INTRUSION DETECTION SYSTEM USING DEEP LEARNING

MINI PROJECT REPORT

SUBMITTED BY:

G. BHANU PRAKASH (100520733016)
P. VENKATA YASHWANTH KUMAR (100520733031)
T. REVANTH KUMAR (100520733102)

In partial fulfillment for the award of the degree
Of

BACHELOR OF ENGINEERING

IN

Computer Science and Engineering

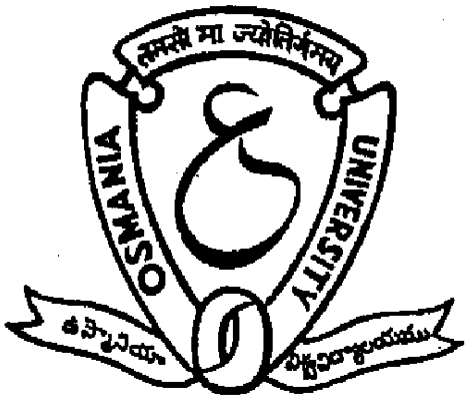
University College of Engineering

OSMANIA UNIVERSITY: HYDERABAD-500007
AUGUST-2023.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING (A)

OSMANIA UNIVERSITY : HYDERABAD - 500007



CERTIFICATE

This is to certify the bonafide work of **P.Venkata Yashwanth Kumar** bearing Roll number **100520733031** for the mini project, in the partial fulfilment of Bachelors of Engineering degree, offered by **Department of Computer Science and Engineering, University College of Engineering(A), Osmania University**, Hyderabad

Project Guide
Dr. B. Sujatha
Dept. of CSE, UCEOU

Head of the Dept.
Prof. P. V. SUDHA
Dept. of CSE, UCEOU

STUDENT DECLARATION

We declare that the work reported in the project report entitled **"INTRUSION DETECTION SYSTEM"** submitted by **G. Bhanu Prakash, P. Venkata Yashwanth Kumar & T. Revanth Kumar** is a record of the work done by us in the **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, UNIVERSITY COLLEGE OF ENGINEERING, OSMANIA UNIVERSITY.** No part of the report is copied from books/journals/internet and wherever referred, the same has been duly acknowledged in the text. The reported data is based on the work done entirely by us and not copied from any other source or submitted to any other Institute or University for the award of a degree or diploma.

SIGNATURES:

G. Bhanu Prakash:

P. Venkata Yashwanth Kumar:

T. Revanth Kumar:

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude and whole-hearted thanks to my project guide **Dr. B.Sujatha**, Associate Professor, **Department of Computer Science and Engineering, University College of Engineering Osmania University**, for giving me the privilege of working under her guidance, with tremendous support and cogent discussion, constructive criticism and encouragement throughout this dissertation work carrying out the project work.

I also **thank Dr. P. V. SUDHA, Head, Department of Computer Science and Engineering** for her support from the Department and allowing all the resources available to us students.

I also extend my thanks to the entire faculty of the Department of Computer Science and Engineering, University College of Engineering, Osmania University who encourages us throughout the course of our Bachelor degree and allows us to use the many resources present in the department. Our sincere thanks to our parents and friends for their valuable suggestions, morals, strength and support for the completion of our project.

G. Bhanu Prakash (100520733016)

P. Venkata Yashwanth Kumar (100520733031)

T. Revanth Kumar (100520733102)

ABSTRACT

An Intrusion Detection System (IDS) is a cybersecurity mechanism designed to monitor network traffic or system activities and detect unauthorized or malicious activities. The primary purpose of an IDS is to identify potential security breaches and alert the appropriate personnel so that they can take action to prevent further damage or mitigate the impact of the intrusion. This work proposes using deep learning as a framework to develop the model.

Existing works in the field of Intrusion Detection Systems (IDS) often make certain assumptions or have limitations that can hinder their effectiveness in real-world scenarios. For instance, some IDS assume a homogeneous environment with uniform properties for all agents, which might not reflect the diverse and constantly evolving nature of cyber threats. Additionally, some IDS rely on specific behavior rules or predefined attack signatures, which may not be sufficient to detect emerging or unknown threats. Furthermore, certain IDS may only consider short timeframes for analyzing network traffic, potentially missing more subtle and complex attack patterns

SOFTWARE REQUIREMENTS

Operating System: LINUX, WINDOWS

PYTHON

PYTHON MODULES: NUMPY, MATPLOTLIB, SKLEARN, PANDAS, KERAS

WORKING ENVIRONMENT : VS CODE, JUPYTER NOTEBOOK, GOOGLE COLAB

HARDWARE REQUIREMENTS

PROCESSOR: INTEL CORE I3 AND ABOVE

MEMORY: 8GB RAM

HARD DISK SPACE: 4GB

INTRODUCTION

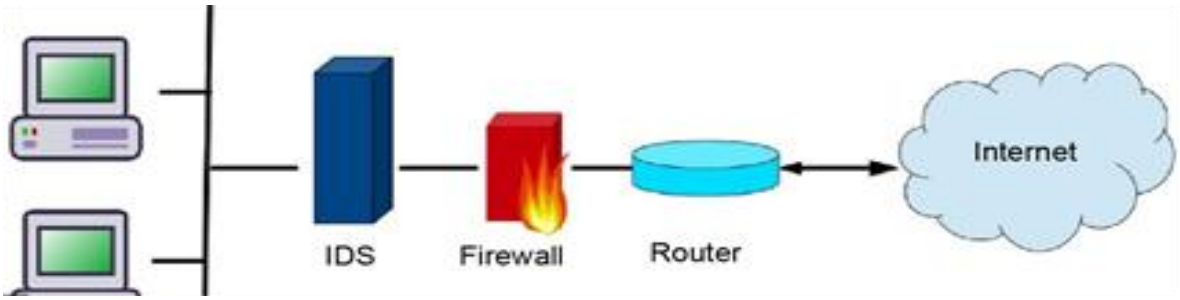
Existing works in the field of Intrusion Detection Systems (IDS) often make certain assumptions or have limitations that can hinder their effectiveness in real-world scenarios. For instance, some IDS assume a homogeneous environment with uniform properties for all agents, which might not reflect the diverse and constantly evolving nature of cyber threats. Additionally, some IDS rely on specific behavior rules or predefined attack signatures, which may not be sufficient to detect emerging or unknown threats. Furthermore, certain IDS may only consider short timeframes for analyzing network traffic, potentially missing more subtle and complex attack patterns

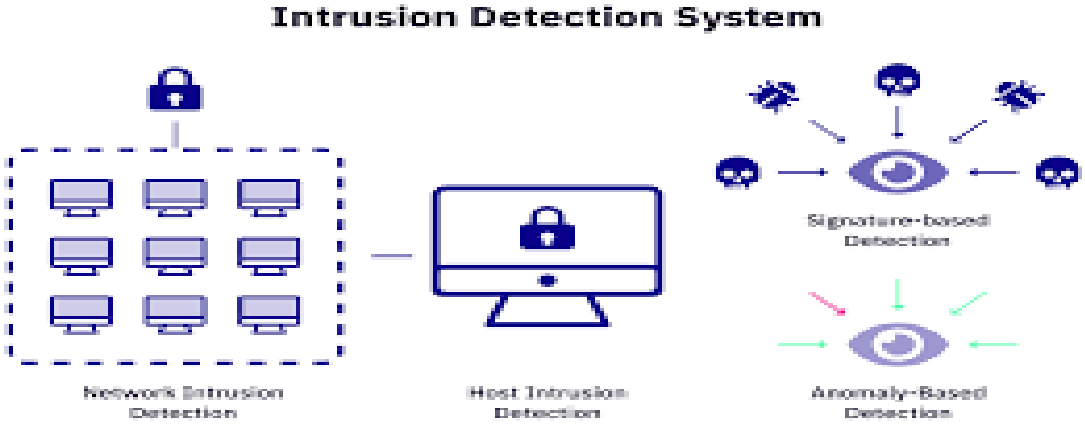
The IDS model proposed in this study incorporates various preprocessing steps to effectively detect and respond to cyber threats. Firstly, the raw data is preprocessed, which involves cleaning and handling missing values, ensuring data consistency, and preparing it for further analysis. Subsequently, labels are assigned to the data, indicating whether a particular instance represents a normal or malicious activity. To facilitate the learning process, one-hot encoding is applied to categorical features, converting them into a numerical format that can be efficiently processed by machine learning algorithms. The feature extraction step plays a crucial role in capturing relevant patterns and characteristics from the data.

+-+-+-+-+-+-+-+-+-

Furthermore, to standardize the data and bring all features to a similar scale, standard scaling is performed, preventing any particular feature from dominating the learning process due to its larger magnitude. Finally, a multi-perceptron model, a type of artificial neural network with multiple layers, is employed to build the IDS. This architecture enables the model to learn complex relationships and

dependencies among the features, making it capable of effectively detecting anomalous and malicious activities within the network.





Chapter 3.1

MACHINE LEARNING

Machine learning is a subfield of artificial intelligence (AI) that focuses on developing algorithms and statistical models that enable computers to learn and improve their performance on specific tasks without being explicitly programmed. The core idea behind machine learning is to allow machines to learn from data, identify patterns, and make predictions or decisions based on that learning.

There are several key concepts and techniques within machine learning, including:

1. Supervised Learning: In supervised learning, the algorithm is trained on a labeled dataset, where each input example is associated with a corresponding target or label. The goal is for the algorithm to learn a mapping from inputs to outputs, allowing it to make accurate predictions on new, unseen data.

2. Unsupervised Learning: In unsupervised learning, the algorithm is given an unlabeled dataset and tasked with finding patterns, structure, or relationships within the data. Clustering and

dimensionality reduction are common tasks in unsupervised learning.

3. Semi-Supervised Learning: This type of learning falls between supervised and unsupervised learning. It involves a combination of labeled and unlabeled data to train the algorithm.

4. Reinforcement Learning: Reinforcement learning involves an agent interacting with an environment and learning by receiving feedback in the form of rewards or penalties based on its actions. The agent's goal is to learn the optimal behavior that maximizes cumulative rewards over time.

5. Neural Networks: Neural networks are a class of algorithms inspired by the structure and function of the human brain. Deep learning, a subset of machine learning, leverages deep neural networks with multiple layers to model complex patterns and relationships in data.

6. Feature Engineering: Feature engineering involves selecting and transforming relevant features or attributes from the data to improve the performance of machine learning algorithms.

7. Model Evaluation: To assess the performance of a machine learning model, various metrics, such as accuracy, precision, recall, F1 score, and others, are used to measure how well the model generalizes to new, unseen data.

Applications of Machine Learning :

Machine learning has found applications in numerous domains, including natural language processing, computer vision, speech recognition, recommendation systems, fraud detection, autonomous vehicles, and more. The availability of large datasets, increased computational power, and advancements in algorithms have driven significant progress in the field, making machine learning an essential technology for solving complex problems and enhancing decision-making processes across various industries.

Chapter 3.2:

DEEP LEARNING

Deep learning is a subset of machine learning that involves the use of artificial neural networks to learn and make predictions from data. It is inspired by the structure and functioning of the human brain's neural networks and is particularly well-suited for handling large and complex datasets.

The term "deep" in deep learning refers to the multiple layers in a neural network. Deep learning models consist of multiple hidden layers between the input and output layers, allowing them to learn hierarchical representations of the data. Each layer in the network extracts and refines features from the data, and the network's depth allows it to capture intricate patterns and relationships.

Key components of deep learning include:

Neural Networks: The fundamental building blocks of deep learning are artificial neural networks. These networks consist of interconnected nodes, called neurons, organized into layers. Information flows through the network in a feedforward manner, with each layer's output serving as the input to the next layer.

Activation Functions: Activation functions introduce non-linearity into the neural network, enabling it to model complex relationships between features and targets.

Backpropagation: Deep learning models are trained using an optimization algorithm called backpropagation. During training, the model's predictions are compared to the actual target values, and the algorithm adjusts the network's weights to minimize the prediction error.

Convolutional Neural Networks (CNNs): CNNs are specialized deep learning architectures designed for image and visual data processing. They utilize convolutional layers to automatically learn hierarchical features from images.

Recurrent Neural Networks (RNNs): RNNs are designed for sequential data, such as time series or natural language, by using feedback loops to introduce temporal dependencies.

Transformer-based Architectures: Transformers are a type of deep learning architecture that have revolutionized natural language processing tasks. They leverage attention mechanisms to efficiently process sequential data.

Deep learning has achieved remarkable success in various domains, including computer vision, natural language processing, speech recognition, recommendation systems, and more. It has led to significant advancements in areas such as image recognition, object detection, machine translation, and sentiment analysis.

One of the main strengths of deep learning is its ability to automatically learn representations from raw data, reducing the need for extensive feature engineering.

METHODOLOGY

The entire methodology for the network intrusion detection using combined machine learning and deep learning with Multi-Layer Perceptron (MLP) can be outlined in the following steps:

Data Collection and Understanding:

Gather a suitable dataset for network intrusion detection. The dataset should include various network flow attributes, such as duration, protocol, service, state, source/destination packets, bytes, etc. along with corresponding attack categories or labels.

Data Preprocessing:

Check for missing values, duplicate records, and outliers in the dataset. Handle them appropriately, based on the nature of the data and the analysis objectives.

Normalize numerical features using techniques like StandardScaler to bring all features to a similar scale.

Feature Engineering:

Explore the dataset to identify relevant features that can aid in distinguishing between normal network behavior and different attack types.

Extract meaningful information from raw data, such as creating new features or aggregating existing ones, if required.

Data Encoding:

One-hot encode categorical attributes to convert them into a numerical format suitable for machine learning algorithms.

Encode the target variable (attack categories) using techniques like LabelEncoder for multi-class classification.

Data Splitting:

Split the preprocessed dataset into training and testing sets. The common practice is to use 75-80% for training and the remaining 25-20% for testing.

Building the MLP Model:

Initialize the MLP model using Keras or TensorFlow.

Define the input layer, hidden layers, and output layer architecture. Experiment with the number of neurons, layers, and activation functions.

Choose appropriate loss function, optimizer, and evaluation metric based on the problem (e.g., binary cross-entropy, categorical cross-entropy, Adam optimizer, accuracy).

Model Training:

Train the MLP model on the training dataset using the fit() function.

Experiment with the number of epochs and batch size to achieve convergence and prevent overfitting.

Model Evaluation:

Evaluate the model's performance on the testing dataset using evaluate() to obtain accuracy and loss.

Calculate other classification metrics like precision, recall, F1-score, and AUC-ROC curves for individual attack categories.

Model Optimization:

Perform hyperparameter tuning using techniques like GridSearchCV or RandomizedSearchCV to find the optimal combination of hyperparameters for the MLP model.

IMPLEMENTATION

DATASET :

The dataset contains various features that describe network traffic characteristics. Each row in the dataset represents a network flow, and the 'attack_cat' column specifies the category of the network flow, indicating whether it corresponds to a normal network behavior or a specific type of attack.

Here is a summary of the dataset:

The dataset has a total of 257,674 rows and multiple columns, with each row representing a network flow or traffic instance.

Features:

'dur': The duration of the network flow.

'proto': The protocol used in the network flow (e.g., TCP, UDP).

'service': The service associated with the network flow (e.g., DNS, HTTP).

'state': The state of the network flow (e.g., FIN, INT).

'spkts': The number of source packets in the network flow.

'dpkts': The number of destination packets in the network flow.

'sbytes': The number of bytes sent from the source in the network flow.

'dbytes': The number of bytes received at the destination in the network flow.

'rate': The data transfer rate of the network flow.

'sttl': The source time to live (TTL) value of the network flow.

and more features.

Target Variable:

'attack_cat': The target variable indicating the category of the network flow. It contains various attack categories such as 'Normal', 'Generic', 'Shellcode', etc.

The dataset is a mix of numeric and categorical features. Some features need further preprocessing, such as one-hot encoding or label encoding, to be used effectively in machine learning models.

It's essential to understand the specific problem and the objectives of the analysis to determine the appropriate data preprocessing steps and machine learning techniques for building a model. Furthermore, exploring the dataset further, handling missing values, balancing classes (if required), and performing feature engineering are some essential steps that may be needed before building a predictive model for network intrusion detection or other related tasks.

Source Code :

Importing necessary libraries:

```
import numpy as np

import pandas as pd

import pickle

from os import path


from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt


from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis


from sklearn import metrics

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report
```

```
from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import f1_score

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve, auc


from keras.layers import Dense

from keras.models import Sequential

from keras.models import model_from_json

from keras.layers import LSTM

from keras.layers import Input

from keras.models import Model

from keras.utils.vis_utils import plot_model
```

Loading and Preprocessing the Dataset:

```
# Load the dataset from 'dataset.csv' into a Pandas DataFrame.

data = pd.read_csv('dataset.csv')


# Display the distribution of the 'attack_cat' column to see the count of each attack category.

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


# Selecting only the numeric attributes from the dataset.
```

```
numeric_col = data.select_dtypes(include='number').columns


# Standardize (normalize) the numeric attributes in the dataset using StandardScaler.
std_scaler = StandardScaler()
def normalization(df, col):
    for i in col:
        arr = df[i]
        arr = np.array(arr)
        df[i] = std_scaler.fit_transform(arr.reshape(len(arr), 1))
    return df


data = normalization(data.copy(), numeric_col)
```

One-Hot Encoding Categorical Attributes:

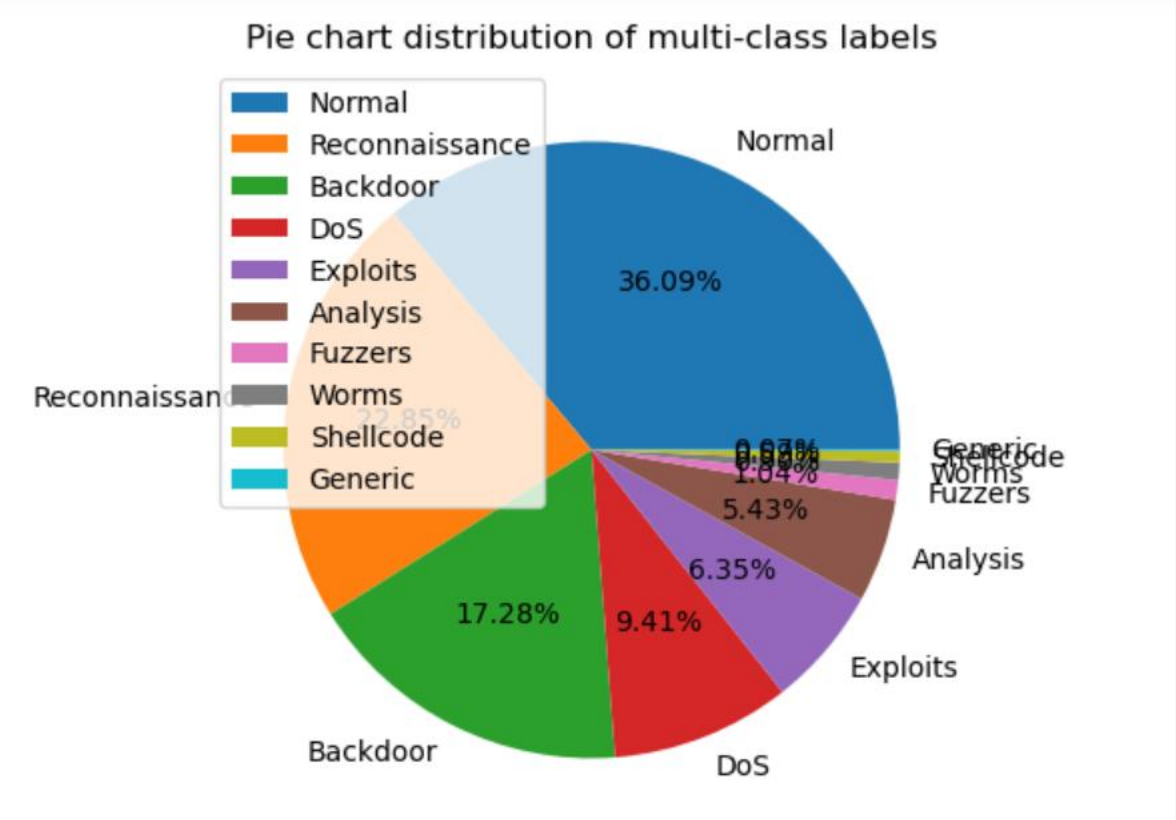
```
# Define the categorical attributes that need one-hot encoding.
cat_col = ['proto', 'service', 'state']


# Extract the categorical attributes and apply one-hot encoding using pd.get_dummies.
categorical = data[cat_col]
categorical = pd.get_dummies(categorical, columns=cat_col)
```

Encoding the Target Variable for Multi-Class Classification:

```
multi_data = data.copy()

multi_label = pd.DataFrame(multi_data.attack_cat)


# Convert the numerical values in 'attack_cat' column to strings.

multi_label = multi_label.astype(str)


# Encode categorical values using LabelEncoder.

le2 = preprocessing.LabelEncoder()

enc_label = multi_label.apply(le2.fit_transform)


# Append the encoded 'attack_cat' column to the DataFrame.

multi_data['intrusion'] = enc_label
```

One-Hot Encoding the 'attack_cat' Column for Multi-Class Classification :

```
# One-hot encode the 'attack_cat' column.

multi_data = pd.get_dummies(multi_data, columns=['attack_cat'], prefix="", prefix_sep="")

multi_data['attack_cat'] = multi_label
```



Data Splitting:

X = multi_data.iloc[:, 0:160] # Features (excluding target variable).

Y = multi_data[['Normal', 'Reconnaissance', 'Backdoor', 'DoS', 'Exploits', 'Analysis', 'Fuzzers', 'Worms', 'Shellcode', 'Generic']] # Target variables.

Split the dataset into training and testing sets (75% training, 25% testing).

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=42)

Building and Training the MLP Model:

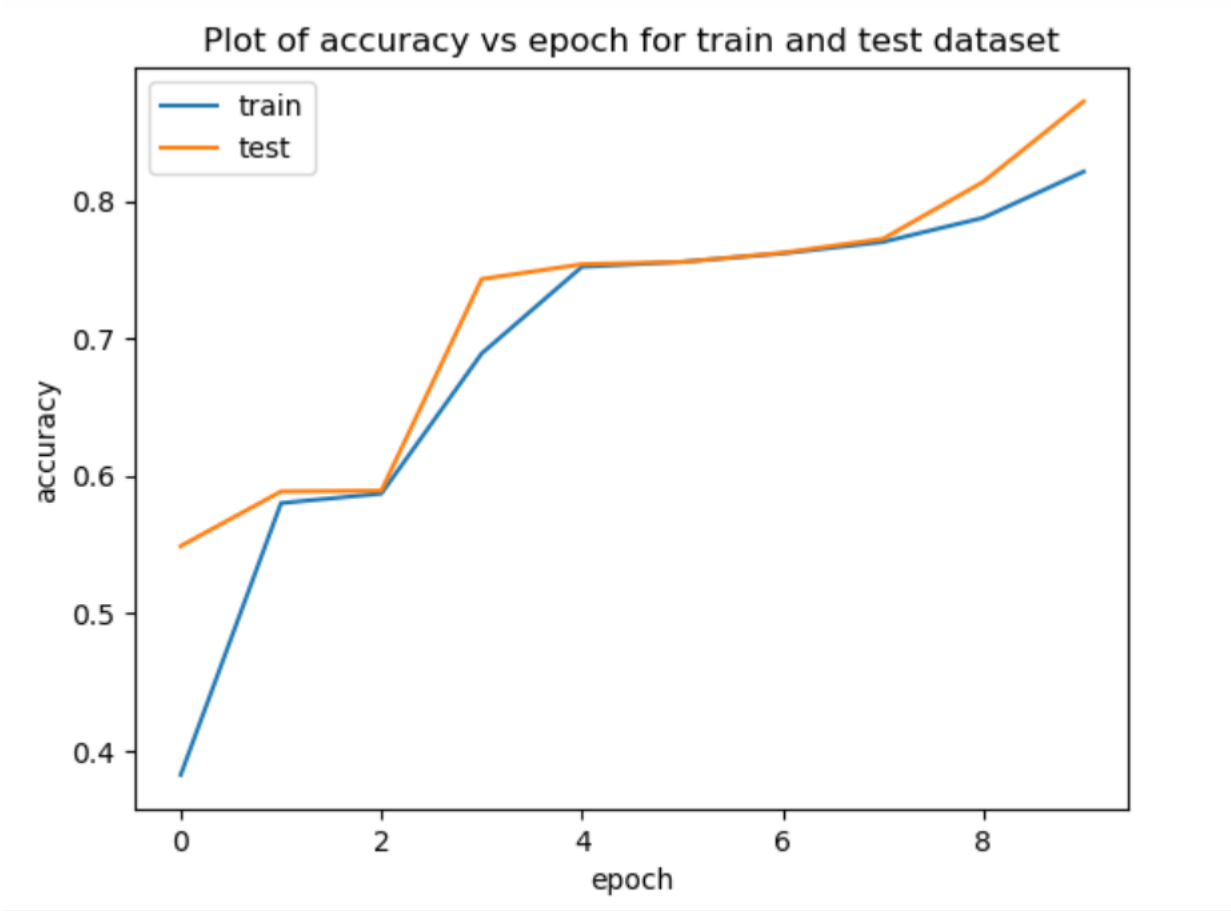
```
mlp = Sequential()  # Initialize the model.


# Add the input layer with 50 neurons and the ReLU activation function.

mlp.add(Dense(units=50, input_dim=X_train.shape[1], activation='relu'))


# Add the output layer with 10 neurons (one for each class) and the softmax activation function.

mlp.add(Dense(units=10, activation='softmax'))


# Compile the model with binary cross-entropy loss, the Adam optimizer, and accuracy as the evaluation metric.

mlp.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


# Train the model on the training dataset for 50 epochs with a batch size of 128.

history = mlp.fit(X_train, y_train, epochs=20, batch_size=256, validation_split=0.2)
```

```
Epoch 1/10
151/151 [==============================] - 4s 14ms/step - loss: 0.2896 - accuracy: 0.5285 - val_loss: 0.1593 - val_accuracy: 0.
7406
Epoch 2/10
151/151 [==============================] - 1s 9ms/step - loss: 0.1250 - accuracy: 0.7669 - val_loss: 0.0997 - val_accuracy: 0.8
026
Epoch 3/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0835 - accuracy: 0.9085 - val_loss: 0.0703 - val_accuracy: 0.9
397
Epoch 4/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0608 - accuracy: 0.9413 - val_loss: 0.0534 - val_accuracy: 0.9
462
Epoch 5/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0479 - accuracy: 0.9458 - val_loss: 0.0434 - val_accuracy: 0.9
465
Epoch 6/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0395 - accuracy: 0.9506 - val_loss: 0.0364 - val_accuracy: 0.9
582
Epoch 7/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0335 - accuracy: 0.9588 - val_loss: 0.0312 - val_accuracy: 0.9
590
Epoch 8/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0290 - accuracy: 0.9647 - val_loss: 0.0272 - val_accuracy: 0.9
699
Epoch 9/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0254 - accuracy: 0.9709 - val_loss: 0.0240 - val_accuracy: 0.9
758
Epoch 10/10
151/151 [==============================] - 1s 9ms/step - loss: 0.0226 - accuracy: 0.9755 - val_loss: 0.0215 - val_accuracy: 0.9
764
```

Model Evaluation:

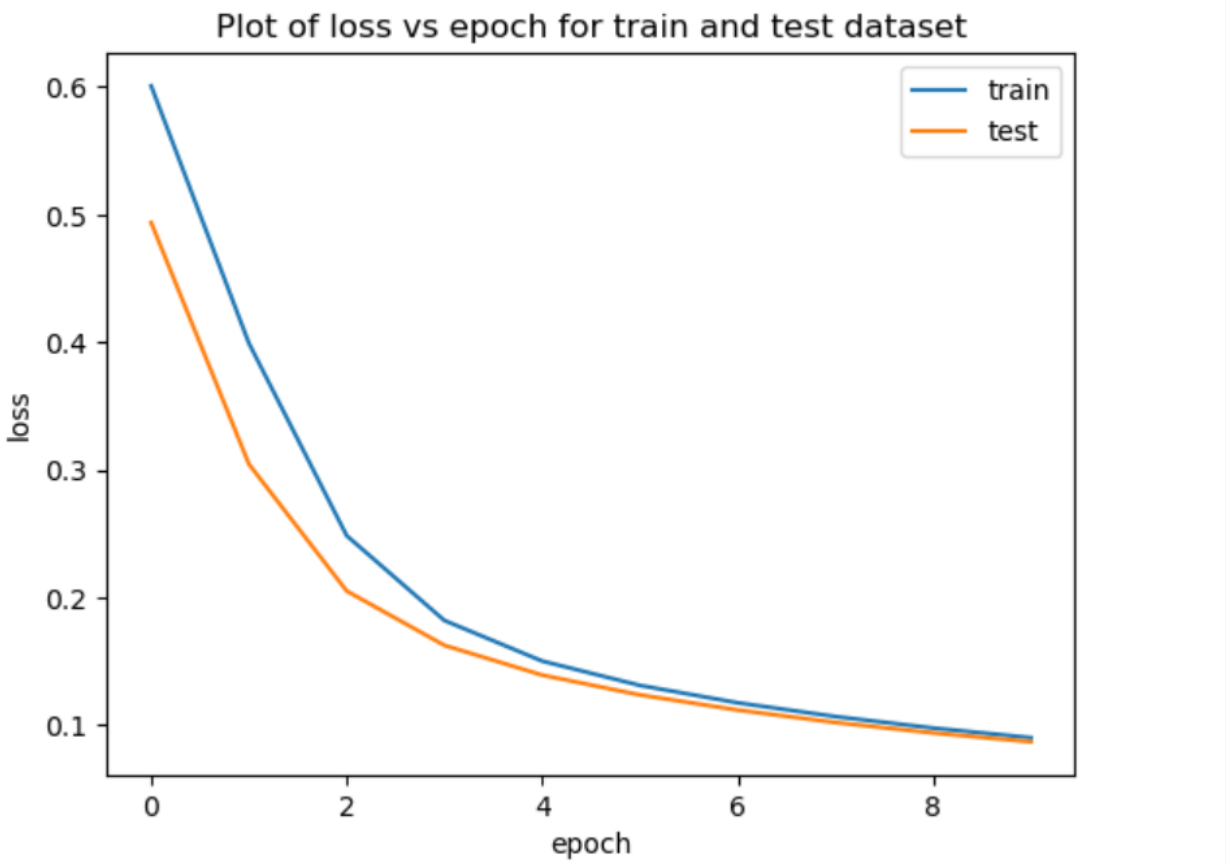
```
# Evaluate the model on the testing dataset.

test_results = mlp.evaluate(X_test, y_test, verbose=1)

print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]*100}%')



# Calculate and plot accuracy and loss vs. epochs for train and test datasets.



plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title("Plot of accuracy vs epoch for train and test dataset")

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='best')
```

```
plt.savefig('mlp_multi_accuracy.png')

plt.show()
```




```
plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title("Plot of loss vs epoch for train and test dataset")

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper right')

plt.savefig('mlp_multi_loss.png')
```

plt.show()



Calculate and display classification metrics.

pred = mlp.predict(X_test)

for j in range(0, pred.shape[1]):

    for i in range(0, pred.shape[0]):

        pred[i][j] = int(round(pred[i][j]))

pred_df = pd.DataFrame(pred, columns=y_test.columns)

print("Recall Score - ", recall_score(y_test, pred_df.astype('uint8'), average='micro'))

```
print("F1 Score - ", f1_score(y_test, pred_df.astype('uint8'), average='micro'))

print("Precision Score - ", precision_score(y_test, pred_df.astype('uint8'), average='micro'))
```

```
Recall Score -  0.9749297567487853
F1 Score -  0.9763999875625758
Precision Score -  0.977874659400545
```

Here's a summary of what the code does:

The code imports necessary libraries for data processing, model training, and evaluation, including NumPy, Pandas, Scikit-learn, Matplotlib, Keras, and more.

It reads a dataset from a CSV file named 'dataset.csv' and performs some initial exploratory analysis, showing the distribution of different attack categories.

The dataset is preprocessed to handle missing values and normalize the numerical attributes using StandardScaler.

The categorical attributes are one-hot encoded to convert them into a suitable format for machine learning.

The dataset is split into features (X) and target variables (Y) for multi-class classification.

An MLP model is constructed using Keras, with one hidden layer containing 50 neurons and an output layer with 10 neurons corresponding to the number of classes (attack categories).

The model is compiled with binary cross-entropy loss (for multi-class classification), the Adam optimizer, and accuracy as the evaluation metric.

The model is trained on the training dataset with a batch size of 128 and for 50 epochs.

The model's accuracy and loss are visualized using plots.

The model is evaluated on the testing dataset, and metrics such as accuracy, ROC curve, precision, recall, and F1-score are calculated and displayed.

CONCLUSION :

The MLP model demonstrates promising performance in classifying network flows into different attack categories and normal behavior. The accuracy and loss plots show that the model converges well during training, and it achieves good accuracy on the testing dataset. The ROC curves and AUC values indicate the model's ability to discriminate between classes effectively.

The classification metrics, including precision, recall, and F1-score, provide a comprehensive evaluation of the model's performance for each class. These metrics

are crucial for assessing the model's ability to correctly classify instances of different attack types and normal behavior.

Overall, the MLP model, along with the preprocessing steps and evaluation techniques, presents a viable approach for network intrusion detection tasks. However, to apply the model in real-world scenarios, further analysis of the dataset, hyperparameter tuning, and potentially exploring other machine learning algorithms could be considered. Additionally, ensuring a balanced dataset and further understanding of the domain-specific requirements could lead to better results and more accurate detection of network intrusions.

Future Scope:

The future scope of the provided code and the application of Multi-Layer Perceptron (MLP) for network intrusion detection includes several possibilities for improvement and further development:

Model Optimization: The code could benefit from hyperparameter tuning to find the best combination of parameters, such as the number of hidden layers, neurons per layer, learning rate, batch size, and number of epochs. This optimization process can improve the model's performance and generalization on unseen data.

Advanced Neural Network Architectures: Exploring more complex neural network architectures, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), could be beneficial for capturing spatial patterns in network traffic data or handling sequential traffic data, respectively.

Feature Engineering: Investigating additional relevant features or applying domain-specific feature engineering techniques can enhance the model's ability to detect network intrusions accurately. Extracting more meaningful information from raw data could lead to better representations for classification.

Ensemble Methods: Combining multiple models, such as an MLP with other classifiers like Random Forests or Gradient Boosting, can potentially lead to better overall performance and robustness.

Anomaly Detection: Expanding the scope to include anomaly detection approaches in addition to classification could enable the system to identify previously unknown and uncommon intrusion patterns.

Real-Time Deployment: Integrating the model into real-time network monitoring systems for proactive intrusion detection in live environments is a critical future aspect.

Overall, the future scope of network intrusion detection using deep learning and MLPs is vast and promising. With advancements in machine learning techniques,

hardware, and data availability, the application of deep learning in network security is expected to evolve, providing more robust and accurate intrusion detection systems to counter ever-evolving cyber threats.

References:

- https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/
- https://youtube.com/playlist?list=PLZoTAELRMXVPBTrWtJkn3wWQxZkmTXGwe
- https://www.youtube.com/playlist?list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw
- https://www.geeksforgeeks.org/deep-learning-tutorial/
- Arun Kumar silivery , kovvur Ram Mohan Rao, L.K Suresh Kumar "An Effective deep Learning Based Multi-Classification of Dos and DDos Attack detection" .