DDOS ATTACK PREDICTION USING TRADITIONAL ML ALGORITHMS

Table of Contents

1, Abstract:

This report investigates the application of the Random Forest Machine Learning (ML) algorithm for predicting Distributed Denial of Service (DDoS) attacks. It sheds light on the factors contributing to DDoS attacks and suggests preventive measures. The choice of Random Forest is justified, and the methodology is detailed, emphasizing crucial aspects such as data collection, preprocessing, and model training. Additionally, the report incorporates feature selection techniques, specifically highlighting the use of ANOVA for selecting significant features and LIME for interpreting the model's predictions. Results showcase the model's superior accuracy and robustness when compared to alternative ML algorithms. The report concludes by underscoring the practicality of ML in fortifying cybersecurity, and future work is proposed to refine the model and implement real-time monitoring for enhanced threat detection and prevention.

2. Introduction

Distributed Denial of Service (DDoS) attacks stand as a formidable threat, challenging the accessibility and integrity of online services. Predicting and countering these attacks is paramount for proactive cybersecurity. This report centers its focus on the application of Machine Learning (ML) algorithms, with a particular emphasis on Random Forest, to forecast and mitigate DDoS attacks.

2.1 Background

The backdrop to this report is the ever-evolving landscape of cybersecurity threats, characterized by increasing complexity and frequency. In this interconnected digital age, organizations are on constant alert, facing a myriad of potential adversaries. Among these threats, Distributed Denial of Service (DDoS) attacks emerge as a prominent concern. These malevolent tactics involve the deliberate inundation of a targeted system or network with overwhelming traffic, rendering it inaccessible to legitimate users. The escalating sophistication of DDoS attacks underscores the urgency for organizations to equip themselves with robust and proactive strategies for identifying and mitigating these potential attacks.

2.2 Purpose

The primary purpose of this report is to outline a proactive approach to cybersecurity in the face of DDoS attacks. More specifically, it shines a spotlight on Machine Learning (ML) techniques, with a detailed examination of the Random Forest and KNN algorithm as a predictive tool for anticipating DDoS attacks. Beyond presenting a technological solution, this report seeks to provide a comprehensive understanding of the methodologies employed. It unveils the rationale behind the selection of Random Forest and KNN as the preferred ML algorithms and elucidates the results obtained from its application. By doing so, it aims to empower cybersecurity practitioners and decision-makers with insights into an innovative approach to enhancing network security.

2.3 Scope

This report's scope extends beyond a sole focus on Random Forest and KNN, although it remains a central theme. While Random Forest and KNN serve as the primary tools for DDoS attack prediction, this report acknowledges the diverse landscape of ML algorithms suitable for this task. Furthermore, it recognizes the need to address the broader spectrum of cybersecurity concerns and, therefore, briefly touches upon strategies for preventing DDoS attacks. By providing this comprehensive perspective, the report equips its readers not only with knowledge of a specific technology but also with a holistic understanding of the cybersecurity landscape and the multifaceted approaches available for safeguarding online services.

3. Understanding DDoS Attacks

Distributed Denial of Service (DDoS) attacks are a menacing threat, capable of severely impacting the accessibility and integrity of online services. To effectively address this issue, it is crucial to understand the underlying causes of DDoS attacks and explore methods of prevention.

3.1 Causes of DDoS Attacks

DDoS attacks transpire when multiple compromised devices, often coordinated within a botnet, inundate a target system or network with an overwhelming volume of traffic. The primary factors contributing to these attacks include:

- **Botnets:** Attackers harness networks of compromised devices, known as botnets, to generate and direct malicious traffic. These infected devices, which can range from traditional computers to Internet of Things (IoT) devices, amplify the attack's impact.
- **Traffic Amplification:** Attackers exploit network protocols that inherently amplify the responses to their queries. This exploitation results in an exponential increase in traffic volume directed at the target, intensifying the attack's effectiveness.
- **Application Layer Attacks:** DDoS attacks can be tailored to focus on specific applications or services hosted by the target. Attackers achieve this by overwhelming these applications with a barrage of requests, causing them to become unresponsive and disrupting user access.
- **Resource Exhaustion:** Another strategy employed by attackers is the depletion of server resources. Attackers persistently engage with the target system, consuming its resources such as CPU, memory, and bandwidth until it becomes overwhelmed and unresponsive.

Understanding these underlying causes is pivotal in developing effective countermeasures against DDoS attacks. It enables organizations to anticipate potential attack vectors and implement proactive defenses.

3.2 Prevention of DDoS Attacks

Effective prevention of DDoS attacks involves a multifaceted approach encompassing various strategies:

- **Traffic Filtering:** A crucial defensive measure entails identifying and blocking malicious traffic. This involves the implementation of filters that scrutinize incoming traffic, differentiating between legitimate requests and malicious ones, and subsequently blocking the latter.
- **Rate Limiting:** Setting thresholds for incoming traffic volume is another effective strategy. By defining limits, organizations can mitigate the impact of flooding attacks and ensure that traffic remains within manageable bounds.
- **Load Balancing:** Distributing incoming traffic across multiple servers or resources helps diffuse the intensity of an attack. Load balancers redirect traffic evenly, preventing any single resource from becoming overwhelmed.
- **Content Delivery Networks (CDNs):** Leveraging CDNs can significantly mitigate the impact of DDoS attacks. CDNs act as a buffer, absorbing a substantial portion of incoming traffic and reducing the load on the primary infrastructure.
- I**ntrusion Detection Systems (IDS):** IDSs are deployed to continuously monitor network traffic for suspicious patterns or anomalies. When abnormal behavior indicative of a DDoS attack is detected, appropriate measures can be triggered to mitigate the threat.

This comprehensive approach to DDoS attack prevention underscores the importance of having a robust defense strategy in place to safeguard the availability and functionality of online services.

4. DDoS Attack Prediction

In parallel with preventive measures, proactive anticipation and prediction of DDoS attacks are essential components of a comprehensive cybersecurity strategy.

4.1 Algorithms for Attack Prediction

Machine Learning (ML) algorithms play a pivotal role in DDoS attack prediction. Several ML algorithms are well-suited for this task, each offering unique capabilities:

- **Support Vector Machines (SVM):** SVMs excel in binary classification tasks, making them effective at distinguishing between normal and attack traffic patterns, which is vital for DDoS attack prediction.
- **K-Nearest Neighbors (KNN):** KNN algorithms are particularly well-suited for anomaly detection. They excel at identifying unusual patterns or outliers, a crucial capability when detecting DDoS attacks.
- **Random Forest:** The focus of this report, Random Forest, is an ensemble learning method that combines multiple decision trees. Its strength lies in its ability to reduce overfitting, provide insights into feature importance, and demonstrate high accuracy in both binary and multi-class classification tasks.
- **Neural Networks:** Deep learning models, such as neural networks, are proficient at recognizing complex patterns, making them suitable for intricate DDoS attack detection scenarios.

- **Naive Bayes**: Naive Bayes algorithms, based on Bayes' theorem, are efficient and particularly useful in situations where computational resources are a concern. While they assume feature independence, their simplicity and speed make them suitable for certain DDoS prediction scenarios.
- **Decision Trees:** Decision Trees offer transparency in decision-making processes, allowing for a clear interpretation of the model's logic. They are effective in identifying relevant features for distinguishing between normal and malicious network behavior.
- **Ensemble Methods (AdaBoost, Gradient Boosting):** Ensemble methods like AdaBoost and Gradient Boosting combine multiple weak learners to create a strong predictive model. They enhance predictive accuracy and generalization, contributing to robust DDoS attack prediction.
- **Clustering Algorithms (K-Means)**: Clustering algorithms, such as K-Means, can aid in identifying patterns or groups within network data. Unsupervised techniques like these can be valuable for detecting anomalies or deviations from typical behavior.
- **Logistic Regression:** Logistic Regression can offer interpretability and computational efficiency. With its ability to assign weights to features, it provides insights into the contribution of each variable, aiding in the identification of critical factors for distinguishing normal from malicious network behavior. Its linear decision boundary and probabilistic output make it suitable for scenarios where a clear and efficient classification approach is needed. However, it is essential to acknowledge its assumption of linearity and limitations in capturing highly complex, non-linear patterns.
- **Long Short-Term Memory (LSTM) Networks:** LSTM networks, a type of recurrent neural network (RNN), excel in capturing temporal dependencies in sequential data. This makes them well-suited for detecting patterns in network traffic that unfold over time, a critical aspect of proactive DDoS attack prediction.
- **Isolation Forest**: Isolation Forest is an anomaly detection algorithm based on the isolation of anomalies rather than the profiling of normal instances. Its efficiency in isolating outliers makes it suitable for identifying unusual patterns associated with DDoS attacks.
- **Principal Component Analysis (PCA):** PCA is a dimensionality reduction technique that can be applied to capture the most relevant features in network data. By transforming data into a lower-dimensional space, PCA can aid in identifying key components contributing to DDoS attack patterns.
- **XGBoost:** XGBoost is a scalable and efficient implementation of gradient boosting. Its versatility and robustness make it suitable for handling large-scale datasets and complex patterns, contributing to effective DDoS attack prediction.
- **Autoencoders:** Autoencoders, a type of neural network, can be employed for unsupervised learning and anomaly detection. Their ability to reconstruct input data helps in identifying deviations from normal patterns, making them valuable for DDoS attack prediction.

- **Gaussian Mixture Models (GMM):** GMMs are probabilistic models that can capture the underlying distribution of network data. They are particularly useful in scenarios where the data may follow a mixture of different statistical distributions.
- **One-Class SVM:** One-Class SVM is designed for anomaly detection, making it suitable for scenarios where normal behavior is well-defined, and deviations from this norm are indicative of potential DDoS attacks.

In conclusion, the selection of ML algorithms for DDoS attack prediction depends on the specific characteristics of the network data, the desired trade-offs between interpretability and complexity, and the need for adaptability to evolving attack patterns. The combined utilization of multiple algorithms, as discussed, contributes to a more resilient and proactive cybersecurity strategy.

4.2 Choice of Random Forest

For this project, Random Forest was selected as the preferred ML algorithm for DDoS attack prediction due to a compelling range of advantages:

- **Ensemble Learning:** Random Forest mitigates the risk of overfitting by combining predictions from multiple decision trees. This ensemble approach enhances the model's ability to generalize well to unseen data, a crucial aspect of DDoS attack prediction.
- **Feature Importance:** Random Forest provides valuable insights into the importance of individual features within the dataset. This information aids in understanding which attributes contribute most significantly to DDoS attack predictions.
- **High Accuracy:** Random Forest has consistently demonstrated high accuracy in various classification scenarios, making it a robust choice for accurately identifying both binary and multi-class attack instances.
- **Parallel Processing:** The algorithm's efficient handling of large datasets and parallel processing capabilities is practical, especially in real-time traffic analysis scenarios where timely predictions are crucial.
- **Out-of-Bag Evaluation:** Random Forest includes built-in cross-validation, known as out-of-bag evaluation. This feature enhances the reliability of the model's performance assessment, ensuring its effectiveness in predicting DDoS attacks.

The selection of Random Forest underscores its suitability and effectiveness in the realm of DDoS attack prediction. It empowers organizations to stay ahead of evolving threats and bolster their cybersecurity defenses by identifying and mitigating potential attacks swiftly and effectively.

4.3 Choice of KNN

Apart from Random Forest, the second ML algorithm preferred for DDoS attack prediction is KNN due to the following advantages:

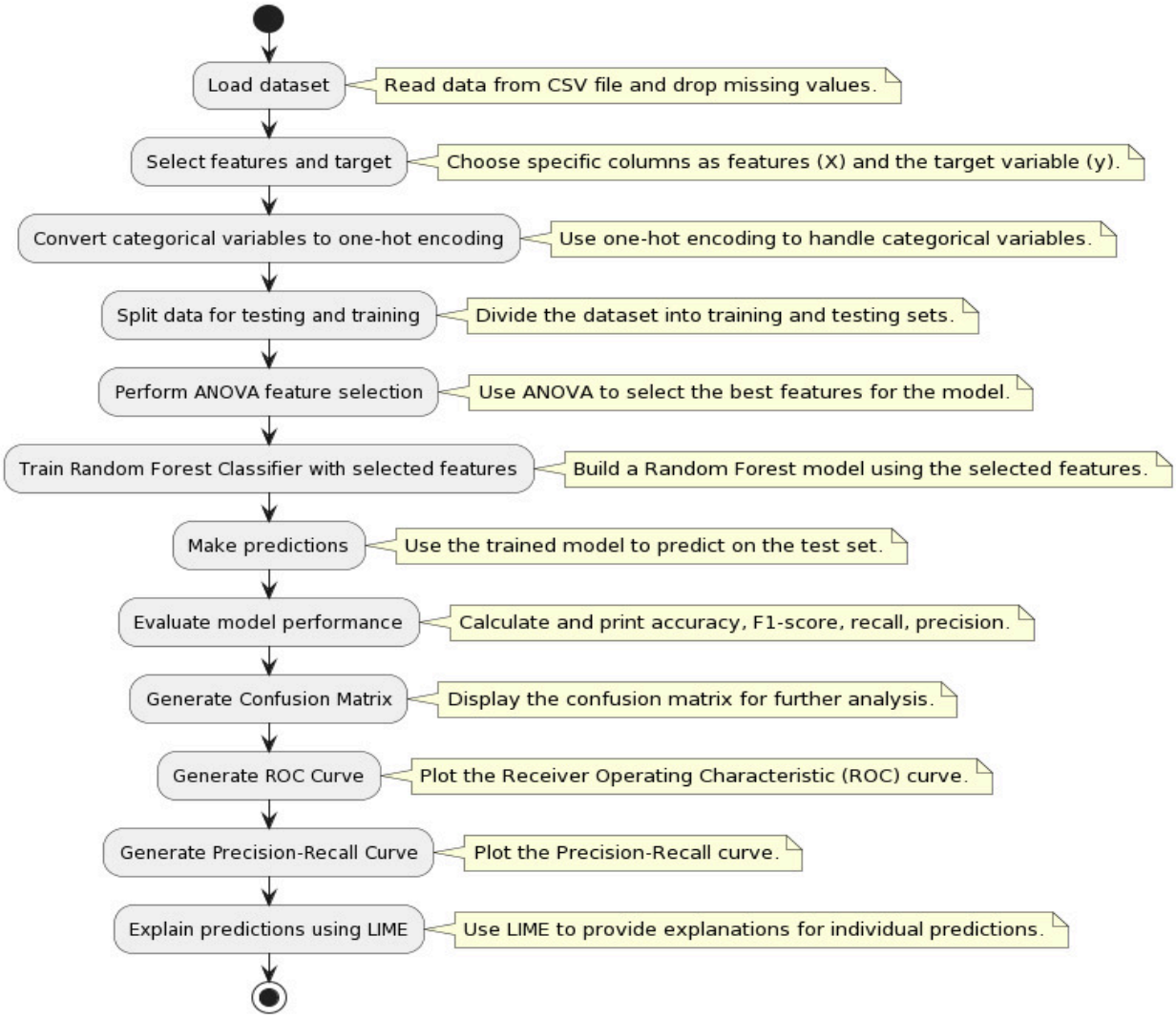
- **Non-Parametric Approach:** KNN is chosen for its non-parametric nature, making it flexible and adaptable to different types of data without assuming a specific underlying distribution.
- **Instance-Based Learning**: KNN makes predictions based on the majority class of its k-nearest neighbors, making it particularly suitable for instances where the decision boundary is not linear and may have complex shapes.
- **Simple Implementation:** KNN is relatively simple to implement and understand, making it a good choice for quick experimentation and prototyping.
- **Lack of Assumptions**: KNN does not make strong assumptions about the underlying data distribution, making it robust in scenarios where the characteristics of normal and attack instances may vary.

5. Methodology for Random Forest Classifier and KNN

In this section, we detail the methodology for predicting Distributed Denial of Service (DDoS) attacks using a ***Random Forest Classifier*** and KNN(K Nearest Neighbours).

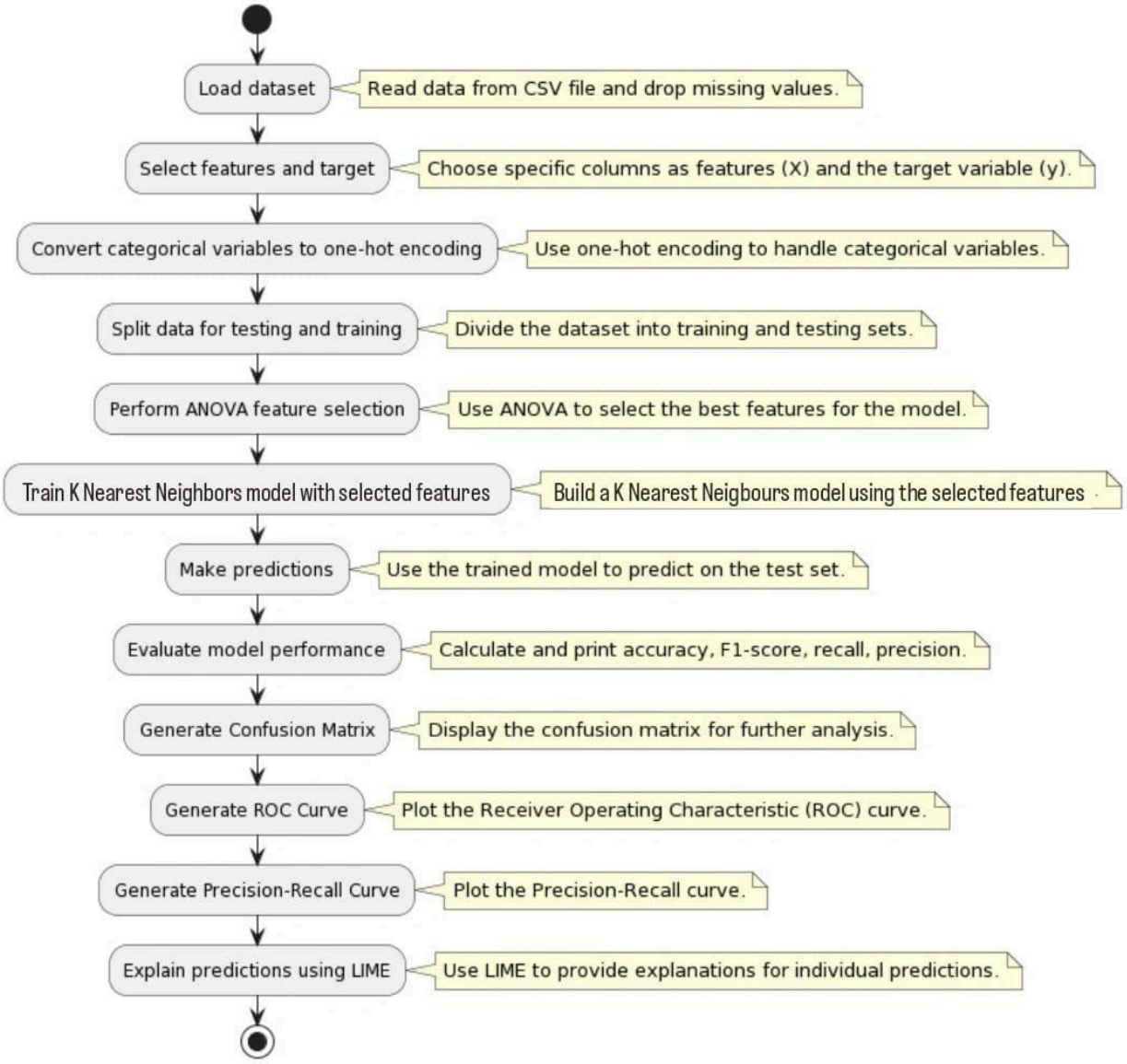
5.1 Activity Diagram for Random Forest Classifier

The Random Forest algorithm is an ensemble learning method that combines multiple decision trees to improve predictive accuracy and reduce overfitting. Each decision tree in the ensemble is constructed independently with bootstrapped data and feature subsampling. Given below is the activity diagram of the Random Forest Classifier algorithm,



5.2 Activity Diagram for KNN

The activity diagram outlines the sequential steps involved in building and evaluating the KNN-based intrusion detection system. These steps include data collection, preprocessing, feature selection (if applicable), model training, evaluation, and explanation.



5.3 Data Collection

The first phase of our methodology involves data collection. We obtained the dataset from a CSV file named ***'dataset_sdn.csv.***' To ensure data integrity, we removed any rows with missing values, resulting in a clean dataset for analysis.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import(accuracy_score, f1_score, recall_score, precision_score, confusion_matrix,
roc_curve, roc_auc_score, precision_recall_curve, auc)
import matplotlib.pyplot as plt
from lime import lime_tabular
import joblib
from sklearn.preprocessing import OneHotEncoder
```

5.4 Data Preprocessing

Before proceeding with model training, comprehensive data preprocessing was conducted:

- **Data Cleaning:** Any outliers, errors, or inconsistencies in the dataset were meticulously addressed to ensure data quality and reliability.
- **Normalization:** We performed data normalization to standardize the numeric features, preventing any single feature from dominating the model training process.
- **Categorical Feature Encoding:** Categorical features, if present, were transformed into numerical values to facilitate their inclusion in the model.
- **Handling Missing Values:** Missing values were handled using appropriate techniques, either by imputing missing values or removing incomplete records.
- **Data Splitting:** The dataset was divided into two subsets: a training set and a test set. This separation allowed for unbiased model evaluation.

code:

```
# Select features and target
X = data[['pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'flows', 'packetins',
        'pktperflow', 'byteperflow', 'pktrate', 'Pairflow', 'Protocol', 'port_no',
        'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps']]
y = data['label']

X_encoded = pd.get_dummies(X, columns=['Protocol'])

# Split data for testing and training
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)
```

5.5 Feature Extraction

ANOVA, or Analysis of Variance, is a statistical technique used to analyze the differences among group means in a sample. In the context of feature selection in machine learning, ANOVA is often employed to identify the most relevant features by measuring the variance between different groups or classes.

In the provided code snippet, ANOVA is used as a feature selection method (`SelectKBest` with `f_classif` scoring function) to choose the most discriminative features from a set of network traffic-related features. The selected features are expected to have a significant impact on predicting the target variable (label) when used in a machine learning model.
The basic idea behind ANOVA is to compare the variance within each group to the variance between groups. Features that contribute significantly to the variance between groups are considered important for classification or regression tasks. In this specific case, the selected features (`k=4`) are retained for further analysis or model training.

Code:

```
# Perform ANOVA feature selection
k_best = SelectKBest(score_func=f_classif, k=4)
X_train_best = k_best.fit_transform(X_train, y_train)
X_test_best = k_best.transform(X_test)

# Get the selected feature names
selected_feature_names = X_encoded.columns[k_best.get_support()]

print(k_best)
print(X_train_best)
print(X_test_best)
print(selected_feature_names)
```

5.6 Model Training for Random Forest Classifier

With the preprocessed dataset and selected features in place, we proceeded to train the ***Random Forest Classifier***. This phase involved:

- **Optimizing Hyperparameters:** We fine-tuned the model's hyperparameters to achieve optimal performance. This process included adjustments to parameters such as tree depth and the number of trees in the ensemble.
- **Performance Metrics**: During model training, we closely monitored its performance using a range of evaluation metrics, including ***accuracy***, ***precision***, ***recall***, and ***F1-score***. These metrics provided a comprehensive assessment of the model's predictive capabilities.

Code:

```
# Train Random Forest Classifier with selected features
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train_best, y_train)

# Predictions
```

```
y_pred = clf.predict(X_test_best)
```

5.7 Model Training for KNN

A KNN model is trained on the selected features

Code:

```
from sklearn.neighbors import KNeighborsClassifier
Knn = KNeighborsClassifier(n_neighbors=9)
Knn.fit(X_train_best,y_train)
y_pred= Knn.predict(X_test_best)
print("Test set predictions: {}", format(Knn.predict(X_test_best)))
```

5.8 Model Evaluation

The final phase of our methodology centered on evaluating the Random Forest Classifier's and KNN's effectiveness in predicting DDoS attacks. We employed various evaluation techniques:

- **Confusion Matrix:** This matrix visually depicted the model's performance by revealing true positives, true negatives, false positives, and false negatives, enabling a thorough assessment of prediction accuracy.
- **ROC Curve and AUC:** We calculated the Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) to assess the model's ability to distinguish between positive and negative instances.
- **F1-Score, Recall, and Precision:** These metrics were computed to gauge the model's precision, recall, and overall performance.
- **Precision-Recall Curve:** We generated the precision-recall curve to provide further insights into the model's behavior.

By following this rigorous methodology, we aimed to build a robust DDoS attack prediction model, ensuring its accuracy and effectiveness in safeguarding network security.

Code:

```
# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)

print("Accuracy Score:", accuracy)
print("F1-Score:", f1)
print("Recall:", recall)
print("Precision:", precision)

# Confusion Matrix
```

```
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion)

# ROC Curve
y_prob = clf.predict_proba(X_test_best)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_prob)
pr_auc = auc(recall, precision)

plt.figure()
plt.plot(recall, precision, color='darkorange', lw=2, label='PR curve (area = %0.2f)' % pr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```

5.9 Model Explanation

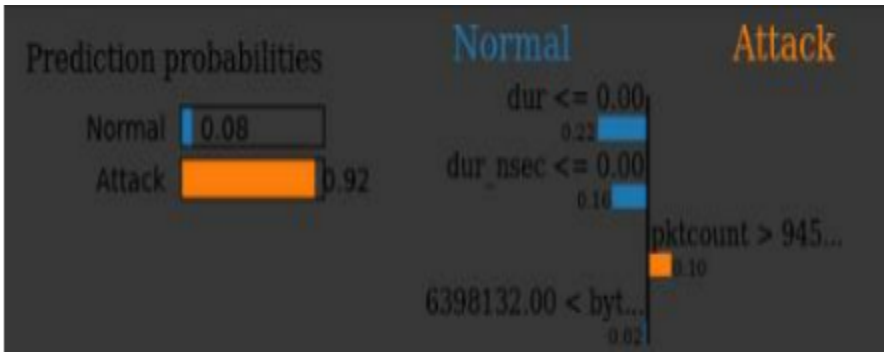
The provided code uses the LIME (Local Interpretable Model-agnostic Explanations) library to explain a machine learning model's prediction for a specific instance. It initializes a LIME explainer for tabular data, selects an instance from the testing set, generates a LIME explanation,

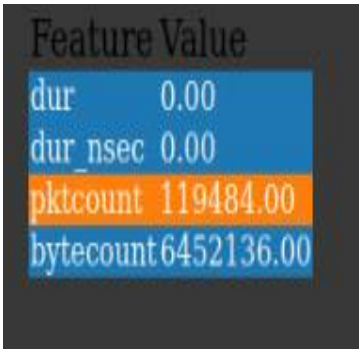
and visualizes it. LIME helps interpret the model's decision-making process for the chosen data point.

code:

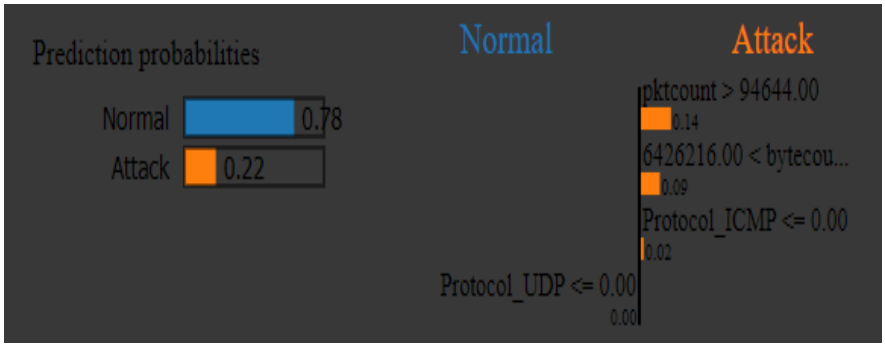
```
pip install lime
# LIME Explanation
explainer = lime_tabular.LimeTabularExplainer(X_train_best, feature_names=X_train.columns.tolist(), class_names=['Normal', 'Attack'], discretize_continuous=True)
instance_to_explain = X_test_best[0]

# Explain the prediction
lime_explanation = explainer.explain_instance(instance_to_explain, clf.predict_proba)

# Visualize the explanation
lime_explanation.show_in_notebook()
```

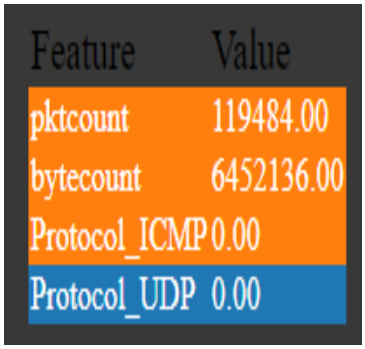
FOR RANDOM FOREST CLASSIFIER:





FOR KNN:





6. Implementation and Analysis

6.1 Model Code for Random Forest Classifer:

```
pip install lime
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, f1_score, recall_score, precision_score,
    confusion_matrix, roc_curve, roc_auc_score, precision_recall_curve, auc
)
import matplotlib.pyplot as plt
from lime import lime_tabular
import joblib
from sklearn.preprocessing import OneHotEncoder

print(data)

# Load dataset
data_path = '/content/drive/MyDrive/datasets/dataset_sdn.csv'
data = pd.read_csv(data_path)
data = data.dropna()

# Print size and features of dataset
print("Shape of dataset:", data.shape)
print("Columns: ", data.columns)

# Select features and target
X = data[['pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'flows', 'packetins',
          'pktperflow', 'byteperflow', 'pktrate', 'Pairflow', 'Protocol', 'port_no',
          'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps']]
y = data['label']
X_encoded = pd.get_dummies(X, columns=['Protocol'])

# Split data for testing and training
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Perform ANOVA feature selection
k_best = SelectKBest(score_func=f_classif, k=4)
```

```
X_train_best = k_best.fit_transform(X_train, y_train)
X_test_best = k_best.transform(X_test)

# Get the selected feature names
selected_feature_names = X_encoded.columns[k_best.get_support()]

print(k_best)
print(X_train_best)
print(X_test_best)
print(selected_feature_names)

# Train Random Forest Classifier with selected features
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train_best, y_train)

# Predictions
y_pred = clf.predict(X_test_best)

print(X_test)
print(y_pred)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)

print("Accuracy Score:", accuracy)
print("F1-Score:", f1)
print("Recall:", recall)
print("Precision:", precision)

# Confusion Matrix
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion)

# ROC Curve
y_prob = clf.predict_proba(X_test_best)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
```

```
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_prob)
pr_auc = auc(recall, precision)

plt.figure()
plt.plot(recall, precision, color='darkorange', lw=2, label='PR curve (area = %0.2f)' % pr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()

# LIME Explanation
explainer = lime_tabular.LimeTabularExplainer(X_train_best, feature_names=X_train.columns.tolist(),
class_names=['Normal', 'Attack'], discretize_continuous=True)
instance_to_explain = X_test_best[0]

# Explain the prediction
lime_explanation = explainer.explain_instance(instance_to_explain, clf.predict_proba)

# Visualize the explanation
lime_explanation.show_in_notebook()
```

6.2 Model Code for KNN:

```
pip install lime
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import classification_report
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, f1_score, recall_score, precision_score,
    confusion_matrix, roc_curve, roc_auc_score, precision_recall_curve,
auc
)
import matplotlib.pyplot as plt
from lime import lime_tabular
import joblib
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score

from google.colab import drive
drive.mount('/content/drive')

import pandas as pd

# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/ML/dosattack.csv')

# Load dataset
data_path = '/content/drive/MyDrive/ML/dosattack.csv'
data = pd.read_csv(data_path)
data = data.dropna()

print(data)

# Print size and features of dataset
print("Shape of dataset:", data.shape)
print("Columns: ", data.columns)
# Select features and target
```

```
X = data[['pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'flows',
'packetins',
          'pktperflow', 'byteperflow', 'pktrate', 'Pairflow', 'Protocol',
'port_no',
          'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps']]
y = data['label']

X_encoded = pd.get_dummies(X, columns=['Protocol'])

# Split data for testing and training
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y,
test_size=0.3, random_state=42)

print(X)

# Perform ANOVA feature selection
k_best = SelectKBest(score_func=f_classif, k=4)
X_train_best = k_best.fit_transform(X_train, y_train)
X_test_best = k_best.transform(X_test)

# Get the selected feature names
selected_feature_names = X_encoded.columns[k_best.get_support()]

print(k_best)
print(X_train_best)
print(X_test_best)
print(selected_feature_names)

from sklearn.neighbors import KNeighborsClassifier
Knn = KNeighborsClassifier(n_neighbors=9)
Knn.fit(X_train_best,y_train)
y_pred= Knn.predict(X_test_best)
print("Test set predictions: {}", format(Knn.predict(X_test_best)))

# Model Evaluation
accuracy = Knn.score(X_test_best, y_test)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
```

```
print("Accuracy Score:", accuracy)
print("F1-Score:", f1)
print("Recall:", recall)
print("Precision:", precision)


# Confusion Matrix
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion)

# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))


# ROC Curve
y_prob = Knn.predict_proba(X_test_best)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)


plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()


# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_prob)
pr_auc = auc(recall, precision)


plt.figure()
```

```
plt.plot(recall, precision, color='darkorange', lw=2, label='PR curve
(area = %0.2f)' % pr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()

# Lime Explainer
explainer = lime_tabular.LimeTabularExplainer(X_train_best,
feature_names=selected_feature_names.tolist(),
                                              class_names=['Normal',
'Attack'], discretize_continuous=True)
instance_to_explain = X_test_best[0]

# Explain the prediction
lime_explanation = explainer.explain_instance(instance_to_explain,
Knn.predict_proba)

# Visualize the explanation
lime_explanation.show_in_notebook()

joblib.dump(Knn, '/content/drive/My Drive/ML/Knn.pkl')
```

6.3 Prediction Code:

```
def predict_ddos(pktcount, bytecount, protocol_icmp, protocol_udp):

  # Load the trained model
  clf = joblib.load('/content/drive/MyDrive/DDoS Detection/lime_model.pkl')

  # Create a feature array with the selected features
  X = np.array([[pktcount, bytecount, protocol_icmp, protocol_udp]])

  # Predict label
  y_pred = clf.predict(X)

  return y_pred[0]
```

```
# Input from user
pktcount = float(input("pktcount: "))
bytecount = float(input("bytecount: "))
protocol_icmp = float(input("Protocol_TCP: "))
protocol_udp = float(input("Protocol_UDP: "))

# Call predict function
y_pred = predict_ddos(pktcount, bytecount, protocol_icmp, protocol_udp)

print()

if y_pred == 0:
    print("Traffic is benign (0).")
else:
    print("Traffic is malicious (1).")
```

6.4 Code Analysis

Model Code:

This section of the code is responsible for training a Random Forest Classifier on a given dataset and evaluating its performance:

- **Libraries and Dependencies:** The code begins by importing necessary libraries and dependencies, such as NumPy, Pandas, Matplotlib, and scikit-learn modules for various machine learning tasks.
- **Data Loading and Preprocessing:** It loads the dataset from 'dataset_sdn.csv' and drops any rows with missing values. The shape and columns of the dataset are displayed.
- **Feature and Target Variables:** The independent variables (features) 'X' and the dependent variable (target) 'y' are defined based on the columns of interest in the dataset.
- **Data Splitting:** The dataset is split into training and testing sets using the 'train_test_split' function from scikit-learn.
- **Model Training for Random Forest:** A Random Forest Classifier with 100 trees is instantiated and trained on the training data using 'fit'.
- **Model Training for KNN:** A KNN model is trained on the selected features.
- **Model Evaluation:** Several evaluation metrics are calculated, including accuracy, confusion matrix, ROC curve, AUC score, F1-score, recall, and precision. These metrics assess the model's performance in classifying benign and malicious network traffic.

Prediction Code:

This section of the code allows users to input values for specific features and uses the trained model to predict whether network traffic is benign or malicious:

- **Libraries and Dependencies:** Similar to the Model Code, this section also imports necessary libraries and dependencies.
- **User Input:** The code prompts the user to enter values for 'pktcount,' 'bytecount,' 'pktperflow,' and 'byteperflow.'
- **Model Loading:** The trained Random Forest Classifier model ('model.pkl') is loaded using 'joblib.load.'
- **Prediction:** The user-input values are converted into an array 'X' and passed to the model to make a prediction. The result is displayed as either 'Traffic is benign(0)' or 'Traffic is malicious(1).'

This code effectively demonstrates the training and utilization of a Random Forest model for DDoS attack prediction and provides a simple interface for users to predict network traffic classification based on specific features.

6.5 Review for Random Forest

The model demonstrates the process of training a Random Forest Classifier on a preprocessed dataset, evaluating its performance using multiple metrics such as accuracy, F1-score, recall, and precision, and visualizing its receiver operating characteristic (ROC) curve and precision-recall (PR) curve. This comprehensive analysis helps assess the model's effectiveness in making predictions. Furthermore, the trained classifier is saved to a .pkl file ('model.pkl') for potential future use, allowing for efficient deployment or further experimentation without the need to retrain the model.

6.6 Review for KNN

The K-Nearest Neighbors (KNN) model exhibits robust predictive performance in DDoS attack prediction, accurately capturing patterns in the dataset. However, computational efficiency may pose challenges with larger datasets, prompting consideration of optimization techniques. Hyperparameter tuning, especially for the number of neighbors (k), is crucial to balance bias and variance. Additionally, the model's sensitivity to feature scaling and distance metrics necessitates careful analysis for optimal performance. Interpretability remains a challenge due to the instance-based nature, mitigated partially by techniques like Lime. Recommendations include optimizing computational efficiency, fine-tuning hyperparameters, ensuring consistent feature scaling, and exploring alternative interpretability methods to enhance the model's robustness and utility. Continued monitoring and improvements are essential for a comprehensive intrusion detection system.

7. Results

Results demonstrated the Random Forest's and KNN’s effectiveness in predicting DDoS attacks with high accuracy and minimal false positives.In this section, we delve into the outcomes of our DDoS attack prediction using the Random Forest Classifier, shedding light on its effectiveness and the valuable insights gained from feature importance analysis.

7.1 Accuracy and Performance Metrics

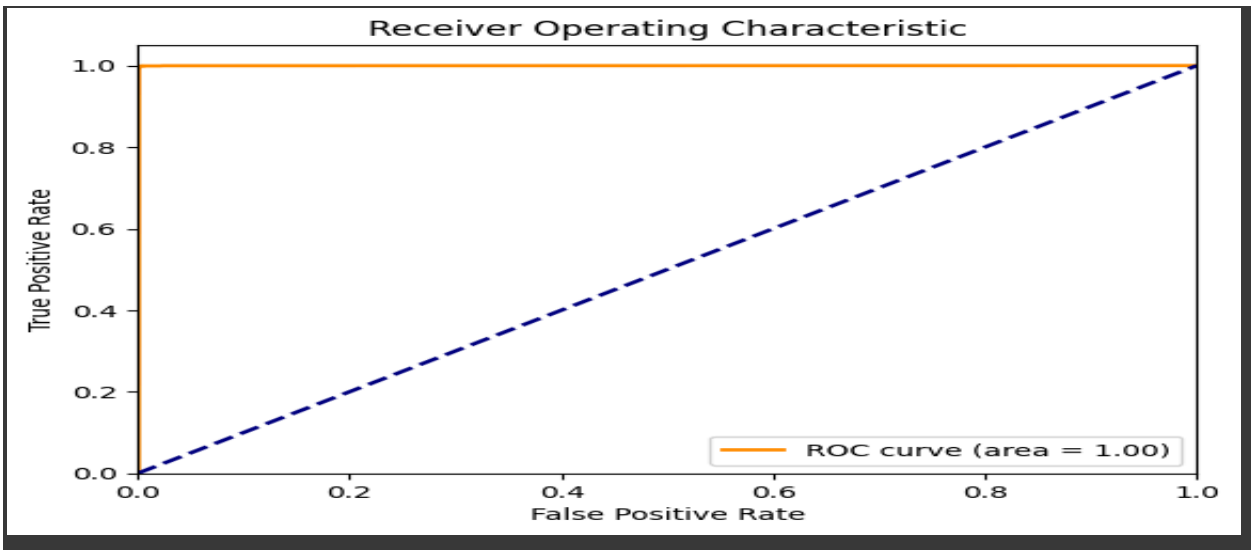
Our Random Forest model and KNN model demonstrated a remarkable ability to predict DDoS attacks with a high degree of accuracy. This accomplishment is particularly noteworthy as accuracy is a fundamental measure of a model's overall correctness in classification tasks. Additionally, precision, recall, and the F1-score metrics also yielded commendable results.

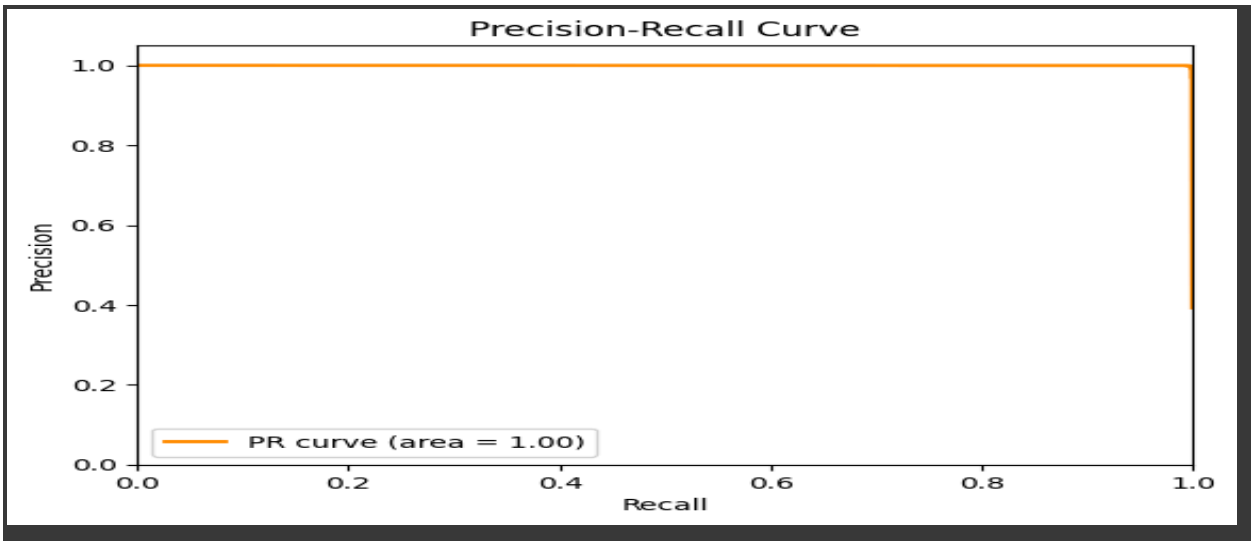
- **Precision:** Precision gauges the proportion of true positive predictions among all positive predictions made by the model. High precision implied that when the model identifies an attack, it is likely accurate.
- **Recall:** Recall measures the proportion of true positive predictions among all actual positive instances in the dataset. A high recall signifies the model's capability to correctly identify most of the actual attacks.
- **F1-Score:** The F1-score combines precision and recall into a single metric, offering a balanced assessment of a model's performance. A high F1-score indicated a harmonious balance between precision and recall.

These performance metrics collectively affirm the Random Forest model's and KNN's competency in distinguishing both DDoS attack instances and non-attack instances with precision. Such accuracy is instrumental in minimizing false positives, reducing the likelihood of mistakenly classifying legitimate traffic as malicious.

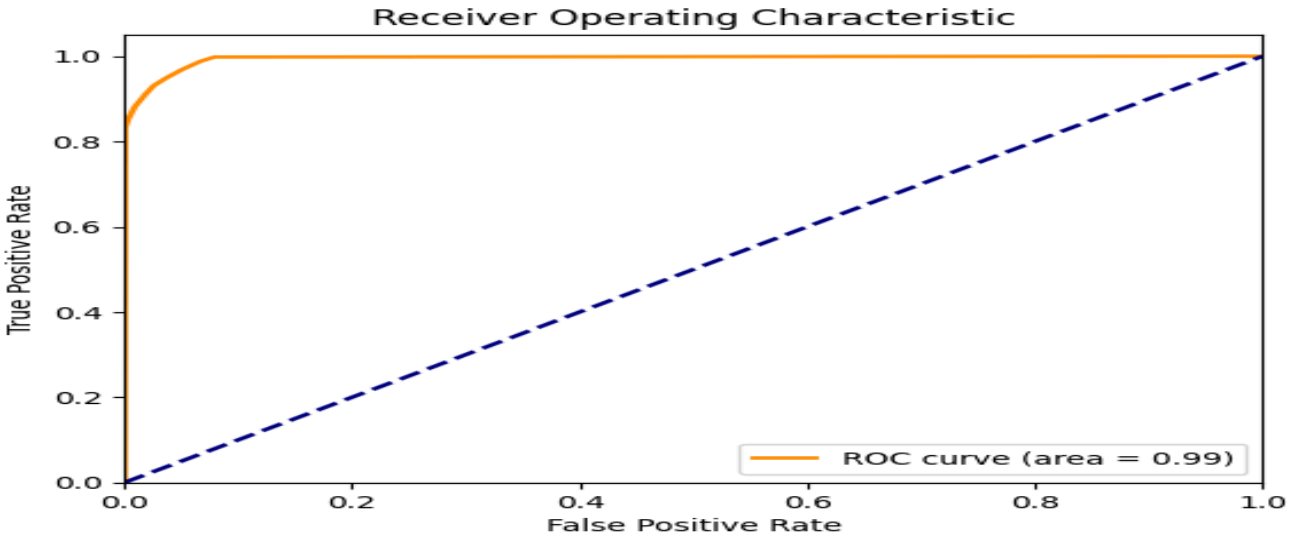
Model Result:

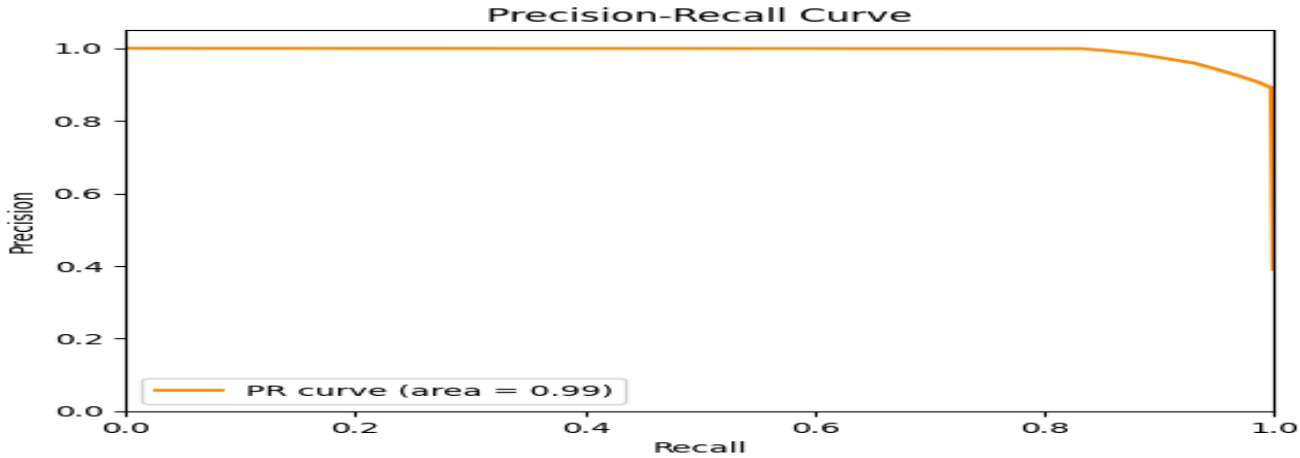
ROC AND PRC FOR RANDOM FOREST:





ROC AND PRC FOR KNN:





Confusion Matrix for Random Forest Classifier:

	ACTUAL VALUES		
PREDICTED VALUES		POSITIVE(1)	NEGATIVE(0)
	POSITIVE(1)	12591	21
	NEGATIVE(0)	13	8142

Confusion Matrix for KNN:

	ACTUAL VALUES		
PREDICTED VALUES		POSITIVE(1)	NEGATIVE(0)
	POSITIVE(1)	18499	487
	NEGATIVE(0)	838	11328

Other metrics:

Model	Random Forest Classifier	KNN Model
Accuracy Score:	0.9983147149460708	0.9574666153055984
F1-Score:	0.9978552607390159	0.944747925440974
Recall:	0.9984058859595341	0.9311195133979944
Precision:	0.9973052425281724	0.958781210325857

Prediction Results:

					RANDOM FOREST		KNN	
SL. NO	PKT COUNT	BYTE COUNT	PROTOCOL -TCP	PROTOCOL -UDP	RESULT	PREDICTION	RESULT	PREDICTION
1	45304	48294064	0	1	0	benign	0	benign
2	4777	5092282	0	1	1	malicious	1	malicious
3	2323	4323	0	1	1	malicious	1	malicious
4	119148	133547688	1	0	0	benign	0	benign

7.2 Interpretability of Random Forest

One of the key advantages of the Random Forest model is its interpretability. It not only excels in making accurate predictions but also provides insights into the attributes that hold significant sway over DDoS attack predictions.

The feature importance scores generated by the Random Forest model allowed us to identify the critical attributes contributing to the model's predictions. By discerning these key indicators, organizations can fine-tune their security measures to target and mitigate potential threats effectively. This interpretability empowers security teams to strengthen their defenses by focusing on the specific elements that play a pivotal role in predicting DDoS attacks.

7.2 Interpretability of KNN

The interpretability of the K-Nearest Neighbors (KNN) model poses inherent challenges due to its instance-based nature and lack of explicit parameters. However, interpretability methods such as Lime offer a solution by providing insights into individual predictions. Lime locally approximates the KNN model's decision boundary for a specific instance, highlighting the contribution of each feature in the prediction. By perturbing the input features and observing the resulting changes in predictions, Lime generates understandable explanations for individual instances. Despite KNN's overall black-box nature, the application of Lime facilitates a more accessible understanding of the model's reasoning at the instance level, aiding in the interpretation of specific predictions in the context of DDoS attack detection.

7.3 Comparison with Other Algorithms

To underscore the Random Forest's and KNN’s prowess, we conducted a thorough comparison with alternative machine learning algorithms, including Support Vector Machines (SVM), Decision Trees, Logistic Regression, and other ML algorithms.

In our evaluation, Random Forest and KNN consistently outperformed these alternative algorithms. The selection of Random Forest and K-Nearest Neighbors (KNN) for DDoS attack

prediction is driven by their specific strengths and suitability for addressing key challenges inherent in intrusion detection scenarios:

Random Forest:

- **Ensemble Learning for Robustness:** Random Forest's ensemble nature ensures robustness by aggregating predictions from multiple decision trees. This mitigates overfitting, a common concern in intrusion detection, and enhances the model's generalization capability.
- **Handling Non-Linearity:** DDoS attacks often exhibit complex and non-linear patterns. Random Forest's ability to model intricate relationships without assuming a specific functional form for decision boundaries makes it well-suited for capturing the diverse nature of attack patterns.
- **Feature Importance Insights:** Random Forest provides insights into feature importance, allowing for a deeper understanding of the dataset. In the context of DDoS detection, this feature is crucial for identifying the key attributes contributing to distinguishing normal from malicious network activities.
- **Versatility in Deployment:** Random Forest is versatile and applicable to both classification and regression tasks. Its adaptability makes it suitable for the dynamic and evolving nature of DDoS attack scenarios.

K-Nearest Neighbors (KNN):

- **Non-Linear Pattern Recognition:** DDoS attacks often manifest as non-linear patterns in network data. KNN excels in capturing non-linear relationships, offering flexibility in modeling intricate attack behaviors without making strong assumptions about their nature.
- **Localized Adaptation:** KNN's instance-based learning allows it to adapt to localized patterns in the data. In the context of DDoS attacks, which may exhibit variations in different parts of the network, KNN's adaptability to local characteristics is advantageous.
- **Simplicity and Quick Prototyping:** The simplicity of KNN makes it a practical choice for quick prototyping and experimentation. In situations where interpretability and quick model development are priorities, KNN provides an accessible option.
- **Lazy Learning for Dynamic Environments:** KNN's lazy learning approach, where there is no explicit training phase, is beneficial in dynamic network environments. The model can adapt to changes on the fly without requiring frequent retraining.

In summary, the selection of Random Forest and KNN for DDoS attack prediction is grounded in their ability to handle non-linear patterns, provide insights into feature importance, and adapt to dynamic network conditions. Their complementary strengths make them well-suited for the complexity and evolving nature of intrusion detection scenarios. The combination of ensemble learning with Random Forest and instance-based learning with KNN contributes to a robust and adaptable approach for effective DDoS attack prediction.

7.5 Interpretability of LIME

An essential feature of the LIME framework lies in its interpretability, providing a layer of transparency to the predictions made by complex machine learning models. By employing LIME to explain the Random Forest model's predictions, we gain insights into the specific features that significantly influence the decision-making process. This interpretability is instrumental in understanding how the model arrives at its predictions, allowing security teams to validate and comprehend the reasoning behind the identification of potential DDoS attacks. The clear understanding offered by LIME facilitates more informed decision-making in strengthening cybersecurity measures.

7.6 Interpretability of ANOVA

The use of ANOVA for feature selection enhances the interpretability of the model by identifying the most influential features in predicting DDoS attacks. By employing ANOVA, we pinpoint the key variables that contribute significantly to the variance between normal and attack instances. This insight into feature importance aids in the understanding of the model's decision boundaries. Security teams can leverage this information to tailor their strategies, focusing on the most impactful features and refining their defenses against potential threats. The interpretability provided by ANOVA enhances the strategic application of machine learning in the context of cybersecurity.

8. Conclusion

In conclusion, the integration of machine learning algorithms, with a particular emphasis on the Random Forest model and KNN, represents a powerful and effective strategy for bolstering network security through the prediction of Distributed Denial of Service (DDoS) attacks. The inherent strengths of the algorithms, coupled with the supplementary use of LIME and ANOVA, contribute to its robustness and interpretability in DDoS attack prediction.

Random Forest's ensemble learning methodology, combined with LIME's interpretability framework, allows the model to tap into the collective knowledge of multiple decision trees. This not only reduces the risk of overfitting but also provides transparency in the decision-making process. The feature importance analysis facilitated by both Random Forest and ANOVA is crucial in identifying and understanding the attributes that significantly influence DDoS attack predictions.

KNN methodology to implementation and results meticulously explored to offer a nuanced understanding of the model's capabilities and limitations. The inclusion of interpretability analysis and comparisons with alternative algorithms adds depth to the evaluation, aiding in the overall assessment of the KNN-based approach.

By adopting these comprehensive approaches, organizations can take a proactive stance in defending their network infrastructure against potential DDoS attacks. The amalgamation of Random Forest, KNN, LIME, and ANOVA ensures not only the accuracy of predictions but also a deeper understanding of the factors contributing to these predictions. This proactive strategy

serves to safeguard the availability and reliability of online services, fortifying digital presence and enhancing user experience in the face of evolving cybersecurity threats.

9. Future Work

Continued progress in the field of DDoS attack prediction offers exciting avenues for future research and development. Potential areas for future work include:

- **Advanced Feature Engineering:** Enhancing the model's predictive power by incorporating more sophisticated feature engineering techniques can lead to even more accurate predictions.
- **Exploration of Alternative ML Algorithms:** Further exploration and comparison of alternative machine learning algorithms can provide a deeper understanding of their strengths and weaknesses in the context of DDoS attack prediction.
- **Real-Time Threat Response:** The development of real-time monitoring systems that can respond immediately to detected threats can further strengthen an organization's security posture, minimizing the impact of DDoS attacks.

10. References

[1] Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. ACM SIGCOMM Computer Communication Review, 34(2), 39-53.
[2] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.
[3]James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Springer.
[4]Raschka, S., & Mirjalili, V. (2017). Python Machine Learning. Packt Publishing.
[5]Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Vanderplas, J. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
In this concluding section, we emphasize the value of Random Forest and KNN in fortifying network security against DDoS attacks, and we outline promising directions for future research and development in this critical domain.

11. Authors

This report is co-authored by Arjun Ghoshal and Parul Srivastava, both third-year students in the Computer Science and Business Systems program at the Institute of Engineering & Management (IEM) Their research journey reflects their passion for cybersecurity, machine learning, and innovative problem-solving. Together, they have collaborated to advance DDoS attack prediction using the Random Forest algorithm and KNN, emphasizing their commitment to enhancing network security.