LAPORAN PROYEK MATA KULIAH
10S3001 - KECERDASAN BUATAN

Classification Data Malware Dengan Menggunakan Metode KNN (K-Nearest Neighbors)

Disusun Oleh:

12S21021 Naomi Elena Lumbanraja

12S21031 Daniel Augustian Girsang

12S21034 Lasni Sinta Uli Simanjuntak

12S21036 Astri Yuliana Siahaan

12S21055 Lamria Magdalena Tampubolon

Tautan GitHub: https://github.com/danielaugust67/Project_CERTAN

Tautan Kaggle: Group 05 | Kaggle

PROGRAM STUDI SARJANA SISTEM INFORMASI

FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO

INSTITUT TEKNOLOGI DEL

DESEMBER 2023

DAFTAR ISI

1. Pendahuluan

1.1 Latar Belakang

Dengan perkembangan internet sangat pesat, tidak sedikit orang yang menggunakan baik itu internet atau alat elektronik. Para pengguna teknologi ini saling berkomunikasi dengan orang lain dan masuk dalam kategori dunia maya. Dengan perkembangan ini, ada banyak ancaman yang dihadapi terhadap keamanan dunia maya.

Diketahui jenis-jenis *malware* yang berkembang yaitu virus, *worm, Trojan Horse, root-kit, spyware, backdoor, botnet, dan adware*, dapat merusak atau mencuri informasi dari komputer tanpa izin. Penelitian-penelitian pun dilakukan dalam upaya untuk mengurangi serangan *cyber*. Banyak peneliti yang berusaha untuk mengurangi serangan *cyber* pada *malware* dengan berbagai pendekatan [1].

Salah satu penelitian yang dilakukan yaitu melalui analisis statis untuk mengenali malware sebagai signature yang diekstrak dari aplikasi [1]. Signature ini berasal dari *payload*, dimana ketika signature ini dikenali, maka signature akan disimpan dalam basis data. Kemudian, signature ini akan digunakan untuk mengklasifikasikan kasus malware yang baru berdasarkan basis data tadi [1]. Melalui penelitian ini, dapat disimpulkan agar dilakukan deteksi untuk mencegah serangan *cyber* dengan efektif.

Keberadaan *malware* ini berakibat juga pada android yang akan rentan terkena serangan. Oleh karena itu, diperlukan adanya analisis untuk mengidentifikasi dan mengklasifikasikan serangan *malware* khususnya pada *software* android. Dalam proyek ini, algoritma K-Nearest Neighbor (KNN) digunakan untuk mengklasifikasikan data serangan *malware* dan *non-malware* di android. Algoritma KNN digunakan karena dengan KNN dapat menangani data non-linier, dan kemudahan dalam implementasinya.

1.2 Tujuan

Tujuan proyek ini adalah mengembangkan model klasifikasi *software malware* dengan menggunakan algoritma *K-Nearest Neighbors* (KNN). Pemodelan klasifikasi ini kemudian akan ditinjau apakah dapat melakukan klasifikasi *malware* dan *non-malware* dengan baik atau kurang baik. Pemodelan dirancang untuk secara otomatis mengklasifikasikan aplikasi sebagai *malware* atau *non-malware* berdasarkan fitur-fitur yang diekstrak dari aplikasi-aplikasi tersebut.

1.3 Manfaat

Manfaat utama dari pengembangan model klasifikasi data malware menggunakan algoritma *K-Nearest Neighbors* (KNN) adalah:

1. Menunjukkan apakah metode KNN dapat melakukan klasifikasi *malware* dan *non-malware* dengan baik.
2. Menampilkan apakah metode ini dapat memberikan tingkat *accuracy* yang baik dalam klasifikasi *malware* dan *non-malware*.

1.4 Ruang Lingkup

Adapun Ruang Lingkup pemodelan klasifikasi *malware* dan *non-malware* dengan Metode KNN:

1. Ruang lingkup proyek ini mencakup beberapa aspek utama, termasuk analisis fitur-fitur malware, pengembangan model KNN, dan evaluasi kinerja model klasifikasi yang dihasilkan. Analisis fitur-fitur malware melibatkan pemahaman mendalam terhadap karakteristik unik yang dapat diekstraksi dari kode atau perilaku program berbahaya.
2. Pengumpulan dataset yang representatif dari berbagai jenis malware yang dapat ditemukan di platform *android*. Dataset ini diperlukan untuk melatih dan menguji model KNN, serta untuk memvalidasi keefektifan model dalam mengklasifikasikan *malware* dan *non-malware*.

1.5 Istilah dan Singkatan

Tabel 1. Istilah

Istilah	Definisi
Malware	Jenis program komputer yang dirancang untuk merusak atau mengakses sistem komputer tanpa izin.
Machine Learning (ML)	Cabang kecerdasan buatan yang memungkinkan komputer belajar dari data dan meningkatkan kinerjanya seiring waktu
K-Nearest Neighbors (KNN)	Algoritma pembelajaran mesin yang mengelompokkan data berdasarkan kemiripan atribut.
Android	Sistem operasi mobile yang dikembangkan oleh Google.
Accuracy	Tingkat kebenaran suatu model klasifikasi atau klasifikasi.
Analisis Fitur	Proses mendalam untuk memahami dan mengekstraksi fitur-fitur unik dari data, dalam konteks ini, dari malware.
Model KNN	Representasi matematis dari algoritma *K-Nearest Neighbors.*
Kinerja Model Klasifikasi	Merujuk pada seberapa baik model tersebut dapat mengklasifikasikan atau memprediksi label kelas yang benar untuk data yang tidak dilihat sebelumnya. Kinerja model umumnya diukur dengan menggunakan berbagai metrik evaluasi, tergantung pada jenis masalah klasifikasi dan preferensi pengguna.
Integritas Data	Kondisi di mana data tetap utuh dan tidak terpengaruh oleh manipulasi atau ancaman.

Malicious dan *Benign*	Istilah "malicious" dan "benign" merujuk pada sifat atau niat dari suatu entitas, seperti program komputer atau aktivitas jaringan.
Missing Value	Data dari suatau kolom atau variable dalam dataset yang tidak terisi.
Scalling Data	Normalisasi Data
Principal Component Analysis (PCA)	Teknik Reduksi Dimensi untuk mengurangi jumlah fitur dalam suatu dataset sambil mempertahankan sebanyak mungkin Informasi.

Tabel 2. Singkatan

Singkatan / Akronim	Kepanjangan
ML	*Machine Learning*
KNN	*K-Nearest Neighbors*
PCA	*Principal Component Analysis*

2. Studi Literatur

2.1. Penelitian Sebelumnya

Pada penelitian yang dilakukan oleh [2], dilakukan klasifikasi *malware* pada *smartphone* yang menggunakan sistem operasi *android*. Penelitian ini membahas tentang bagaimana mengembangkan sistem klasifikasi *malware* yang dapat mendeteksi berbagai jenis *malware*, bagaimana mendeteksi *malware* sebelum proses instalasi melalui beberapa tahap. Tahapan yang dilakukan yaitu dari mengekstrak string dari aplikasi *android* dan mengeksplorasi file *android* manifest.xml, kemudian memisahkan kata kunci string dari *android* manifest.xml, dan terakhir mengklasifikasikan *malware* dan aplikasi sah dengan menggunakan kata kunci dan string sebagai fitur input. Hasil yang diharapkan yaitu kemampuan sistem dalam mengidentifikasi aplikasi berbahaya dan aplikasi aman. Hasil dari penelitian ini didapatkan bahwa KNN merupakan algoritma paling cocok dengan binary dibandingkan dengan algoritma lain dan model yang diusulkan meningkatkan akurasi hingga 85,51%.

Untuk penelitian lainnya dengan menggunakan metode K-Nearest Neighbor yang bertujuan untuk melakukan proses klasifikasi jenis malware dengan nilai akurasi yang lebih besar dan seimbang dibanding dengan riset sebelumnya. Algoritma KNN merupakan algoritma *data mining* yang relatif sederhana yang memiliki kelebihan tingkat akurasi yang tinggi dan dapat menangani data dalam jumlah besar. Sehingga dalam penelitian ini akan digunakan nilai K dengan akurasi yang paling tinggi dalam melakukan klasifikasi malware [3].

Selain itu, dari penelitian sebelumnya yang membahas tentang klasifikasi penyakit demam dengan menggunakan metode KNN. Dikatakan bahwa metode KNN mempelajari pola dari data hasil pemeriksaan sebelumnya, dengan berfokus pada 15 gejala penyakit. Hasil akhir dari proses ini digunakan untuk menentukan kelas klasifikasi berdasarkan nilai K yang telah ditentukan sebelumnya. Hasil pengujian menunjukkan bahwa perubahan

nilai K, jumlah data latih, dan komposisi data latih memiliki dampak signifikan pada akurasi metode KNN. Pengujian terhadap pengaruh nilai K menunjukkan rata-rata akurasi sebesar 88.55%. Variasi jumlah data latih memperlihatkan rata-rata akurasi sebesar 92.42%, sedangkan pengujian terhadap komposisi data latih menunjukkan rata-rata akurasi sebesar 87.89%. Selain itu, pengujian terhadap pengaruh komposisi data latih dan data uji terhadap akurasi menunjukkan nilai rata-rata akurasi sebesar 96.35%. Temuan ini memberikan indikasi bahwa metode KNN memiliki potensi untuk memberikan diagnosa sementara dengan akurasi yang tinggi, terutama dengan pertimbangan optimalisasi nilai K, jumlah data latih, dan komposisi data latih [4].

Penelitian [5] bertujuan untuk mengimplementasikan *Internet of Things* (IoT) dengan fokus pada keamanan. Sebagai langkah preventif, teknik machine learning diterapkan untuk mengklasifikasikan anomali *traffic* menggunakan algoritma KNN. Tujuan utama penerapan KNN adalah membedakan *data traffic* antara *benign* dan *malicious*, dengan memanfaatkan dataset aposemat IoT-23 yang mencakup 23 scenario, termasuk 20 dataset scenario malicious dan 3 dataset scenario *benign*. Hasil penelitian menunjukkan bahwa setelah proses *training* model, nilai akurasi yang diperoleh mencapai 94%. Model yang telah dilatih mampu memprediksi dengan tepat *data traffic* baru, dengan 20 dari 25 data baru diprediksi sesuai dan 5 data diprediksi tidak sesuai.

Dari penelitian [6] mengatakan bahwa metode KNN memberikan prediksi *malicious* website sebesar 2,42%. Penelitian ini bersifat proaktif dengan fokus pada prediksi *malicious* website berdasarkan karakteristik *application layer* dan *network*. Metode KNN digunakan untuk melakukan prediksi tersebut.

2.2 K-Nearest Neighbor

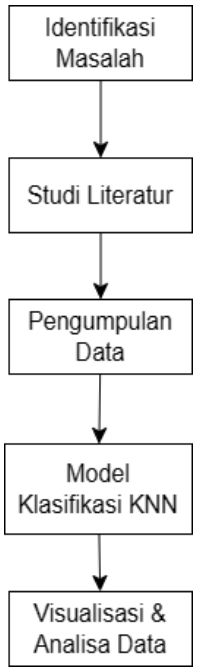
Algoritma *K-Nearest Neighbors* (KNN) adalah algoritma *supervised learning* yang digunakan untuk mengklasifikasikan objek baru berdasarkan kedekatannya dengan objek-objek yang telah diketahui kelasnya [7]. Algoritma ini bekerja dengan menghitung jarak

antara objek baru dengan objek-objek yang telah diketahui kelasnya. Jarak yang paling sering digunakan adalah jarak *Euclidean*, dan jarak lain yang juga dapat digunakan, yaitu jarak *Manhattan* atau jarak *Minkowski*. Pada dasarnya, algoritma KNN mengasumsikan bahwa objek-objek yang mirip cenderung memiliki kelas yang sama. Dengan kata lain, objek-objek yang jaraknya dekat cenderung memiliki kelas yang sama [8].

KNN merupakan teknik klasifikasi yang *fundamental* dan paling sederhana ketika sedikit atau tidak ada pengetahuan sebelumnya tentang distribusi data [9]. Aturan ini secara sederhana menyimpan seluruh set pelatihan selama pembelajaran dan menetapkan kepada setiap pertanyaan kelas yang direpresentasikan oleh label mayoritas dari k tetangga terdekatnya dalam set pelatihan. Aturan *Nearest Neighbor* (NN) merupakan bentuk paling sederhana dari KNN ketika K = 1 [10]. Jika nilai k adalah bilangan genap, maka kemungkinan jumlah poin di setiap kelas menjadi sama dan tidak dapat memprediksi titik kelas dengan tepat. Jika menetapkan nilai k = 1 maka data akan overfit dengan membuat batas keputusan non-linier. KNN membuat batas keputusan non-linier yang kompleks yang dapat mengklasifikasi data secara efisien meskipun data dipisahkan secara non linier.

3. Metode

Pembuatan model klasifikasi malware dengan KNN dilakukan dalam beberapa tahap yaitu seperti gambar dibawah ini.



Gambar 1. Tahapan Pengerjaan Model

3.1 Identifikasi Masalah

Pada tahap ini, peneliti perlu menentukan terlebih dahulu masalah apa yang akan diteliti. Identifikasi masalah yang dilakukan harus masalah yang memiliki solusi bermanfaat dan sedang terjadi saat ini.

3.2 Studi Literatur

Di tahap ini, peneliti perlu memahami serta mempelajari penelitian sebelumnya yang relevan dengan masalah yang telah dipilih untuk menjadi topik dari proyek ini. Studi literatur dilakukan untuk mendapatkan pemahaman yang lebih baik dan mendalam

tentang masalah yang diteliti, dan membantu dalam memahami metode-metode yang telah digunakan untuk menyelesaikan masalah yang serupa dengan masalah yang akan diteliti.

3.3 Pengumpulan Data

Data yang digunakan pada penelitian ini yaitu menggunakan data *Malware Android Traffic* berupa file csv yang telah diberikan sebagai data yang akan digunakan untuk model klasifikasi *malware*. Sampel ini mencakup berbagai jenis *malware* seperti *malicious*. Selain itu, juga disertakan sampel data *non-malware*, seperti *benign*, untuk melatih model KNN dalam mengenali pola yang berbeda antara *malware* dan *non-malware*.

3.4 Model Klasifikasi dengan KNN

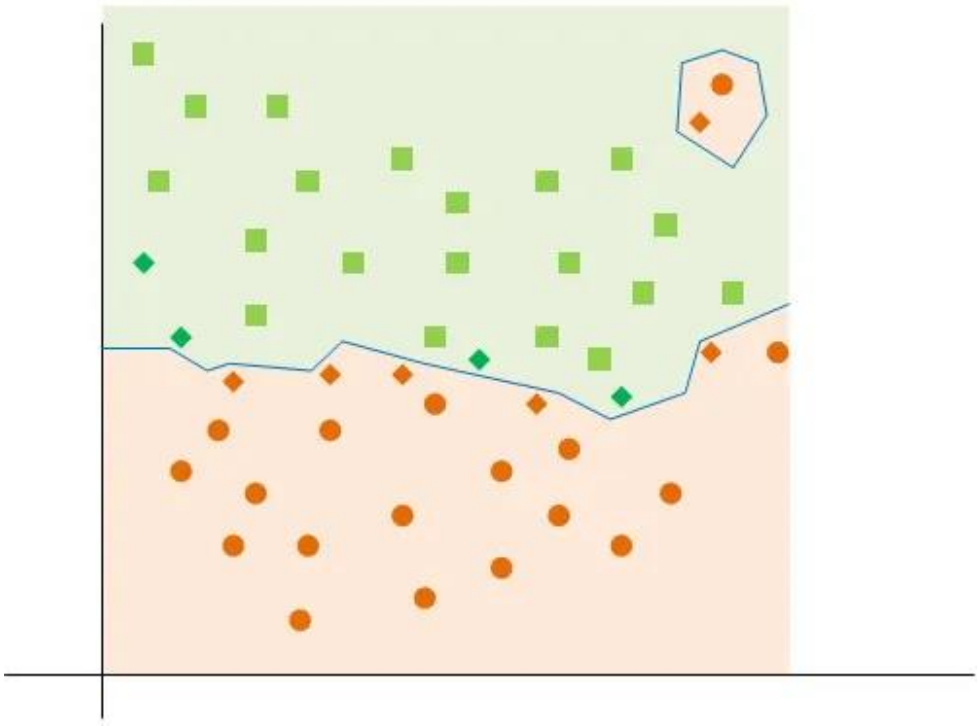
Peneliti menggunakan metode KNN untuk mengklasifikasi data yang dikumpulkan. Algoritma KNN digunakan dengan menetapkan nilai k, yaitu *hyperparameter*[11]. *Hyperparameter* adalah parameter yang ditentukan sebelum proses pelatihan model machine learning dimulai. Langkah-langkah yang dilakukan yaitu:

1. Hitung jarak antara titik pengujian, misalnya ‘t’, dan semua titik pelatihan lainnya, misalnya xi dimana i = 1 sampai n. Menghitung jarak ini dapat dilakukan dengan perhitungan jarak *Euclidean*:

$$D_{(t,x_j)} = \sqrt{\sum_{i=1}^{n}(t_i - x_{ji})^2} \qquad \ldots\ldots\ldots\ldots (1)$$

2. Ekstrak titik ‘k’ yang berdekatan dengan ‘t’. Jarak yang dihitung yaitu d(t, x1), d(t, x2),..., d(t, xn) diurutkan d(t, xpl) <= d(t, xp2)...<= d(t, xpm) dan k poin pertama xp1, xp2,..., xpk dipilih. Kemudian, untuk melakukan klasifikasinya, tentukan dulu nilai k dan mendapatkan kelas mayoritas.

3. Jika nilai k ditingkatkan, maka batas keputusan menjadi semakin sederhana dan kekuatan generalisasi meningkat. Jika k = 1, maka klasifikasi dilakukan ke semua titik ke kelas yang memiliki jumlah data terbanyak dalam kelas tersebut.



Gambar 2. Contoh klasifikasi data dengan k = 1.

3.5. Visualisasi dan Analisa Data

3.5.1 Visualisasi

Akan dilakukan visualisasi data terhadap model klasifikasi yang telah dikembangkan dengan algoritma KNN. Tujuan visualisasi untuk memudahkan pembaca dalam memahami algoritma yang dikembangkan dan hasil yang akan diperoleh dari algoritma ini.

3.5.2 Analisa Data

Setelah model klasifikasi dikembangkan dan dilatih, dilakukan pengujian terhadap berbagai jenis *malware*. Hasil pengujian dievaluasi secara mendalam, memperhatikan akurasi, recall, precision, dan f-1 score. Analisis hasil bertujuan untuk memperlihatkan apakah model klasifikasi dapat mengklasifikasi *malicious* dan *benign*. *Accuracy, recall,* dan *precision* dapat dilakukan dengan rumus-rumus berikut:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad \dots\dots\dots\dots (2)$$

$$Precision = \frac{TP}{TP + FP} \quad \dots\dots\dots\dots (3)$$

$$Recall = \frac{TP}{TP + FN} \quad \dots\dots\dots\dots (4)$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad \dots\dots\dots\dots (5)$$

Catatan:

TP = *True Positive,* FP = *False Positive,*

TN = *True Negative,* FN = *False Negative.*

Analisa data akan dilakukan dengan mengukur akurasi, recall, precision, dan F1-Score model klasifikasi.

- *Accuracy* adalah persentase sampel yang diklasifikasikan dengan benar oleh model klasifikasi.
- *Recall* adalah persentase sampel malware yang diklasifikasikan dengan benar sebagai malware oleh model klasifikasi.
- *Precision* adalah persentase sampel yang diklasifikasikan sebagai *malware* oleh model klasifikasi yang benar-benar *malware*.
- *F1- Score* adalah perbandingan antara *precision* dan *recall*, untuk memberi keseimbangan antara keduanya, dan berguna ketika keseimbangan antara *false positive* dan *false negative*.

4. Hasil Pengujian

Pemodelan klasifikasi dibuat dengan menggunakan Bahasa *python* di *Jupyter Notebook*. Berikut pengerjaan pemodelan dilakukan:

4.1. Import Library

```
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
```

Gambar 3. Code *import library*

Library yang di-*import* sesuai dengan kebutuhan untuk melakukan pemodelan klasifikasi dengan algoritma KNN, melakukan Analisa data, dan Visualisasi data.

4.2. Membaca Dataset

```
df = pd.read_csv('android_traffic.csv', sep = ';')
```

Gambar 4. Pembacaan dataset

Dataset yang dibaca adalah dataset yang sebelumnya sudah diperoleh dari Kaggle. Dengan menggunakan “sep = ‘;’” untuk dilakukan separasi dengan semicolon agar dataset dapat dibaca.

4.3. Mengecek Missing Value

Sebelum pemodelan dilakukan, dataset harus dicek apakah memiliki nilai NaN. Hal ini dilakukan karena model klasifikasi tidak dapat mengelola missing value secara langsung. Jika tetap digunakan, maka hasil yang diberikan tidak akurat. Nilai missing value harus ditangani agar pemodelan dapat menghasilkan nilai yang akurat.

```
In [3]: df.isnull().sum()

Out[3]: name                    0
        tcp_packets             0
        dist_port_tcp           0
        external_ips            0
        vulume_bytes            0
        udp_packets             0
        tcp_urg_packet          0
        source_app_packets      0
        remote_app_packets      0
        source_app_bytes        0
        remote_app_bytes        0
        duracion             7845
        avg_local_pkt_rate   7845
        avg_remote_pkt_rate  7845
        source_app_packets.1    0
        dns_query_times         0
        type                    0
        dtype: int64
```

Gambar 5. Menampilkan data yang NaN

4.4. Menghapus nilai Missing Value

```
In [4]: df_cleaned = df.drop(columns = ['duracion', 'avg_local_pkt_rate', 'avg_remote_pkt_rate'], axis = 1)

In [5]: df_cleaned.isnull().sum()

Out[5]: name                  0
        tcp_packets           0
        dist_port_tcp         0
        external_ips          0
        vulume_bytes          0
        udp_packets           0
        tcp_urg_packet        0
        source_app_packets    0
        remote_app_packets    0
        source_app_bytes      0
        remote_app_bytes      0
        source_app_packets.1  0
        dns_query_times       0
        type                  0
        dtype: int64
```

Gambar 6. Penghapusan nilai NaN

Missing value dapat ditangani dengan dilakukan penghapusan. Karena jumlah dataset yaitu 7845, dan ada 3 kolom yang memiliki nilai NaN yaitu kolom *'duracion'*, *'avg_local_pkt_rate'*, *'avg_remote_pkt_rate*', maka dilakukan penghapusan pada ketiga kolom tersebut. Dan dilakukan pemanggilan kembali pada kolom-kolom yang tidak memiliki nilai NaN, sehingga data inilah yang akan dilakukan pemodelan klasifikasi.

4.5. Mendownload dan mengecek kembali data yang sudah dibersihkan.

```
In [6]: df_cleaned.to_csv('android_traffic_cleaned.csv', index = False)

In [7]: df = pd.read_csv('android_traffic_cleaned.csv')

In [8]: df.head()
```

Out[8]:

	name	tcp_packets	dist_port_tcp	external_ips	vulume_bytes	udp_packets	tcp_urg_packet	source_app_packets	remote_app_packets	source_app_bytes
0	AntiVirus	36	6	3	3911	0	0	39	33	5100
1	AntiVirus	117	0	9	23514	0	0	128	107	26248
2	AntiVirus	196	0	6	24151	0	0	205	214	163887
3	AntiVirus	6	0	1	889	0	0	7	6	819
4	AntiVirus	6	0	1	882	0	0	7	6	819

Gambar 7. Pengecekan kembali nilai NaN

Setelah dilakukan penghapusan missing value, data yang terbaru didownload dan dicek kembali untuk memastikan bahwa tidak ada nilai NaN dalam tabel.

4.6. Split Data

```
In [9]: x = df.iloc[:, :-1].values
        y = df.iloc[:, 13].values

In [10]: print(x)
[['AntiVirus' 36 6 ... 4140 39 3]
 ['AntiVirus' 117 0 ... 24358 128 11]
 ['AntiVirus' 196 0 ... 24867 205 9]
 ...
 ['Zsone' 0 0 ... 143 2 2]
 ['Zsone' 0 0 ... 143 2 2]
 ['Zsone' 0 0 ... 143 2 2]]

In [9]: x = df.iloc[:, :-1].values
        y = df.iloc[:, 13].values

In [14]: print(y)
['benign' 'benign' 'benign' ... 'malicious' 'malicious' 'malicious']

In [17]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 50)
```

Gambar 8. Melakukan split data dengan x dan y

Split data dilakukan untuk membagi data ke *training set* dan *testing set*. Karena dalam x dan y terdapat variable kategorikal maka perlu dilakukan encoding data.

4.7. Encoding

Encoding dilakukan untuk membuat varibel kategorikal ke numeric. Hal ini membantu dalam melakukan pemodelan klasifikasi.

4.7.1 *Encoding* x

```
In [11]: encoder = OneHotEncoder(handle_unknown='ignore')

In [12]: x = encoder.fit_transform(x).toarray()

In [13]: x
Out[13]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])
```

Gambar 9. *Encoding* x

Untuk *encoding* x, digunakan OneHotEncoder yang melakukan encoding pada variable kategorikal. Variabal kategorikal diubah menjadi bentuk biner.

4.7.2 *Encoding* y

```
In [15]: label_mapping = {'benign': 0, 'malicious': 1}

In [16]: y_numeric = [label_mapping[label] for label in y]
```

Gambar 10. *Encoding* y

Enoding y, digunakan label karena hanya ada 2 vairabel kategorikal yaitu *benign* dan *malicious*. *Benign* diberi nilai biner 0 dan *malicious* diberi nilai

```
print(y_numeric)
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Gambar 11. Isi dari encoding y

4.8. Menghitung *n_neighbors*

```
In [4]: len(y_test)
Out[4]: 1569

In [5]: import math
        math.sqrt(len(y_test))
Out[5]: 39.61060464067672
```

Gambar 12. Perhitungan n_neighbors

N_neighbors dihitung untuk menentukan berapa banyak tetangga yang akan digunakan dalam melakukan klasifikasi karena akan mempengaruhi pemodelan klasifikasi. Nilai *n_neighbors* bernilai ganjil untuk menghindari kesulitan dalam menentukan kelas yang benar berdasarkan k tertentu dan dengan nilai ganjil dapat memiliki jumlah tetangga yang unggul dari satu kelas.

4.9. Classifier KNeighbors

```
In [23]: classifier = KNeighborsClassifier(n_neighbors = 39, metric = 'euclidean')
         classifier.fit(x_train, y_train)
Out[23]: KNeighborsClassifier(metric='euclidean', n_neighbors=39)

In [24]: y_pred = classifier.predict(x_test)

         y_pred
Out[24]: array([1, 1, 1, ..., 1, 1, 1])
```

Gambar 13. *Classifier Kneighbors*

Parameter *n_neighbors* = 39 artinya model akan menggunakan 39 tetangga terdekat saat melakukan prediksi dan menggunakan *metric Euclidean* untuk mengukur jarak antar titik data. Kemudian pemodelan akan melakukan prediksi pada x_text dan hasil dari x_test akan disimpan dalam y_pred.

4.10. Confusion Matrix

```
In [8]: cm = confusion_matrix(y_test, y_pred)

In [9]: print(cm)
        [[845  78]
         [ 71 575]]
```

Gambar 14. *Confusion matrix*

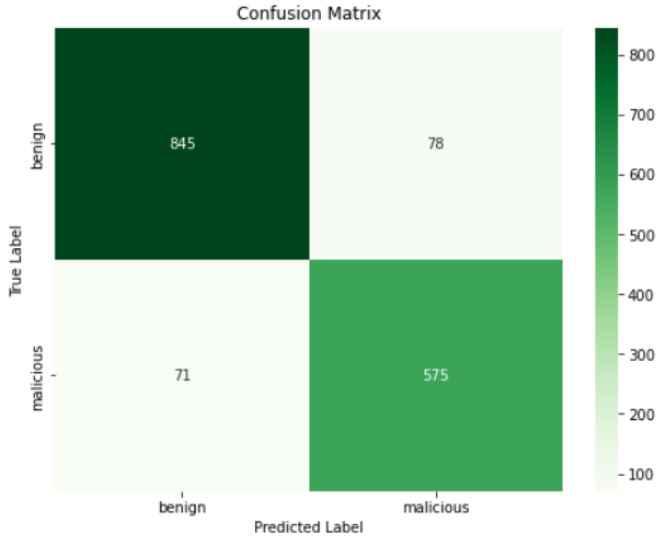
Confusion matrix membantu dalam mengevaluasi pemodelan klasifikasi yang sudah dibuat untuk menunjukkan seberapa baik model dapat mengklasifikasikan data ke dalam kelas yang benar. *Confusion matrix* memberikan informasi yang rinci untuk memahami apakah model dapat mengatasi kasus *positife* dan *negative* dengan benar.

4.11. Visualization

```
In [15]: def plot_confusion_matrix(conf_matrix, target_names, title='Confusion Matrix', cmap=None, normalize=False):
             if normalize:
                 conf_matrix = conf_matrix.astype('float') / conf_matrix.sum(axis=1)[:, np.newaxis]

             plt.figure(figsize=(8, 6))
             sns.heatmap(conf_matrix, annot=True, fmt=".2f" if normalize else "d", cmap=cmap, xticklabels=target_names, yticklabels=target
             plt.title(title)
             plt.xlabel('Predicted Label')
             plt.ylabel('True Label')
             plt.show()

         plot_confusion_matrix(cm, target_names=['benign', 'malicious'], title='Confusion Matrix', cmap='Greens', normalize=False)
```

Gambar 15. *Visualization*

Colormap digunakan untuk *heatmap*, *color* yang digunakan yaitu *Greens*. Sns.heatmap() membuat heatmap Menggunakan library *seaborn* dengan parameter ‘annot = True’ untuk menunjukkan bahwa nilai-nilai di sel akan ditampilkan. Plt.xlabel untuk menambahkan label pada sumbu x dan plt.ylabel y untuk menunjukkan label prediksi dan label sebenarnya.



Gambar 16. *Visualization confusion matrix*

Kemudian, dilakukan visualisasi untuk melakukan perbandingan performa algoritma KNN dengan berbagai nilai parameter k yang diberikan dari dataset.

```
In [13]: import matplotlib.pyplot as plt
         import pandas as pd
         import seaborn as sns
         from sklearn import neighbors
         from mlxtend.plotting import plot_decision_regions
         from sklearn.preprocessing import LabelEncoder
```

```
In [15]: from sklearn.decomposition import PCA
         import matplotlib.pyplot as plt

         pca = PCA(n_components=2)
         X_pca = pca.fit_transform(x_train)

         knn_model = KNeighborsClassifier(n_neighbors=39)
         knn_model.fit(x_train, y_train)

         y_pred = knn_model.predict(x_test)

         X_pca_test = pca.transform(x_test)

         X_pca_with_pred = np.column_stack((X_pca_test, y_pred))

         plt.figure(figsize=(10, 8))
         scatter = plt.scatter(X_pca_with_pred[:, 0], X_pca_with_pred[:, 1], c=X_pca_with_pred[:, 2], cmap='viridis',
                               edgecolors='k', label='Kelas')

         legend_labels = ['Kelas 0', 'Kelas 1']  # Ganti sesuai dengan kelas Anda
         plt.legend(handles=scatter.legend_elements()[0], labels=legend_labels)

         plt.title('Visualisasi Hasil Pengelompokan KNN dengan PCA (2D)')
         plt.xlabel('Principal Component 1')
         plt.ylabel('Principal Component 2')
         plt.show()
```

Gambar 17. Code Visualization

Dengan k yaitu parameter jumlah tetangga yang digunakan yaitu 39. Digunakan *plot_decision_regions* untuk memvisualisasikan batas keputusan dari model KNN yang telah dibuat. Perlu diketahui juga bahwa x adalah variable *independent* dan y adalah variable *dependent*. Dan dengan label persegi untuk jenis *benign* dan segitiga untuk *malicious*. Visualisasi dibuat dengan menggunakan teknik *Principal Component Analysis* (PCA). Tujuan penggunaan teknik ini adalah untuk mengidentifikasi pola atau struktur yang signifikan dalam data dan merepresentasikannya dengan sejumlah variable yang bersifat komponen utama.

Digunakan dua komponen utama dari dataset dan k = 39 dilatih menggunakan data x_train dan y_train. Hasil prediksi pada *training set* diproyeksikan ke ruang 2 dimensi dengan Menggunakan transformasi PCA.

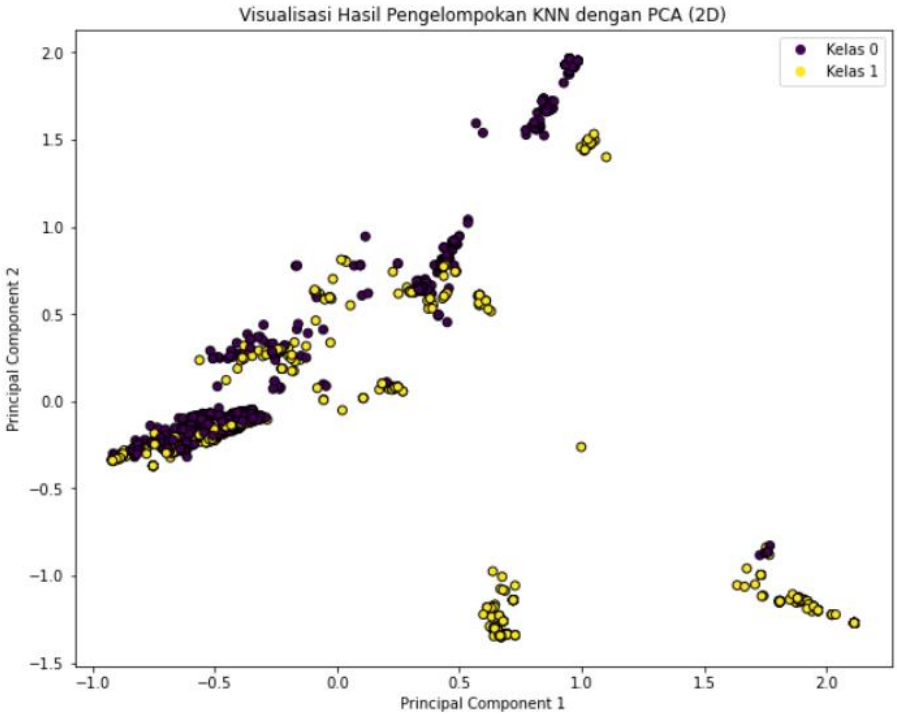
5. Analisis

Pemodelan klasifikasi dengan metode KNN dibuat dengan menggunakan scalling data. Namun, penggunaan scalling data ini bergantung pada sifat data apa yang digunakan. Karena sifat data android_traffic.csv memiliki skala yang tidak signifikan maka scalling data tidak perlu. Berdasarkan hasil uji yang dilakukan, diperoleh *accuracy* sebesar 90%. Selain itu, melalui hasil *recall* (2), *precision* (3), dan *f1-score* (4), diperoleh yaitu:

```
In [10]: report = classification_report(y_test, y_pred)

In [11]: print(report)
                precision    recall  f1-score   support

             0       0.92      0.92      0.92       923
             1       0.88      0.89      0.89       646

      accuracy                           0.91      1569
     macro avg       0.90      0.90      0.90      1569
  weighted avg       0.91      0.91      0.91      1569
```

Gambar 19. Hasil untuk *precision, recall,* dan *f1-score*

Precision untuk kelas 0 adalah 0.92 yang berarti 92%, dan untuk kelas 1 adalah 0.88 yang berarti 88%. *Recall* untuk kelas 0 adalah 0.92 yang berarti 92% dan untuk kelas 1 adalah 0.89 yang berarti bahwa 89% untuk data 1. *F1-Score* merupakan rata-rata harmonic dari *precision* dan *recall*. *F1-score* untuk kelas 0 adalah 0.92 dan untuk kelas 1 adalah 0.89. *macro avg* adalah rata-rata dari *precision*, *recall*, dan *f1-score*, dan nilai rata-ratanya yaitu 0.90. Nilai rata-rata ini sama untuk kelas 0 dan kelas 1 di *precision*, *recall*, dan *f1-score*. Dapat dinilai bahwa model dapat bekerja dengan baik untuk melakukan klasifikasi pada data *benign* dan *malicious*. Hal ini ditunjukkan dari *hasil presicion, recall, f1-score* yang tinggi untuk kelas 0 dan kelas 1, serta nilai *accuracy* yang tinggi.



Gambar 18. *Visualization plot KNN*

Dari visualisasi model klasifikasi dengan menggunakan Teknik PCA. Diperoleh pada grafik menunjukkan tcp_packets yang diamati dari nilai disct_port_tcp yang berbeda. Hasil menunjukkan semakin tinggi nilai disct_port_tcp dan tcp_packets maka *software* bukan *malicious*. Kelas 0 adalah *beging* dan kelas 1 adalah *malicious.*

6. Kesimpulan

Hasil penilitian yang diperoleh menunjukkan bahwa pemodelan klasifikasi *software* pada data android_traffic.csv dengan algoritma KNN dapat dilakukan dengan baik. Pemodelan klasifikasi dengan algoritma ini memperoleh *accuracy* sampai sebesar 90% untuk melakukan klasifikasi data *malware* dan *non-malware* pada *software*. Hal ini ditunjukkan dari hasil *precision, recall, f1-score* yang tinggi untuk kelas 0 dan kelas 1, dan memiliki *accuracy* yang tinggi. Metode KNN digunakan dengan melakukan normalisasi, namun normalisasi dilakukan tergantung pada sifat dari data yang digunakan untuk melakukan pemodelan klasifikasi. Karena sifat data ini memiliki sifat ada yang tidak beragam, maka tidak perlu dilakukan normalisasi.

7. Pembagian Pekerjaan

Nama Anggota Kelompok	Pembagian Pekerjaan
Lamria Tampubolon	Bab 1. Pendahuluan, code labeling x dan y, dan code Penentuan fitur Independent dan Dependent
Lasni Simanjuntak	Bab 2. Studi Literatur dan code classifier
Astri Siahaan	Bab 3. Metode, code training set dan testing set, dan code n_neighbors
Naomi Lumbanraja	Bab 3. Metode, Bab 4. Hasil dan Pengujian dan code data cleaned, dan code confusion matrix dan accuracy, dan code data visualisasi confusion matrix
Daniel Girsang	Bab 5 Analisis, data visualisasi model KNN
Semua anggota kelompok	Kesimpulan

REFERENSI

[1] A. K. Neighbors, “Analisis Klasifikasi Keamanan dalam Shorting Malware Android dengan,” vol. 5, pp. 321–329, 2023.

[2] R. B. Hadiprakoso and et all Aditya, “Analisis Statis Deteksi Malware Android Menggunakan Algoritma Supervised Machine Learning,” *Cyber Secur. dan Forensik Digit.*, vol. 5, no. 1, pp. 1–5, 2022, doi: 10.14421/csecurity.2022.5.1.3116.

[3] N. Chitayae, A. H. Muhammad, T. Komputer, I. Teknologi, and N. Ulama, “Identifikasi Malware pada Android menggunakan Algoritma K- Nearest Neighbor,” vol. 3, no. 2, 2023.

[4] F. Wafiyah, N. Hidayat, and R. S. Perdana, “Implementasi Algoritma Modified K-Nearest Neighbor (MKNN) untuk Klasifikasi Penyakit Demam,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 1, no. 10, pp. 1210–1219, 2017.

[5] Ari Sandriana, Rianto, and Firmansyah Maulana, “Klasifikasi serangan Malware terhadap Lalu Lintas Jaringan Internet of Things menggunakan Algoritma K-Nearest Neighbour (K-NN),” *E-JOINT (Electronica Electr. J. Innov. Technol.*, vol. 3, no. 1, pp. 12–22, 2022, doi: 10.35970/e-joint.v3i1.1559.

[6] G. A. Sandag, J. Leopold, and V. F. Ong, “Klasifikasi Malicious Websites Menggunakan Algoritma K-NN Berdasarkan Application Layers dan Network Characteristics,” *CogITo Smart J.*, vol. 4, no. 1, pp. 37–45, 2018, doi: 10.31154/cogito.v4i1.100.37-45.

[7] S. K. P. Loka and A. Marsal, “Perbandingan Algoritma K-Nearest Neighbor dan Naïve Bayes Classifier untuk Klasifikasi Status Gizi Pada Balita,” *MALCOM Indones. J. Mach. Learn. Comput. Sci.*, vol. 3, no. 1, pp. 8–14, 2023, doi: 10.57152/malcom.v3i1.474.

[8] P. Cunningham and S. J. Delany, “K-Nearest Neighbour Classifiers-A Tutorial,” *ACM Comput. Surv.*, vol. 54, no. 6, 2021, doi: 10.1145/3459665.

[9] S. B. Mohammed, A. Khalid, and S. F. Osman, “A Survey of Classification

Methods," *Int. J. Adv. Eng. Res. Sci.*, vol. 3, no. 10, pp. 148–152, 2016, doi: 10.22161/ijaers/310.24.

[10] S. B. Imandoust and M. Bolandraftar, "Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events : Theoretical Background," *Int. J. Eng. Res. Appl.*, vol. 3, no. 5, pp. 605–610, 2013.

[11] S. Karmakar, "K-Nearest Neighbor," *17 Juli*, 2020. https://medium.com/@sourinkarmakar/k-nearest-neighbor-1715144f2a9c.