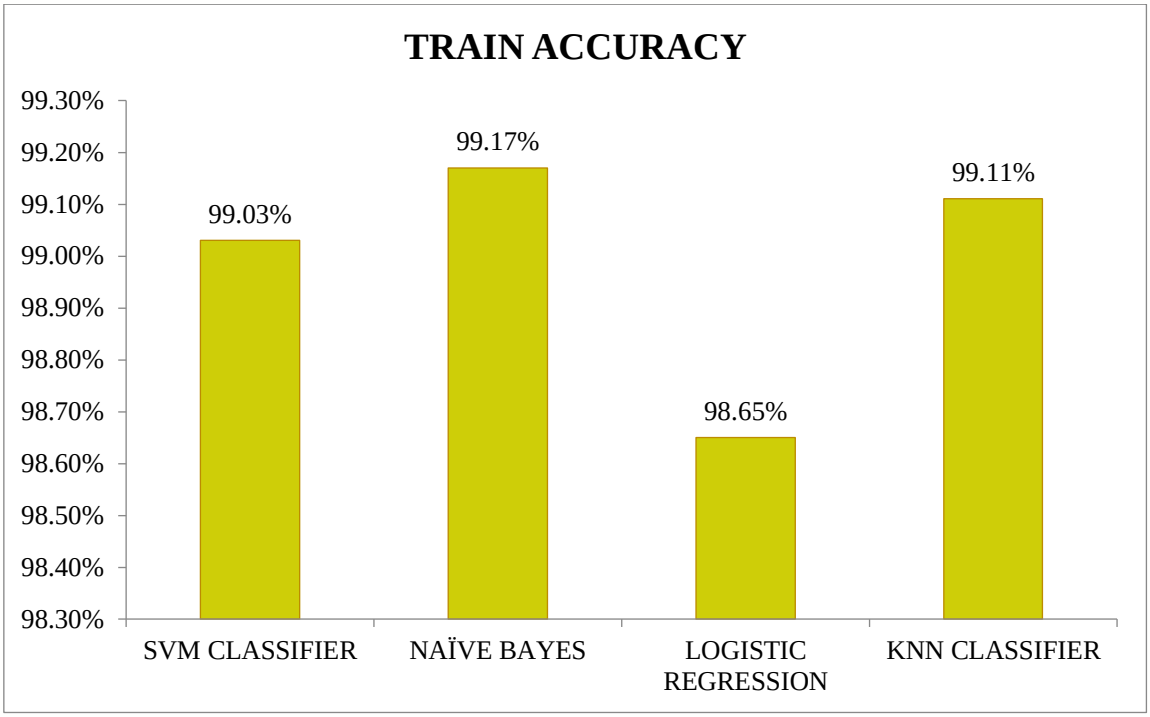


FIG.4.5.1 TRAIN ACCURACY COMPARISON

PRECISION ACCURACY MICRO – COMPARISON

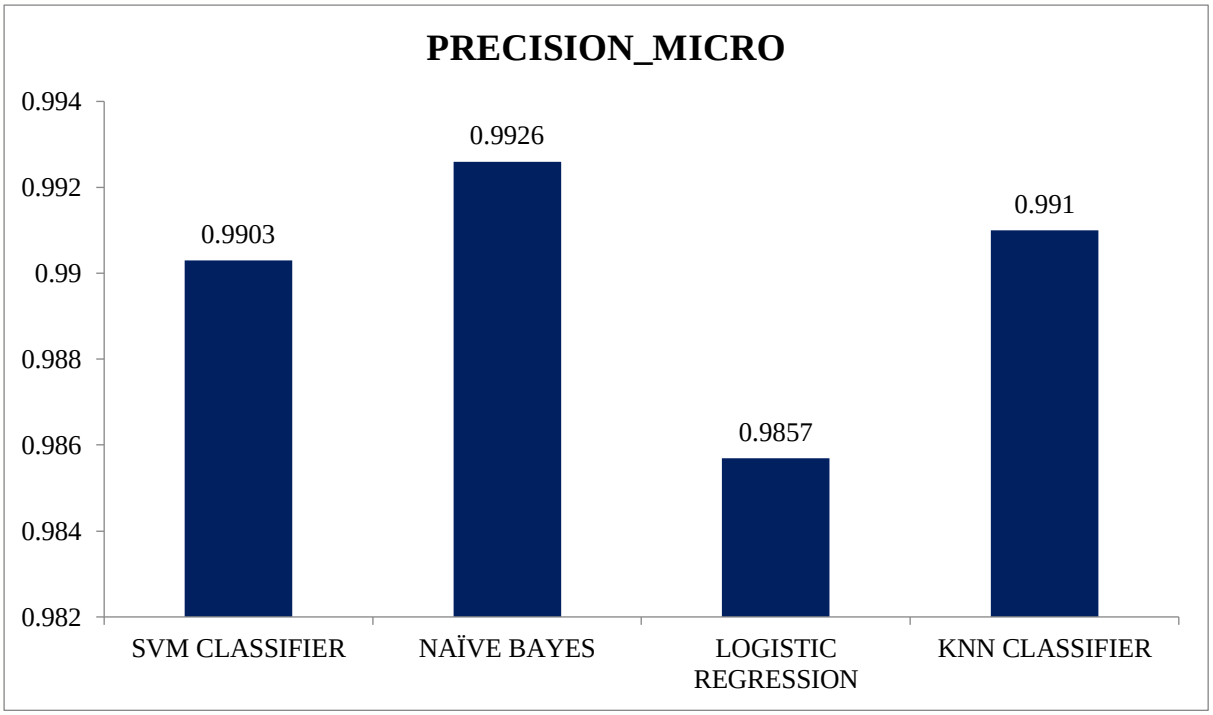


FIG.4.5.2 PRECISION ACCURACY MICRO - COMPARISON

PRECISION ACCURACY MACRO – COMPARISON

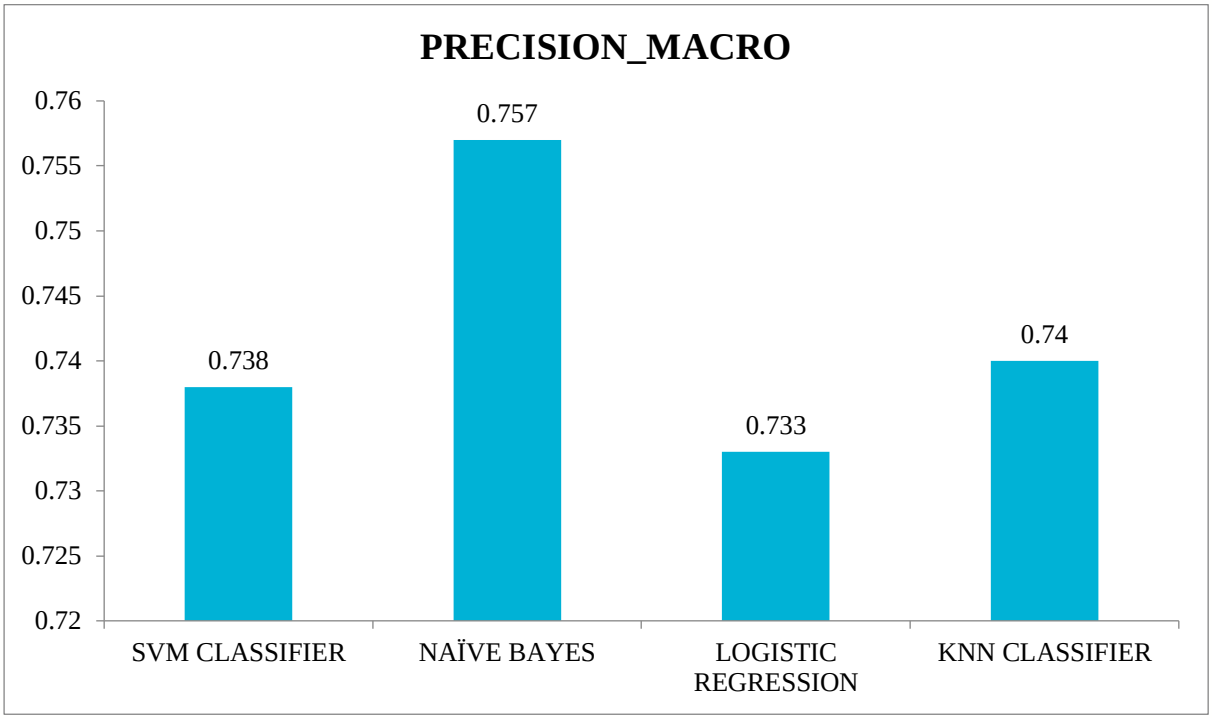


FIG.4.5.3 PRECISION ACCURACY MACRO - COMPARISON

PRECISION ACCURACY WEIGHTED – COMPARISON

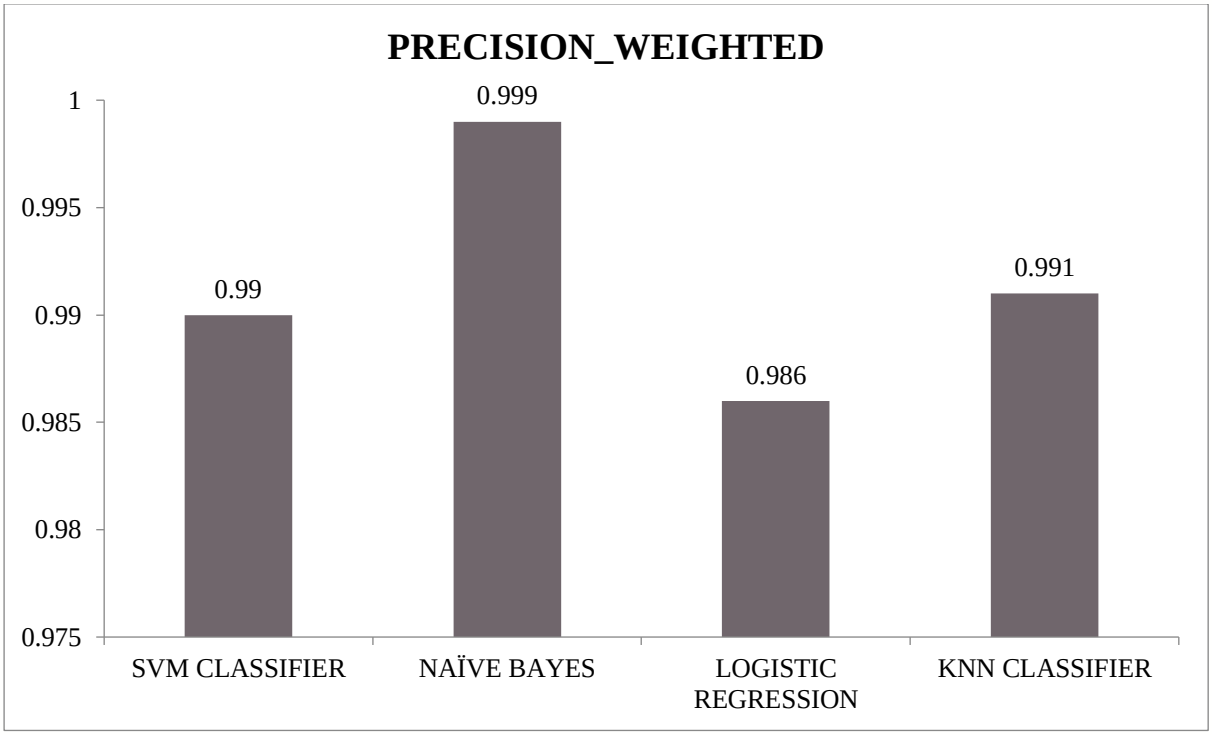


FIG.4.5.4 PRECISION ACCURACY WEIGHTED - COMPARISON

F1 SCORE ACCURACY MICRO – COMPARISON

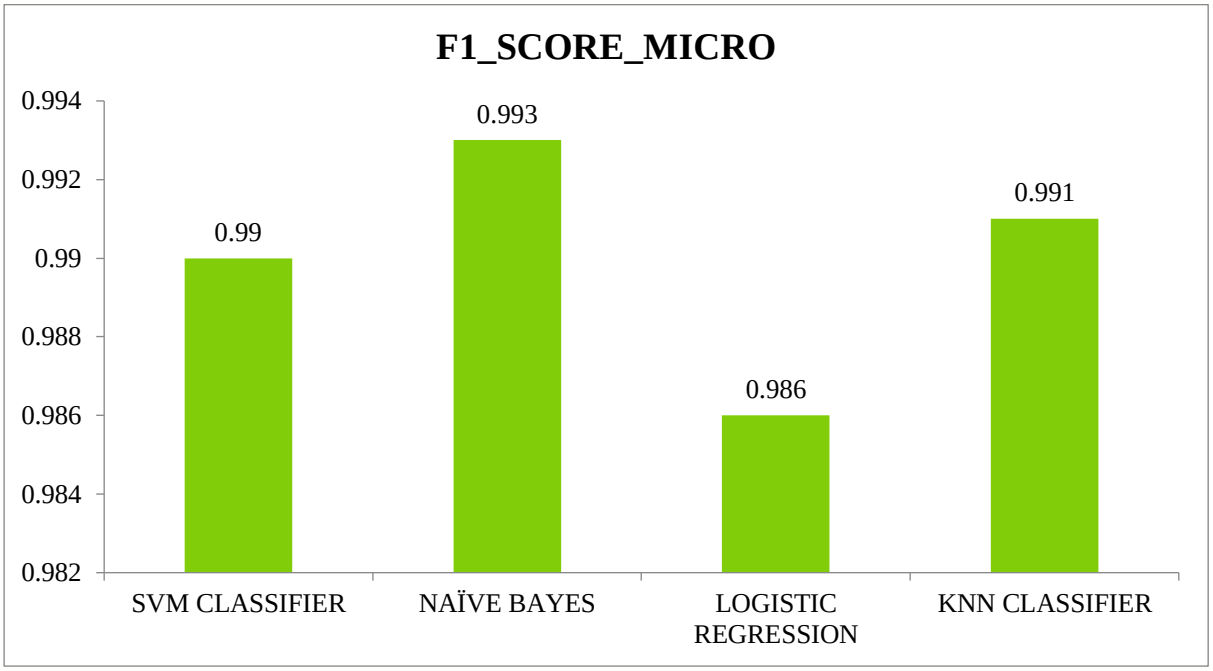


FIG.4.5.5 F1 SCORE ACCURACY MICRO - COMPARISON

F1 SCORE ACCURACY MACRO – COMPARISON

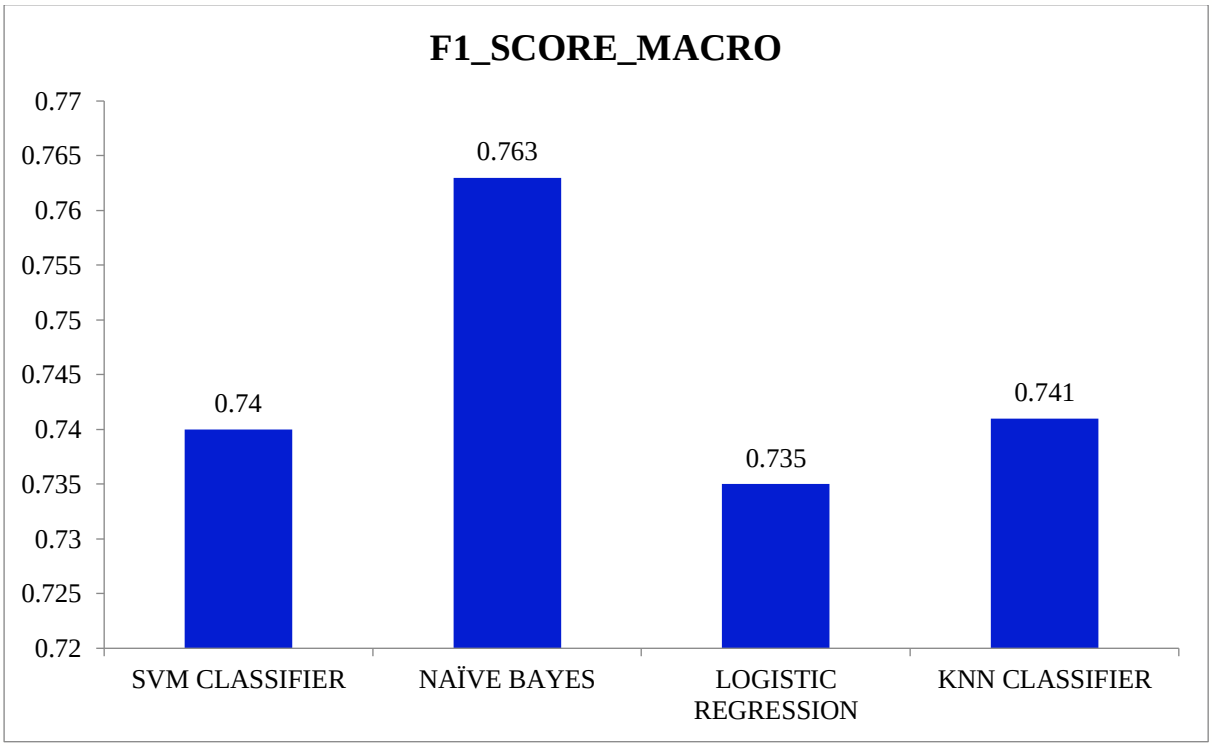


FIG.4.5.6 F1 SCORE ACCURACY MACRO - COMPARISON

F1 SCORE ACCURACY WEIGHTED – COMPARISON

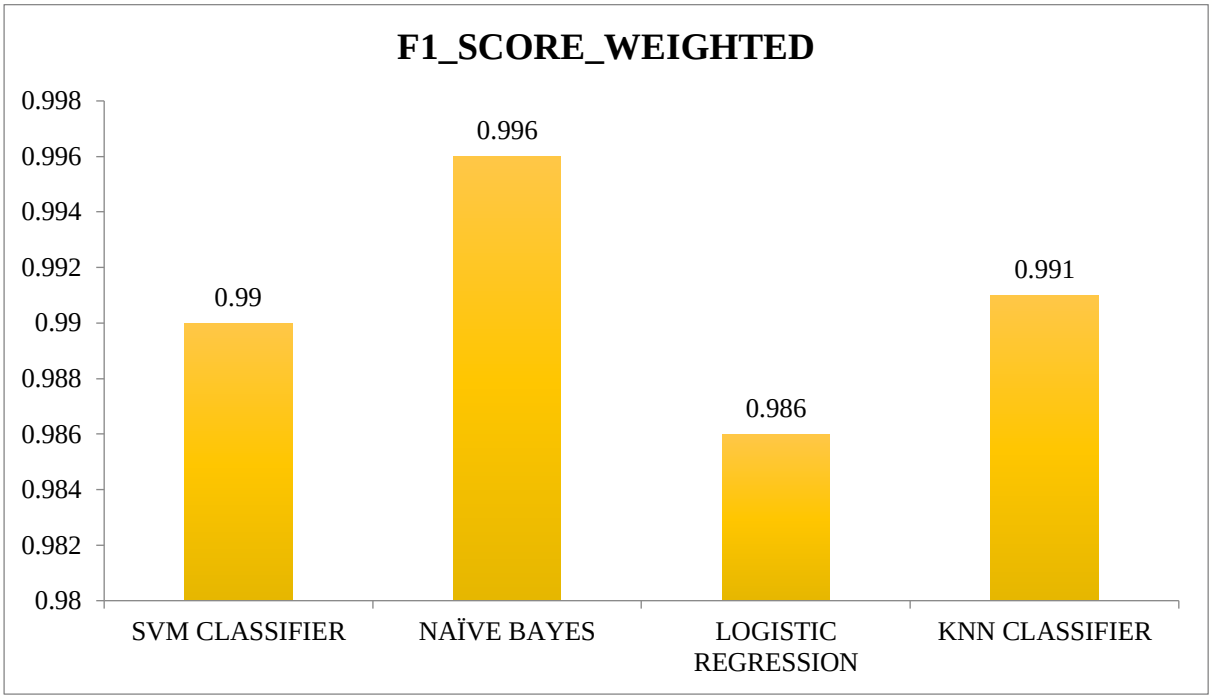


FIG.4.5.7 F1 SCORE ACCURACY WEIGHTED - COMPARISON

RECALL SCORE ACCURACY MICRO – COMPARISON

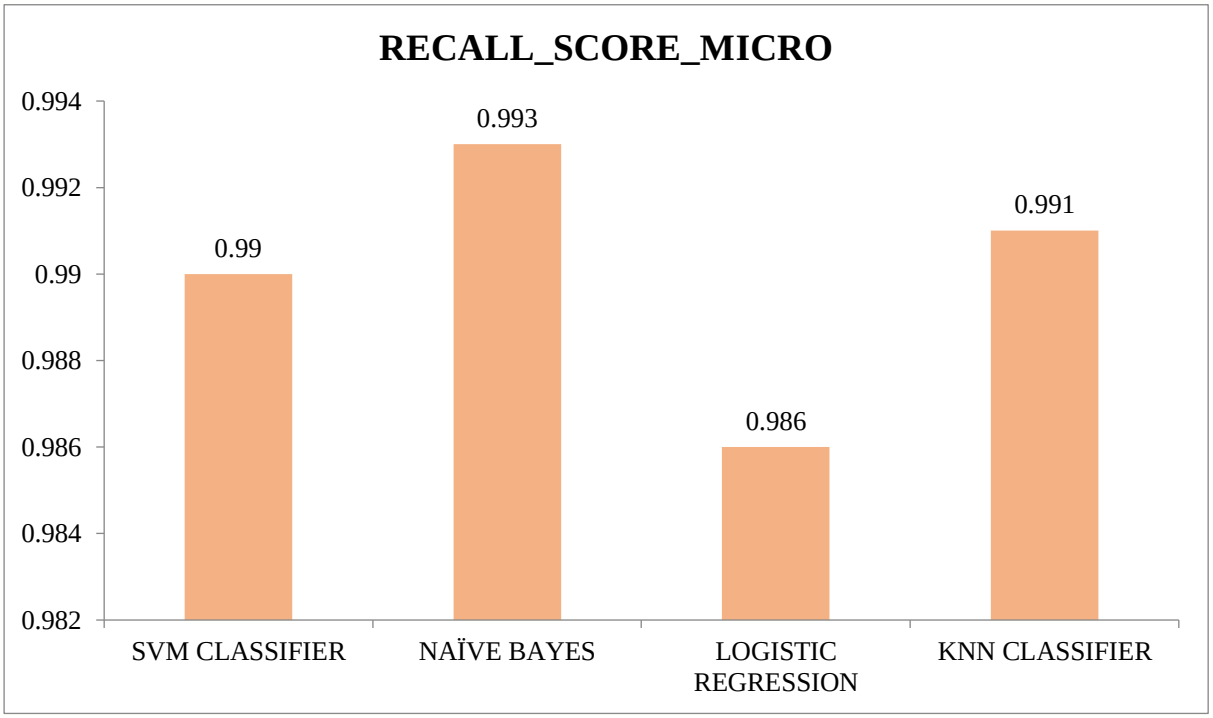


FIG.4.5.8 RECALL SCORE ACCURACY MICRO - COMPARISON

RECALL SCORE ACCURACY MACRO – COMPARISON

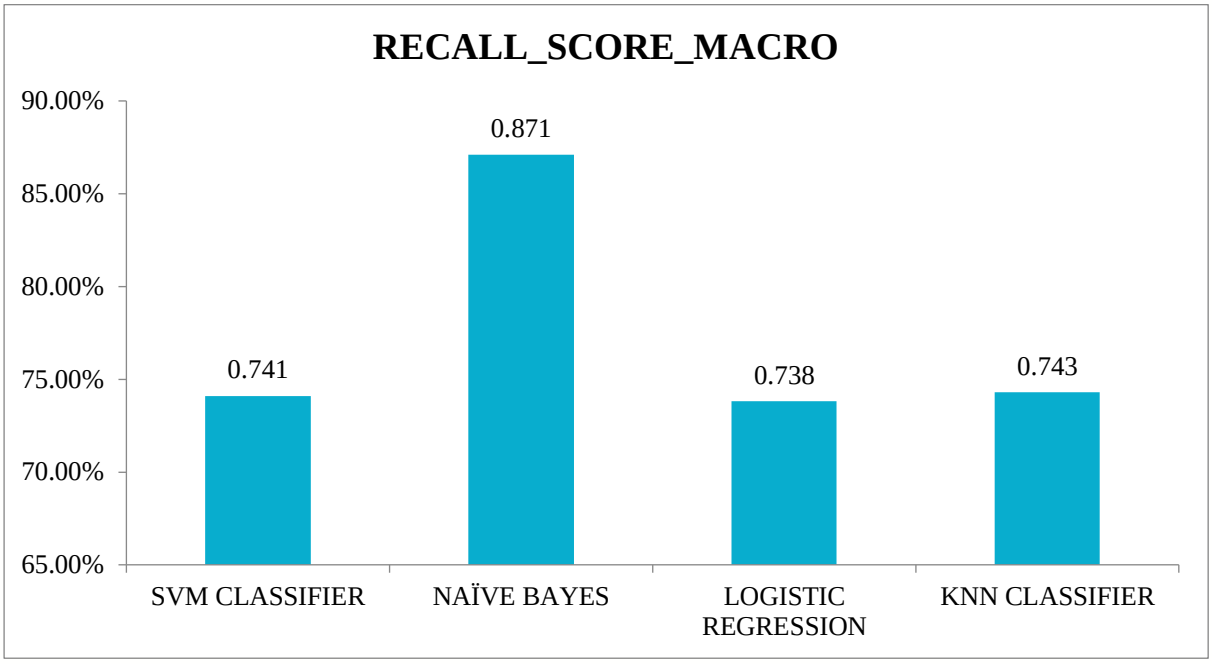


FIG.4.5.9 RECALL SCORE ACCURACY MACRO - COMPARISON

RECALL SCORE ACCURACY WEIGHTED – COMPARISON

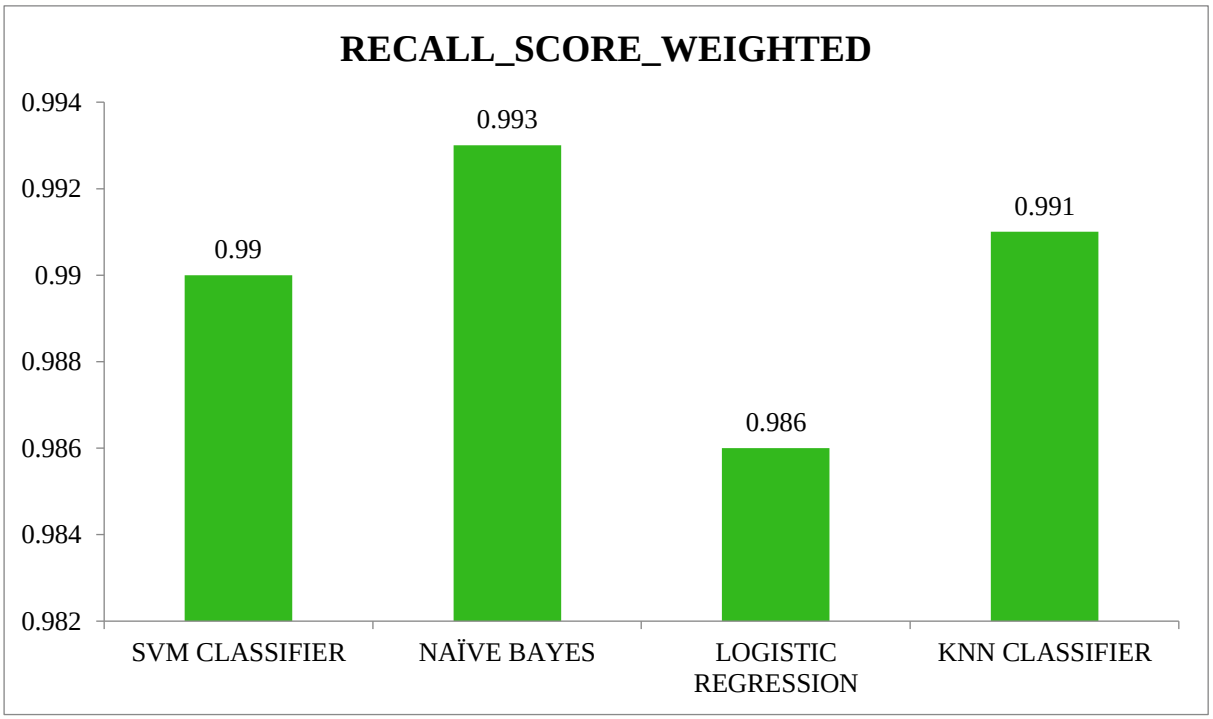


FIG.4.5.10 RECALL SCORE ACCURACY WEIGHTED - COMPARISON

4.6 PHASE 6 - RESULTS AND OUTCOME

PHASE 2 - DATA PRE-PROCESSING (AFTER LABEL ENCODING)

FIG. 4.6.1 LABEL ENCODED FOR THE CLASS - ACTION

The above figure represents that the label has been encoded, for the Action – Allow, Deny, Drop and Reset, based on the firewall rules the packets decides to pass it or not.

STANDARDIZATION

```
[[ 0.00410758 -0.54874891 -0.65779368 ...  0.03860065 -0.008746
  -0.02077207]
 [ 0.75810782 -0.54864056 -0.87840622 ... -0.2140437  -0.01182699
  -0.02577167]
 [ 0.00410758 -0.56987603  1.35874068 ... -0.11687279 -0.01126681
  -0.02452177]
 ...
 [-0.88185253 -0.54874891 -0.5945663  ... -0.15898018 -0.00790573
  -0.02077207]
 [ 0.48389612 -0.54874891  0.21012511 ... -0.16545824 -0.00902609
  -0.02202197]
 [ 0.46672101 -0.56987603  2.08230243 ... -0.11363376 -0.01182699
  -0.02535504]]
[[ 0.12708658  2.90223129 -0.87840622 ... -0.2140437  -0.01182699
  -0.02577167]
 [ 0.03767115 -0.54874891 -0.55015226 ...  0.07422998 -0.00958627
  -0.02202197]
 [ 0.96335688 -0.56987603  0.35808083 ... -0.11687279 -0.01182699
  -0.02535504]
```

FIG. 4.6.2 FEATURE SCALING USING STANDARDIZATION

The above figure represents, Feature scaling using the method - Standardization.

HEAT MAP

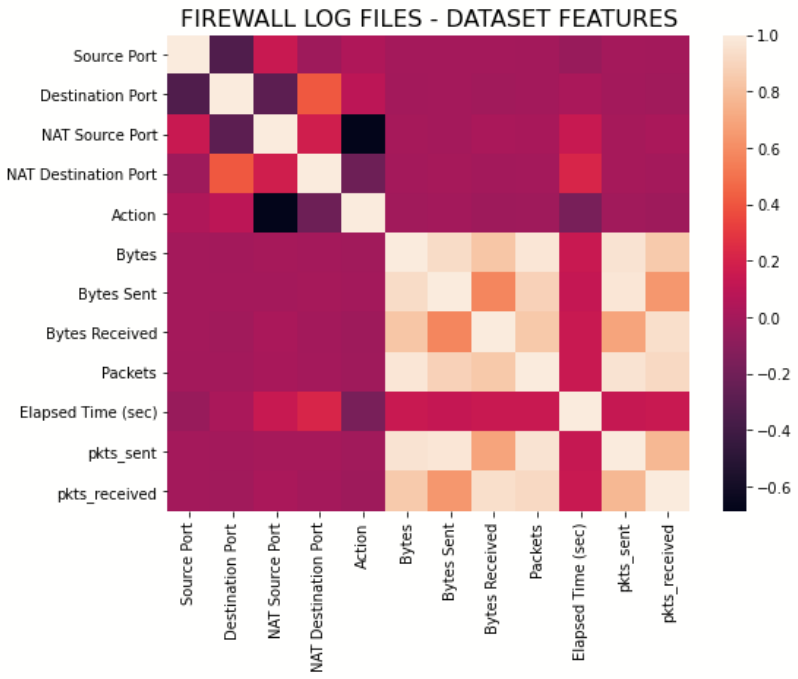


FIG.4.6.3 HEAT MAP OF ALL FEATURES

The above figure represents the heat map of all the features from the dataset.

PHASE 3 – FEATURE SELECTION

Ranker + InfoGainAttributeEval

Attribute Evaluator: Information Gain Attribute Evaluator&Search method: Ranker

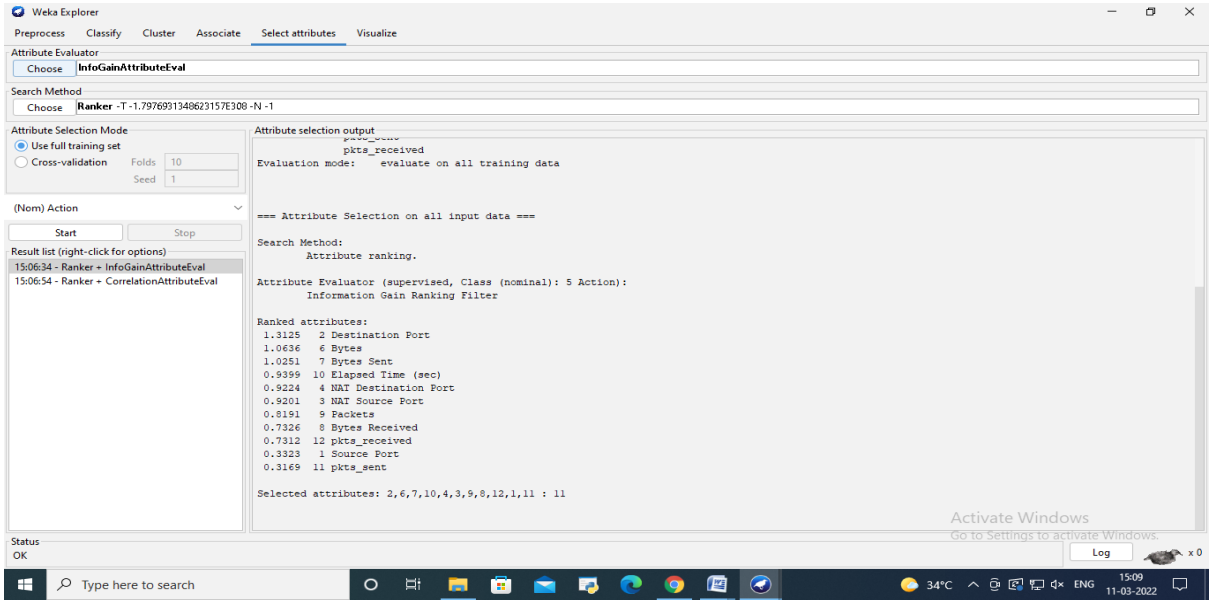


FIG. 4.6.4 FEATURE SELECTION - InfoGainAttributeEval

Ranker + CorrelationAttributeEval

Attribute Evaluator: Correlation Attribute Evaluator&Search method: Ranker

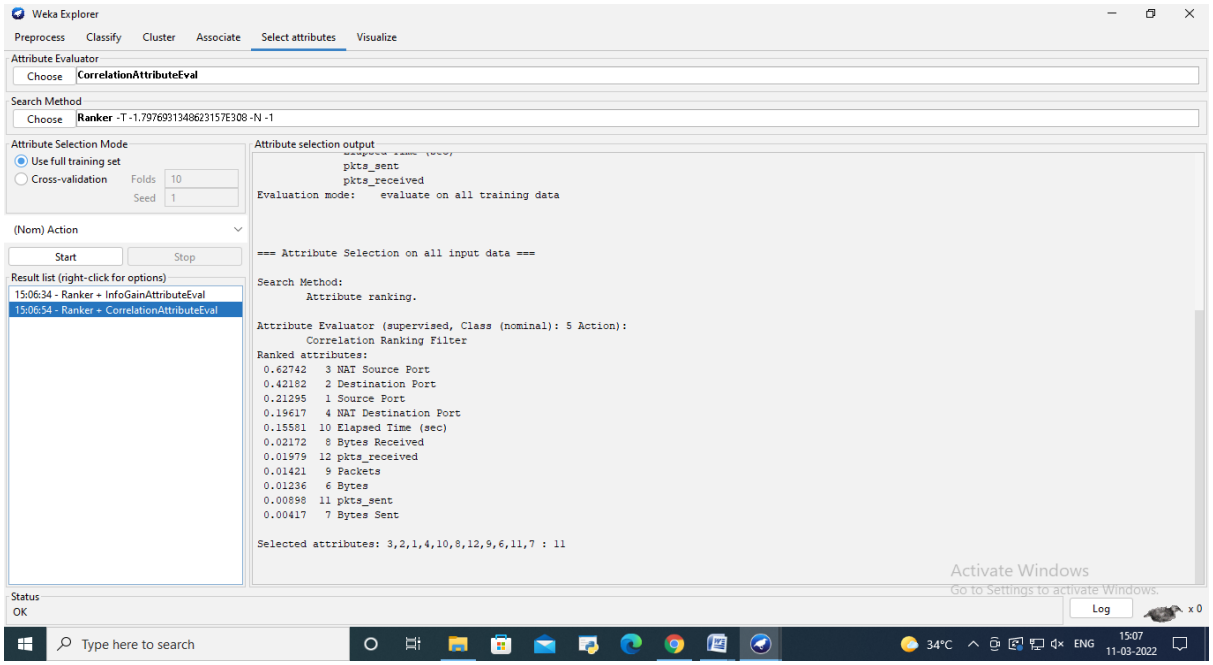


FIG. 4.6.5 FEATURE SELECTION – CorrelationAttributeEval

PHASE 4 – MODEL BUILDING

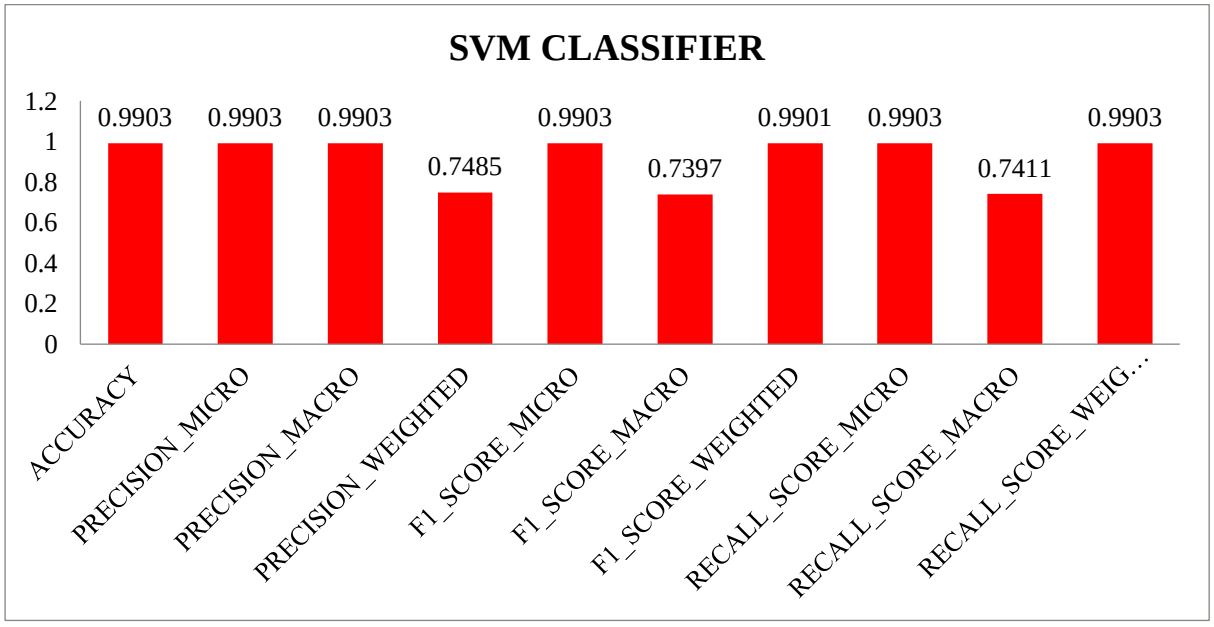


FIG. 4.6.6 SVM CLASSIFIER COMPARISON

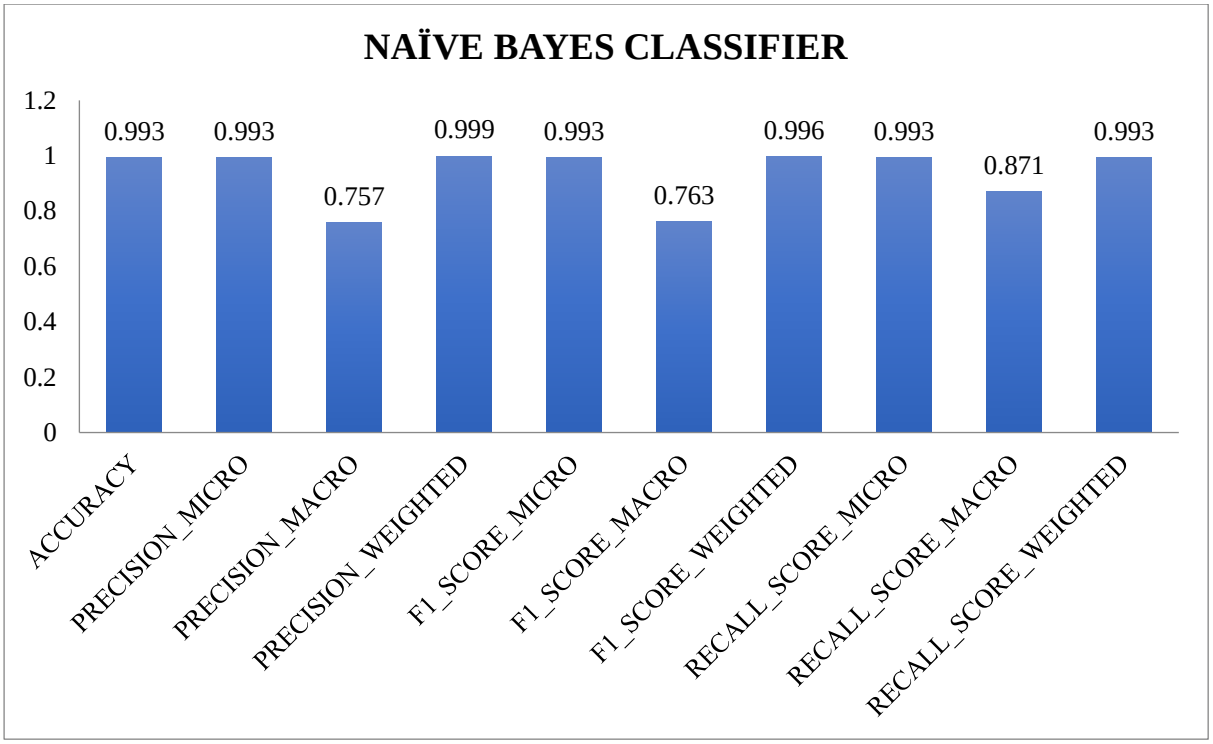


FIG. 4.6.7 NAÏVE BAYES CLASSIFIER COMPARISON

FIG. 4.6.8 LOGISTIC REGRESSION CLASSIFIER COMPARISON

FIG. 4.6.9 KNN CLASSIFIER COMPARISON

The above four figures gives the final results of comparison of different algorithms, based on their accuracy, precision micro, macro and weighted, f1-score micro, macro and weighted, recall score micro, macro and weighted.

CONCLUSION

CONCLUSION

The firewall is the most crucial elements of a network, and there should be no contradiction in the security policies employed, because to do so would expose the network to security risks. So, all the people should be aware of the risks employed in all the components. Here, considered only the 5 major features: Action (Allow, Deny, Drop, Reset-Both) Source Port, Destination Port, NAT Source Port, NAT Destination Port, Bytes. The Naive Bayes method performed better than the other classification methods included in the model. With 99.26% accuracy, the Naive Bayes classifier was found to have the highest Accuracy value.

SCOPE FOR THE FUTURE ENHANCEMENT

SCOPE FOR THE FUTURE ENHANCEMENT

Network technology and applications are continually evolving, yet network security is lagging behind. Many computer security threats originate from networking, while others are amplified by it. The secure network is dependent on secure computing, and vice versa. It's no surprise that as networking technology becomes more vulnerable, individuals are beginning to take network security more seriously. The usage of these methods, in a real network with many scenarios in the future to analyse the depth and classify the threats based on their level of vulnerability. Further the model can be developed using other different algorithms which can give more accuracy in terms of selected features.

REFERENCES

REFERENCES

[1] AL-Behadili, H. (2021). Decision Tree for Multiclass Classification of Firewall Access. *International Journal of Intelligent Engineering and Systems*, *14*(3), 294–302. https://doi.org/10.22266/ijies2021.0630.25

[2] Allagi, S., & Rachh, R. (2019). Analysis of Network log data using Machine Learning. *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. https://doi.org/10.1109/i2ct45611.2019.9033737

[3] As-Suhbani, H. E., & Khamitkar, S. (2019). Classification of Firewall Logs Using Supervised Machine Learning Algorithms. *International Journal of Computer Sciences and Engineering*, *7*(8), 301–304. https://doi.org/10.26438/ijcse/v7i8.301304

[4] Ertam, F., & Kaya, M. (2018). Classification of firewall log files with multiclass support vector machine. *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. https://doi.org/10.1109/isdfs.2018.8355382

[5] Hommes, S., State, R., & Engel, T. (2012). A distance-based method to detect anomalous attributes in log files. *2012 IEEE Network Operations and Management Symposium*. https://doi.org/10.1109/noms.2012.6211940

[6] Kamiya, K., Aoki, K., Nakata, K., Sato, T., Kurakami, H., &Tanikawa, M. (2015). The method of detecting malware-infected hosts analyzing firewall and proxy logs. *2015 10th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT)*. https://doi.org/10.1109/apsitt.2015.7217113

[7] Kowalski, K., & Beheshti, M. (2006). Analysis of Log Files Intersections for Security Enhancement. *Third International Conference on Information Technology: New Generations (ITNG'06)*.https://doi.org/10.1109/itng.2006.32

[8] Nimbalkar, P., Mulwad, V., Puranik, N., Joshi, A., &Finin, T. (2016). Semantic Interpretation of Structured Log Files. *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*. https://doi.org/10.1109/iri.2016.81

[9] Sharma, D., Wason, V., &Johri, P. (2021). Optimized Classification of Firewall Log Data using Heterogeneous Ensemble Techniques. *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. https://doi.org/10.1109/icacite51222.2021.9404732

[10] Winding, R., Wright, T., &Chapple, M. (2006). System Anomaly Detection: Mining Firewall Logs. *2006 Securecomm and Workshops*. https://doi.org/10.1109/seccomw.2006.359572

WEBSITE REFERENCES

- https://www.researchgate.net/publication/335826729_Classification_of_Firewall_Logs_Using_Supervised_Machine_Learning_Algorithms
- https://www.exabeam.com/siem-guide/siem-concepts/firewall-logs
- https://medium.com/mlearning-ai/routing-network-traffic-based-on-firewall-logs-using-machine-learning-dc5e5c8c6bb3
- http://www.ijstr.org/final-print/feb2020/Discovering-Anomalous-Rules-In-Firewall-Logs-Using-Data-Mining-And-Machine-Learning-Classifiers.pdf
- https://www.kdnuggets.com/2017/02/machine-learning-driven-firewall.html
- https://towardsdatascience.com/how-data-science-could-make-cybersecurity-troubleshooting-easier-firewall-logs-analysis-591e4832f7e6
- https://www.loganalysis.org/firewall-logging/
- https://repositorio-aberto.up.pt/bitstream/10216/128588/2/412447.pdf

APPENDIX

APPENDIX

CODING

IMPORT LIBRARIES

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler # used for feature scaling
from sklearn.model_selection import train_test_split
%matplotlib inline
from sklearn.metrics import accuracy_score
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')


# Reading CSV file
df = pd.read_csv('log2.csv')
df
```

DATA PRE-PROCESSING - LABEL ENCODING

```
data = df
label_encode = pd.DataFrame(data)
from sklearn import preprocessing
```

```
label = preprocessing.LabelEncoder()
print ("LABEL ENCODING - allow - 0, drop - 1, deny - 2 reset-both - 3")
label_encode['Action']= label.fit_transform(label_encode['Action'])
print(label_encode['Action'].unique())
print(df)

# Frequencies based on Action(allow, deny, drop and reset both)
y = df['Action'].values
y = y.reshape(-1,1)
x_data = df.drop(['Action'],axis = 1)

plt.title('Class - Allow, Deny, Drop, Reset-both', size = 16)
plt.ylim([0, 500])

sns.countplot(x='Action',data=df,palette='hls')
plt.show()
```

DIVIDE THAT DATA INTO TRAIN AND TEST SPLIT

```
from sklearn.model_selection import train_test_split

# Splitting the dataset into dependant and independant feature

X = df.drop(["Action"],axis =1)
y = df["Action"]
```

```
# Splitting the dataset into train and test sets: 80-20 split
fromsklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

FEATURE SCALING - STANDARD SCALER

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
print(X_train)
print(X_test)
```

FEATURE SELECTION

Selected Features from Weka tool

```
feature_df =  df[[ 'Source Port','Destination Port', 'NAT Source Port', 'NAT Destination Port','Bytes']]

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import precision_recall_fscore_support
from sklearn import svm, datasets
import warnings
```

```
warnings.filterwarnings('ignore')

# independent variable

X=feature_df

# dependent variable

y=df['Action']
```

MODEL BUILDING

SUPPORT VECTOR MACHINE

```
from sklearn import svm

classifier = svm.SVC(kernel='linear', gamma='auto', C= 2, probability = True)

classifier.fit(X_train, y_train)


y_predict = classifier.predict(X_test)

print(y_predict)


from sklearn.metrics import classification_report

print(classification_report(y_test, y_predict))

print('Support Vector Machine Accuracy: {:.02%}'.format(classifier.score(X_test, y_test)))
```

NAIVE BAYES CLASSIFIER

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb = nb.fit(X_train, y_train.ravel())

y_nbforecast=nb.predict(X_test)
```

```
from sklearn.metrics import classification_report

print(classification_report(y_test, nb.predict(X_test)))

print('Naive Bayes Classifier Accuracy: {:.02%}'.format(nb.score(X_test, y_test)))
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression


lr = LogisticRegression(solver='lbfgs')

lr = lr.fit(X_train, y_train.ravel())

y_lrforecast=lr.predict(X_test)


from sklearn.metrics import classification_report

print(classification_report(y_test, lr.predict(X_test)))

print('Logistic Regression Accuracy: {:.02%}'.format(lr.score(X_test, y_test)))
```

KNN CLASSIFIER

```
from sklearn.neighbors import KNeighborsClassifier

K = 1

knn = KNeighborsClassifier(n_neighbors=K)

knn = knn.fit(X_train,y_train.ravel())


print("k = {}neighbors , knn test:{}".format(K, knn.score(X_test, y_test)))

print("knn Classifier: {}".format(K, knn.score(X_train, y_train)))
```

```
ran = np.arange(1,40)

train_list = []

test_list = []

fori,each in enumerate(ran):

knn = KNeighborsClassifier(n_neighbors=each)

knn = knn.fit(X_train, y_train.ravel())

test_list.append(knn.score(X_test, y_test))

train_list.append(knn.score(X_train, y_train))


print("Best train: {} , k={}".format(np.max(train_list),train_list.index(np.max(train_list))+1))

print("Best test accuracy: {} ,
k={}".format(np.max(test_list),test_list.index(np.max(test_list))+1))

from sklearn.metrics import classification_report

print(classification_report(y_test, lr.predict(X_test)))

print('KNN Accuracy: {:.02%}'.format(knn.score(X_test, y_test)))
```

COMPARITIVE ANALYSIS OF ALGORITHMS

```
Model = []

Model_Test_Accuracy = []

defstoreResults(model, a2):

Model.append(model)

Model_Test_Accuracy.append(a2)
```

SCREENSHOTS

CLASSIFICATION OF FIREWALL LOG FILES USING SUPERVISED MACHINE LEARNING METHODS

```
Methodology followed as:

Phase 1 - Data Collection

Phase 2 - Data Pre-processing
    Label Encoding
    Train and Test Split
    Feature Scaling - Standardization

Phase 3 - Feature Selection using Weka Tool

Phase 4 - Model Building
    Support Vector Machine
    Naive Bayes Classifier
    Logistic Regression Classifier
    Knn Classifier

Phase 5 - Comparitive Analysis of Algorithms

Phase 6 - Results and Outcome
```

In [1]:
```
# Import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

In [1]:
```
# Import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler # used for feature scaling
from sklearn.model_selection import train_test_split
%matplotlib inline
from sklearn.metrics import accuracy_score
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

PHASE 1 - DATA ACQUISITION

In [2]:
```
# Reading CSV file
df = pd.read_csv('log2.csv')
df
```

Out[2]:

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received
0	57222	53	54587	53	allow	177	94	83	2	30	1	1
1	56258	3389	56258	3389	allow	4768	1600	3168	19	17	10	9
2	6881	50321	43265	50321	allow	238	118	120	2	1199	1	1
3	50553	3389	50553	3389	allow	3327	1438	1889	15	17	8	7
4	50002	443	45848	443	allow	25358	6778	18580	31	16	13	18
...	...	...	...	...	...	...	...	...	...	...	...	...
65527	63691	80	13237	80	allow	314	192	122	6	15	4	2
65528	50964	80	13485	80	allow	4680740	67312	4613428	4675	77	985	3690
65529	54871	445	0	0	drop	70	70	0	1	0	1	0
65530	54870	445	0	0	drop	70	70	0	1	0	1	0
65531	54867	445	0	0	drop	70	70	0	1	0	1	0

65532 rows × 12 columns

In [3]:
```
# summarize the shape of the dataset
print(df.shape)
print(df.size)
print(df.count())
df.dtypes
df.describe()
df.info()
```

```
(65532, 12)
786384
Source Port             65532
Destination Port        65532
NAT Source Port         65532
NAT Destination Port    65532
Action                  65532
Bytes                   65532
Bytes Sent              65532
Bytes Received          65532
Packets                 65532
Elapsed Time (sec)      65532
pkts_sent               65532
pkts_received           65532
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65532 entries, 0 to 65531
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Source Port           65532 non-null  int64
 1   Destination Port      65532 non-null  int64
 2   NAT Source Port       65532 non-null  int64
```

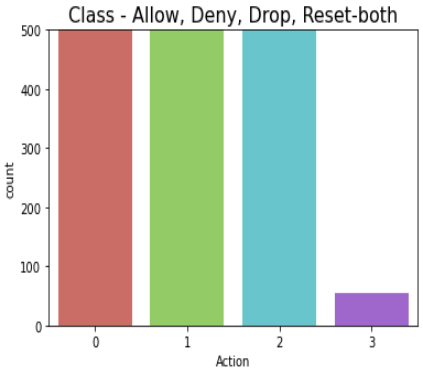
PHASE 2 - DATA PRE-PROCESSING

1) Label Encoding

In [7]:
```
# LABEL ENCODING

data = df
label_encode = pd.DataFrame(data)
from sklearn import preprocessing
label = preprocessing.LabelEncoder()
print ("LABEL ENCODING - allow - 0, drop - 1, deny - 2 reset-both - 3")
label_encode['Action']= label.fit_transform(label_encode['Action'])

print(label_encode['Action'].unique())
print(df)
```

```
LABEL ENCODING - allow - 0, drop - 1, deny - 2 reset-both - 3
[0 2 1 3]
       Source Port  Destination Port  NAT Source Port  NAT Destination Port  \
0            57222                53            54587                    53
1            56258              3389            56258                  3389
2             6881             50321            43265                 50321
3            50553              3389            50553                  3389
4            50002               443            45848                   443
...            ...               ...              ...                   ...
65527        63691                80            13237                    80
65528        50964                80            13485                    80
65529        54871               445                0                     0
65530        54870               445                0                     0
```

In [8]:
```
# Frequencies based on Action(allow, deny, drop and reset both)

y = df['Action'].values
y = y.reshape(-1,1)
x_data = df.drop(['Action'],axis = 1)

plt.title('Class - Allow, Deny, Drop, Reset-both', size = 16)
plt.ylim([0, 500])

sns.countplot(x='Action',data=df,palette='hls')
plt.show()
```



2) Train and Test Split

In [9]:
```
# Divide that data into train and test split
from sklearn.model_selection import train_test_split

# Splitting the dataset into dependant and independant feature

X = df.drop(["Action"],axis =1)
y = df["Action"]

# Splitting the dataset into train and test sets: 80-20 split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

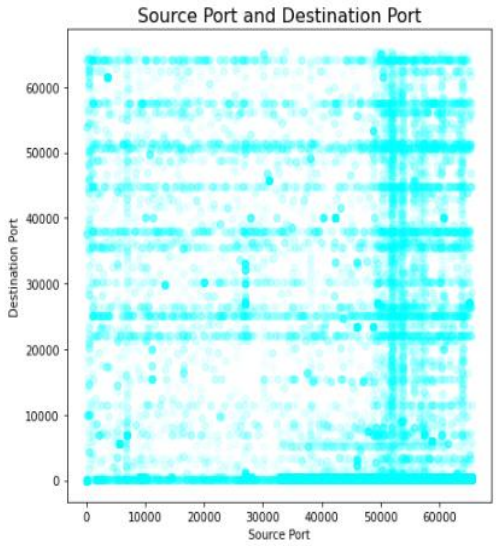
Out[9]: ((52425, 11), (52425,), (13107, 11), (13107,))

3) Feature Scaling - Standard Scaler

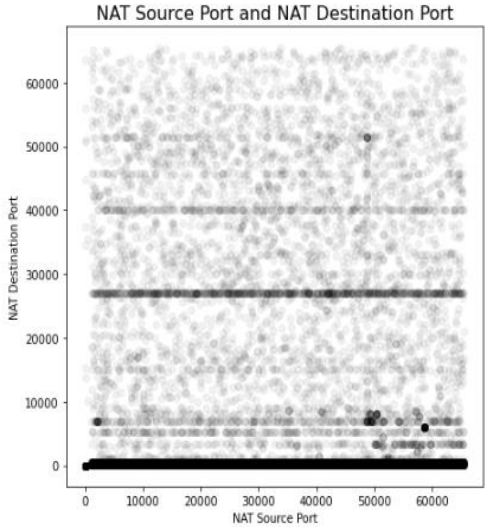
In [10]:
```
# FEATURE SCALING - STANDARD SCALER
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
print(X_train)
print(X_test)
```

```
[[ 0.00410758 -0.54874891 -0.65779368 ...  0.03860065 -0.008746
  -0.02077207]
 [ 0.75810782 -0.54864056 -0.87840622 ... -0.2140437  -0.01182699
  -0.02577167]
 [ 0.00410758 -0.56987603  1.35874068 ... -0.11687279 -0.01126681
  -0.02452177]
 ...
 [-0.88185253 -0.54874891 -0.5945663  ... -0.15898018 -0.00790573
  -0.02077207]
 [ 0.48389612 -0.54874891  0.21012511 ... -0.16545824 -0.00902609
  -0.02202197]
 [ 0.46672101 -0.56987603  2.08230243 ... -0.11363376 -0.01182699
  -0.02535504]]
[[ 0.12708658  2.90223129 -0.87840622 ... -0.2140437  -0.01182699
  -0.02577167]
 [ 0.03767115 -0.54874891 -0.55015226 ...  0.07422998 -0.00958627
  -0.02202197]
 [ 0.96335688 -0.56987603  0.35808083 ... -0.11687279 -0.01182699
  -0.02535504]
```

In [13]:
```
# Source Port and Destination port
plt.figure(figsize=(7,7))
plt.scatter(df['Source Port'], df['Destination Port'], alpha=0.05, color='cyan')
plt.xlabel('Source Port')
plt.ylabel('Destination Port')
plt.title('Source Port and Destination Port', size = 16)
plt.show()
```

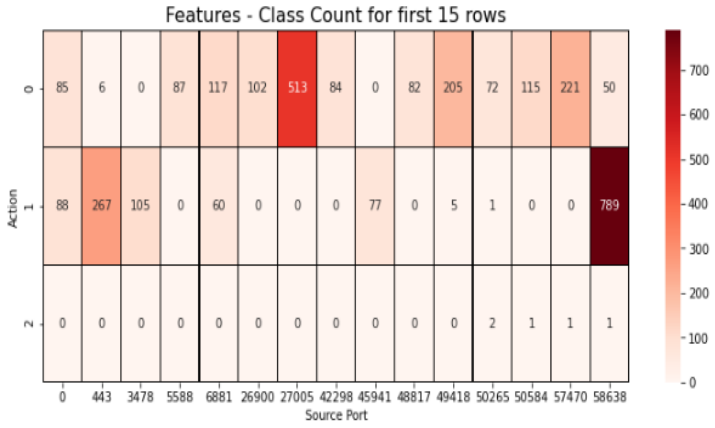


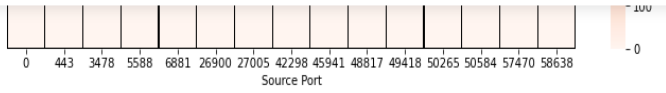
In [14]:
```
# Source and Destination NAT (Network Address Translation)
plt.figure(figsize=(7,7))
plt.scatter(df['NAT Source Port'], df['NAT Destination Port'], alpha=0.05, color = 'black')
plt.xlabel('NAT Source Port')
plt.ylabel('NAT Destination Port')
plt.title('NAT Source Port and NAT Destination Port', size = 16)
plt.show()
```



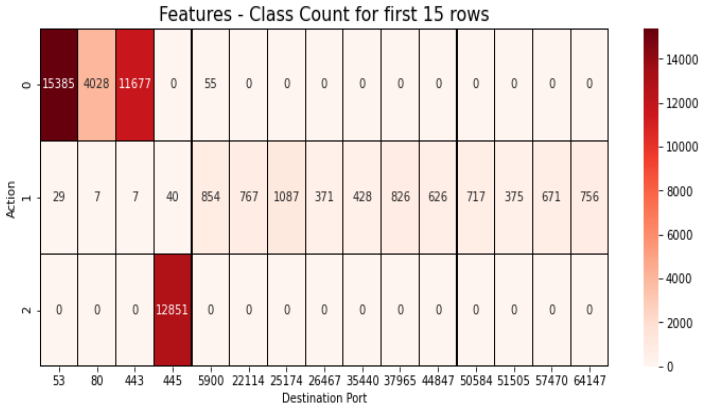
In [15]:
```
# Class vs features
for f in features:
    top15 = df[f].value_counts()[0:15].index.to_list()
    df_temp = df[df[f].isin(top15)]
    ctab = pd.crosstab(df_temp.Action, df_temp[f])
    print('Feature:' + f )
    plt.figure(figsize=(12,5))
    plt.title('Features - Class Count for first 15 rows', size = 16)
    sns.heatmap(ctab, annot=True, fmt='d',
                cmap='Reds',
                linecolor='black',
                linewidths=0.1)
    plt.show()
```

Feature:Source Port

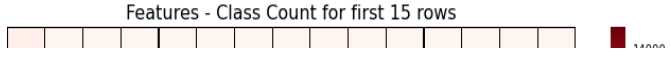




Feature:Destination Port

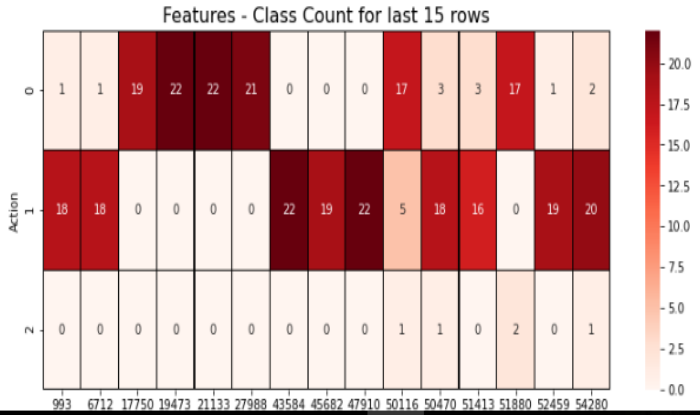


Feature:NAT Source Port

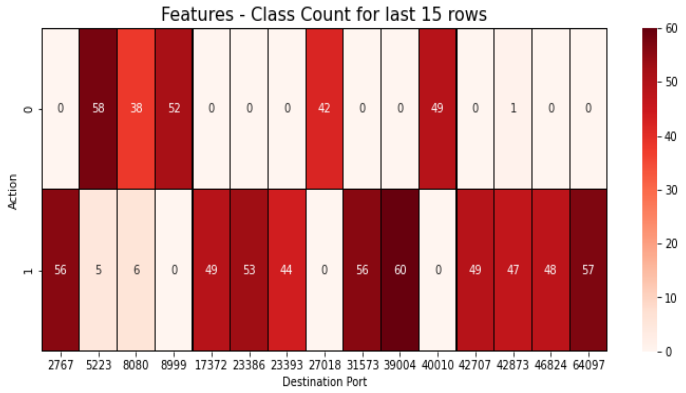


In [16]:
```
# Class vs features
for f in features:
    top15 = df[f].value_counts()[50:65].index.to_list()
    df_temp = df[df[f].isin(top15)]
    ctab = pd.crosstab(df_temp.Action, df_temp[f])
    print('Feature:' + f )
    plt.figure(figsize=(12,5))
    plt.title('Features - Class Count for last 15 rows', size = 16)
    sns.heatmap(ctab, annot=True, fmt='d',
                cmap='Reds',
                linecolor='black',
                linewidths=0.1)
    plt.show()
```

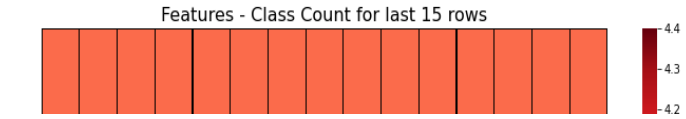
```
Feature:Source Port
```



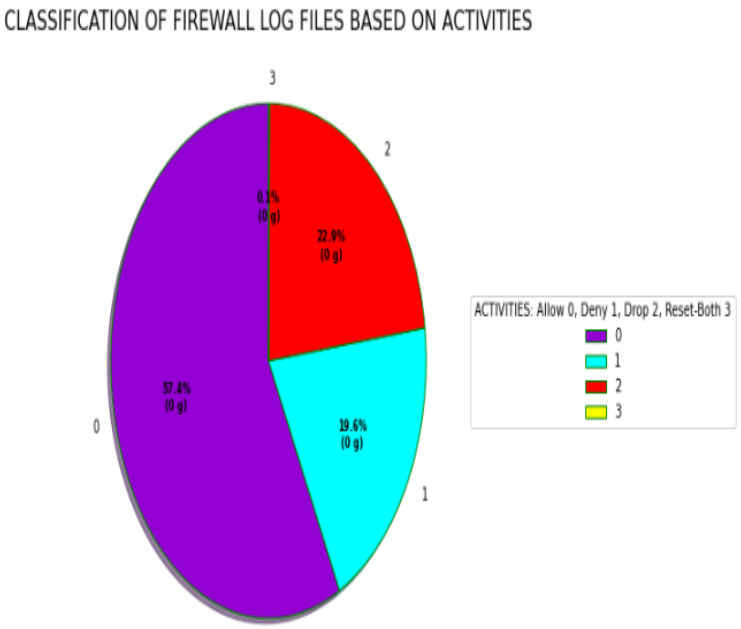

```
Feature:Destination Port
```




```
Feature:NAT Source Port
```

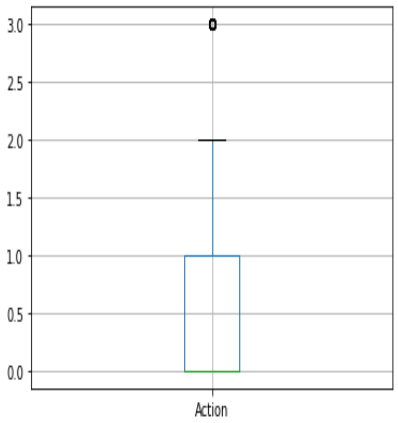


```
plt.title('CLASSIFICATION OF FIREWALL LOG FILES BASED ON ACTIVITIES', size = 16)

# show plot
plt.show()
```



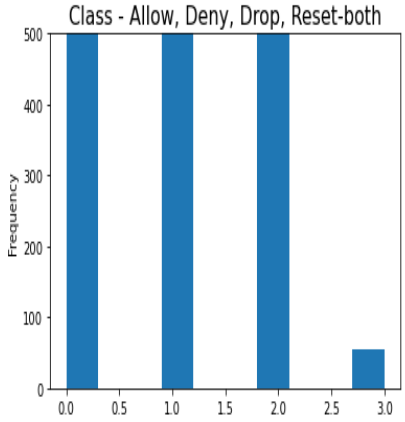
In [165]:
```
boxplot=df.boxplot(column=['Action'])
print(boxplot)
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



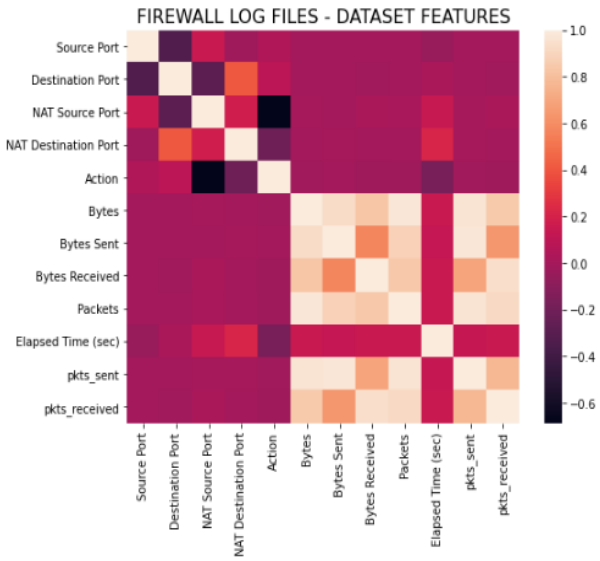
In [171]:
```
plt.title('Class - Allow, Deny, Drop, Reset-both', size = 16)
median_column.plot(kind="hist")
plt.ylim([0, 500])
```

Out[171]: (0.0, 500.0)



In [20]: plt.figure(figsize=(8,6))

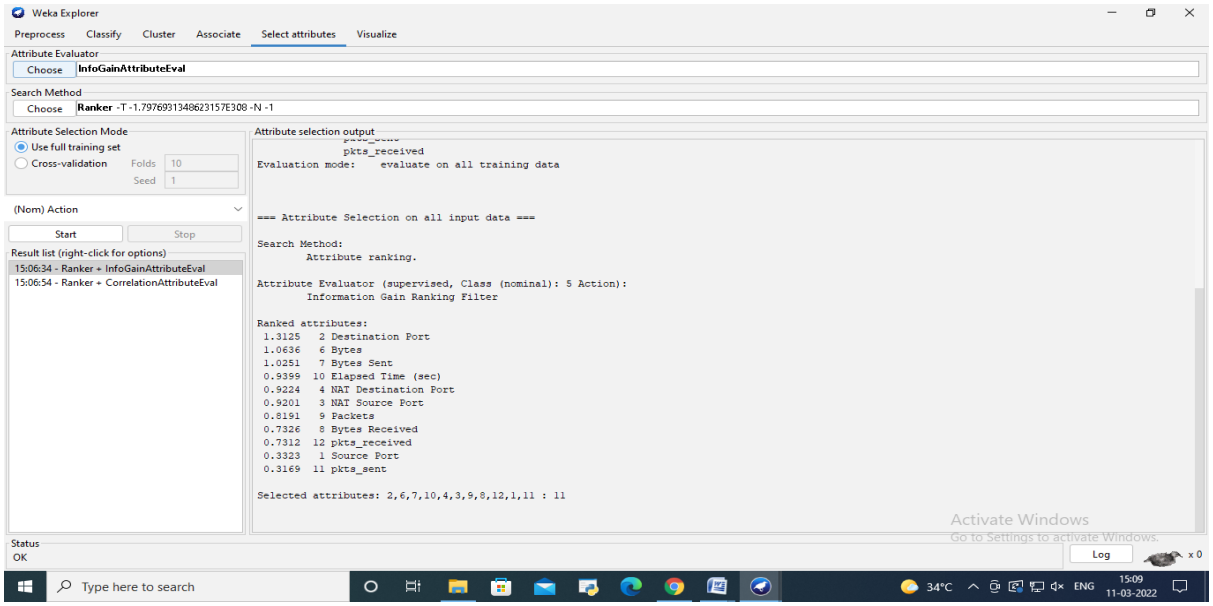
In [18]:
```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr())
plt.title('FIREWALL LOG FILES - DATASET FEATURES', size = 16)
plt.show()
```



Ranker + InfoGainAttributeEval

Attribute Evaluator: Information Gain Attribute Evaluator

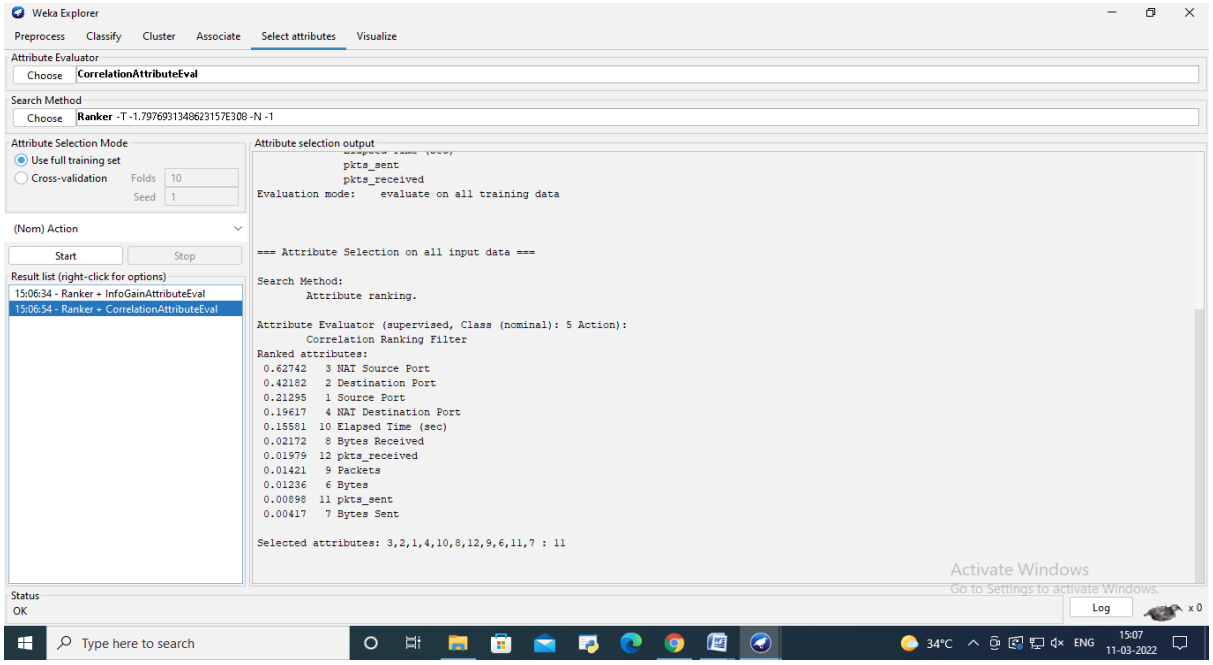
Search method: Ranker



Ranker + CorrelationAttributeEval

Attribute Evaluator: Correlation Attribute Evaluator

Search method: Ranker



PHASE 3 - FEATURE SELECTION

FEATURES SELECTED FROM WEKA TOOL

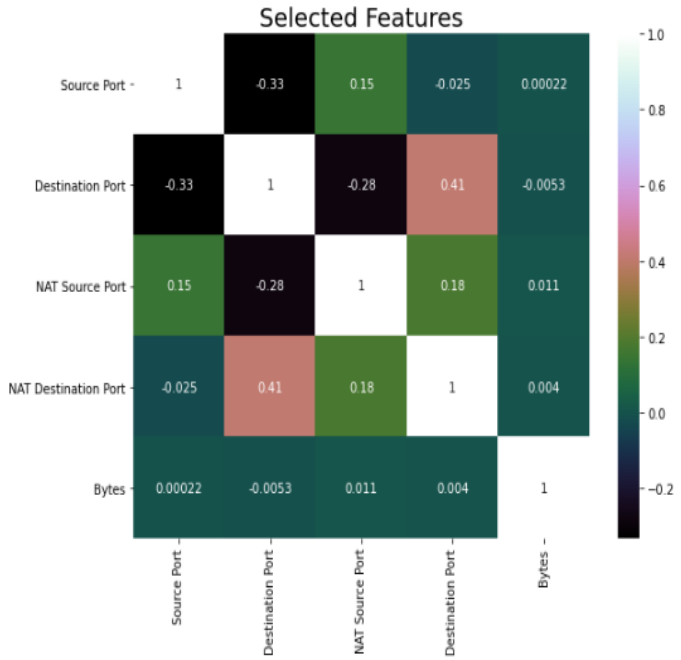
In [19]:
```
# FEATURE SELECTION
# Selected Features from Weka tool

feature_df =  df[[ 'Source Port','Destination Port', 'NAT Source Port', 'NAT Destination Port','Bytes']]
```

In [20]:
```
feature_df
```

Out[20]:

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Bytes
0	57222	53	54587	53	177
1	56258	3389	56258	3389	4768
2	6881	50321	43265	50321	238
3	50553	3389	50553	3389	3327
4	50002	443	45848	443	25358
...	...	...	...	...	...
65527	63691	80	13237	80	314
65528	50964	80	13485	80	4680740
65529	54871	445	0	0	70
65530	54870	445	0	0	70
65531	54867	445	0	0	70

```
plt.figure(figsize=(9,7))
plt.title("Selected Features",size=20)
sns.heatmap(feature_df.corr(), annot=True, cmap="cubehelix")
plt.show()
```



PHASE 4 - MODEL BUILDING

In [22]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import precision_recall_fscore_support
from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import r2_score
from sklearn.metrics import roc_auc_score

import warnings
warnings.filterwarnings('ignore')
```

In [23]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

In [24]:
```
# independent variable
X=feature_df

# dependent variable
y=df['Action']
```

1) SUPPORT VECTOR MACHINE

In [25]:
```
# MODEL BUILDING
# SUPPORT VECTOR MACHINE

from sklearn import svm
classifier = svm.SVC(kernel='linear', gamma='auto', C= 2, probability = True)
classifier.fit(X_train, y_train)

y_predict = classifier.predict(X_test)
print(y_predict)

from sklearn.metrics import classification_report

print(classification_report(y_test, y_predict))
print('Support Vector Machine Accuracy: {:.02%}'.format(classifier.score(X_test, y_test)))
```

```
[1 0 0 ... 0 0 1]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7545
           1       0.99      0.97      0.98      2994
           2       0.96      1.00      0.98      2562
           3       0.00      0.00      0.00         6

    accuracy                           0.99     13107
   macro avg       0.74      0.74      0.74     13107
weighted avg       0.99      0.99      0.99     13107

Support Vector Machine Accuracy: 99.03%
```

In [28]:
```
print("Accuracy on training set for Support Vector Machine Classifier: {:.02%}".format(accuracy_score(y_train,classifier.predict(
print("Accuracy on test set for Support Vector Machine Classifier: {:.02%}".format(accuracy_score(y_test,classifier.predict(X_te
```

```
Accuracy on training set for Support Vector Machine Classifier: 99.03%
Accuracy on test set for Support Vector Machine Classifier: 99.03%
```

In [29]:
```
print("SUPPORT VECTOR MACHINE")
print("Precision score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for SVM Classifier: {:.4f}".format(precision_score(y_train,classifier.predict(X_train),average='mic
print("micro avg on test set for SVM Classifier: {:.4f}\n".format(precision_score(y_test,classifier.predict(X_test),average='micr

print("macro avg on train set for SVM Classifier: {:.4f}".format(precision_score(y_train,classifier.predict(X_train),average='mac
print("macro avg on test set for SVM Classifier: {:.4f}\n".format(precision_score(y_test,classifier.predict(X_test),average='macr

print("weighted avg on train set for SVM Classifier: {:.4f}".format(precision_score(y_train,classifier.predict(X_train),average='
print("weighted avg on test set for SVM Classifier: {:.4f}\n".format(precision_score(y_test,classifier.predict(X_test),average='w
```

```
SUPPORT VECTOR MACHINE
Precision score - micro avg, macro avg and weighted avg:

micro avg on train set for SVM Classifier: 0.9903
micro avg on test set for SVM Classifier: 0.9903

macro avg on train set for SVM Classifier: 0.7386
macro avg on test set for SVM Classifier: 0.7385

weighted avg on train set for SVM Classifier: 0.9896
weighted avg on test set for SVM Classifier: 0.9901
```

In [30]:
```
print("SUPPORT VECTOR MACHINE")
print("f1_score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for SVM Classifier: {:.4f}".format(f1_score(y_train,classifier.predict(X_train),average='micro')))
print("micro avg on test set for SVM Classifier: {:.4f}\n".format(f1_score(y_test,classifier.predict(X_test),average='micro')))

print("macro avg on train set for SVM Classifier: {:.4f}".format(f1_score(y_train,classifier.predict(X_train),average='macro')))
print("macro avg on test set for SVM Classifier: {:.4f}\n".format(f1_score(y_test,classifier.predict(X_test),average='macro')))

print("weighted avg on train set for SVM Classifier: {:.4f}".format(f1_score(y_train,classifier.predict(X_train),average='weighte
print("weighted avg on test set for SVM Classifier: {:.4f}\n".format(f1_score(y_test,classifier.predict(X_test),average='weighted
```

```
SUPPORT VECTOR MACHINE
f1_score - micro avg, macro avg and weighted avg:

micro avg on train set for SVM Classifier: 0.9903
micro avg on test set for SVM Classifier: 0.9903

macro avg on train set for SVM Classifier: 0.7399
macro avg on test set for SVM Classifier: 0.7397

weighted avg on train set for SVM Classifier: 0.9898
weighted avg on test set for SVM Classifier: 0.9901
```

In [31]:
```
print("SUPPORT VECTOR MACHINE")
print("recall_score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for SVM Classifier: {:.4f}".format(recall_score(y_train,classifier.predict(X_train),average='micro'
print("micro avg on test set for SVM Classifier: {:.4f}\n".format(recall_score(y_test,classifier.predict(X_test),average='micro'

print("macro avg on train set for SVM Classifier: {:.4f}".format(recall_score(y_train,classifier.predict(X_train),average='macro'
print("macro avg on test set for SVM Classifier: {:.4f}\n".format(recall_score(y_test,classifier.predict(X_test),average='macro'

print("weighted avg on train set for SVM Classifier: {:.4f}".format(recall_score(y_train,classifier.predict(X_train),average='wei
print("weighted avg on test set for SVM Classifier: {:.4f}\n".format(recall_score(y_test,classifier.predict(X_test),average='weig
```

```
SUPPORT VECTOR MACHINE
recall_score - micro avg, macro avg and weighted avg:

micro avg on train set for SVM Classifier: 0.9903
micro avg on test set for SVM Classifier: 0.9903

macro avg on train set for SVM Classifier: 0.7414
macro avg on test set for SVM Classifier: 0.7411

weighted avg on train set for SVM Classifier: 0.9903
weighted avg on test set for SVM Classifier: 0.9903
```

2) NAIVE BAYES CLASSIFIER

In [32]:
```
# NAIVE BAYES CLASSIFIER

from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb = nb.fit(X_train, y_train.ravel())
y_nbforecast=nb.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test, nb.predict(X_test)))
print('Naive Bayes Classifier Accuracy: {:.02%}'.format(nb.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      7545
           1       1.00      1.00      1.00      2994
           2       1.00      1.00      1.00      2562
           3       0.03      0.50      0.06         6

    accuracy                           0.99     13107
   macro avg       0.76      0.87      0.76     13107
weighted avg       1.00      0.99      1.00     13107

Naive Bayes Classifier Accuracy: 99.26%
```

In [33]:
```
print("Accuracy on training set for Naive Bayes Classifier: {:.02%}".format(accuracy_score(y_train,nb.predict(X_train))))
print("Accuracy on test set for Naive Bayes Classifier: {:.02%} ".format(accuracy_score(y_test,nb.predict(X_test))))
```

```
Accuracy on training set for Naive Bayes Classifier: 99.17%
Accuracy on test set for Naive Bayes Classifier: 99.26%
```

In [33]:
```
print("Accuracy on training set for Naive Bayes Classifier: {:.02%}".format(accuracy_score(y_train,nb.predict(X_train))))
print("Accuracy on test set for Naive Bayes Classifier: {:.02%} ".format(accuracy_score(y_test,nb.predict(X_test))))
```

```
Accuracy on training set for Naive Bayes Classifier: 99.17%
Accuracy on test set for Naive Bayes Classifier: 99.26%
```

In [34]:
```
print("NAIVE BAYES CLASSIFIER")
print("Precision score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Naive Bayes Classifier: {:.4f}".format(precision_score(y_train,nb.predict(X_train),average='mic
print("micro avg on test set for Naive Bayes Classifier: {:.4f}\n".format(precision_score(y_test,nb.predict(X_test),average='micr

print("macro avg on train set for Naive Bayes Classifier: {:.4f}".format(precision_score(y_train,nb.predict(X_train),average='mac
print("macro avg on test set for Naive Bayes Classifier: {:.4f}\n".format(precision_score(y_test,nb.predict(X_test),average='macr

print("weighted avg on train set for Naive Bayes Classifier: {:.4f}".format(precision_score(y_train,nb.predict(X_train),average='
print("weighted avg on test set for Naive Bayes Classifier: {:.4f}\n".format(precision_score(y_test,nb.predict(X_test),average='w
```

```
NAIVE BAYES CLASSIFIER
Precision score - micro avg, macro avg and weighted avg:

micro avg on train set for Naive Bayes Classifier: 0.9917
micro avg on test set for Naive Bayes Classifier: 0.9926

macro avg on train set for Naive Bayes Classifier: 0.7544
macro avg on test set for Naive Bayes Classifier: 0.7574

weighted avg on train set for Naive Bayes Classifier: 0.9976
weighted avg on test set for Naive Bayes Classifier: 0.9988
```

In [35]:
```
print("NAIVE BAYES CLASSIFIER")
print("f1_score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Naive Bayes Classifier: {:.4f}".format(f1_score(y_train,nb.predict(X_train),average='micro')))
print("micro avg on test set for Naive Bayes Classifier: {:.4f}\n".format(f1_score(y_test,nb.predict(X_test),average='micro')))

print("macro avg on train set for Naive Bayes Classifier: {:.4f}".format(f1_score(y_train,nb.predict(X_train),average='macro')))
print("macro avg on test set for Naive Bayes Classifier: {:.4f}\n".format(f1_score(y_test,nb.predict(X_test),average='macro')))

print("weighted avg on train set for Naive Bayes Classifier: {:.4f}".format(f1_score(y_train,nb.predict(X_train),average='weighte
print("weighted avg on test set for Naive Bayes Classifier: {:.4f}\n".format(f1_score(y_test,nb.predict(X_test),average='weighted
```

```
NAIVE BAYES CLASSIFIER
f1_score - micro avg, macro avg and weighted avg:

micro avg on train set for Naive Bayes Classifier: 0.9917
micro avg on test set for Naive Bayes Classifier: 0.9926

macro avg on train set for Naive Bayes Classifier: 0.7580
macro avg on test set for Naive Bayes Classifier: 0.7633

weighted avg on train set for Naive Bayes Classifier: 0.9946
weighted avg on test set for Naive Bayes Classifier: 0.9956
```

In [36]:
```
print("NAIVE BAYES CLASSIFIER")
```

In [36]:
```
print("NAIVE BAYES CLASSIFIER")
print("recall_score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Naive Bayes Classifier: {:.4f}".format(recall_score(y_train,nb.predict(X_train),average='micro'
print("micro avg on test set for Naive Bayes Classifier: {:.4f}\n".format(recall_score(y_test,nb.predict(X_test),average='micro')

print("macro avg on train set for Naive Bayes Classifier: {:.4f}".format(recall_score(y_train,nb.predict(X_train),average='macro'
print("macro avg on test set for Naive Bayes Classifier: {:.4f}\n".format(recall_score(y_test,nb.predict(X_test),average='macro')

print("weighted avg on train set for Naive Bayes Classifier: {:.4f}".format(recall_score(y_train,nb.predict(X_train),average='wei
print("weighted avg on test set for Naive Bayes Classifier: {:.4f}\n".format(recall_score(y_test,nb.predict(X_test),average='weig
```

```
NAIVE BAYES CLASSIFIER
recall_score - micro avg, macro avg and weighted avg:

micro avg on train set for Naive Bayes Classifier: 0.9917
micro avg on test set for Naive Bayes Classifier: 0.9926

macro avg on train set for Naive Bayes Classifier: 0.7927
macro avg on test set for Naive Bayes Classifier: 0.8712

weighted avg on train set for Naive Bayes Classifier: 0.9917
weighted avg on test set for Naive Bayes Classifier: 0.9926
```

3) LOGISTIC REGRESSION CLASSIFIER

In [39]:
```
# LOGISTIC REGRESSION

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='lbfgs')
lr = lr.fit(X_train, y_train.ravel())
y_lrforecast=lr.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test, lr.predict(X_test)))
print('Logistic Regression Accuracy: {:.02%}'.format(lr.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      7545
           1       0.99      0.96      0.97      2994
           2       0.94      1.00      0.97      2562
           3       0.00      0.00      0.00         6

    accuracy                           0.99     13107
   macro avg       0.73      0.74      0.74     13107
weighted avg       0.99      0.99      0.99     13107

Logistic Regression Accuracy: 98.57%
```

In [40]:
```
print("Accuracy on training set for Logistic Regression Classifier: {:.02%}".format(accuracy_score(y_train,lr.predict(X_train))))
print("Accuracy on test set for Logistic Regression Classifier: {:.02%} ".format(accuracy_score(y_test,lr.predict(X_test))))
```

```
Accuracy on training set for Logistic Regression Classifier: 98.65%
Accuracy on test set for Logistic Regression Classifier: 98.57%
```

In [39]:
```
print("LOGISTIC REGRESSION CLASSIFIER")
print("Precision score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Logistic Regression Classifier: {:.4f}".format(precision_score(y_train,lr.predict(X_train),aver
print("micro avg on test set for Logistic Regression Classifier: {:.4f}\n".format(precision_score(y_test,lr.predict(X_test),avera

print("macro avg on train set for Logistic Regression Classifier: {:.4f}".format(precision_score(y_train,lr.predict(X_train),aver
print("macro avg on test set for Logistic Regression Classifier: {:.4f}\n".format(precision_score(y_test,lr.predict(X_test),avera

print("weighted avg on train set for Logistic Regression Classifier: {:.4f}".format(precision_score(y_train,lr.predict(X_train),a
print("weighted avg on test set for Logistic Regression Classifier: {:.4f}\n".format(precision_score(y_test,lr.predict(X_test),av
```

```
LOGISTIC REGRESSION CLASSIFIER
Precision score - micro avg, macro avg and weighted avg:

micro avg on train set for Logistic Regression Classifier: 0.9865
micro avg on test set for Logistic Regression Classifier: 0.9857

macro avg on train set for Logistic Regression Classifier: 0.7342
macro avg on test set for Logistic Regression Classifier: 0.7331

weighted avg on train set for Logistic Regression Classifier: 0.9860
weighted avg on test set for Logistic Regression Classifier: 0.9859
```

In [40]:
```
print("LOGISTIC REGRESSION CLASSIFIER")
print("f1_score- micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Logistic Regression Classifier: {:.4f}".format(f1_score(y_train,lr.predict(X_train),average='mi
print("micro avg on test set for Logistic Regression Classifier: {:.4f}\n".format(f1_score(y_test,lr.predict(X_test),average='mic

print("macro avg on train set for Logistic Regression Classifier: {:.4f}".format(f1_score(y_train,lr.predict(X_train),average='ma
print("macro avg on test set for Logistic Regression Classifier: {:.4f}\n".format(f1_score(y_test,lr.predict(X_test),average='mac

print("weighted avg on train set for Logistic Regression Classifier: {:.4f}".format(f1_score(y_train,lr.predict(X_train),average=
print("weighted avg on test set for Logistic Regression Classifier: {:.4f}\n".format(f1_score(y_test,lr.predict(X_test),average='
```

```
LOGISTIC REGRESSION CLASSIFIER
f1_score- micro avg, macro avg and weighted avg:

micro avg on train set for Logistic Regression Classifier: 0.9865
micro avg on test set for Logistic Regression Classifier: 0.9857

macro avg on train set for Logistic Regression Classifier: 0.7364
macro avg on test set for Logistic Regression Classifier: 0.7353

weighted avg on train set for Logistic Regression Classifier: 0.9861
weighted avg on test set for Logistic Regression Classifier: 0.9856
```

In [41]:

In [41]:
```
print("LOGISTIC REGRESSION CLASSIFIER")
print("recall_score- micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Logistic Regression Classifier: {:.4f}".format(recall_score(y_train,lr.predict(X_train),average
print("micro avg on test set for Logistic Regression Classifier: {:.4f}\n".format(recall_score(y_test,lr.predict(X_test),average=

print("macro avg on train set for Logistic Regression Classifier: {:.4f}".format(recall_score(y_train,lr.predict(X_train),average
print("macro avg on test set for Logistic Regression Classifier: {:.4f}\n".format(recall_score(y_test,lr.predict(X_test),average=

print("weighted avg on train set for Logistic Regression Classifier: {:.4f}".format(recall_score(y_train,lr.predict(X_train),aver
print("weighted avg on test set for Logistic Regression Classifier: {:.4f}\n".format(recall_score(y_test,lr.predict(X_test),avera
```

```
LOGISTIC REGRESSION CLASSIFIER
recall_score- micro avg, macro avg and weighted avg:

micro avg on train set for Logistic Regression Classifier: 0.9865
micro avg on test set for Logistic Regression Classifier: 0.9857

macro avg on train set for Logistic Regression Classifier: 0.7390
macro avg on test set for Logistic Regression Classifier: 0.7380

weighted avg on train set for Logistic Regression Classifier: 0.9865
weighted avg on test set for Logistic Regression Classifier: 0.9857
```

4) KNN CLASSIFIER

In [46]:
```
# Knn Classifier
from sklearn.neighbors import KNeighborsClassifier
K = 1
knn = KNeighborsClassifier(n_neighbors=K)
knn = knn.fit(X_train,y_train.ravel())

print("k = {}neighbors , knn test:{}".format(K, knn.score(X_test, y_test)))
print("knn Classifier: {}".format(K, knn.score(X_train, y_train)))

ran = np.arange(1,40)
train_list = []
test_list = []
for i,each in enumerate(ran):
    knn = KNeighborsClassifier(n_neighbors=each)
    knn = knn.fit(X_train, y_train.ravel())
    test_list.append(knn.score(X_test, y_test))
    train_list.append(knn.score(X_train, y_train))

print("Best train: {} , k={}".format(np.max(train_list),train_list.index(np.max(train_list))+1))
print("Best test accuracy: {} , k={}".format(np.max(test_list),test_list.index(np.max(test_list))+1))
```

```
k = 1neighbors , knn test:0.9972533760585947
knn Classifier: 1
Best train: 0.9997329518359561 , k=1
Best test accuracy: 0.9980163271534295 , k=4
```

In [47]:
```
y_knnforecast=knn.predict(X_test)
```

In [48]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, lr.predict(X_test)))
```

In [44]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, lr.predict(X_test)))
print('KNN Accuracy: {:.02%}'.format(knn.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      7545
           1       0.99      0.96      0.97      2994
           2       0.94      1.00      0.97      2562
           3       0.00      0.00      0.00         6

    accuracy                           0.99     13107
   macro avg       0.73      0.74      0.74     13107
weighted avg       0.99      0.99      0.99     13107

KNN Accuracy: 99.10%
```

In [45]:
```
print("Accuracy on training set for Knn Classifier: {:.02%}".format(accuracy_score(y_train,knn.predict(X_train))))
print("Accuracy on test set for Knn Classifier: {:.02%}".format(accuracy_score(y_test,knn.predict(X_test))))
```

```
Accuracy on training set for Knn Classifier: 99.11%
Accuracy on test set for Knn Classifier: 99.10%
```

In [46]:
```
print("KNN CLASSIFIER")
print("Precision score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Knn Classifier: {:.4f}".format(precision_score(y_train,knn.predict(X_train),average='micro')))
print("micro avg on test set for Knn Classifier: {:.4f}\n".format(precision_score(y_test,knn.predict(X_test),average='micro')))

print("macro avg on train set for Knn Classifier: {:.4f}".format(precision_score(y_train,knn.predict(X_train),average='macro')))
print("macro avg on test set for Knn Classifier: {:.4f}\n".format(precision_score(y_test,knn.predict(X_test),average='macro')))

print("weighted avg on train set for Knn Classifier: {:.4f}".format(precision_score(y_train,knn.predict(X_train),average='weighte
print("weighted avg on test set for Knn Classifier: {:.4f}\n".format(precision_score(y_test,knn.predict(X_test),average='weighted
```

```
KNN CLASSIFIER
Precision score - micro avg, macro avg and weighted avg:

micro avg on train set for Knn Classifier: 0.9911
micro avg on test set for Knn Classifier: 0.9910

macro avg on train set for Knn Classifier: 0.7400
macro avg on test set for Knn Classifier: 0.7398

weighted avg on train set for Knn Classifier: 0.9903
weighted avg on test set for Knn Classifier: 0.9906
```

In [47]:
```
print("KNN CLASSIFIER")
print("f1_score- micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Knn Classifier: {:.4f}".format(f1_score(y_train,knn.predict(X_train),average='micro')))
print("micro avg on test set for Knn Classifier: {:.4f}\n".format(f1_score(y_test,knn.predict(X_test),average='micro')))

print("macro avg on train set for Knn Classifier: {:.4f}".format(f1_score(y_train,knn.predict(X_train),average='macro')))
print("macro avg on test set for Knn Classifier: {:.4f}\n".format(f1_score(y_test,knn.predict(X_test),average='macro')))

print("weighted avg on train set for Knn Classifier: {:.4f}".format(f1_score(y_train,knn.predict(X_train),average='weighted')))
print("weighted avg on test set for Knn Classifier: {:.4f}\n".format(f1_score(y_test,knn.predict(X_test),average='weighted')))
```

```
KNN CLASSIFIER
f1_score- micro avg, macro avg and weighted avg:

micro avg on train set for Knn Classifier: 0.9911
micro avg on test set for Knn Classifier: 0.9910

macro avg on train set for Knn Classifier: 0.7414
macro avg on test set for Knn Classifier: 0.7413

weighted avg on train set for Knn Classifier: 0.9907
weighted avg on test set for Knn Classifier: 0.9908
```

In [49]:
```
print("KNN CLASSIFIER")
print("recall_score - micro avg, macro avg and weighted avg:\n")

print("micro avg on train set for Knn Classifier: {:.4f}".format(recall_score(y_train,knn.predict(X_train),average='micro')))
print("micro avg on test set for Knn Classifier: {:.4f}\n".format(recall_score(y_test,knn.predict(X_test),average='micro')))

print("macro avg on train set for Knn Classifier: {:.4f}".format(recall_score(y_train,knn.predict(X_train),average='macro')))
print("macro avg on test set for Knn Classifier: {:.4f}\n".format(recall_score(y_test,knn.predict(X_test),average='macro')))

print("weighted avg on train set for Knn Classifier: {:.4f}".format(recall_score(y_train,knn.predict(X_train),average='weighted')
print("weighted avg on test set for Knn Classifier: {:.4f}\n".format(recall_score(y_test,knn.predict(X_test),average='weighted')
```

```
KNN CLASSIFIER
recall_score - micro avg, macro avg and weighted avg:

micro avg on train set for Knn Classifier: 0.9911
micro avg on test set for Knn Classifier: 0.9910

macro avg on train set for Knn Classifier: 0.7429
macro avg on test set for Knn Classifier: 0.7429

weighted avg on train set for Knn Classifier: 0.9911
weighted avg on test set for Knn Classifier: 0.9910
```

PHASE 5 - COMPARITIVE ANALYSIS OF ALGORITHMS

In [82]:
```
Model = []

Model_Test_Accuracy = []

def storeResults(model, a2):
  Model.append(model)
  Model_Test_Accuracy.append(a2)
```

In [83]:
```
SVM_Train_Accuracy=("{:.02%}".format(accuracy_score(y_train,classifier.predict(X_train))))

SVM_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,classifier.predict(X_test))))

NB_Train_Accuracy=("{:.02%}".format(accuracy_score(y_train,nb.predict(X_train))))

NB_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,nb.predict(X_test))))

lr_Train_Accuracy=("{:.02%}".format(accuracy_score(y_train,lr.predict(X_train))))

lr_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,lr.predict(X_test))))

knn_Train_Accuracy=("{:.02%}".format(accuracy_score(y_train,knn.predict(X_train))))

knn_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,knn.predict(X_test))))
```

In [84]:
```
storeResults('SUPPORT VECTOR MACHINE', SVM_Test_Accuracy)
```

In [85]:
```
storeResults('NAIVE BAYES CLASSIFIER', NB_Test_Accuracy)
```

In [86]:
```
storeResults('LOGISTIC REGRESSION CLASSIFIER', lr_Test_Accuracy)
```

In [87]:
```
storeResults('KNN CLASSIFIER', knn_Test_Accuracy)
```

In [89]:
```
result = pd.DataFrame({'Model': Model, 'Test_Accuracy' : Model_Test_Accuracy})
```

In [90]:
```
result
```

Out[90]:

	Model	Test_Accuracy
0	SUPPORT VECTOR MACHINE	99.03%
1	NAIVE BAYES CLASSIFIER	99.26%
2	LOGISTIC REGRESSION CLASSIFIER	98.57%
3	KNN CLASSIFIER	99.10%

In [96]:
```
# Highlight the Max values in each column
print("COMPARITIVE ANALYSIS OF MACHINE LEARNING ALGORITHMS")

def highlight_max(s):
    '''
    highlight the maximum in a Series green.
    '''
    is_max = s == s.max()
    return ['background-color: violet' if v else '' for v in is_max]

print("\nHighlight the maximum value in each column:")
result.style.apply(highlight_max,subset=pd.IndexSlice[:, ['Test_Accuracy']])
```

```
COMPARITIVE ANALYSIS OF MACHINE LEARNING ALGORITHMS

Highlight the maximum value in each column:
```

Out[96]:

	Model	Test_Accuracy
0	SUPPORT VECTOR MACHINE	99.03%
1	NAIVE BAYES CLASSIFIER	99.26%
2	LOGISTIC REGRESSION CLASSIFIER	98.57%
3	KNN CLASSIFIER	99.10%

In [109]:
```
Analysis_Model = []

Model_precision_micro = []
Model_precision_macro = []
Model_precision_weighted = []

Model_f1_score_micro = []
Model_f1_score_macro = []
Model_f1_score_weighted = []

Model_recall_micro = []
Model_recall_macro = []
Model_recall_weighted = []


def storeResults(model, b,c,d, i,j,k, x,y,z):
  Analysis_Model.append(model)
  Model_precision_micro.append(b)
  Model_precision_macro.append(c)
  Model_precision_weighted.append(d)

  Model_f1_score_micro.append(i)
  Model_f1_score_macro.append(j)
  Model_f1_score_weighted.append(k)

  Model_recall_micro.append(x)
  Model_recall_macro.append(y)
  Model_recall_weighted.append(z)
```

In [110]:

```
SVM_Precision_micro=("{:.4f}".format(precision_score(y_test,classifier.predict(X_test),average='micro')))
SVM_Precision_macro=("{:.4f}".format(precision_score(y_test,classifier.predict(X_test),average='macro')))
SVM_Precision_weighted=("{:.4f}".format(precision_score(y_test,classifier.predict(X_test),average='weighted')))
SVM_f1_score_micro=("{:.4f}".format(f1_score(y_test,classifier.predict(X_test),average='micro')))
SVM_f1_score_macro=("{:.4f}".format(f1_score(y_test,classifier.predict(X_test),average='macro')))
SVM_f1_score_weighted=("{:.4f}".format(f1_score(y_test,classifier.predict(X_test),average='weighted')))
SVM_recall_score_micro=("{:.4f}".format(recall_score(y_test,classifier.predict(X_test),average='micro')))
SVM_recall_score_macro=("{:.4f}".format(recall_score(y_test,classifier.predict(X_test),average='macro')))
SVM_recall_score_weighted=("{:.4f}".format(recall_score(y_test,classifier.predict(X_test),average='weighted')))
```

In [111]:

```
NB_Precision_micro=("{:.4f}".format(precision_score(y_test,nb.predict(X_test),average='micro')))
NB_Precision_macro=("{:.4f}".format(precision_score(y_test,nb.predict(X_test),average='macro')))
NB_Precision_weighted=("{:.4f}".format(precision_score(y_test,nb.predict(X_test),average='weighted')))
NB_f1_score_micro=("{:.4f}".format(f1_score(y_test,nb.predict(X_test),average='micro')))
NB_f1_score_macro=("{:.4f}".format(f1_score(y_test,nb.predict(X_test),average='macro')))
```



```
NB_Precision_micro=("{:.4f}".format(precision_score(y_test,nb.predict(X_test),average='micro')))
NB_Precision_macro=("{:.4f}".format(precision_score(y_test,nb.predict(X_test),average='macro')))
NB_Precision_weighted=("{:.4f}".format(precision_score(y_test,nb.predict(X_test),average='weighted')))
NB_f1_score_micro=("{:.4f}".format(f1_score(y_test,nb.predict(X_test),average='micro')))
NB_f1_score_macro=("{:.4f}".format(f1_score(y_test,nb.predict(X_test),average='macro')))
NB_f1_score_weighted=("{:.4f}".format(f1_score(y_test,nb.predict(X_test),average='weighted')))
NB_recall_score_micro=("{:.4f}".format(recall_score(y_test,nb.predict(X_test),average='micro')))
NB_recall_score_macro=("{:.4f}".format(recall_score(y_test,nb.predict(X_test),average='macro')))
NB_recall_score_weighted=("{:.4f}".format(recall_score(y_test,nb.predict(X_test),average='weighted')))
```


In [112]:

```
lr_Precision_micro=("{:.4f}".format(precision_score(y_test,lr.predict(X_test),average='micro')))
lr_Precision_macro=("{:.4f}".format(precision_score(y_test,lr.predict(X_test),average='macro')))
lr_Precision_weighted=("{:.4f}".format(precision_score(y_test,lr.predict(X_test),average='weighted')))
lr_f1_score_micro=("{:.4f}".format(f1_score(y_test,lr.predict(X_test),average='micro')))
lr_f1_score_macro=("{:.4f}".format(f1_score(y_test,lr.predict(X_test),average='macro')))
lr_f1_score_weighted=("{:.4f}".format(f1_score(y_test,lr.predict(X_test),average='weighted')))
```

```
lr_f1_score_weighted=("{:.4f}".format(f1_score(y_test,lr.predict(X_test),average='weighted')))

lr_recall_score_micro=("{:.4f}".format(recall_score(y_test,lr.predict(X_test),average='micro')))

lr_recall_score_macro=("{:.4f}".format(recall_score(y_test,lr.predict(X_test),average='macro')))

lr_recall_score_weighted=("{:.4f}".format(recall_score(y_test,lr.predict(X_test),average='weighted')))
```

In [113]:

```
knn_Precision_micro=("{:.4f}".format(precision_score(y_test,knn.predict(X_test),average='micro')))

knn_Precision_macro=("{:.4f}".format(precision_score(y_test,knn.predict(X_test),average='macro')))

knn_Precision_weighted=("{:.4f}".format(precision_score(y_test,knn.predict(X_test),average='weighted')))

knn_f1_score_micro=("{:.4f}".format(f1_score(y_test,knn.predict(X_test),average='micro')))

knn_f1_score_macro=("{:.4f}".format(f1_score(y_test,knn.predict(X_test),average='macro')))

knn_f1_score_weighted=("{:.4f}".format(f1_score(y_test,knn.predict(X_test),average='weighted')))

knn_recall_score_micro=("{:.4f}".format(recall_score(y_test,knn.predict(X_test),average='micro')))

knn_recall_score_macro=("{:.4f}".format(recall_score(y_test,knn.predict(X_test),average='macro')))

knn_recall_score_weighted=("{:.4f}".format(recall_score(y_test,knn.predict(X_test),average='weighted')))
```

```
                                    SVM_recall_score_micro, SVM_recall_score_macro, SVM_recall_score_weighted)
```

In [116]:

```
storeResults('NAIVE BAYES CLASSIFIER', NB_Precision_micro, NB_Precision_macro,
                                    NB_Precision_weighted,
                                    NB_f1_score_micro, NB_f1_score_macro, NB_f1_score_weighted,
                                    NB_recall_score_micro, NB_recall_score_macro, NB_recall_score_weighted)
```

In [117]:

```
storeResults('LOGISTIC REGRESSION CLASSIFIER', lr_Precision_micro, lr_Precision_macro,
                                    lr_Precision_weighted,
                                    lr_f1_score_micro, lr_f1_score_macro, lr_f1_score_weighted,
                                    lr_recall_score_micro, lr_recall_score_macro, lr_recall_score_weighted)
```

In [118]:

```
storeResults('KNN CLASSIFIER', knn_Precision_micro, knn_Precision_macro,
                                    knn_Precision_weighted,
                                    knn_f1_score_micro, knn_f1_score_macro,  knn_f1_score_weighted,
                                    knn_recall_score_micro, knn_recall_score_macro, knn_recall_score_weighted)
```

In [119]:

```
results = pd.DataFrame({'Model': Analysis_Model,
                        'Precision_micro' : Model_precision_micro, 'Precision_macro' : Model_precision_macro,
                        'Precision_weighted' : Model_precision_weighted, 'f1_score_micro' : Model_f1_score_micro,
                        'f1_score_macro' : Model_f1_score_macro, 'f1_score_weighted' : Model_f1_score_weighted,
                        'recall_score_micro' : Model_recall_micro, 'recall_score_macro' : Model_recall_macro,
                        'recall_score_weighted' : Model_recall_weighted})
```

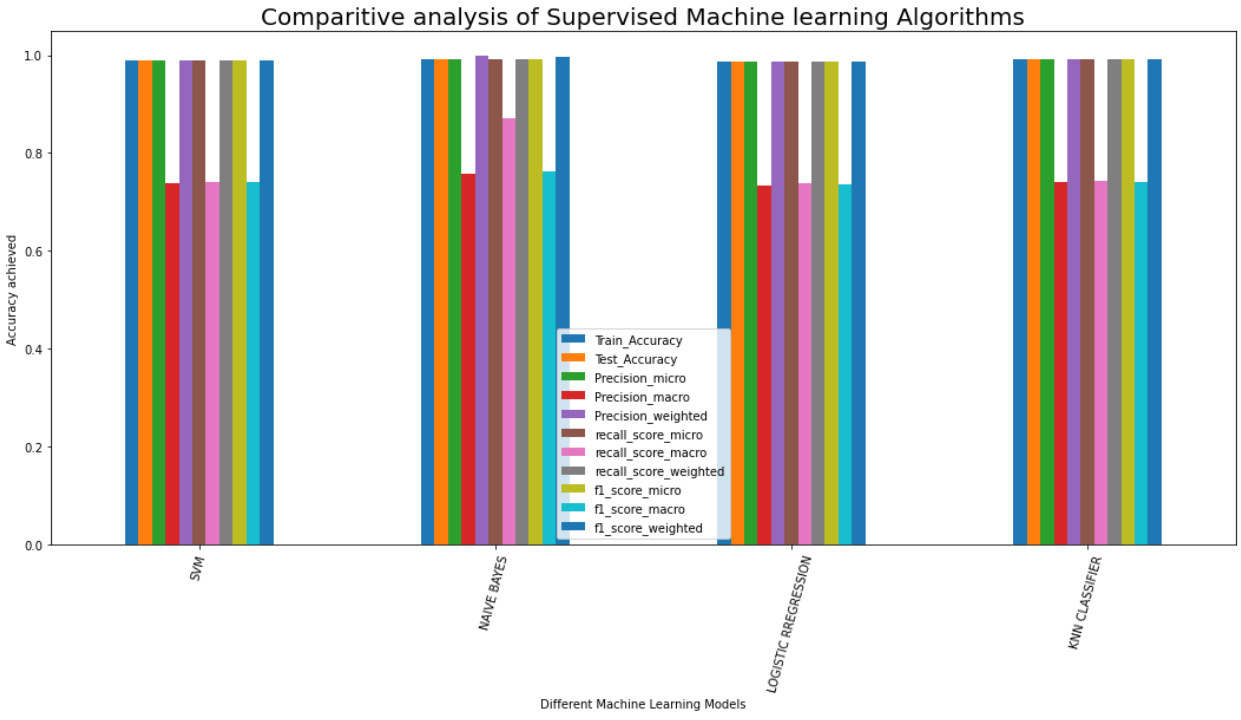
```
                                                    'recall_score_micro', 'recall_score_macro', 'recall_score_weighted']])
```

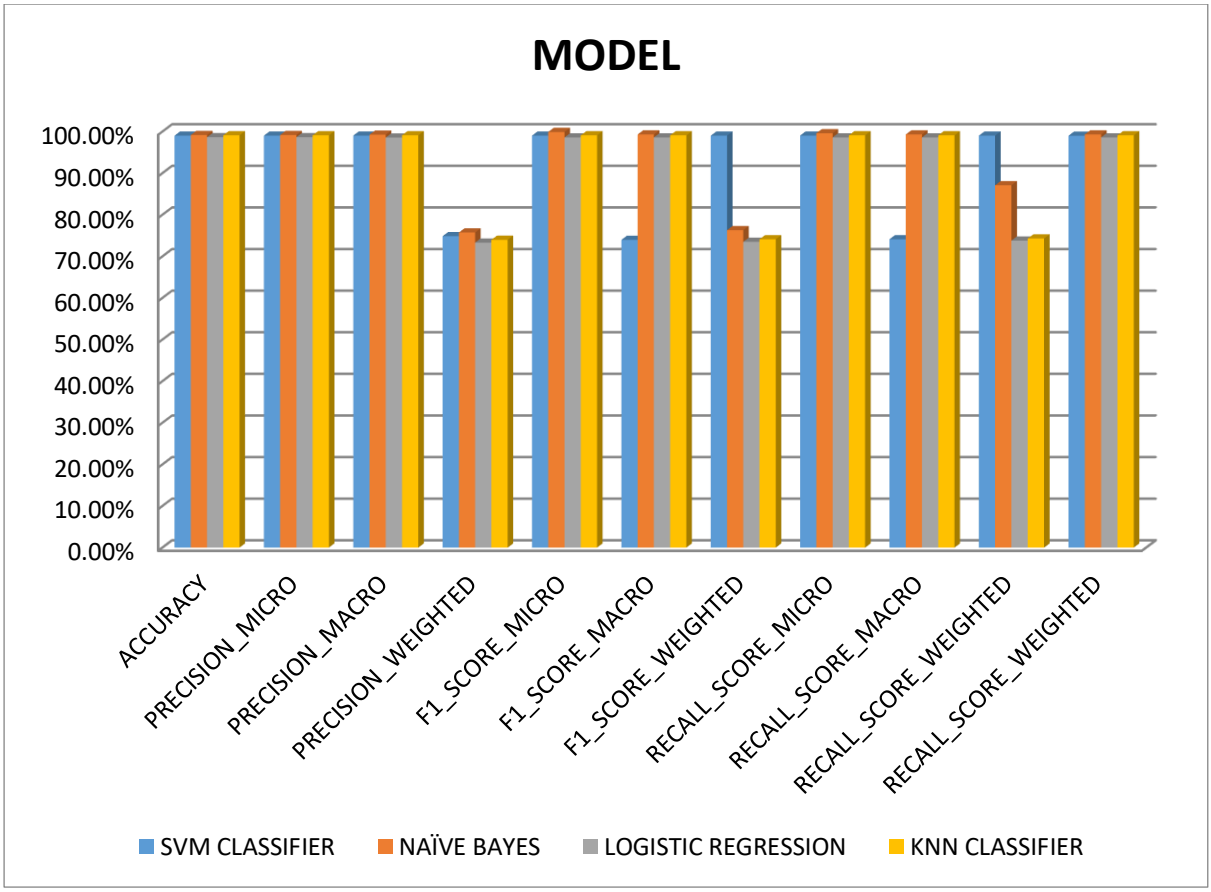
CLASSIFICATION METRICS: COMPARITIVE ANALYSIS OF MACHINE LEARNING ALGORITHMS

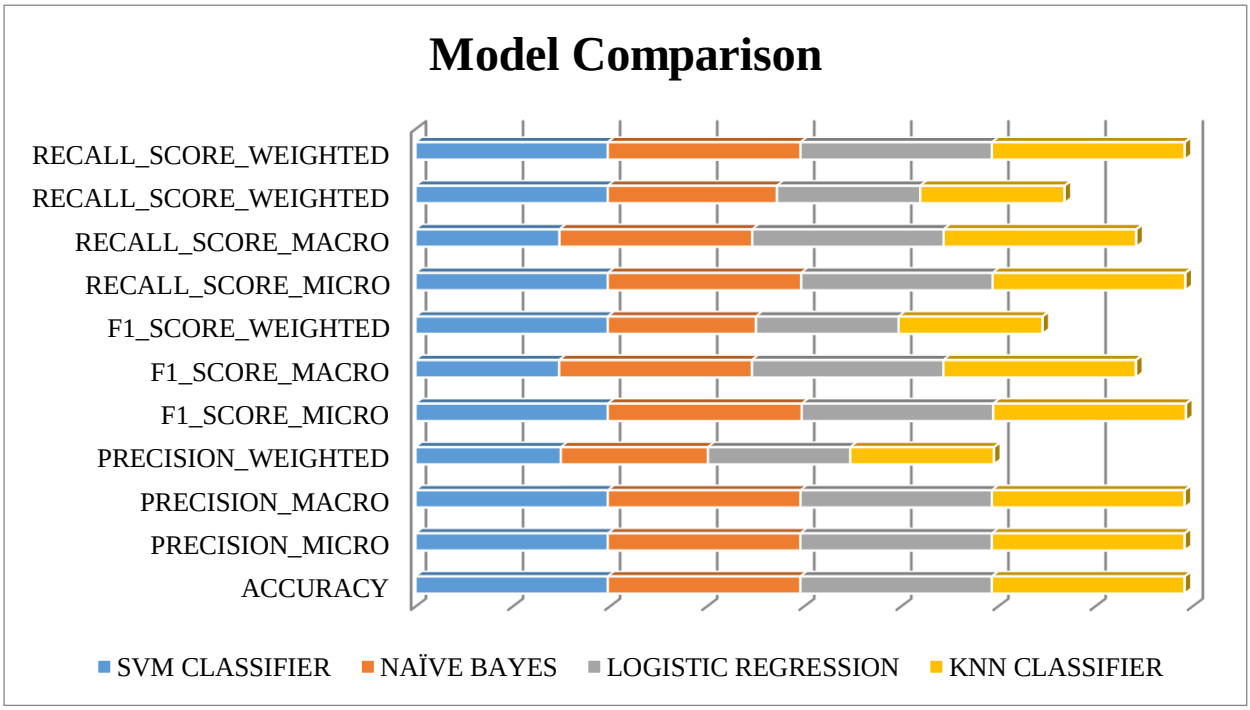
Highlight the maximum value in each column:

Out[139]:

	Model	Precision_micro	Precision_macro	Precision_weighted	f1_score_micro	f1_score_macro	f1_score_weighted	recall_score_micro	recall_score_m
0	SVM CLASSIFIER	0.9903	0.7385	0.9901	0.9903	0.7397	0.9901	0.9903	0.
1	NAIVE BAYES CLASSIFIER	0.9926	0.7574	0.9988	0.9926	0.7633	0.9956	0.9926	0.
2	LOGISTIC REGRESSION CLASSIFIER	0.9857	0.7331	0.9859	0.9857	0.7353	0.9856	0.9857	0.
3	KNN CLASSIFIER	0.9910	0.7398	0.9906	0.9910	0.7413	0.9908	0.9910	0.




MODEL
100.00%
90.00%
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
ACCURACY
PRECISION_MICRO
PRECISION_MACRO
PRECISION_WEIGHTED
F1_SCORE_MICRO
F1_SCORE_MACRO
F1_SCORE_WEIGHTED
RECALL_SCORE_MICRO
RECALL_SCORE_MACRO
RECALL_SCORE_WEIGHTED
RECALL_SCORE_WEIGHTED
SVM CLASSIFIER
NAÏVE BAYES
LOGISTIC REGRESSION
KNN CLASSIFIER


Model Comparison
RECALL_SCORE_WEIGHTED
RECALL_SCORE_WEIGHTED
RECALL_SCORE_MACRO
RECALL_SCORE_MICRO
F1_SCORE_WEIGHTED
F1_SCORE_MACRO
F1_SCORE_MICRO
PRECISION_WEIGHTED
PRECISION_MACRO
PRECISION_MICRO
ACCURACY
SVM CLASSIFIER
NAÏVE BAYES
LOGISTIC REGRESSION
KNN CLASSIFIER