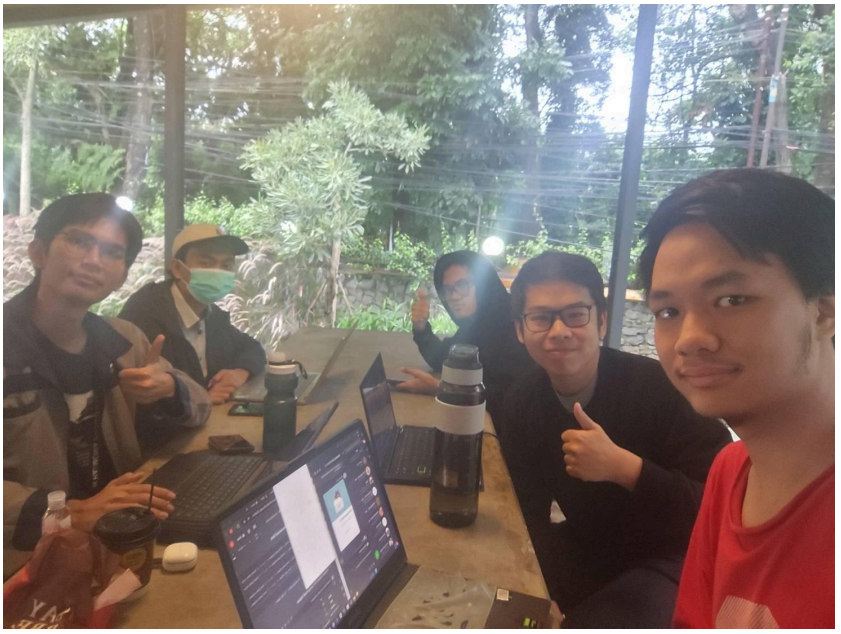
Implementasi Algoritma Pembelajaran Mesin

Laporan Tugas Besar 2

IF3170 Intelegensi Artifisial



Disusun Oleh Kelompok 36:

10321009	Sahabista Arkitanego A.
10821019	Dean Hartono
12821046	Fardhan Indrayesa
13522051	Kharris Khisunica
18321011	Wikan Priambudi

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

BAB 1
JUDUL

- Cover
- Penjelasan singkat implementasi KNN.
- Penjelasan singkat implementasi Naive-Bayes.
- Penjelasan singkat implementasi ID3.
- Penjelasan tahap cleaning dan preprocessing yang dilakukan beserta dengan alasannya.
- Perbandingan hasil prediksi dari algoritma yang diimplementasikan dengan hasil yang didapatkan dengan menggunakan pustaka. Jelaskan insight yang kalian dapatkan dari perbandingan tersebut.
- Perbandingan hasil dapat menggunakan metrics yang sesuai dengan permasalahan yang ada.
- Kontribusi setiap anggota dalam kelompok.
- Referensi

hhhhh

BAB 1
Pendahuluan

1.1. K-Nearest Neighbor (KNN)

K-Nearest Neighbors (KNN) merupakan algoritma *supervised machine learning* yang biasanya digunakan untuk menyelesaikan permasalahan klasifikasi dan regresi. KNN tidak membangun model eksplisit untuk melakukan prediksi, tetapi menggunakan data yang ada secara langsung untuk membuat prediksi berdasarkan kedekatan dengan titik data yang lain. Pembentukan model KNN dilakukan dengan menentukan jumlah minimal tetangga dengan jarak terdekat tiap *data point* pada dataset. Penentuan jarak ini pada umumnya menggunakan metrik jarak sebagai berikut, yaitu Manhattan, Euclidean, dan Minkowski.

1. Jarak Manhattan

$$d(p,\ q) = \sum_{i=1}^{n} \left| p_i - q_i \right|$$

2. Jarak Euclidean

$$d(p,\ q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

3. Jarak Minkowski

$$d(p,\ q) = \left(\sum_{i=1}^{n} \left| p_i - q_i \right|^r \right)^{1/r}$$

1.2. Naive Bayes

Model klasifikasi Naive Bayes, merupakan model probabilitas yang dibangun dengan mengacu pada teorema Naive-Bayes. Model ini mengasumsikan bahwa seluruh fitur independen dan tidak bergantung sama lain. Pemodelan dan implementasi model Naive-Bayes dilakukan dengan terlebih dahulu menghitung *prior* dan *likelihood* masing-masing kelas pada dataset. Kemudian, model ini akan melakukan prediksi dengan menggunakan teorema Bayes, sebagai berikut.

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

Pada model Naive-Bayes ini, digunakan model Gaussian Naive-Bayes. Gaussian Naive-Bayes sendiri dapat didefinisikan sebagai model *machine learning* yang menggunakan distribusi probabilitas dan fungsi normal / fungsi Gaussian. Fungsi Gaussian sendiri, dapat dituliskan sebagai berikut.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

1.3. Iterative Dichotomiser 3 (ID3)

Iterative Dichotomiser 3 (ID3) merupakan algoritma dalam membuat pohon keputusan (*decision tree*). Model ID3 dibangun dengan menentukan *root node* dan *leaf node*, yang mana *root node* merupakan simpul pertama dan paling atas dalam pohon keputusan dan *leaf node* merupakan simpul pohon keputusan yang tidak memiliki cabang lebih lanjut. Untuk menentukan *root node* dan *leaf node*, pada setiap iterasi model ini akan menentukan nilai entropi untuk setiap atribut yang belum digunakan, nilai entropi terbesar akan menjadi *node*. Entropi menggambarkan ketidakteraturan dataset. Penentuan nilai entropi ini mengikuti persamaan berikut.

$$H(S) = -\sum_{x \in X} p(x) \cdot log_2 p(x)$$

Setelah menentukan nilai entropi, kemudian ditentukan nilai *information gain* untuk datasets terhadap suatu atribut pada dataset, yang mengukur seberapa banyak berkurangnya nilai ketidakteraturan setelah dilakukan *split* dengan atribut tertentu.

$$IG(S, A) = H(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \cdot H(S_v)$$

Untuk menghindari bias dan *overfitting* akibat adanya atribut dengan banyak nilai, dilakukan pula mekanisme *gain ratio* dengan membagi nilai *information gain* terhadap informasi intrinsik pada dataset. Persamaan untuk *gain ratio* adalah sebagai berikut.

$$GR(S, A) = \frac{IG(S, A)}{SI(S, A)} = \frac{H(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \cdot H(S_v)}{-\sum_{v \in A} \frac{|S_v|}{|S|} \cdot log_2\left(\frac{|S_v|}{|S|}\right)}$$

Umumnya, nilai *gain ratio* ini yang akan menentukan *feature importance* dari dataset dalam proses membangun pohon keputusan. Atribut yang dipilih adalah atribut yang bisa menghasilkan nilai *gain ratio* yang paling besar.

BAB 2

Pengolahan Data

2.1. Exploratory Data Analysis (EDA)

2.1.1. Data Size

Tahapan ini bertujuan untuk mengetahui ukuran data yang akan digunakan pada model. Pada data train didapati ukuran (175341, 43) dan data test (20583, 42).

2.1.2. Data Types

Tahapan ini bertujuan mengetahui tipe data pada tiap kolom dari data yang akan digunakan. Rupanya terdapat 4 atribut yang bertipe ‘object’, termasuk atribut target, sehingga keempat atribut ini pasti bersifat kategorikal. 39 atribut lainnya adalah data yang bertipe ‘float64’ dan ‘int64’, artinya atribut-atribut ini berisikan angka. Meskipun berisikan angka, belum tentu atribut-atribut ini memang bersifat numerik, ada kemungkinan atribut-atribut ini bersifat kategorikal, tetapi direpresentasikan dengan angka.

2.1.3. Seperate Categorical and Numerical Data

Mengacu pada metadata, didukung dengan eksplorasi tipe data, terdapat 7 atribut yang bersifat kategorikal dan sisanya bersifat numerik. Kolom atribut yang bersifat kategorikal dikelompokkan, begitu pula yang bersifat numerik, karena akan dilakukan beberapa analisis setelahnya yang berbeda untuk kedua golongan ini.

2.1.4. Categorical Values Unique Count

Tahapan ini bertujuan untuk mengetahui ada berapa banyak jenis input yang unik (berbeda) pada data dengan tipe kategorikal.

2.1.5. Statistical Attributes of Numerical Values

Tahapan ini bertujuan untuk mengetahui statistik dari masing-masing atribut numerik pada data. Mengetahui statistik dari masing-masing atribut numerik dapat berguna untuk mengambil keputusan metode yang tepat untuk data cleaning dan data preprocessing.

2.1.6. Missing Values

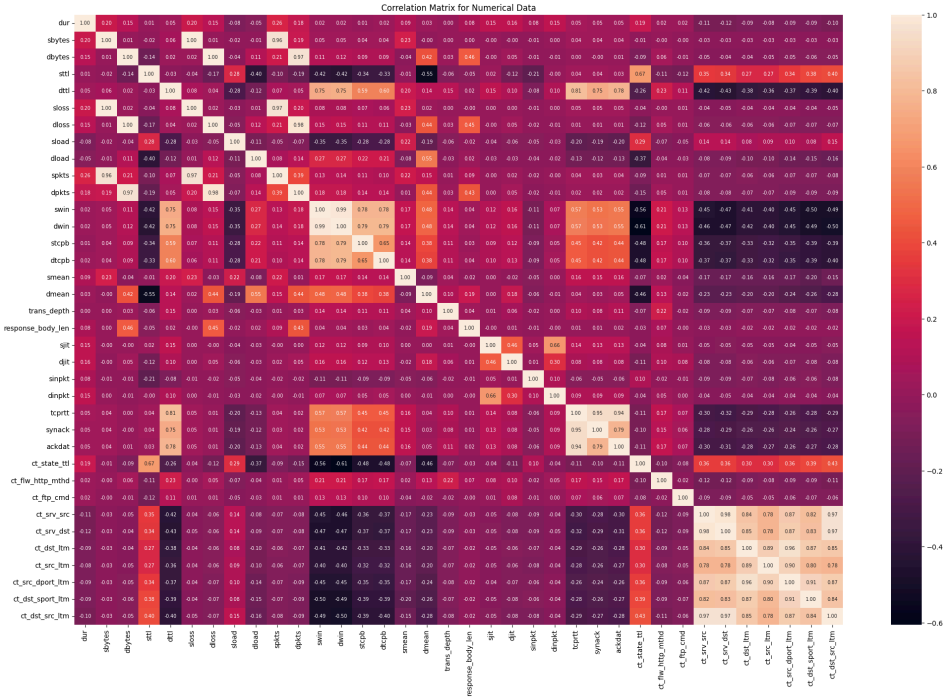
Tahapan ini bertujuan untuk menghitung total nilai yang hilang pada kolom numerikal dan kategorikal dari data.

2.1.7. Duplicate Data

Tahapan ini bertujuan untuk mengetahui jumlah data yang memiliki duplikat. Tahapan ini penting disebabkan jika terdapat suatu data yang memiliki duplikat, model akan cenderung overfit.

2.1.8. Numerical Correlation Matrix

Untuk atribut-atribut numerik, dapat dibangun matriks korelasi antar-atributnya, berikut ini matriks korelasinya



Matriks korelasi ini membantu untuk melihat adanya atribut-atribut yang memiliki karakteristik serupa dan untuk tujuan klasifikasi, dalam kondisi tertentu, misalnya untuk mengurangi kompleksitas algoritma, dapat digunakan salah satu atribut saja

2.2. Data Cleaning

2.2.1. Handling Missing Data

Dalam dataset, ditemukan terdapat kurang lebih 8500-8900 data yang hilang di setiap kolomnya. Metode yang digunakan untuk menangani missing data pada kolom kategorikal adalah menggunakan modus. Untuk data numerik, imputasi nilai yang dipakai bergantung pada skewness kolom tersebut tanpa data hilang. Untuk data yang skewness nya lebih dari 0.5, maka data hilang pada kolom tersebut diimputasi dengan median karena kolom tersebut memiliki skewness yang tinggi. Untuk data yang skewness nya

kurang dari sama dengan 0.5, maka data hilang pada kolom tersebut diimputasi dengan mean karena kolom tersebut memiliki skewness yang rendah.

2.2.2. Dealing with Outliers

Pada tahapan ini digunakan tiga metode yaitu imputasi, transformasi, dan *clipping*. Program pertama akan mendeteksi *outlier* menggunakan metode IQR, setelah itu dilakukan perhitungan statistik pada tiap kolom agar diketahui metode yang tepat untuk masing masing kolom. Semua kolom yang terdapat nilai *outlier* akan melalui tahapan imputasi terlebih dahulu, nilai yang dideteksi sebagai *outlier* akan digantikan dengan nilai median, penggantian dengan nilai median dinilai lebih *robust* terhadap nilai outlier pada kolom tersebut. Setelah melalui tahapan imputasi kemudian kolom-kolom dengan nilai *skewness* lebih dari 0.5 maka akan melewati tahapan transformasi, tahapan ini berguna untuk mengurangi *skewness* dari data tersebut. Kemudian untuk kolom yang memiliki nilai di luar persentil 0.01 dan 0.99 akan dilakukan *clipping*, dengan adanya *clipping* diharapkan nilai outlier tidak akan memengaruhi model.

2.2.3. Removing Duplicates

Dalam dataset, ditemukan terdapat 5756 kolom yang memiliki nilai yang sama di setiap kolom nya. Karena ukuran ini relatif kecil terhadap jumlah kolom pada data, maka strategi yang dilakukan untuk menghilangkan duplikat adalah dengan membuang kolom duplikat dan hanya mempertahankan satu salinan dari kolom tersebut.

2.3. Data Preprocessing

2.3.1. Feature Scaling

Feature scaling bertujuan untuk menstandarisasi nilai tiap-tiap atribut pada dataset, sehingga memberikan peran yang sama dalam proses pelatihan model *machine learning* dan model tersebut dapat bekerja secara optimal untuk dataset. Pada tahap ini, digunakan algoritma MinMaxScaler yang mana

melakukan standarisasi dengan menggunakan nilai minimum dan maksimum. *Output* dari algoritma ini berada pada *range* 0 hingga 1.

2.3.2. Feature Encoding

Saat dataset dari data kategorikal ditelusuri, ditemukan bahwa tidak bisa ditemukan adanya urutan dari data tersebut, yakni data kategorikal yang ada bersifat nominal. Oleh sebab itu, metode encoding yang digunakan adalah One-Hot encoding yang dapat melakukan encoding data kategorikal yang tidak memiliki urutan.

2.3.3. Feature Engineering

Pada feature engineering bertujuan untuk mengetahui fitur apa saja yang dinilai cukup penting untuk membangun sebuah model, hal ini disebabkan tidak semua fitur atau variabel dapat membantu model memprediksi data.

Pertama digunakan random forest dan selectfrommodel untuk menghitung *feature importance*. Jika ada fitur yang masuk pada kedua hasil perhitungan tersebut maka akan dimasukkan.

Kemudian dengan menggunakan koefisien Lasso (L1 regularization) akan dilakukan penyaringan fitur lagi. Jika fitur dari metode pertama terpilih tetapi memiliki koefisien Lasso negatif, maka fitur tersebut akan dihapus. Setelah melewati tahapan ini maka didapatkan fitur-fitur yang akan digunakan untuk melatih model.

2.3.4. Handling Imbalanced Classes

Pada tahapan EDA, diperoleh bahwa terdapat ketidakseimbangan class pada datasets, sehingga perlu dilakukan adanya penyeimbangan pada data tersebut. Untuk menangani ketidakseimbangan ini, dilakukan proses *resampling* dengan menggunakan metode *oversampling*, *undersampling*, ataupun keduanya. Untuk metode *oversampling* sendiri, digunakan algoritma Synthetic Minority Oversampling Techniques (SMOTE), sementara untuk metode *undersampling* menggunakan algoritma NearMiss.

2.3.5. Dimensionality Reduction

Proses pengurangan dimensi data atau biasa disebut *dimensionality reduction* merupakan proses yang dilakukan untuk mengurangi dimensi data input yang dibutuhkan, dengan tetap mempertahankan informasi yang dapat diperoleh. Proses ini menjadi penting, dikarenakan pada dataset terdapat cukup banyak *feature*. Proses ini juga akan mempermudah pada saat melakukan analisis. Untuk melakukan pengurangan data dapat dilakukan dengan beberapa metode, salah satunya adalah Principal Component Analysis (PCA). PCA akan mentransformasi fitur-fitur yang berkorelasi menjadi tidak berkorelasic

2.3.6. Data Normalization

Pada Data Normalization, data yang ada akan dinormalisasi agar dataset yang ada memiliki distribusi yang normal. Hal ini dilakukan untuk memastikan proses yang berjalan dengan asumsi data berdistribusi normal dapat berjalan dengan baik. Untuk menormalkan data, dilakukan transformasi dengan PowerTransformer dari library sklearn.

BAB 3
Pemodelan dan Evaluasi Model

3.1. Pembentukan Model

3.1.1. K-Nearest Neighbors (KNN)

Berikut adalah implementasi dari algoritma K-Nearest Neighbours (KNN).

Class	
class KNN	Class sebagai implementasi dari K-Nearest Neighbours.
Attribute	
k (int)	Banyak tetangga terdekat yang akan dijadikan kandidat kelas.
metric (str)	Metrik yang akan digunakan untuk menghitung jarak antar data. Terdapat metrik Manhattan, Euclidean, dan Minkowski.
p (int)	Derajat pangkat yang akan digunakan dalam perhitungan jarak menggunakan metrik Minkowski.
Fungsi dan Prosedur	
fit (x, y)	Fungsi ini bertujuan untuk menyimpan data latih, data target, kolom target, dan kategori seluruh kelas.
predict (x)	Fungsi ini bertujuan untuk menghasilkan prediksi berdasarkan data inputan terhadap dataset yang sudah di-fitting.

__manhattan_distance (x)	Fungsi private untuk mendapatkan jarak antar data menggunakan metrik Manhattan (p = 1).
__euclidean_distance (x)	Fungsi private untuk mendapatkan jarak antar data menggunakan metrik Euclidean (p = 2).
__minkowski_distance (x)	Fungsi private untuk mendapatkan jarak antar data menggunakan metrik Minkowski (p ϵ natural numbers).

Berikut merupakan potongan source code untuk class KNN.

```
class KNN:
    def __init__(self, k, metric="euclidean", p=None):
        self.k: int = k
        self.metric: str = metric
        if (self.metric == "manhattan"):
            self.p: int = 1
        elif (self.metric == "euclidean"):
            self.p: int = 2
        elif (self.metric == "minkowski"):
            if p == None:
                raise Exception("For minkowski metric, p is
required")
            elif p <= 0:
                raise Exception("For minkowski metric, p must be
more than 0")
            self.p: int = p

    def fit(self, x, y) -> 'knn':
        """Save data train, label, and classes"""
        self.x_ = pd.DataFrame(x)
        self.y_ = pd.DataFrame(y)

        self.labels_ = self.y_.columns
        self.classes_: np.ndarray = np.array([self.y_[lab].unique()
for lab in self.labels_], dtype=object)

        return self

    def predict(self, x) -> np.ndarray:
        """Predict for a dataset"""
```

```
        x = pd.DataFrame(x)

        classes_final = []
        distances_all = []

        for instance in x.values:
            if (self.metric == "manhattan"):
                d = self.__manhattan_distance(instance)

            elif (self.metric == "euclidean"):
                d = self.__euclidean_distance(instance)

            elif (self.metric == "minkowski"):
                if self.p == 1:
                    d = self.__manhattan_distance(instance)
                elif self.p == 2:
                    d = self.__euclidean_distance(instance)
                elif self.p > 2:
                    d = self.__minkowski_distance(instance)

            d_df = pd.DataFrame({"distance": d})

            for label in self.labels_:
                d_df["label"] = self.y_[label].values
                d_nearest = d_df.sort_values("distance")     # Sort
nearest distance
                clas = d_nearest.iloc[:self.k]               # Take
k neighbours
                distances_all.append(d_df["distance"].values)

                class_freq =
clas.groupby(["label"]).count().rename(columns={"distance":"count"})
# Class frequency
                check_if_tie = False
                if len(clas) % 2 == 0: # Even
                    if len(class_freq) > 1:
                        check_if_tie = all(i ==
class_freq.iloc[0].values for i in class_freq.values.squeeze())

                    if check_if_tie: # Tie check
                        weight = clas.groupby(["label"]).sum()
# Sum distance every class
```

```
                        final_class =
weight.iloc[np.argmin(weight)].name # Take class with nearest
distance
                    else:
                        final_class =
class_freq.iloc[np.argmax(class_freq)].name

                else: # Odd
                    final_class =
class_freq.iloc[np.argmax(class_freq)].name

                classes_final.append(final_class)

        if len(self.labels_) > 1:
            classes_final = np.array(classes_final).reshape((len(x),
len(self.labels_))).astype("O")
        else:
            classes_final = np.array(classes_final).astype("O")
        self.distances = np.array(distances_all)

        return classes_final

    def __manhattan_distance(self, x) -> np.ndarray:
        """Calculate distance with Manhattan metric"""
        return np.array(abs(self.x_ - x).sum(axis=1))

    def __euclidean_distance(self, x) -> np.ndarray:
        """Calculate distance with Euclidean metric"""
        return np.array(np.sqrt(((self.x_ - x)**2).sum(axis=1)))

    def __minkowski_distance(self, x) -> np.ndarray:
        """Calculate distance with Minkowski metric"""
        return np.array((abs((knear.x_.values -
x)**self.p).sum(axis=1))**(1/self.p))
```

3.1.2. Naive-Bayes

Berikut adalah implementasi dari algoritma Naive-Bayes.

Class

class NaiveBayes	Class sebagai implementasi dari Naive-Bayes.
Attribute	
-	-
Fungsi dan Prosedur	
fit (x, y)	Fungsi ini bertujuan untuk menyimpan data latih, data target, kolom target, kategori seluruh kelas, menghitung probabilitas prior masing-masing kelas, rata-rata setiap kolom, dan standar deviasi setiap kolom.
predict (x)	Fungsi ini bertujuan untuk menghasilkan prediksi berdasarkan data inputan terhadap dataset yang sudah di-fitting.
__mean (x)	Fungsi private untuk mendapatkan rata-rata setiap kolom pada dataset.
__standarddev (x)	Fungsi private untuk mendapatkan standar deviasi setiap kolom pada dataset.
__gaussian (x, miu, sigma)	Fungsi private untuk mendapatkan nilai probabilitas gaussian setiap kolom.
__classification (proba)	Fungsi private ini bertujuan untuk mendapatkan hasil klasifikasi berdasarkan nilai probabilitas maksimum.

Berikut merupakan potongan source code untuk class Naive-Bayes.

```
class NaiveBayes:
    def fit(self, x, y):
        """Train model from train dataset"""
        self.x_ = pd.DataFrame(x)
        self.y_ = pd.DataFrame(y)
        self.n_ = len(x)
```

```
        self.features_ = self.x_.columns
        self.labels_ = self.y_.columns
        self.classes_: np.ndarray = np.array([self.y_[lab].unique()
for lab in self.labels_], dtype=object)

        self.prior_y = []
        self.x_mean = []
        self.x_std = []

        data_train = pd.concat([self.x_.reset_index(drop=True),
self.y_.reset_index(drop=True)], axis=1)
        for lab in self.labels_:
            prior_temp = []
            mean_temp = []
            std_temp = []

            data_train_groupped =
data_train.groupby(self.y_[lab].values.squeeze())
            for i, data in data_train_groupped:
                prob_prior = len(data)/self.n_
                mean_x = self.__mean(data)
                std_x = self.__standarddev(data)

                prior_temp.append(prob_prior)
                mean_temp.append(mean_x)
                std_temp.append(std_x)

            self.prior_y.append(prior_temp)
            self.x_mean.append(mean_temp)
            self.x_std.append(std_temp)

        self.prior_y = np.array(self.prior_y)
        self.x_mean = np.array(self.x_mean)
        self.x_std = np.array(self.x_std)

        return self

    def predict(self, x):
        """Predict for a dataset"""
        x = pd.DataFrame(x)
        self.prob_all = []

        # Calculate the probability of every instance
```

```
        for i, lab in enumerate(self.labels_):
            prob_lab = []
            for j, uniq in enumerate(self.classes_[i]):
                gauss = 1
                for k, col in enumerate(self.features_):
                    gauss = gauss * self.__gaussian(x[col],
self.x_mean[i, j, k], self.x_std[i, j, k])
                prob_lab.append(gauss * self.prior_y[i, j])
            self.prob_all.append(prob_lab)

        self.prob_all = np.array(self.prob_all)

        return self.__classification(self.prob_all)

    def __mean(self, x):
        """Calculate the mean of every column"""
        return self.x_.iloc[x.index].mean(axis=0)

    def __standarddev(self, x):
        """Calculate the standard deviation of every column"""
        return self.x_.iloc[x.index].std(axis=0)

    def __gaussian(self, x, miu, sigma):
        """Calculate the gaussian probability of every instance"""
        return 1/(np.sqrt(2*np.pi*(sigma**2))) * np.exp(-1/2 *
((x-miu)/sigma)**2)

    def __classification(self, proba):
        """Take the result by maximum probability of every
instance"""
        final_result = []
        for i, lab in enumerate(self.labels_):
            idx_prob_max = np.argmax(proba[i, :, :], axis=0)
            final = self.classes_[i, :][idx_prob_max]
            final_result.append(final)

        return np.array(final_result).squeeze()
```

3.1.3. Iterative Dichotomiser 3 (ID3)

Berikut merupakan implementasi dari algoritma ID3 dalam membuat pohon keputusan.

Class	
class ID3	Class sebagai implementasi dari Iterative Dichotomiser 3 (ID3).
Attribute	
Fungsi dan Prosedur	
total_entropy(S)	Fungsi untuk menghitung total entropi dari label S
subset_entropy(S, A)	Fungsi untuk menghitung entropi dari subset A dalam label S
information_gain(S, A)	Fungsi untuk menghitung information gain dari subset A dalam label S
split_information(S, A)	Fungsi untuk menghitung information split dari subset A
gain_ratio(S, A)	Fungsi untuk menghitung gain ratio dan menormalisasi nilai dari information gain
best_split(S, A)	Fungsi untuk menentukan atribut dengan gain ratio terbaik dan atribut apa yang menghasilkan gain ratio tersebut

build_tree(S, A, depth)	Fungsi untuk membuat tree ID3 secara rekursif dengan kedalaman maksimal = depth
fit(S, A)	Fungsi untuk menyimpan data latih, subset dari data latih, dan pohon yang dibuat
predict_single(x, tree)	Fungsi untuk memprediksi satu instance
predict_row(row, tree)	Fungsi untuk memprediksi satu row
predict(x)	Fungsi untuk memprediksi satu dataset
visualize_tree(tree, features, parent_name, graph)	Fungsi untuk memvisualisasikan tree yang terbentuk

Berikut merupakan potongan *source code* untuk class ID3.

```
class ID3:
    def __init__(self, S=None, A=None, max_depth=None):
        self.S = S
        self.A = A
        self.max_depth = max_depth
        self.tree = {}
        self.classes = np.unique(S) if S is not None else []

    def total_entropy(self, S):
        label_counts = Counter(S)
        data_counts = len(S)
        total_entropy = 0
        for label in label_counts.values():
            probability = label / data_counts
            total_entropy -= probability *
np.log2(probability)
        return total_entropy

    def subset_entropy(self, S, A):
```

```
        data_counts = len(S)
        subset_entropy = 0

        # If no data, return entropy as 0
        if data_counts == 0:
            return 0

        # If A is empty, return 0
        if A.empty:
            return 0

        # Get the first column of A (assuming it's a
DataFrame)
        attribute_column = A.iloc[:, 0]

        subsets = pd.DataFrame({'label': S, 'attribute':
attribute_column}).groupby('attribute')

        for _, subset in subsets:
            subset_counts = len(subset)
            if subset_counts == 0:
                continue

            subset_label_counts = Counter(subset['label'])
            entropy = 0
            for count in subset_label_counts.values():
                subset_probability = count / subset_counts
                entropy -= subset_probability *
np.log2(subset_probability)

            subset_entropy += (subset_counts / data_counts) *
entropy

        return subset_entropy

    def information_gain(self, S, A):
        return self.total_entropy(S) - self.subset_entropy(S,
A)

    def split_information(self, S, A):
```

```
        data_counts = len(S)
        split_info = 0

        if data_counts == 0:
            return 0

        # Instead of looking for 'attribute', use the first
column of A:
        first_column = A.columns[0] # Assuming A is a
DataFrame
        # Check if A is empty or has only one unique value
        if A.empty or len(A[first_column].unique()) <= 1:
            return 0  # Return 0 if A is empty or has only
one unique value

        # Use the first column in groupby:
        subsets = A.groupby(first_column)

        for _, subset in subsets:
            subset_counts = len(subset)
            if subset_counts == 0:
                continue

            split_info_prob = subset_counts / data_counts
            split_info -= split_info_prob *
np.log2(split_info_prob)

        return split_info

    def gain_ratio(self, S, A):
        split_info = self.split_information(S, A)
        if split_info == 0:
            return 0
        return self.information_gain(S, A) / split_info

    def best_split(self, S, A):
        """Find the best attribute to split on"""
        best_gain_ratio = -np.inf
        best_attribute = None
```

```
        for atr in A.columns:
            if A[atr].dtype in ['int64', 'float64']:  # 
Pastikan tipe data numerik
                values = np.sort(A[atr].unique())  # Urutkan 
nilai unik
                possible_splits = (values[:-1] + values[1:]) 
/ 2  # Ambil midpoints

                for split in possible_splits:
                    left_mask = A[atr] <= split
                    right_mask = A[atr] > split

                    S_left, A_left = S[left_mask], 
A[left_mask]
                    S_right, A_right = S[right_mask], 
A[right_mask]

                    gain_left = self.gain_ratio(S_left, 
A_left)
                    gain_right = self.gain_ratio(S_right, 
A_right)
                    gain_ratio = gain_left + gain_right

                    if gain_ratio > best_gain_ratio:
                        best_gain_ratio = gain_ratio
                        best_attribute = atr

        return best_attribute, best_gain_ratio

    def build_tree(self, S, A, depth=0):
        # Jika hanya ada satu kelas di S, kembalikan kelas 
tersebut
        if len(np.unique(S)) == 1:
            return np.unique(S)[0]

        # Jika tidak ada fitur yang tersisa atau kedalaman 
maksimum tercapai, return kelas mayoritas
        if len(A.columns) == 0 or (self.max_depth is not None 
and depth >= self.max_depth):
            return Counter(S).most_common(1)[0][0]
```

```
        # Temukan atribut terbaik untuk pemisahan
        best_attribute, _ = self.best_split(S, A)
        tree = {best_attribute: {}}

        # Membagi dataset berdasarkan nilai atribut terbaik
        for value in np.unique(A[best_attribute]):
            left_mask = A[best_attribute] == value
            subset_S = S[left_mask]
            subset_A = A[left_mask]

            if len(subset_S) == 0:
                tree[best_attribute][value] = None  # Jika
kosong, beri nilai default (None)
            else:
                tree[best_attribute][value] =
self.build_tree(subset_S,
subset_A.drop(columns=[best_attribute]), depth + 1)

        return tree

    def fit(self, S, A):
        self.S = S
        self.A = A
        self.tree = self.build_tree(self.S, self.A)
        return self.tree

    def predict_single(self, x, tree):
        """Predict a single instance"""
        if not isinstance(tree, dict):
            return tree
        attribute = next(iter(tree))
        value = x[attribute]
        print(f"Predicting with attribute: {attribute},
value: {value}")  # Debugging
        subtree = tree[attribute].get(value, None)
        if subtree is None:
            print("No matching value found in the tree.")  #
Debugging
            return None
```

```
        return self.predict_single(x, subtree)


    def predict(self, X):
        return X.apply(lambda x: self.predict_single(x,
self.tree), axis=1)

    def visualize_tree(self, tree, features, parent_name='',
graph=None):
        if graph is None:
            graph = Digraph(format='png')
            graph.attr('node', shape='box')

        for node, subtree in tree.items():
            node_label = str(node)
            graph.node(node_label)
            if parent_name:
                graph.edge(parent_name, node_label)

            if isinstance(subtree, dict):
                self.visualize_tree(subtree, features,
parent_name=node_label, graph=graph)
            else:
                leaf_label = f"Class: {subtree}"
                graph.node(leaf_label, style='filled',
fillcolor='lightblue')
                graph.edge(node_label, leaf_label)

        return graph
```

3.2. Perbandingan dengan Referensi

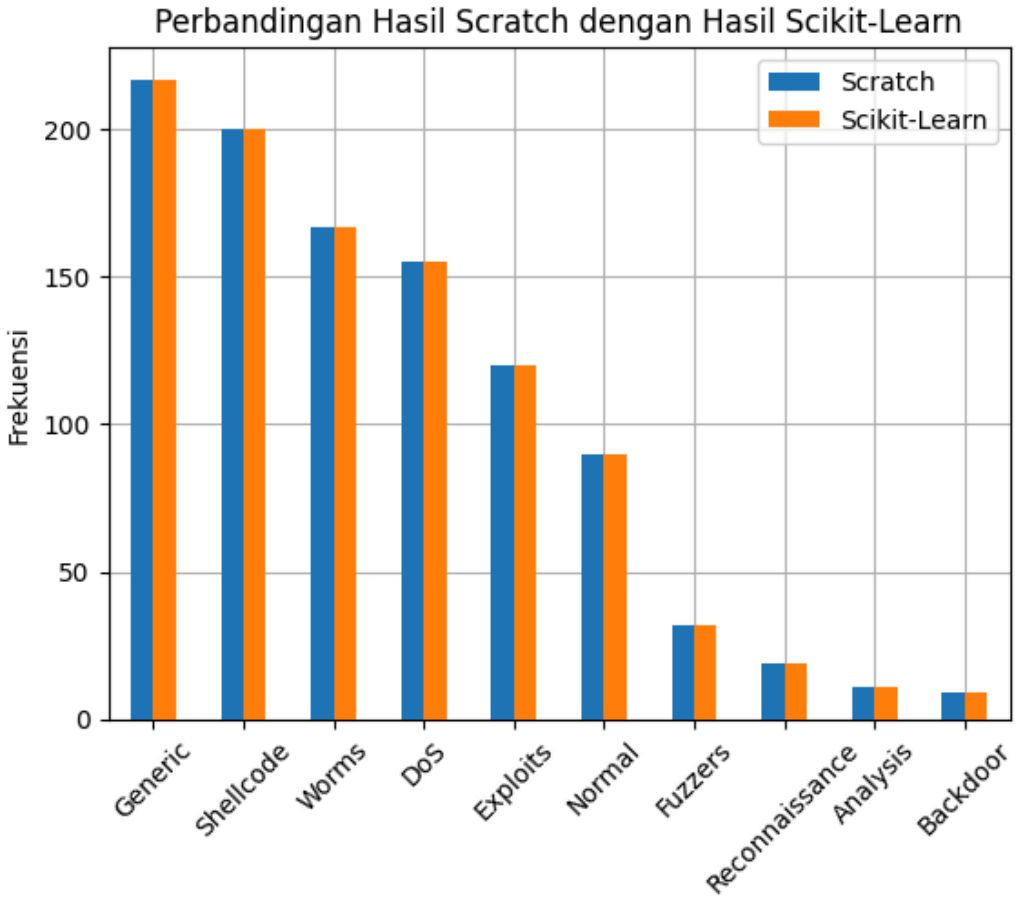
3.2.1. K-Nearest Neighbors (KNN)

Berdasarkan hasil model, diperoleh akurasi model KNN scratch (dengan hyperparameter k = 7, metric minkowski, dan p = 3) dan scikit-learn sebagai berikut.

Model	Akurasi

Akurasi model scratch terhadap data target	0.225
Akurasi model Scikit-Learn terhadap data target	0.225
Akurasi model scratch terhadap model Scikit-Learn	1

Akurasi yang dihasilkan pada model scratch dan model scikit-learn terhadap data target menandakan bahwa model hanya dapat memprediksi dengan akurasi sekitar 22.5% (dari nilai perfect 100%). Hasil ini berarti bahwa model masih kurang baik dalam memprediksi jenis attack pada dataset. Hal ini dapat terjadi karena terdapat beberapa data yang salah diklasifikasikan oleh model. Selain itu, terlihat juga bahwa hasil prediksi model KNN scratch terhadap scikit-learn menghasilkan jenis attack yang sama dengan nilai akurasi 1 (100%), yang menandakan bahwa kedua model memiliki performa yang sama dalam memprediksi jenis attack.



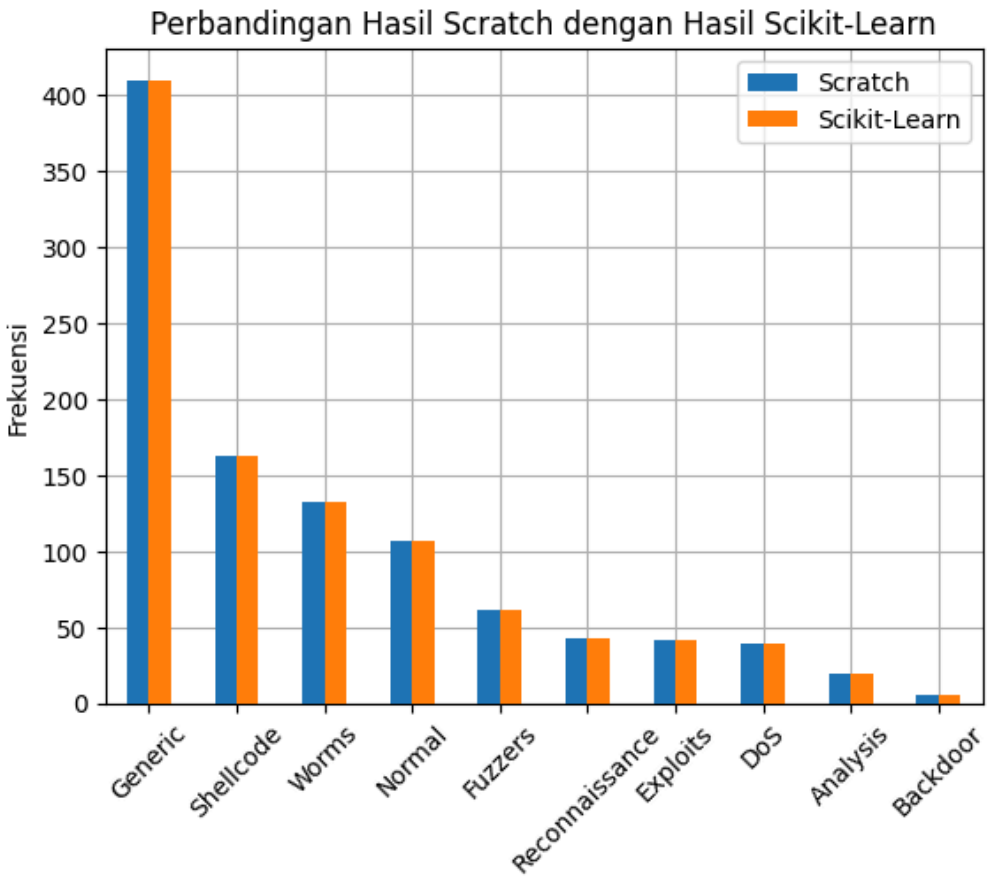
Berdasarkan gambar di atas, jenis attack yang diprediksi oleh model scratch dengan model scikit-learn menghasilkan prediksi yang sama persis pada setiap jenis attack. Selain itu, jenis attack yang paling banyak diprediksi oleh kedua model adalah jenis Generic dan yang paling sedikit adalah jenis Reconnaissance, Analysis, dan Backdoor.

3.2.2. Naive-Bayes

Berdasarkan hasil model, diperoleh akurasi model Naive-Bayes scratch dan scikit-learn sebagai berikut.

Model	Akurasi
Akurasi model scratch terhadap data target	0.2127
Akurasi model Scikit-Learn terhadap data target	0.2127
Akurasi model scratch terhadap model Scikit-Learn	1

Akurasi yang dihasilkan pada model scratch dan model scikit-learn terhadap data target menandakan bahwa model hanya dapat memprediksi dengan akurasi sekitar 21.27% (dari nilai perfect 100%). Hasil ini berarti bahwa model masih kurang baik dalam memprediksi jenis attack pada dataset. Hal ini dapat terjadi karena terdapat beberapa data yang salah diklasifikasikan oleh model. Selain itu, terlihat juga bahwa hasil prediksi model Naive Bayes scratch terhadap scikit-learn menghasilkan jenis attack yang sama dengan nilai akurasi 1 (100%), yang menandakan bahwa kedua model memiliki performa yang sama dalam memprediksi jenis attack.



Berdasarkan gambar di atas, jenis attack yang diprediksi oleh model scratch dengan model scikit-learn menghasilkan prediksi yang sama persis pada setiap jenis attack. Selain itu, jenis attack yang paling banyak diprediksi oleh kedua model adalah jenis Generic dan yang paling sedikit adalah jenis Backdoor. Jenis attack DoS dan Exploits memiliki jumlah yang hampir sama dalam hasil prediksi model.

3.2.3. Iterative Dichotomiser 3 (ID3)

Berdasarkan hasil model, diperoleh akurasi model ID3 *scratch* dan scikit-learn sebagai berikut.

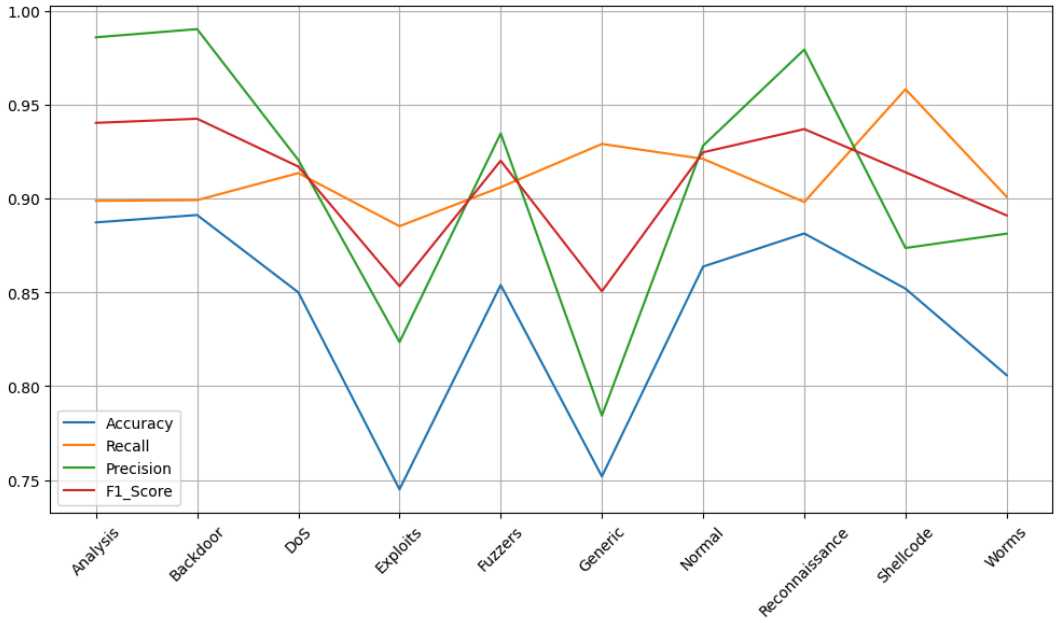
Model	Akurasi
Akurasi model scratch terhadap data target	-
Akurasi model Scikit-Learn terhadap data target	0.1137

Akurasi model scratch terhadap model Scikit-Learn	-

Algoritma ID3 gagal diimplementasikan, sehingga tidak dapat dinilai akurasi untuk model scratch. Untuk referensi sendiri, digunakan model scikit-learn dengan nilai akurasi 11.37%.

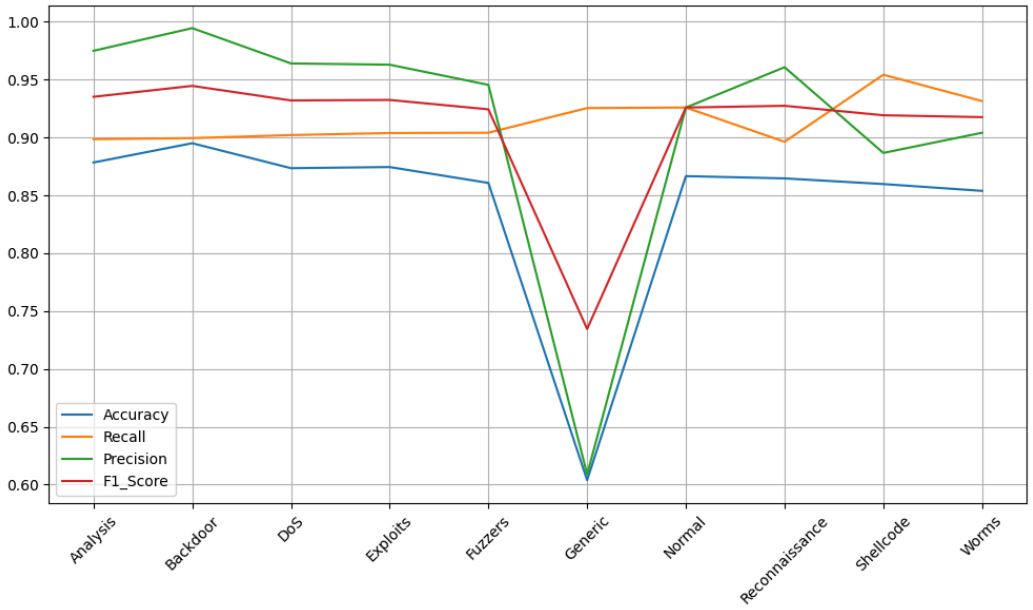
3.3. Evaluasi

3.3.1. K-Nearest Neighbors (KNN)



Gambar di atas adalah hasil dari metrik confusion matrix, yaitu akurasi, recall, precision, dan F1 score antara model scratch dengan data validasi. Berdasarkan gambar tersebut, terlihat bahwa model memiliki akurasi, recall, precision, dan F1 score yang berfluktuasi di setiap jenis attack. Jenis attack Exploits, Generic, dan Worms memiliki nilai metrik akurasi, recall, dan F1 score yang lebih kecil dan menurun secara signifikan daripada jenis attack yang lain. Nilai recall cenderung lebih stabil di seluruh jenis attack daripada metrik-metrik yang lain. Selain itu, model KNN memiliki nilai metrik di atas 0.75 yang berarti model cukup baik dalam memprediksi jenis attack berdasarkan true positive dan negative-nya.

3.3.2. Naive-Bayes



Gambar di atas adalah hasil dari metrik confusion matrix, yaitu akurasi, recall, precision, dan F1 score antara model scratch dengan data validasi. Berdasarkan gambar tersebut, terlihat bahwa model memiliki akurasi, recall, precision, dan F1 score yang cukup tinggi dengan nilai di atas 0.85 dalam memprediksi hampir semua jenis attack, kecuali Generic yang memiliki akurasi mendekati 0.6. Jenis attack generic memiliki nilai akurasi, precision, dan F1 score paling rendah di antara model lainnya. Hal ini menandakan bahwa model memiliki performa yang cukup baik dalam memprediksi seluruh jenis attack, kecuali Generic. Nilai recall cenderung memiliki nilai yang stabil daripada metrik lain. Nilai akurasi, precision, dan F1 Score juga cenderung stabil, tetapi menurun secara signifikan pada jenis attack Generic.

3.3.3. Iterative Dichotomiser 3 (ID3)

Model ID3 gagal untuk diimplementasikan, sehingga tidak dapat dilakukan evaluasi pada model terkait.

BAB 4
Kesimpulan dan Saran

4.1. Kesimpulan

Dari hasil eksperimen di atas, dapat disimpulkan bahwa model KNN adalah model dengan akurasi yang paling baik, dengan tingkat akurasi 0.2225. Selain itu, didapatkan juga bahwa model yang dibuat dari awal dan model yang disediakan dalam library memiliki perbandingan yang tidak terlalu jauh. Dimana tingkat akurasi dari model K-NN dari awal adalah 0.2225 dan tingkat akurasi dari model KNN library adalah 0.2225. Tingkat akurasi dari model Naive Bayes dari awal adalah 0.2127 dan tingkat akurasi dari model Naive Bayes library adalah 0.2127 dengan selisih akurasi adalah 0%. Tingkat akurasi dari model ID3 tidak dapat dinilai, sementara tingkat akurasi dari model ID3 library adalah 0.1137.

4.2. Saran

Saran yang bisa dilakukan adalah untuk menyamaratakan implementasi algoritma yang dilakukan. Tak hanya itu, perlu pula dilakukan pengimplementasian dan percobaan terhadap data sebenarnya, sehingga dapat dikompilasi tanpa adanya kendala dan program telah dipastikan bekerja secara baik.

PEMBAGIAN TUGAS

NIM	Nama	Pembagian Tugas
10321009	Sahabista Arkitanego A.	- ID3, Data Balancer, Penjelasan singkat implementasi model (KNN, Naive-Bayes, ID3), Laporan pembentukan model ID3, Penjelasan proses Data Preprocessing, Pipeline.
10821019	Dean Hartono	- Penjelasan EDA, Set Up Program, Debugging
12821046	Fardhan Indrayesa	- Feature Scaling, Feature Encoding, Dimensionality Reduction, KNN, Naive-Bayes, Confusion Matrix KNN, Confusion Matrix Naive-Bayes, laporan pembentukan model (KNN, Naive-Bayes), Perbandingan dengan referensi (KNN, Naive-Bayes), dan Evaluasi (KNN, Naive-Bayes)
13522051	Kharris Khisunica	- EDA, Pipeline, Preprocessing
18321011	Wikan Priambudi	- Penjelasan EDA, Dealing with Outlier, Feature Engineering, ID3

REFERENSI

https://www.researchgate.net/publication/367868051_Perbandingan_Akurasi_Recall_dan_Presisi_Klasifikasi_pada_Algoritma_C45_Random_Forest_SVM_dan_Naive_Bayes

https://towardsdatascience.com/gaussian-naive-bayes-explained-a-visual-guide-with-code-examples-for-beginners-04949cef383c (Main Mechanism)

https://medium.com/@chellehdwjy/gaussian-naive-bayes-f05ec0b61d91