Machine Learning

Project Submission Report(UML501)

To Classify DDos Traffic and Regular Traffic

Submitted by

Sukirti Sharma - 102103039

Kshitij Gupta - 102103056

Sub-Group

3CO2

Submitted to

Dr. Anjula Mehto

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB

INDIA

July-December 2023

1. Introduction:

In today's interconnected digital landscape, the escalation of cybersecurity threats has become an alarming reality. As businesses and individuals rely more heavily on online platforms, the risk of malicious activities disrupting the integrity and availability of digital services is on the rise. Cybersecurity, once considered a niche concern, has now taken center stage as a critical aspect of safeguarding sensitive information and ensuring the continuous operation of essential services.

Amidst the myriad of cyber threats, Distributed Denial of Service (DDoS) attacks stand out as particularly pervasive and damaging. Recent years have witnessed a staggering increase in the frequency and sophistication of DDoS attacks, posing a severe challenge to the resilience of online infrastructures.

The motives behind DDoS attacks have evolved beyond mere nuisance, with cybercriminals often employing them as a smokescreen to facilitate other malicious activities or to extort targeted organizations. This escalating trend necessitates proactive and adaptive cybersecurity measures to protect against the potentially devastating consequences of service disruptions, financial losses, and reputational damage.

In response to this growing threat landscape, this project addresses the imperative need for advanced tools and methodologies in the realm of cybersecurity. By focusing on the development of a machine learning-based solution for the classification of DDoS and regular network traffic, we aim to contribute to the fortification of digital ecosystems and the preservation of the availability and reliability of online services. The following sections delve into the specific problem statement, dataset characteristics, chosen methodology, and the anticipated outcomes of this endeavor.

2. Problem Statement:

The manual identification of DDoS traffic amidst the vast volume of data in network logs is a complex and time-consuming task. Human intervention in distinguishing between malicious and regular traffic is not only impractical but also prone to errors. The project addresses the need for an automated system that can accurately and swiftly classify network traffic to mitigate the risks associated with DDoS attacks. The goal is to

develop a machine learning model that can effectively differentiate between the patterns of normal network activity and those indicative of a DDoS attack, thus bolstering the overall security posture of networked systems.

3. Dataset:

This is a SDN specific data set generated by using mininet emulator and used for traffic classification by machine learning and deep learning algorithms. The project starts by creating ten topologies in mininet in which switches are connected to a single Ryu controller. Network simulation runs for benign TCP, UDP and ICMP traffic and malicious traffic which is the collection of TCP Syn attack, UDP Flood attack, ICMP attack. Total 23 features are available in the data set in which some are extracted from the switches and others are calculated. Extracted features include Switch-id, Packet_count, byte_count, duration_sec, duration_nsec which is duration in nano-seconds, total duration is sum of duration_sec and durstaion_nsec, Source IP, Destination IP, Port number, tx_bytes is the number of bytes transferred from the switch port, rx_bytes is the number of bytes received on the switch port. dt field shows the date and time which has been converted into number and a flow is monitored at a monitoring interval of 30 seconds. Calculated features include Packet per flow which is packet count during a single flow, Byte per flow is byte count during a single flow, Packet Rate is number of packets send per second and calculated by dividing the packet per flow by monitoring interval, number of Packet_ins messages, total flow entries in the switch, tx_kbps, rx_kbps are data transfer and receiving rate and Port Bandwidth is the sum of tx_kbps and rx_kbps.

Last column indicates the class label which indicates whether the traffic type is benign or malicious. Benign traffic has label 0 and malicious traffic has label 1. Network simulation is run for 250 minutes and 1,04,345 rows of data is collected. The simulation is run for a defined interval again and more data can be collected.

DATASET LINK:
https://datasetsearch.research.google.com/search?query=sdn&docid=L2cvMTFtbV92c3gyMQ%3D%3D

4. Methodology:

1. Loading and Preprocessing Data:

- The dataset is loaded using the `load_data` function, and then the `preprocess_data` function is called to prepare the data for training.
- Label encoding is applied to convert categorical labels ('DDoS' and 'Normal') into numerical values.
- One-hot encoding is performed on categorical features ('src', 'dst', 'Protocol') to represent them as binary vectors.
- Null values are dropped from the dataset.
- The data is split into training and testing sets using the `train_test_split` function.

```
data = load_data("/content/dataset_sdn.csv")
X_train, X_test, y_train, y_test = preprocess_data(data)
```

```
def preprocess_data(data):
    label_encoder = LabelEncoder()
    data['label'] = label_encoder.fit_transform(data['label'])

    data = pd.get_dummies(data, columns=['src', 'dst', 'Protocol'])

    #print(combined_data.isnull().sum())
    data = data.dropna()

    X = data.drop('label', axis=1)
    y = data['label']

    X_train, X_test, y_train, y_test = train_test_spli
```

2. Training and Evaluation the Model:

- The `train_model_and_evaluate` function is designed to train various machine learning models such as Random Forest, Decision Tree, Logistic Regression, and KMeans.
- The models are trained using the training data.

```
def train_model_and_evaluate(X_train, y_train,method,X_test, y_test):
    methods = {
        'RandomForest': RandomForestClassifier(n_estimators=100, random_state=42),
        'LogisticRegression': LogisticRegression(),
        'KMeans': KMeans(n_clusters=2,n_init=10),  # Specify the number of clusters as needed
        'DecisionTree': DecisionTreeClassifier(),
```

```
    }
    model = methods[method]
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy
```

3. Model Saving and Loading:

- The trained model with the best performance (assuming 'oc_model' was intended to be the best model) is saved using the `save_model` function

```
save_model(model, 'DDos_Normal.pkl')
```

- Later, the saved model can be loaded for further use

```
loaded_model = load_model('DDos_Normal.pkl)
```

4. Prediction for a Single Instance:

- The `predict_instance` function predicts the class (DDoS or Normal) for a single instance using each trained model and then determines the most common prediction.

```
example_instance = [feature_values_for_single_instance]
predict_instance(example_instance)
```

5. FINAL OUTCOME:

We have Compared the Accuracy for different methods at different test-train size.

```
import numpy as np
import matplotlib.pyplot as plt
def evaluate_accuracy(X, y, model, color):
    train_sizes = [0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
    accuracies = []
    for train_size in train_sizes:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1-train_size,
random_state=42)
        model.fit(X_train, y_train)
```

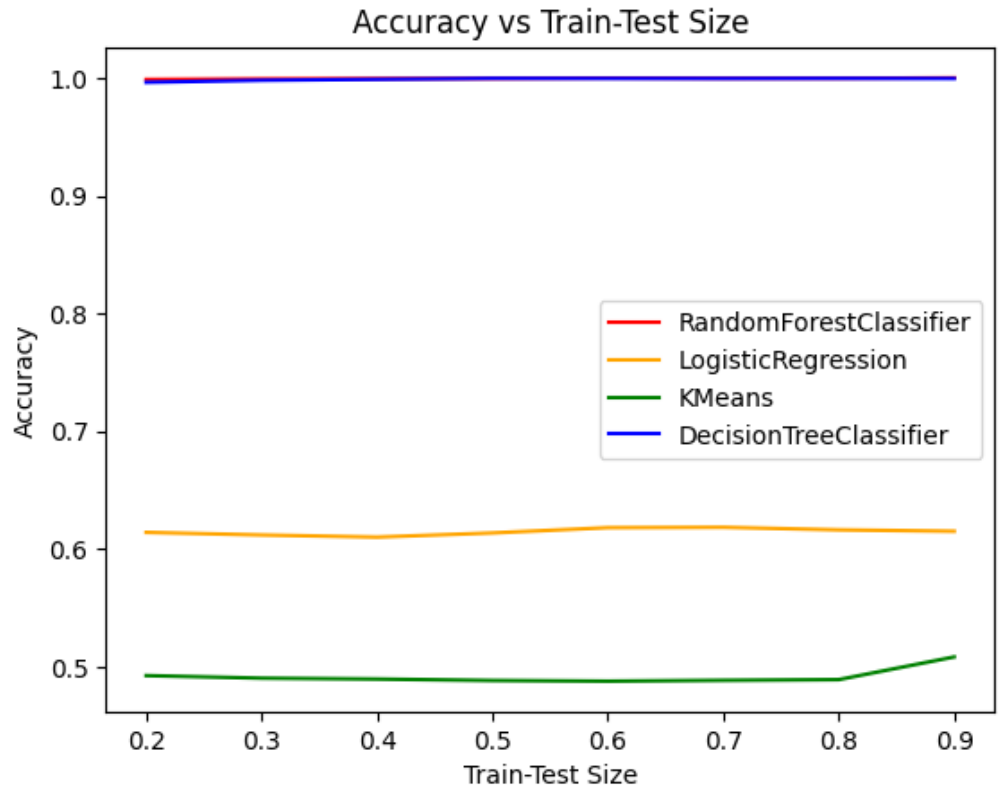
```
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)

    plt.plot(train_sizes, accuracies, label=model.__class__.__name__, color=color)
    plt.xlabel('Train-Test Size')
    plt.ylabel('Accuracy')

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
lr_model = LogisticRegression()
kmeans_model = KMeans(n_clusters=2,n_init=10)  # Specify the number of clusters as needed
dt_model = DecisionTreeClassifier()

evaluate_accuracy(X_train, y_train, rf_model, 'red')
evaluate_accuracy(X_train, y_train, lr_model, 'orange')
evaluate_accuracy(X_train, y_train, kmeans_model, 'green')
evaluate_accuracy(X_train, y_train, dt_model, 'blue')

plt.title('Accuracy vs Train-Test Size')
plt.legend()
plt.show()
```



Hence, we will proceed with the Random Forest Classifier.

{'RandomForest': 1.0, 'LogisticRegression': 0.6050654853620955, 'KMeans': 0.49081278890600927, 'DecisionTree': 0.9996340523882897}
{'RandomForest': 1.0, 'LogisticRegression': 0.7008378274268104, 'KMeans': 0.507752311248074, 'DecisionTree': 1.0}
{'RandomForest': 1.0, 'LogisticRegression': 0.6048407806882383, 'KMeans': 0.49155752439650746, 'DecisionTree': 1.0}

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         2

    accuracy                           1.00         5
   macro avg       1.00      1.00      1.00         5
weighted avg       1.00      1.00      1.00         5
```