

# **Tugas Besar 2**

## **Implementasi Algoritma Pembelajaran Mesin**

**IF3170 Inteligensi Artifisial**



Kelompok 21 Beban Kaggle

Elbert Chailes 13522045

Juan Alfred Widjaya 13522073

Albert 13522081

William Glory Henderson 13522113

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2024**

# Daftar Isi

|   |           |
|---|-----------|
| <b>Daftar Isi</b>   | <b>2</b>  |
| <b>BAB I : Implementasi Model</b>   | <b>3</b>  |
| 1.1. Implementasi KNN   | 3         |
| 1.2. Implementasi Gaussian Naive-Bayes  | 6         |
| 1.3. Implementasi ID3   | 9         |
| <b>BAB II : Cleaning dan Preprocessing data</b>                               | <b>20</b> |
| 2.1. Tahap Cleaning   | 20        |
| 2.1.1. Handling Missing Data  | 20        |
| 2.1.2. Dealing with Outliers  | 20        |
| 2.1.3. Remove Duplicates  | 20        |
| 2.1.4. Feature Engineering  | 20        |
| 2.2. Tahap Preprocessing  | 21        |
| 2.2.1. Feature Scaling  | 21        |
| 2.2.2. Feature Encoding   | 21        |
| 2.2.3. Handling Imbalanced Dataset  | 22        |
| 2.2.4. Data Normalization   | 22        |
| 2.2.5. Dimensionality Reduction   | 22        |
| <b>BAB III : Analisis</b>   | <b>23</b> |
| 3.1. Perbandingan Hasil Prediksi Algoritma KNN Scratch dengan Pustaka         | 23        |
| 3.1.1. Algoritma KNN dengan Jarak Euclidean                                   | 23        |
| 3.1.2. Algoritma KNN dengan Jarak Manhattan                                   | 23        |
| 3.1.3. Algoritma KNN dengan Jarak Minkowski                                   | 24        |
| 3.1.4. Algoritma KNN dengan Jarak Hamming                                     | 25        |
| 3.2. Perbandingan Hasil Prediksi Algoritma Naive Bayes Scratch dengan Pustaka | 25        |

|   |           |
|---|-----------|
| 3.3. Perbandingan Hasil Prediksi Algoritma ID3 Scratch dengan Pustaka | 26        |
| <b>Referensi</b>  | <b>28</b> |
| <b>Lampiran</b>   | <b>29</b> |

# BAB I : Implementasi Model

## 1.1. Implementasi KNN

K-Nearest Neighbors (KNN) adalah algoritma klasifikasi yang memprediksi kelas data berdasarkan kedekatannya dengan titik-titik lain dalam ruang fitur. Dalam algoritma KNN, setiap data uji akan diprediksi kelasnya dengan mengidentifikasi kelas dari k tetangga terdekat yang berasal dari data latih yang sudah diketahui kelasnya. Implementasi algoritma ini memungkinkan pengguna untuk memilih metrik jarak yang digunakan dalam perhitungan kedekatan antar titik data.

Dalam rangka untuk melakukan implementasi algoritma KKN tersebut, terdapat langkah-langkah yang dilakukan, yaitu sebagai berikut.

### a. Persiapan Data

Sebelum memulai proses klasifikasi, data latih dan data uji akan dipersiapkan terlebih dahulu. Data latih berfungsi untuk melatih model, sementara data uji digunakan untuk memvalidasi prediksi yang dihasilkan oleh model. Pada tahap ini, setiap data akan diubah ke dalam format NumPy ndarray menggunakan fungsi `_ensure_ndarray` untuk meningkatkan efisiensi pemrosesan.

### b. Perhitungan Jarak

Setelah data siap, langkah berikutnya adalah menghitung jarak antara setiap titik data uji dengan seluruh titik data latih. Pada implementasi ini, fungsi `cdist` dari pustaka `scipy.spatial.distance` digunakan untuk menghitung jarak antar titik data dengan berbagai pilihan metrik jarak. Metrik jarak yang paling umum digunakan adalah jarak Euclidean, namun pengguna dapat memilih metrik lain sesuai kebutuhan. Terdapat 4 jenis metrik yang diimplementasikan pada model KNN oleh penulis.

- i. Jarak Euclidean merupakan jarak yang mengukur jarak garis lurus antara dua titik dalam ruang multidimensi, menggunakan rumus

$$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- ii. Jarak Manhattan merupakan jarak yang mengukur jarak sepanjang sumbu antara dua titik, menggunakan rumus  $\sum_{i=1}^n |x_i - y_i|$ .
- iii. Jarak Minkowski merupakan generalisasi dari jarak Euclidean dan Manhattan, dengan rumus  $(\sum_{i=1}^n |x_i - y_i|^p)^{\frac{1}{p}}$ , di mana  $p$  menentukan jenis jarak.
- iv. Jarak Hamming mengukur jumlah posisi berbeda antara dua vektor dengan panjang yang sama, cocok untuk data kategorikal atau biner.

c. Menentukan Tetangga Terdekat

Berdasarkan perhitungan jarak antara data uji dan data latih, algoritma akan memilih k tetangga terdekat. Nilai k ini adalah parameter yang dapat disesuaikan sesuai dengan kebutuhan. Semakin kecil nilai k, semakin sensitif model terhadap perubahan data, sedangkan semakin besar nilai k, model cenderung lebih stabil namun mungkin kehilangan beberapa detail yang penting. Pemilihan nilai k juga harus disesuaikan.

d. Klasifikasi Berdasarkan Mayoritas

Setelah mendapatkan k tetangga terdekat, langkah selanjutnya adalah menentukan kelas dari data uji tersebut. Kelas data uji ditentukan berdasarkan mayoritas kelas dari k tetangga terdekat. Artinya, jika sebagian besar dari k tetangga memiliki kelas tertentu, maka data uji tersebut akan diklasifikasikan ke kelas yang paling banyak ditemukan di antara tetangga-tetangga tersebut.

Berdasarkan langkah-langkah di atas dan penjelasan implementasinya, maka implementasi dalam kode dapat dilihat pada poin berikut.

a. Fungsi `__init__(self, k: int = 3, distance_metric: str = 'euclidean')`

```

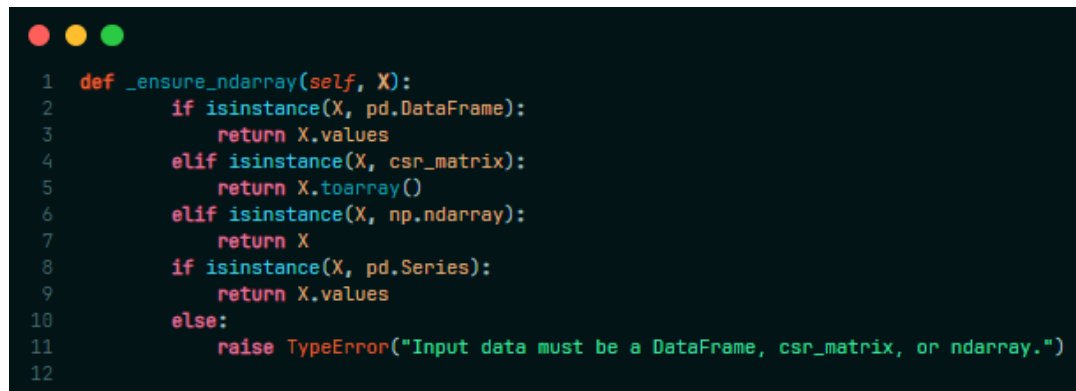
1 class KNN:
2     def __init__(self, k: int = 3, distance_metric: str = 'euclidean'):
3         self.k = k
4         self.distance_metric = distance_metric

```

**Gambar 1.1.1.** Cuplikan kode untuk melakukan inisialisasi model KNN

Pada bagian ini, kita menginisialisasi objek KNN dengan dua parameter utama: jumlah tetangga terdekat ( $k$ ) dan metrik jarak yang digunakan untuk menghitung kedekatan antar titik data. Pada percobaan digunakan nilai  $k = 5$ ;

**b. Fungsi `_ensure_ndarray(self, X)`**

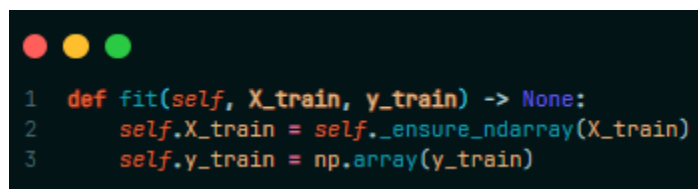
A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Python and defines a function named `_ensure_ndarray` that takes `self` and `X` as arguments. The function uses a series of `if` and `elif` statements to check the type of `X`: if it's a `pd.DataFrame`, it returns `X.values`; if it's a `csr_matrix`, it returns `X.toarray()`; if it's a `np.ndarray`, it returns `X`; if it's a `pd.Series`, it returns `X.values`; otherwise, it raises a `TypeError` with the message "Input data must be a DataFrame, csr\_matrix, or ndarray." The lines are numbered from 1 to 12.

```
1 def _ensure_ndarray(self, X):
2     if isinstance(X, pd.DataFrame):
3         return X.values
4     elif isinstance(X, csr_matrix):
5         return X.toarray()
6     elif isinstance(X, np.ndarray):
7         return X
8     if isinstance(X, pd.Series):
9         return X.values
10    else:
11        raise TypeError("Input data must be a DataFrame, csr_matrix, or ndarray.")
12
```

**Gambar 1.1.2.** Cuplikan kode untuk melakukan validasi data

Fungsi ini memastikan bahwa data yang dimasukkan ke model KNN berada dalam format yang sesuai, yaitu array NumPy (`ndarray`). Fungsi ini akan memeriksa tipe data input dan mengkonversinya jika diperlukan. Jika data berupa Data Frame atau Series dari pandas, fungsi ini mengkonversinya menjadi array NumPy. Jika data berupa matriks sparse (`csr_matrix`), fungsi ini mengubahnya menjadi array biasa. Jika data sudah dalam bentuk `ndarray`, fungsi ini langsung mengembalikannya tanpa perubahan. Jika tipe data tidak sesuai, fungsi ini akan melemparkan `TypeError`, memastikan hanya tipe data yang kompatibel yang diproses oleh model.

**c. Fungsi `fit(self, X_train, y_train)`**

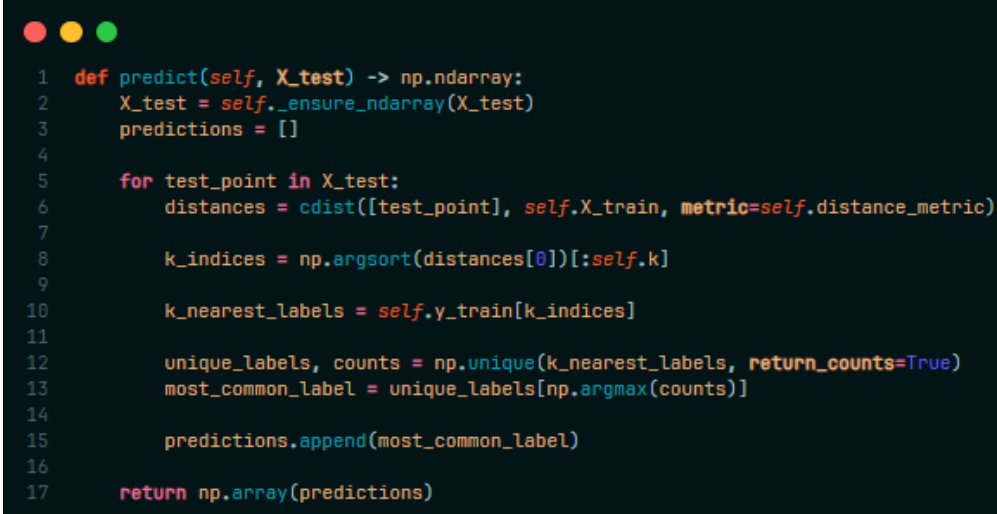
A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Python and defines a function named `fit` that takes `self`, `X_train`, and `y_train` as arguments. The function has a return type annotation of `-> None`. It calls `self._ensure_ndarray(X_train)` to process the training features and then converts `y_train` to a NumPy array using `np.array(y_train)`. The lines are numbered from 1 to 3.

```
1 def fit(self, X_train, y_train) -> None:
2     self.X_train = self._ensure_ndarray(X_train)
3     self.y_train = np.array(y_train)
```

**Gambar 1.1.3.** Cuplikan kode untuk menyimpan data latih yang telah diproses

Fungsi fit bertugas untuk menyimpan data latih yang telah diproses dan mengkonversinya menjadi format yang dapat diproses oleh model. Setelah itu, data latih akan disimpan dalam atribut `X_train` dan `y_train` untuk digunakan pada tahap prediksi.

d. Fungsi `predict(self, X_test)`



```
1 def predict(self, X_test) -> np.ndarray:
2     X_test = self._ensure_ndarray(X_test)
3     predictions = []
4
5     for test_point in X_test:
6         distances = cdist([test_point], self.X_train, metric=self.distance_metric)
7
8         k_indices = np.argsort(distances[0])[:self.k]
9
10        k_nearest_labels = self.y_train[k_indices]
11
12        unique_labels, counts = np.unique(k_nearest_labels, return_counts=True)
13        most_common_label = unique_labels[np.argmax(counts)]
14
15        predictions.append(most_common_label)
16
17    return np.array(predictions)
```

Gambar 1.1.4. Cuplikan kode untuk memprediksi kelas data uji

Fungsi ini digunakan untuk memprediksi kelas data uji berdasarkan data latih yang sudah diproses. Fungsi ini melakukan beberapa langkah penting: pertama, menghitung jarak antara data uji dengan data latih menggunakan metrik yang telah ditentukan, kemudian memilih `k` tetangga terdekat. Terakhir, kelas dari data uji diprediksi berdasarkan mayoritas kelas dari tetangga terdekat.

## 1.2. Implementasi Gaussian Naive-Bayes

Gaussian Naive Bayes (GNB) merupakan algoritma *probabilistic classification* yang berdasarkan *Bayes' Theorem*. Algoritma ini mengasumsikan bahwa setiap fitur dari setiap *class* mengikuti distribusi Gaussian (normal). Selain itu, algoritma ini juga mengasumsikan bahwa semua fitur bersifat *conditionally independent* terhadap kelas tertentu. Dengan sifat dan asumsi ini, Gaussian Naive Bayes cenderung efektif untuk data yang sifatnya kontinu.

Dalam implementasi dalam bentuk kode program, implementasi *Gaussian Naive-Bayes* direpresentasikan oleh kelas Naive Bayes yang terdapat pada file `NaiveBayes.py` pada *repository*

yang terdapat pada lampiran. Langkah-langkah perlakuan implementasi dalam kode dapat dilihat sebagai berikut.

a. Inisialisasi kelas NaiveBayes

Pada tahap inisialisasi ini, terdapat parameter berupa **var\_smoothing** dengan tujuan untuk meng-*handle* isu ketika nilai varians pada kelas tertentu sangat kecil. Dengan adanya parameter ini, maka nilai kecil ini akan ditambahkan ke varians untuk mencegah masalah numerik, seperti kasus pembagian dengan nol.

Selain itu, terdapat juga atribut-atribut kelas yang harus disimpan seperti berikut.

- i. `classes_` : *class* unik di dataset
- ii. `class_prior_` : *prior probabilities* dari setiap *class*
- iii. `theta_` : nilai mean untuk setiap fitur untuk setiap kelas
- iv. `sigma_` : nilai varians untuk setiap fitur untuk setiap kelas

$$P(X_j | C_i) = -\frac{1}{2} \log(2\pi\sigma_{ij}^2) - \frac{(X_j - \mu_{ij})^2}{2\sigma_{ij}^2}$$

$$\sum_{j=1}^{n_{features}} \log P(X_j | C_i) = -\frac{1}{2} \sum_{j=1}^{n_{features}} \log(2\pi\sigma_{ij}^2) - \frac{1}{2} \sum_{j=1}^{n_{features}} \frac{(X_j - \mu_{ij})^2}{\sigma_{ij}^2}$$

$$\log P(C_i | X) \propto \log P(C_i) + \sum_{j=1}^{n_{features}} \log P(X_j | C_i)$$

**Formula 1.2.1.** Log prior + Log likelihood Formula

Note:

- i merepresentasikan indeks *class*, melakukan looping pada `self.classes_`
- j merepresentasikan indeks *feature* (column)

b. Melakukan training model (dengan fungsi **fit()**)

Cuplikan kode implementasi fungsi `fit()` dapat dilihat pada **Gambar 1.2.1**. Pada implementasi ini, langkah-langkah yang dilakukan adalah sebagai berikut.

- i. `self.classes_` terlebih dahulu ditentukan sebagai inisialisasi, dengan mengisi atribut tersebut dengan nama label unik yang terdapat pada kolom y.



- ii. Kemudian, dilakukan filtrasi sampel untuk setiap kelas ( $X_c$ ).  $X_c$  tersebut kemudian dilakukan perhitungan *prior probability* dan disimpan ke **self.class\_prior\_**, dengan data berupa banyak data sampel untuk kelas tersebut dibagi dengan total banyak data.
- iii. Kalkulasi rata-rata (*mean*) dari setiap fitur untuk kelas tersebut dihitung dan disimpan ke dalam atribut **self.theta\_**. Varians dari setiap fitur untuk kelas tersebut juga dihitung dan disimpan ke dalam atribut **self.sigma\_**. Untuk memastikan stabilitas numerik, sebuah nilai kecil (**var\_smoothing**) ditambahkan ke varians.

```

1 def fit(self, X: Union[pd.DataFrame, np.ndarray], y: Union[pd.Series, np.ndarray]):
2     """
3     Fit Gaussian Naive Bayes according to X, y.
4
5     Args:
6         X (DataFrame or ndarray): Training vectors
7         y (Series or ndarray): Target values
8
9     Returns:
10        self: Fitted estimator
11    """
12    X = X.values if isinstance(X, pd.DataFrame) else X
13    y = y.values if isinstance(y, pd.Series) else y
14
15    self.classes_ = np.unique(y)
16    n_classes = len(self.classes_)
17    n_features = X.shape[1]
18
19    self.theta_ = np.zeros((n_classes, n_features))
20    self.sigma_ = np.zeros((n_classes, n_features))
21    self.class_prior_ = np.zeros(n_classes)
22
23    for i, c in enumerate(self.classes_):
24        X_c = X[y == c]
25        self.class_prior_[i] = X_c.shape[0] / X.shape[0]
26
27        self.theta_[i, :] = X_c.mean(axis=0)
28
29        var = X_c.var(axis=0)
30
31        var_max = np.max(var) if len(var) > 0 else 1.0
32        self.sigma_[i, :] = var + self.var_smoothing * var_max
33
34    return self

```

**Gambar 1.2.1.** Cuplikan kode metode **fit()** dalam implementasi algoritma Gaussian Naive Bayes

c. Melakukan prediksi (dengan fungsi **predict()**)

Sebelum melakukan prediksi, maka untuk setiap kelas perlu dihitung nilai *log likelihood*-nya. Kalkulasi ini dilakukan pada fungsi **\_joint\_log\_likelihood()** yang implementasinya dapat dilihat pada **Gambar 1.2.2**. Fungsi ini merupakan representasi kode dari **Formula 1.2.1**.

```

1 def _joint_log_likelihood(self, X: np.ndarray) -> np.ndarray:
2     """
3     Calculate joint log likelihood.
4
5     Args:
6         X (ndarray): Input samples
7
8     Returns:
9         ndarray: Joint log likelihood
10    """
11    joint_log_likelihood = []
12    for i in range(len(self.classes_)):
13        jointi = np.log(self.class_prior_[i])
14
15        n_ij = -0.5 * np.sum(np.log(2. * np.pi * self.sigma_[i, :]))
16        n_ij -= 0.5 * np.sum(((X - self.theta_[i, :]) ** 2) / (self.sigma_[i, :]), axis=1)
17
18        joint_log_likelihood.append(jointi + n_ij)
19
20    joint_log_likelihood = np.array(joint_log_likelihood).T
21    return joint_log_likelihood
22

```

**Gambar 1.2.2.** Cuplikan kode implementasi fungsi `_joint_log_likelihood()`

Cara perhitungan ini merupakan adaptasi sesuai dengan formula yang telah ditulis pada **Formula 1.2.1**.

Setelah mendapatkan nilai log prior + log likelihood untuk setiap kelasnya, maka diambil kelas dengan nilai tertinggi sebagai hasil dari prediksi terhadap data X tersebut.

Fungsi `predict()` dapat dilihat pada **Gambar 1.2.3**.

```

1 def predict(self, X: Union[pd.DataFrame, np.ndarray]) -> np.ndarray:
2     """
3     Perform classification on an array of test vectors X.
4
5     Args:
6         X (DataFrame or ndarray): Input samples
7
8     Returns:
9         ndarray: Predicted class label for X
10    """
11    X = X.values if isinstance(X, pd.DataFrame) else X
12
13    jll = self._joint_log_likelihood(X)
14
15    return self.classes_[np.argmax(jll, axis=1)]

```

**Gambar 1.2.3.** Cuplikan kode implementasi fungsi `predict()`

### 1.3. Implementasi ID3

Model ID3 adalah model yang bekerja dengan cara membagi data menjadi beberapa subset berdasarkan nilai fitur tertentu, dan setiap pembagian bertujuan untuk memudahkan klasifikasi data ke dalam kategori-kategori yang relevan. ID3 adalah salah satu algoritma yang

digunakan untuk membangun pohon keputusan, yang mengandalkan konsep information gain untuk memilih fitur terbaik dalam membagi data di setiap simpul pohon.

Algoritma ID3 membangun pohon keputusan secara rekursif dengan memilih fitur yang dapat memberikan informasi paling banyak saat membagi data. Hal ini dilakukan dengan mengukur seberapa baik pembagian data berdasarkan entropi, yang menggambarkan ketidakpastian atau kekacauan dalam data. Semakin besar penurunan entropi setelah pembagian, semakin baik fitur tersebut dalam mengklasifikasikan data.

Berdasarkan penjelasan singkat terkait algoritma ID3, maka implementasi model dapat dipindahkan ke dalam kode program seperti berikut.

#### a. Class DecisionNode



```
1 class DecisionNode:
2     def __init__(
3         self,
4         feature: Optional[int] = None,
5         threshold: Optional[float] = None,
6         left: Optional[Union['DecisionNode', int]] = None,
7         right: Optional[Union['DecisionNode', int]] = None,
8         *,
9         value: Optional[int] = None
10    ):
11        self.feature = feature
12        self.threshold = threshold
13        self.left = left
14        self.right = right
15        self.value = value
```

**Gambar 1.3.1.** Cuplikan kode Class Decision Node

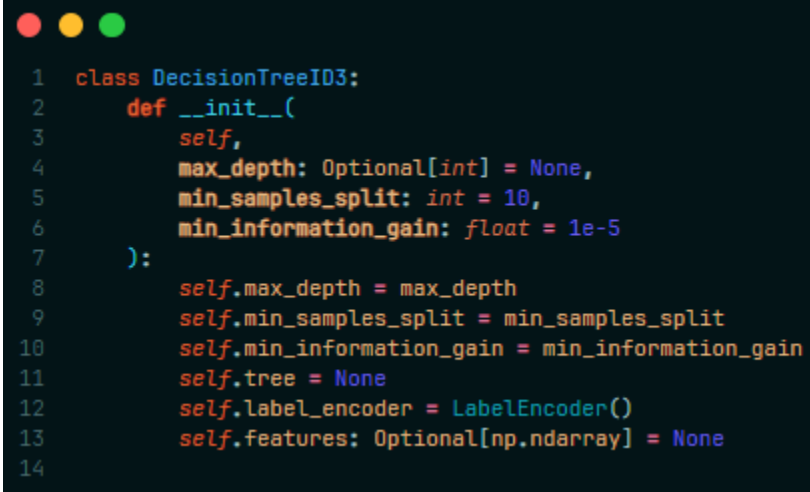
Kelas DecisionNode merepresentasikan sebuah simpul dalam pohon keputusan. Setiap simpul dalam pohon dapat memiliki dua tipe: simpul internal atau simpul daun. Simpul internal digunakan untuk membagi data berdasarkan fitur dan ambang batas tertentu, sedangkan simpul daun berisi nilai kelas yang diprediksi.

Pada kelas ini, setiap simpul menyimpan beberapa atribut penting:

- feature: Menyimpan indeks fitur yang digunakan untuk membagi data pada simpul tersebut.
- threshold: Menyimpan nilai ambang batas yang digunakan untuk membagi data berdasarkan nilai fitur.

- left dan right: Menyimpan referensi ke simpul anak kiri dan kanan. Anak kiri berisi data yang memenuhi kriteria pembagian, sedangkan anak kanan berisi data yang tidak memenuhi kriteria.
- value: Menyimpan nilai kelas pada simpul daun.

## b. Class DecisionTreeID3



```

1  class DecisionTreeID3:
2      def __init__(
3          self,
4          max_depth: Optional[int] = None,
5          min_samples_split: int = 10,
6          min_information_gain: float = 1e-5
7      ):
8          self.max_depth = max_depth
9          self.min_samples_split = min_samples_split
10         self.min_information_gain = min_information_gain
11         self.tree = None
12         self.label_encoder = LabelEncoder()
13         self.features: Optional[np.ndarray] = None
14

```

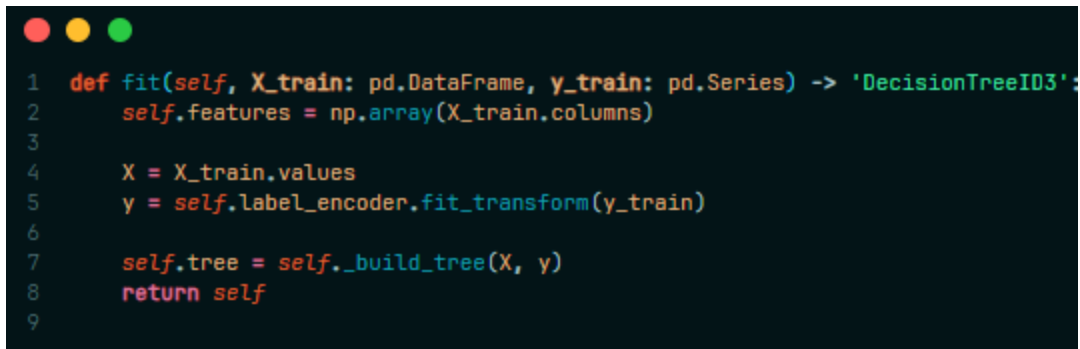
**Gambar 1.3.2.** Cuplikan kode Class DecisionTreeID3

Pada metode ini, dilakukan inisialisasi parameter-parameter yang digunakan dalam pembuatan pohon keputusan. Beberapa parameter yang diinisialisasi adalah:

- max\_depth: Menentukan kedalaman maksimal pohon. Ini bertujuan untuk mencegah pohon menjadi terlalu dalam, yang bisa menyebabkan overfitting.
- min\_samples\_split: Menentukan jumlah minimum sampel yang diperlukan agar suatu simpul dapat dibagi lebih lanjut.
- min\_information\_gain: Menentukan batas minimum information gain yang diperlukan untuk melakukan pembagian data pada suatu simpul.

Metode ini juga menginisialisasi label\_encoder, yang digunakan untuk mengubah label kelas menjadi nilai numerik. Hal ini diperlukan karena ID3 bekerja dengan data numerik untuk menghitung entropi dan information gain.

## c. Fungsi fit(self, X\_train: pd.DataFrame, y\_train: pd.Series)



```

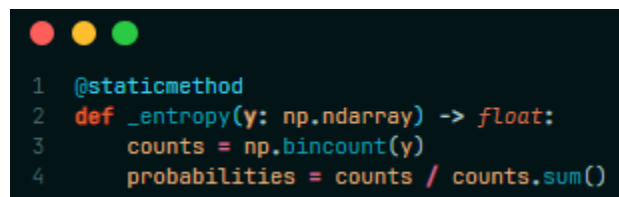
1  def fit(self, X_train: pd.DataFrame, y_train: pd.Series) -> 'DecisionTreeID3':
2      self.features = np.array(X_train.columns)
3
4      X = X_train.values
5      y = self.label_encoder.fit_transform(y_train)
6
7      self.tree = self._build_tree(X, y)
8      return self
9

```

**Gambar 1.3.3.** Cuplikan kode fit

Metode fit pada kelas DecisionTreeID3 bertujuan untuk melatih model pohon keputusan menggunakan data pelatihan. Pertama-tama, nama-nama kolom pada data fitur ( $X_{\text{train}}$ ) disalin ke dalam atribut `self.features`. Data fitur kemudian diubah menjadi format array NumPy agar dapat diproses oleh algoritma, sementara label target ( $y_{\text{train}}$ ) diubah menjadi nilai numerik dengan menggunakan LabelEncoder, yang memudahkan pengolahan data oleh model. Setelah itu, proses pembuatan pohon keputusan dimulai dengan memanggil metode `_build_tree`, yang akan memilih fitur terbaik untuk membagi data berdasarkan perhitungan information gain dan membuat cabang-cabang pohon sesuai dengan kriteria tersebut. Pohon keputusan yang terbentuk kemudian disimpan dalam atribut `self.tree`. Dengan metode ini, model akan siap untuk digunakan dalam memprediksi label pada data baru setelah tahap pelatihan selesai.

#### d. Fungsi `_entropy(y: np.ndarray)`



```

1  @staticmethod
2  def _entropy(y: np.ndarray) -> float:
3      counts = np.bincount(y)
4      probabilities = counts / counts.sum()

```

**Gambar 1.3.3.** Cuplikan kode `_entropy`

Metode ini digunakan untuk menghitung entropi dari target atau label kelas. Entropi merupakan ukuran ketidakpastian atau ketidakteraturan dalam data. Semakin tinggi entropi, semakin besar ketidakpastian dalam data tersebut. Rumus entropi yang digunakan adalah:

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

**Formula 1.3.3.** Cuplikan kode `_entropy`

Pada Formula 1.3.3.,  $p_i$  adalah probabilitas munculnya kelas  $i$ . Entropi yang lebih tinggi menunjukkan bahwa data tersebut lebih tidak teratur, sehingga membutuhkan lebih banyak informasi untuk memprediksi kelasnya.

**e. Fungsi `_best_split(self, X: np.ndarray, y: np.ndarray)`**

```

1 def _best_split(self, X: np.ndarray, y: np.ndarray) -> tuple:
2     parent_entropy = self._entropy(y)
3     _, n_features = X.shape
4     best_feature, best_threshold, best_info_gain = None, None, -1
5
6     for feature_idx in range(n_features):
7         column = X[:, feature_idx]
8         sorted_indices = np.argsort(column)
9         sorted_column = column[sorted_indices]
10        sorted_labels = y[sorted_indices]
11
12        unique_values, unique_indices = np.unique(sorted_column, return_index=True)
13        if len(unique_values) <= 1:
14            continue
15
16        thresholds = (sorted_column[unique_indices[:-1]] + sorted_column[unique_indices[1:]]) / 2
17
18        max_info_gain_idx, max_info_gain = self._calculate_info_gain(
19            sorted_labels, unique_indices, thresholds, parent_entropy
20        )
21
22        if max_info_gain > best_info_gain:
23            best_info_gain = max_info_gain
24            best_feature = feature_idx
25            best_threshold = thresholds[max_info_gain_idx]
26
27    return best_feature, best_threshold, best_info_gain

```

**Gambar 1.3.4.** Cuplikan kode `_best_split`

Metode `_best_split` bertugas untuk menentukan fitur terbaik dan nilai ambang batas (threshold) yang digunakan untuk membagi data pada setiap simpul pohon keputusan. Langkah pertama adalah menghitung entropy dari label target (y), yang digunakan sebagai ukuran ketidakpastian awal sebelum pembagian data. Untuk setiap fitur dalam dataset, algoritma mengurutkan nilai-nilai fitur tersebut dan mencocokkannya dengan label target yang sesuai. Hal ini memungkinkan algoritma untuk mencari titik pemisah yang optimal antara dua grup data. Titik pemisah ini ditemukan dengan

menghitung nilai tengah antara dua nilai fitur yang berbeda untuk setiap pasangan nilai berturut-turut.

Setelah mendapatkan titik-titik pemisah yang mungkin, metode ini kemudian menghitung information gain untuk setiap kemungkinan pembagian data menggunakan fungsi `_calculate_info_gain`. Information gain mengukur seberapa besar pengurangan ketidakpastian yang terjadi setelah pembagian. Pembagian yang menghasilkan information gain tertinggi dianggap sebagai pemisahan terbaik dan menjadi dasar untuk memilih fitur dan ambang batas yang akan digunakan. Fitur dan ambang batas dengan information gain tertinggi kemudian dikembalikan untuk digunakan dalam membangun pohon keputusan lebih lanjut.

- f. Fungsi `_calculate_info_gain` (`self`, `sorted_labels: np.ndarray`, `unique_indices: np.ndarray`, `thresholds: np.ndarray`, `parent_entropy: float`)

```
1 def _calculate_info_gain(  
2     self,  
3     sorted_labels: np.ndarray,  
4     unique_indices: np.ndarray,  
5     thresholds: np.ndarray,  
6     parent_entropy: float  
7 ) -> tuple:  
8     n_samples = len(sorted_labels)  
9     n_classes = len(np.unique(sorted_labels))  
10  
11     # Prepare arrays for entropy calculation  
12     entropies_left = np.zeros(len(thresholds))  
13     entropies_right = np.zeros(len(thresholds))  
14  
15     for cls in range(n_classes):  
16         cls_mask = sorted_labels == cls  
17         cls_cumsum = np.cumsum(cls_mask)  
18  
19         left_counts = cls_cumsum[unique_indices[1:] - 1]  
20         right_counts = cls_cumsum[-1] - left_counts  
21  
22         split_sizes_left = unique_indices[1:]  
23         split_sizes_right = n_samples - split_sizes_left  
24  
25         probs_left = np.divide(left_counts, split_sizes_left,  
26                                out=np.zeros_like(left_counts, dtype=float),  
27                                where=split_sizes_left != 0)  
28         probs_right = np.divide(right_counts, split_sizes_right,  
29                                 out=np.zeros_like(right_counts, dtype=float),  
30                                 where=split_sizes_right != 0)  
31  
32         entropies_left -= probs_left * np.log2(probs_left + 1e-9)  
33         entropies_right -= probs_right * np.log2(probs_right + 1e-9)  
34  
35     weighted_entropy = (  
36         (split_sizes_left / n_samples) * entropies_left +  
37         (split_sizes_right / n_samples) * entropies_right  
38     )  
39     info_gains = parent_entropy - weighted_entropy  
40  
41     max_info_gain_idx = np.argmax(info_gains)  
42     return max_info_gain_idx, info_gains[max_info_gain_idx]
```

**Gambar 1.3.5.** Cuplikan kode `_calculate_info_gain`

Metode `_calculate_info_gain` digunakan untuk menghitung information gain dari setiap kemungkinan pembagian data berdasarkan threshold yang ada. Proses dimulai dengan menentukan jumlah sampel (`n_samples`) dan jumlah kelas yang terdapat dalam label target (`y`). Selanjutnya, dua array kosong disiapkan untuk menghitung entropy di sisi kiri dan kanan dari setiap pembagian (`split`). Untuk setiap kelas dalam target label, algoritma pertama-tama membuat sebuah mask yang menandakan mana saja data yang termasuk dalam kelas tersebut. Kemudian, dihitung jumlah kumulatif data dalam setiap kelas tersebut di sepanjang fitur yang diurutkan. Berdasarkan nilai-nilai kumulatif ini, dihitung jumlah data yang masuk ke sisi kiri dan kanan untuk setiap kemungkinan threshold. Setelah itu, probabilitas untuk setiap kelas pada sisi kiri dan kanan dihitung, yang digunakan untuk menghitung entropy masing-masing sisi. Perhitungan entropy dilakukan dengan menggunakan rumus yang melibatkan logaritma basis dua untuk mengukur ketidakpastian distribusi kelas pada sisi kiri dan kanan.

Setelah menghitung entropy kiri dan kanan untuk setiap threshold, langkah berikutnya adalah menghitung weighted entropy, yaitu entropy gabungan dari kedua sisi yang dipengaruhi oleh jumlah data yang ada pada masing-masing sisi. Information gain diperoleh dengan mengurangi weighted entropy dari entropy induk (parent entropy), yang menunjukkan pengurangan ketidakpastian setelah data dibagi. Dari hasil perhitungan information gain untuk semua threshold, algoritma memilih threshold dengan nilai information gain tertinggi. Indeks dan nilai information gain tertinggi kemudian dikembalikan sebagai hasil, yang menunjukkan pembagian data yang paling optimal untuk fitur yang sedang dianalisis.

**g. Fungsi `_build_tree(self, X: np.ndarray, y: np.ndarray, depth: int = 0)`**



```

1 def _build_tree(self, X: np.ndarray, y: np.ndarray, depth: int = 0) -> Union[DecisionNode, int]:
2     num_samples, _ = X.shape
3     num_labels = len(np.unique(y))
4
5     if (num_labels == 1 or
6         num_samples < self.min_samples_split or
7         (self.max_depth is not None and depth >= self.max_depth)):
8         return np.bincount(y).argmax()
9
10    best_feature, best_threshold, best_info_gain = self._best_split(X, y)
11
12    if (best_info_gain < self.min_information_gain or
13        best_feature is None):
14        return np.bincount(y).argmax()
15
16    left_mask = X[:, best_feature] < best_threshold
17    right_mask = ~left_mask
18
19    if not np.any(left_mask) or not np.any(right_mask):
20        return np.bincount(y).argmax()
21
22    left_subtree = self._build_tree(X[left_mask], y[left_mask], depth + 1)
23    right_subtree = self._build_tree(X[right_mask], y[right_mask], depth + 1)
24
25    return DecisionNode(
26        feature=best_feature,
27        threshold=best_threshold,
28        left=left_subtree,
29        right=right_subtree
30    )

```

**Gambar 1.3.6.** Cuplikan kode `_calculate_info_gain`

Metode `_build_tree` bertanggung jawab untuk membangun pohon keputusan secara rekursif. Fungsi ini dimulai dengan memeriksa apakah kondisi berhenti untuk pembagian lebih lanjut sudah terpenuhi. Ada beberapa kondisi berhenti yang perlu diperiksa:

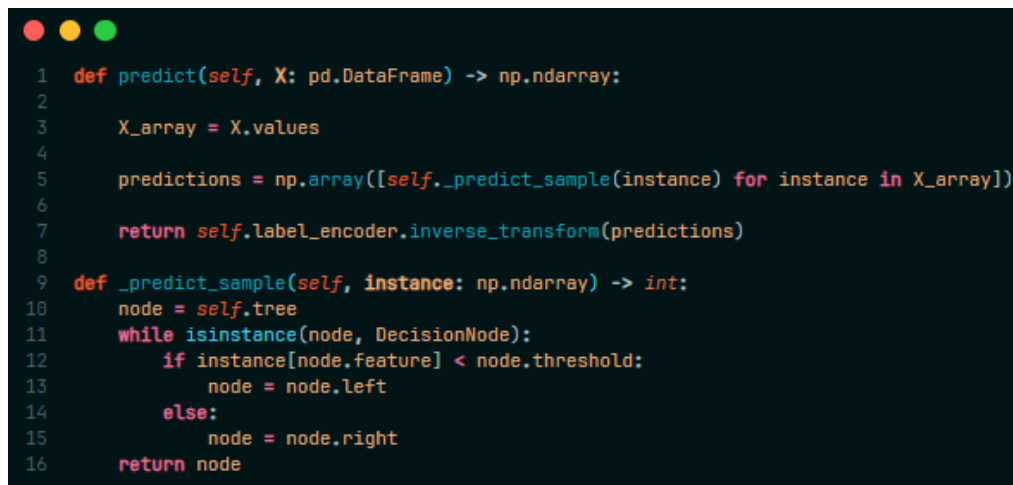
- Jika hanya ada satu label yang tersisa dalam target (y), artinya seluruh data sudah berada dalam satu kelas dan tidak perlu dilakukan pemisahan lebih lanjut. Fungsi ini kemudian mengembalikan nilai kelas tersebut.
- Jika jumlah sampel lebih sedikit dari batas minimal pembagian (`min_samples_split`), artinya tidak cukup banyak data untuk dibagi lebih lanjut, sehingga fungsi ini mengembalikan nilai kelas yang paling sering muncul di data saat itu.
- Jika kedalaman pohon sudah mencapai batas maksimal (`max_depth`), pembagian dihentikan dan kelas terbanyak yang ada pada data tersebut dikembalikan.

Jika tidak ada kondisi berhenti yang tercapai, metode kemudian melanjutkan untuk menemukan fitur terbaik dan threshold terbaik untuk membagi data menggunakan metode `_best_split`. Hasilnya adalah fitur dan threshold yang memberikan information gain tertinggi. Jika pembagian ini tidak memberikan information gain yang cukup

(kurang dari ambang batas yang ditentukan oleh `min_information_gain`), maka proses rekursif berhenti dan nilai kelas terbanyak pada data saat itu dikembalikan.

Setelah pembagian yang optimal ditemukan, data dibagi menjadi dua bagian berdasarkan threshold yang ditentukan. Fungsi kemudian melanjutkan secara rekursif untuk membangun sub-pohon untuk sisi kiri (data yang lebih kecil dari threshold) dan sisi kanan (data yang lebih besar atau sama dengan threshold). Setiap panggilan rekursif ini menghasilkan sebuah sub-pohon yang akan menjadi anak dari simpul keputusan yang sedang dianalisis.

**h. Fungsi `predict(self, X: pd.DataFrame)` dan `_predict_sample(self, instance: np.ndarray)`**



```
1 def predict(self, X: pd.DataFrame) -> np.ndarray:
2
3     X_array = X.values
4
5     predictions = np.array([self._predict_sample(instance) for instance in X_array])
6
7     return self.label_encoder.inverse_transform(predictions)
8
9 def _predict_sample(self, instance: np.ndarray) -> int:
10     node = self.tree
11     while isinstance(node, DecisionNode):
12         if instance[node.feature] < node.threshold:
13             node = node.left
14         else:
15             node = node.right
16     return node
```

**Gambar 1.3.6.** Cuplikan kode `predict` dan `_predict_sample`

Fungsi `predict` bertujuan untuk melakukan prediksi terhadap dataset yang diberikan berdasarkan model pohon keputusan yang sudah dilatih sebelumnya. Langkah pertama yang dilakukan oleh fungsi ini adalah mengonversi input `X`, yang berupa sebuah pandas `DataFrame`, menjadi sebuah `numpy.ndarray`. Hal ini dilakukan untuk mempermudah pengolahan data pada tahap-tahap selanjutnya.

Setelah itu, fungsi ini melakukan prediksi terhadap setiap baris data dalam `X_array`. Untuk setiap baris data (yang merepresentasikan satu sampel), fungsi ini memanggil metode `_predict_sample`, yang akan mengembalikan hasil prediksi untuk sampel tersebut. Hasil prediksi berupa kelas yang diprediksi oleh pohon keputusan. Setelah melakukan prediksi untuk seluruh sampel, fungsi ini mengembalikan hasil

prediksi dalam bentuk array yang sudah didekode kembali menggunakan metode `inverse_transform` dari `LabelEncoder`. Langkah Ini penting agar hasil prediksi yang awalnya dalam bentuk angka dapat diterjemahkan kembali ke dalam label asli (misalnya, kelas dalam kasus klasifikasi).

Metode `_predict_sample` adalah bagian dari proses prediksi untuk setiap sampel tunggal. Fungsi ini bekerja dengan cara menelusuri pohon keputusan yang telah dibangun selama proses pelatihan. Dimulai dengan simpul akar (root node), fungsi ini membandingkan nilai fitur sampel dengan threshold pada simpul saat ini. Jika nilai fitur lebih kecil dari threshold, fungsi ini melanjutkan penelusuran ke anak kiri (`node.left`), dan jika lebih besar atau sama, penelusuran dilanjutkan ke anak kanan (`node.right`). Proses ini diulang hingga mencapai simpul daun (leaf node), yang mengandung hasil prediksi. Nilai pada simpul daun ini kemudian dikembalikan sebagai hasil prediksi untuk sampel tersebut.

Secara keseluruhan, kedua fungsi ini bekerja sama untuk memberikan prediksi bagi sejumlah sampel yang diberikan, dengan menelusuri pohon keputusan yang telah dibangun sebelumnya dan memilih jalur yang sesuai berdasarkan nilai-nilai fitur dalam setiap sampel.

## BAB II : Cleaning dan Preprocessing data

### 2.1. Tahap Cleaning

#### 2.1.1. Handling Missing Data

Pada tahap ini, kami membuat kelas baru *CustomSimpleImputer* menggunakan *SimpleImputer* dari library *sklearn.impute*. Pada fungsi *SimpleImputer* terdapat *strategy* seperti *mean*, *median*, dan *most\_frequent*. Untuk data numerik, kami menggunakan *strategy mean*. Untuk data kategorik, kami menggunakan *strategy most\_frequent*.

#### 2.1.2. Dealing with Outliers

Pada tahap ini, kami tidak melakukan handling outliers. Ketika kami mencoba menghapus outlier, hasil f1 score dari model menjadi lebih buruk tetapi masih ada kemungkinan bahwa model overfit.

#### 2.1.3. Remove Duplicates

Pada tahap ini, kami tidak menghapus duplikat data karena data yang duplikat cukup banyak. Jika data duplikat dihapus, maka data train akan menjadi sedikit.

#### 2.1.4. Feature Engineering

Pada tahap ini, kami mencoba menggabungkan beberapa atribut yang sekiranya dapat digabung dan mempunyai pengaruh terhadap target. Atribut yang dibuat adalah

- $\text{total\_bytes} = \text{sbytes} + \text{dbytes}$
- $\text{byte\_ratio} = \text{sbytes} / (\text{dbytes} + 1)$
- $\text{pkt\_rate} = \text{spkts} / (\text{dur} + 1)$
- $\text{pkt\_interval} = 1 / (\text{pkt\_rate} + 10^{-9})$
- $\text{log\_sbytes} = \log_{10}(\text{sbytes})$
- $\text{log\_dbytes} = \log_{10}(\text{dbytes})$

Atribut nilainya binary ( 1 = true dan 0 = false):

- $\text{high\_traffic} = \text{total\_bytes} > 10^6$

- `short_duration = dur < 1`
- `frequent_src = ct_src_ltm > 50`
- `frequent_dst = cr_dst_ltm > 50`
- `high_pkt_rate = pkt_rate > 100`
- `rare_proto = (proto != tcp / udp / icmp)`
- `low_tcprtt = tcprtt < 0.01`
- `small_response = response_body_len < 100`
- `suspect_synack_ratio = (synack / (ackdat + 1)) > 10`

Pada tahap ini, kami juga membuat fungsi `reduce_label` untuk mengurangi jumlah kategori pada data kategorik. Kami mempertahankan 5 kategori teratas (yang paling sering muncul) dan mengubah sisanya menjadi '-' dengan tujuan agar tidak perlu memproses banyak kategori. Hal ini juga dikarenakan adanya kategori yang munculnya sedikit. Untuk fungsi ini dipakai di data kategorik saja.

## 2.2. Tahap Preprocessing

### 2.2.1. Feature Scaling

Pada tahap ini, kami membuat kelas baru *CustomStandardScaler* dengan menggunakan *StandardScaler* dari library *sklearn.preprocessing*. *Scaler* ini berguna untuk melakukan standarisasi pada fitur (*mean* = 0, standar deviasi = 1). *Scaler* ini digunakan untuk data numerik saja.

### 2.2.2. Feature Encoding

Pada tahap ini, kami membuat kelas baru *CustomOneHotEncoder* dengan menggunakan *OneHotEncoder* dari library *sklearn.preprocessing*. *Encoder* ini berguna untuk mengkonversi data yang kategorik menjadi numerik dalam bentuk angka *binary* (1 = true atau 0 = false). *Encoder* ini digunakan untuk data kategorik saja.

### 2.2.3. Handling Imbalanced Dataset

Pada tahap ini, kami tidak melakukan *handling imbalanced dataset* karena kami ketika dicoba, hasilnya tidak lebih baik.

#### **2.2.4. Data Normalization**

Pada tahap ini, kami membuat fungsi baru yaitu *log transform*. *Transform* ini berguna untuk mentransformasi kolom yang memiliki nilai unik lebih dari 50 agar distribusi data tidak terlalu skew.

#### **2.2.5. Dimensionality Reduction**

Pada tahap ini, kami tidak melakukan *dimensionality Reduction* karena ketika dicoba, hasilnya tidak lebih baik.

## BAB III : Analisis

### 3.1. Perbandingan Hasil Prediksi Algoritma KNN Scratch dengan Pustaka

Pemilihan nilai k pada setiap percobaan adalah 5.

#### 3.1.1. Algoritma KNN dengan Jarak Euclidean

**Tabel 3.1.1.1.** Perbandingan antara Scratch dan Pustaka (Euclidean)

| Scratch   |           |        |          |         | Pustaka   |           |        |          |         |
|---|-----------|--------|----------|---------|---|-----------|--------|----------|---------|
|   | precision | recall | f1-score | support |   | precision | recall | f1-score | support |
| Analysis  | 0.12      | 0.12   | 0.12     | 400     | Analysis  | 0.12      | 0.12   | 0.12     | 400     |
| Backdoor  | 0.04      | 0.01   | 0.02     | 349     | Backdoor  | 0.04      | 0.01   | 0.02     | 349     |
| DoS   | 0.33      | 0.31   | 0.32     | 2453    | DoS   | 0.33      | 0.31   | 0.32     | 2453    |
| Exploits  | 0.64      | 0.71   | 0.67     | 6679    | Exploits  | 0.64      | 0.71   | 0.67     | 6679    |
| Fuzzers   | 0.53      | 0.67   | 0.59     | 3637    | Fuzzers   | 0.53      | 0.67   | 0.59     | 3637    |
| Generic   | 1.00      | 0.94   | 0.97     | 8000    | Generic   | 1.00      | 0.94   | 0.97     | 8000    |
| Normal  | 0.92      | 0.86   | 0.89     | 11200   | Normal  | 0.92      | 0.86   | 0.89     | 11200   |
| Reconnaissance  | 0.61      | 0.60   | 0.60     | 2098    | Reconnaissance  | 0.60      | 0.60   | 0.60     | 2098    |
| Shellcode   | 0.41      | 0.12   | 0.18     | 227     | Shellcode   | 0.41      | 0.12   | 0.18     | 227     |
| Worms   | 0.00      | 0.00   | 0.00     | 26      | Worms   | 0.00      | 0.00   | 0.00     | 26      |
| accuracy  |           |        | 0.76     | 35069   | accuracy  |           |        | 0.76     | 35069   |
| macro avg   | 0.46      | 0.44   | 0.44     | 35069   | macro avg   | 0.46      | 0.44   | 0.44     | 35069   |
| weighted avg  | 0.76      | 0.76   | 0.76     | 35069   | weighted avg  | 0.76      | 0.76   | 0.76     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.4375481111901609% |           |        |          |         | F1 Score (Macro Average) on Validation Set: 0.43744636414486954 |           |        |          |         |

Perbandingan hasil antara algoritma KNN dengan jarak Euclidean yang diimplementasikan sendiri dengan pustaka sklearn yang dapat dilihat pada **Tabel 3.1.1.1.** menunjukkan bahwa akurasi dan F1 Score (Macro Average) yang dihasilkan sama, yaitu 76% untuk akurasi dan 0.43 untuk F1 Score. Hal ini membuktikan bahwa logika perhitungan jarak Euclidean dan pemilihan tetangga terdekat ( $k=5$ ) pada implementasi sudah sesuai dengan pustaka. Insight yang didapat adalah implementasi KNN pribadi dapat bekerja dengan baik dan konsisten.

#### 3.1.2. Algoritma KNN dengan Jarak Manhattan

**Tabel 3.1.2.1.** Perbandingan antara Scratch dan Pustaka (Manhattan)

| Scratch | Pustaka |
|---------|---------|
|---------|---------|

|  | precision | recall | f1-score | support |
|--|-----------|--------|----------|---------|
| Analysis   | 0.15      | 0.14   | 0.14     | 400     |
| Backdoor   | 0.11      | 0.06   | 0.08     | 349     |
| DoS  | 0.33      | 0.32   | 0.33     | 2453    |
| Exploits   | 0.66      | 0.73   | 0.69     | 6679    |
| Fuzzers  | 0.66      | 0.67   | 0.66     | 3637    |
| Generic  | 1.00      | 0.98   | 0.99     | 8000    |
| Normal   | 0.92      | 0.90   | 0.91     | 11200   |
| Reconnaissance   | 0.75      | 0.71   | 0.73     | 2098    |
| Shellcode  | 0.55      | 0.24   | 0.33     | 227     |
| Worms  | 0.33      | 0.04   | 0.07     | 26      |
| accuracy   |           |        | 0.79     | 35069   |
| macro avg  | 0.54      | 0.48   | 0.49     | 35069   |
| weighted avg   | 0.79      | 0.79   | 0.79     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.4928650644023526 |           |        |          |         |

|   | precision | recall | f1-score | support |
|---|-----------|--------|----------|---------|
| Analysis  | 0.15      | 0.14   | 0.14     | 400     |
| Backdoor  | 0.11      | 0.06   | 0.08     | 349     |
| DoS   | 0.33      | 0.33   | 0.33     | 2453    |
| Exploits  | 0.66      | 0.73   | 0.69     | 6679    |
| Fuzzers   | 0.66      | 0.67   | 0.66     | 3637    |
| Generic   | 1.00      | 0.98   | 0.99     | 8000    |
| Normal  | 0.92      | 0.90   | 0.91     | 11200   |
| Reconnaissance  | 0.75      | 0.70   | 0.73     | 2098    |
| Shellcode   | 0.55      | 0.24   | 0.33     | 227     |
| Worms   | 0.33      | 0.04   | 0.07     | 26      |
| accuracy  |           |        | 0.79     | 35069   |
| macro avg   | 0.54      | 0.48   | 0.49     | 35069   |
| weighted avg  | 0.79      | 0.79   | 0.79     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.49309134839770463 |           |        |          |         |

Perbandingan hasil antara algoritma KNN dengan jarak Manhattan yang diimplementasikan sendiri dengan pustaka sklearn yang dapat dilihat pada **Tabel 3.1.2.1.** menunjukkan bahwa akurasi dan F1 Score (Macro Average) yang dihasilkan sama, yaitu 79% untuk akurasi dan 0.49 untuk F1 Score. Hal ini membuktikan bahwa logika perhitungan jarak Manhattan dan pemilihan tetangga terdekat ( $k=5$ ) pada implementasi sudah sesuai dengan pustaka. *Insight* yang didapat adalah implementasi KNN pribadi dapat bekerja dengan baik dan konsisten.

### 3.1.3. Algoritma KNN dengan Jarak Minkowski

**Tabel 3.1.3.1.** Perbandingan antara Scratch dan Pustaka (Minkowski)

| Scratch   |           |        |          |         | Pustaka   |           |        |          |         |
|---|-----------|--------|----------|---------|---|-----------|--------|----------|---------|
|   | precision | recall | f1-score | support |   | precision | recall | f1-score | support |
| Analysis  | 0.12      | 0.12   | 0.12     | 400     | Analysis  | 0.12      | 0.12   | 0.12     | 400     |
| Backdoor  | 0.04      | 0.01   | 0.02     | 349     | Backdoor  | 0.04      | 0.01   | 0.02     | 349     |
| DoS   | 0.33      | 0.31   | 0.32     | 2453    | DoS   | 0.33      | 0.31   | 0.32     | 2453    |
| Exploits  | 0.64      | 0.71   | 0.67     | 6679    | Exploits  | 0.64      | 0.71   | 0.67     | 6679    |
| Fuzzers   | 0.53      | 0.67   | 0.59     | 3637    | Fuzzers   | 0.53      | 0.67   | 0.59     | 3637    |
| Generic   | 1.00      | 0.94   | 0.97     | 8000    | Generic   | 1.00      | 0.94   | 0.97     | 8000    |
| Normal  | 0.92      | 0.86   | 0.89     | 11200   | Normal  | 0.92      | 0.86   | 0.89     | 11200   |
| Reconnaissance  | 0.61      | 0.60   | 0.60     | 2098    | Reconnaissance  | 0.60      | 0.60   | 0.60     | 2098    |
| Shellcode   | 0.41      | 0.12   | 0.18     | 227     | Shellcode   | 0.41      | 0.12   | 0.18     | 227     |
| Worms   | 0.00      | 0.00   | 0.00     | 26      | Worms   | 0.00      | 0.00   | 0.00     | 26      |
| accuracy  |           |        | 0.76     | 35069   | accuracy  |           |        | 0.76     | 35069   |
| macro avg   | 0.46      | 0.44   | 0.44     | 35069   | macro avg   | 0.46      | 0.44   | 0.44     | 35069   |
| weighted avg  | 0.76      | 0.76   | 0.76     | 35069   | weighted avg  | 0.76      | 0.76   | 0.76     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.43754811119016096 |           |        |          |         | F1 Score (Macro Average) on Validation Set: 0.43744636414486954 |           |        |          |         |

Hasil perbandingan antara algoritma KNN dengan jarak Minkowski yang diimplementasikan sendiri dengan pustaka sklearn yang dapat dilihat pada **Tabel 3.1.3.1.** menunjukkan bahwa akurasi dan F1 Score (Macro Average) yang sama, yaitu 0.74 untuk akurasi dan 0.43 untuk F1 Score. Nilai *precision*, *recall*, dan



F1-score untuk setiap kelas juga hampir identik. *Insight* yang diperoleh adalah implementasi KNN dengan jarak Minkowski yang dibuat sendiri sudah berjalan dengan baik dan konsisten dengan pustaka sklearn. Hal ini membuktikan bahwa proses perhitungan jarak, pemilihan tetangga terdekat, dan prediksi label mayoritas sudah benar.

### 3.1.4. Algoritma KNN dengan Jarak Hamming

**Tabel 3.1.4.1.** Perbandingan antara Scratch dan Pustaka (Hamming)

| Scratch  |           |        |          |         | Pustaka  |           |        |          |         |
|--|-----------|--------|----------|---------|--|-----------|--------|----------|---------|
|  | precision | recall | f1-score | support |  | precision | recall | f1-score | support |
| Analysis   | 0.24      | 0.14   | 0.18     | 400     | Analysis   | 0.26      | 0.15   | 0.19     | 400     |
| Backdoor   | 0.26      | 0.10   | 0.14     | 349     | Backdoor   | 0.25      | 0.10   | 0.14     | 349     |
| DoS  | 0.34      | 0.40   | 0.37     | 2453    | DoS  | 0.34      | 0.38   | 0.36     | 2453    |
| Exploits   | 0.67      | 0.73   | 0.70     | 6679    | Exploits   | 0.66      | 0.73   | 0.70     | 6679    |
| Fuzzers  | 0.73      | 0.69   | 0.71     | 3637    | Fuzzers  | 0.72      | 0.69   | 0.71     | 3637    |
| Generic  | 1.00      | 0.98   | 0.99     | 8000    | Generic  | 1.00      | 0.98   | 0.99     | 8000    |
| Normal   | 0.93      | 0.91   | 0.92     | 11200   | Normal   | 0.93      | 0.91   | 0.92     | 11200   |
| Reconnaissance   | 0.75      | 0.75   | 0.75     | 2098    | Reconnaissance   | 0.75      | 0.75   | 0.75     | 2098    |
| Shellcode  | 0.55      | 0.34   | 0.42     | 227     | Shellcode  | 0.55      | 0.33   | 0.42     | 227     |
| Worms  | 0.62      | 0.38   | 0.48     | 26      | Worms  | 0.77      | 0.38   | 0.51     | 26      |
| accuracy   |           |        | 0.80     | 35069   | accuracy   |           |        | 0.80     | 35069   |
| macro avg  | 0.61      | 0.54   | 0.57     | 35069   | macro avg  | 0.62      | 0.54   | 0.57     | 35069   |
| weighted avg   | 0.80      | 0.80   | 0.80     | 35069   | weighted avg   | 0.80      | 0.80   | 0.80     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.5651865288516201 |           |        |          |         | F1 Score (Macro Average) on Validation Set: 0.5686289560356386 |           |        |          |         |

Hasil perbandingan antara algoritma KNN dengan jarak Hamming yang diimplementasikan sendiri dan pustaka yang dapat dilihat pada **Tabel 3.1.4.1.** menunjukkan bahwa keduanya memiliki hasil yang hampir serupa. Akurasi yang dicapai sama, yaitu 0.80, namun terdapat sedikit perbedaan pada F1 Score (Macro Average), di mana model pustaka memiliki nilai 0.56, sedangkan model dari *scratch* menghasilkan 0.56 juga, namun lebih rendah di desimal terakhir. *Insight* yang diperoleh adalah implementasi KNN dengan jarak Hamming sudah berjalan cukup baik dan mendekati performa pustaka sklearn. Perbedaan kecil ini bisa disebabkan oleh implementasi teknis dalam perhitungan jarak.

## 3.2. Perbandingan Hasil Prediksi Algoritma Naive Bayes Scratch dengan Pustaka

**Tabel 3.2.1.** Perbandingan antara Naive Bayes Scratch dengan Pustaka

| Scratch   |           |        |          |         | Pustaka   |           |        |          |         |
|---|-----------|--------|----------|---------|---|-----------|--------|----------|---------|
|   | precision | recall | f1-score | support |   | precision | recall | f1-score | support |
| Analysis  | 0.11      | 0.06   | 0.07     | 400     | Analysis  | 0.07      | 0.06   | 0.07     | 400     |
| Backdoor  | 0.05      | 0.77   | 0.09     | 349     | Backdoor  | 0.05      | 0.76   | 0.09     | 349     |
| DoS   | 0.06      | 0.00   | 0.00     | 2453    | DoS   | 0.07      | 0.00   | 0.00     | 2453    |
| Exploits  | 0.61      | 0.17   | 0.27     | 6679    | Exploits  | 0.59      | 0.17   | 0.27     | 6679    |
| Fuzzers   | 0.09      | 0.04   | 0.06     | 3637    | Fuzzers   | 0.09      | 0.04   | 0.06     | 3637    |
| Generic   | 0.99      | 0.78   | 0.87     | 8000    | Generic   | 0.99      | 0.78   | 0.87     | 8000    |
| Normal  | 0.97      | 0.30   | 0.45     | 11200   | Normal  | 0.97      | 0.27   | 0.43     | 11200   |
| Reconnaissance  | 0.00      | 0.00   | 0.00     | 2098    | Reconnaissance  | 0.00      | 0.00   | 0.00     | 2098    |
| Shellcode   | 0.03      | 0.96   | 0.06     | 227     | Shellcode   | 0.03      | 0.96   | 0.06     | 227     |
| Worms   | 0.00      | 0.92   | 0.01     | 26      | Worms   | 0.00      | 0.92   | 0.01     | 26      |
| accuracy  |           |        | 0.33     | 35069   | accuracy  |           |        | 0.32     | 35069   |
| macro avg   | 0.29      | 0.40   | 0.19     | 35069   | macro avg   | 0.29      | 0.40   | 0.18     | 35069   |
| weighted avg  | 0.67      | 0.33   | 0.40     | 35069   | weighted avg  | 0.66      | 0.32   | 0.39     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.18762605638524457 |           |        |          |         | F1 Score (Macro Average) on Validation Set: 0.18366716233407016 |           |        |          |         |

Hasil perbandingan antara Naive Bayes yang diimplementasikan sendiri dengan pustaka yang dapat dilihat pada **Tabel 3.2.1.** menunjukkan bahwa akurasi dan F1 Score (Macro Average) kedua model hampir sama, dengan akurasi 0.33 untuk model *scratch* dan 0.32 untuk model pustaka, dan nilai F1 Score Macro Average juga sangat hampir sama, yaitu 0.187 untuk model *scratch* dan 0.183 untuk model *scratch*. Insight yang diperoleh adalah performa kedua model menunjukkan konsistensi, terutama dalam mendeteksi kelas mayoritas seperti "Generic" dan "Normal". Perbedaan kecil yang terjadi kemungkinan disebabkan oleh perbedaan implementasi teknis, seperti penanganan probabilitas kecil dan smoothing pada model pustaka GaussianNB. Secara keseluruhan, implementasi Naive Bayes sudah berjalan dengan baik dan memberikan hasil yang sejalan dengan pustaka.

### 3.3. Perbandingan Hasil Prediksi Algoritma ID3 Scratch dengan Pustaka

**Tabel 3.3.1.1.** Perbandingan antara ID3 Scratch dengan Pustaka

| Scratch | Pustaka |
|---------|---------|
|---------|---------|

|   | precision | recall | f1-score | support |
|---|-----------|--------|----------|---------|
| Analysis  | 0.12      | 0.21   | 0.16     | 400     |
| Backdoor  | 0.08      | 0.21   | 0.12     | 349     |
| DoS   | 0.32      | 0.33   | 0.32     | 2453    |
| Exploits  | 0.68      | 0.66   | 0.67     | 6679    |
| Fuzzers   | 0.66      | 0.70   | 0.68     | 3637    |
| Generic   | 0.99      | 0.98   | 0.99     | 8000    |
| Normal  | 0.93      | 0.89   | 0.91     | 11200   |
| Reconnaissance  | 0.88      | 0.73   | 0.80     | 2098    |
| Shellcode   | 0.50      | 0.44   | 0.47     | 227     |
| Worms   | 0.18      | 0.08   | 0.11     | 26      |
| accuracy  |           |        | 0.78     | 35069   |
| macro avg   | 0.54      | 0.52   | 0.52     | 35069   |
| weighted avg  | 0.80      | 0.78   | 0.79     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.521677063936014 |           |        |          |         |

|  | precision | recall | f1-score | support |
|--|-----------|--------|----------|---------|
| Analysis   | 0.10      | 0.20   | 0.13     | 400     |
| Backdoor   | 0.08      | 0.21   | 0.12     | 349     |
| DoS  | 0.31      | 0.28   | 0.30     | 2453    |
| Exploits   | 0.70      | 0.64   | 0.67     | 6679    |
| Fuzzers  | 0.64      | 0.65   | 0.64     | 3637    |
| Generic  | 0.99      | 0.98   | 0.98     | 8000    |
| Normal   | 0.91      | 0.90   | 0.90     | 11200   |
| Reconnaissance   | 0.78      | 0.74   | 0.76     | 2098    |
| Shellcode  | 0.49      | 0.54   | 0.52     | 227     |
| Worms  | 0.38      | 0.38   | 0.38     | 26      |
| accuracy   |           |        | 0.77     | 35069   |
| macro avg  | 0.54      | 0.55   | 0.54     | 35069   |
| weighted avg   | 0.79      | 0.77   | 0.78     | 35069   |
| F1 Score (Macro Average) on Validation Set: 0.5405790566086546 |           |        |          |         |

Hasil perbandingan antara ID3 yang diimplementasikan sendiri dan pustaka yang dapat dilihat pada **Tabel 3.3.1.** menunjukkan performa yang hampir sama. Akurasi model *scratch* mencapai 0.78, sedangkan model pustaka mencapai 0.77. Nilai F1 Score (Macro Average) menunjukkan sedikit perbedaan, yaitu 0.54 untuk model *scratch* dan 0.521 untuk pustaka.

*Insight* yang diperoleh adalah implementasi ID3 yang dibuat sudah bekerja dengan baik dan mendekati performa pustaka *sklearn*. Perbedaan kecil terjadi pada distribusi metrik *precision*, *recall*, dan *F1-score* di beberapa kelas, yang kemungkinan disebabkan oleh perbedaan detail teknis dalam proses pemilihan *threshold* dan *handling data* pada *split*. Perbedaan ini juga dapat dipengaruhi oleh parameter default yang digunakan, seperti nilai *min\_samples\_split* pada model *scratch* yang diatur ke 10, sementara pada pustaka *sklearn* menggunakan nilai default 2. Hal ini membuat model *scratch* cenderung membatasi proses *split* lebih ketat dibandingkan pustaka. Selain itu, model *sklearn* juga memiliki parameter *random\_state*, yang memastikan hasil *split* lebih konsisten, sedangkan pada implementasi manual ID3, hasil *split* bisa bervariasi. Perbedaan lain mungkin berasal dari optimasi algoritma di pustaka *sklearn*, terutama dalam proses pemilihan *threshold* terbaik atau penanganan perhitungan *information gain*. Secara keseluruhan, kedua model memiliki performa yang konsisten, terutama pada kelas mayoritas.

## Referensi

- Bandung Institute of Technology. “Decision Tree Learning (DTL).” *Edunex*, [https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/298936-Modeling-Decision-Tree-Learning-DTL/120485-Modul-Introduction-to-DTL/1731401412073\\_IF3170\\_SupervisedLearning\\_DTL.pdf](https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/298936-Modeling-Decision-Tree-Learning-DTL/120485-Modul-Introduction-to-DTL/1731401412073_IF3170_SupervisedLearning_DTL.pdf). Accessed 15 12 2024.
- Bandung Institute of Technology. “K-Nearest Neighbor.” *Edunex*, [https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250331293\\_IF3170\\_Materi09\\_Seg01\\_AI-kNN.pdf](https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250331293_IF3170_Materi09_Seg01_AI-kNN.pdf). Accessed 15 12 2024.
- Bandung Institute of Technology. “Naive Bayes.” *Edunex*, [https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250380758\\_IF3170\\_Materi09\\_Seg02\\_AI-NaiveBayes.pdf](https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250380758_IF3170_Materi09_Seg02_AI-NaiveBayes.pdf). Accessed 15 12 2024.
- Bandung Institute of Technology. “Prediction Measurement.” *Edunex*, [https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250430397\\_IF3170\\_Materi09\\_Seg03\\_AI-PredictionMeasurement.pdf](https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250430397_IF3170_Materi09_Seg03_AI-PredictionMeasurement.pdf). Accessed 15 12 2024.

# Lampiran

Link Github : <https://github.com/AlbertChoe/Tubes2-AI.git>

**Tabel Pembagian Kerja**

| NIM      | Nama                    | Kontribusi  |
|----------|-------------------------|---|
| 13522045 | Elbert Chailes          | <ul style="list-style-type: none"><li>- Pipeline</li><li>- Data preprocessing</li><li>- Model</li><li>- Documentation</li></ul> |
| 13522073 | Juan Alfred Widjaya     | <ul style="list-style-type: none"><li>- Pipeline</li><li>- Data preprocessing</li><li>- Model</li><li>- Documentation</li></ul> |
| 13522081 | Albert                  | <ul style="list-style-type: none"><li>- Pipeline</li><li>- Data preprocessing</li><li>- Model</li><li>- Documentation</li></ul> |
| 13522113 | William Glory Henderson | <ul style="list-style-type: none"><li>- Pipeline</li><li>- Data preprocessing</li><li>- Model</li><li>- Documentation</li></ul> |