Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

15th October 2025. Vol. 5. No. 1; e-ISSN: 2808-4519

Classification of Distributed Denial Service (DDoS) Attacks Using the K-Nearest Neighbor (KNN) Method

R.Adinda Hamba Banju[1]*, Fajar Hariadi[2], Raynesta Mikaela Indri Malo[3]

[1,2,3]*Informatics Engineering Study Program, Wira Wacana Christian University Sumba*
adindahambabanju@gmail.com[1*]

Abstract

Distributed Denial of Service (DDoS) attacks pose a significant security threat by disrupting network services through a flood of data traffic or exploiting system vulnerabilities. Early detection of DDoS attacks is essential to reduce their potential impact. This study aims to classify DDoS attacks using the K-Nearest Neighbor (KNN) algorithm to improve network security. The research data is sourced from a publicly available DDoS Software-Defined Networking (SDN) dataset. The research stages include data collection, pre-processing, implementation of the KNN algorithm, and model evaluation. Data pre-processing involves data cleansing, feature transformation, and normalization to optimize model performance. The KNN algorithm determines the number of Ks of the nearest neighbor and uses geometric distances to classify DDoS attacks. The conclusion of this study assesses the accuracy of the KNN model in detecting DDoS attacks. The results of the evaluation showed that the KNN model reached a level of accuracy

Keywords: *Network Security, Distributed Denial of Service (DDoS), K-Nearest Neighbor (KNN), Attack Classification*

1. Introduction

Network security is an important thing to pay attention to, especially in today's technological era, because almost all business, communication, and public service activities depend on computer networks. At this time, human needs are highly dependent on the existence of information or data, especially information or digital data. The greater the need for information, the more incidents or security disturbances to network systems will increase. Therefore, computer system security is needed to prevent such attacks [1]. A computer network attack is an attempt to gain unauthorized access to an organization's or individual network for the purpose of searching for data or performing other malicious activities. Without adequate security sensitive data can be stolen, services can be compromised, and financial and reputational losses can occur. Therefore, maintaining network security is a top priority for organizations and companies.

One of the most dangerous security threats is Distributed Denial of Service (DDOS) attacks. DDOS attacks work by flooding the target server or network with very high traffic, resulting in overload and disrupting service availability. The way a DDOS attack works typically involves thousands to millions of devices that have been infected with malware and are controlled by an attacker, known as a botnet. These botnets send fake requests massively to the target server or network, causing resources such as bandwidth, CPU, and memory to become overwhelmed and services become unresponsive or even paralyzed. Throughout 2024, Indonesia faces a significant surge in cyberattacks with nearly 19.2 million web-based attack attempts and 330 cases of digital attacks reported, an increase compared to the previous year. Especially Distributed Denial Of Service (DDOS) attacks, Indonesia is one of the countries with the highest number of attacks in the world, recorded to receive around 260 billion attacks from 2023 to mid-2024 [2]. The largest attacks reached 1.7 Tbps which mostly targeted the financial, e-commerce, media, and government sectors. This data shows that cyber threats in Indonesia are getting bigger and more complex, so strengthening network security is very important to protect data, services, and reputations of organizations and governments.

The effects of DDOS attacks are particularly detrimental to the attacked networks and services. The immediate impact is that service interruptions that can last are service interruptions that can last from a few minutes to days, causing users to be unable to access important services such as websites, applications, or online transaction systems, furthermore DDOS attacks are sometimes used as a distraction tool to launch other cyber attacks such as data theft or system intrusion [3]. To address and prevent the adverse effects of DDOS attacks, early detection of attacks is critical. The characteristics of DDOS attacks can be detected through several indicators and variables that include several features that are used to classify more accurately including the type of Protocol, number of connections, connection duration, packet size, connection duration, IP source, IP destination, frequency of request, and connection status [4].

In the context of detecting and classifying DDOS attacks, the K-Nearest Neighbor (KKN) algorithm is one of the effective machine learning methods. KNN is a basic and easy-to-implement machine learning technique this algorithm is used for classification methods that use the distances between data points to predict labels from unknown data [5]. KNN works by classifying data based on feature similarity with the data of its nearest neighbors in the feature space. This algorithm calculates the geometric distance, such as Euclidean distance, between the

data to be tested and the training data, and then determines a class based on the majority of the nearest neighbor's K. Thus, KNN can recognize DDOS attack patterns based on the variables mentioned earlier [6].

Based on the above problems, the author conducted a study on "Classification of Distributed Denial of Service (DDOS) Attacks Using the K-Nearest Neighbor (KNN) Method. This research was conducted to measure the accuracy of the KNN algorithm in classifying DDOS attacks using data sets derived from public data sets such as DDOS SDN. By preprocessing data which includes cleaning, feature transformation, and normalization, it is expected to be an important and effective solution to detect the increasing DDOS attacks early in Indonesia and globally. So that the KNN model can work optimally in accurately detecting DDOS attacks, as well as contributing to digital business people, government agencies, and the cybersecurity community and strengthening network security systems and maintaining the availability of critical services [7].

2. Library Studies

2.1. Network Security

Network security is a collection of devices designed to protect data when it is transmitted against the threat of access, alteration, and obstruction by unauthorized parties [8].

2.2. Distributed Denial Of Service (DDOS)

DDOS is a type of attack that aims to hinder legitimate users' access to a service or network by flooding data traffic or exploiting a system security loophole [9].

2.3. K-Nearest Neighbor (KNN)

K-Nearest Neighbor (KNN) is a method that uses a supervised algorithm in which the results of a newly query instance are classified based on the majority of the K-NN category [10].

3. Research Methodology

At this stage, the researcher creates a diagram flow to present the process systematically and logically in the course of the research carried out.

Figure 1: Research Flow

3.1. Data Collection

This study uses SDN DDOS set data obtained from the Kaggle platform. This data set, created by Aiken Kazin, contains 104,345 network traffic data entries from a Software Defined Network (SDN) environment, with 23 features that include packet, byte, duration, protocol, and other network information statistics. Each entry is labeled a class (0 for normal, 1 for DDoS attacks). The scraping process is carried out using the Python programming language which is run through the Google Colab tool.

3.2. Preproccesing Data

The data preprocessing process is an important step that aims to prepare the raw data so that it is ready for use in the machine learning model. This process has several stages, each of which has a specific goal of improving data quality and consistency.

a) Data cleansing is identifying and fixing missing, duplicate, or outlier values to make the data set more accurate and consistent.
b) Feature Transformation is turning raw data into more meaningful representations.
c) Data normalization, which equalizes the scale of various features, does not dominate the model results and accelerates the convergence of distance-based algorithms or gradient descent.

3.3. Feature Selection

The selection of features is carried out based on the analysis of the correlation and relevance of the features to the classification of DDOS attacks. The selected feature must be able to effectively distinguish between normal traffic patterns and attacks.

3.4. Implementation of KNN

The K-Nearest Neighbor (KNN) method is used to classify DDOS attack data based on proximity to previous data. This algorithm works as follows:

a) Data preparation is preparing a data set containing features (numerical features) and labels (class/target)
b) Determine the value of K the number of nearest neighbors i.e. K = 3
c) Calculate distance i.e. using Euclidean distance
d) Choose a nearby neighbor

e) Majority Vote
f) The majority class prediction of the neighbors is set as a result
g) Model evaluation
Measuring how accurate the model is in predicting the data is done by comparing the prediction results with the actual data using metrics such as accuracy, precision, recall, and F1-score.
h) Presentation of Results
The stage of presenting results is the final step in the process of implementing the classification model, where the results of the model evaluation are presented clearly and informatively.

4. Results and Discussion

4.1. Data Collection

This study uses SDN DDOS set data obtained from the Kaggle platform. This data set, created by Aiken Kazin, contains 104,345 network traffic data entries from a Software Defined Network (SDN) environment, with 23 features that include packet, byte, duration, protocol, and other network information statistics. Each entry is labeled a class (0 for normal, 1 for DDOS attacks). The data is collected through SDN network monitoring and labeled by security experts, ensuring the credibility and relevance of the data for DDOS attack detection. This data set has gone through initial validation and cleanup, making it a reliable source for analysis. This data is executed and executed using the python programming language inside the google colab.

```
# import the necessary packages
from os import defpath
df = pd.read_csv("ddos.csv", delimiter=';')
df.head(3)
```

Figure 2: Script Reading Data Set

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps	tot_kbps	label
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	3917	0	0	0.0	0.0
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	3520	0	0	0.0	0.0
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	1242	0	0	0.0	0.0

3 rows × 23 columns

Figure 3: Display Data Set

4.2. Preproccesing Data

The data preprocessing process is an important step that aims to prepare the raw data so that it is ready for use in the machine learning model. This process has several stages, each of which has a specific goal of improving data quality and consistency.

a) Deleting Null Data
The goal is to clean the data set from rows that have an empty value (Null/NaN) so that the data used is of quality and free from interference during modeling. Below is the script to run the stages of deleting null data and the results of the stages.

```
df = df.dropna()
df.isnull().sum()
```

Figure 4: Data Preproccesing (Script Deletes Null Data)

```
<class 'pandas.core.frame.DataFrame'>
Index: 103839 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   dt           103839 non-null  int64
 1   switch       103839 non-null  int64
 2   src          103839 non-null  object
 3   dst          103839 non-null  object
 4   pktcount     103839 non-null  int64
 5   bytecount    103839 non-null  int64
 6   dur          103839 non-null  int64
 7   dur_nsec     103839 non-null  int64
 8   tot_dur      103839 non-null  float64
 9   flows        103839 non-null  int64
 10  packetins    103839 non-null  int64
 11  pktperflow   103839 non-null  int64
 12  byteperflow  103839 non-null  int64
 13  pktrate      103839 non-null  int64
 14  Pairflow     103839 non-null  int64
 15  Protocol     103839 non-null  object
 16  port_no      103839 non-null  int64
 17  tx_bytes     103839 non-null  int64
 18  rx_bytes     103839 non-null  int64
 19  tx_kbps      103839 non-null  int64
 20  rx_kbps      103839 non-null  int64
 21  tot_kbps     103839 non-null  float64
 22  label        103839 non-null  float64
dtypes: float64(3), int64(17), object(3)
memory usage: 19.0+ MB
```

Figure 5: Data Preproccesing (Null Data Removal Result)

b) One Hot Encoding

This stage converts categorical features such as src, etc., and protocols into numerical representations by creating a new binary column for each unique category, resulting in a value of 1 if the row has that category, and if 0 if it doesn't. This avoids the false sequence assumptions that can arise if categories are labeled as direct numbers, and makes the data compatible with the KNN algorithm requiring numerical input to calculate Euclidean distances. Below is a script image of the one hot encoding stage and the results of the stages.

```
df = df.drop(columns=['Pairflow'])

# Menggunakan one-hot encoding untuk kolom 'src'
df_encoded = pd.get_dummies(df, columns=['src', 'dst', 'Protocol'])

df_numeric = df_encoded.astype(int)
df_encoded.info()
print(df_encoded)
```

Figure 6: Preproccesing Data (Script One Hot Encoding)

```
<class 'pandas.core.frame.DataFrame'>
Index: 103839 entries, 0 to 104344
Data columns (total 59 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   dt             103839 non-null  int64
 1   switch         103839 non-null  int64
 2   pktcount       103839 non-null  int64
 3   bytecount      103839 non-null  int64
 4   dur            103839 non-null  int64
 5   dur_nsec       103839 non-null  int64
 6   tot_dur        103839 non-null  float64
 7   flows          103839 non-null  int64
 8   packetins      103839 non-null  int64
 9   pktperflow     103839 non-null  int64
 10  byteperflow    103839 non-null  int64
 11  pktrate        103839 non-null  int64
 12  port_no        103839 non-null  int64
 13  tx_bytes       103839 non-null  int64
 14  rx_bytes       103839 non-null  int64
 15  tx_kbps        103839 non-null  int64
 16  rx_kbps        103839 non-null  int64
 17  tot_kbps       103839 non-null  float64
 18  label          103839 non-null  float64
 19  src_10.0.0.1   103839 non-null  bool
 20  src_10.0.0.10  103839 non-null  bool
 21  src_10.0.0.11  103839 non-null  bool
 22  src_10.0.0.12  103839 non-null  bool
 23  src_10.0.0.13  103839 non-null  bool
 24  src_10.0.0.14  103839 non-null  bool
 25  src_10.0.0.15  103839 non-null  bool
```

Figure 7: Data Preproccesing (One Hot Encoding Results)

c) Correlation Heatmap

This stage is to identify highly correlated features (which can be removed due to redundancy) and features that have a high correlation with the target, thus helping to minimize the data dimension, avoid multicollinearity, and improve the efficiency and accuracy of the KNN model. Below are the scripts and results of the heatmap correlation stages.

```
correlation_matrix = df_numeric.corr()

plt.figure(figsize=(30, 20))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap sebelum Preprocessing")
plt.show()
```

Figure 8: Data Preproccesing (Script Correlation Heatmap)

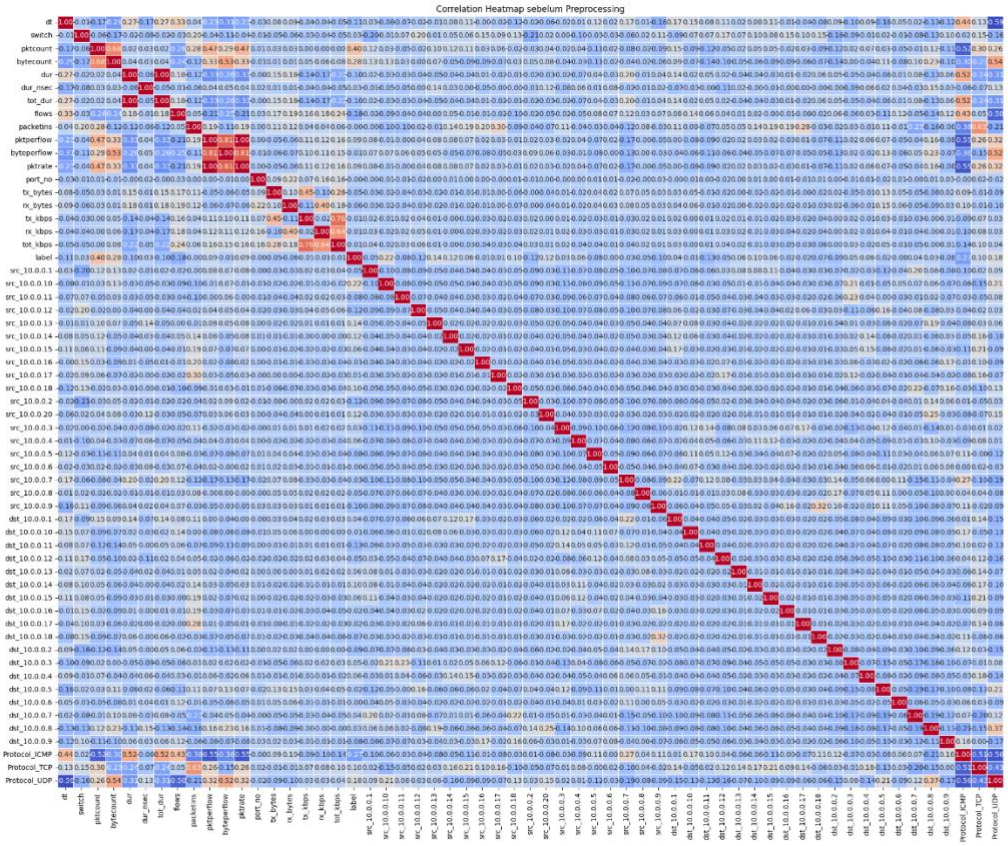


Figure 9: Data Preprocesing (Correlation Heatmap Results)

1. Data Cleanup is identifying and correcting missing, duplicate, or outlier values to make the data set more accurate and consistent. Below are the scripts and results of the data cleansing stages.

```
# Drop duplicate data
df.drop_duplicates( keep ="first", inplace =True)
df.info()
```

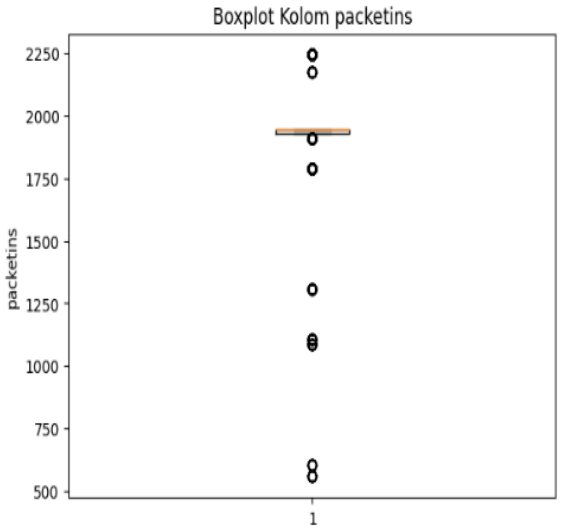
```
<class 'pandas.core.frame.DataFrame'>
Index: 67051 entries, 0 to 104343
Data columns (total 47 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   dt              67051 non-null  int64
 1   dur_nsec        67051 non-null  int64
 2   flows           67051 non-null  int64
 3   packetins       67051 non-null  int64
 4   rx_kbps         67051 non-null  int64
 5   tot_kbps        67051 non-null  int64
 6   label           67051 non-null  int64
 7   src_10.0.0.1    67051 non-null  int64
 8   src_10.0.0.10   67051 non-null  int64
 9   src_10.0.0.11   67051 non-null  int64
 10  src_10.0.0.12   67051 non-null  int64
 11  src_10.0.0.13   67051 non-null  int64
 12  src_10.0.0.14   67051 non-null  int64
 13  src_10.0.0.15   67051 non-null  int64
 14  src_10.0.0.16   67051 non-null  int64
 15  src_10.0.0.17   67051 non-null  int64
 16  src_10.0.0.18   67051 non-null  int64
 17  src_10.0.0.2    67051 non-null  int64
 18  src_10.0.0.20   67051 non-null  int64
 19  src_10.0.0.3    67051 non-null  int64
 20  src_10.0.0.4    67051 non-null  int64
 21  src_10.0.0.5    67051 non-null  int64
 22  src_10.0.0.6    67051 non-null  int64
 23  src_10.0.0.7    67051 non-null  int64
 24  src_10.0.0.8    67051 non-null  int64
 25  src_10.0.0.9    67051 non-null  int64
 26  dst_10.0.0.1    67051 non-null  int64
 27  dst_10.0.0.10   67051 non-null  int64
 28  dst_10.0.0.11   67051 non-null  int64
 29  dst_10.0.0.12   67051 non-null  int64
 30  dst_10.0.0.13   67051 non-null  int64
 31  dst_10.0.0.14   67051 non-null  int64
 32  dst_10.0.0.15   67051 non-null  int64
 33  dst_10.0.0.16   67051 non-null  int64
 34  dst_10.0.0.17   67051 non-null  int64
 35  dst_10.0.0.18   67051 non-null  int64
 36  dst_10.0.0.2    67051 non-null  int64
 37  dst_10.0.0.3    67051 non-null  int64
 38  dst_10.0.0.4    67051 non-null  int64
 39  dst_10.0.0.5    67051 non-null  int64
 40  dst_10.0.0.6    67051 non-null  int64
 41  dst_10.0.0.7    67051 non-null  int64
 42  dst_10.0.0.8    67051 non-null  int64
 43  dst_10.0.0.9    67051 non-null  int64
 44  Protocol_ICMP   67051 non-null  int64
 45  Protocol_TCP    67051 non-null  int64
 46  Protocol_UDP    67051 non-null  int64
dtypes: int64(47)
memory usage: 24.6 MB
```

(a) (b)

Figure 10: Preprocessing (a) Data Cleanup Script (b) Data Cleanup Display

```
import matplotlib.pyplot as plt

# kolom 'dt'
plt.boxplot(df['dt'])
plt.title('Boxplot Kolom dt')
plt.ylabel('dt')
plt.figure(figsize=(2, 2))
plt.show()
# kolom 'dur_nsec'
plt.boxplot(df['dur_nsec'])
plt.title('Boxplot Kolom dur_nsec')
plt.ylabel('dur_nsec')
plt.figure(figsize=(2, 2))
plt.show()
# kolom 'flows'
plt.boxplot(df['flows'])
plt.title('Boxplot Kolom flows')
plt.ylabel('flows')
plt.figure(figsize=(2, 2))
plt.show()
# kolom 'packetins'
plt.boxplot(df['packetins'])
plt.title('Boxplot Kolom packetins')
plt.ylabel('packetins')
plt.figure(figsize=(2, 2))
plt.show()
# kolom 'rx_kbps'
plt.boxplot(df['rx_kbps'])
plt.title('Boxplot Kolom rx_kbps')
plt.ylabel('rx_kbps')
plt.figure(figsize=(2, 2))
plt.show()
# kolom 'tot_kbps'
plt.boxplot(df['tot_kbps'])
plt.title('Boxplot Kolom  tot_kbps')
plt.ylabel(' tot_kbps')
plt.figure(figsize=(2, 2))
plt.show()
```

(a)



(b)

```
from scipy import stats
from scipy.stats import zscore
import numpy as np

# Menghitung Z-score untuk setiap kolom dalam DataFrame
z_scores = df.apply(zscore)  # Menghitung Z-score hanya untuk kolom numerik

# Menentukan threshold Z-score untuk mengidentifikasi outlier
threshold = 3

# Menggunakan absolute Z-score untuk menyaring data tanpa outlier
df_no_outliers = df[(z_scores.abs() < threshold).all(axis=1)]  # Menyaring baris yang tidak memiliki outlier pada semua kolom

print(df_no_outliers[['label']])
```

(c) (d)

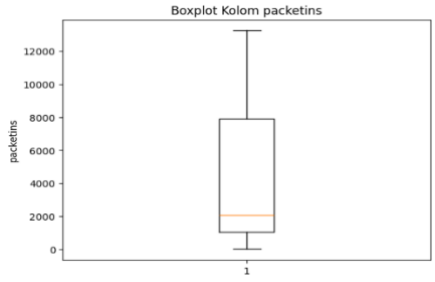


Figure 10: Preprocessing (a) Script Viewing Outlier Data (b) Outlier Data Display (C) Script Omitting Outlier Data (D) Display Omitting Outlier Data.

2. Data Normalization

Data normalization is the process of changing the scale of a feature to be within a uniform range, usually between 0 and 1. This is important to ensure all features have equal weight in the analysis, especially for scale-sensitive algorithms. Common methods include Min-Max Scaling and Z-score Standardization, which help the learning model be more effective and improve.

```
from sklearn.preprocessing import MinMaxScaler

# Exclude 'label' from normalization if it exists and is the target variable
numerical_features = df_no_outliers.select_dtypes(include=[np.number]).columns.tolist()
if 'label' in numerical_features:
    numerical_features.remove('label')

scaler = MinMaxScaler()
df_normalized = df_no_outliers.copy()
df_normalized[numerical_features] = scaler.fit_transform(df_normalized[numerical_features])
print(df_normalized[numerical_features].head())
```

(a)

```
          dt  dur_nsec  flows  packetins   rx_kbps  tot_kbps  src_10.0.0.1  \
1520  0.16623  0.105316    0.0   0.083428  0.000000  0.000000           0.0
1523  0.16623  0.105316    0.0   0.083428  0.000000  0.499961           0.0
1527  0.16623  0.105316    0.0   0.083428  0.810546  0.499961           0.0
1529  0.16623  0.105316    0.0   0.083428  0.000000  0.797104           0.0
1530  0.16623  0.105316    0.0   0.083428  0.481733  0.297142           0.0

      src_10.0.0.10  src_10.0.0.11  src_10.0.0.12  ...  dst_10.0.0.3  \
1520            0.0            0.0            0.0  ...           0.0
1523            0.0            0.0            0.0  ...           0.0
1527            0.0            0.0            0.0  ...           0.0
1529            0.0            0.0            0.0  ...           0.0
1530            0.0            0.0            0.0  ...           0.0

      dst_10.0.0.4  dst_10.0.0.5  dst_10.0.0.6  dst_10.0.0.7  dst_10.0.0.8  \
1520           0.0           0.0           0.0           1.0           0.0
1523           0.0           0.0           0.0           1.0           0.0
1527           0.0           0.0           0.0           1.0           0.0
1529           0.0           0.0           0.0           1.0           0.0
1530           0.0           0.0           0.0           1.0           0.0

      dst_10.0.0.9  Protocol_ICMP  Protocol_TCP  Protocol_UDP
1520           0.0            0.0           0.0           1.0
1523           0.0            0.0           0.0           1.0
1527           0.0            0.0           0.0           1.0
1529           0.0            0.0           0.0           1.0
1530           0.0            0.0           0.0           1.0

[5 rows x 46 columns]
```

(b)

Figure 10: Preprocessing (a) Data Normalization Script (b) Data Normalization Display

4.3. Feature Selection

The selection of features is carried out based on the analysis of the correlation and relevance of the features to the classification of DDOS attacks. The selected feature must be able to effectively distinguish between normal traffic patterns and attacks. The main features used include dt, dur_nsec, flows, packetins, rx_kbps, total_kbps. From the initial set of data, 23 features were then selected and as many as 6 features were selected.

4.4. Implementation of KNN

The selection of the K-Nearest Neighbors (KNN) Implementation feature in this context involves several key steps that have been taken, and here is an explanation of each step:

1. Data Preparation:

   The data set used is df_no_outliers, where the 'label', 'src', and 'etc' columns are removed to get the relevant features. The 'label' column is used as the classification target.

2. Data Sharing:

   The data is divided into two parts: training data (80%) and test data (20%) using train_test_split functions. This ensures that the model can be trained on one subset of data and tested on a different subset to avoid overfitting.

3. KNN Model Training:

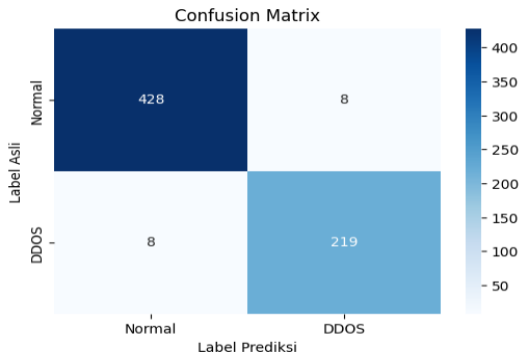
   The KNN model was made using 3 closest neighbors (n_neighbors= 3). The model is trained using trained data by calling the fit method.

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd

y = df_normalized['label']  # Label (kolom 'label')
X = df_normalized.drop(['label'], axis=1)

# Gabungkan X dan y menjadi satu DataFrame sementara untuk memastikan konsistensi baris
df_combined = pd.concat([X, y], axis=1)

# Hapus baris yang mengandung NaN di X atau y
df_combined = df_combined.dropna()

# Pisahkan kembali X dan y setelah pembersihan
X = df_combined.drop(['label'], axis=1)  # Mengambil kembali X tanpa kolom 'label'

y = df_combined['label']  # Mengambil kembali y

# 2. **Membagi data menjadi data latih dan data uji (80% latih, 20% uji)**
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. **Melatih model KNN**
knn = KNeighborsClassifier(n_neighbors=3)  # Menggunakan 3 tetangga
knn.fit(X_train, y_train)  # Melatih model dengan data pelatihan

y_pred = knn.predict(X_test)
```

Figure 10: KNN Implementation Script

4.5. Model Evaluation

This stage is the final step in the process of implementing the classification model, where the results of the model evaluation are presented clearly and informatively. At this stage, performance metrics such as accuracy, precision, recall, and f1-score are displayed to provide a comprehensive picture of how well the model performs at classifying.



(a)

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

           0       0.98      0.98      0.98       436
           1       0.96      0.96      0.96       227

    accuracy                           0.98       663
   macro avg       0.97      0.97      0.97       663
weighted avg       0.98      0.98      0.98       663

Data Latih: 2652
Data Uji  : 663
Akurasi   : 0.9759
Presisi   : 0.9759
Recall    : 0.9759
F1-Score  : 0.9759
```

(b)

Figure 10: Model Evaluation (a) Confusion Matrix Results (b) KNN Implementation Results

The results of the classification report above show the performance of the KNN model in detecting DDOS attacks on test data as many as 663 samples, with a division of 436 for the "Normal" class and 227 for the "DDOS" class. An accuracy of 0.98 for class "0" (Normal), and 0.96 for class "1" (DDOS), means that 98% of "Normal" predictions and 96% of "DDOS" predictions are accurate. The recall is also 0.98 and 0.96, respectively, indicating high sensitivity in detecting both classes. The F1 score (harmonic between precision and recall) was recorded at 0.98 for "Normal" and 0.96 for "DDOS", reflecting a good balance between false positives and false negatives. The overall accuracy is 0.98, reflecting that about 98% of all predictions are correct. Macro avg (simple average) and weighted avg (measured average according to sample count) show values of around 0.97–0.98, indicating performance stability across classes. Overall, this KNN model is capable of detecting DDoS attacks with very high accuracy and sensitivity, demonstrating excellent classification quality on the test set data.

5. Conclusion

Based on the evaluation results, the K Nearest Neighbors (KNN) model showed excellent performance in detecting DDoS attacks, with an accuracy of 97.6% and precision metrics, recall, and F1-score in the range of 96–98%. The pre-processing process that includes data cleanup (eliminating duplication, outliers), one-hot encoding categorical features, normalization, and heatmap-based feature selection plays an important role in forming a clean, consistent, and fully numerical data set. This allows the KNN to work optimally because there are no features that dominate the distance calculation. This success is in line with findings in the literature that simple ML models such as Logistic Regression, Random Forest, and KNN are effective for DDoS detection in the context of real networks.

To strengthen the detection capabilities of the KNN model in detecting DDoS attacks, it is further recommended to implement advanced feature selection (e.g. PCA, chi-square, or multi-filter) to reduce dimensions and eliminate irrelevant features, as research shows that this step can improve accuracy by close to 100%. In addition, it is advisable to compare and test alternative models such as Random Forest, SVM, XGBoost or ensemble hybrid because survey results and meta papers say that RF and XGBoost often outperform KNN in terms of accuracy and reliability consistently. Finally, it is critical to implement k-fold cross-validation and hyperparameter tuning, as well as address data imbalance (via SMOTE or undersampling), to ensure that the model is not overfitted and its performance remains stable under various real-world data set conditions.

Confession

The author would like to thank Wira Wacana Christian University Sumba and the Informatics Engineering Study Program for the support and facilities provided during this research process. Thank you also to the supervisors and colleagues who have helped in collecting and processing data, so that this research can be completed properly.

Reference

[1] Y. Ariyanto, V. A. H. Firdaus, and H. Pramana, "Classification of DOS and Probing Attack Types on IDS Using the K- Nearest Neighbor Method," *Sem. Inform. Apps. Polynesian*, vol. 3, no. ISSN 2460-1160, pp. 1–5, 2020.

[2] M. Alfi, "Cyber Security Risk Analysis in the Digital Transformation of Public Services in Indonesia," *J. Study. Strat. Nas Resilience.*, vol. 6, no. 2, 2023, doi: 10.7454/jkskn.v6i2.10082.

[3] L. Sari, M. N. Faiz, and A. W. Muhammad, "Comparison of Machine Learning Approaches in Detection of DDoS Attacks on Computer Networks," *Infotek*, vol. 16, no. 1, pp. 153–159, 2025, doi: 10.35970/infotekmesin.v16i1.2556.

[4] M. Fluoride Fibrianda and A. Bhawiyuga, "Comparative Analysis of the Accuracy of Attack Detection in Computer Networks Using Naïve Bayes Method and Support Vector Machine (SVM)," *J. Pengemb. Technology. Inf. and Computing Science.*, vol. 2, no. 9, pp. 3112–3123, 2018, [Online]. Available: http://j-ptiik.ub.ac.id

[5] M. Iqbal, R. Rohmat Saedudin, and M. Fathinuddin, "Comparative Analysis of K-Nearest Neighbor and Naïve Bayes Accuracy for Classification of Computer Network Attack Data," *EDUSAINTEK J. Education, Science and Technology.*, vol. 9, no. 3, pp. 920–929, 2022, doi: 10.47668/edusaintek.v9i3.611.

[6] M. F. E. Erlangga, N. Fahriani, and ..., "Detection of Syn Flood Attacks on Servers Using the K-Nearest Neightbor Algorithm Method," *Sem. Nas. Technology. Inf. Computer Science.*, vol. 2, no. 1, pp. 68–72, 2023, [Online]. Available: https://journal.unilak.ac.id/index.php/Semaster/article/view/18458

[7] D. Surya Prasetyo, K. Auliasari, and M. Ridho Putra Syalabi, "Classification of Network Attacks Using the K-Nearest Neighbour Method on Network History Data," *Pros. SANDIX*, vol. 7, no. 1, pp. 63–71, 2023, doi: 10.36040/seniati.v7i1.7874.

[8] A. Bustami and S. Bahri, "Threats, Attacks and Protection Measures on Network or Information System Security: Systematic Review," *Unistek*, vol. 7, no. 2, pp. 59–70, 2020, doi: 10.33592/unistek.v7i2.645.
[9] Z. I. Sumayyah, S. D. S. Permana, M. Tsabit, and A. Setiawan, "Application and Mitigation of Slowloris Technique in Distributed Denial-of-Service (DDos) Attacks on Illegal Websites with Kali Linux," *J. Internet Softw. Eng.*, vol. 1, no. 2, p. 14, 2024, doi: 10.47134/pjise.v1i2.2694.
[10] J. Supriyanto, D. Alita, and A. R. Isnain, "Application of K-Nearest Neighbor (K-NN) Algorithm for Public Sentiment Analysis of Online Learning," *J. Inform. and Software Engineering*, vol. 4, no. 1, pp. 74–80, 2023, doi: 10.33365/jatika.v4i1.2468.