



# **Intrusion Detection System using Machine Learning and Deep Learning**

by

**Rupal Mayo Diline DSouza**

**Supervisor: Dr Valerio Selis**

Dissertation submitted to the University of Liverpool in partial fulfilment of the requirements for the degree of Master of Science in Microelectronic Systems.

September, 2024

## **Abstract**

An increasing use of the internet has led to many cyber-attacks on the network. Several attacks have been recorded on the network, one of the most prominent attacks is Distributed Denial of Service (DDoS) attack. This attack over floods the target user device with the unwanted messages causing congestion in the device. The intrusion detection system (IDS) is designed in this project is a software program designed to identify the abnormal activities in the network and provide availability of a system. The primary aim of this project is to create an AI-driven Intrusion Detection System, capable of detecting Distributed Denial of Service attack within the communication network. The model uses pre-existing dataset CIC-DDoS 2019 to train and test the model developed.

The project is implemented using Machine learning models- Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF) and KNN for detecting the attack. The performance of the model is determined by calculating accuracy, precision, F1-score and Recall. The models are implemented without feature selection and feature selection method. The performance of all the models is compared to find the best model for DDoS attack.

The Machine learning model is also compared with the existing deep learning models like CNN and LSTM. It is found that Random Forest has best accuracy among the four machine learning models.

# Contents

---

Acknowledgements	i
List of Figures	ii
List of Tables	iii
1 INTRODUCTION.....	1
2 BACKGROUND AND LITERATURE REVIEW .....	3
2.1 Introduction to intrusion detection system	3
2.2 Related Work	4
2.3 Machine Learning and Deep Learning Model	6
2.3.1 Random Forest Classifier	6
2.3.2 Support Vector Machines (SVM)	7
2.3.3 Logistic Regression	7
2.3.4 KNN Model	8
2.3.5 Convolutional Neural Network (CNN) Model	8
2.3.6 Long Short-Term Memory (LSTM) Model	10
2.4 Equations	10
3 METHODOLOGY .....	12
3.1 Design of the project	12
3.2 Data Pre-processing	13
3.3 Splitting of the dataset	14
3.4 Label Encoder.	14
3.5 Feature selection	14
3.6 Classification	15
3.7 Cross validation	15
3.8 Machine Learning Model	16
3.8.1 Logistic Regression (LR)	16
3.8.2 Support Vector Machines (SVM)	17
3.8.3 Random Forest (RF) Model	17
3.8.4 KNN model	17
3.9 Deep Learning Models	18
3.9.1 CNN Model	18
3.9.2 LSTM	20
4 RESULTS AND DISCUSSION .....	22

4.1	Binary Classification	22
4.1.1	Without Feature selection	22
4.1.2	Principal Component Analysis (PCA)	25
4.2	Multiclass classification	27
4.2.1	Without Feature selection	27
4.2.2	Principal Component Analysis (PCA)	32
4.2.3	SelectKBest Features	37
4.3	Deep Learning Model	50
4.3.1	CNN Model	51
4.3.2	LSTM Model	55
4.4	Discussions	59
5	CONCLUSIONS .....	61
6	REFERENCES .....	63
7.	APPENDICES .....	66
	Appendix A .....	66
	A.1 Gantt Chart .....	66
	A.1 Risk Management .....	67
	Appendix B .....	69
	B.1 The 88 features of dataset .....	69
	Appendix C	
	C.1 Code for Intrusion Detection System .....	75

## **Acknowledgement**

---

I would like to thank Almighty God for all the blessing and good health showered on me throughout my life.

I would like to express my sincere gratitude to my Supervisor Dr Valerio Selis for his invaluable guidance and constant encouragement during this project. His timely suggestion and feedback have contributed a lot in the successful completion of the project.

I am also grateful to my friends and Department of Electrical and Electronics, University of Liverpool for all the support I have received.

I would like to thank my Mother, Father and Sister and her family for all the blessings and the tireless encouragement they have given me.

# List of Figures

Figure 2.1: Random forest classifier .....	7
Figure 2.2: KNN Example .....	8
Figure 2.3: Structure of CNN .....	9
Figure 3.1: Different stages of the project.....	13
Figure 3.2: Cross validation using 5-folds .....	16
Figure 3.3: Flow diagram of Machine learning .....	16
Figure 3.4: Layers of CNN model .....	19
Figure 3.5: Layers of LSTM model .....	21
Figure 4.1: Binary classification of Logistic Regression without Feature selection ....	23
Figure 4.2: Binary classification of SVM without Feature selection .....	23
Figure 4.3: Binary classification of Random Forest without Feature selection .....	24
Figure 4.4: Binary classification of KNN without Feature selection .....	24
Figure 4.5: Binary classification of Logistic Regression using PCA .....	25
Figure 4.6: Binary classification of SVM using PCA .....	26
Figure 4.7: Binary classification of Random Forest using PCA .....	26
Figure 4.8: Binary classification of KNN using PCA .....	27
Figure 4.9: Multiclass classification of Logistic Regression without feature selection	29
Figure 4.10: Multiclass classification of SVM without feature selection .....	30
Figure 4.11: Multiclass classification of Random Forest without feature selection .....	31
Figure 4.12: Multiclass classification of KNN without feature selection .....	32
Figure 4.13: Multiclass classification for Logistic Regression using PCA .....	34
Figure 4.14: Multiclass classification for SVM using PCA .....	35
Figure 4.15: Multiclass classification for Random Forest using PCA .....	36
Figure 4.16: Multiclass classification for KNN using PCA .....	37
Figure 4.17: Multiclass classification for Logistic Regression using SelectKBest =10	39
Figure 4.18: Multiclass classification for SVM using SelectKBest =10 .....	40
Figure 4.19: Multiclass classification for RF using SelectKBest =10 .....	41
Figure 4.20: Multiclass classification for KNN using SelectKBest =10 .....	42
Figure 4.21: Multiclass classification for Logistic Regression using SelectKBest =20 ..	43
Figure 4.22: Multiclass classification for SVM using SelectKBest =20 .....	44
Figure 4.23: Multiclass classification for RF using SelectKBest =20 .....	45
Figure 4.24: Multiclass classification for KNN using SelectKBest =20 .....	46
Figure 4.25: Multiclass classification for Logistic Regression using SelectKBest =30	47
Figure 4.26: Multiclass classification for SVM using SelectKBest =30 .....	48

Figure 4.27: Multiclass classification for RF using SelectKBest =30 .....	49
Figure 4.28: Multiclass classification for KNN using SelectKBest =30 .....	50
Figure 4.29 Confusion matrix for CNN .....	55
Figure 4.30 Confusion matrix for LSTM .....	59

## List of Tables

Table 3.1 Types of attack .....	13
Table 4.1 Binary classification without feature selection method .....	22
Table 4.2 Binary classification with PCA .....	25
Table 4.3 Multiclass classification without feature selection method .....	28
Table 4.4 Multiclass classification using PCA .....	33
Table 4.5 Multiclass classification using SelectKBest feature =10 .....	38
Table 4.6 Multiclass classification using SelectKBest feature =20 .....	43
Table 4.7 Multiclass classification using SelectKBest feature =30 .....	47



# 1 INTRODUCTION

In today's world the Internet is an integral part of the day-to-day life and the security of communication networks is a critical aspect. A robust Intrusion Detection Systems (IDS) has become crucial with the rise of cyber threats. This project proposes an Intrusion detection system for Distributed Denial of Service (DDoS) attack. DDoS is a type of attack where the intruder floods the target device with a large amount of data packets resulting in unavailability of the system for the legitimate users [1]. This attack also leads to failure of the network. This attack in 2013 caused congestion on the Spamhaus network, resulting in a significant loss for the organisation. In 2018, the Github developer platform was overwhelmed with 1.3 TBps of traffic. This was the most powerful attack recorded to date [2]. In 2020, a 2.3 Tbsp. packets hit the Amazon Web services leading to the failure of the network [3].

In this project DDOS attack is detected using Machine learning and deep learning models. The dataset used to implement this system is CIC IDS2019. This dataset is developed by the Canadian Institute for Cybersecurity (CIC). This dataset covers the most recent attacks on the network [3]. This dataset 2.8 million network packets gathered over a period of seven days. This dataset has information of seven types of attacks on the network like brute force, Heartbleed, Botnet, DoS, DDoS, Web Attack and Infiltration.

The main aim of this project is to use existing dataset to perform attack against the network and to create AI models to act as an Intrusion detection system. This project also focuses on training the machine learning and deep learning models and evaluating/testing the model designed. The evaluation process analyses the accuracy, recall, f1-score and confusion matrix of each model. Finally, the models are compared to find the best model based on the performance measurements.

The objectives of the project are

- Selecting the most recent dataset and analysing the dataset to remove the missing and infinity value.
- Creation of Train and test datasets from existing dataset.

- Implementation of SVM, Logistic Regression, Random Forest and KNN models for the detection of attack and evaluate IDS performance of the model in recognising the attacks.
- Implement the machine learning model without feature selection and with feature selection methods.
- Implement CNN and LSTM deep learning models.
- Comparing the performance of all the models to find the best suitable model for the Intrusion Detection System.

## **2 BACKGROUND AND LITERATURE REVIEW**

This section is divided as introduction to intrusion detection system (section 2.1), the section 2.2 describes the related work and next section 2.3 explains the Machine learning and deep learning models. The section 2.4 explains the equation used for the performance evaluation of the model.

### **2.1 Introduction to intrusion detection system**

Much research has been made on intrusion detection in the network. The IDS is classified into three types: signature based, anomalies based and hybrid systems. In signature-based attacks are detected by comparing the current activity on the network with the database. In the second type, it detects by identifying the deviation in the behaviour of the current network from the normal activity stored in the system [4]. If the deviation between the two models exceeds the threshold value, then there is a malicious attack.

The signature-based approach has lower false alarm rate compared to the second method. However, later method has ability to detect unknown attacks, but it requires computational resources [5].

The DDoS is the subcategory of Denial of service (DoS) attack. In DoS, the attacker single internet connection to overflow the target device, whereas DDoS uses thousands/millions of connected networks to barrage a target device. The use of high-volume connection makes DDoS much harder to fight.

There are three main categories of DDoS attack namely Volume-based attacks, Protocol-based attacks, and Application-based attacks. [6]

- i. Volume-based attacks: In this type of attack the intruder floods the target system with large volume of traffic from various sources. This overloads the target device and results in failure of the system.
- ii. Protocol-based attacks: In this type of attack the network layer protocols are targeted by the intruder. It involves transmitting malformed packet and packets with the high rate of error. This overload the system leading to unavailability or crashing of the device.
- iii. Application-based attacks: Here the applications used by the target system is attacked. This involves transmitting malicious packets or requests that exploit vulnerabilities in the target application. This overload the target system.

In this project five main types of attacks are detected: MSSQL, NetBIOS, LDAP, Portmap and UDP. The MSSQL (Microsoft structured Query Language), in this type of attack the attacker poses as the Microsoft SQL Server and sends response messages to the target. This overflows the Microsoft SQL Server Resolution Protocol and spoofs the IP address of the MS SQL server. [7].

The Network Basic Input/output System (NetBIOS) is protocol present in Session layer of OSI model that enables file sharing, printer sharing and network services in the Windows operating system. Attackers can spoof the responses of this protocol to redirect network traffic to malicious system. [8]. The Lightweight Directory Access Protocol (LDAP) is used for authentication in emails, servers, and workstations. Therefore, this protocol attracts many hackers to gain unauthorised access to the target network [9]. The Portmap is a mechanism to which Remote Procedure Call (RPC) services register to allow for calls to be made to the Internet.

In the User Datagram Protocol (UDP) flood DDoS attack, the intruder targets a random port on the host device with a high volume of UDP packets. This type of attacks is established by spoofed IP address to ensure that the return packets do not reach the sender device. This conceals the identity of the sender network.

## **2.2 Related Work**

A very few works are conducted on the CIC-DDoS 2019 data set. In the paper [10], correlation method is used for the feature selection. This paper also uses PCA, ANOVA and Auto Encoder for reducing the dimensionality only on UDP flooding attack.

The authors of [11] has achieved up to 99% of accuracy using correlation method for feature selection. This paper uses Binary classification.

The work conducted by [12] has applied Logistic Regression, Decision Tree, Random Forest, Ada Boost, Gradient Boost, K Nearest Neighbors, and Naive Bayes classification algorithms. The authors did not use any feature selection method for the classification.

The paper [13] uses correlation method for feature selection and binary classification methods for detection.

The authors of [3] has achieved an accuracy ranging from 93- 97 % on different machine learning models using binary classification. The paper [14] has implemented five supervised machine learning algorithms in detecting efficiency in detecting 4 types of DDoS attacks (UDP, SYN, Portmap, and MSSQL) utilizing performance measures like

training time, accuracy, precision, and recall. By using wrapper selection method, 4 sets of features are determined.

The paper [15] uses three machine learning model for designing a DDoS system in Cloud computing environment. The system uses Random Forest, SVM and Logistic Regression model and accuracy of Random Forest is 97%, the SVM has 94% and Logistic Regression has 90% accuracy. The model uses binary classification.

The paper [16] used RF 20-features set and 30 feature sets, Light GB 20-features set, CatBoost 30-features set and CNN 20 and 30-features sets. It is found that the accuracy of random forest machine learning model with 20-features set is 99.99%.

The paper [17] uses PCA and SelectKBest feature selection method and found that accuracy of Random Forest model with PCA is greater than the SelectKbest Features in detection of cancer.

The paper [18] uses SelectKBest feature with  $K = 30$  for Random Forest model and achieved an accuracy of 90% in detection of Intrusion in Cyber-Physical System.

The paper [19] uses PCA with Random Forest detection on NSL-KDD dataset and obtained an accuracy of 97%.

The work on DDoS attack is further extended with the deep Learning method. The paper tests the CIC DDoS2019 datasets and obtained an accuracy of 99.99% for binary and 99.30% for multiclass using the CNN-based inception like model gave the best results among the proposed models. [20].

The paper [21] proposed BI-LSTM-GMM deep Learning model which achieves recall, precision, and accuracy up to 94%.

The paper [22] presents an approach to detect and classify the DDoS attacks using a hybrid model called AE-MLP, which combines an Autoencoder (AE) with a Multi-layer Perceptron Network (MLP). The model's performance is impressive, boasting high metrics like Precision (97.91%), Recall (98.48%), F1-score (98.18%), and Accuracy (98.34%), outperforming other similar methods. This makes the AE-MLP model a promising tool for enhancing DDoS defence mechanisms, particularly given the increasing prevalence of these attacks.

Most of the work on the DDoS CIC 2019 dataset uses Binary classification, in this project the Multiclass classification is used.

The datasets KDDCUP99 and NSL-KDD were generated decades ago and do not contain recent network traffic information. The most recent available datasets are CIC-DoS2017 and CIC DoS2019. Lots of work has already been done using CIC DDoS 2017 which has fewer features compared to CIC DDoS 2019. Many features are updated in the CIC DoS2019 which mitigates the shortcoming of the previous versions. This dataset has analysed new attacks conducted by TCP/UDP protocols. [23]

This project also implements the deep learning models like CNN and LSTM which are already implemented by the paper 20 and 21. These two models are implemented to compare the performance of the Machine learning models implemented in this project.

## **2.3 Machine Learning and Deep Learning Model**

In this project Random Forest classification, Support Vector Machines (SVM), Logistic Regression (LR) and KNN Machine learning models are used for performance evaluation. The two deep learning models CNN and LSTM are also implemented to compare the performance with machine learning models.

### **2.3.1 Random Forest (RF) Classifier**

The Random Forest classifier is used for classification, Regression. This classifier randomly selects the subset of the training set forming multiple decision trees to improve the prediction performance and robustness of the model [24]

The algorithm for random Forest classifier is as follows [25]

1. Randomly select “k” features out of total m features and build the random forest where  $k < m$ .
2. calculate the node d using the best split point.
3. Split the node into leaf nodes using the best split.
4. Repeat 1 to 3 steps until number of nodes has been reached.

Build forest by repeating steps 1 to 4 for n number of times to create n number of trees.

For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e. the most frequent categorical variable—will yield the predicted class [26].

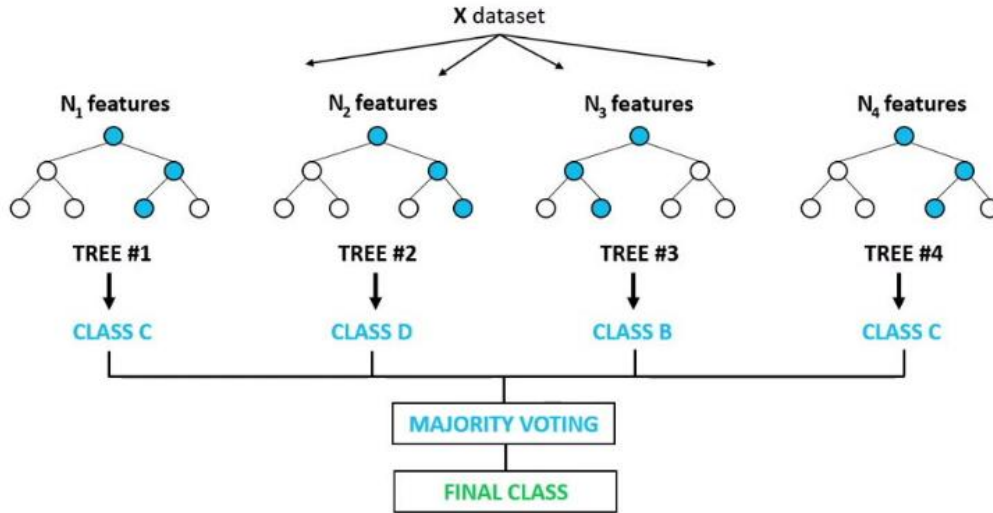


Figure 2.1: Random forest classifier [25]

### 2.3.2 Support Vector Machines (SVM)

Support Vector Machines (SVM) are a powerful supervised learning algorithm used for both classification and regression tasks. SVMs are used for finding the optimal hyperplane (decision boundary) that best separates data points into different classes (in case of classification) or fits the data with minimal error (in case of regression). [27].

In order to make it simple to classify new information in the future, the SVM method seeks to identify the optimal line or decision boundary that can divide n-dimensional space into classes. This optimal decision boundary is referred as a hyperplane. SVM selects the extreme vectors and points to aid in the creation of the hyperplane. The algorithm is referred regarded as a Support Vector Machine since these extreme situations are known as support vectors.

### 2.3.3 Logistic Regression

Logistic Regression is a classification algorithm used when the dependent (target) variables are categorical in nature- meaning the data can be grouped into discrete outputs  $\{0, 1, \dots, k - 1\}$ . Since we are dealing with categorical variables, logistical models must be used to map probabilities to predicted labels of the data. Examples of Logistic

Regression classification include spam detection in email, cancer detection, and credit fraud detection.

### 2.3.4 KNN Model

K-NN is the supervised machine learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories [20]. The K-NN method classifies a new data point based on similarity after storing all the relevant data. This indicates that the K-NN algorithm can quickly classify newly discovered data into a well-suited category.

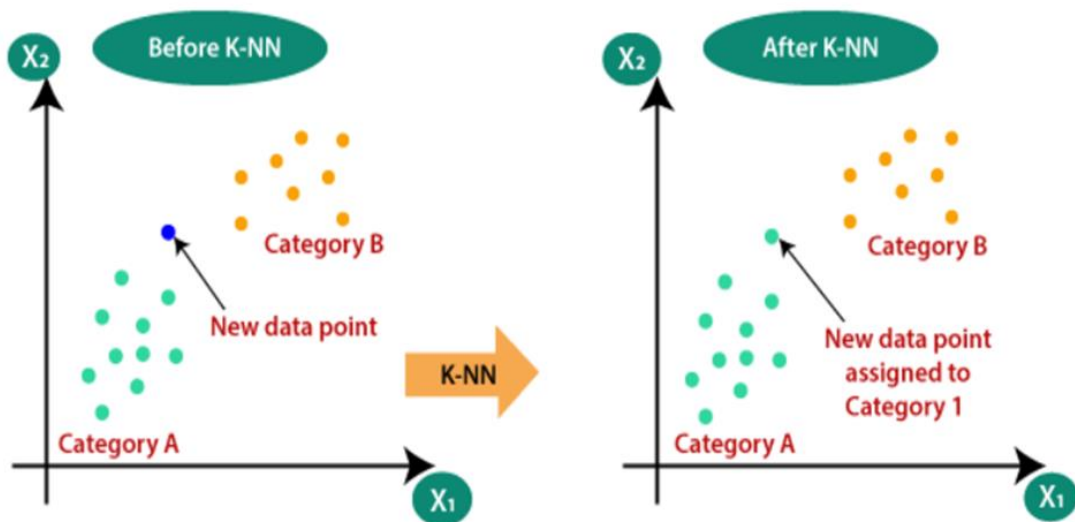


Figure 2.2: K-NN Example [28]

The new data is placed in the classification plane. The Euclidean distance between the new data and the neighbour is calculated. The nearest neighbouring point is considered as the category to which the data belongs.

### 2.3.5 Convolutional Neural Network (CNN) Model

Convolutional networks use convolutional technique instead of matrix multiplication, which combines two functions to show how one changes the shape of the other. The CNN has following layers. The input layer, convolutional layer, activation layer, Flatten layer, Dense layer, Dropout layer, Output Layer.



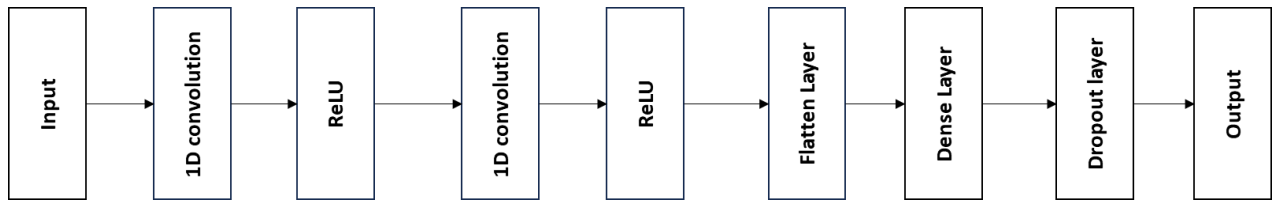


Figure 2.3: Structure of CNN

**The input layer:** This layer takes input for the model. In this project there are 87 features, therefore the input is 87 features.

**The convolutional layer:** In this work, the 1D convolution operation is applied to the input data. Since the input is text data, 1D convolution is used.

This layer has 128 filters used in the training model. The output of this layer is 128 channels. The kernel size is 6, refers to the convolution window that moves over the input data.

**The activation function:** The activation function used is Rectified Linear Unit (ReLU). This function introduces non-linearity in the model allowing the model to learn the complex patterns. It is a linear function that outputs the input directly if it's positive, and zero if it's negative.

**The Flatten Layer:** In a CNN, the data retains its multi-dimensional structure beyond the convolutional and pooling layers. The Flatten layer is used to connect this data to fully connected (Dense) layers, which require a 1D input. While reshaping the data into a flat array that may be sent to the Dense layers for additional processing, it maintains the information from earlier layers.

**The Dense Layer:** Each neuron in a Dense layer is connected to every neuron in the previous layer. These layers perform the final decision-making based on the features extracted by earlier layers (like convolutional layers). The Dense layer weights are learned during training, and it applies an activation function like ReLU or sigmoid to produce the final output.

**Dropout Layer:** This layer reduces the overfitting of the model. This helps the model avoid becoming too reliant on specific neurons, forcing it to learn more robust features by not depending too heavily on any single neuron.

**The output layer:** The Softmax output would provide probabilities for each of the five classes, with the highest probability indicating the predicted class.

### 2.3.6 Long Short-Term Memory (LSTM) Model

The Long short-term phrase suggests that the network has a short-term memory for making decisions about recent events, but it also has a long-term memory for making decisions. At each stage the new input updates the data from the recurrent connection outputs. The output at each stage is the weighted sum of the old and new input, with the weights based on the relative relevance or content of the two sets of data. The old data would remain unchanged, or the network would entirely ignore the new input if the new input were multiplied by zero. The network is said to have forgotten the old information if the old information is multiplied by a zero weight, making the new information totally the new input [29].

They are particularly useful for tasks involving time series, natural language processing, and any scenario where data points depend on previous points in the sequence.

#### **The Key components of LSTM layers:**

**Memory Cells:** This retains the important information across the sequences of input and ignores irrelevant information.

**Gates:** LSTMs control the flow of information using three primary gates:

**Forget Gate:** Decides which parts of the previous information to be discarded. This gate forgets the information that is no longer relevant.

**Input Gate:** Selects new data to be stored in the memory cell.

**Output Gate:** Controls the output based on the current memory and input. It decides what information from the memory cell will be used in the current step. [30]

## 2.4 Equations

The performance of the system can be evaluated using Accuracy, Precision, Recall and F1 measures. Accuracy is the measure of the correct prediction made by the model across all the datasets.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad (1)$$

The ratio of correctly predicted positive instances to the total predicted positives. It indicates how many of the predicted positive instances were positive.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

Recall The ratio of correctly predicted positive instances to all actual positives. It represents many of the actual positive instances were captured by the model.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3)$$

The harmonic means of precision and recall. It balances the trade-off between precision and recall, providing a single metric that considers both false positives and false negatives.

$$\text{F1 Score} = \frac{TP2*Precision*Recall}{Precision + Recall} \quad (4)$$

TP: True Positive when model correctly identifies the attack.

TN: True Negative when model identifies normal activity.

FP: False Positive when model identifies the normal activity as attack.

FN: False Negative when model identifies the attack as normal activity.

### **3 METHODOLOGY**

This section goes into the project's design, highlighting the structured approach used to achieve the objectives. It discusses data preparation strategies for cleaning and preparing datasets so that they are suitable for analysis. Feature extraction approaches are explored, including how relevant features were chosen to improve model performance. The machine learning models used are explained in terms of how they apply to the dataset. This section provides a full explanation of the project's techniques and workflows.

#### **3.1 Design of the project**

The project is to design an intrusion detection system to detect any malicious or abnormal activities in the network. The CIC DDoS 2019 dataset is used which is the most recent dataset available. There are 88 features (listed in the Appendix B) and 6087134 samples. The chosen dataset has five attacks UDP, LDAP, NetBIOS, Portmap, MSSQL and Benign type.

The project is divided in various stages, the first stage is to import the relevant libraries and pre-processing of the data. Later, the label encoding is performed to convert the categorical entries into numerical values. Since there are different instances for each type of the attacks the data balancing is done. The balanced data is split into training and test set. The training set trains the machine learning model, and the test set evaluates the model. In this project, 5000 samples of each type of attack is chosen for the training and testing the data (since the size of the dataset is very large only 5000 samples of each type of attack is randomly selected). Two types of classification used is binary and multiclass classifications. There are 5 types of attack detected.

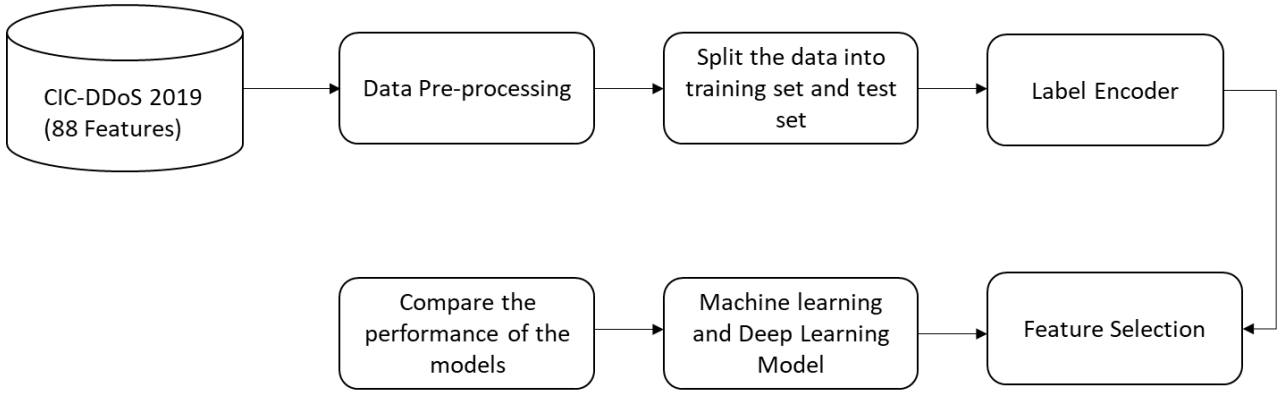


Figure 3.1: Different stages of the project

Table 3.1 Types of attack

SL. No	Attack	No. of instance
1	UDP flood	3754680
2	LDAP	1905191
3	NetBIOS	202919
4	Portmap	186960
5	MSSQL	24392
6	BENIGN	12992

### 3.2 Data Pre-processing

The downloaded CIC DDoS 2019 dataset includes 88 features and 6087134 instances. The data pre-processing is an important in cleaning the data. In this step data cleaning is done to identify the missing and duplicate entries in the dataset. The duplicate data is removed from the data set. The missing data is replaced by the median value of that particular column. The infinity value in the dataset is replaced with the Not a Number (Nan) values. In the chosen dataset, the missing values and infinite values are found in two columns 'Flow Bytes/s' and 'Flow Packets/s'. The number of missing values is 140959 which is 2.32% of the total entries of the dataset. This value is replaced with the median value of the 'Flow Bytes/s' and 'Flow Packets/s' respectively.

### 3.3 Splitting of the dataset

The dataset is split into training set and the test set. 75% of each type of the attack is randomly chosen to be training set and remaining 25% is used to test the performance of the model.

```
train_class, test_class = train_test_split(df_class, test_size=0.25, random_state=0, stratify=df_class['Attack Type']).
```

The *train\_test\_split* is a function provided by the *sklearn.model\_selection* module in the scikit-learn library. This method drops overfitting of the model by evaluating the model on the unseen test dataset.

The *df\_class* defines the x and y values of the dataset. The x value is the features of the dataset, and the y is the label in the dataset. The stratify is particularly useful in classification tasks to maintain the same class distribution in both training and testing sets.

### 3.4 Label Encoder.

Label encoder converts the categorical or string data into numerical format. Most of the machine learning model works on the numerical data, therefore it is especially important to convert the non-numerical data into numeric data. Therefore, converting categorical variables into numbers allows the model to process the data. In this project the label (name of the attack) is encoded into numerical value.

### 3.5 Feature selection

Feature selection is the important process in training the machine learning Model. Feature selection method helps in improving the performance of the system. The performance of the model can be improved by selecting the best subset of the features available in the dataset. The best feature selection method speeds up the computational efficiency of the process. Increase in the feature size also leads to overfitting in machine learning model. In this project the without feature selection, Principal Component Analysis (PCA) and SelectKBest feature selection methods are implemented.

In without feature selection method all the features of the dataset are considered during training and testing the model.

Principal Components Analysis (IPCA) is a linear dimensionality reduction method for large datasets. In this project, the training and test datasets are processed in batches to fit the PCA model incrementally. IPCA processes the datasets in smaller batches which makes it suitable for larger datasets to be loaded into memory at once. This method is used for visualization, pattern recognition regression and time series prediction [31].

SelectKBest Feature selection provided by Scikit- Learn library, it selects the prominent features on the K highest score. It has two parameter score function and K. Score function evaluates the features.

### **3.6 Classification**

There are two types of classification binary and multiclass classification. In binary classification, the machine learning model predicts whether there is any attack or not. It predicts two classes.

The multiclass classification predicts more than two classes. In this project 6 classes of the attacks are detected namely UDP, LDAP, NetBIOS, Portmap, MSSQL and Benign type.

### **3.7 Cross validation**

Cross validation is the process in which the training data is split into K-1 set and validated using the remaining data. The Figure 3.3 shows the cross-validation process. The entire data is split into training data and test data. The training data is further split into 5 sets. The four sets are used as training data and the remaining 1 set is used for validation. The result obtained at each subset is measure the model's performance on training data. The mean cross validation indicates overall accuracy of the model.

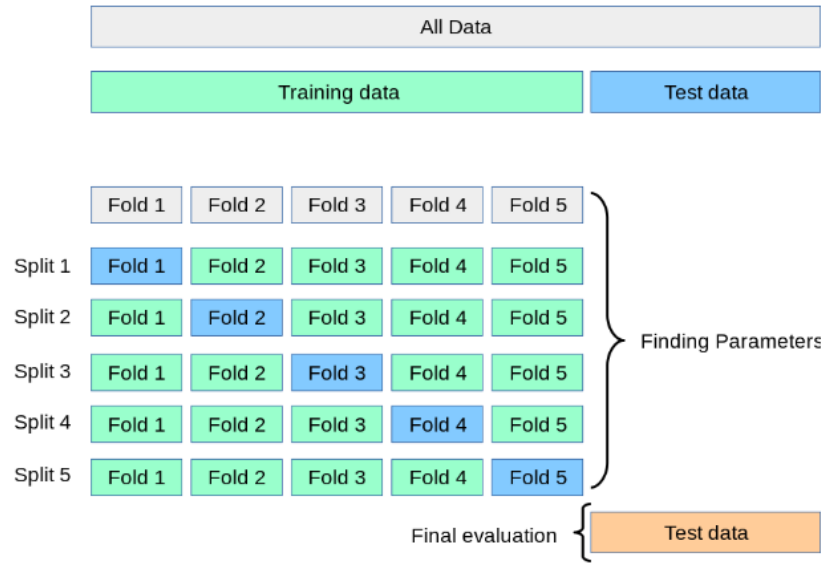


Figure 3.2: Cross validation using 5-folds [27].

### 3.8 Machine Learning Model

In this project four machine learning models are implemented namely Random Forest, SVM, Logistic Regression and KNN. Each Machine learning Model is applied using without feature section method, SelectKBest feature selection and Principal Component Analysis Method. The detail structure of implementation Machine learning Model is shown in the figure 3.4.

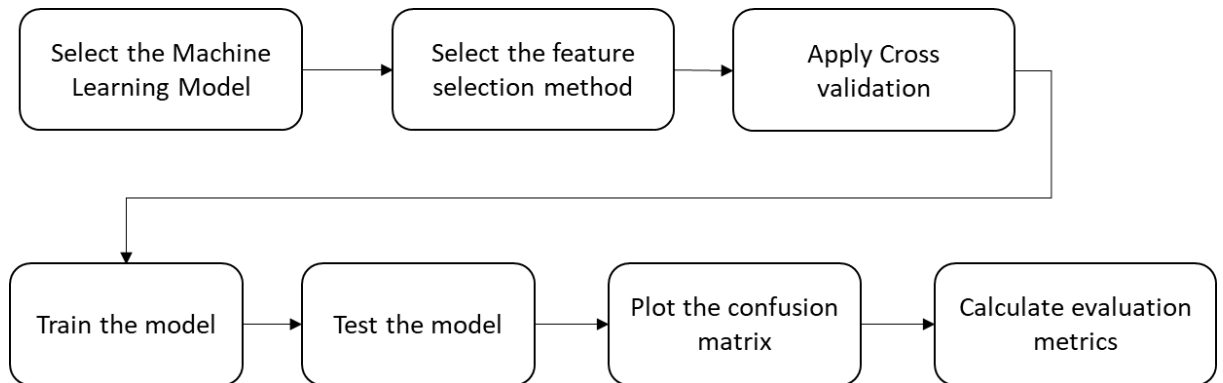


Figure 3.3: Flow diagram of Machine learning

#### 3.8.1 Logistic Regression (LR)

This is the scikit-learn class used to create a logistic regression model for classification tasks. The `max_iter=10000`, specifies the **maximum number of iterations** for the optimization algorithm (the solver) to converge. This process is used to find the best model parameters. Setting `max_iter=10000` means the algorithm will run for at most



10,000 iterations before stopping. This is useful when the default number of iterations (100 or 200) isn't enough for the model to converge, especially with large or complex datasets.

### 3.8.2 Support Vector Machines (SVM)

The SVM model uses the Support Vector Classifier (SVC) with a radial basis function (RBF) kernel. SVM works by finding a hyperplane that best separates different classes in the dataset.

**Radial Basis Function (RBF)** is a popular kernel in SVM. It maps the data into a higher-dimensional space to make it easier to find a linear separation in that transformed space. The RBF kernel is especially useful when the data is not linearly separable in its original space.

### 3.8.3 Random Forest (RF) Model

The Random Forest Model uses `n_estimators = 100`. The number of decision trees that are constructed independently and then combined is indicated by the Random Forest algorithm's `n_estimators` parameter. The final prediction is usually the mean or mode of the predictions of each individual decision tree in the Random Forest. Each decision tree in the Random Forest is trained on a random subset of the training data (sampling with replacement).

Maximum Depth of Trees is defined as the number of nodes from the root to the farthest leaf node. In model Maximum depth is 6. Overall the performance of the Random forest Model is good. It has highest accuracy

### 3.8.4 KNN model

The `KNeighborsClassifier` is the scikit-learn class used to create a KNN model for classification tasks. The `n_neighbors=5`, the algorithm looks at the **5 nearest points** (in terms of distance) from the training dataset. The class that occurs the most frequently among those 5 neighbors is chosen as the predicted class for the new data point.

## 3.9 Deep Learning Models

The two Deep Learning Models are implemented namely Convolutional Neural Network (CNN), Long short-term memory (LSTM).

### 3.9.1 CNN Model

The **Adam** optimizer is used to compile the model. The learning rate is set to 0.001, which adapts the rate during training for faster and more stable convergence. Since multi-class classification is done, the **categorical\_crossentropy** loss function is used measuring the difference between predicted and actual class probabilities. The model's performance is evaluated using accuracy, precision, Recall and F1 score.

The result of initial component setup for the CNN is shown in the figure below.

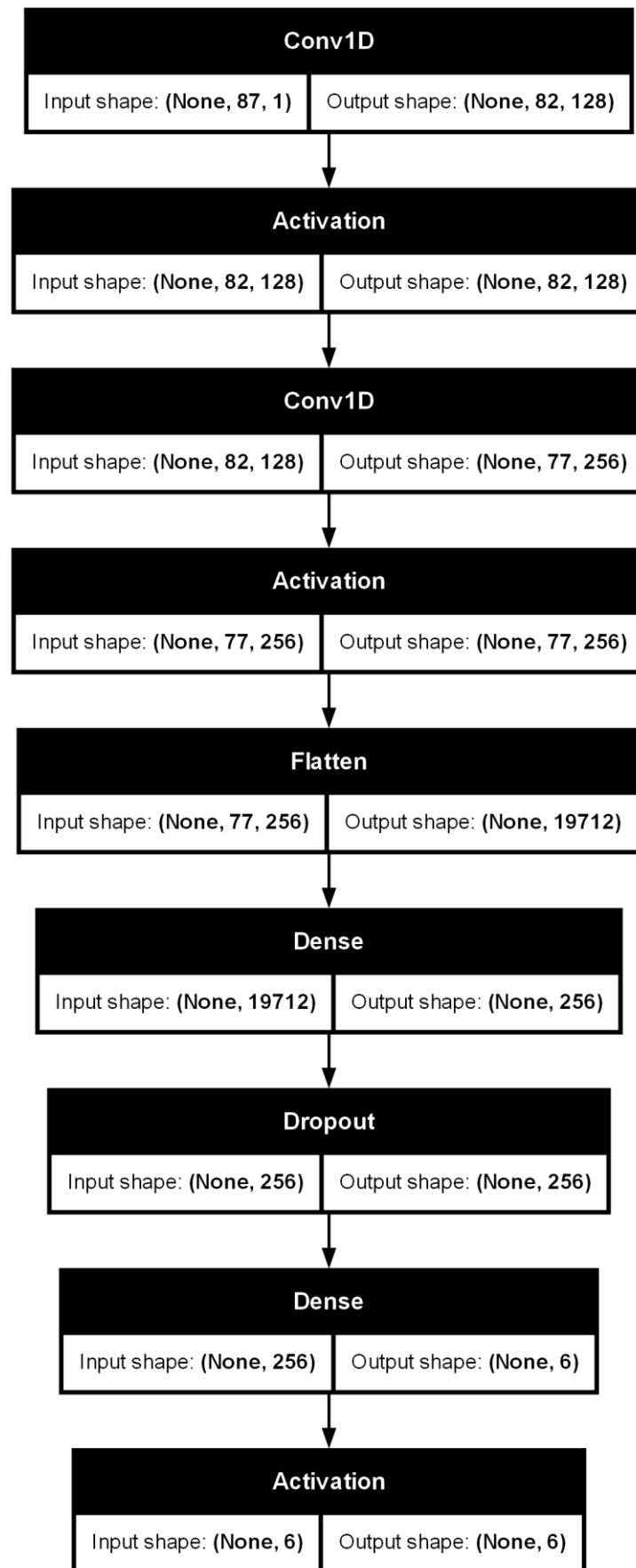


Figure 3.4: Layers of CNN model

The first layer is **Conv1D** (Convolutional Layer 1D), in this the input is time series with 88 features. It takes one feature at a time applies convolution function on it.

The output has 128 filters after applying convolution, and the time dimension is reduced from 87 to 82 due to the filter size and stride.

The next layer is **Activation Layer** which gets input from previous layer Conv1D layer. The activation function is applied to introduce non-linearity, but the shape remains the same.

The second **Conv1D Layer** is applied has 256 filters, reducing the time dimension from 82 to 77.

The second **Activation Layer** retains the shape of the data.

The **Flatten Layer** flattens the output of previous layer into a 1D vector for the fully connected layers.

The **First Dense (Fully Connected) Layer** has 256 neurons, reducing the flattened feature space to 256 dimensions.

The **Dropout Layer** is used to prevent overfitting by randomly dropping a fraction of the neurons during training. It doesn't affect the output shape.

The **Second Dense Layer**, is the final dense layer outputs 6 values, corresponding to the 6 possible classes in this classification task.

The **Final Activation Layer**, softmax function, which converts the 6 output values into probabilities, one for each class.

### 3.9.2 LSTM

LSTM Layer 1 takes 87 features and processes it sequentially one feature at a time. The output of this step is 128 units for each step.

This layer maintains the shape of the previous stage, but randomly drops some connections to prevent overfitting.

The LSTM Layer 2 receives the output of the dropout layer and reduces the output to 64 units. The next layer is Dropout layer 2 this layer takes the 64 units from the previous LSTM layer and drops few connections to avoid overfitting but retains the shape to 64 units.

The next layer **Dense (Fully Connected) Layer**, produces 6 outputs, corresponding to the number of classes in the classification problem (multi-class classification).

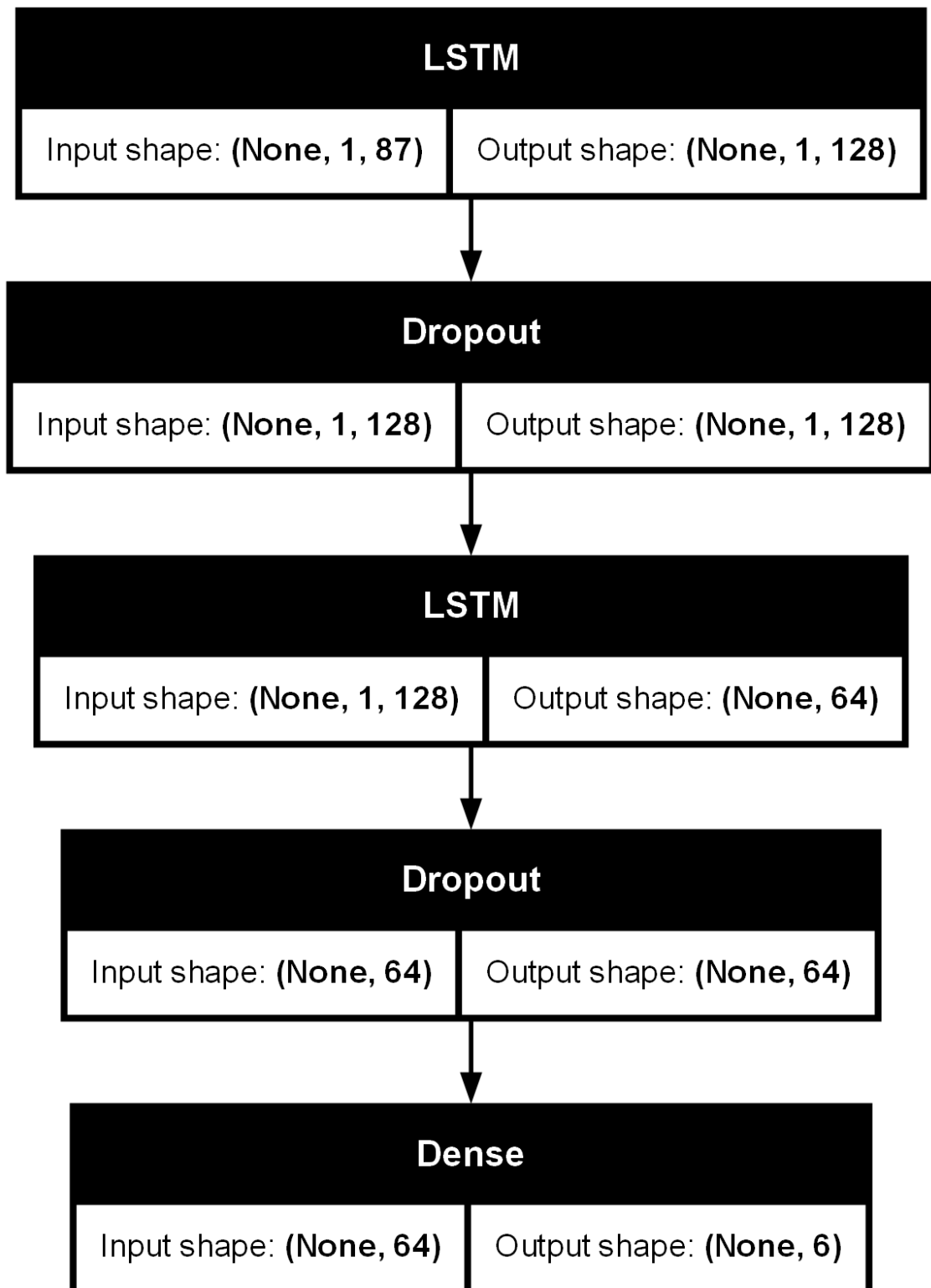


Figure 3.5: Layers of LSTM model

## 4 RESULTS AND DISCUSSION

In this section the output of Machine Learning models namely Logistic Regression, Random Forest, SVM and KNN and Deep Learning models like CNN and LSTM are discussed. These four models are implemented using without feature selection, PCA and SeletKBest feature selection method. Both the multiclass and binary classification is discussed in this section. The dataset used for the detection is CIC DDoS 2019.

### 4.1 Binary Classification

Binary Classification is applied on all four-machine learning model without feature selection method and PCA method. It can be observed that the binary classification gives an accuracy of 99.99% for all the models.

#### 4.1.1 Without Feature selection

The binary classification results of all four Machine Learning model without any feature selection method. The binary classification has highest accuracy.

Table 4.1 Binary classification without feature selection method

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	1.000	1.000	1.000	1.000
Support Vector Machines	1.000	1.000	1.000	1.000
Random Forest	1.000	1.000	1.000	1.000
K-Nearest Neighbor	0.999	0.999	1.000	0.999

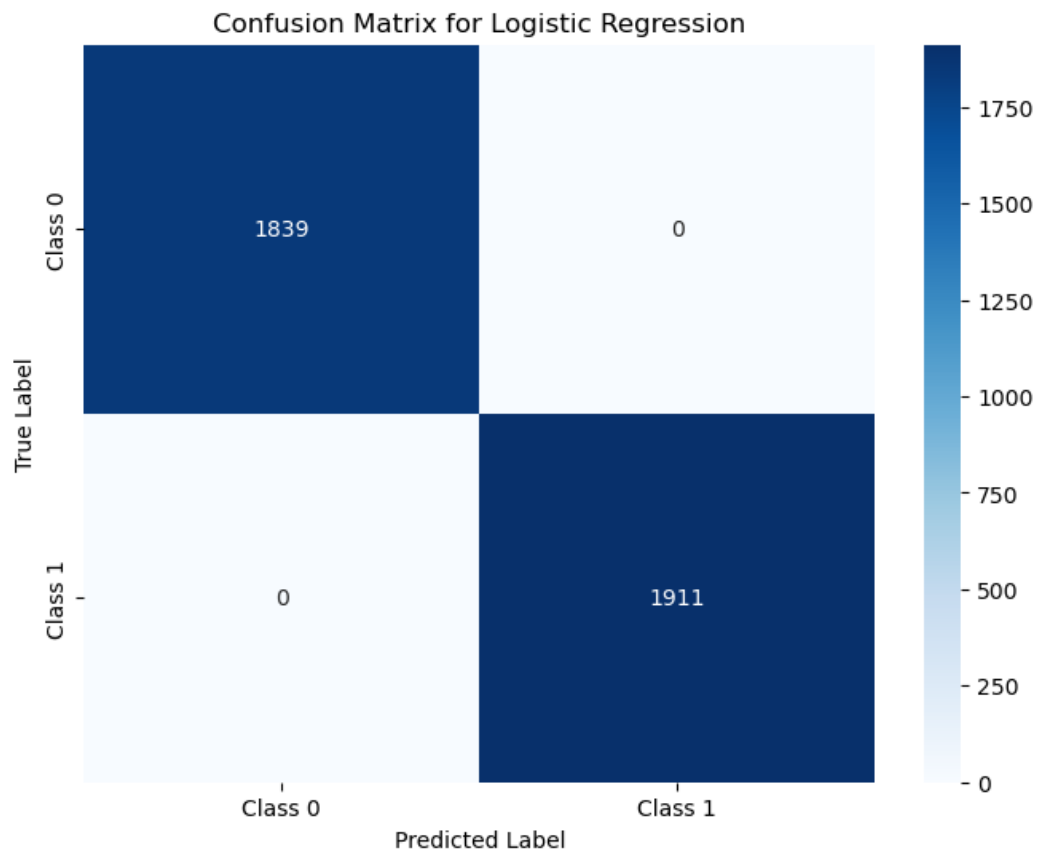


Figure 4.1: Binary classification of Logistic Regression without Feature selection

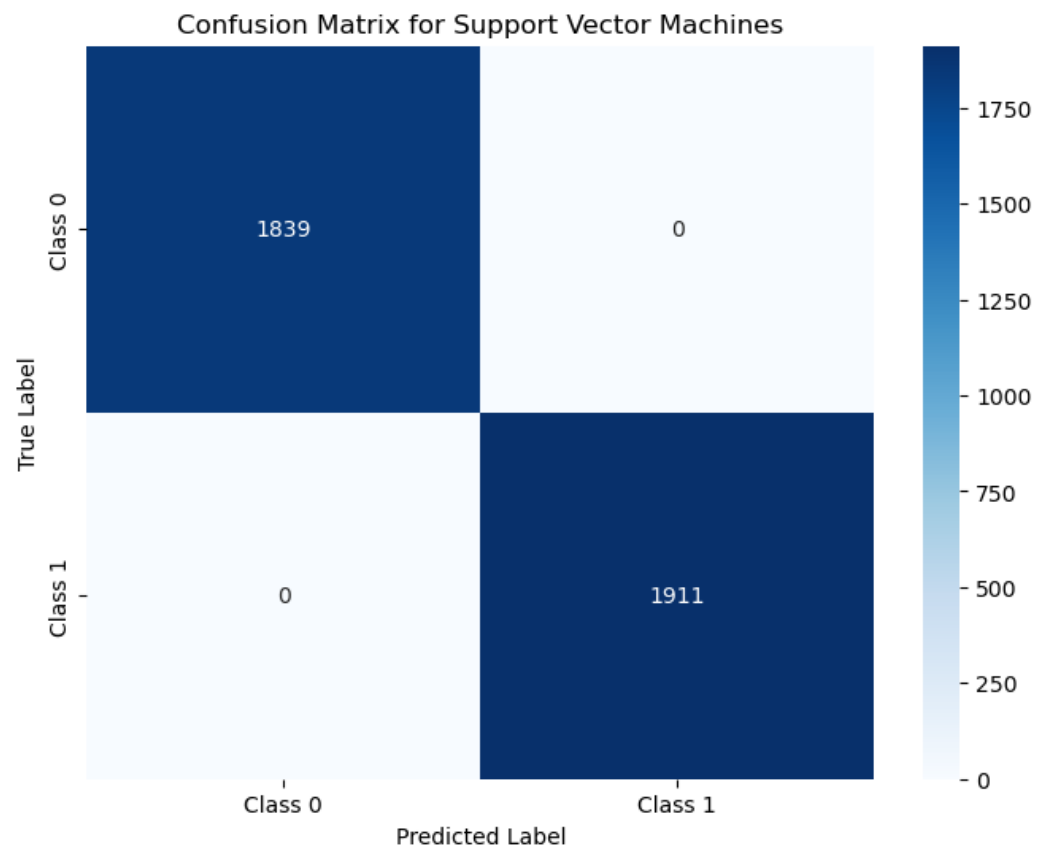


Figure 4.2: Binary classification of SVM without Feature selection

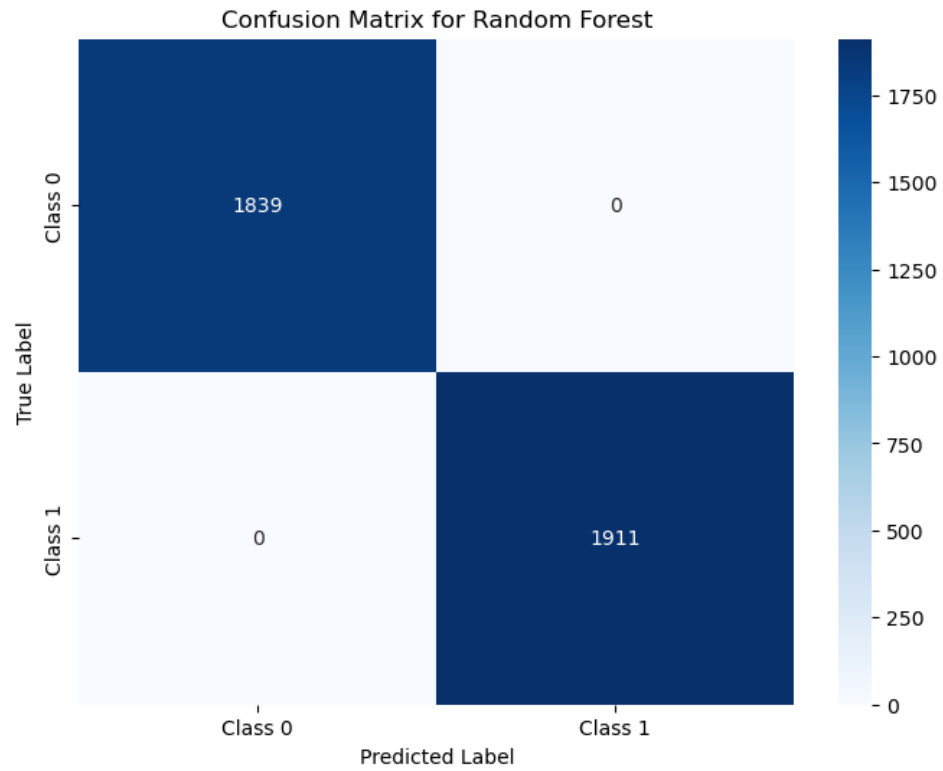


Figure 4.3: Binary classification of Random Forest without Feature selection

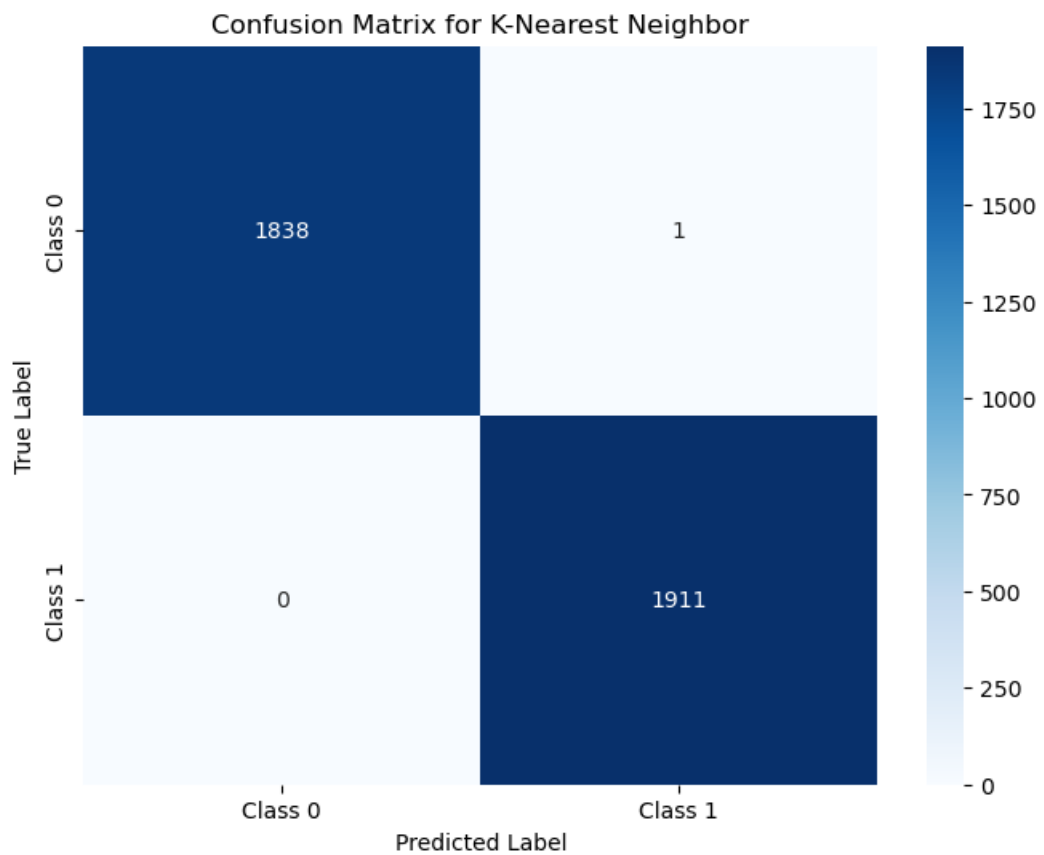


Figure 4.4: Binary classification of KNN without Feature selection



### 4.1.2 Principal Component Analysis (PCA)

Binary classification with PCA also has best accuracy. All the models has accuracy more than 99%.

Table 4.2 Binary classification with PCA

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.999	0.998	1.00	0.999
Support Vector Machines	0.999	0.998	1.000	0.999
Random Forest	1.000	1.000	1.000	1.000
K-Nearest Neighbor	1.000	1.000	1.000	1.00

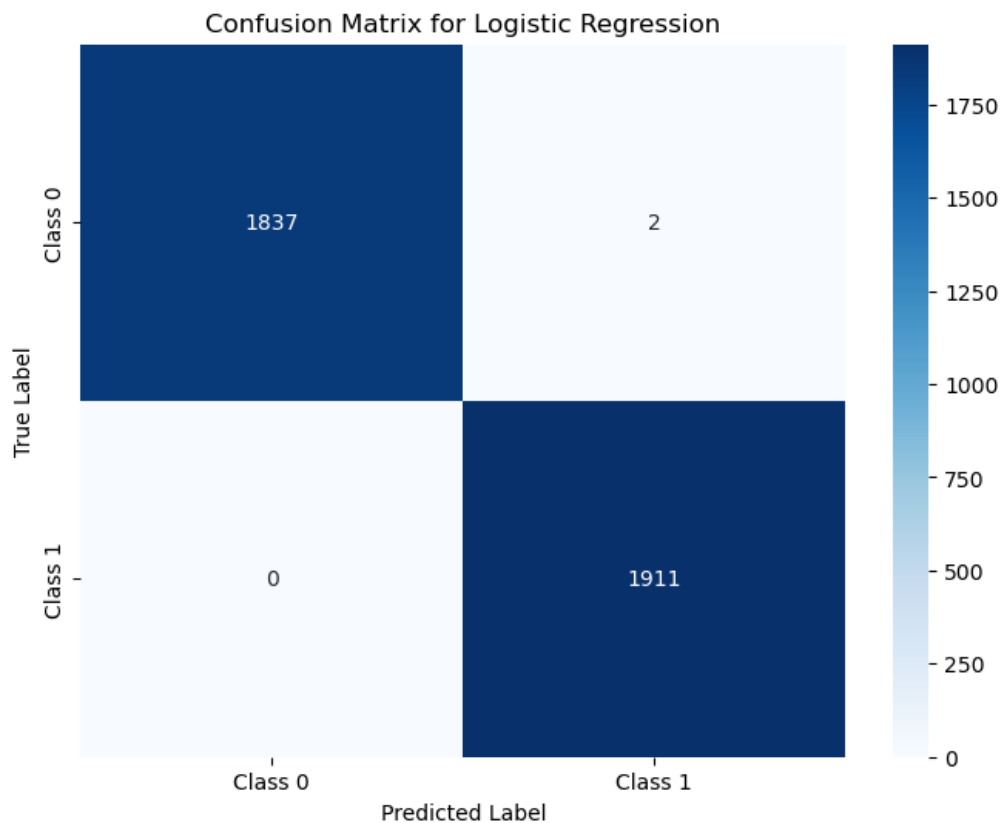


Figure 4.5: Binary classification of Logistic Regression using PCA

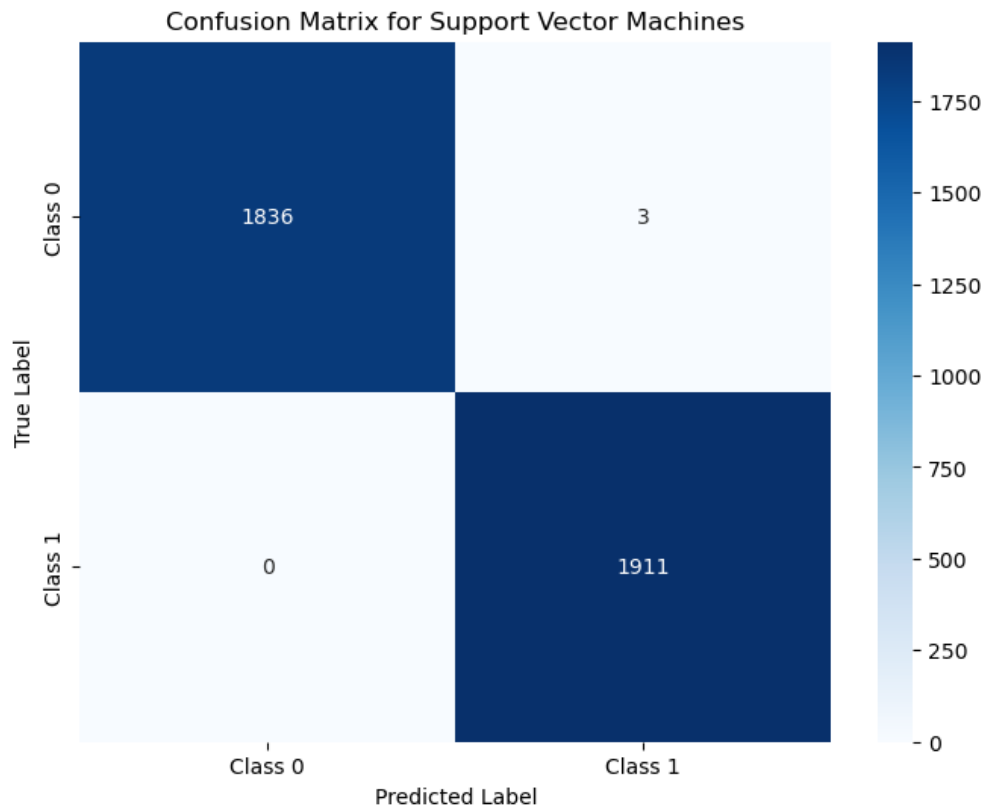


Figure 4.6: Binary classification of SVM using PCA

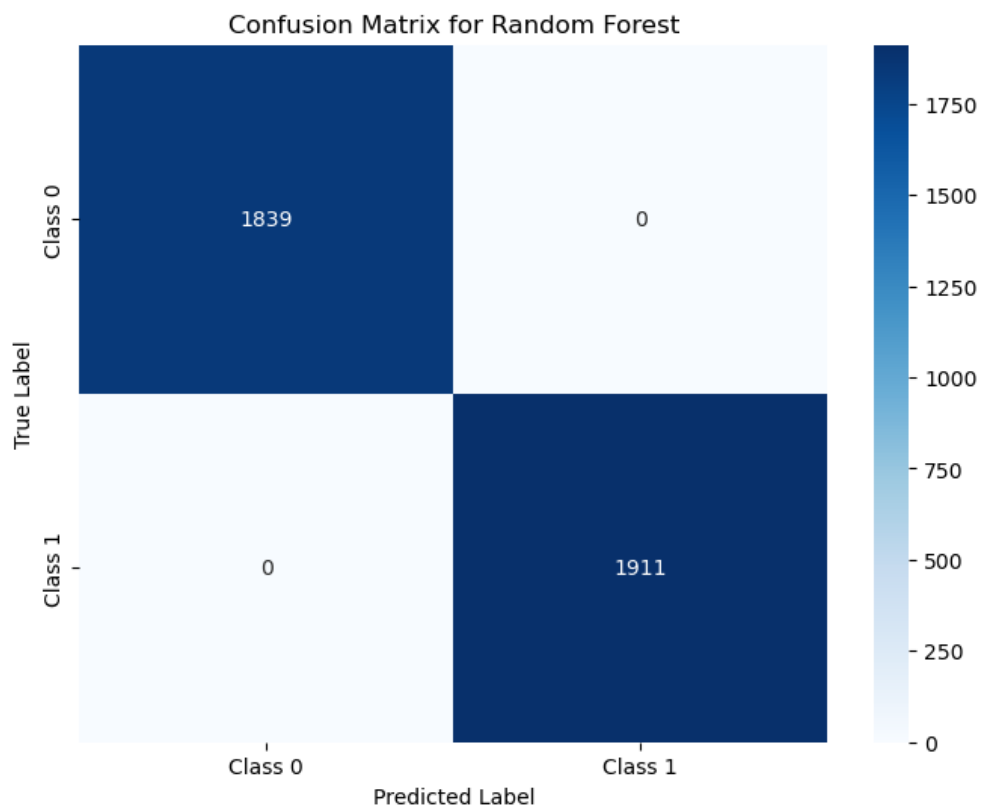


Figure 4.7: Binary classification of Random Forest using PCA

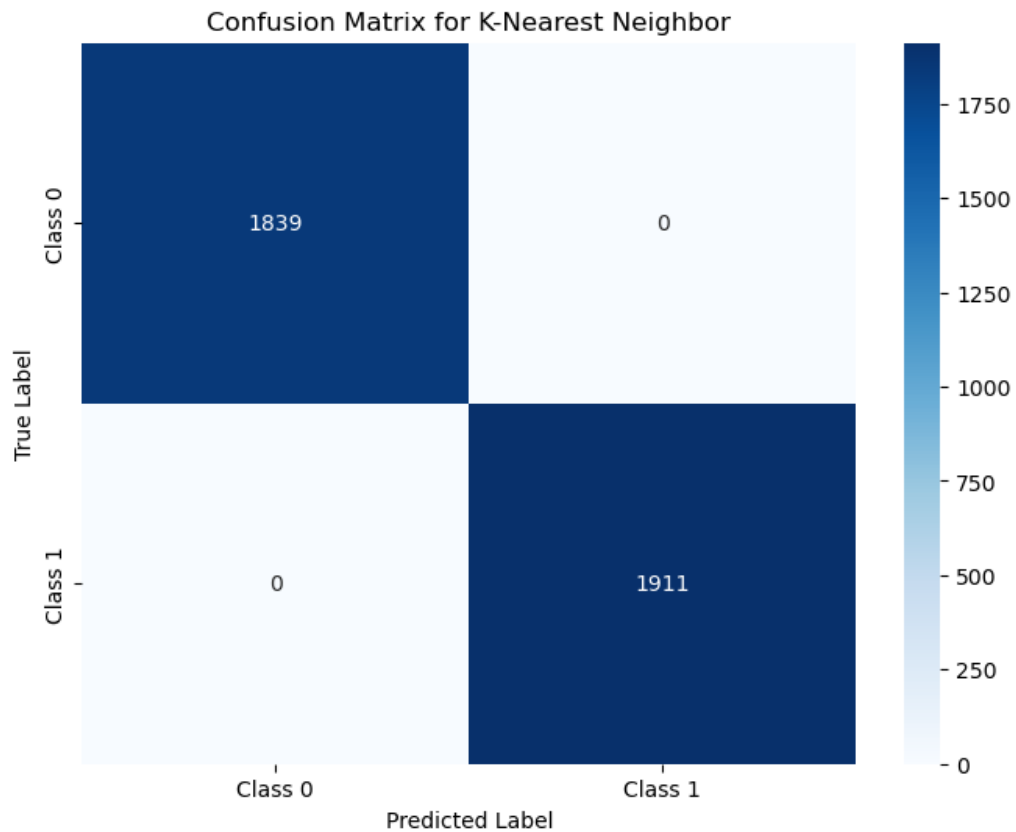


Figure 4.8: Binary classification of KNN using PCA

## 4.2 Multiclass classification

Since there are more than one class of the attacks, multiclass classification is performed on the models. The multiclassification is conducted without feature selection method, PCA method and the selectKbest method. Multiclass classification gives better understanding about the detection of each type of attack.

### 4.2.1 Without Feature selection

In without feature selection method all the 88 features are considered for training and testing the model. The table 4.2 shows the accuracy, precision, Recall and F1 score of each model. The table also has the time consumed by each model for training and testing.

It can be observed from the table that the Random Forest has highest accuracy of 100%, followed by the KNN model with 91% of accuracy. The accuracy of Logistic Regression

is 78% and the SVM model has an accuracy of 16%. With this information it can be concluded that the SVM model completely fails without feature selection method.

Table 4.3 Multiclass classification without feature selection method

<b>Model</b>	<b>Mean Cross validation</b>	<b>Accura cy</b>	<b>Precisio n</b>	<b>Recall</b>	<b>F1 Score</b>	<b>Trainin g time (s)</b>	<b>Testing time (s)</b>
<b>Logistic Regression</b>	0.78	0.78	0.79	0.78	0.78	605.03	0.08
<b>Support Vector Machines</b>	0.17	0.16	0.19	0.16	0.05	3676.97	67.47
<b>Random Forest</b>	1.00	1.00	1.00	1.00	1.00	12.81	0.20
<b>K-Nearest Neighbor</b>	0.90	0.91	0.91	0.91	0.91	2.99	1.66

#### 4.2.1.1 Confusion Matrix for without feature selection method

A summary of the machine learning model's performance on a set of test data can be obtained by a confusion matrix. It is a way to demonstrate the number of instances, depending on the model's predictions, are accurate and inaccurate. It is frequently used to assess how well categorization models which attempt to assign a categorical label to each instance of input perform.

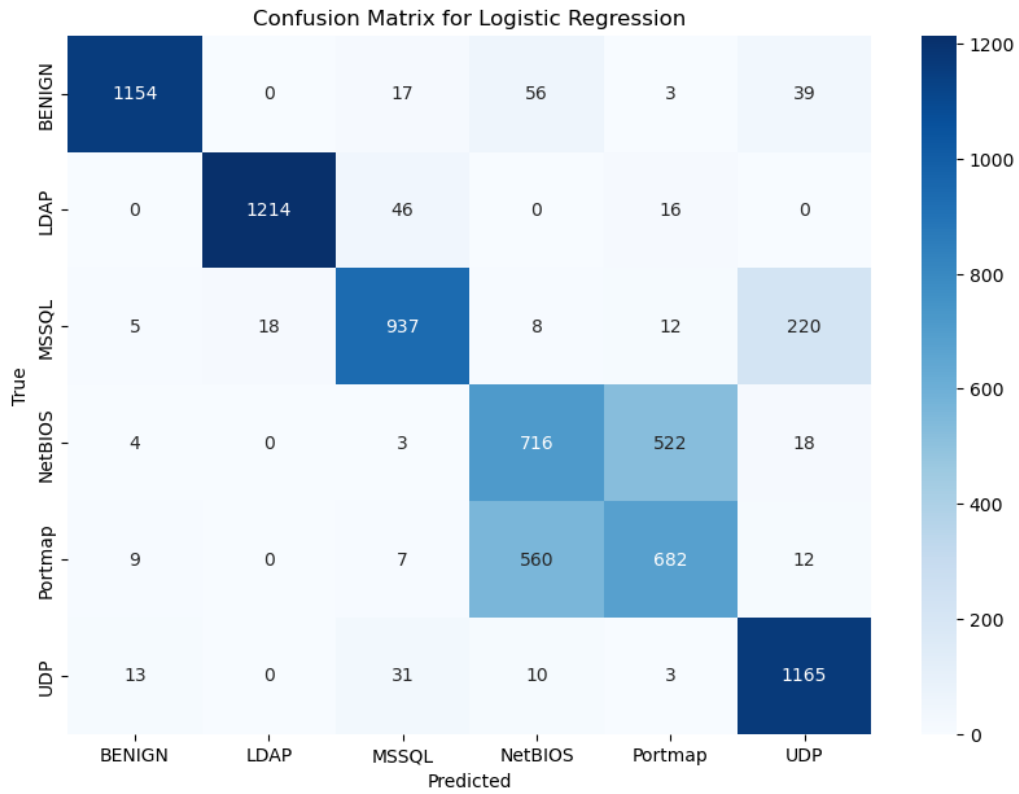


Figure 4.9: Multiclass classification of Logistic Regression without feature selection

The confusion matrix for Logistic Regression (LR) shows the predicted and actual label of the attack. It can be observed that the LR has an accuracy of 78% and precision of prediction is 79%. The training time of the model is 605.03 seconds (appr. 10 minutes) And the testing time is 0.08 seconds. The model performs very well in detecting the Benign, LDAP and UDP Flood attack. There is a slight miscalculation in detecting the MSSQL attack and the accuracy drops considerably while predicting the NetBIOS and Portmap attacks.

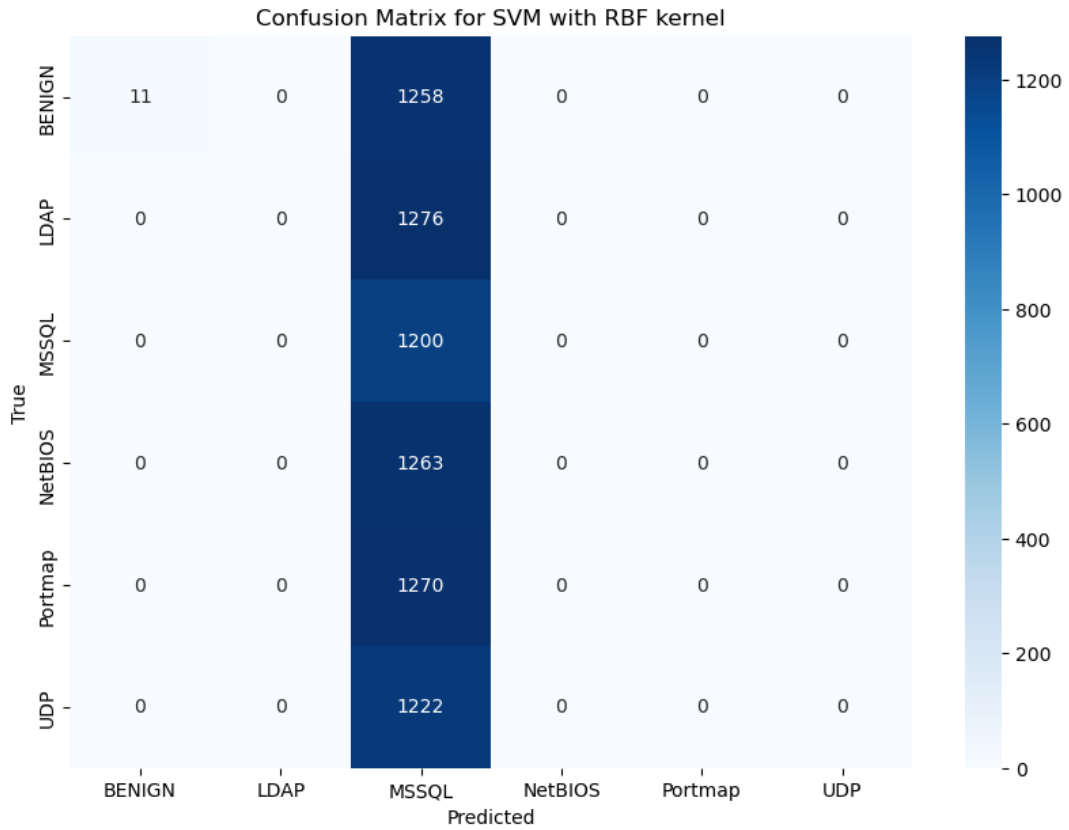


Figure 4.10: Multiclass classification of SVM without feature selection

The confusion matrix of SVM is shown in the Figure 4.10. The model completely fails for the predicting the attack without feature selection method. The accuracy of the model is 16% which indicates that the model cannot detect any of the attack when all the features are used. The performance of Support Vector Machines (SVM) is highly dependent on the feature selection and dimensionality reduction techniques used. Without feature selection, SVM may struggle to handle high-dimensional datasets efficiently, leading to poor accuracy due to the inclusion of irrelevant or noisy features.

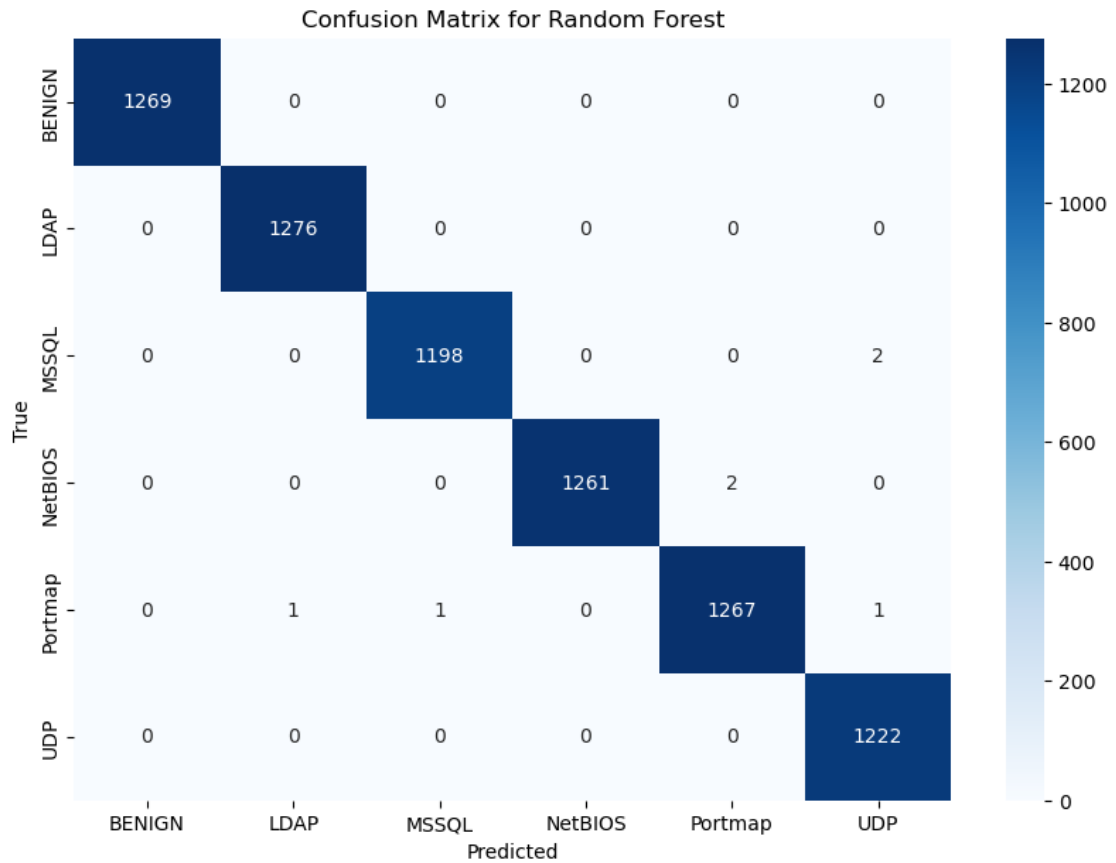


Figure 4.11: Multiclass classification of Random Forest without feature selection

The confusion matrix for Random Forest (RF) is shown. It can be observed that the RF has an accuracy of 100% and precision of prediction is 100%. The training time of the model is 12.81sec and the testing time is 0.20 sec.

The model detects the Benign, LDAP and UDP flood attack without any mis-prediction. There is slight negligible amount of mis prediction for other types of attack. This shows that the Random Forest model performs excellently in terms of accuracy and time consumption.

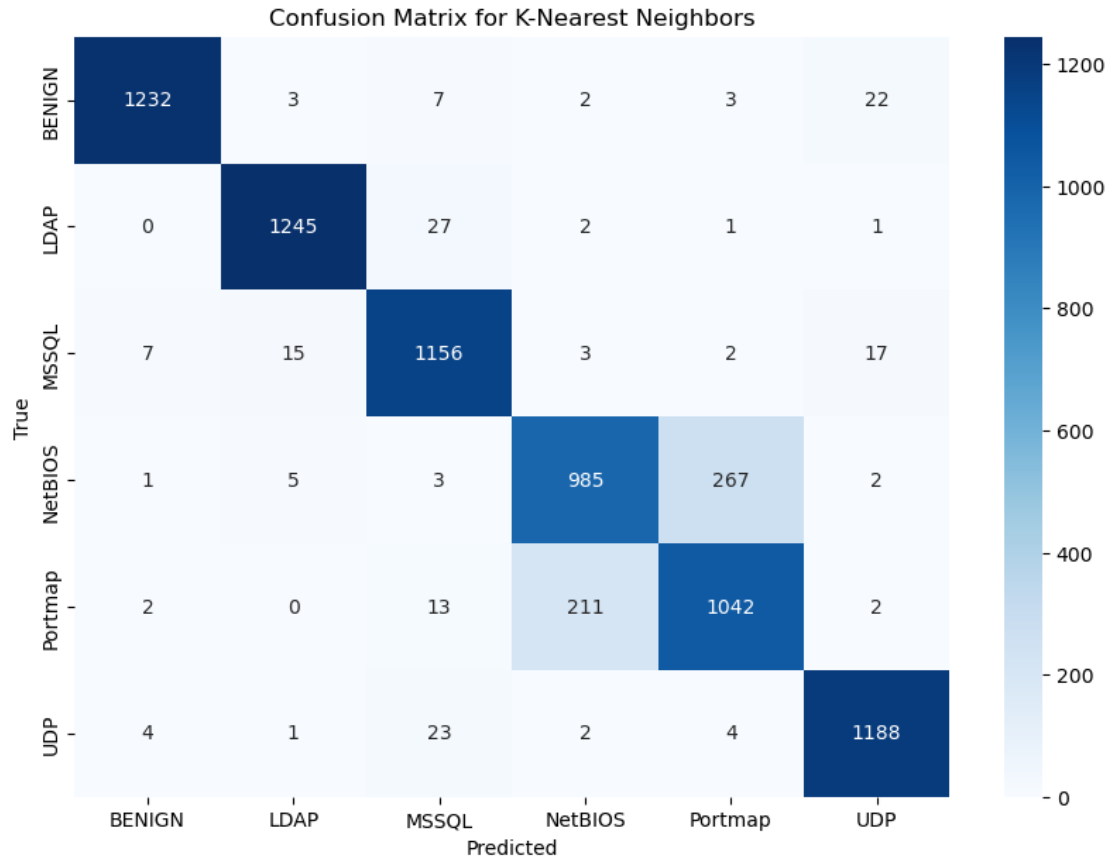


Figure 4.12: Multiclass classification of KNN without feature selection

The KNN confusion matrix is shown in the figure below. The model has an accuracy of 91% and time consumed by the model for training is 2.099 seconds and the testing time is 1.66seconds. The has detects the Benign, LDAP, MSSQL and UDP attack with slight mis prediction but there is a large gap in predicting the NETBIOS| and Portmap attack. The model's accuracy drops due to mis calculation of NETBIOS and Portmap attack.

#### 4.2.2 Principal Component Analysis (PCA)

In this section PCA dimension reduction method is used for detecting the attacks. It can be observed that the SVM has an accuracy of 85%, followed by the KNN with 84% accuracy.



Table 4.4 Multiclass classification using PCA.

<b>Model</b>	<b>Mean Cross validation</b>	<b>Accur acy</b>	<b>Precisio n</b>	<b>Recall</b>	<b>F1 Score</b>	<b>Training time (s)</b>	<b>Testing time (s)</b>
<b>LR</b>	0.79	0.79	0.79	0.79	0.79	153.63	0.02
<b>SVM</b>	0.85	0.85	0.86	0.85	0.84	99.25	26.33
<b>RF Model</b>	0.83	0.83	0.84	0.83	0.81	58.74	0.14
<b>KNN</b>	0.84	0.84	0.84	0.84	0.84	0.03	3.38

The results shown above demonstrate the performance of the models. The accuracy of each model demonstrates the ability to detect the intrusion precisely. It can be observed that using PCA method KNN has accuracy of 84% and time consumed by the model is 0.03 seconds for training and 3.38 seconds for testing.

#### 4.2.2.1 Confusion Matrix

A confusion matrix can be used to provide an overview of the machine learning model's performance on a collection of test data. It's a means of illustrating how many cases, based on the model's predictions, are right or wrong. It is often used to evaluate the performance of categorisation models that try to give a categorical label to every input occurrence. The confusion matrix for all the four models is shown below.

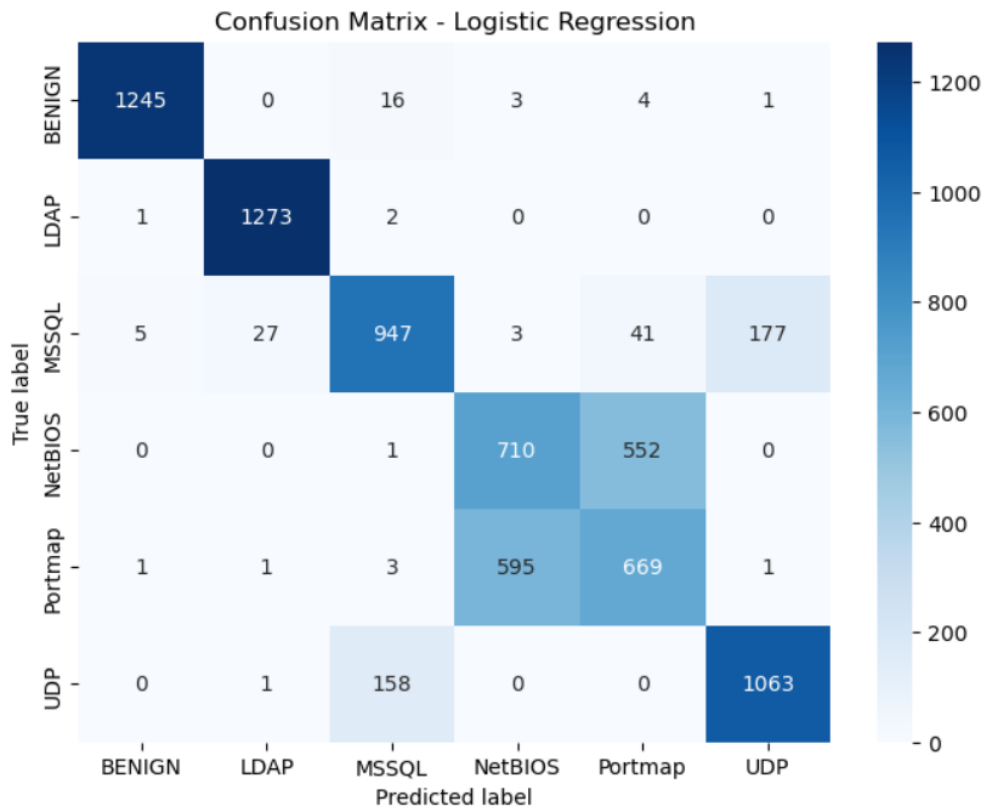


Figure 4.13: Multiclass classification for Logistic Regression using PCA

The confusion matrix for Logistic Regression shows the predicted and actual label. It can be seen that Benign attack is predicted as Benign for 1245 instance and there is a miss prediction as MSSQL, NetBIOS, Portmap and UDP. The LDAP is predicted as LDAP for 1273 instance and there is miss prediction as MSSQL for 2 times and as Benign for 1-time instance.

The MSSQL is rightly predicted for 947 instances. But there are considerable times of miss prediction. It is wrongly predicted as UDP for 177 and Portmap for 41 instances.

The NetBIOS is rightly predicted for 710 instances and miss predicted as Portmap 552 times.

In case of Portmap attack detection the model detected the attack rightly for 669 instances by had wrongly predicted as NetBIOS for 595 instances, which is roughly equals to correct detection.

The UDP attack is rightly predicted for 1063 instance but there is miss prediction as MSSQL for 158 times.

Since, the NetBIOS and Portmap attack has higher miss prediction the accuracy of the model is reduced to 79%.

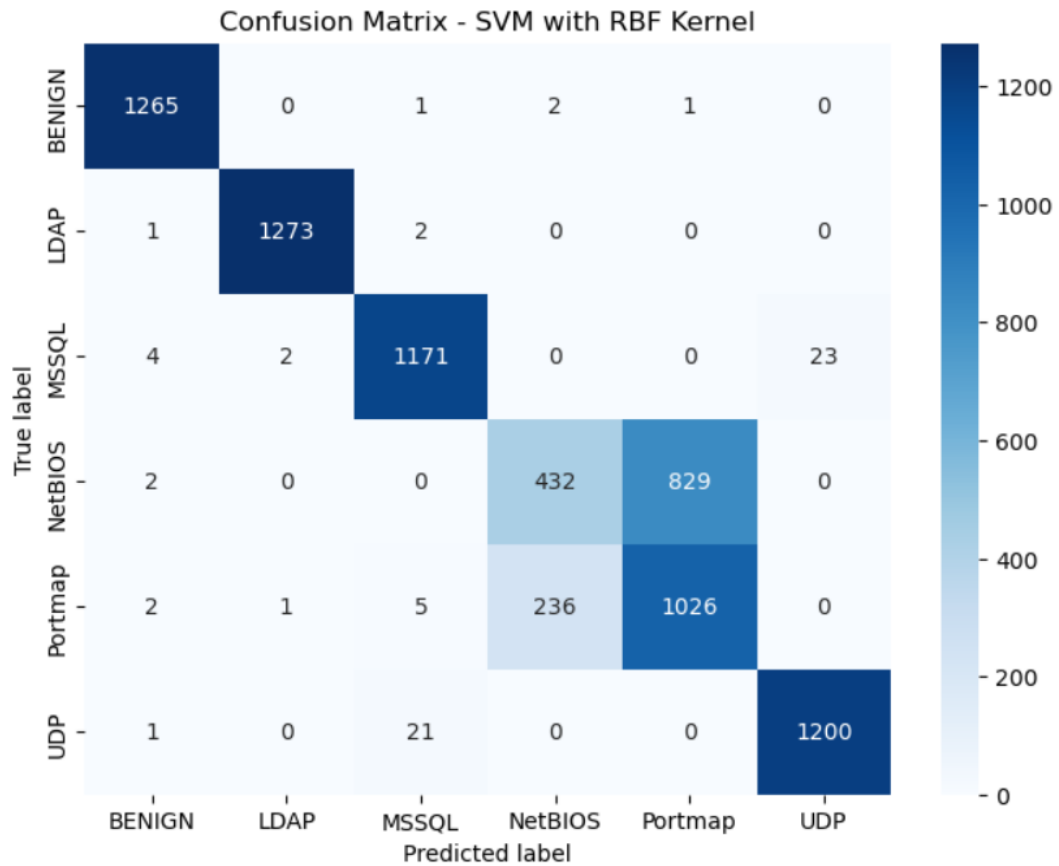


Figure 4.14: Multiclass classification for SVM using PCA

The confusion matrix for SVM shows the predicted and actual label. It can be observed that Benign attack is predicted as Benign for 1265 instance and there is a miss prediction as MSSQL, NetBIOS and Portmap. The LDAP is predicted as LDAP for 1273 instance and there is miss prediction as MSSQL for 2 times and as Benign for 1-time instance.

The MSSQL is rightly predicted for 1171 instances. There is miss prediction of 4 times as Benign, 2 times as LDAP and 23 instances of UDP.

The NetBIOS is rightly predicted for 432 instances and miss predicted as Portmap 829 times. Miss prediction is more than the right prediction.

In case of Portmap attack detection the model detected the attack rightly for 1026 instances by had wrongly predicted as NetBIOS for 236 instances, which is roughly equals to correct detection.

The UDP attack is rightly predicted for 1200 instance but there is miss prediction as MSSQL for 21 times.

Since, the NetBIOS and Portmap attack has higher miss prediction, the accuracy of the model is reduced to 85%.

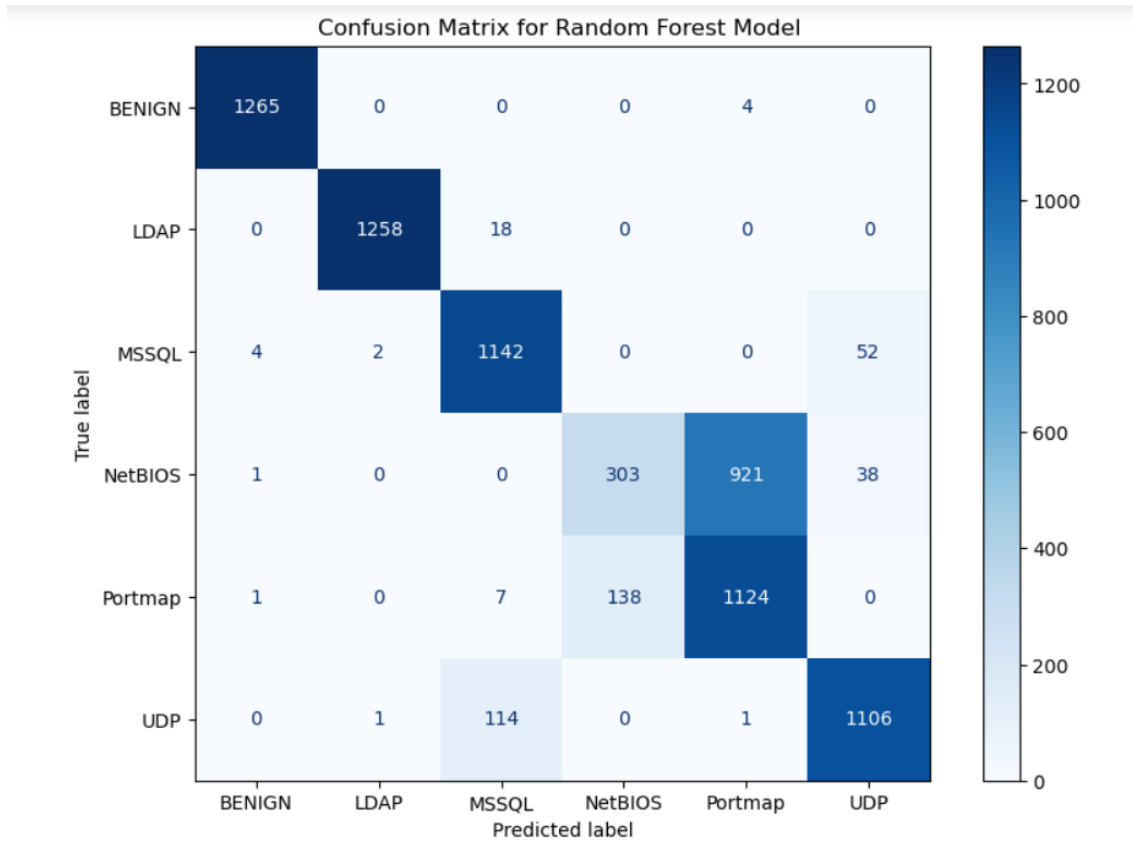


Figure 4.15: Multiclass classification for Random Forest using PCA

The confusion matrix for Random Forest Model shows the predicted and actual label. It can be observed that Benign attack is predicted as Benign for 1265 instance and there is a negligible amount of miss prediction as Portmap and NetBIOS. The LDAP is predicted as LDAP for 1258 instance and there is miss prediction as MSSQL for 7 times.

The MSSQL is miss predicted as Benign for 4 instance, 2 time as LDAP and 52 times as UDP. It can be seen there is huge miss prediction in case of NETBIOS attack. The NetBIOS is miss predicted as Portmap 921 times which is greater than correct prediction by the model.

In case of Portmap attack detection the model detected the attack rightly for 1124 instance by had wrongly predicted as NetBIOS for 138 instances.

The UDP has better prediction score, but it is wrongly predicted as MSSQL for 114 times. Since, the NetBIOS attack is wrongly predicted 905 times the accuracy of the model has come down to 83%.

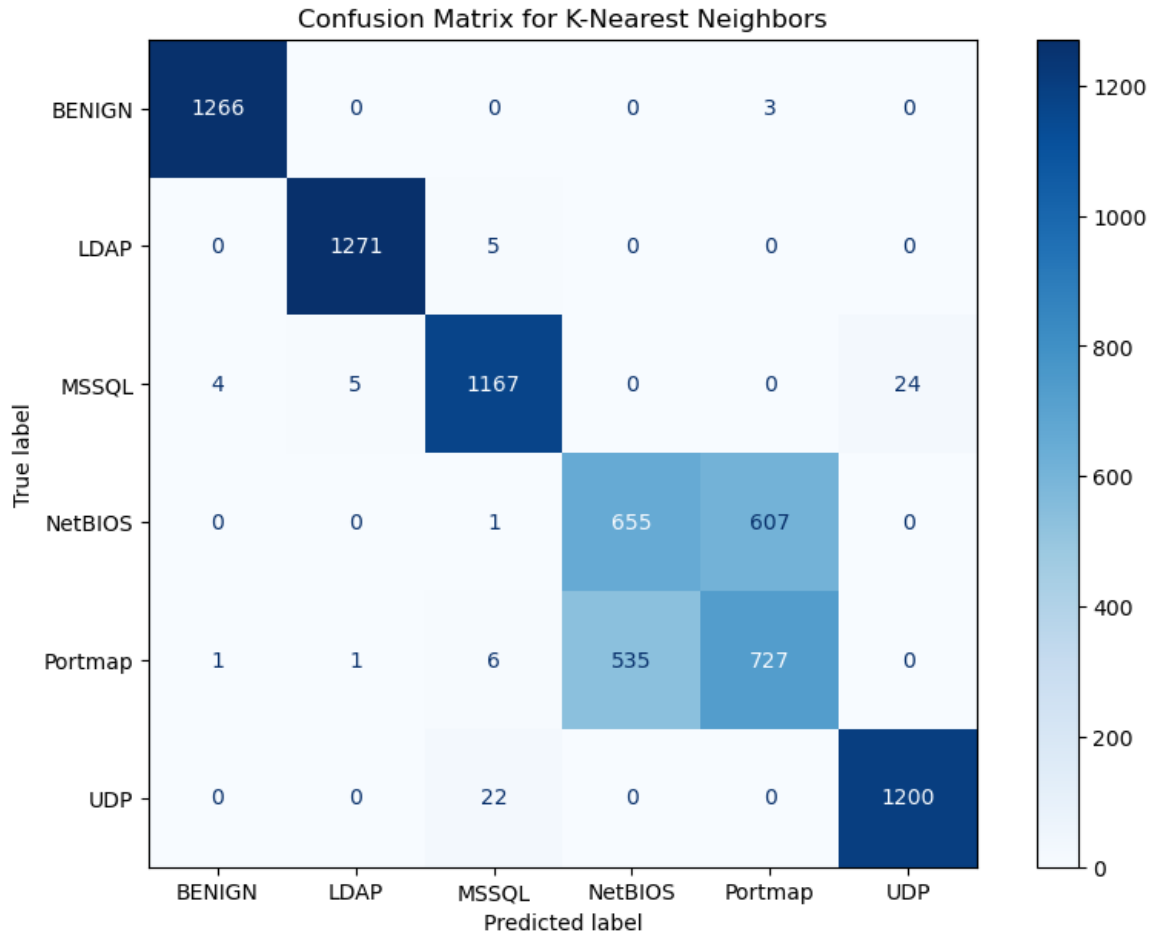


Figure 4.16: Multiclass classification for KNN using PCA

The confusion matrix for KNN is shown in the Figure 4.16. The confusion matrix shows that the model identifies the Benign and LDAP attack with the high accuracy. The accuracy of the MSSQL and UDP is slightly reduced. But the accuracy of the overall model is reduced because of the miscalculation of the NetBIOS and Portmap attacks.

### 4.2.3 SelectKBest Features

In this section the model prediction with SelectKBest feature method is explained. There are three K features selected K=10, 20 and 30. The comparison of all the models are shown in the section below. It can be observed that as the number of features increases the accuracy of Logistic Regression model increases and accuracy of SVM model decreases. The Random Forest and KNN models performed with greater accuracy for SelectKBest feature selection irrespective of number of features.

#### 4.2.3.1 SelectKBest with 10 features

The K=10 value gives the following performance metrics shown in the table 4.5. It can be observed from the table that the Random Forest model performance has 100% accuracy and time consumed by the model to train is 5.79 and test is 0.19 seconds. The KNN model gives an accuracy of 98% and time consumed by the model is 0.96s for a training and 0.5 seconds for testing.

It can be concluded that the RF and KNN both performed well, and time consumed by KNN model is less than the RF model. It can also be observed that the SVM model performance increases for the fewer features.

The Selected 10 features are: Source IP, Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Flow Bytes/s, Min Packet Length, Packet Length Mean, Average Packet Size, Avg Fwd Segment Size, Attack Type\_encoded .

Table 4.5 Multiclass classification using SelectKBest feature =10

<b>Model</b>	<b>Mean Cross validation</b>	<b>Accur acy</b>	<b>Precisio n</b>	<b>Recall</b>	<b>F1 Score</b>	<b>Training time (s)</b>	<b>Testing time (s)</b>
<b>LR</b>	0.19	0.20	0.19	0.20	0.10	0.95	0.14
<b>SVM</b>	0.94	0.95	0.96	0.95	0.95	401.37	23.08
<b>RF Model</b>	1.00	1.00	1.00	1.00	1.00	5.79	0.19
<b>KNN</b>	0.98	0.98	0.98	0.98	0.98	0.96	0.50

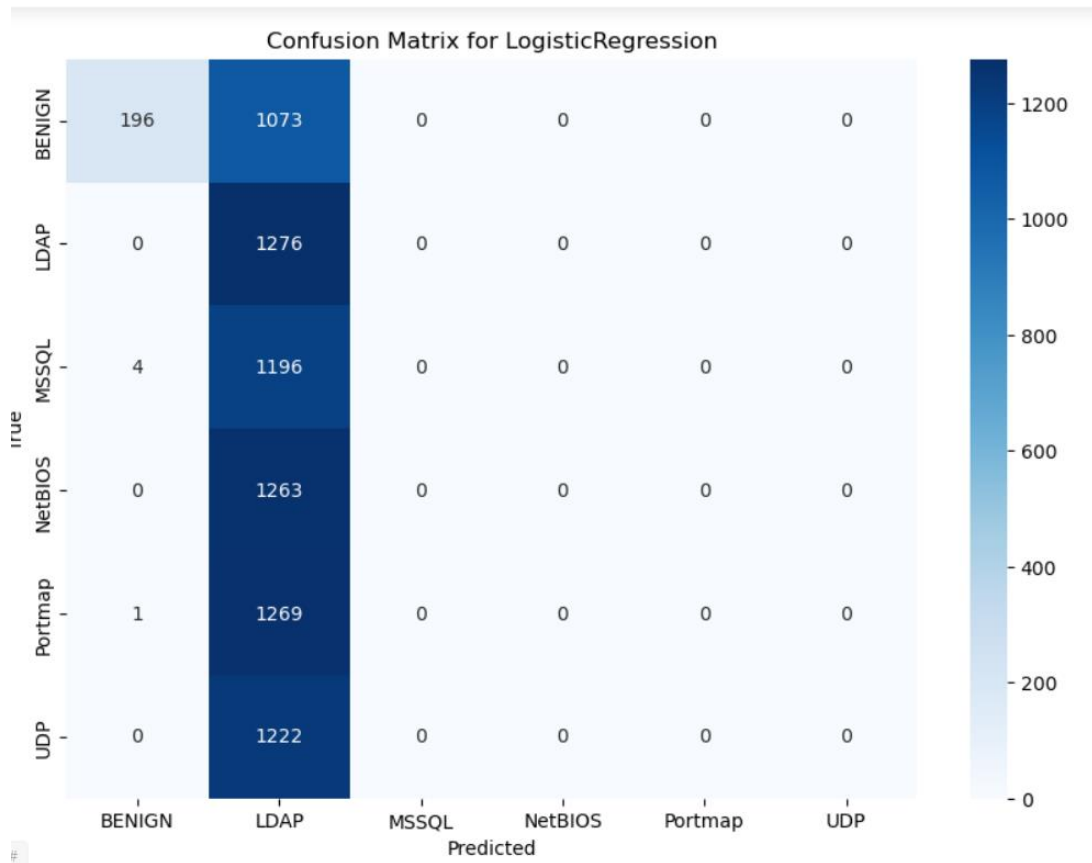


Figure 4.17: Multiclass classification for Logistic Regression using SelectKBest =10

The Logistic Regression method failed in predicting the attack accurately with K=10 features. This model cannot be used for predicting the attack with few features. As logistic regression assumes a linear relationship, adding more informative features can help create a clearer separation between classes, improving the model's performance.

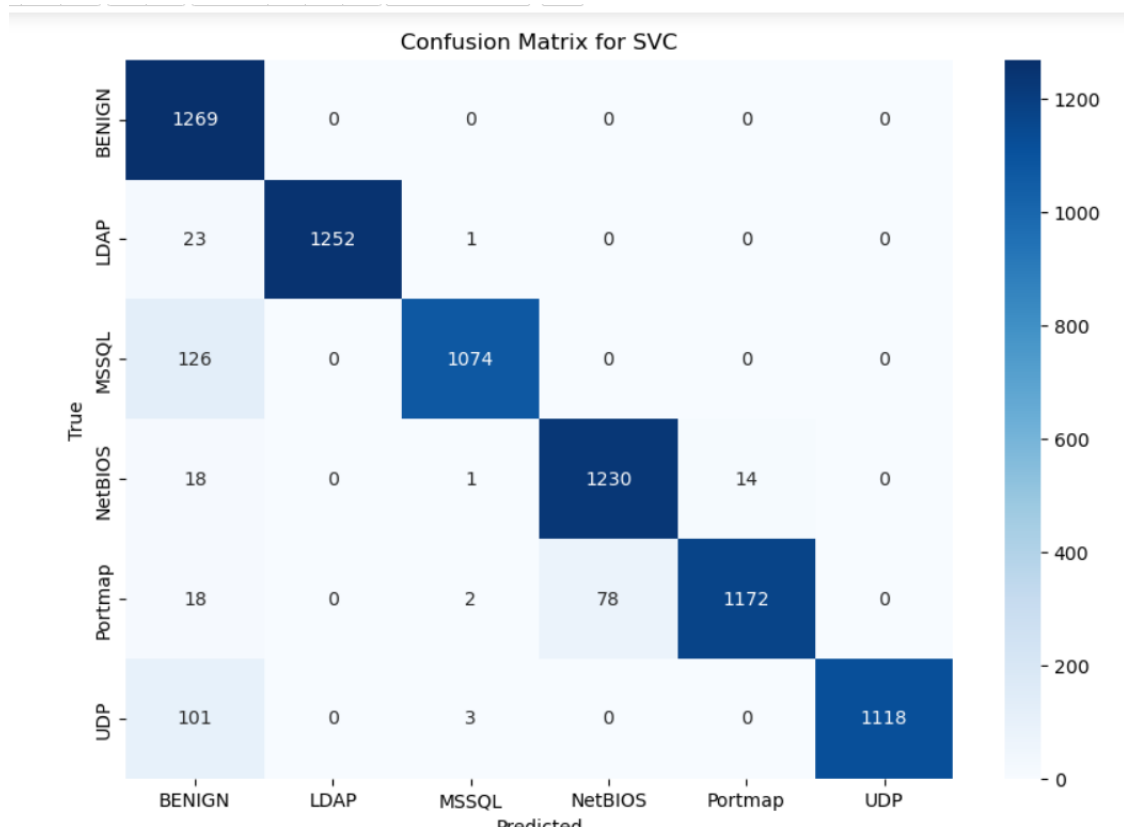


Figure 4.18: Multiclass classification for SVM using SelectKBest =10

The SVM model performed well in predicting the attack. It has an accuracy of 95%. The model takes 401.37s to train the model and 23.08s to test the model. The SVM model performed well with minimum number of features. Unlike the PCA, the with 10 features SVM has better detection for Portmap and NETBIOS attacks.



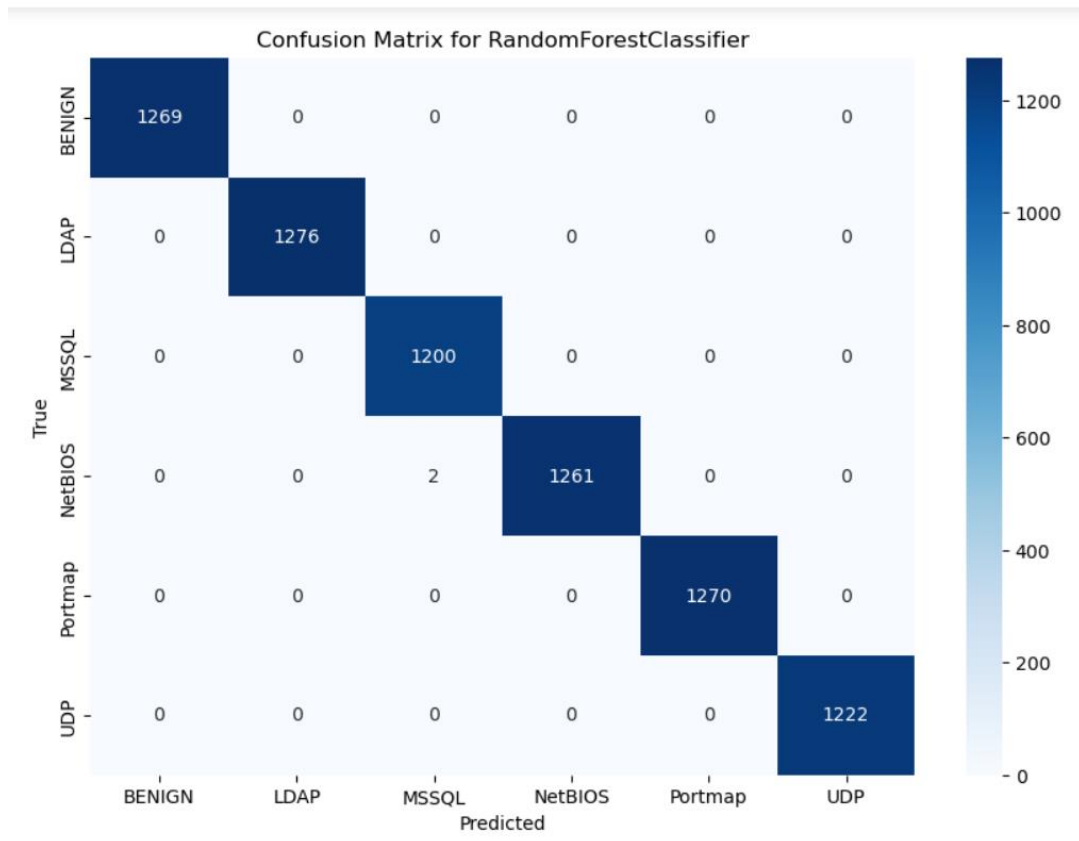


Figure 4.19: Multiclass classification for RF using SelectKBest =10

The Random Forest model has a 100% accuracy. This model has predicted all the attacks accurately. There is a negligible amount of mis calculation in case of NetBIOS. This model has consumed 5.79s to train the model and 0.19s to test the model

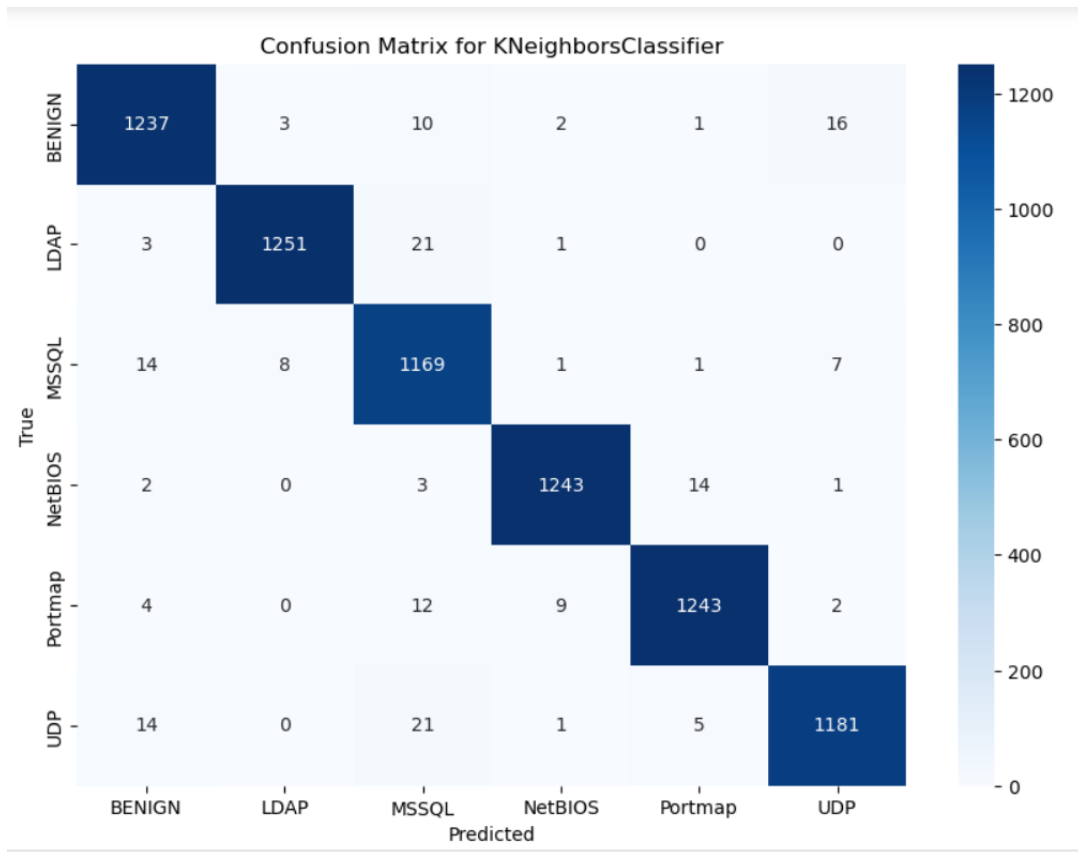


Figure 4.20: Multiclass classification for KNN using SelectKBest =10

The performance of KNN model with 10 features is 98%. The model is able to detect all the attacks precisely. The model takes 0.96s for training and 0.5s s for testing the model. It can be said that this model is faster compared to other models with the accuracy of 98%.

#### 4.2.3.2 The SelectKBest =20 features.

The SelectKBest features, where K=20 has 20 features selected from 88 features. With this feature selection method Random Forest model has 100% accuracy and KNN has 97% accuracy. The time consumed by the RF for training the model is 8.45s and 0.18s for testing the model. The KNN model takes 2.68s to train and 2.00s to test the model. The Logistic Regression has 81% accuracy, and it can be observed that the accuracy of LR increases with the number of features. The accuracy of SVM model decreases with the number of features.

The K=20 features selected are Flow ID, Source IP, Source Port, Protocol, Timestamp, Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Flow Bytes/s, Flow Packets/s, Fwd Packets/s, Min Packet Length, Max Packet Length, Packet

Length Mean, URG Flag Count, Down/Up Ratio, Average Packet Size, Avg Fwd Segment Size, Inbound, Attack Type\_encoded.

Table 4.6 Multiclass classification using SelectKBest feature =20

Model	Mean Cross validation	Accuracy	Precision	Recall	F1 Score	Training time (s)	Testing time (s)
LR	0.81	0.86	0.86	0.86	0.86	196.39	0.17
SVM	0.18	0.18	0.86	0.18	0.08	1425.86	45.96
RF Model	1.00	1.00	1.00	1.00	1.00	8.45	0.18
KNN	0.97	0.97	0.97	0.97	0.97	2.68	2.00

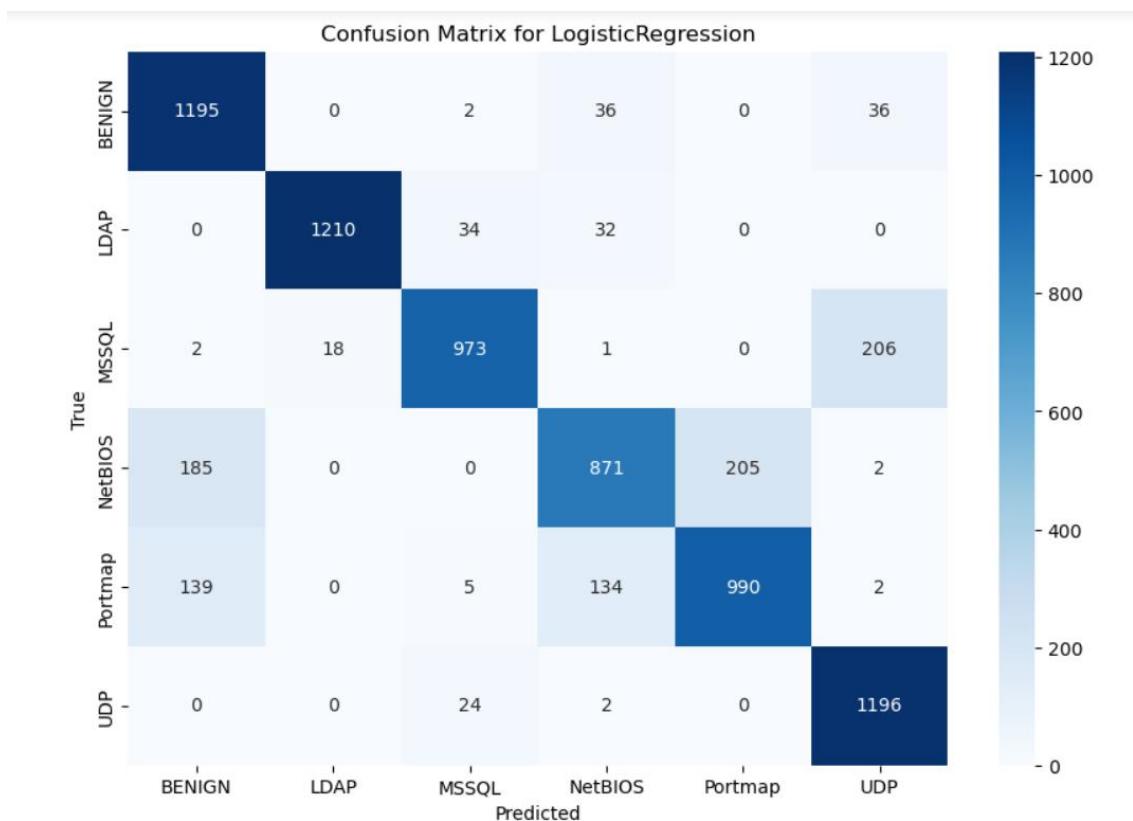


Figure 4.21: Multiclass classification for Logistic Regression using SelectKBest =20

The LR confusion matrix shows that the model has good prediction capabilities for Benign, LDAP and UDP attack. There is more mis calculation of attacks for MSSQL,

NetBIOS and Portmap. Overall the model has 86% of accuracy and time consumed for training the model is 196.39s and 0.17 for testing the model.

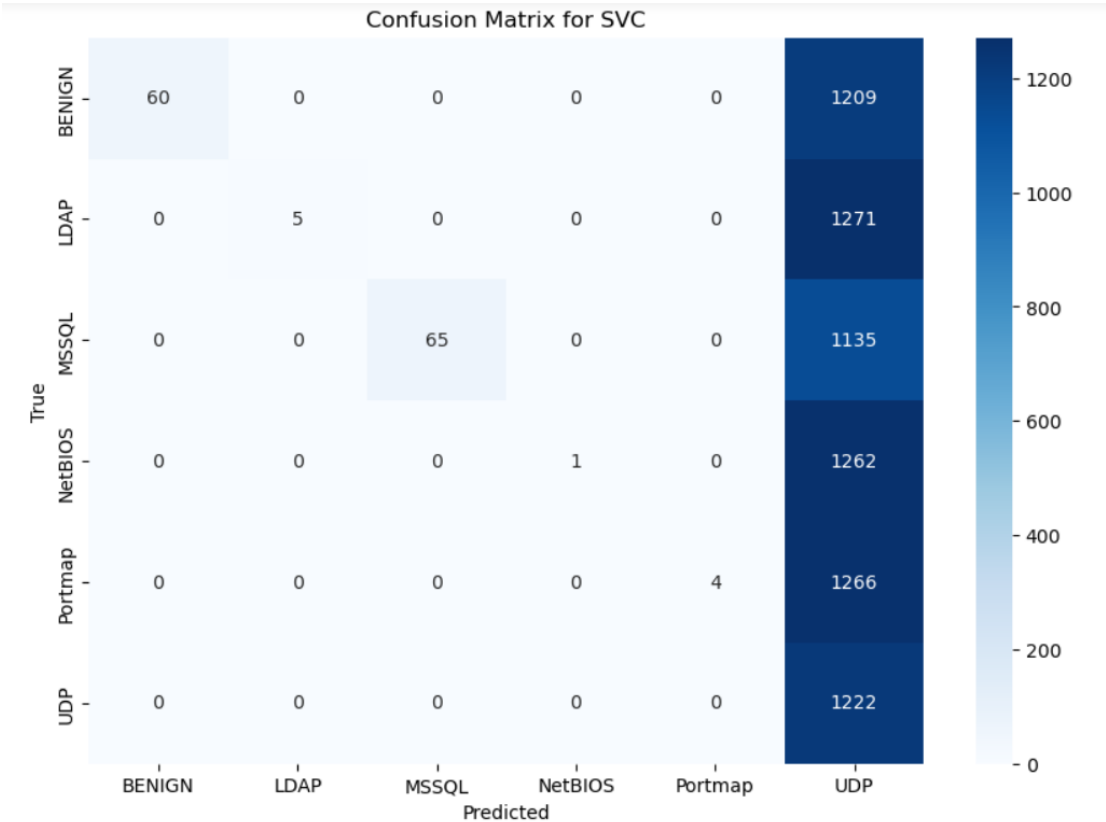


Figure 4.22: Multiclass classification for SVM using SelectKBest =20

The confusion matrix of SVM shows that the model completely fails in detecting the attacks. The accuracy of the model is 18%. This shows that the SVM model fails for SelectKBest Feature Selection method where the number features are more.

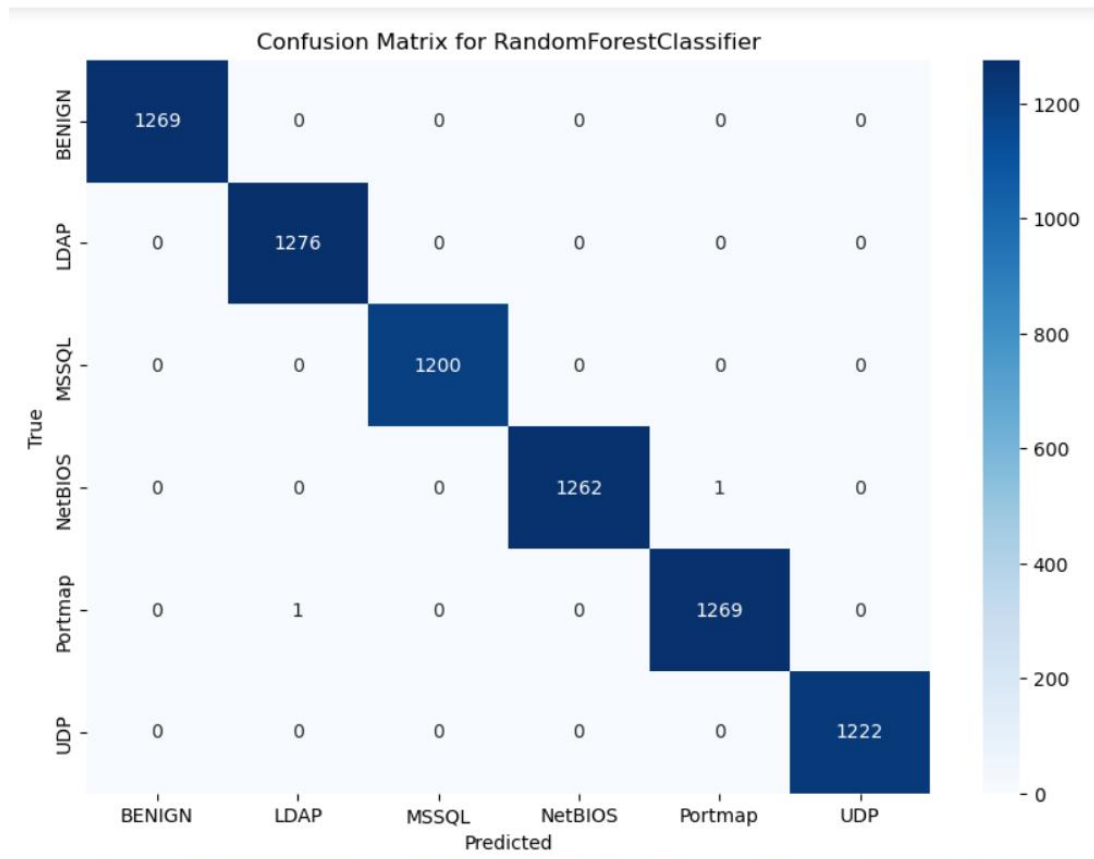


Figure 4.23: Multiclass classification for RF using SelectKBest =20

The Random Forest model performs excellently for SelectKBest Feature selection method irrespective of number of features. This model has 100% accuracy and time consumed by the model 8.45 and 0.18s for training and testing the model respectively.

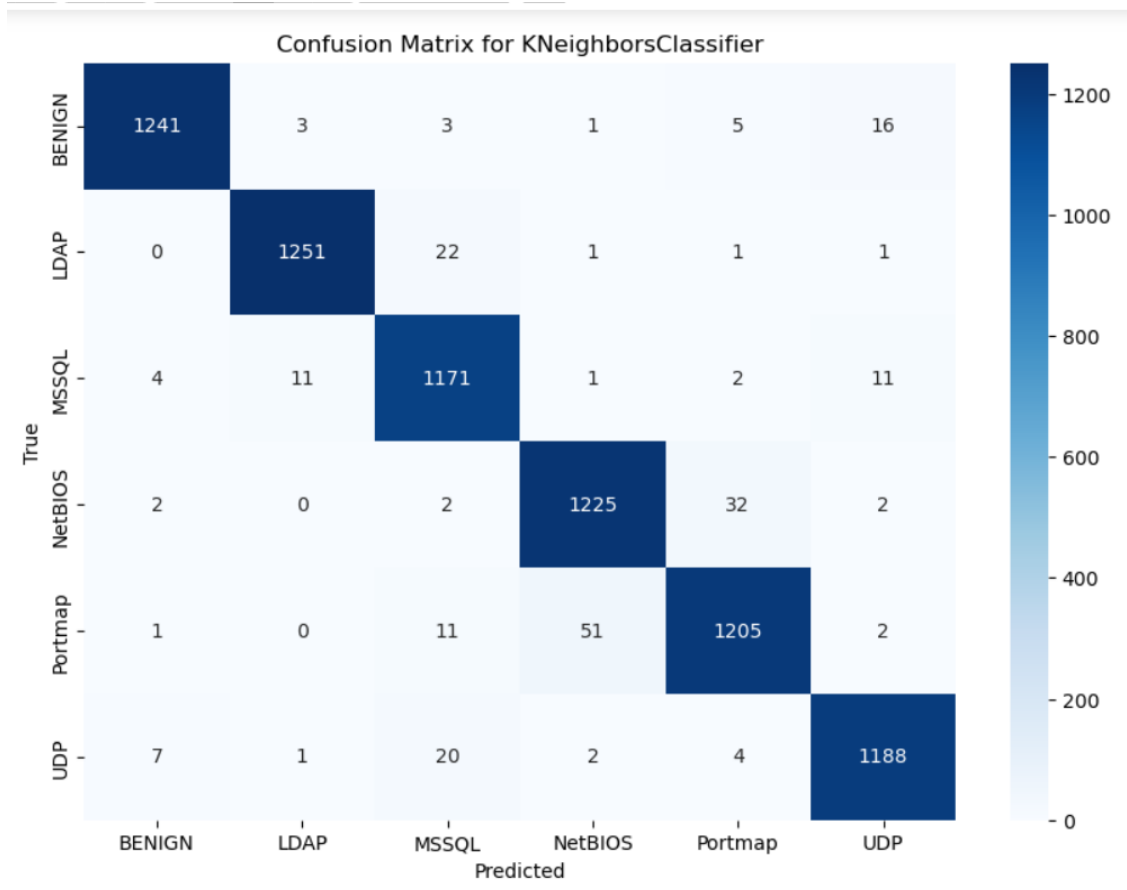


Figure 4.24: Multiclass classification for KNN using SelectKBest =20

The KNN model performance has been second best for the SelectKBest Feature selection method. This model has taken very less time compared to all the other models for training and testing the model. This model is very effective when there is time restriction.

This model with 20 features has 97% of accuracy and time consumed for training and testing the model is 2.68s and 2.00s.

#### 4.2.3.3 SelectKBest =30

With SelectKbest features = 30, the Rf model has 100% accuracy. The KNN model has slightly drop in the accuracy with increase in the features. The accuracy of LR model increases with the features and SVM accuracy decreases dramatically with the slight increase in the features.

The 30 features selected are as follows: Flow ID, Source IP, Source Port, Destination IP, Destination Port, Protocol, Timestamp, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Bwd Packet Length Min, Flow Bytes/s, Flow Packets/s, Fwd Packets/s, Min Packet Length, Max Packet

Length, Packet Length Mean, RST Flag Count, ACK Flag Count, URG Flag Count, CWE Flag Count, Down/Up Ratio, Average Packet Size, Avg Fwd Segment Size, Subflow Fwd Bytes, Init\_Win\_bytes\_forward, Inbound, Attack Type\_encoded

Table 4.7 Multiclass classification using SelectKBest feature =30

Model	Mean Cross validation	Accuracy	Precision	Recall	F1 Score	Training time (s)	Testing time (s)
LR	0.83	0.79	0.80	0.79	0.79	194.14	0.14
SVM	0.17	0.16	0.19	0.16	0.05	2729.59	80.08
RF Model	1.00	1.00	1.00	1.00	1.00	21.54	0.53
KNN	0.90	0.92	0.92	0.92	0.92	5.11	3.39

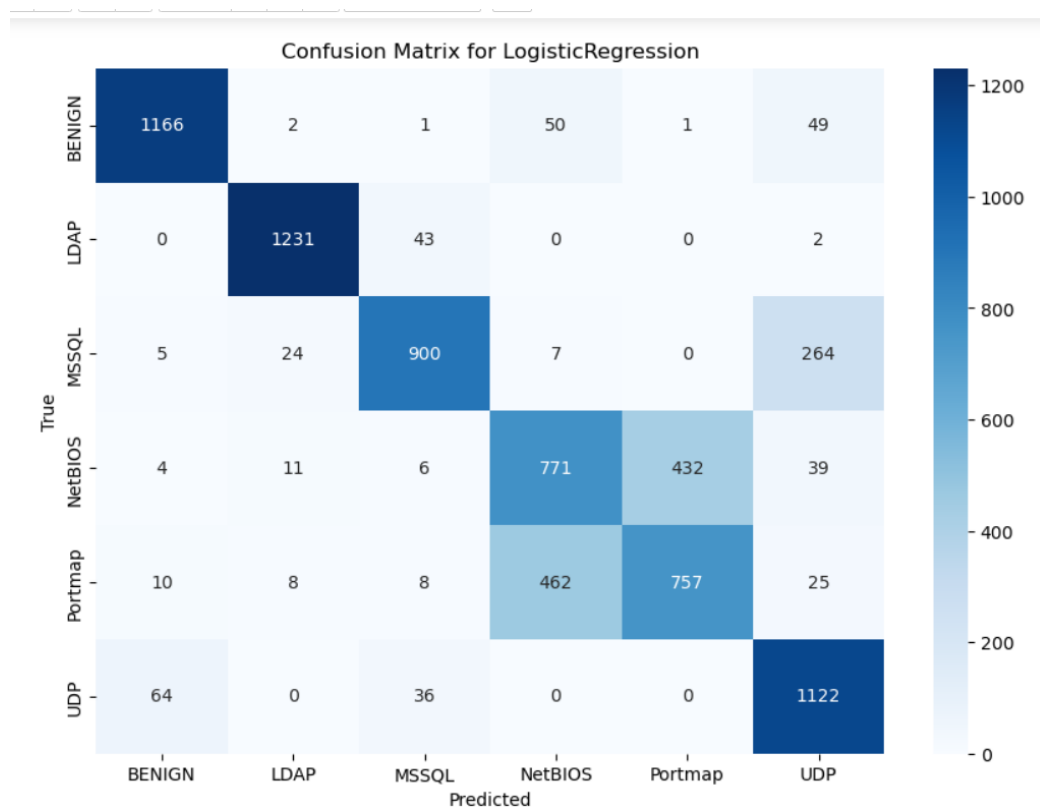


Figure 4.25: Multiclass classification for Logistic Regression using SelectKBest =30

The confusion matrix of LR shows that the model detects the Benign, LDAP and UDP. There MSSQL, NetBIOS, Portmap has lower detection rate. The accuracy of model 79% and has 194.14s (3.37 min) for training and 0.14s for testing.

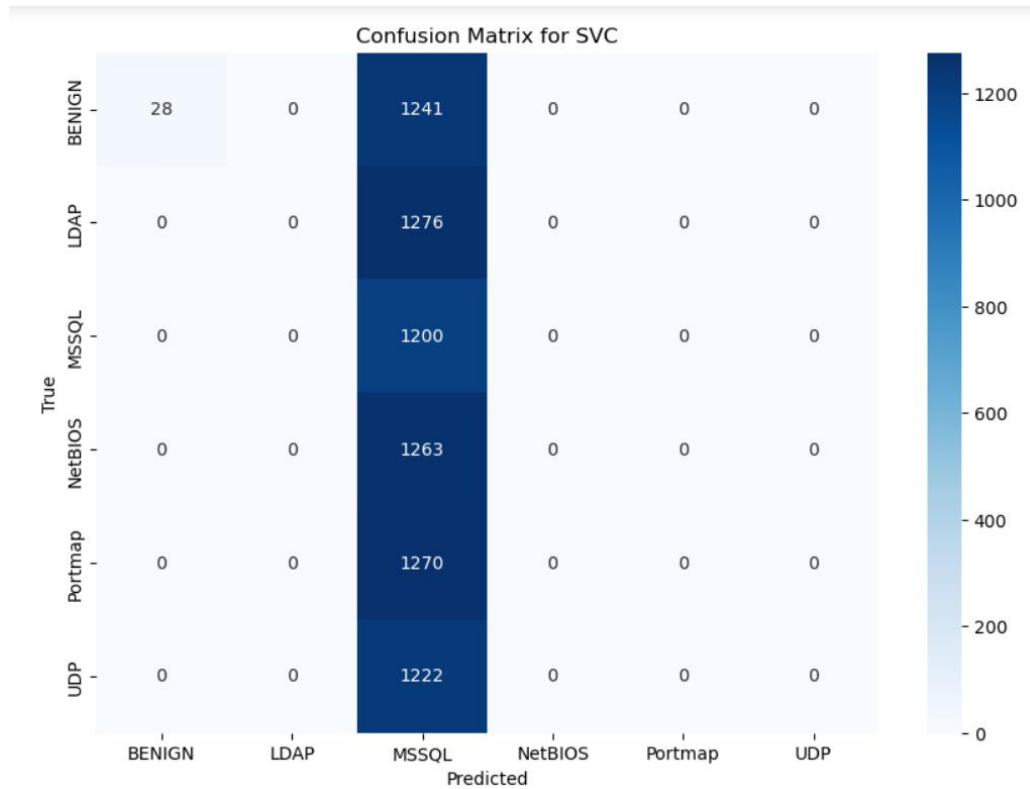


Figure 4.26: Multiclass classification for SVM using SelectKBest =30

The confusion matrix of SVM shows that the model completely fails in detecting the attacks. The accuracy of the model is 16%. This shows that the SVM model fails for SelectKBest Feature Selection method where the number features are more.



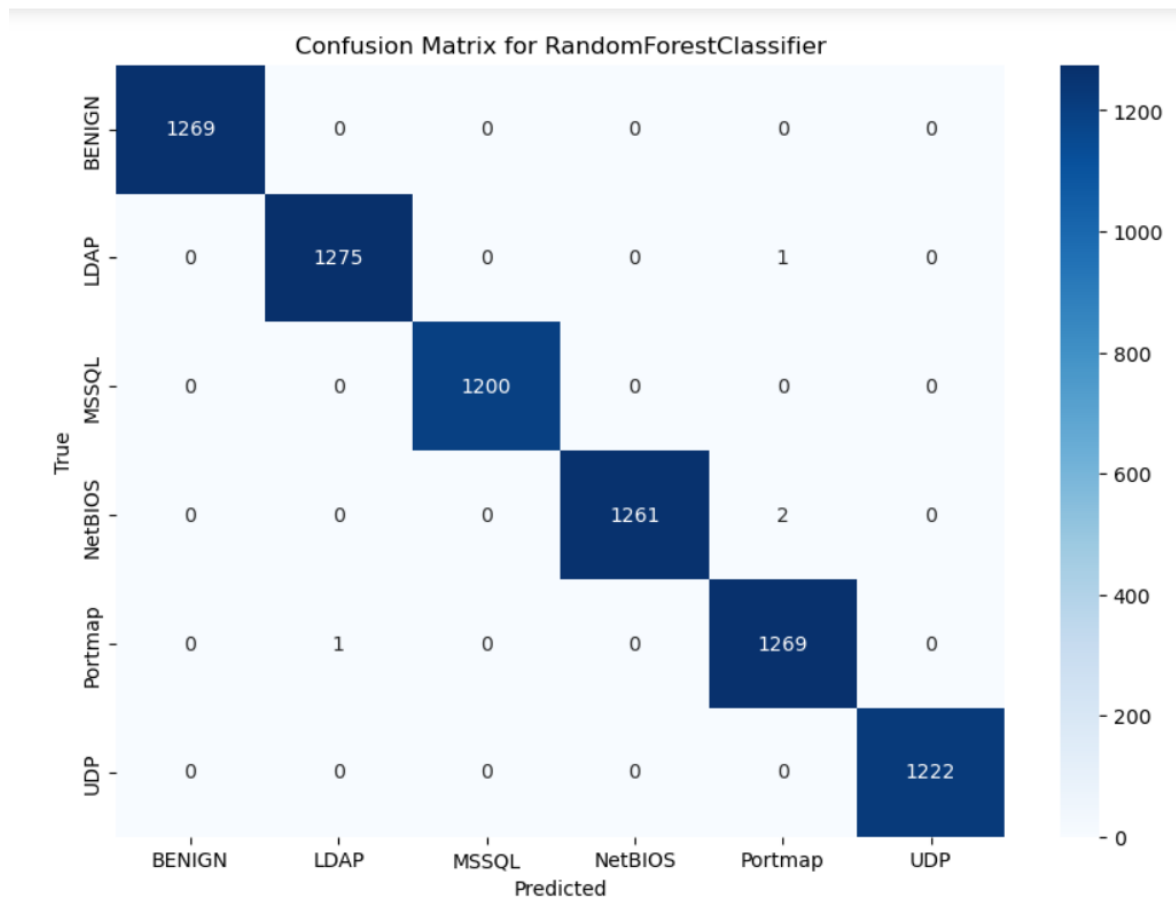


Figure 4.27: Multiclass classification for RF using SelectKBest =30

The Random Forest model performs excellently for SelectKBest Feature selection method irrespective of number of features. This model has 100% accuracy and time consumed by the model 21.54s and 0.53s for training and testing the model respectively.

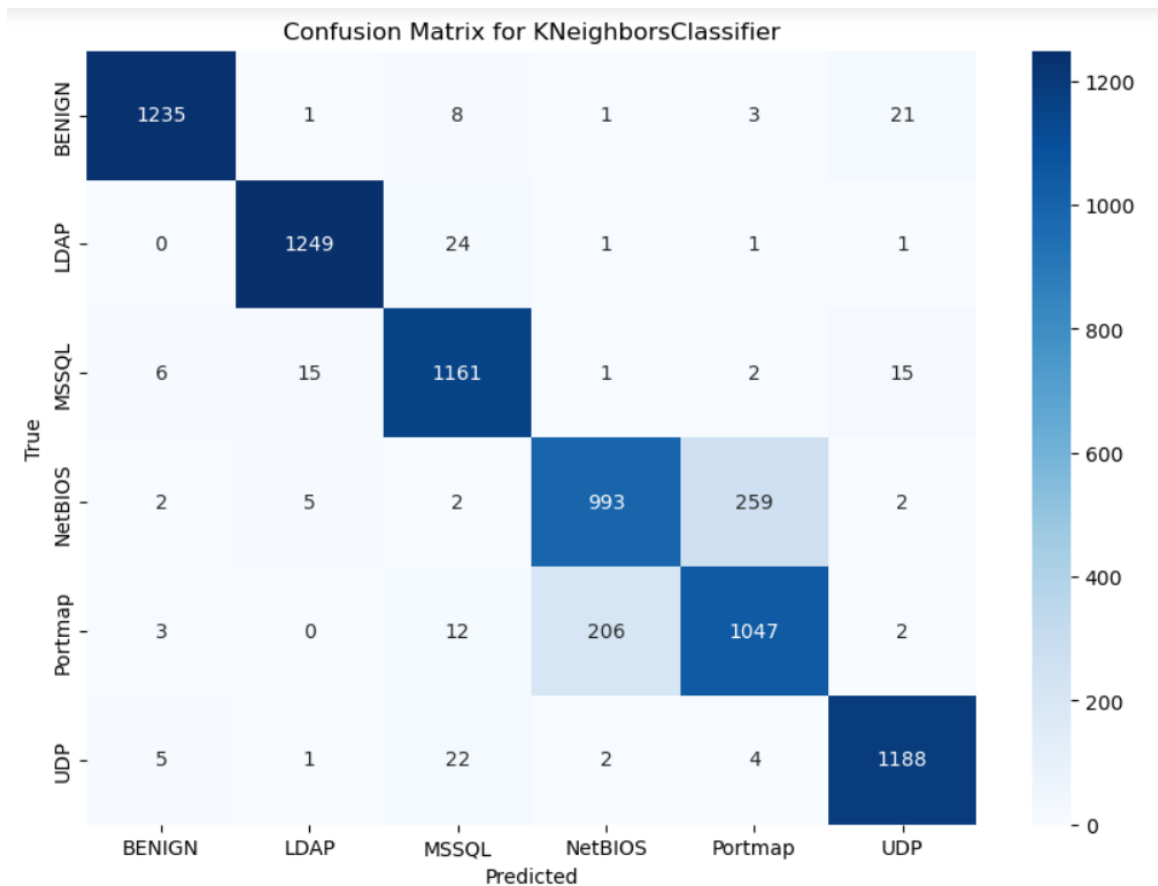


Figure 4.28: Multiclass classification for KNN using SelectKBest =30

The KNN model performance has been second best for the SelectKBest Feature selection method. This model has taken very less time compared to all the other models for training and testing the model. This model is very effective when there is time restriction.

This model with 30 features has 92% of accuracy and time consumed for training and testing the model is 5.11s and 3.39s.

### 4.3 Deep Learning Model

Two deep learning models are implemented namely CNN and LSTM. The CNN model has an accuracy of 100% and LSTM has 99%. The components in output are Epoch, Step, Loss, accuracy, Val\_loss, Val\_accuracy

**Epoch:** Each epoch represents the training of the model over all samples.

**Step:** Refers to the number of batches processed in each epoch. Here, you have 563 steps per epoch, meaning the dataset is divided into 563 batches.

**Loss:** A metric that quantifies how well or poorly the model is performing. Lower loss is better.

**Accuracy:** The proportion of correctly classified attacks.

**Val\_loss:** Validation loss, used to monitor how well the model generalizes to unseen data.

**Val\_accuracy:** Validation accuracy, which measures the performance on the validation dataset.

**ms/step:** The time it takes to process one batch of data.

### 4.3.1 CNN Model

The output of CNN model is shown below. It can be observed that the model has good accuracy in detecting the attacks and low validation loss. The confusion matrix also shows how well the model can identify each attack.

Training accuracy increases at every epoch. It started with 90.11% and increased up to 99.82%. The loss of 0.2949 at the first epoch also decreased at the later epochs.

The validation accuracy starts at 99.16% and remained high throughout the training. The validation loss fluctuates more, starting at 0.0306, but gradually decreases to 0.0062 by the last epoch.

Epoch 1/50

**563/563** ————— **87s** 150ms/step - accuracy: 0.9011 - loss: 0.2949 - val\_accuracy: 0.9916 - val\_loss: 0.0306

Epoch 2/50

**563/563** ————— **123s** 116ms/step - accuracy: 0.9833 - loss: 0.0459 - val\_accuracy: 0.9904 - val\_loss: 0.0224

Epoch 3/50

**563/563** ————— **29s** 51ms/step - accuracy: 0.9890 - loss: 0.0300 - val\_accuracy: 0.9860 - val\_loss: 0.0343

Epoch 4/50

**563/563** ————— **29s** 52ms/step - accuracy: 0.9887 - loss: 0.0293 - val\_accuracy: 0.9944 - val\_loss: 0.0144

Epoch 5/50

**563/563** ————— **33s** 59ms/step - accuracy: 0.9933 - loss: 0.0193 - val\_accuracy: 0.9931 - val\_loss: 0.0198

Epoch 6/50

**563/563** ————— **31s** 55ms/step - accuracy: 0.9923 - loss: 0.0189 - val\_accuracy: 0.9913 - val\_loss: 0.0230

Epoch 7/50

**563/563** ————— **30s** 54ms/step - accuracy: 0.9937 - loss: 0.0172 - val\_accuracy: 0.9967 - val\_loss: 0.0084

Epoch 8/50

**563/563** ————— **30s** 54ms/step - accuracy: 0.9935 - loss: 0.0181 - val\_accuracy: 0.9920 - val\_loss: 0.0368

Epoch 9/50

**563/563** ————— **31s** 55ms/step - accuracy: 0.  
 9925 - loss: 0.0202 - val\_accuracy: 0.9918 - val\_loss: 0.0198  
 Epoch 10/50

**563/563** ————— **32s** 57ms/step - accuracy: 0.  
 9930 - loss: 0.0195 - val\_accuracy: 0.9956 - val\_loss: 0.0109  
 Epoch 11/50

**563/563** ————— **33s** 59ms/step - accuracy: 0.  
 9951 - loss: 0.0117 - val\_accuracy: 0.9938 - val\_loss: 0.0146  
 Epoch 12/50

**563/563** ————— **37s** 65ms/step - accuracy: 0.  
 9932 - loss: 0.0154 - val\_accuracy: 0.9967 - val\_loss: 0.0094  
 Epoch 13/50

**563/563** ————— **33s** 58ms/step - accuracy: 0.  
 9947 - loss: 0.0165 - val\_accuracy: 0.9967 - val\_loss: 0.0078  
 Epoch 14/50

**563/563** ————— **32s** 57ms/step - accuracy: 0.  
 9954 - loss: 0.0102 - val\_accuracy: 0.9982 - val\_loss: 0.0059  
 Epoch 15/50

**563/563** ————— **33s** 58ms/step - accuracy: 0.  
 9974 - loss: 0.0076 - val\_accuracy: 0.9980 - val\_loss: 0.0068  
 Epoch 16/50

**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9966 - loss: 0.0077 - val\_accuracy: 0.9989 - val\_loss: 0.0056  
 Epoch 17/50

**563/563** ————— **33s** 59ms/step - accuracy: 0.  
 9961 - loss: 0.0093 - val\_accuracy: 0.9947 - val\_loss: 0.0148  
 Epoch 18/50

**563/563** ————— **32s** 57ms/step - accuracy: 0.  
 9960 - loss: 0.0110 - val\_accuracy: 0.9951 - val\_loss: 0.0121  
 Epoch 19/50

**563/563** ————— **34s** 60ms/step - accuracy: 0.  
 9964 - loss: 0.0091 - val\_accuracy: 0.9964 - val\_loss: 0.0088  
 Epoch 20/50

**563/563** ————— **35s** 61ms/step - accuracy: 0.  
 9973 - loss: 0.0065 - val\_accuracy: 0.9976 - val\_loss: 0.0070  
 Epoch 21/50

**563/563** ————— **33s** 59ms/step - accuracy: 0.  
 9975 - loss: 0.0062 - val\_accuracy: 0.9973 - val\_loss: 0.0075  
 Epoch 22/50

**563/563** ————— **34s** 61ms/step - accuracy: 0.  
 9954 - loss: 0.0131 - val\_accuracy: 0.9976 - val\_loss: 0.0144  
 Epoch 23/50

**563/563** ————— **34s** 61ms/step - accuracy: 0.  
 9968 - loss: 0.0073 - val\_accuracy: 0.9907 - val\_loss: 0.0280  
 Epoch 24/50

**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9965 - loss: 0.0107 - val\_accuracy: 0.9964 - val\_loss: 0.0083  
 Epoch 25/50

**563/563** ————— **38s** 57ms/step - accuracy: 0.  
 9976 - loss: 0.0065 - val\_accuracy: 0.9960 - val\_loss: 0.0106  
 Epoch 26/50

**563/563** ————— **33s** 59ms/step - accuracy: 0.  
 9974 - loss: 0.0065 - val\_accuracy: 0.9904 - val\_loss: 0.0351  
 Epoch 27/50

**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9975 - loss: 0.0059 - val\_accuracy: 0.9967 - val\_loss: 0.0079  
 Epoch 28/50

**563/563** ————— **34s** 61ms/step - accuracy: 0.  
 9962 - loss: 0.0078 - val\_accuracy: 0.9982 - val\_loss: 0.0049  
 Epoch 29/50

**563/563** ————— **32s** 57ms/step - accuracy: 0.  
 9964 - loss: 0.0086 - val\_accuracy: 0.9967 - val\_loss: 0.0079  
 Epoch 30/50

**563/563** ————— **34s** 61ms/step - accuracy: 0.  
 9962 - loss: 0.0069 - val\_accuracy: 0.9947 - val\_loss: 0.0141  
 Epoch 31/50

**563/563** ————— **35s** 63ms/step - accuracy: 0.  
 9971 - loss: 0.0062 - val\_accuracy: 0.9964 - val\_loss: 0.0098  
 Epoch 32/50

**563/563** ————— **36s** 63ms/step - accuracy: 0.  
 9977 - loss: 0.0068 - val\_accuracy: 0.9927 - val\_loss: 0.0201  
 Epoch 33/50

**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9926 - loss: 0.0282 - val\_accuracy: 0.9976 - val\_loss: 0.0076  
 Epoch 34/50

**563/563** ————— **34s** 61ms/step - accuracy: 0.  
 9966 - loss: 0.0074 - val\_accuracy: 0.9971 - val\_loss: 0.0067  
 Epoch 35/50

**563/563** ————— **34s** 59ms/step - accuracy: 0.  
 9980 - loss: 0.0057 - val\_accuracy: 0.9967 - val\_loss: 0.0067  
 Epoch 36/50

**563/563** ————— **35s** 63ms/step - accuracy: 0.  
 9979 - loss: 0.0048 - val\_accuracy: 0.9976 - val\_loss: 0.0068  
 Epoch 37/50

**563/563** ————— **34s** 60ms/step - accuracy: 0.  
 9972 - loss: 0.0061 - val\_accuracy: 0.9958 - val\_loss: 0.0087  
 Epoch 38/50

**563/563** ————— **33s** 59ms/step - accuracy: 0.  
 9976 - loss: 0.0065 - val\_accuracy: 0.9980 - val\_loss: 0.0061  
 Epoch 39/50

**563/563** ————— **40s** 57ms/step - accuracy: 0.  
 9962 - loss: 0.0097 - val\_accuracy: 0.9973 - val\_loss: 0.0058  
 Epoch 40/50

**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9974 - loss: 0.0054 - val\_accuracy: 0.9980 - val\_loss: 0.0066  
 Epoch 41/50

**563/563** ————— **36s** 64ms/step - accuracy: 0.  
 9980 - loss: 0.0049 - val\_accuracy: 0.9964 - val\_loss: 0.0087  
 Epoch 42/50  
**563/563** ————— **35s** 63ms/step - accuracy: 0.  
 9978 - loss: 0.0050 - val\_accuracy: 0.9978 - val\_loss: 0.0072  
 Epoch 43/50  
**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9985 - loss: 0.0037 - val\_accuracy: 0.9967 - val\_loss: 0.0068  
 Epoch 44/50  
**563/563** ————— **34s** 60ms/step - accuracy: 0.  
 9977 - loss: 0.0052 - val\_accuracy: 0.9976 - val\_loss: 0.0093  
 Epoch 45/50  
**563/563** ————— **35s** 63ms/step - accuracy: 0.  
 9973 - loss: 0.0086 - val\_accuracy: 0.9964 - val\_loss: 0.0108  
 Epoch 46/50  
**563/563** ————— **36s** 64ms/step - accuracy: 0.  
 9967 - loss: 0.0083 - val\_accuracy: 0.9987 - val\_loss: 0.0065  
 Epoch 47/50  
**563/563** ————— **36s** 64ms/step - accuracy: 0.  
 9985 - loss: 0.0039 - val\_accuracy: 0.9969 - val\_loss: 0.0085  
 Epoch 48/50  
**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9977 - loss: 0.0065 - val\_accuracy: 0.9962 - val\_loss: 0.0090  
 Epoch 49/50  
**563/563** ————— **35s** 62ms/step - accuracy: 0.  
 9976 - loss: 0.0070 - val\_accuracy: 0.9969 - val\_loss: 0.0090  
 Epoch 50/50  
**563/563** ————— **34s** 61ms/step - accuracy: 0.  
 9982 - loss: 0.0044 - val\_accuracy: 0.9973 - val\_loss: 0.0062  
**235/235** ————— **4s** 17ms/step  
 Accuracy: 1.00  
 Precision: 1.00  
 Recall: 1.00  
 F1 Score: 1.00  
 AUC: 1.00  
 Training Time: 1841.24 seconds  
 Testing Time: 4.32 seconds

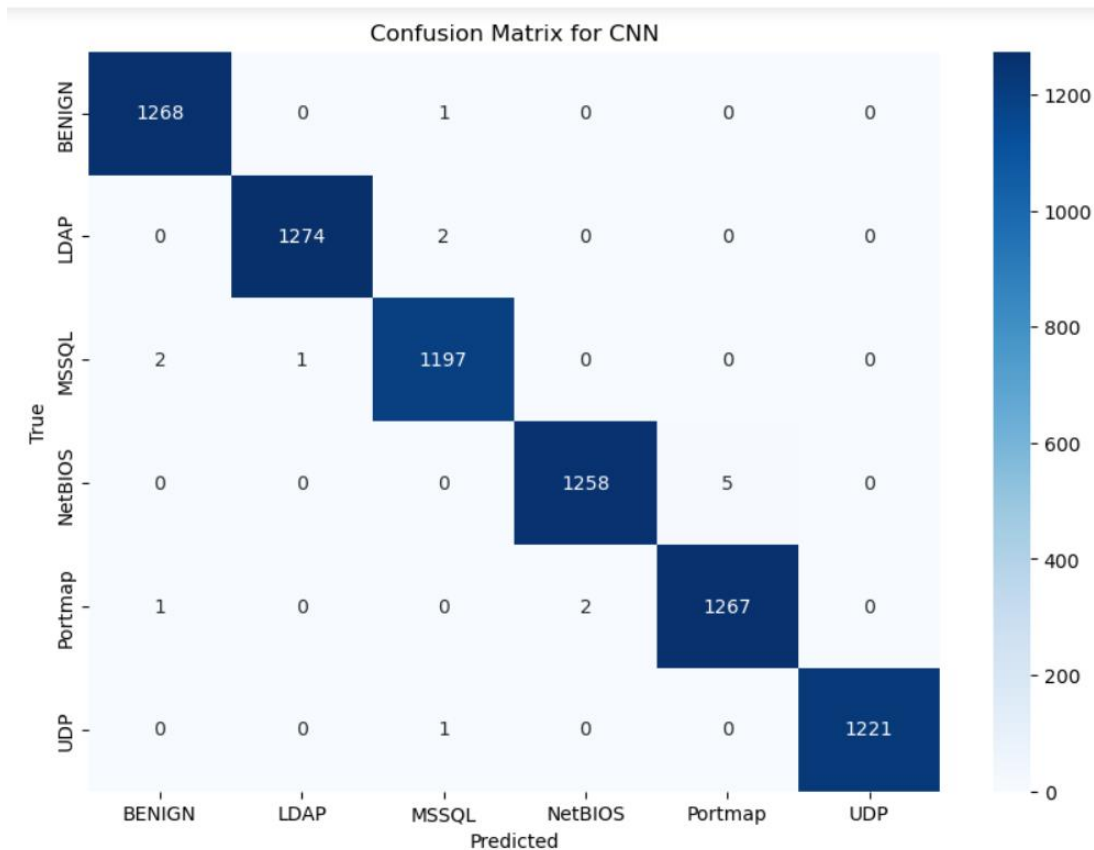


Figure 29: Confusion matrix for CNN

### 4.3.2 LSTM Model

The LSTM model is trained for 50 epochs. The training Accuracy consistently improves, starting at 78.42% in the first epoch and reaching around 99.54% by the 50th epoch. Training Loss decreases from 0.71 in the first epoch to around 0.0126 in the final epoch, indicating that the model is learning effectively. Validation accuracy starts high (98.24% in Epoch 1) and remains mostly above 99%, while validation loss fluctuates but generally decreases, indicating good generalization.

Epoch 1/50

**563/563** ————— **6s** 5ms/step - accuracy: 0.7842 - loss: 0.7124 - val\_accuracy: 0.9824 - val\_loss: 0.0621

Epoch 2/50

**563/563** ————— **2s** 4ms/step - accuracy: 0.9708 - loss: 0.0811 - val\_accuracy: 0.9713 - val\_loss: 0.0629

Epoch 3/50

**563/563** ————— **2s** 4ms/step - accuracy: 0.9796 - loss: 0.0557 - val\_accuracy: 0.9920 - val\_loss: 0.0280

Epoch 4/50

**563/563** ————— **3s** 4ms/step - accuracy: 0.9848 - loss: 0.0394 - val\_accuracy: 0.9927 - val\_loss: 0.0231

Epoch 5/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9869 - loss: 0.0348 - val\_accuracy: 0.9904 - val\_loss: 0.0261  
Epoch 6/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9897 - loss: 0.0282 - val\_accuracy: 0.9944 - val\_loss: 0.0166  
Epoch 7/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9884 - loss: 0.0311 - val\_accuracy: 0.9956 - val\_loss: 0.0148  
Epoch 8/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9888 - loss: 0.0282 - val\_accuracy: 0.9960 - val\_loss: 0.0142  
Epoch 9/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9891 - loss: 0.0249 - val\_accuracy: 0.9900 - val\_loss: 0.0213  
Epoch 10/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9899 - loss: 0.0243 - val\_accuracy: 0.9960 - val\_loss: 0.0128  
Epoch 11/50  
**563/563** \_\_\_\_\_ **3s** 4ms/step - accuracy:  
0.9913 - loss: 0.0234 - val\_accuracy: 0.9904 - val\_loss: 0.0238  
Epoch 12/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9888 - loss: 0.0288 - val\_accuracy: 0.9867 - val\_loss: 0.0293  
Epoch 13/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9896 - loss: 0.0272 - val\_accuracy: 0.9967 - val\_loss: 0.0121  
Epoch 14/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9911 - loss: 0.0240 - val\_accuracy: 0.9958 - val\_loss: 0.0126  
Epoch 15/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9921 - loss: 0.0194 - val\_accuracy: 0.9953 - val\_loss: 0.0139  
Epoch 16/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9942 - loss: 0.0153 - val\_accuracy: 0.9960 - val\_loss: 0.0117  
Epoch 17/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9935 - loss: 0.0157 - val\_accuracy: 0.9913 - val\_loss: 0.0187  
Epoch 18/50  
**563/563** \_\_\_\_\_ **3s** 4ms/step - accuracy:  
0.9933 - loss: 0.0170 - val\_accuracy: 0.9960 - val\_loss: 0.0118  
Epoch 19/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9911 - loss: 0.0225 - val\_accuracy: 0.9962 - val\_loss: 0.0125  
Epoch 20/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9931 - loss: 0.0158 - val\_accuracy: 0.9958 - val\_loss: 0.0131  
Epoch 21/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9923 - loss: 0.0223 - val\_accuracy: 0.9936 - val\_loss: 0.0144  
Epoch 22/50  
**563/563** \_\_\_\_\_ **3s** 3ms/step - accuracy:  
0.9937 - loss: 0.0159 - val\_accuracy: 0.9951 - val\_loss: 0.0132  
Epoch 23/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9917 - loss: 0.0231 - val\_accuracy: 0.9953 - val\_loss: 0.0138



Epoch 24/50  
**563/563** \_\_\_\_\_ **3s** 4ms/step - accuracy:  
0.9929 - loss: 0.0200 - val\_accuracy: 0.9867 - val\_loss: 0.0296  
Epoch 25/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9918 - loss: 0.0178 - val\_accuracy: 0.9956 - val\_loss: 0.0109  
Epoch 26/50  
**563/563** \_\_\_\_\_ **3s** 3ms/step - accuracy:  
0.9922 - loss: 0.0175 - val\_accuracy: 0.9940 - val\_loss: 0.0137  
Epoch 27/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9942 - loss: 0.0150 - val\_accuracy: 0.9960 - val\_loss: 0.0108  
Epoch 28/50  
**563/563** \_\_\_\_\_ **3s** 4ms/step - accuracy:  
0.9930 - loss: 0.0171 - val\_accuracy: 0.9938 - val\_loss: 0.0140  
Epoch 29/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9944 - loss: 0.0141 - val\_accuracy: 0.9933 - val\_loss: 0.0156  
Epoch 30/50  
**563/563** \_\_\_\_\_ **3s** 3ms/step - accuracy:  
0.9934 - loss: 0.0153 - val\_accuracy: 0.9951 - val\_loss: 0.0133  
Epoch 31/50  
**563/563** \_\_\_\_\_ **3s** 4ms/step - accuracy:  
0.9946 - loss: 0.0140 - val\_accuracy: 0.9851 - val\_loss: 0.0367  
Epoch 32/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9940 - loss: 0.0159 - val\_accuracy: 0.9969 - val\_loss: 0.0106  
Epoch 33/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9953 - loss: 0.0115 - val\_accuracy: 0.9944 - val\_loss: 0.0135  
Epoch 34/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9940 - loss: 0.0147 - val\_accuracy: 0.9936 - val\_loss: 0.0139  
Epoch 35/50  
**563/563** \_\_\_\_\_ **3s** 4ms/step - accuracy:  
0.9943 - loss: 0.0130 - val\_accuracy: 0.9938 - val\_loss: 0.0136  
Epoch 36/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9949 - loss: 0.0133 - val\_accuracy: 0.9978 - val\_loss: 0.0087  
Epoch 37/50  
**563/563** \_\_\_\_\_ **3s** 3ms/step - accuracy:  
0.9948 - loss: 0.0141 - val\_accuracy: 0.9956 - val\_loss: 0.0106  
Epoch 38/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9938 - loss: 0.0147 - val\_accuracy: 0.9889 - val\_loss: 0.0294  
Epoch 39/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9937 - loss: 0.0178 - val\_accuracy: 0.9953 - val\_loss: 0.0127  
Epoch 40/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9957 - loss: 0.0115 - val\_accuracy: 0.9962 - val\_loss: 0.0108  
Epoch 41/50  
**563/563** \_\_\_\_\_ **2s** 3ms/step - accuracy:  
0.9952 - loss: 0.0127 - val\_accuracy: 0.9967 - val\_loss: 0.0100  
Epoch 42/50  
**563/563** \_\_\_\_\_ **2s** 4ms/step - accuracy:  
0.9940 - loss: 0.0147 - val\_accuracy: 0.9958 - val\_loss: 0.0111

Epoch 43/50  
**563/563** ————— **2s** 3ms/step - accuracy:  
0.9949 - loss: 0.0123 - val\_accuracy: 0.9962 - val\_loss: 0.0110  
Epoch 44/50  
**563/563** ————— **3s** 4ms/step - accuracy:  
0.9955 - loss: 0.0110 - val\_accuracy: 0.9962 - val\_loss: 0.0107  
Epoch 45/50  
**563/563** ————— **2s** 4ms/step - accuracy:  
0.9960 - loss: 0.0115 - val\_accuracy: 0.9869 - val\_loss: 0.0330  
Epoch 46/50  
**563/563** ————— **2s** 4ms/step - accuracy:  
0.9928 - loss: 0.0171 - val\_accuracy: 0.9953 - val\_loss: 0.0112  
Epoch 47/50  
**563/563** ————— **3s** 4ms/step - accuracy:  
0.9948 - loss: 0.0117 - val\_accuracy: 0.9918 - val\_loss: 0.0189  
Epoch 48/50  
**563/563** ————— **3s** 4ms/step - accuracy:  
0.9943 - loss: 0.0140 - val\_accuracy: 0.9918 - val\_loss: 0.0185  
Epoch 49/50  
**563/563** ————— **3s** 3ms/step - accuracy:  
0.9937 - loss: 0.0168 - val\_accuracy: 0.9973 - val\_loss: 0.0086  
Epoch 50/50  
**563/563** ————— **3s** 4ms/step - accuracy:  
0.9954 - loss: 0.0126 - val\_accuracy: 0.9942 - val\_loss: 0.0146  
**235/235** ————— **1s** 3ms/step  
Accuracy: 0.99  
Precision: 0.99  
Recall: 0.99  
F1 Score: 0.99  
AUC: 1.00  
Training Time: 116.76 seconds  
Testing Time: 1.12 seconds

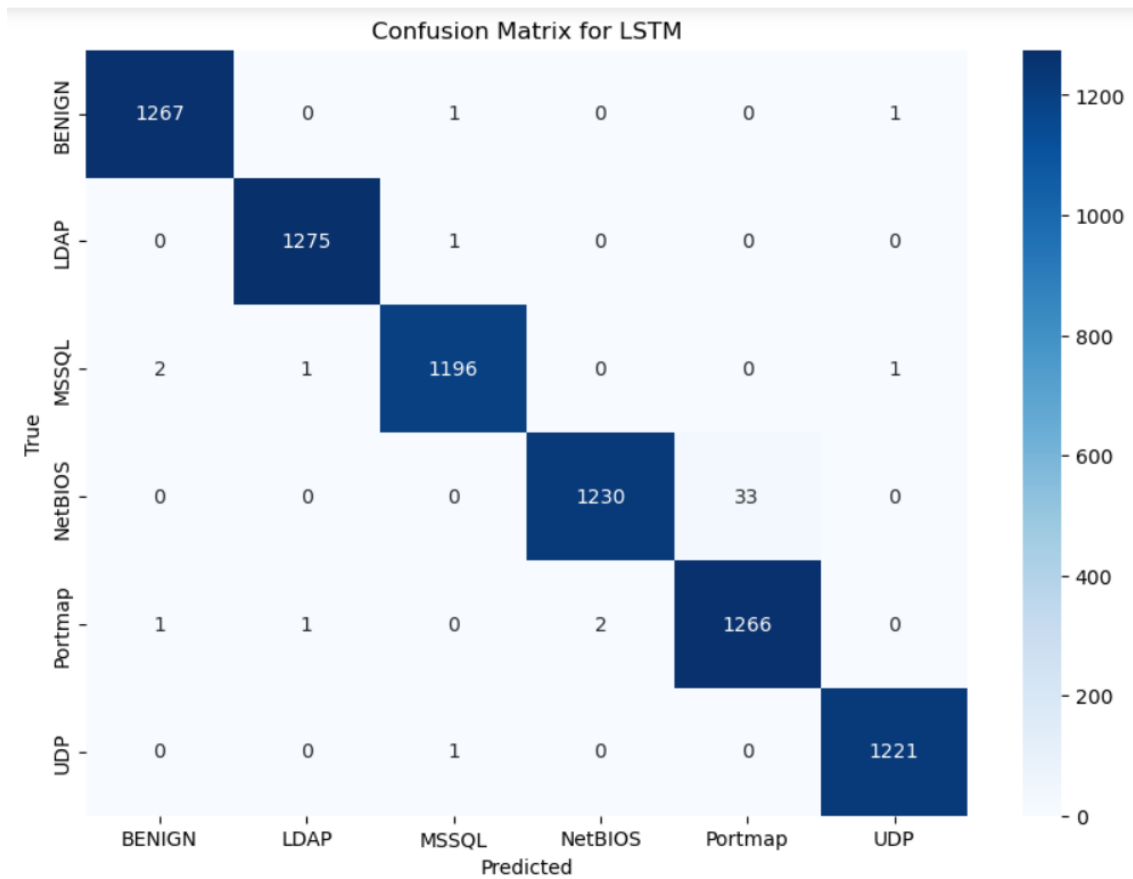


Figure 30: Confusion matrix for LSTM

## 4.4 Discussions

The Intrusion Detection system is designed using Machine learning models like Support Vector Model (SVM), Logistic Regression (LR), Random Forest (RF) and KNN models and Deep Learning models like CNN and LSTM.

The binary classification has more than 99% accuracy for the all the machine learning models for without feature selection and PCA methods. To find the accuracy of the model in detecting each type of attack multiclass classification is implemented using four machine learning and two deep learning models.

The results of multiclass classification without feature selection method shows that Random Forest has highest accuracy of 100% followed by the KNN method with 91% accuracy. It can be also observed that the Logistic Regress has 78% accuracy, but The SVM model completely fails for without feature selection method.

In case of PCA, the SVM model has highest accuracy of 85% compared to other models. In this method all the model's performance very well and has an accuracy of more than 79%. The LR has 79%, Random Forest has 83% and KNN has 84% of accuracy. It can

be observed from the Confusion matrix that there is incorrect prediction in case of Portmap and NetBIOS attack which reduced the accuracy of the model.

The SelectKBest method is implemented by choosing three values for  $K=10,20,30$ . For all the values of  $K$ , the Random Forest has approximately 100% accuracy. The KNN also performs well for this method and has accuracy more than 90%. The performance of Logistics Regression model improves as the number of features increases. The performance of SVM decreases with the increase of features.

The output of Machine learning model is also compared with the two deep learning models like CNN and LSTM. Both the deep learning model has a good accuracy. It can be observed that the Random Forest model performs well for this system.

## 5 CONCLUSIONS

Intrusion Detection system is a software used for detecting the DDoS attack on the network. This attack over floods the network with unwanted messages leading to denial of service to the legitimate users. There are many cases reported about damage caused by these attacks.

This project focuses on detection of the attack on the network. The pre-existing dataset is used to conduct the experiment. The dataset used is CIC-DDoS 2019 which is most recent dataset. The machine Learning models used to implement this detection mechanism are SVM, Logistic Regression, Random Forest and KNN. The performance of each model is calculated by using precision, accuracy, F1 score and Recall. The performance of all the models is compared for the selecting the better model.

Without feature selection and SelectKBest feature selection methods, the Random Forest model has greater accuracy. The SelectKBest method is implemented for three different values of  $K=10,20,30$ . For all the values of  $K$ , the Random Forest has approximately 100% accuracy. The KNN also performs well for this method and has accuracy more than 90%. The Logistics Regression model has better performance with more features than the fewer features. The performance of SVM decreases with the increase of features.

When the PCA dimension reduction method is used the SVM has 85% accuracy. Because of its large dimensionality and noisy features, SVM performs badly when feature selection is not used or when SelectKBest is used. On the other hand, by converting the data into a lower-dimensional space that preserves the most crucial information, dimensionality reduction techniques like PCA can greatly improve SVM's performance by lowering noise and raising classification accuracy. Because it streamlines the feature space and makes it easier to use, SVM typically performs better when used with PCA.

Logistic regression accuracy tends to improve with an increasing number of features because more features provide the model with additional information to better capture the relationships between the input data and the target variable. As logistic regression assumes a linear relationship, adding more informative features can help create a clearer separation between classes, improving the model's performance.

Future research might focus on enhancing feature selection strategies by investigating more complex methodologies that may improve model performance, particularly in addressing misclassifications for specific attacks such as Portmap and NetBIOS. There is also potential

for hybrid models that combine machine learning and deep learning methodologies to improve detection accuracy. Furthermore, future research might concentrate on lowering the computational cost and memory footprint of the models, ensuring that the system can be deployed efficiently in resource-constrained situations such as IoT devices or edge computing platforms.

## 6 REFERENCES

1. A. L. G. Rios, Z. Li, K. Bekshentayeva and L. Trajković, "Detection of Denial of Service Attacks in Communication Networks," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain.
2. N Ahuja, G Singal, Debajyoti Mukhopadhyay, Neeraj Kumar, 2021, Automated DDOS attack detection in software defined networking, Journal of Network and Computer Applications, Volume 187.
3. S. Aktar, A.Y.Nur, 2023, Towards DDoS attack detection using deep learning approach. Computers & Security, Volume 129.
4. J. Li, "Network Intrusion Detection Algorithm and Simulation of Complex System in Internet Environment," 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India.
5. F.S.d.L. Filho, F. A. F. Silveira, A. dM. B. Junior, G. V.Solar, L. F. Silveira, "Smart Detection: An Online Approach for DoS/DDoS Attack Detection Using Machine Learning", Security and Communication Networks, vol. 2019.
6. What is a DDoS Attack? DDoS Meaning, Definition & Types | Fortinet (no date). <https://www.fortinet.com/resources/cyberglossary/ddos-attack>
7. M. Thilakraj, S. Anupriya, M. M. Cibi and A. Divya, "Detection of SQL Injection Attacks," 2024 International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2024
8. S. Mekala and K. B. Dasari, "NetBIOS DDoS Attacks Detection with Machine Learning Classification Algorithms," 2023 International Conference on Advancement in Computation & Computer Technologies (InCACCT), Gharuan, India, 2023
9. S. Srinivasa, J. M. Pedersen and E. Vasilomanolakis, "Deceptive directories and “vulnerable” logs: a honeypot study of the LDAP and log4j attack landscape," 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Genoa, Italy, 2022
10. Li, Junhong. "Detection of DDoS attacks based on dense neural networks, autoencoders and Pearson correlation coefficient." (2020).
11. Kousar, Heena, Mohammed Moin Mulla, Pooja Shettar, and D. G. Narayan. "Detection of DDoS attacks in software defined network using decision tree." In 2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)

12. D. Kishore, N. Devarakonda "Detection of DDoS Attacks Using Machine Learning Classification Algorithms", International Journal of Computer Network and Information Security, 2022
13. Omer Aslan "Using Machine Learning Techniques to Detect Attacks in Computer Networks", Aegean Summit 4th International Applied Sciences Congress, 2022, Mugla, Turkey
14. M. Jawad, G. H. Majeed, "Enhancing Multi-Class DDoS Attack Classification using Machine Learning Techniques", Journal of Advanced Research in Applied Sciences and Engineering Technology
15. K. N. Rajapraveen. and R. Pasumarty, "A Machine Learning Approach for DDoS Prevention System in Cloud Computing Environment," 2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), Bangalore, India, 2021
16. Ebtihal Sameer Alghoson, Onytra Abbass, "Detecting Distributed Denial of Service Attacks using Machine Learning Models", International Journal of Advanced Computer Science and Applications, Vol. 12, No. 12, 2021
17. K. M. M. Uddin, A. Al Mamun, Aa Chakrabarti, R. Mostafiz, S. K. Dey, "An ensemble machine learning-based approach to predict cervical cancer using hybrid feature selection", Neuroscience Informatics, Volume 4, Issue 3, 2024
18. Ji, R., Kumar, N. and Padha, D., "Hybrid Enhanced Intrusion Detection Frameworks for Cyber-Physical Systems via Optimal Features Selection", Indian Journal of Science and Technology, 2024
19. S. Waskle, L. Parashar and U. Singh, "Intrusion Detection System Using PCA with Random Forest Approach," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2020
20. Devrim Akgun, Selman Hizal, Unal Cavusoglu, "A new DDoS attacks intrusion detection model based on deep learning for cybersecurity", Computers & Security, 2022
21. T. T. Nguyen, C. S. Shieh, C. H. Chen, D. Miu, "Detection of Unknown DDoS Attacks with Deep Learning and Gaussian Mixture Model", Conference: 2021 4th International Conference on Information and Computer Technologies (ICICT), March 2021












22. Y. Wei, J. Jang-Jaccard, F. Sabrina, A. Singh, W. Xu and S. Camtepe, "AE-MLP: A Hybrid Deep Learning Approach for DDoS Detection and Classification," in IEEE Access
23. <https://www.unb.ca/cic/datasets/ids-2017.html>
24. <https://www.geeksforgeeks.org>
25. <https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial>
26. What Is Random Forest? | IBM
27. [https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
28. <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
29. Rahul Kala, "In Emerging Methodologies and Applications in Modelling, Autonomous Mobile Robots", Academic Press, 2024
30. Goodfellow, I., Bengio, Y., Courville, A. "Deep Learning". MIT Press. (Chapter on Recurrent Neural Networks, including LSTMs)
31. S. Khalid, T. Khalil and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," 2014 Science and Information Conference, London, UK
32. I. Sharafaldin, A. H. Lashkari, S. Hakak and A. A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," 2019 International Carnahan Conference on Security Technology (ICCST), Chennai, India, 2019.

## 7. APPENDICES

### Appendix A

#### A.1 Gantt Chart

The timeline of the project is shown in the Gantt Chart below. The major milestones of the project are highlighted. The project is divided into different stages and timeline for achieving each step is shown below. The dataset used for the project is CIC-DDoS 2019. The AI models like Random forest, Logistic Regression, SVM, KNN, CNN and LSTM is used to implement this idea.

Task	Timeline in weeks												
	Description	1	2	3	4	5	6	7	8	9	10	11	12
1	Literature review on ideas, data and AI model												
2	Analysis of existing dataset												
3	Creation of Train and test datasets from existing dataset												
4	Implementation of Random forest model, SVM and Logistic Regression for the detection												
5	Train and test datasets on AI model												
6	Evaluate IDS performance of the model in recognising the attacks												
7	Implementation of KNN model for the detection and evaluate IDS performance of the model in recognising the attacks												
8	Implement all the models without feature selection and SelectKBest feature selection method												
9	Implement CNN and LSTM models												



<div>0 – 5 = Low Risk</div> <div>6 – 10 = Moderate Risk</div> <div>11 – 15 = High Risk</div> <div>16 – 25 = extremely high unacceptable risk</div>		Severity of the potential injury/damage				
		Insignificant damage to Property, Equipment or Minor Injury	Non-Reportable Injury, minor loss of Process or slight damage to Property	Reportable Injury moderate loss of Process or limited damage to Property	Major Injury, Single Fatality critical loss of Process/damage to Property	Multiple Fatalities Catastrophic Loss of Business
		1	2	3	4	5
Likelihood of the hazard happening	Almost Certain 5	5	10	15	20	25
	Will probably occur 4	4	8	12	16	20
	Possible occur 3	3	6	9	12	15
	Remote possibility 2	2	4	6	8	10
	Extremely Unlikely 1	1	2	3	4	5

## APPENDIX B

### B.1 The 88 Features of the dataset

Features	Description
Flow ID	Flow ID
Source IP	Source IP address
Source Port	Source Port Number
Destination IP	Destination IP address
Destination Port	Destination Port Number
Protocol	Protocol used
Timestamp	Timestamp
Flow duration	Flow duration
Total Fwd Packets	Total packets in the forward direction
Total Backward Packets	Total packets in the backward direction
Total length of Fwd Packets	The total size of packets in the forward direction
Total length of Bwd packets	The total size of packets in the backward direction
Fwd Packet Length Max	Maximum size of packets in the forward direction
Fwd packet length Min	The minimum size of packets in the forward direction
Fwd Packet Lenth Mean	The average size of packets in the forward direction

Fwd Packet Length Std	Standard deviation size of packets in the forward direction
Bwd Packet Length Max	Maximum size of packets in the backward direction
Bwd Packet Length Min	The minimum size of packets in the backward direction
Bwd Packet Length Mean	Mean size of packets in the backward direction
Bwd Packet Length Std	Standard deviation size of packets in the backward direction
Flow Bytes/s	flow byte rate that is the number of packets transferred per second
Flow Packets/s	flow packets rate that is the number of packets transferred per second
Flow IAT Mean	The average time between the two flows
Flow IAT Std	Standard deviation time two flows
Flow IAT Max	Maximum time between two flows
Flow IAT Min	Minimum time between two flows
Fwd IAT Total	Total time between two packets sent in the forward direction
Fwd IAT Mean	The mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction

Fwd IAT Min	Minimum time between two packets sent in the forward direction
Bwd IAT Total	Total time between two packets sent in the backward direction
Bwd IAT Mean	The mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH ag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH ag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG ag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG ag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Length	Total bytes used for headers in the forward direction
Bwd Header Length	Total bytes used for headers in the forward direction
Fwd Packets/s	Number of forwarding packets per second

Bwd Packets/s	Number of backward packets per second
Min Packet Length	Minimum length of a flow
Max Packet Length	The maximum length of a flow
Packet Length Mean	Mean length of a flow
Packet Length Std	Standard deviation length of a flow
Packet Length Variance	Minimum inter-arrival time of packet
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Count	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Average Packet size	The average size of packets
Avg Fwd Segment Size	Average size observed in the forward direction
Avg Bwd Segment Size	Average size observed in the backward direction
Fwd Header Length	The average number of bytes bulk rate in the forward direction
Fwd Avg Bytes/Bulk	The average number of packets bulk rate in the forward direction



Fwd Avg Packets/Bulk	The average number of packets bulk rate in the forward direction
Fwd Avg Bulk Rate	The average number of bulk rates in the forward direction
Bwd Avg Bytes/Bulk	The average number of bytes bulk rate in the backward direction
Bwd Avg Packets/Bulk	The average number of bulk rate in the forward direction
Bwd Avg Bulk Rate	The average number of bytes bulk rate in the backward direction
Subflow Fwd Packets	The average number of packets in a sub-ow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub-ow in the forward direction
Subflow Bwd Packets	The average number of packets in a sub-ow in the backward direction
Subflow Bwd Bytes	The average number of bytes in a sub-ow in the backward direction
Init_Win_bytes_forward	Number of bytes sent in the initial window in the forward direction
Init_Win_bytes_backward	The number of bytes sent in the initial window in the backward direction
Act_data_pkt_fwd	The number of packets with at least 1 byte of TCP data payload in the forward direction
Min_seg_size_forward	The number of packets with at least 1 byte of TCP data payload in the forward direction

Active Mean	The mean time a ow was active before becoming idle
Active Std	Standard deviation time a ow was active before becoming idle
Active Max	The maximum time a ow was active before becoming idle
Active Min	The minimum time a ow was active before becoming idle
Idle Mean	Meantime a ow was idle before becoming active
Idle std	Standard deviation time a ow was idle before becoming active
Idle Max	The maximum time a ow was idle before becoming active
Idle Min	The minimum time a ow was idle before becoming active
SimilarHTTP	HTTP
Inbound	inbound data transfer
Label	Attack Type

## APPENDIX C

### C.1 Code for Intrusion Detection System

#### **#Import necessary library**

```
import numpy as np
import pandas as pd
import csv
import seaborn as sns
import math
import missingno as msno
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
from sklearn.feature_selection import RFE
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold
import time
from imblearn.over_sampling import SMOTE
from sklearn.impute import SimpleImputer
from sklearn.decomposition import IncrementalPCA
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_score
```

#### **# Read Dataset**

```

df1 = pd.read_csv("UDP.csv",low_memory=False) #read the csv file in df
df2 = pd.read_csv("LDAP.csv",low_memory=False) #read the csv file in df
df3 = pd.read_csv("Portmap.csv",low_memory=False) #read the csv file in df
data_list = [df1, df2, df3]

print('Data dimensions: ')

for i, data in enumerate(data_list, start = 1):

    rows, cols = data.shape

    print(f'Data{i} -> {rows} rows, {cols} columns')

df = pd.concat(data_list)

rows, cols = df.shape

print('New dimension:')

print(f'Number of rows: {rows}')

print(f'Number of columns: {cols}')

print(f'Total cells: {rows * cols}')

```

#### **# Deleting dataframes after concating to save memory**

```

for d in data_list: del d

```

#### **# Renaming the columns by removing leading/trailing whitespace**

```

col_names = {col: col.strip() for col in df.columns}

df.rename(columns = col_names, inplace = True)

rows, cols = df.shape # rows and column count

print('New dimension:')

print(f'Number of rows: {rows}')

print(f'Number of columns: {cols}')

print(f'Total cells: {rows * cols}')

```

#### **# information about the datatypes**

```

df.info()

```

#### **#finding missing value**

```

df.isnull().sum()

```

**#finding percentage of missing value**

```
df.isnull().sum()/df.shape[0]*100
```

**#finding duplicate value**

```
df.duplicated().sum()
```

**# dropping all duplicate rows**

```
df = df.drop_duplicates()
```

```
df
```

**#Identify garbage value (if there is garbage value it is in form of object)**

```
for i in df.select_dtypes(include="object").columns:
```

```
    print(df[i].value_counts())
```

```
    print("****"*10)
```

**# Checking for infinity values**

**# Select numeric columns**

```
numeric_cols = df.select_dtypes(include = np.number).columns
```

**# Check for infinity values and count them**

```
inf_count = np.isinf(df[numeric_cols]).sum()
```

```
print(inf_count[inf_count > 0])
```

**# Replacing any infinite values (positive or negative) with NaN (not a number)**

```
print(f'Initial missing values: {df.isna().sum().sum()}')
```

```
df.replace([np.inf, -np.inf], np.nan, inplace = True)
```

```
print(f'Missing values after processing infinite values: {df.isna().sum().sum()}')
```

```
missing = df.isna().sum()
```

```
print(missing.loc[missing > 0])
```

### **# Calculating missing value percentage in the dataset**

```
mis_per = (missing / len(df)) * 100  
mis_table = pd.concat([missing, mis_per.round(2)], axis = 1)  
mis_table = mis_table.rename(columns = {0 : 'Missing Values', 1 : 'Percentage of Total  
Values'})  
print(mis_table.loc[mis_per > 0])
```

### **#Plot the missing value**

```
sns.set_palette('pastel')  
colors = sns.color_palette()  
missing_vals = [col for col in df.columns if df[col].isna().any()]  
fig, ax = plt.subplots(figsize = (2, 6))  
msno.bar(df[missing_vals], ax = ax, fontsize = 12, color = colors)  
ax.set_xlabel('Features', fontsize = 12)  
ax.set_ylabel('Non-Null Value Count', fontsize = 12)  
ax.set_title('Missing Value Chart', fontsize = 12)  
plt.show()
```

### **#Find the median of the Flow Bytes/s and Flow Packets/s**

```
med_flow_bytes = df['Flow Bytes/s'].median()  
med_flow_packets = df['Flow Packets/s'].median()  
print('Median of Flow Bytes/s: ', med_flow_bytes)  
print('Median of Flow Packets/s: ', med_flow_packets)
```

### **# Filling missing values with median**

```
df['Flow Bytes/s'].fillna(med_flow_bytes, inplace = True)  
df['Flow Packets/s'].fillna(med_flow_packets, inplace = True)  
print('Number of \'Flow Bytes/s\' missing values:', df['Flow Bytes/s'].isna().sum())  
print('Number of \'Flow Packets/s\' missing values:', df['Flow Packets/s'].isna().sum())
```

**# Analysing Patterns using Visualisations**

**#find the label(attack types)**

```
df['Label'].unique()
```

**# number of attacks in each type**

```
df['Label'].value_counts()
```

**# Creating a dictionary that maps each label to its attack type**

```
attack_map = {'UDP': 'UDP', 'MSSQL': 'MSSQL', 'BENIGN': 'BENIGN', 'NetBIOS': 'NetBIOS', 'LDAP': 'LDAP', 'Portmap': 'Portmap' }
```

**# Creating a new column 'Attack Type' in the DataFrame based on the attack\_map dictionary**

```
df['Attack Type'] = df['Label'].map(attack_map)
```

**# Create balanced dataset for binary classification**

```
normal_traffic = df[df['Attack Type'] == 'BENIGN']
```

```
intrusions = df[df['Attack Type'] != 'BENIGN']
```

**# Determine the number of samples needed**

```
sample_size = min(len(normal_traffic), len(intrusions))
```

**# Sample from both datasets to match the smaller size**

```
normal_traffic = normal_traffic.sample(n=sample_size, replace=False, random_state=42)
```

```
intrusions = intrusions.sample(n=sample_size, replace=False, random_state=42)
```

**# Combine the datasets and create binary labels**

```
ids_data = pd.concat([intrusions, normal_traffic])
```

```
ids_data['Attack Type'] = np.where(ids_data['Attack Type'] == 'BENIGN', 0, 1)
```

**# Adjust the sample size based on available data**

```

target_sample_size = min(15000, len(ids_data))
if target_sample_size > 0:
    bc_data = ids_data.sample(n=target_sample_size, random_state=42)
    print(bc_data['Attack Type'].value_counts())
else:
    print("Insufficient data even after balancing.")

```

### **#Identify non-numeric columns**

```

non_numeric_columns = bc_data.select_dtypes(include=['object']).columns

```

### **# Convert non-numeric data to numeric using LabelEncoder**

```

le = LabelEncoder()
for col in non_numeric_columns:
    bc_data[col] = le.fit_transform(bc_data[col])

```

### **# Split the data into features (X) and target (y)**

```

X = bc_data.drop('Attack Type', axis=1)
y = bc_data['Attack Type']

```

### **# Split the data into training and test sets**

```

X_train_bc, X_test_bc, y_train_bc, y_test_bc = train_test_split(X, y, test_size=0.25,
random_state=0)

```

### **# Displaying the shapes of the resulting datasets**

```

print(f'X_train_bc shape: {X_train_bc.shape}')
print(f'X_test_bc shape: {X_test_bc.shape}')
print(f'y_train_bc shape: {y_train_bc.shape}')
print(f'y_test_bc shape: {y_test_bc.shape}')

```

### **# Scale numeric data**



```

ss = StandardScaler()

X_train_bc = ss.fit_transform(X_train_bc) # Fit on train data
X_test_bc = ss.transform(X_test_bc)      # Transform test data using the same scaler

#Binary classification without feature selection

# Define models

models = {
    'Logistic Regression': LogisticRegression(),
    'Support Vector Machines': LinearSVC(),
    'Random Forest': RandomForestClassifier(),
    'K-Nearest Neighbor': KNeighborsClassifier()
}

# Dictionaries to store metrics

accuracy, precision, recall, f1_scores= {}, {}, {}, {}

# Fit models and calculate metrics

for key in models.keys():

    # Fit the classifier

    models[key].fit(X_train_bc, y_train_bc)


    # Make predictions

    predictions = models[key].predict(X_test_bc)


    # Calculate metrics

    accuracy[key] = accuracy_score(y_test_bc, predictions)
    precision[key] = precision_score(y_test_bc, predictions)
    recall[key] = recall_score(y_test_bc, predictions)
    f1_scores[key] = f1_score(y_test_bc, predictions)


# Create a DataFrame to display metrics

df_model = pd.DataFrame(index=models.keys(), columns=['Accuracy', 'Precision',
'Recall'])

```

```

df_model['Accuracy'] = accuracy.values()
df_model['Precision'] = precision.values()
df_model['Recall'] = recall.values()
df_model['f1 Score'] = f1_scores.values()
print(df_model)

```

### **# Plot confusion matrices**

```

for key in models.keys():

```

#### **# Make predictions**

```

    predictions = models[key].predict(X_test_bc)

```

#### **# Compute confusion matrix**

```

    cm = confusion_matrix(y_test_bc, predictions)

```

#### **# Plot confusion matrix**

```

    plt.figure(figsize=(8, 6))

```

```

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'],
yticklabels=['Class 0', 'Class 1'])

```

```

    plt.title(f'Confusion Matrix for {key}')

```

```

    plt.xlabel('Predicted Label')

```

```

    plt.ylabel('True Label')

```

```

    plt.show()

```

### **#Multiclass classification**

```

df.drop('Label', axis = 1, inplace = True)

```

#### **#Balance the data and randomly select 5000 samples for each type of attack**

```

class_counts = df['Attack Type'].value_counts()

```

```

selected_classes = class_counts[class_counts > 1500]

```

```

class_names = selected_classes.index

```

```

selected = df[df['Attack Type'].isin(class_names)]

```

```

dfs = []

```

```

for name in class_names:

    df = selected[selected['Attack Type'] == name]

    if len(df) > 5000:

        df = df.sample(n=5000, replace=False, random_state=0)

    dfs.append(df)

df = pd.concat(dfs, ignore_index=True)

df['Attack Type'].value_counts()


# Drop columns that are not numeric or that are not useful for the model

non_numeric_columns = df.select_dtypes(include=['object']).columns

# Convert categorical features to numeric using Label Encoding

feature_encoders = { }

for column in non_numeric_columns:

    if column != 'Attack Type': # Exclude the target column

        le = LabelEncoder()

        df[column] = le.fit_transform(df[column])

        feature_encoders[column] = le


# Encode the target variable

attack_label_encoder = LabelEncoder()

df['Attack Type_encoded'] = attack_label_encoder.fit_transform(df['Attack Type'])

class_labels = attack_label_encoder.classes_ # Get the class labels


# Define features and target

X = df.drop('Attack Type', axis=1)

y = df['Attack Type']


# Apply SMOTE to upsample the minority classes

smote = SMOTE(sampling_strategy='auto', random_state=0)

X_upsampled, y_upsampled = smote.fit_resample(X, y)

```

**# Create a new DataFrame with the upsampled data**

```
blnc_data = pd.DataFrame(X_upsampled, columns=X.columns)
blnc_data['Attack Type'] = y_upsampled
blnc_data = blnc_data.sample(frac=1, random_state=0) # Shuffle the DataFrame
```

**# Check the class distribution to verify balance**

```
print(blnc_data['Attack Type'].value_counts())
features = blnc_data.drop('Attack Type', axis = 1)
labels = blnc_data['Attack Type']
```

**#Split the dataset into training and test set**

```
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size = 0.25,
random_state = 0)
```

**# without Feature Selection**

**# Define models**

```
models = {
    'Logistic Regression': LogisticRegression(max_iter=10000, random_state=0),
    'SVM with RBF kernel': SVC(kernel='rbf', C=1, gamma=0.1, random_state=0,
probability=True),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=0),
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5)
}
```

**# Function to evaluate models**

```
def evaluate_model_no_fs(model, model_name, X_train, X_test, y_train, y_test):
    # Start timer for training
    start_train_time = time.time()

    # Cross-validation scores
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
    scores = cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
```

```

# Fit the model on the entire training data

model.fit(X_train, y_train)

# End timer for training

end_train_time = time.time()

train_time = end_train_time - start_train_time

# Start timer for testing

start_test_time = time.time()


# Predict on the test set

y_pred = model.predict(X_test)

y_pred_proba = model.predict_proba(X_test) if hasattr(model, 'predict_proba') else
None

auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr') if y_pred_proba is not
None else "N/A"


# End timer for testing

end_test_time = time.time()

test_time = end_test_time - start_test_time


# Calculate metrics

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred, average='weighted')

recall = recall_score(y_test, y_pred, average='weighted')

f1 = f1_score(y_test, y_pred, average='weighted')


# Print results

print(f"--- {model_name} ---")

print(f"Cross-validation scores: {scores.tolist()}")

print(f"Mean cross-validation score: {scores.mean():.2f}")

print(f"Accuracy: {accuracy:.2f}")

```

```

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"AUC: {auc:.2f}")
print(f"Training Time: {train_time:.2f} seconds")
print(f"Testing Time: {test_time:.2f} seconds")

```

### **# Plot confusion matrix with original labels**

```

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
            yticklabels=class_labels)

plt.title(f'Confusion Matrix for {model_name}')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

```

### **# Evaluate each model**

```

for name, model in models.items():
    print(f"Evaluating {name}...")
    evaluate_model_no_fs(model, name, X_train, X_test, y_train, y_test)

```

### **# Feature Selection -SelectKBest= 10**

```

from sklearn.feature_selection import SelectKBest, f_classif

# Apply SelectKBest to select top 10 features using f_classif

k_best = SelectKBest(score_func=f_classif, k=10)
X_train_kbest = k_best.fit_transform(X_train, y_train)
X_test_kbest = k_best.transform(X_test)

```

### **# Define cross-validation**

```

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)

```

### **# Define models**

```
models = {  
    'Logistic Regression': LogisticRegression(max_iter=10000, random_state=0),  
    'SVM with RBF kernel': SVC(kernel='rbf', C=1, gamma=0.1, random_state=0,  
    probability=True),  
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=0),  
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5)  
}
```

### **# Function to evaluate models with SelectKBest feature selection**

```
def evaluate_model_selectkbest(model, model_name, X_train, X_test, y_train, y_test):
```

```
    # Start timer for training
```

```
    start_train_time = time.time()
```

#### **# Cross-validation scores**

```
scores = cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
```

#### **# Fit the model on the entire training data**

```
model.fit(X_train, y_train)
```

#### **# End timer for training**

```
end_train_time = time.time()
```

```
train_time = end_train_time - start_train_time
```

#### **# Start timer for testing**

```
start_test_time = time.time()
```

#### **# Predict on the test set**

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

```
y_pred_proba = model.predict_proba(X_test) if hasattr(model, 'predict_proba') else  
None
```

```
auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr') if y_pred_proba is not
None else "N/A"
```

### **# Calculate metrics**

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

### **# End timer for testing**

```
end_test_time = time.time()
test_time = end_test_time - start_test_time
```

### **# Print results**

```
print(f"--- {model_name} with SelectKBest ---")
print(f"Cross-validation scores: {scores.tolist()}")
print(f"Mean cross-validation score: {scores.mean():.2f}")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"AUC: {auc:.2f}")
print(f"Training Time: {train_time:.2f} seconds")
print(f"Testing Time: {test_time:.2f} seconds")
```

### **# Plot confusion matrix**

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
yticklabels=class_labels)

plt.title(f'Confusion Matrix for {type(model).__name__}')
```



```
plt.xlabel('Predicted')
```

```
plt.ylabel('True')
```

```
plt.show()
```

### **# Evaluate each model with SelectKBest feature selection**

```
for name, model in models.items():
```

```
    print(f'Evaluating {name}...')
```

```
    evaluate_model_selectkbest(model, name, X_train_kbest, X_test_kbest, y_train,
y_test)
```

### **# Feature Selection -SelectKBest= 20**

```
from sklearn.feature_selection import SelectKBest, f_classif
```

### **# Apply SelectKBest to select top 20 features using f\_classif**

```
k_best = SelectKBest(score_func=f_classif, k=20)
```

```
X_train_kbest = k_best.fit_transform(X_train, y_train)
```

```
X_test_kbest = k_best.transform(X_test)
```

### **# Define cross-validation**

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

### **# Define models**

```
models = {
```

```
    'Logistic Regression': LogisticRegression(max_iter=10000, random_state=0),
```

```
    'SVM with RBF kernel': SVC(kernel='rbf', C=1, gamma=0.1, random_state=0,
probability=True),
```

```
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=0),
```

```
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5)
```

```
}
```

### **# Function to evaluate models with SelectKBest feature selection**

```
def evaluate_model_selectkbest(model, model_name, X_train, X_test, y_train, y_test):
```

```
    # Start timer for training
```

```

start_train_time = time.time()

# Cross-validation scores

scores = cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')

# Fit the model on the entire training data

model.fit(X_train, y_train)

# End timer for training

end_train_time = time.time()

train_time = end_train_time - start_train_time

# Start timer for testing

start_test_time = time.time()

# Predict on the test set

y_pred = model.predict(X_test)

y_pred_proba = model.predict_proba(X_test) if hasattr(model, 'predict_proba') else
None

auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr') if y_pred_proba is not
None else "N/A"


# Calculate metrics

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred, average='weighted')

recall = recall_score(y_test, y_pred, average='weighted')

f1 = f1_score(y_test, y_pred, average='weighted')


# End timer for testing

end_test_time = time.time()

test_time = end_test_time - start_test_time

# Print results

print(f"--- {model_name} with SelectKBest ---")

print(f"Cross-validation scores: {scores.tolist()}")

print(f"Mean cross-validation score: {scores.mean():.2f}")

print(f"Accuracy: {accuracy:.2f}")

```

```

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"AUC: {auc:.2f}")
print(f"Training Time: {train_time:.2f} seconds")
print(f"Testing Time: {test_time:.2f} seconds")

```

### **# Plot confusion matrix**

```

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
            yticklabels=class_labels)

plt.title(f'Confusion Matrix for {type(model).__name__}')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

```

### **# Evaluate each model with SelectKBest feature selection**

```

for name, model in models.items():

    print(f"Evaluating {name}...")

    evaluate_model_selectkbest(model, name, X_train_kbest, X_test_kbest, y_train,
                              y_test)

```

### **# Feature Selection -SelectKBest= 30**

```

from sklearn.feature_selection import SelectKBest, f_classif

```

### **# Apply SelectKBest to select top 30 features using f\_classif**

```

k_best = SelectKBest(score_func=f_classif, k=30)

X_train_kbest = k_best.fit_transform(X_train, y_train)

X_test_kbest = k_best.transform(X_test)

```

### **# Define cross-validation**

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

### **# Define models**

```
models = {  
    'Logistic Regression': LogisticRegression(max_iter=10000, random_state=0),  
    'SVM with RBF kernel': SVC(kernel='rbf', C=1, gamma=0.1, random_state=0,  
    probability=True),  
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=0),  
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5)  
}
```

### **# Function to evaluate models with SelectKBest feature selection**

```
def evaluate_model_selectkbest(model, model_name, X_train, X_test, y_train, y_test):
```

#### **# Start timer for training**

```
start_train_time = time.time()
```

#### **# Cross-validation scores**

```
scores = cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
```

#### **# Fit the model on the entire training data**

```
model.fit(X_train, y_train)
```

#### **# End timer for training**

```
end_train_time = time.time()
```

```
train_time = end_train_time - start_train_time
```

#### **# Start timer for testing**

```
start_test_time = time.time()
```

#### **# Predict on the test set**

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

```
y_pred_proba = model.predict_proba(X_test) if hasattr(model, 'predict_proba') else  
None
```

```
auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr') if y_pred_proba is not  
None else "N/A"
```

### **# Calculate metrics**

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

### **# End timer for testing**

```
end_test_time = time.time()
test_time = end_test_time - start_test_time
```

### **# Print results**

```
print(f"--- {model_name} with SelectKBest ---")
print(f"Cross-validation scores: {scores.tolist()}")
print(f"Mean cross-validation score: {scores.mean():.2f}")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"AUC: {auc:.2f}")
print(f"Training Time: {train_time:.2f} seconds")
print(f"Testing Time: {test_time:.2f} seconds")
```

### **# Plot confusion matrix**

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
yticklabels=class_labels)

plt.title(f'Confusion Matrix for {type(model).__name__}')
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
plt.show()
```

```
# Evaluate each model with SelectKBest feature selection
```

```
for name, model in models.items():
```

```
    print(f'Evaluating {name}...')
```

```
    evaluate_model_selectkbest(model, name, X_train_kbest, X_test_kbest, y_train,  
y_test)
```

## **#Principal Component Analysis**

```
# df is the DataFrame with features and 'Attack Type'
```

```
# Separate features and target
```

```
features = df.drop('Attack Type', axis=1)
```

```
attacks = df['Attack Type']
```

```
# Step 1: Handle non-numeric values (replace '-' with NaN)
```

```
features = features.replace('-', np.nan)
```

```
# Step 2: Identify numeric columns properly
```

```
numeric_columns = features.select_dtypes(include=[np.number]).columns
```

```
# Step 3: Convert to float, handling errors='coerce' to handle NaNs
```

```
features[numeric_columns] = features[numeric_columns].apply(pd.to_numeric,  
errors='coerce')
```

```
# Step 4: Impute missing values
```

```
imputer = SimpleImputer(strategy='mean')
```

```
features_imputed = imputer.fit_transform(features[numeric_columns])
```

```
# Verify there are no NaN values after imputation
```

```
assert not np.isnan(features_imputed).any(), "There are still NaN values after  
imputation."
```

### **# Step 5: Standardize the data**

```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features_imputed)
```

### **# Apply IncrementalPCA**

```
size = len(numeric_columns) // 2
ipca = IncrementalPCA(n_components=size, batch_size=500)
for batch in np.array_split(scaled_features, len(scaled_features) // 500):
    ipca.partial_fit(batch)
print(f'Information retained: {sum(ipca.explained_variance_ratio_):.2%}')
```

### **# If needed, transform the data using the fitted IPCA**

```
transformed_features = ipca.transform(scaled_features)
```

### **# Optionally, convert transformed features back to a DataFrame and combine with the target variable**

```
transformed_features_df = pd.DataFrame(transformed_features, columns=[f'PC{i+1}'
for i in range(size)])
final_df = pd.concat([transformed_features_df, attacks.reset_index(drop=True)],
axis=1)
```

### **# Display the final DataFrame**

```
print(final_df.head())
transformed_features = ipca.transform(scaled_features)
new_data = pd.DataFrame(transformed_features, columns = [f'PC{i+1}' for i in
range(size)])
new_data['Attack Type'] = attacks.values
```

### **# Assuming new\_data is DataFrame with principal components and 'Attack Type'**

### **# Create balanced dataset for binary classification**

```
normal_traffic = new_data[new_data['Attack Type'] == 'BENIGN']
intrusions = new_data[new_data['Attack Type'] != 'BENIGN']
```

### **# Determine the number of samples needed**

```

sample_size = min(len(normal_traffic), len(intrusions))

# Sample from both datasets to match the smaller size

normal_traffic = normal_traffic.sample(n=sample_size, replace=False,
random_state=42)

intrusions = intrusions.sample(n=sample_size, replace=False, random_state=42)

# Combine the datasets and create binary labels

ids_data = pd.concat([intrusions, normal_traffic])

ids_data['Attack Type'] = np.where(ids_data['Attack Type'] == 'BENIGN', 0, 1)

# Adjust the sample size based on available data

target_sample_size = min(15000, len(ids_data))

if target_sample_size > 0:

    bc_data = ids_data.sample(n=target_sample_size, random_state=42)

    print(bc_data['Attack Type'].value_counts())

else:

    print("Insufficient data even after balancing.")

# Splitting the data into features (X) and target (y)

from sklearn.model_selection import train_test_split

X_bc = bc_data.drop('Attack Type', axis = 1)

y_bc = bc_data['Attack Type']

X_train_bc, X_test_bc, y_train_bc, y_test_bc = train_test_split(X_bc, y_bc, test_size =
0.25, random_state = 0)

# Displaying the shapes of the resulting datasets

print(f'X_train_bc shape: {X_train_bc.shape}')

print(f'X_test_bc shape: {X_test_bc.shape}')

print(f'y_train_bc shape: {y_train_bc.shape}')

print(f'y_test_bc shape: {y_test_bc.shape}')

#Binary Classification

```



```

from sklearn.linear_model import LogisticRegression

lr1 = LogisticRegression(max_iter = 10000, C = 0.1, random_state = 0, solver = 'saga')

lr1.fit(X_train_bc, y_train_bc)

cv_lr1 = cross_val_score(lr1, X_train_bc, y_train_bc, cv = 5)

print('Logistic regression Model 1')

print(f'\nCross-validation scores:', ', '.join(map(str, cv_lr1)))

print(f'\nMean cross-validation score: {cv_lr1.mean():.2f}')

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Predict on the test set

y_pred_bc = lr1.predict(X_test_bc)

# Calculate accuracy

test_accuracy = accuracy_score(y_test_bc, y_pred_bc)

print(f'Test Accuracy: {test_accuracy:.2f}')

# Print confusion matrix

print('Confusion Matrix:')

print(confusion_matrix(y_test_bc, y_pred_bc))

# Print classification report

print('Classification Report:')

print(classification_report(y_test_bc, y_pred_bc))

print('Logistic Regression Model 1 coefficients:')

print(*lr1.coef_, sep = ', ')

print('\nLogistic Regression Model 1 intercept:', *lr1.intercept_)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_curve, roc_auc_score

# Predict on the test set

y_pred_bc = lr1.predict(X_test_bc)

y_pred_proba_bc = lr1.predict_proba(X_test_bc)[: , 1]

```

### **# Calculate metrics**

```
accuracy = accuracy_score(y_test_bc, y_pred_bc)
precision = precision_score(y_test_bc, y_pred_bc)
recall = recall_score(y_test_bc, y_pred_bc)
f1 = f1_score(y_test_bc, y_pred_bc)
auc = roc_auc_score(y_test_bc, y_pred_proba_bc)
```

```
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')
print(f'AUC: {auc:.2f}')
```

### **# Plot ROC curve**

```
fpr, tpr, _ = roc_curve(y_test_bc, y_pred_proba_bc)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='best')
plt.show()
```

### **#Creating a Balanced Dataset for Multi-class Classification**

```
new_data['Attack Type'].value_counts()
class_counts = new_data['Attack Type'].value_counts()
selected_classes = class_counts[class_counts > 1500]
class_names = selected_classes.index
selected = new_data[new_data['Attack Type'].isin(class_names)]
```

```

dfs = []

for name in class_names:

    df = selected[selected['Attack Type'] == name]

    if len(df) > 5000:

        df = df.sample(n=5000, replace=False, random_state=0)

    dfs.append(df)

df = pd.concat(dfs, ignore_index=True)
df['Attack Type'].value_counts()

import sklearn
import imblearn

from imblearn.over_sampling import SMOTE
import pandas as pd

# Assuming X and y are already defined

X = df.drop('Attack Type', axis=1)
y = df['Attack Type']

# Apply SMOTE to upsample the minority class

smote = SMOTE(sampling_strategy='auto', random_state=0)
X_upsampled, y_upsampled = smote.fit_resample(X, y)

# Create a new dataframe with the upsampled data

blnc_data = pd.DataFrame(X_upsampled)
blnc_data['Attack Type'] = y_upsampled
blnc_data = blnc_data.sample(frac=1, random_state=0) # Shuffle the dataframe

# Check the class distribution to verify balance

print(blnc_data['Attack Type'].value_counts())
features = blnc_data.drop('Attack Type', axis = 1)

```

```
labels = blnc_data['Attack Type']
```

```
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size = 0.25,  
random_state = 0)
```

```
print(f"\nTraining set size: {X_train.shape[0]} samples")
```

```
print(f"Test set size: {X_test.shape[0]} samples")
```

## **#SVM and Logistic Regression**

```
import time
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,  
roc_auc_score
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

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

### **# Logistic Regression**

```
lr = LogisticRegression(max_iter=10000, C=0.1, random_state=0, solver='saga')
```

### **# Measure training time for Logistic Regression**

```
start_train_lr = time.time()
```

```
lr.fit(X_train, y_train)
```

```
end_train_lr = time.time()
```

```
train_time_lr = end_train_lr - start_train_lr
```

### **# Cross Validation**

```
cv_lr = cross_val_score(lr, X_train, y_train, cv=5)
```

```
print('Logistic Regression')
```

```
print(f"\nCross-validation scores: ', ', '.join(map(str, cv_lr)))
```

```
print(f"\nMean cross-validation score: {cv_lr.mean():.2f}')
```

### **# Measure testing time for Logistic Regression**

```

start_test_lr = time.time()
y_pred_lr = lr.predict(X_test)
y_pred_prob_lr = lr.predict_proba(X_test)
end_test_lr = time.time()
test_time_lr = end_test_lr - start_test_lr

# Evaluate Logistic Regression

accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr, average='weighted')
recall_lr = recall_score(y_test, y_pred_lr, average='weighted')
f1_lr = f1_score(y_test, y_pred_lr, average='weighted')
auc_lr = roc_auc_score(y_test, y_pred_prob_lr, multi_class='ovo')

print('Logistic Regression Model')
print(f'Training Time: {train_time_lr:.2f} seconds')
print(f'Testing Time: {test_time_lr:.2f} seconds')
print(f'Accuracy: {accuracy_lr:.2f}')
print(f'Precision: {precision_lr:.2f}')
print(f'Recall: {recall_lr:.2f}')
print(f'F1 Score: {f1_lr:.2f}')
print(f'AUC: {auc_lr:.2f}')

# Compute confusion matrix

cm_lr = confusion_matrix(y_test, y_pred_lr, labels=lr.classes_)

# Create Confusion Matrix Display

disp = ConfusionMatrixDisplay(confusion_matrix=cm_lr, display_labels=lr.classes_)

# Plot confusion matrix

fig, ax = plt.subplots(figsize=(10, 7))
disp.plot(ax=ax, cmap=plt.cm.Blues, values_format='d')
plt.title('Confusion Matrix for Logistic Regression')

```

```
plt.show()
```

### **# SVM with RBF kernel**

```
svm = SVC(kernel='rbf', C=1, gamma=0.1, random_state=0, probability=True)
```

### **# Measure training time for SVM**

```
start_train_svm = time.time()
```

```
svm.fit(X_train, y_train)
```

```
end_train_svm = time.time()
```

```
train_time_svm = end_train_svm - start_train_svm
```

### **# Cross Validation**

```
cv_svm = cross_val_score(svm, X_train, y_train, cv=5)
```

```
print('\nSVM with RBF kernel')
```

```
print(f'\nCross-validation scores:', ', '.join(map(str, cv_svm)))
```

```
print(f'\nMean cross-validation score: {cv_svm.mean():.2f}')
```

### **# Measure testing time for SVM**

```
start_test_svm = time.time()
```

```
y_pred_svm = svm.predict(X_test)
```

```
y_pred_prob_svm = svm.predict_proba(X_test)
```

```
end_test_svm = time.time()
```

```
test_time_svm = end_test_svm - start_test_svm
```

### **# Evaluate SVM**

```
accuracy_svm = accuracy_score(y_test, y_pred_svm)
```

```
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
```

```
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
```

```
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')
```

```
auc_svm = roc_auc_score(y_test, y_pred_prob_svm, multi_class='ovo')
```

```
print('\nSVM Model')
```

```
print(f'Training Time: {train_time_svm:.2f} seconds')
```

```

print(f'Testing Time: {test_time_svm:.2f} seconds')
print(f'Accuracy: {accuracy_svm:.2f}')
print(f'Precision: {precision_svm:.2f}')
print(f'Recall: {recall_svm:.2f}')
print(f'F1 Score: {f1_svm:.2f}')
print(f'AUC: {auc_svm:.2f}')

# Compute confusion matrix

cm_svm = confusion_matrix(y_test, y_pred_svm, labels=svm.classes_)

# Create Confusion Matrix Display

disp = ConfusionMatrixDisplay(confusion_matrix=cm_svm,
display_labels=svm.classes_)

# Plot confusion matrix

fig, ax = plt.subplots(figsize=(10, 7))

disp.plot(ax=ax, cmap=plt.cm.Blues, values_format='d')

plt.title('Confusion Matrix for SVM with RBF kernel')

plt.show()

#Random Forest Model

rf1 = RandomForestClassifier(n_estimators=100, max_depth=6, max_features=None,
random_state=0)

# Measure training time for Random Forest Model 1

start_train_rf1 = time.time()

rf1.fit(X_train, y_train)

end_train_rf1 = time.time()

train_time_rf1 = end_train_rf1 - start_train_rf1

# Cross-validation

cv_rf1 = cross_val_score(rf1, X_train, y_train, cv=5)

print('Random Forest Model')

```

```

print(f'\nCross-validation scores:', ', '.join(map(str, cv_rf1)))
print(f'\nMean cross-validation score: {cv_rf1.mean():.2f}')

# Measure testing time for Random Forest Model

start_test_rf1 = time.time()
y_pred_rf1 = rf1.predict(X_test)
y_pred_prob_rf1 = rf1.predict_proba(X_test)
end_test_rf1 = time.time()
test_time_rf1 = end_test_rf1 - start_test_rf1

# Evaluate Random Forest Model

accuracy_rf1 = accuracy_score(y_test, y_pred_rf1)
precision_rf1 = precision_score(y_test, y_pred_rf1, average='weighted')
recall_rf1 = recall_score(y_test, y_pred_rf1, average='weighted')
f1_rf1 = f1_score(y_test, y_pred_rf1, average='weighted')
auc_rf1 = roc_auc_score(y_test, y_pred_prob_rf1, multi_class='ovo',
average='weighted')

print('Random Forest Model 1')
print(f'Training Time: {train_time_rf1:.2f} seconds')
print(f'Testing Time: {test_time_rf1:.2f} seconds')
print(f'Accuracy: {accuracy_rf1:.2f}')
print(f'Precision: {precision_rf1:.2f}')
print(f'Recall: {recall_rf1:.2f}')
print(f'F1 Score: {f1_rf1:.2f}')
print(f'AUC: {auc_rf1:.2f}')

# Compute confusion matrix

cm_rf1 = confusion_matrix(y_test, y_pred_rf1, labels=rf1.classes_)

# Create Confusion Matrix Display

disp = ConfusionMatrixDisplay(confusion_matrix=cm_rf1,
display_labels=rf1.classes_)

```



### **# Plot confusion matrix**

```
fig, ax = plt.subplots(figsize=(10, 7))  
disp.plot(ax=ax, cmap=plt.cm.Blues, values_format='d')  
plt.title('Confusion Matrix for Random Forest Model ')  
plt.show()
```

### **#KNN Model**

#### **# Initialize K-Nearest Neighbors model**

```
knn = KNeighborsClassifier(n_neighbors=5)
```

#### **# Measure training time for K-Nearest Neighbors**

```
start_train_knn = time.time()  
knn.fit(X_train, y_train)  
end_train_knn = time.time()  
train_time_knn = end_train_knn - start_train_knn
```

#### **# Cross Validation for KNN**

```
cv_knn = cross_val_score(knn, X_train, y_train, cv=5)  
print("\nK-Nearest Neighbors")  
print(f"\nCross-validation scores: {", ".join(map(str, cv_knn))}")  
print(f"\nMean cross-validation score: {cv_knn.mean():.2f}")
```

#### **# Measure testing time for K-Nearest Neighbors**

```
start_test_knn = time.time()  
y_pred_knn = knn.predict(X_test)  
y_pred_prob_knn = knn.predict_proba(X_test)  
end_test_knn = time.time()  
test_time_knn = end_test_knn - start_test_knn
```

#### **# Evaluate KNN**

```
accuracy_knn = accuracy_score(y_test, y_pred_knn)
```

```

precision_knn = precision_score(y_test, y_pred_knn, average='weighted')
recall_knn = recall_score(y_test, y_pred_knn, average='weighted')
f1_knn = f1_score(y_test, y_pred_knn, average='weighted')
auc_knn = roc_auc_score(y_test, y_pred_prob_knn, multi_class='ovo')

print('\nK-Nearest Neighbors Model')
print(f'Training Time: {train_time_knn:.2f} seconds')
print(f'Testing Time: {test_time_knn:.2f} seconds')
print(f'Accuracy: {accuracy_knn:.2f}')
print(f'Precision: {precision_knn:.2f}')
print(f'Recall: {recall_knn:.2f}')
print(f'F1 Score: {f1_knn:.2f}')
print(f'AUC: {auc_knn:.2f}')

# Compute confusion matrix

cm = confusion_matrix(y_test, y_pred_knn, labels=knn.classes_)

# Create Confusion Matrix Display

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knn.classes_)

# Plot confusion matrix

fig, ax = plt.subplots(figsize=(10, 7))

disp.plot(ax=ax, cmap=plt.cm.Blues, values_format='d')

plt.title('Confusion Matrix for K-Nearest Neighbors')

plt.show()

#Deep Learning Model

#CNN

# To ignore FutureWarning

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

warnings.simplefilter(action='ignore', category=DeprecationWarning)

```

### **#import the libraries**

```
from keras.models import Sequential

from keras import callbacks

from keras.layers import Dense, Activation, Flatten, Convolution1D, Dropout

from sklearn import metrics

from hyperopt import fmin, hp, tpe, Trials, STATUS_OK

from hyperopt.plotting import main_plot_history, main_plot_vars

import uuid

import gc

from tensorflow import keras

import tensorflow as tf

import pydot

from tensorflow.keras.utils import plot_model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import LSTM, Dense, Dropout
```

### **#Initial component of CNN**

```
model = Sequential()

model.add(Convolution1D(filters=128, kernel_size=6, input_shape=(87, 1)))

model.add(Activation('relu'))

model.add(Convolution1D(filters=256, kernel_size=6))

model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.1))

model.add(Dense(6))

model.add(Activation('softmax'))

model.summary()
```

### **#Plot the shape**

```
tf.keras.utils.plot_model(  
    model,  
    show_shapes=True,  
    show_dtype=False,  
    show_layer_names=False,  
)
```

### **# Compile the model**

```
model.compile(  
    optimizer=Adam(learning_rate=0.001),  
    loss='categorical_crossentropy',  
    metrics=['accuracy']  
)
```

### **# Ensure 'Attack Type\_encoded' is used for the target variable**

```
X = df.drop(['Attack Type', 'Attack Type_encoded'], axis=1)  
y = df['Attack Type_encoded']
```

### **# Apply SMOTE to upsample the minority classes**

```
smote = SMOTE(sampling_strategy='auto', random_state=0)  
X_upsampled, y_upsampled = smote.fit_resample(X, y)
```

### **# Create a new DataFrame with the upsampled data**

```
blnc_data = pd.DataFrame(X_upsampled, columns=X.columns)  
blnc_data['Attack Type_encoded'] = y_upsampled  
blnc_data = blnc_data.sample(frac=1, random_state=0) # Shuffle the DataFrame
```

### **# Check the class distribution to verify balance**

```
print(blnc_data['Attack Type_encoded'].value_counts())
```

### **# Define features and target with encoded labels**

```
features = blnc_data.drop('Attack Type_encoded', axis=1)
```

```
labels = blnc_data['Attack Type_encoded']
```

### **# Split the data into train and test sets**

```
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.25,  
random_state=0)
```

### **# Standardize the data**

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

### **# Reshape data for CNN**

```
X_train_cnn = X_train_scaled.reshape(X_train_scaled.shape[0],  
X_train_scaled.shape[1], 1)
```

```
X_test_cnn = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1],  
1)
```

### **# One-hot encode target variables**

```
y_train_cnn = tf.keras.utils.to_categorical(y_train, num_classes=len(class_labels))
```

```
y_test_cnn = tf.keras.utils.to_categorical(y_test, num_classes=len(class_labels))
```

### **# Train the model**

```
start_train_time = time.time()
```

```
history = model.fit(X_train_cnn, y_train_cnn, epochs=50, batch_size=32,  
validation_split=0.2, verbose=1)
```

```
end_train_time = time.time()
```

```
train_time = end_train_time - start_train_time
```

### **# Evaluate the model**

```
start_test_time = time.time()
```

```
y_pred_cnn = model.predict(X_test_cnn)
```

```
y_pred_classes = np.argmax(y_pred_cnn, axis=1)
```

```
y_test_classes = np.argmax(y_test_cnn, axis=1)
```

```
end_test_time = time.time()
```

```

test_time = end_test_time - start_test_time

# Calculate metrics

accuracy = accuracy_score(y_test_classes, y_pred_classes)
precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')
auc = roc_auc_score(y_test_cnn, y_pred_cnn, multi_class='ovr')

# Print results

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"AUC: {auc:.2f}")
print(f"Training Time: {train_time:.2f} seconds")
print(f"Testing Time: {test_time:.2f} seconds")

# Plot confusion matrix

cm = confusion_matrix(y_test_classes, y_pred_classes)
plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
            yticklabels=class_labels)

plt.title(f'Confusion Matrix for CNN')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

```

## **#LSTM**

### **# Reshape data for LSTM**

```
X_train_lstm = X_train_scaled.reshape(X_train_scaled.shape[0], 1,  
X_train_scaled.shape[1])
```

```
X_test_lstm = X_test_scaled.reshape(X_test_scaled.shape[0], 1,  
X_test_scaled.shape[1])
```

### **# One-hot encode target variables**

```
y_train_lstm = tf.keras.utils.to_categorical(y_train, num_classes=len(class_labels))
```

```
y_test_lstm = tf.keras.utils.to_categorical(y_test, num_classes=len(class_labels))
```

### **# Build LSTM model**

```
model = Sequential()
```

```
model.add(LSTM(128, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2]),  
return_sequences=True))
```

```
model.add(Dropout(0.2))
```

```
model.add(LSTM(64, return_sequences=False))
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(len(class_labels), activation='softmax'))
```

### **# Compile the model**

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

### **# Train the model**

```
start_train_time = time.time()
```

```
history = model.fit(X_train_lstm, y_train_lstm, epochs=50, batch_size=32,  
validation_split=0.2, verbose=1)
```

```
end_train_time = time.time()
```

```
train_time = end_train_time - start_train_time
```

### **# Evaluate the model**

```
start_test_time = time.time()
```

```
y_pred_lstm = model.predict(X_test_lstm)
```

```

y_pred_classes = np.argmax(y_pred_lstm, axis=1)
y_test_classes = np.argmax(y_test_lstm, axis=1)
end_test_time = time.time()
test_time = end_test_time - start_test_time

# Calculate metrics

accuracy = accuracy_score(y_test_classes, y_pred_classes)
precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')
auc = roc_auc_score(y_test_lstm, y_pred_lstm, multi_class='ovr')

# Print results

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"AUC: {auc:.2f}")
print(f"Training Time: {train_time:.2f} seconds")
print(f"Testing Time: {test_time:.2f} seconds")

# Plot confusion matrix

cm = confusion_matrix(y_test_classes, y_pred_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
yticklabels=class_labels)
plt.title(f'Confusion Matrix for LSTM')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

```