

## **LAPORAN TUGAS BESAR 2**

### **IF3170 - Inteligensi Artifisial**

Implementasi Algoritma Pembelajaran Mesin pada Dataset UNSW-NB15



Disusun oleh :

Thea Josephine Halim	13522012
Debrina Veisha Rashika W	13522025
Melati Anggraini	13522035
Raffael Boymian Siahaan	13522046

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG**

**2024**

## DAFTAR ISI

<b>DESKRIPSI MASALAH.....</b>	<b>3</b>
<b>DASAR TEORI.....</b>	<b>4</b>
2.1 Algoritma KNN.....	4
2.2 Algoritma Naive Bayes.....	4
<b>IMPLEMENTASI.....</b>	<b>8</b>
3.1 Implementasi Algoritma KNN.....	8
3.2 Implementasi Algoritma Gaussian Naive-Bayes.....	9
3.3 Implementasi Algoritma ID3.....	11
3.4 Tahap Cleaning dan Preprocessing.....	13
3.5 Tahap Modeling dan Validasi.....	17
<b>PERBANDINGAN HASIL PREDIKSI.....</b>	<b>19</b>
4.1 Perbandingan Implementasi KNN dengan library.....	19
4.2 Perbandingan Implementasi Naive Bayes dengan library.....	22
4.3 Perbandingan Implementasi ID3 dengan library.....	24
<b>KESIMPULAN.....</b>	<b>26</b>
5.1 Kesimpulan.....	26
5.2 Saran.....	26
<b>KONTRIBUSI.....</b>	<b>27</b>
<b>LAMPIRAN.....</b>	<b>27</b>
<b>DAFTAR PUSTAKA.....</b>	<b>27</b>

## **BAB I**

### **DESKRIPSI MASALAH**

[Dataset UNSW-NB15](#) merupakan dataset berisi raw network packets yang dibuat menggunakan IXIA PerfectStorm oleh Cyber Range Lab UNSW Canberra. Dataset ini terdiri dari 10 jenis aktivitas (9 jenis attack dan 1 aktivitas normal). Sembilan jenis attack yang termasuk ke dalam dataset ini adalah Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode dan Worms.

Pada tugas ini akan dilakukan analisis data dan prediksi target label *attack\_cat*. Untuk membuat prediksi ini data akan melalui beberapa tahap:

1. Data Cleaning

Tahap ini bertujuan untuk membersihkan dataset dari nilai yang hilang (missing values), data duplikat, atau data yang tidak valid sehingga dataset siap digunakan untuk analisis.

2. Data Transformation

Transformasi data melibatkan langkah-langkah seperti encoding variabel kategori, normalisasi atau standarisasi fitur numerik, serta penanganan ketidakseimbangan data (imbalanced data) untuk memastikan data berada dalam format yang sesuai dengan algoritma pembelajaran mesin.

3. Feature Selection

Pemilihan fitur yang relevan bertujuan untuk mengurangi kompleksitas model, menghindari overfitting, serta meningkatkan kinerja model. Langkah ini melibatkan identifikasi fitur yang memiliki pengaruh signifikan terhadap variabel target.

4. Dimensionality Reduction

Jika dataset memiliki jumlah fitur yang besar, reduksi dimensi dapat digunakan untuk mengurangi dimensi tanpa kehilangan informasi penting. Teknik seperti Principal Component Analysis (PCA) sering digunakan pada tahap ini.

5. Modeling dan Validation

Pada tahap ini, algoritma pembelajaran mesin seperti K-Nearest Neighbors (KNN), Naive Bayes, dan ID3 (Iterative Dichotomiser 3) diterapkan pada dataset. Model yang telah dibuat divalidasi menggunakan metode seperti train-test split atau k-fold cross-validation untuk memastikan kinerja yang optimal.

## BAB II

### DASAR TEORI

*Supervised Learning* adalah salah satu jenis pembelajaran mesin (*machine learning*) di mana model dilatih menggunakan dataset yang sudah memiliki label atau output yang diketahui. Model *supervised learning* belajar dari data yang berpasangan antara input (fitur) dan output (target) untuk membuat prediksi pada data baru. Data input merupakan kumpulan variabel yang digunakan untuk memprediksi output. Sedangkan data output merupakan nilai target yang ingin diprediksi.

#### 2.1 Algoritma KNN

KNN (*K-Nearest Neighbor*) adalah salah algoritma *supervised learning* yang hanya menyimpan seluruh data training dan tidak menghasilkan hipotesis. Algoritma KNN merupakan *lazy learner* karena sifatnya yang tidak melakukan proses *learning* atau pembuatan model eksplisit selama tahap pelatihan. Caranya memprediksi data adalah dengan mengkategorikan kelas berdasarkan kemiripannya dalam data yang disimpan.

Cara kerja KNN adalah menghitung jarak *unseen data (query)* ke data yang ada di dalam dataset menggunakan metrik jarak. Tahap selanjutnya adalah mencari *k nearest neighbor* data terdekat (data dengan jarak terkecil dari *query*). Berdasarkan hasil *k nearest neighbor* ini akan dicari kelas yang menjadi mayoritas untuk menjadi kelas bagi *unseen data* tersebut.

#### 2.2 Algoritma Naive Bayes

Algoritma Naive Bayes adalah salah satu algoritma *supervised learning* yang berbasis pada Teorema Bayes dengan asumsi "naïve" bahwa semua fitur dalam dataset saling independen, setiap kelas tidak saling mempengaruhi satu sama lain. Pada algoritma Naive Bayes akan dilakukan perhitungan probabilitas suatu data X masuk dalam kelas y.

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)}$$

Keterangan:

- $P(y|X)$ : Probabilitas data X termasuk dalam kelas y (posterior probability)

- $P(X|y)$ : Probabilitas data X terjadi jika kelasnya adalah y
- $P(y)$ : Probabilitas awal kelas y (prior probability of y)
- $P(X)$ : Probabilitas total data X (prior probability of X)

$$X = (x_1, x_2, x_3, \dots, x_n)$$

Nilai X ini dapat dituliskan sebagai atribut, dan dengan substitusi nilai X pada fungsi perhitungan probabilitas Naive Bayes sebelumnya kita mendapatkan:

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(x_1|y) \cdot P(x_2|y) \dots P(x_n|y) \cdot P(y)}{P(x_1) \cdot P(x_2) \dots P(x_n)}$$

Untuk setiap value pada data, nilai pembagi  $P(X)$  tidak akan berubah, sehingga dapat kita hilangkan.

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y)$$

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

Pada materi yang telah dipelajari sebelumnya kita menggunakan Multinomial Gaussian Bayes. Multinomial Gaussian Bayes menggunakan fungsi prediksi kelas di atas akan dilakukan penghitungan frekuensi  $P(y|X)$  untuk setiap atribut, membentuk sebuah *frequency table*, dan dari tabel frekuensi ini kita akan menghitung *likelihood table*, tabel probabilitas.

$P(a_i v_j)$								$P(v_j)$	
outlook		temperature		humidity		windy		play	
	yes no		yes no		yes no		yes no		
sunny	2/9 3/5	hot	2/9 2/5	high	3/9 4/5	false	6/9 2/5	9/14	5/14
overcast	4/9 0/5	mild	4/9 2/5	normal	6/9 1/5	true	3/9 3/5		
rainy	3/9 2/5	cool	3/9 1/5						

Gambar 2.2.1 Contoh Tabel Probabilitas

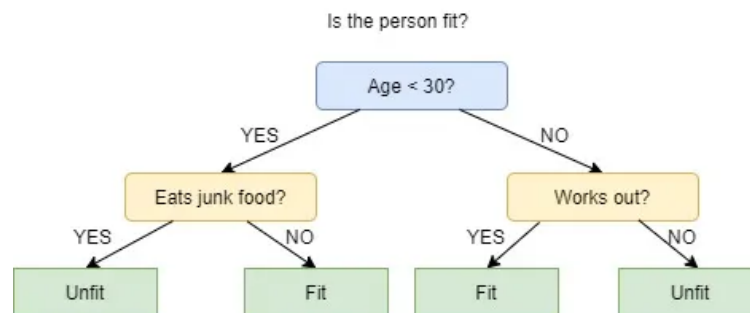
Setelah itu akan dilakukan perhitungan untuk  $P(y|X)$  dan pilih kelas yang memiliki nilai probabilitas tertinggi.

Akan tetapi, pada dataset kali ini terdapat banyak data kontinu sehingga akan digunakan Gaussian Naive Bayes yang lebih cocok digunakan untuk data bukan diskrit. Berbeda dengan Multinomial yang menggunakan tabel frekuensi, dengan  $\mu$  adalah rata-rata dan  $\sigma$  adalah standar deviasi, Gaussian akan menggunakan rumus:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

Gambar 2.2.2 Probabilitas  $P(X|y)$ 

## 2.3 ID3/ Decision Tree



Gambar 2.3.1 Contoh ID3

ID3 menggunakan algoritma top-greedy untuk ID3 (Iterative Dichotomiser 3) membangun pohon keputusan berdasarkan entropi dan information gain untuk memisahkan data. Decision tree ini terdiri dari beberapa node berupa persegi panjang, panah (edges) mewakili value yang mungkin dari suatu atribut, dan daun (node terakhir tanpa cabang lagi). Node berwarna biru dinamakan *root node*, node berwarna kuning adalah *intermediate* atau *internal node*, sedangkan node berwarna hijau adalah *leaf node*. *Root node* dan *intermediate node* adalah input, sedangkan *leaf node* adalah hasil keluaran.

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

$$\begin{aligned}
 \text{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\
 &= \text{Entropy}(0.36, 0.64) \\
 &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\
 &= 0.94
 \end{aligned}$$

Gambar 2.3.2 Perhitungan Entropi

Algoritma ID3 dimulai dari menghitung entropi dataset dengan  $P_i$  adalah probabilitas dari kelas  $i$ . Nilai entropi akan 0 apabila semua nilai dalam target kolom kita (*attack\_cat*) homogen, sedangkan akan bernilai 1 apabila jumlah nilai pada kedua kelas adalah sama. Entropi yang bernilai 0 ini akan membentuk *leaf node*. Langkah selanjutnya adalah menghitung *information gain* untuk setiap atribut yang ada. Atribut dengan *information gain* tertinggi akan digunakan sebagai node berikutnya. Proses yang sama akan berlangsung hingga dihasilkan hingga semua data dalam subset memiliki kelas yang sama, atau tidak ada atribut yang tersisa.

$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

$$\begin{aligned}
 E(\text{PlayGolf, Outlook}) &= P(\text{Sunny}) * E(3,2) + P(\text{Overcast}) * E(4,0) + P(\text{Rainy}) * E(2,3) \\
 &= (5/14) * 0.971 + (4/14) * 0.0 + (5/14) * 0.971 \\
 &= 0.693
 \end{aligned}$$

Gambar 2.3.3 Perhitungan Information Gain

## BAB III

### IMPLEMENTASI

#### 3.1 Implementasi Algoritma KNN

Pada fitting model KNN, data latih `X_train` dan label `y_train` akan disimpan tanpa proses perhitungan karena KNN termasuk algoritma lazy learning. Saat prediksi, untuk setiap data uji akan dihitung jarak terhadap semua data latih menggunakan metrik yang telah ditentukan, seperti euclidean, manhattan, atau custom. Setelah menghitung jarak, data latih akan diurutkan berdasarkan jarak terkecil, kemudian `k` tetangga terdekat diambil. Selanjutnya, kelas mayoritas dari tetangga tersebut ditentukan menggunakan fungsi `np.unique` untuk menghitung jumlah kemunculan setiap kelas. Kelas dengan frekuensi tertinggi akan dipilih sebagai prediksi akhir untuk data uji. Proses ini diulang untuk setiap data uji hingga semua prediksi diperoleh. Akurasi model dapat dihitung dengan membandingkan prediksi dan label sebenarnya melalui fungsi `score`.

<b>Deskripsi</b>	Fungsi untuk menginisialisasi parameter <code>k</code> (jumlah tetangga), metrik, dan <code>p</code> (untuk Minkowski)
<pre>def __init__(self, k=3, metric='euclidean', p=2):     if k &lt;= 0:         raise ValueError("Number of neighbors, k, must be greater than 0.")     if metric not in ['euclidean', 'manhattan', 'custom']:         raise ValueError("Supported metrics are 'euclidean', 'manhattan', and 'custom'.")     self.k = k     self.metric = metric     self.p = p</pre>	

<b>Deskripsi</b>	Fungsi untuk menyimpan data latih ( <code>X_train</code> ) dan labelnya ( <code>y_train</code> ) untuk digunakan dalam perhitungan jarak selama prediksi
<pre>def fit(self, X_train, y_train):     if len(X_train) != len(y_train):         raise ValueError("Training data and labels must have the same length.")     self.X_train = np.array(X_train)     self.y_train = np.array(y_train)     return self</pre>	



<b>Deskripsi</b>	Fungsi untuk memprediksi kelas untuk setiap data uji dengan mencari k tetangga terdekat, lalu memilih kelas mayoritas sesuai dengan teori KNN
<pre>def predict(self, X_test):     preds = []     for test_row in X_test:         nearest_neighbours = self.get_neighbours(test_row)         values, counts = np.unique(nearest_neighbours, return_counts=True)         majority = values[np.argmax(counts)]         preds.append(majority)     return np.array(preds)</pre>	

<b>Deskripsi</b>	Fungsi untuk menghitung jarak antara satu titik uji dengan semua titik latih berdasarkan metrik yang dipilih, lalu mengembalikan kelas k tetangga terdekat
<pre>def get_neighbours(self, test_row):     if self.metric == 'euclidean':         distances = np.sqrt(np.sum((self.X_train - test_row)**2, axis=1))     elif self.metric == 'manhattan':         distances = np.sum(np.abs(self.X_train - test_row), axis=1)     elif self.metric == 'custom':         distances = np.sum(np.abs(self.X_train - test_row), axis=1)     distances = list(zip(distances, self.y_train))     distances.sort(key=lambda x: x[0])     neighbours = [distances[i][1] for i in range(self.k)]     return neighbours</pre>	

### 3.2 Implementasi Algoritma Gaussian Naive-Bayes

Pada fitting model akan dibuat 3 buah list untuk rata-rata, variansi, dan probabilitas prior dengan key berupa nama label. Pada iterasi pertama sebanyak jumlah label akan dilakukan penghitungan untuk probabilitas prior, rata-rata, dan nilai variansi untuk setiap label/kelas. Setelah memiliki data probabilitas prior, rata-rata, dan variansi, kita akan melakukan perhitungan likelihood  $P(X|y)$  dengan fungsi `calculate_likelihood` dan menjumlahkannya dengan probabilitas prior yang sudah kita simpan pada list sebelumnya. Dari setiap probabilitas ini akan kita pilih kelas dengan nilai probabilitas yang tertinggi.

$$\begin{aligned}
 p(y_k | x) &\propto p(y_k) \prod p(x_i | y_k) \\
 \text{Applying log on both sides} \\
 \log(p(y_k | x)) &\propto \log(p(y_k)) + \sum \log(p(x_i | y_k)) \\
 &\stackrel{\text{GNB}}{\leq} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} * \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)\right) \\
 &\leq -0.5 * \log(2\pi\sigma^2) - \frac{(x_i - \mu)^2}{2\sigma^2}
 \end{aligned}$$

Gambar 3.2.1

Perlu ditambahkan bahwa dalam proses prediksi, untuk setiap data poin, model menghitung logaritma dari probabilitas posterior untuk semua kelas. Ini dilakukan dengan menjumlahkan log-likelihood dari fitur-fitur dan log dari prior probability. Logaritma digunakan untuk mencegah underflow akibat perkalian nilai probabilitas yang kecil yang bisa menyebabkan hasil perkalian posterior probability menjadi 0.

<b>Deskripsi</b>	Fungsi untuk melatih model Naive Bayes dengan menghitung frekuensi setiap fitur untuk setiap kelas pada train data X dan target y
<pre> def fit(self, X, y):     self.n_features = X.shape[1]     self.classes = np.unique(y)     self.n_classes = len(self.classes)      self.class_means = []     self.class_variances = []     self.class_counts = []     self.class_priors = []      for label in self.classes:         mask = (y == label)         X_class = X[mask]         class_count = len(X_class)         self.class_counts.append(class_count)         self.class_priors.append(class_count / len(X))         self.class_means.append(np.mean(X_class, axis=0))         self.class_variances.append(np.var(X_class, axis=0) + 1e-7) </pre>	

<b>Deskripsi</b>	Fungsi untuk melakukan perhitungan likelihood ( $P(X y)$ )
<pre>def calculate_likelihood(self, x, mean, var):     eps = 1e-6     log_likelihood = -0.5 * (np.log(2 * np.pi * (var + eps))) - ((x - mean) ** 2) / (2 * (var + eps))     return log_likelihood</pre>	

<b>Deskripsi</b>	Fungsi untuk memprediksi kelas untuk setiap sampel dalam suatu dataset data
<pre>def predict(self, X):     return np.array([self.predict_single(x) for x in X])</pre>	

<b>Deskripsi</b>	Fungsi untuk memprediksi kelas untuk satu sampel data menggunakan algoritma Naive Bayes
<pre>def predict_single(self, x):     posteriors = []      for i, _ in enumerate(self.classes):         prior = np.log(self.class_priors[i])         likelihood = np.sum(self.calculate_likelihood(x, self.class_means[i], self.class_variances[i]))         posterior = prior + likelihood         posteriors.append(posterior)     return self.classes[np.argmax(posteriors)]</pre>	

### 3.3 Implementasi Algoritma ID3

Pada algoritma ID3, proses pembangunan model dilakukan dengan membangun pohon keputusan berdasarkan perhitungan Information Gain dari setiap fitur yang tersedia. Semakin tinggi nilai Information Gain suatu fitur, semakin mudah untuk memisahkan data untuk setiap kelas yang ada.

Pertama dalam membuat pohon keputusan adalah menghitung Information Gain untuk setiap fitur yang tersedia. Fitur dengan nilai Information Gain tertinggi akan dipilih sebagai akar pohon keputusan. Setelah akar pohon dipilih, data akan dibagi menjadi subset-subset berdasarkan nilai-nilai fitur tersebut. Proses ini dilakukan secara rekursif untuk setiap subset,

dengan menghitung kembali Information Gain pada fitur lainnya, hingga mencapai kelas atau tidak ada fitur yang tersisa untuk dibagi.

Pohon keputusan yang dihasilkan dapat digunakan untuk memprediksi kelas dari data baru. Untuk melakukan prediksi, ditelusuri cabang-cabang pohon berdasarkan nilai atribut input yang dimiliki oleh data tersebut. Setiap cabang pohon merepresentasikan pengkategorian berdasarkan nilai atribut tertentu dan daun pohon menunjukkan prediksi kelas.

Deskripsi	Fungsi untuk menghitung Information Gain dan entropy dari setiap fitur.
<pre>def entropy(self,y):     class_counts = np.bincount(y)     probabilities = class_counts / len(y)     return -np.sum(probabilities * np.log2(probabilities + 1e-9))  def informationGain(self,X, y, index):     allEntropy = self.entropy(y)     featureVal = X[:, index]     uniqueVal = np.unique(featureVal)      featureEntropy = 0     for val in uniqueVal:         subset_y = y[featureVal == val]         featureEntropy += (len(subset_y) / len(y)) * self.entropy(subset_y)      return allEntropy - featureEntropy</pre>	

Deskripsi	Fungsi untuk membangun tree berdasarkan perhitungan dari nilai Information Gain masing-masing fitur.
<pre>def treeModel(self, X, y, depth=0):     if len(np.unique(y)) == 1:         return {'label': y[0]}      if self.depth is not None and depth &gt;= self.depth:         return {'label': np.bincount(y).argmax()}      if X.shape[1] == 0:         return {'label': np.bincount(y).argmax()}      bestGain = -1     bestFeature = None     for idx in range(X.shape[1]):</pre>	

```

gain = self.informationGain(X, y, idx)
if gain > bestGain:
    bestGain = gain
    bestFeature = idx

if bestGain == 0:
    return {'label': np.bincount(y).argmax()}

tree = {'feature': bestFeature, 'branch': {}}
best_feature_values = np.unique(X[:, bestFeature])
for value in best_feature_values:
    X_subset, y_subset = self.splitData(X, y, bestFeature, value)
    tree['branch'][value] = self.treeModel(X_subset, y_subset, depth + 1)

return tree

```

### Deskripsi

Fungsi untuk memprediksi kelas dari data baru berdasarkan tree yang telah dibuat sebelumnya.

```

def predict(self, X):
    return np.array([self.predictChild(x) for x in X])

def predictChild(self, x):
    node = self.tree
    while 'label' not in node:
        feature_value = x[node['feature']]
        if feature_value in node['branch']:
            node = node['branch'][feature_value]
        else:
            return 6
    return node['label']

```

## 3.4 Tahap *Cleaning* dan *Preprocessing*

### 3.4.1 Feature Scaling

Feature scaling digunakan untuk memastikan seluruh atribut pada data berkontribusi dengan adil dan mencegah adanya dominasi dari atribut tertentu. Feature scaling ini penting terutama untuk algoritma yang menggunakan jarak (*distance*) sebagai penentu prediksi. Feature scaling bisa dilakukan dengan beberapa cara, diantaranya adalah Min-Max Scaling, Standardization (Z-Score), robust scaling, dan transformasi log. Kami mengimplementasikan algoritma scaler dengan menggunakan Min-Max dan Z-Score. Min-Max menskalakan data antara rentang yang ditentukan (biasanya 0 dan 1), sedangkan Z-Score menskalakan data sehingga memiliki rata-rata 0

dan deviasi standar 1. Selama uji coba, kami memutuskan untuk menggunakan Z-Score karena sifatnya yang lebih robust terhadap outlier dibandingkan MinMax karena tidak memaksa data ke rentang tertentu.

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import MinMaxScaler, StandardScaler

class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self, scaling_method='z-score'):
        self.scaling_method = scaling_method
        self.scaler = None

    def fit(self, X, y=None):
        # Identify numerical columns
        self.numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns

        if self.scaling_method == 'min-max':
            self.scaler = MinMaxScaler()
        elif self.scaling_method == 'z-score':
            self.scaler = StandardScaler()
        else:
            raise ValueError("Unsupported scaling method. Use 'min-max' or 'z-score'.")

        self.scaler.fit(X[self.numerical_cols])
        return self

    def transform(self, X):
        X = X.copy()

        if self.scaler:
            X[self.numerical_cols] = self.scaler.transform(X[self.numerical_cols])

        return X
```

Feature scaling dengan Z-Score menggunakan rumus:

$$Z = \frac{(x - \mu)}{\sigma}$$

Dengan nilai:

- Z adalah nilai Z-Score
- x nilai yang ingin kita standarisasi
- $\mu$  rata-rata dari dataset
- $\sigma$  standar deviasi dari dataset

### 3.4.2 Feature Encoding

Feature encoding (categorical encoding) adalah proses mengubah data numerik menjadi kategorikal. Feature encoding bisa dilakukan dalam beberapa cara, seperti label encoding, one hot encoding, dan target encoding. Label encoding cocok untuk data yang memiliki urutan yang signifikan. Target encoding mengganti nilai kategoris dengan angka yang berasal dari variabel target. Pada analisis kali ini kami menggunakan one hot encoding karena di antara value data tidak ada urutan yang signifikan.

```

import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin

class FeatureEncoder(BaseEstimator, TransformerMixin):

    def fit(self, X, y=None):
        self.cat_cols = X.select_dtypes(include=['object']).columns.difference(['attack_cat'])
        return self

    def transform(self, X):
        X_encoded = X.copy()

        for col in self.cat_cols:
            one_hot = pd.get_dummies(X_encoded[col], prefix=col)
            X_encoded = pd.concat([X_encoded, one_hot], axis=1)

            X_encoded.drop(col, axis=1, inplace=True)

        return X_encoded

```

### 3.4.3 Feature Imputer

Feature imputer adalah proses untuk mengatasi nilai yang kosong pada dataset. Kami menggunakan library SimpleImputer dari Sklearn untuk mengisi nilai yang kosong dengan rata-rata (mean) untuk data numerik. Untuk atribut state, proto, dan service kami mengisi dengan nilai mayoritas pada masing-masing atribut.

```

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer

class FeatureImputer(BaseEstimator, TransformerMixin):

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        for col in ['sjit', 'djit', 'sinpkt', 'dinpkt', 'tcprtt', 'synack', 'ackdat', 'dur', 'sbytes', 'dbytes',
                    'sttl', 'dttl', 'sloss', 'dloss', 'sload', 'dload', 'spkts', 'dpkts',
                    'swin', 'dwin', 'stcpb', 'dtcpb', 'smean', 'dmean', 'trans_depth', 'response_body_len',
                    'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd', 'is_ftp_login', 'ct_ftp_cmd',
                    'ct_srv_src', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm',
                    'ct_dst_src_ltm']:
            imputer = SimpleImputer(strategy='mean')
            X[col] = imputer.fit_transform(X[[col]])

        X['state'] = X['state'].fillna('INT')
        X['proto'] = X['proto'].fillna('-')
        X['service'] = X['service'].fillna('tcp')

        null_columns = X.columns[X.isnull().any()]
        print(X[null_columns].isnull().sum())

        return X

```

### 3.4.4 Feature Engineering

Feature engineering adalah proses memodifikasi data yang sudah ada menjadi informasi yang lebih relevan untuk digunakan. Kami melakukan binning/diskritisasi pada data numerik dalam skala 10 agar menyederhanakan data, sebab data kontinu yang ada pada dataset sangatlah luas dan akan lebih mudah untuk diproses apabila dilakukan

binning. Selain itu kami juga membuat feature Interaction yang merepresentasikan hubungan-hubungan antara fitur dengan membuat feature baru seperti 'sload\_dload\_ratio', 'sbytes\_dbytes\_ratio', 'smean\_dmean\_diff', dan 'tcprrt\_synack\_diff'.

```
import pandas as pd
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

class FeatureCreator(BaseEstimator, TransformerMixin):
    def __init__(self, bin_features=None, n_bins=10):
        self.bin_features = bin_features
        self.n_bins = n_bins

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        if self.bin_features is None:
            self.bin_features = X.select_dtypes(include=[np.float64]).columns.tolist()

        for feature in self.bin_features:
            X[feature] = pd.cut(X[feature], bins=self.n_bins, labels=False, include_lowest=True)

        return X
```

```
# Feature Interaction
class FeatureInteraction(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()
        if all(col in X.columns for col in ['sload', 'dload']):
            X['sload_dload_ratio'] = X['sload'] / (X['dload'] + 1e-5)
        if all(col in X.columns for col in ['sbytes', 'dbytes']):
            X['sbytes_dbytes_ratio'] = X['sbytes'] / (X['dbytes'] + 1e-5)
        if all(col in X.columns for col in ['smean', 'dmean']):
            X['smean_dmean_diff'] = X['smean'] - X['dmean']
        if all(col in X.columns for col in ['tcprrt', 'synack']):
            X['tcprrt_synack_diff'] = X['tcprrt'] - X['synack']
        return X
```

### 3.4.5 Drop Features

Dropping features digunakan untuk mengurangi jumlah data yang ada dan mempercepat proses modelling. Dengan melakukan drop pada fitur-fitur yang berkorelasi rendah dan hanya berfokus pada fitur yang memiliki korelasi tinggi dengan kelas target diharapkan dapat meningkatkan kualitas hasil prediksi.

```
class FeatureDropper(BaseEstimator, TransformerMixin):

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        columns_to_keep = [
            'dur', 'smean', 'ct_flw_http_mthd', 'tcprrt', 'sload',
            'ackdat', 'dtti', 'stti', 'ct_state_tti', 'state',
            'service', 'attack_cat'
        ]
```



```
X = X[[col for col in columns_to_keep if col in X.columns]]
return X
```

### 3.4.6 Dimensionality Reduction

Dimensionality Reduction digunakan untuk mengurangi jumlah fitur (dimensi) dalam kumpulan data. Dalam algoritma dibawah ini, kami mengambil 10 fitur menggunakan teknik PCA.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)

print("Dimensi X_train setelah PCA:", X_train.shape)
print("Dimensi X_test setelah PCA:", X_test.shape)
```

### 3.5 Tahap Modeling dan Validasi

Dalam melakukan modeling dilakukan *splitting* pada data train dengan proporsi (20:80) untuk training dan test, hal ini dilakukan untuk mempelajari dan menggeneralisasi pola selama *training* data. Data tes akan digunakan untuk mengevaluasi model pada data dan memeriksa keakuratan dari model tersebut. Pada proses ini juga dilakukan dengan mengaplikasikan `random_state = 42` dan `stratify=y` agar distribusi kelas target lebih seragam.

```
X = train_set.drop(['attack_cat'], axis=1)
y = train_set['attack_cat']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Untuk modeling diambil tiga model yakni, KNN, Naive Bayes, dan ID3 yang diimplementasikan dari *scratch* untuk memprediksi hasil. Proses validasi dilakukan dengan melihat hasil *cross validation score* dari masing-masing model. Hasil *Average cross-validation* dari hasil modeling untuk algoritma KNN, Naive Bayes, dan ID3, sebagai berikut:

- KNN: 0.28418294835914015
- Naive Bayes: 0.22549225702278392
- ID3: 0.3233508894228158

Dari hasil validasi didapatkan bahwa ID3 memberikan hasil terbaik dalam memprediksi data diikuti dengan KNN lalu Naive Bayes. Hal ini disebabkan karena dataset yang diberikan sangat kompleks memiliki banyak fitur dan jumlah data yang besar. Oleh karena itu ID3 cocok sebagai model dalam memprediksi data ini. Karena ID3 memberikan hasil terbaik saat melakukan validasi menggunakan data training. Maka akan dilakukan prediksi menggunakan model ID3 pada *test data*.

## BAB IV

### PERBANDINGAN HASIL PREDIKSI

#### 4.1 Perbandingan Implementasi KNN dengan library

Hasil Implementasi
<p>k = 5 , metric = manhattan</p> <pre>X_test= pca.fit_transform(X_test)  knn_model = knn(k=5, metric='manhattan') knn_scores = cross_val_score(knn_model, X_train, y_train, cv=5)  print("Cross-validation scores:", knn_scores) print("Average cross-validation score:", np.mean(knn_scores))  # Fit and predict on the test set knn_model.fit(X_train, y_train) y_pred = knn_model.predict(X_test) print( "y_pred",y_pred) print(len(y_pred))  print("Accuracy on test set:", accuracy_score(y_test, y_pred))</pre> <p>Cross-validation scores: [0.41770788 0.36966825 0.37233355 0.37664296 0.36867055] Average cross-validation score: 0.38100463796031125 y_pred ['Backdoor' 'Backdoor' 'Backdoor' ... 'Generic' 'Normal' 'Backdoor'] 5802 Accuracy on test set: 0.36280592899000347</p>
Hasil Library

```

from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

```

```

knn_model = KNeighborsClassifier(n_neighbors=5, metric='manhattan')
knn_scores = cross_val_score(knn_model, X_train, y_train, cv=5)
print("Cross-validation scores:", knn_scores)
print("Average cross-validation score:", np.mean(knn_scores))
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
print("Predicted values:", y_pred)
print("Number of predictions:", len(y_pred))
print("Accuracy on test set:", accuracy_score(y_test, y_pred))

```

Cross-validation scores: [0.4196467 0.39013356 0.37642749 0.38181426 0.36996337]  
 Average cross-validation score: 0.3875970790662734  
 Predicted values: ['Analysis' 'Analysis' 'Backdoor' ... 'Generic' 'Normal' 'Analysis']  
 Number of predictions: 5802  
 Accuracy on test set: 0.36297828335056875

### Hasil Implementasi

k = 5 , metric = euclidean

```

from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
X_test= pca.fit_transform(X_test)
knn_model = KNN(k=5, metric='euclidean')
knn_scores = cross_val_score(knn_model, X_train, y_train, cv=5)
print("Cross-validation scores:", knn_scores)
print("Average cross-validation score:", np.mean(knn_scores))
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
print( "y_pred",y_pred)
print(len(y_pred))
print("Accuracy on test set:", accuracy_score(y_test, y_pred))

```

Cross-validation scores: [0.41770788 0.37031452 0.37297996 0.37599655 0.36867055]  
 Average cross-validation score: 0.38113389259193553  
 y\_pred ['Backdoor' 'Backdoor' 'Backdoor' ... 'Generic' 'Normal' 'Backdoor']  
 5802  
 Accuracy on test set: 0.35436056532230265

### Hasil Library

```

from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

```

```

knn_model = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_scores = cross_val_score(knn_model, X_train, y_train, cv=5)
print("Cross-validation scores:", knn_scores)
print("Average cross-validation score:", np.mean(knn_scores))
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
print("Predicted values:", y_pred)
print("Number of predictions:", len(y_pred))
print("Accuracy on test set:", accuracy_score(y_test, y_pred))

```

```

Cross-validation scores: [0.41943128 0.39034899 0.37772032 0.38052144 0.37017884]
Average cross-validation score: 0.3876401732270146
Predicted values: ['Analysis' 'Analysis' 'Backdoor' ... 'Generic' 'Normal' 'Analysis']
Number of predictions: 5802
Accuracy on test set: 0.35832471561530505

```

### Hasil Implementasi

k = 5 , metric = minkowski

```

from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
X_test= pca.fit_transform(X_test)
knn_model = kNN(k=5, metric='minkowski')
knn_scores = cross_val_score(knn_model, X_train, y_train, cv=5)
print("Cross-validation scores:", knn_scores)
print("Average cross-validation score:", np.mean(knn_scores))
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
print( "y_pred",y_pred)
print(len(y_pred))
print("Accuracy on test set:", accuracy_score(y_test, y_pred))

```

```

Cross-validation scores: [0.41770788 0.37031452 0.37297996 0.37599655 0.36867055]
Average cross-validation score: 0.38113389259193553
y_pred ['Backdoor' 'Backdoor' 'Backdoor' ... 'Generic' 'Normal' 'Backdoor']
5802
Accuracy on test set: 0.35436056532230265

```

### Hasil Library

```

from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

knn_model = KNeighborsClassifier(n_neighbors=5, metric='minkowski')
knn_scores = cross_val_score(knn_model, X_train, y_train, cv=5)
print("Cross-validation scores:", knn_scores)
print("Average cross-validation score:", np.mean(knn_scores))
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
print("Predicted values:", y_pred)
print("Number of predictions:", len(y_pred))
print("Accuracy on test set:", accuracy_score(y_test, y_pred))

Cross-validation scores: [0.41943128 0.39034899 0.37772032 0.38052144 0.37017884]
Average cross-validation score: 0.3876401732270146
Predicted values: ['Analysis' 'Analysis' 'Backdoor' ... 'Generic' 'Normal' 'Analysis']
Number of predictions: 5802
Accuracy on test set: 0.35832471561530505

```

Berdasarkan perbandingan di atas dapat diketahui bahwa dari 3 hasil implementasi algoritma KNN *from scratch* dibandingkan dengan menggunakan *library* mendapatkan nilai *cross-validation* dan *Accuracy on test set* yang cenderung mirip. Pada Algoritma KNN mendapatkan rata-rata 0.356, sedangkan jika menggunakan *library* mendapat nilai rata-rata 0.359. Hal ini menandakan bahwa algoritma yang diimplementasikan sudah benar dan dapat memprediksi kelas target seperti menggunakan *library* KNeighborsClassifier. Selain itu, untuk perbandingan tingkat akurasi manhattan memberikan nilai akurasi yang paling baik karena dataset yang digunakan berdimensi tinggi.

#### 4.2 Perbandingan Implementasi Naive Bayes dengan library

<p style="text-align: center;"><b>Hasil Implementasi</b></p>
--

```

from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
X_test= pca.fit_transform(X_test)

naivebayes_model = NaiveBayes()
nb_scores = cross_val_score(naivebayes_model, X_train, y_train, cv=5)

print("Cross-validation scores:", nb_scores)
print("Average cross-validation score:", np.mean(nb_scores))

# Fit and predict on the test set
naivebayes_model.fit(X_train, y_train)
y_pred = naivebayes_model.predict(X_test)
print("y_pred",y_pred)
print(len(y_pred))

print("Accuracy on test set:", accuracy_score(y_test, y_pred))

```

✓ 62s

C:\Users\Asus\AppData\Local\Temp\ipykernel\_16424\3751977710.py:42: RuntimeWarning: divide by zero encountered in log  
likelihood = np.sum(np.log(self.calculate\_likelihood(x, self.class\_means[label], self.class\_variances[label])))  
Cross-validation scores: [0.27509694 0.28716071 0.26459815 0.28614523 0.27580263]  
Average cross-validation score: 0.2777607301164591  
y\_pred ['Shellcode' 'Shellcode' 'Reconnaissance' ... 'Reconnaissance' 'Normal'  
'Shellcode']  
5802  
Accuracy on test set: 0.17873147190623923

### Hasil Library

```

>
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.naive_bayes import GaussianNB
X_test= pca.fit_transform(X_test)

naivebayes_model = GaussianNB()
nb_scores = cross_val_score(naivebayes_model, X_train, y_train, cv=5)

print("Cross-validation scores:", nb_scores)
print("Average cross-validation score:", np.mean(nb_scores))

# Fit and predict on the test set
naivebayes_model.fit(X_train, y_train)
y_pred = naivebayes_model.predict(X_test)
print("y_pred",y_pred)
print(len(y_pred))

print("Accuracy on test set:", accuracy_score(y_test, y_pred))

```

[116] ✓ 02s

''' Cross-validation scores: [0.27725118 0.28780698 0.27537169 0.28506787 0.27731092]  
Average cross-validation score: 0.28056172987870787  
y\_pred ['Shellcode' 'Shellcode' 'Reconnaissance' ... 'Reconnaissance' 'Normal'  
'Shellcode']  
5802  
Accuracy on test set: 0.17873147190623923

Berdasarkan perbandingan di atas, diketahui bahwa hasil implementasi algoritma Gaussian Naive Bayes *from scratch* dengan menggunakan *library* mendapatkan nilai *cross-validation* dan *Accuracy on test set* yang cenderung mirip. Pada Algoritma Naive

Bayes mendapatkan rata-rata 0.2777 dan jika menggunakan *library* mendapat nilai rata-rata 0.2805. Hal ini menandakan bahwa algoritma yang diimplementasikan mendekati fungsi Naive Bayes dari *library* dan dapat memprediksi kelas target seperti menggunakan *library* GaussianNB.

#### 4.3 Perbandingan Implementasi ID3 dengan library

Hasil Implementasi	
<pre> from sklearn.preprocessing import LabelEncoder from sklearn.metrics import accuracy_score from sklearn.model_selection import cross_val_score, train_test_split encoder_X = LabelEncoder() encoder_y = LabelEncoder()  X_train_encoded = np.array([encoder_X.fit_transform(X_train[:, i]) for i in range(X_train.shape[1])]).T y_train_encoded = encoder_y.fit_transform(y_train)  id3_model = ID3(max_depth=5) id3_scores = cross_val_score(id3_model, X_train_encoded, y_train_encoded, cv=5)  print("Cross-validation scores:", id3_scores) print("Average cross-validation score:", np.mean(id3_scores))  X_test_encoded = np.array([encoder_X.fit_transform(X_test[:, i]) for i in range(X_train.shape[1])]).T y_test_encoded = encoder_y.fit_transform(y_test)  id3_model.fit(X_train_encoded, y_train_encoded) y_pred = id3_model.predict(X_test_encoded)  print("Accuracy on test set:", accuracy_score(y_pred, y_test_encoded)) </pre>	
	<pre> Cross-validation scores: [0.32162861 0.32098234 0.32342168 0.32686921 0.32385262] Average cross-validation score: 0.3233508894228158 Accuracy on test set: 0.17597380213719407 </pre>
Hasil Library	



```

from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import numpy as np

encoder_X = LabelEncoder()
encoder_y = LabelEncoder()

X_train_encoded = np.array([encoder_X.fit_transform(X_train[:, i]) for i in range(X_train.shape[1])]).T
X_test_encoded = np.array([encoder_X.fit_transform(X_test[:, i]) for i in range(X_test.shape[1])]).T

y_train_encoded = encoder_y.fit_transform(y_train)
y_test_encoded = encoder_y.fit_transform(y_test)


dt_model = DecisionTreeClassifier(max_depth=10)

dt_scores = cross_val_score(dt_model, X_train_encoded, y_train_encoded, cv=5)
print("Cross-validation scores:", dt_scores)
print("Average cross-validation score:", np.mean(dt_scores))

dt_model.fit(X_train_encoded, y_train_encoded)
y_pred = dt_model.predict(X_test_encoded)

print("Accuracy on test set:", accuracy_score(y_test_encoded, y_pred))

```

 Cross-validation scores: [0.32701422 0.32313658 0.32234432 0.32643827 0.32385262]  
 Average cross-validation score: 0.32455720099991076  
 Accuracy on test set: 0.15270596346087556

Berdasarkan perbandingan di atas dapat diketahui bahwa hasil implementasi algoritma ID3 *from scratch* dengan menggunakan *library* mendapatkan nilai *cross-validation* dan *Accuracy on test set* yang cenderung mirip. Pada Algoritma ID3 mendapatkan rata-rata 0.323 dan jika menggunakan *library* mendapat nilai rata-rata 0.324. Hal ini menandakan bahwa algoritma yang diimplementasikan sudah benar dan dapat memprediksi kelas target seperti menggunakan *library* DecisionTreeClassifier.

## **BAB V**

### **KESIMPULAN**

#### **5.1 Kesimpulan**

Selama proses mengerjakan tugas besar ini, kami semakin memahami proses untuk membangun model prediksi yang tepat dalam analisis data. Berdasarkan analisis yang telah dilakukan, algoritma yang paling akurat dalam memprediksi dataset UNSW NB15 adalah Algoritma ID3 karena algoritma ini dapat menangani data numerik dan kategorikal sekaligus, fleksibel dalam mengolah jenis fitur yang beragam dan kompleks dalam dataset, dan memiliki interpretasi hasil yang lebih jelas mengenai logika di balik keputusan yang diambil oleh model.

#### **5.2 Saran**

Selama pengerjaan tugas besar ini kami memiliki beberapa saran untuk pengerjaan tugas besar berikutnya, di antaranya:


1. Untuk pengembangan berikutnya, bisa dilakukan analisis yang lebih mendalam sebelum melakukan *pre-processing* data.
2. Kode program yang digunakan bisa dibuat lebih optimal dengan memanfaatkan *sampling*, *scaler*, dan lainnya.

## KONTRIBUSI

NIM	Nama	Kontribusi
13522012	Thea Josephine Halim	Algoritma Naive Bayes, Laporan
13522025	Debrina Veisha Rashika W	Algoritma ID3, Laporan
13522035	Melati Anggraini	Algoritma KNN, Laporan
13522046	Raffael Boymian Siahaan	PreProcessing, Laporan

## LAMPIRAN

**Tautan Repository GitHub :** [https://github.com/slnklr01/IF3170\\_Tubes2\\_UNSW-NB15](https://github.com/slnklr01/IF3170_Tubes2_UNSW-NB15)

**Link Colab :**  PengcarryAI | Tugas Besar 2 Notebook.ipynb (digunakan versi 15/12/2024 Pk 17:31 WIB)

## DAFTAR PUSTAKA

- Chauhan, N. (2024). Naïve Bayes Algorithm: Everything You Need to Know. Diakses dari kdnuggets: [Naïve Bayes Algorithm: Everything You Need to Know - KDnuggets](#).
- Coursesteach. (2024). Natural Language Processing(Part 17)-Laplacian Smoothing, [Natural Language Processing\(Part 17\)-Laplacian Smoothing](#) (diakses pada Des. 14, 2024).
- Chowta, G. (2020). Gaussian Naive Bayes from scratch in python, [Gaussian Naive Bayes from scratch in python | Kaggle](#) (diakses pada Des. 15, 2024).
- GeeksforGeeks, “K-Nearest Neighbor(KNN) Algorithm” geeksforgeeks, [K-Nearest Neighbor\(KNN\) Algorithm - GeeksforGeeks](#) (diakses pada Des. 10, 2024).
- GeeksforGeeks, “K-Nearest Neighbor(KNN) Algorithm” geeksforgeeks, [Sklearn | Iterative Dichotomiser 3 \(ID3\) Algorithms - GeeksforGeeks](#) (diakses pada Des. 15, 2024).
- J. Tareq. (2021). Step by Step Decision Tree: ID3 Algorithm From Scratch in Python [No Fancy Library], [Step by Step Decision Tree: ID3 Algorithm From Scratch in Python \[No Fancy Library\] | by Tareq Rahman Joy | Geek Culture | Medium](#) (diakses pada Des. 15, 2024).
- Sakkaf, Y., Decision Trees: ID3 Algorithm Explained [Decision Trees: ID3 Algorithm Explained | Towards Data Science](#) (diakses pada Des. 11, 2024).