

Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin



Oleh

Kelompok 34

Tamara Mayranda Lubis (18222026)

Kayla Dyara (18222074)

Monica Angela Hartono (18222078)

Yovanka Sandrina Maharaja (18222094)

Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132
2024

Daftar Isi

Daftar Isi	2
Daftar Tabel	2
Daftar Gambar	4
BAB I	
Landasan Teori	5
A. Konsep K-Nearest Neighbors (KNN)	5
B. Konsep Gaussian Naive-Bayes	7
BAB II	8
Metodologi	8
A. Tahapan Cleaning dan Preprocessing	8
1. Handling Missing Data	8
2. Dealing with Outliers	9
3. Removing Duplicates	12
4. Feature Engineering	12
1. Feature Scaling	14
2. Encoding Categorical Variables	15
3. Handling Imbalanced Classes	16
4. Dimensionality Reduction	17
5. Normalization	18
B. Implementasi Algoritma	19
1. K-Nearest Neighbors (KNN)	19
a. Tanpa Pustaka (from scratch)	19
b. Dengan Pustaka (using library)	22
2. Gaussian Naive-Bayes	24
a. Tanpa Pustaka (from scratch)	24
b. Dengan Pustaka (using library)	27
BAB III	

Hasil dan Analisis	30
A. Perbandingan Model Sklearn VS Model From Scratch	30
Pembagian Tugas	33
Referensi	34

Daftar Tabel

Tabel 2.1 Algoritma Handling Missing Data	9
Tabel 2.2 Algoritma Dealing with Outliers	11
Tabel 2.3 Algoritma Removing Duplicates	13
Tabel 2.4 Algoritma Feature Engineering	13
Tabel 2.5 Algoritma Feature Scaling	15
Tabel 2.6 Algoritma Encoding Categorical Variables	16
Tabel 2.7 Algoritma Handling Imbalanced Classes	17
Tabel 2.8 Algoritma Dimensionality Reduction	19
Tabel 2.9 Algoritma Normalization	19
Tabel 2.10 Algoritma K-Nearest Neighbors (KNN) From Scratch	20
Tabel 2.11 Algoritma K-Nearest Neighbors (KNN) Using Library	23
Tabel 2.12 Algoritma Gaussian Naive-Bayes From Scratch	25
Tabel 2.13 Algoritma Gaussian Naive-Bayes Using Library	28
Tabel 3.1 Perbandingan Model Sklearn vs Model From Scratch	31

Daftar Gambar

Make sure to select headings in the sidebar to see a table of contents.

BAB I

Landasan Teori

A. Konsep K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) merupakan algoritma *machine learning* berbasis jarak yang digunakan untuk tugas klasifikasi, regresi, dan tugas tetangga terdekat. Algoritma ini mencari data di sekitar titik tertentu berdasarkan jarak yang dihitung menggunakan metrik tertentu seperti Euclidean, Manhattan, dan lainnya. Terdapat jenis-jenis Nearest Neighbors seperti

Pada Tugas Besar 2 - IF3070 Dasar Intelegensi Artifisial, jenis Algoritma Nearest Neighbors yang digunakan merupakan K-Nearest Neighbors (KNN). K-Nearest Neighbors (KNN) tergolong ke dalam metode *lazy learning* atau *instance-based learning* karena tidak membangun model internal dari data, tetapi menyimpan data pelatihan untuk digunakan pada saat prediksi. KNN bekerja dengan cara mengukur jarak antara data baru dengan semua data dalam dataset pelatihan menggunakan metrik jarak tertentu. KNN juga menentukan tetangga terdekat dengan memilih data pelatihan terdekat berdasarkan jarak tersebut. Parameter yang digunakan untuk menentukan jumlah tetangga yang digunakan adalah k . Setelah itu, kelas data baru diprediksi berdasarkan mayoritas kelas dari tetangganya atau nilai data baru diprediksi berdasarkan rata-rata atau bobot nilai tetangganya.

Metrik jarak yang digunakan untuk menghitung jarak antara dua titik sebagai berikut :

1. Euclidean Distance :

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

Dengan :

- x_i : Koordinat dari dua titik data x dan y pada dimensi ke- i
- n : Jumlah dimensi dari data

Fungsi ini digunakan untuk menghitung jarak lurus (linear) antara dua titik dalam ruang Euclidean (ruang berdimensi). Euclidean Distance cocok untuk data kontinu dan banyak digunakan pada data berdimensi rendah.

2. Manhattan Distance :

$$d = \sum_{i=1}^n |x_i - y_i|$$

Dengan :

- x_i : Koordinat dari dua titik data x dan y pada dimensi ke- i
- n : Jumlah dimensi dari data

Fungsi ini digunakan untuk menghitung jarak berbasis grid, yaitu jumlah total jarak absolut antara koordinat dua titik. Manhattan Distance cocok untuk kasus dimana arah penting.

3. Minkowski Distance :

$$d = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Dengan :

- x_i : Koordinat dari dua titik data x dan y pada dimensi ke- i
- n : Jumlah dimensi dari data
- p : Parameter kekuatan (power) yang menentukan jenis jarak:
 - Jika $p = 1$, Minkowski Distance sama dengan Manhattan Distance.
 - Jika $p = 2$, Minkowski Distance sama dengan Euclidean Distance.
 - Jika $p > 2$, Minkowski Distance lebih peka terhadap perbedaan besar antar dimensi.

Fungsi ini fleksibel untuk berbagai jenis data dengan parameter p yang dapat disesuaikan.

K-Nearest Neighbors (KNN) memiliki kelebihan seperti sederhana dan intuitif karena tidak memerlukan pelatihan model. Algoritma ini fleksibel, tidak mengasumsikan distribusi apa pun, karena tidak mengharuskan data mengikuti distribusi tertentu sehingga KNN dapat digunakan pada data dengan distribusi tidak normal. Namun, terdapat kekurangan pada K-Nearest Neighbors, yaitu kecepatannya yang lambat karena algoritma

ini menghitung jarak terhadap semua titik data pelatihan saat prediksi sehingga algoritma ini kurang efisien untuk dataset besar. KNN sangat bergantung pada pemilihan parameter k . Nilai k yang terlalu kecil dapat berdampak pada model yang menjadi terlalu sensitif terhadap noise, yaitu variasi yang tidak relevan, sedangkan nilai k yang terlalu besar dapat menyebabkan underfitting, yaitu model tidak dapat menangkap pola dalam data.

B. Konsep Gaussian Naive-Bayes

Naive Bayes merupakan algoritma *machine learning* berbasis probabilistik yang didasarkan pada Teorema Bayes. Istilah “*naive*” berasal dari asumsi bahwa semua fitur dianggap saling independen satu sama lain.

Pada Tugas Besar 2 - IF3070 Dasar Intelegensi Artifisial, jenis Algoritma Naive Bayes yang digunakan merupakan Gaussian Naive Bayes. Gaussian Naive Bayes merupakan salah satu jenis dari Naive Bayes yang dirancang untuk *continuous features* dan mengasumsikan bahwa nilai kontinu dari setiap fitur mengikuti Gaussian (normal) *distribution*.

Distribusi Gaussian didefinisikan dengan fungsi probabilitas sebagai berikut.

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

Dengan:

- x : Nilai fitur.
- μ_y : Rata-rata (*mean*) nilai fitur untuk kelas y .
- σ_y^2 : Varians nilai fitur untuk kelas y .

Gaussian Naive Bayes efektif digunakan untuk data kontinu dan cocok untuk data yang mendekati distribusi normal. Namun, terdapat kekurangan pada Gaussian Naive Bayes, yaitu adanya asumsi independensi antar fitur yang seringkali kurang realistis pada dunia nyata, serta asumsi normalitas distribusi fitur yang akan menjadi masalah jika data yang ada tidak mengikuti distribusi normal.

.BAB II

Metodologi

A. Tahapan *Cleaning* dan *Preprocessing*

Data cleaning merupakan proses awal yang sangat penting dalam mempersiapkan dataset untuk digunakan dalam analisis *machine learning*. Data mentah yang diperoleh dari berbagai sumber sering kali tidak langsung dapat digunakan karena berpotensi memiliki banyak masalah, seperti nilai yang hilang, kesalahan input, hingga ketidakkonsistenan format. Oleh karena itu, proses *data cleaning* melibatkan beberapa langkah utama untuk memastikan kualitas data yang optimal sebelum masuk ke tahap pemodelan.

1. *Handling Missing Data*

Langkah pertama adalah menangani data yang hilang (*handling missing data*). Pada tahap ini, data yang kosong diidentifikasi dan diatasi dengan cara mengganti nilai yang hilang menggunakan metode imputasi. Untuk kolom numerik, digunakan metode seperti rata-rata (*mean*) dan median tergantung pada pola distribusi data. Rata-rata biasanya diterapkan pada data dengan distribusi normal, sedangkan median lebih cocok untuk data dengan *outlier*, karena mampu mengurangi dampak negatif dari nilai-nilai ekstrim. Sebelum dan sesudah proses imputasi, dilakukan visualisasi seperti histogram dan *boxplot* untuk memastikan bahwa metode yang digunakan tidak mengubah pola asli data. Sementara itu, untuk kolom kategori, nilai yang paling sering muncul (*mode*) dipilih untuk menggantikan data yang hilang, karena dianggap paling mewakili kelompok data tersebut.

Tabel 2.1 Algoritma *Handling Missing Data*

```
# Separate numerical and categorical columns
numerical_columns = df_new.select_dtypes(include=['float64',
'int64']).columns
categorical_columns = df_new.select_dtypes(include=['object']).columns

# Compare mean and median imputation
```

```

# Impute using mean and median for demonstration
mean_imputer = SimpleImputer(strategy='mean')
median_imputer = SimpleImputer(strategy='median')

mean_imputed = df_new[sample_columns].copy()
median_imputed = df_new[sample_columns].copy()

mean_imputed = mean_imputer.fit_transform(mean_imputed)
median_imputed = median_imputer.fit_transform(median_imputed)

# Impute Numerical Features
# For normally distributed numerical data: Use median imputation
median_imputer = SimpleImputer(strategy='median')
df_new[numerical_columns] =
median_imputer.fit_transform(df_new[numerical_columns])

# Impute Categorical Features
# For categorical data: Use most frequent value (mode)
mode_imputer = SimpleImputer(strategy='most_frequent')
df_new[categorical_columns] =
mode_imputer.fit_transform(df_new[categorical_columns])

```

2. Dealing with Outliers

Langkah berikutnya adalah *Dealing with Outliers* atau mengatasi *outlier*. *Outlier* adalah data yang nilainya sangat jauh berbeda dari mayoritas data lainnya, yang sering kali dapat mempengaruhi kinerja model secara negatif. *Outlier* dalam dataset ini ditangani menggunakan dua metode utama, yaitu *Z-score* dan *Interquartile Range* (IQR). *Z-score* mengidentifikasi *outlier* berdasarkan seberapa jauh nilai data dari rata-rata dalam satuan standar deviasi, dan cocok untuk data yang memiliki distribusi normal. Di sisi lain, IQR mengidentifikasi *outlier* dengan menghitung rentang antar kuartil ($Q3 - Q1$) dan menentukan batas bawah dan atas, yaitu $Q1 - 1.5IQR$ dan $Q3 + 1.5IQR$. Dalam proses ini digunakan metode *clipping*, yaitu membatasi nilai-nilai *outlier* ke dalam rentang yang ditentukan oleh ambang batas masing-masing metode. Metode *Z-score* diterapkan pada fitur-fitur tertentu yang sensitif terhadap nilai ekstrim, seperti

NoOfURLRedirects, sedangkan metode IQR digunakan pada fitur lainnya yang lebih beragam.

Tabel 2.2 Algoritma *Dealing with Outliers*

```
# Z-score handling function
def handle_outliers_zscore(data, features, threshold=3):
    for feature in features:
        mean = data[feature].mean()
        std = data[feature].std()
        lower_limit = mean - threshold * std
        upper_limit = mean + threshold * std
        data[feature] = np.clip(data[feature], lower_limit,
upper_limit)

# IQR handling function
def handle_outliers_iqr(data, features, factor=1.5):
    for feature in features:
        Q1 = data[feature].quantile(0.25)
        Q3 = data[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_limit = Q1 - factor * IQR
        upper_limit = Q3 + factor * IQR
        data[feature] = np.clip(data[feature], lower_limit,
upper_limit)

# Features with Z-score method
zscore_features = [
    'NoOfURLRedirect', 'DomainLength', 'NoOfSelfRedirect',
    'NoOfDegitsInURL', 'NoOfPopup'
]

# Features with IQR method
iqr_features = [
    'CharContinuationRate', 'SpacialCharRatioInURL', 'DegitRatioInURL',
    'URLCharProb', 'LetterRatioInURL', 'NoOfiFrame', 'NoOfSubDomain',
    'LineOfCode', 'NoOfJS', 'NoOfImage', 'NoOfExternalRef',
    'NoOfOtherSpecialCharsInURL', 'NoOfLettersInURL', 'NoOfEmptyRef',
```

```

        'URLLength', 'TLDLength', 'LargestLineLength', 'NoOfCSS',
        'DomainTitleMatchScore', 'URLTitleMatchScore'
    ]

    # Handle outliers using Z-score
    handle_outliers_zscore(df_new, zscore_features)

    # Handle outliers using IQR
    handle_outliers_iqr(df_new, iqr_features)

    outlier_results = {}

    for col in numeric_cols:
        if col.lower() in ['id', 'label']: # Skip ID and label columns
            continue
        col_data = df_new[col].dropna()
        outliers_z = detect_outliers_zscore(col_data)
        outliers_iqr = detect_outliers_iqr(col_data)
        outlier_results[col] = {
            'zscore': outliers_z.sum(),
            'iqr': outliers_iqr.sum(),
            'total': len(col_data)
        }

    # Summary of Outliers
    outlier_summary = pd.DataFrame(outlier_results).T
    if not outlier_summary.empty:
        outlier_summary['zscore_percent'] = (outlier_summary['zscore'] /
        outlier_summary['total']) * 100
        outlier_summary['iqr_percent'] = (outlier_summary['iqr'] /
        outlier_summary['total']) * 100

```

3. Removing Duplicates

Langkah selanjutnya adalah *Removing Duplicates* atau menghapus data duplikat. Data yang berulang sering kali ditemukan dalam dataset dan dapat menyebabkan bias dalam hasil analisis jika tidak ditangani. Dalam proses ini,

baris-baris duplikat diidentifikasi dan dihapus untuk memastikan bahwa setiap data dalam dataset benar-benar unik.

Tabel 2.3 Algoritma *Removing Duplicates*

```
# Check for duplicates
duplicates = df_new.duplicated()

# Count the number of True and False
duplicate_counts = duplicates.value_counts()

# Print the results
print("Duplicates in DataFrame:")
print(duplicate_counts)
```

4. *Feature Engineering*

Terakhir, dilakukan *Feature Engineering*, yaitu proses menciptakan fitur baru atau memodifikasi fitur yang sudah ada untuk mengekstraksi informasi relevan yang dapat membantu meningkatkan performa model. Dalam tahap ini, berbagai cara digunakan untuk menambah informasi dan meningkatkan kualitas data. Misalnya, membuat fitur baru yang menghubungkan variabel-variabel tertentu, mengelompokkan data ke dalam kategori untuk menemukan pola, serta menyederhanakan informasi dengan menggunakan rasio atau hubungan antara variabel. Fitur tambahan juga dibuat untuk menandai data yang tidak biasa, seperti `NoOfURLRedirect` atau `NoOfPopup` yang banyak. Selain itu, transformasi logaritmik dilakukan untuk menormalkan data yang distribusinya tidak merata.

Tabel 2.4 Algoritma *Feature Engineering*

```
# 2.1 Polynomial Features:
# Reason: To capture potential nonlinear relationships, create squared
and cubed terms for features showing high variance or significant
nonlinear trends.
polynomial_features = ['CharContinuationRate', 'URLCharProb']
for feature in polynomial_features:
    df_new[f'{feature}_squared'] = df_new[feature] ** 2
    df_new[f'{feature}_cubed'] = df_new[feature] ** 3
```

```

# 2.2 Interaction Features:
# Reason: Interactions between features like length and counts can
# reveal important combined effects (e.g., URL length and special
# character ratios).
df_new['Length_SpecialCharRatio'] = df_new['DomainLength'] *
df_new['SpacialCharRatioInURL']

# 2.3 Binning or Discretization:
# Reason: Group continuous features like `URLLength` into bins to
# identify patterns based on ranges.
bins = [0, 50, 100, 150, 200, np.inf]
labels = ['Very Short', 'Short', 'Medium', 'Long', 'Very Long']
df_new['URLLength_binned'] = pd.cut(df_new['URLLength'], bins=bins,
labels=labels)

# 2.4 Aggregation Features:
# Reason: Aggregate features to simplify data representation (e.g.,
# ratio of letters and digits in a URL).
df_new['LetterDigitRatio'] = df_new['LetterRatioInURL'] /
(df_new['DegitRatioInURL'] + 1e-9)

# 2.5 Domain-Specific Features:
# Reason: Construct features specific to URL analysis, such as the
# ratio of special characters and total URL length.
df_new['SpecialCharToTotalLength'] = df_new['SpacialCharRatioInURL'] *
df_new['URLLength']

# 2.6 Binary Features from Counts:
# Reason: Identify rows with unusual counts (e.g., number of redirects,
# popups).
df_new['HighRedirects'] = (df_new['NoOfURLRedirect'] > 5).astype(int)
# High redirects flag
df_new['HighPopups'] = (df_new['NoOfPopup'] > 3).astype(int)
# High popups flag

# 2.7 Log Transformation:

```

```
# Reason: Normalize skewed distributions (e.g., counts or rates).
skewed_features = ['NoOfJS', 'NoOfiFrame', 'NoOfSelfRedirect']
for feature in skewed_features:
    df_new[f'{feature}_log'] = np.log1p(df_new[feature]) # log1p
    avoids log(0)
```

Setelah *data cleaning*, tahap berikutnya adalah ***data preprocessing***, yang bertujuan untuk mempersiapkan data agar siap digunakan dalam model *machine learning*. Langkah-langkah dalam data *preprocessing* meliputi scaling fitur numerik, *encoding* variabel kategorikal, menangani ketidakseimbangan kelas, reduksi dimensi, dan normalisasi data. Proses ini membantu memastikan data berada dalam format yang tepat, meningkatkan akurasi model, dan mengurangi bias agar model dapat memberikan hasil yang optimal saat pelatihan.

1. **Feature Scaling**

Kelas `FeatureScaler` adalah transformer yang dirancang untuk melakukan *scaling* dan imputasi pada fitur numerik dalam dataset. Pada langkah pertama, `SimpleImputer` digunakan untuk menggantikan nilai yang hilang pada kolom numerik dengan nilai rata-rata (*mean*). Kemudian, `StandardScaler` diterapkan untuk menormalkan data dan mengubah agar memiliki distribusi dengan rata-rata 0 dan standar deviasi 1. Tujuan dari proses ini adalah agar fitur numerik memiliki skala yang seragam, yang sangat penting untuk algoritma *machine learning* yang peka terhadap skala fitur.

Tabel 2.5 Algoritma Feature Scaling

```
class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self, numerical_columns=None):
        self.numerical_columns = numerical_columns
        self.scaler = StandardScaler()
        self.imputer = SimpleImputer(strategy="mean")

    def fit(self, X, y=None):
        # Fit the scaler and imputer to the numerical columns
        self.imputer.fit(X[self.numerical_columns])
```

```

        self.scaler.fit(self.imputer.transform(X[self.numerical_columns]))
    return self

    def transform(self, X):
        # Transform numerical columns using imputer and scaler
        X[self.numerical_columns] =
self.imputer.transform(X[self.numerical_columns])
        X[self.numerical_columns] =
self.scaler.transform(X[self.numerical_columns])
        return X

```

2. Encoding Categorical Variables

Kelas `FeatureEncoder` adalah sebuah transformer yang dirancang untuk menangani kolom kategorikal dalam dataset. Kelas ini melakukan dua langkah penting dalam memproses data yang pertama adalah menggunakan `SimpleImputer` untuk mengisi nilai yang hilang dengan kategori yang paling sering muncul di kolom tersebut. Kemudian, `OrdinalEncoder` digunakan untuk mengubah kategori menjadi angka yang mana setiap kategori diberi nilai numerik sesuai urutan yang ada. Untuk kategori yang tidak dikenal akan diberikan nilai -1. Kelas ini cocok digunakan dalam *pipeline machine learning* karena memastikan bahwa data kategorikal yang hilang dapat diisi dengan benar dan dapat diproses oleh model dalam format numerik.

Tabel 2.6 Algoritma *Encoding Categorical Variables*

```

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

class FeatureEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, categorical_columns=None):
        self.categorical_columns = categorical_columns
        self.encoder =
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
        self.imputer = SimpleImputer(strategy="most_frequent")

```



```

def fit(self, X, y=None):
    # Fit imputer and encoder to the categorical columns
    self.imputer.fit(X[self.categorical_columns])

    self.encoder.fit(self.imputer.transform(X[self.categorical_columns]))
    return self

def transform(self, X):
    # Impute and encode categorical columns
    X[self.categorical_columns] =
self.imputer.transform(X[self.categorical_columns])
    X[self.categorical_columns] =
self.encoder.transform(X[self.categorical_columns])
    return X

```

3. *Handling Imbalanced Classes*

Kelas `HandleImbalance` merupakan transformer yang digunakan untuk mengatasi masalah ketidakseimbangan kelas dalam dataset dengan menerapkan *undersampling*. Pada bagian konstruktor `__init__`, kelas ini menginisialisasi objek `RandomUnderSampler` dari pustaka *imblearn*, yang berfungsi untuk mengurangi jumlah sampel pada kelas mayoritas hingga mencapai keseimbangan dengan kelas minoritas. Fungsi `fit` hanya digunakan untuk persiapan tanpa melakukan perubahan pada data karena proses resampling dilakukan pada metode `transform`. Di dalam metode `transform` jika target (*y*) diberikan, *undersampling* diterapkan untuk mengurangi sampel dari kelas mayoritas dan menghasilkan dataset yang lebih seimbang. Kelas ini sangat berguna dalam *pipeline machine learning* untuk menangani ketidakseimbangan kelas, sehingga model dapat dilatih dengan data yang lebih proporsional dan meningkatkan kinerja prediksi terhadap kelas minoritas.

Tabel 2.7 Algoritma *Handling Imbalanced Classes*

```

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

```

```

class FeatureEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, categorical_columns=None):
        self.categorical_columns = categorical_columns
        self.encoder =
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
        self.imputer = SimpleImputer(strategy="most_frequent")

    def fit(self, X, y=None):
        # Fit imputer and encoder to the categorical columns
        self.imputer.fit(X[self.categorical_columns])

self.encoder.fit(self.imputer.transform(X[self.categorical_columns]))
        return self

    def transform(self, X):
        # Impute and encode categorical columns
        X[self.categorical_columns] =
self.imputer.transform(X[self.categorical_columns])
        X[self.categorical_columns] =
self.encoder.transform(X[self.categorical_columns])
        return X

```

4. Dimensionality Reduction

Kelas `DimensionalityReduction` adalah transformer yang digunakan untuk mengurangi dimensi fitur dalam dataset menggunakan teknik reduksi dimensi. Pada konstruktor `__init__`, kelas ini menginisialisasi parameter `n_components` yang mengontrol jumlah komponen yang dipertahankan setelah reduksi dimensi. Secara default, `n_components` diatur ke 0.95, yang berarti mempertahankan komponen yang menjelaskan 95% varians dalam data. Metode `fit` digunakan untuk melatih model reduksi dimensi pada dataset, sementara `transform` mengubah dataset dengan mereduksi jumlah fitur sesuai dengan komponen yang telah dipilih. Kelas ini menggunakan `TruncatedSVD` dari *scikit-learn* yang merupakan salah satu teknik reduksi dimensi yang efektif untuk dataset besar dan *sparse*. Kelas `DimensionalityReduction` dapat digunakan dalam *pipeline machine learning*

untuk menyederhanakan data, mengurangi kompleksitas, dan meningkatkan kinerja model dengan menghilangkan fitur yang kurang relevan atau redundan.

Tabel 2.8 Algoritma *Dimensionality Reduction*

```
class DimensionalityReduction(BaseEstimator, TransformerMixin):
    def __init__(self, n_components=0.95, method="SVD"):
        self.n_components = n_components
        self.reducer = TruncatedSVD(n_components=self.n_components)

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

    def transform(self, X):
        return self.reducer.transform(X)
```

5. *Normalization*

Kelas *Normalization* adalah transformer yang digunakan untuk menormalkan fitur dalam dataset dengan menerapkan *Min-Max scaling*. Di dalam konstruktor `__init__`, kelas ini membuat objek *MinMaxScaler* dari *scikit-learn* yang akan digunakan untuk mengubah rentang nilai fitur menjadi lebih terkendali yang biasanya antara 0 dan 1. Metode `fit` digunakan untuk menghitung parameter yang diperlukan untuk transformasi seperti nilai minimum dan maksimum setiap fitur, sementara *transform* kemudian mengaplikasikan normalisasi tersebut pada data dan memastikan semua fitur berada dalam rentang yang konsisten. Kelas *Normalization* sangat berguna dalam *pipeline machine learning* untuk memastikan bahwa setiap fitur numerik berada dalam skala yang seragam.

Tabel 2.9 Algoritma *Normalization*

```
class Normalization(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.scaler = MinMaxScaler()

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

```

        self.scaler.fit(X)
        return self

    def transform(self, X):
        return self.scaler.transform(X)

```

B. Implementasi Algoritma

1. K-Nearest Neighbors (KNN)

a. Tanpa Pustaka (*from scratch*)

Tabel 2.10 Algoritma K-Nearest Neighbors (KNN) From Scratch

Algoritma K-Nearest Neighbors (KNN) From Scratch
<pre> # K-Nearest Neighbors (KNN) Implementation From Scratch class KNN: def __init__(self, k, metric, p): """ Initialize parameters: k, distance metric, and p (for Minkowski metric). """ self.k = k self.metric = metric self.p = p self.X_train = None self.y_train = None def fit(self, X, y): """ Store training data (X_train and y_train) as attributes. """ self.X_train = np.array(X) self.y_train = np.array(y) def _distance(self, X): """ Compute distances between test points and training points based on the selected metric. </pre>

```

        """
        if self.metric == 'euclidean':
            return np.sqrt(np.sum((self.X_train - X[:, np.newaxis]) **
2, axis=2))
        elif self.metric == 'manhattan':
            return np.sum(np.abs(self.X_train - X[:, np.newaxis]),
axis=2)
        elif self.metric == 'minkowski':
            return np.sum(np.abs(self.X_train - X[:, np.newaxis]) **
self.p, axis=2) ** (1 / self.p)
        else:
            raise ValueError("Unknown metric: Choose 'euclidean',
'manhattan', or 'minkowski'")

    def predict(self, X):
        """
        Predict the class labels for the input data.
        """
        X = np.array(X)
        distances = self._distance(X)
        k_neighbors = np.argsort(distances, axis=1)[: , :self.k]
        k_labels = self.y_train[k_neighbors]
        predictions = [np.bincount(labels).argmax() for labels in
k_labels]
        return np.array(predictions)

    def batch_predict(self, X, batch_size=100):
        """
        Predicts the labels for the input data in batches.
        """
        predictions = []
        for i in range(0, len(X), batch_size):
            batch = X[i:i+batch_size]
            batch_predictions = self.predict(batch)
            predictions.extend(batch_predictions)
        return np.array(predictions)

```

```

# Inisialisasi dan pelatihan model KNN
knn = KNN(k=3, metric='minkowski', p=3)
knn.fit(X_train_balanced, y_train_balanced)

# Prediksi pada data validasi
y_predictions = knn.batch_predict(X_val_balanced, batch_size=100)

# Evaluasi performa model
print("KNN (from scratch) Classification Report:")
print(classification_report(y_val, y_predictions))
knn_accuracy = accuracy_score(y_val, y_predictions)
print(f"KNN Accuracy: {knn_accuracy}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_val, y_predictions)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues")
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

```

Kelas `KNN` memiliki lima atribut utama untuk menjalankan algoritmanya, yaitu `self.k` (menyimpan jumlah tetangga terdekat yang digunakan untuk menentukan kelas sampel yang diprediksi), `self.metric` (menyimpan metode perhitungan jarak yang dipilih, yaitu 'euclidean', 'manhattan', atau 'minkowski'), `self.p` (menyimpan parameter eksponen untuk metrik Minkowski), dan `self.X_train` serta `self.Y_train` (data pelatihan untuk perhitungan jarak dan prediksi). Penjelasan *method* lainnya adalah sebagai berikut:

1. `__init__`

Method `__init__` merupakan konstruktor yang digunakan untuk menginisialisasi objek kelas `KNN`.

2. `fit`

Method `fit` digunakan untuk menyimpan data pelatihan, yaitu fitur `x` dan label `Y`, ke dalam atribut `self.X_train` dan `self.Y_train` yang kemudian akan digunakan selama proses prediksi.

3. `_distance`

Method internal `_distance` digunakan untuk menghitung jarak antara data uji dengan data pelatihan berdasarkan numerik yang dipilih, yaitu `'euclidean'`, `'manhattan'`, atau `'minkowski'`. *Method* ini akan mengembalikan array berisi jarak setiap titik uji terhadap seluruh titik pelatihan.

4. `predict`

Method `predict` digunakan untuk menghitung jarak antara data uji dan data pelatihan menggunakan *Method* `_distance`. *Method* ini akan memilih `k` tetangga terdekat berdasarkan jarak. Selanjutnya, menentukan kelas mayoritas di antara tetangga untuk memprediksi label. Menghitung distribusi label tetangga dan memilih yang terbanyak dilakukan dengan fungsi `np.bincount`.

5. `batch_predict`

Method `batch_predict` digunakan untuk membagi data uji ke dalam batch agar proses prediksi bisa dilakukan secara bertahap untuk menangani data besar.

b. Dengan Pustaka (*using library*)

Tabel 2.11 Algoritma K-Nearest Neighbors (KNN) Using Library

Algoritma K-Nearest Neighbors (KNN) Using Library
<pre># K-Nearest Neighbors (KNN) Implementation Using Library from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import accuracy_score, classification_report, confusion_matrix import seaborn as sns import matplotlib.pyplot as plt # K-Nearest Neighbors (KNeighborsClassifier) print("\n--- K-Nearest Neighbors ---") k = 3 # Nilai K</pre>

```

knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Gunakan data hasil resampling untuk melatih model
knn_classifier.fit(X_train_balanced, y_train_balanced)

# Prediksi pada data validasi
y_pred_knn = knn_classifier.predict(X_val_balanced)

# Evaluasi KNN
print("KNN Classification Report:")
print(classification_report(y_val, y_pred_knn))
knn_accuracy = accuracy_score(y_val, y_pred_knn)
print(f"KNN Accuracy: {knn_accuracy * 100:.2f}%")

# Matriks Kebingungan
conf_matrix = confusion_matrix(y_val, y_pred_knn)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues")
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

```

Algoritma K-Nearest Neighbors (KNN) memiliki dua *method* utama yang menjadi inti dari proses proses klasifikasi, yaitu `fit` (melatih model menggunakan data pelatihan) dan `predict` (memprediksi label kelas untuk data baru). Penjelasan *method* lainnya adalah sebagai berikut:

1. `KNeighborsClassifier()`

Method `KNeighborsClassifier()` digunakan untuk membuat objek model K-Nearest Neighbors dari *library* Scikit-learn. Parameter utama yang digunakan adalah `n_neighbors` yang menentukan jumlah tetangga terdekat yang digunakan dalam prediksi.

2. `fit`

Method `fit` digunakan untuk melatih model menggunakan data pelatihan yang telah disiapkan. Model ini menyimpan data pelatihan untuk digunakan dalam proses pencarian tetangga terdekat ketika prediksi dilakukan.

3. `predict`

Method `predict` digunakan untuk memprediksi label kelas pada data validasi. Proses prediksi dilakukan dengan menghitung jarak antara data yang akan diprediksi dengan data pelatihan, kemudian menentukan kelas mayoritas dari tetangga terdekat sesuai nilai `n_neighbors`.

4. `classification_report`

Method `classification_report` digunakan untuk mengevaluasi performa model. *Method* ini menghasilkan laporan klasifikasi yang mencakup *precision*, *F1-score*, dan *support* untuk setiap kelas.

5. `accuracy_score`

Method `accuracy_score` digunakan untuk mengevaluasi performa model dengan menunjukkan rasio prediksi benar terhadap total data validasi.

6. `confusion_matrix`

Method `confusion_matrix` digunakan untuk memberikan gambaran lebih mendetail mengenai performa model. Matriks ini menunjukkan jumlah prediksi benar dan salah untuk setiap kelas, memberikan wawasan mengenai kesalahan spesifik dalam klasifikasi.

2. Gaussian Naive-Bayes

a. Tanpa Pustaka (*from scratch*)

Tabel 2.12 Algoritma Gaussian Naive-Bayes From Scratch

Algoritma Gaussian Naive-Bayes From Scratch
<pre># Gaussian Naive Bayes Implementation From Scratch class GaussianNaiveBayes: def __init__(self): self.classes = None self.class_stats = {}</pre>

```

def fit(self, X, y):
    """
    X: Training features (numpy array)
    y: Training labels (numpy array)
    """
    self.classes = np.unique(y) # Identify unique classes
    self.class_stats = defaultdict(dict)

    for c in self.classes:
        X_c = X[y == c] # Filter data belonging to class c
        self.class_stats[c]['mean'] = X_c.mean(axis=0)
        self.class_stats[c]['std'] = X_c.std(axis=0)
        self.class_stats[c]['prior'] = X_c.shape[0] / X.shape[0] #
P(c)

def _calculate_likelihood(self, X, mean, std):
    """Calculate Gaussian likelihood for given data."""
    eps = 1e-9 # Small constant to avoid division by zero
    coeff = 1 / (np.sqrt(2 * np.pi) * std + eps)
    exponent = -((X - mean) ** 2) / (2 * (std ** 2 + eps))
    return coeff * np.exp(exponent)

def _calculate_posterior(self, X):
    """Calculate posterior probabilities for each class."""
    posteriors = []

    for c in self.classes:
        prior = np.log(self.class_stats[c]['prior']) # log(P(c))
        likelihood = np.sum(
            np.log(self._calculate_likelihood(X,
self.class_stats[c]['mean'], self.class_stats[c]['std']))
        ) # log(P(x|c))
        posterior = prior + likelihood # log(P(c)) + log(P(x|c))
        posteriors.append(posterior)

    return self.classes[np.argmax(posteriors)]

```

```

def predict(self, X):
    """Predict class labels for the given data."""
    return np.array([self._calculate_posterior(x) for x in X])

gnb_scratch = GaussianNaiveBayes()
gnb_scratch.fit(X_train_balanced, y_train_balanced)
y_pred_scratch = gnb_scratch.predict(X_val_balanced)

# Evaluate the model
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
print("\n--- Naive Bayes ---")
print("Gaussian Naive Bayes (from scratch) Classification Report:")
print(classification_report(y_val, y_pred_scratch))
scratch_accuracy = accuracy_score(y_val, y_pred_scratch)
print(f"Accuracy: {scratch_accuracy}")

# Confusion Matrix
conf_matrix_scratch = confusion_matrix(y_val, y_pred_scratch)
sns.heatmap(conf_matrix_scratch, annot=True, fmt='d', cmap="Blues")
plt.title("Gaussian Naive Bayes (from scratch) Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

```

Kelas `GaussianNaiveBayes` merupakan implementasi algoritma Gaussian Naive Bayes *from scratch* yang dirancang untuk melakukan klasifikasi dengan asumsi setiap fitur kontinu mengikuti distribusi Gaussian. Kelas ini memiliki dua atribut utama, yaitu `self.classes` (menyimpan label unik dari setiap kelas) dan `self.class_stats` (menyimpan statistik penting untuk setiap kelas) Penjelasan *method* lainnya adalah sebagai berikut:

1. `__init__`

Method `__init__` merupakan konstruktor yang digunakan untuk menginisialisasi objek kelas `GaussianNaiveBayes`.

2. `fit`

Method `fit` digunakan untuk melatih model dengan menghitung statistik yang diperlukan untuk setiap kelas berdasarkan data pelatihan. Statistik yang dihitung meliputi *mean*, standar deviasi, dan *prior* (probabilitas awal kelas).

3. `_calculate_likelihood`

Method `_calculate_likelihood` digunakan untuk menghitung *likelihood* (kemungkinan) menggunakan distribusi Gaussian.

4. `_calculate_posterior`

Method `_calculate_posterior` digunakan untuk menghitung probabilitas *posterior* untuk setiap kelas. *Posterior* diperoleh dengan menjumlahkan *prior* dan *likelihood* dalam skala logaritmik.

5. `predict`

Method `predict` digunakan untuk memprediksi kelas untuk sekumpulan data. *Method* ini berfungsi sebagai antarmuka utama untuk melakukan klasifikasi menggunakan model Gaussian Naive Bayes.

b. Dengan Pustaka (*using library*)

Tabel 2.13 Algoritma Gaussian Naive-Bayes Using Library

Algoritma Gaussian Naive-Bayes Using Library
<pre># Naive Bayes Classifier (GaussianNB) print("\n--- Naive Bayes ---") nb = GaussianNB() # Melatih model dengan data yang sudah diresampling nb.fit(X_train_balanced, y_train_balanced) # Melakukan prediksi pada data validasi y_pred_nb = nb.predict(X_val_balanced) # Evaluasi Naive Bayes print("Naive Bayes Classification Report:") print(classification_report(y_val, y_pred_nb)) nb_accuracy = accuracy_score(y_val, y_pred_nb) print(f"Naive Bayes Accuracy: {nb_accuracy * 100:.2f}%")</pre>

```
# Matriks Kebingungan Naive Bayes
conf_matrix_nb = confusion_matrix(y_val, y_pred_nb)
sns.heatmap(conf_matrix_nb, annot=True, fmt='d', cmap="Blues")
plt.title("Naive Bayes Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Algoritma `GaussianNB` memiliki dua *method* utama yang menjadi inti dari proses proses klasifikasi, yaitu `fit` (melatih model menggunakan data pelatihan) dan `predict` (memprediksi label kelas untuk data baru). Penjelasan *method* lainnya adalah sebagai berikut:

1. `GaussianNB()`

Method `GaussianNB()` digunakan untuk membuat objek model Gaussian Naive Bayes dari *library* Scikit-learn. Model ini dirancang untuk menangani data kontinu dengan asumsi fitur yang ada mengikuti distribusi Gaussian (normal).

2. `fit`

Method `fit` digunakan untuk data pelatihan yang telah dipreproses. Proses pelatihan ini menghitung *mean*, standar deviasi, dan probabilitas *prior* untuk setiap kelas dalam data pelatihan.

3. `predict`

Method `predict` digunakan untuk memprediksi label kelas pada data validasi. Probabilitas *posterior* dihitung menggunakan Teorema Bayes dan model memilih kelas dengan probabilitas tertinggi sebagai prediksi.

4. `classification_report`

Method `classification_report` digunakan untuk mengevaluasi performa model. *Method* ini menghasilkan laporan klasifikasi yang mencakup *precision*, *F1-score*, dan *support* untuk setiap kelas.

5. `accuracy_score`

Method `accuracy_score` digunakan untuk mengevaluasi performa model dengan menunjukkan rasio prediksi benar terhadap total data validasi.

6. `confusion_matrix`

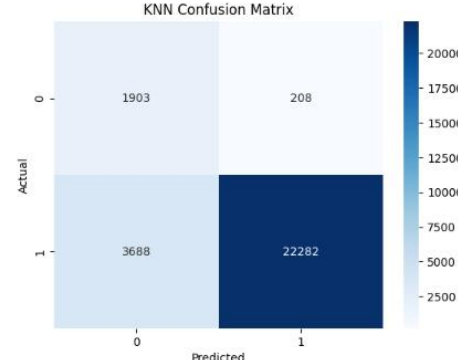
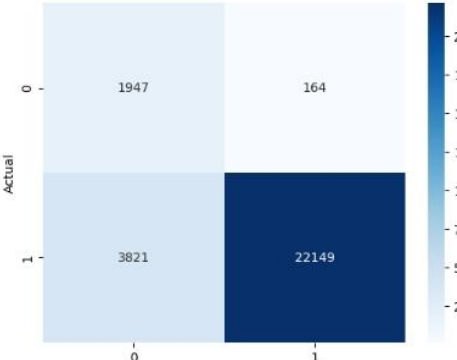
Method `confusion_matrix` digunakan untuk memberikan gambaran lebih mendetail mengenai performa model.

BAB III

Hasil dan Analisis

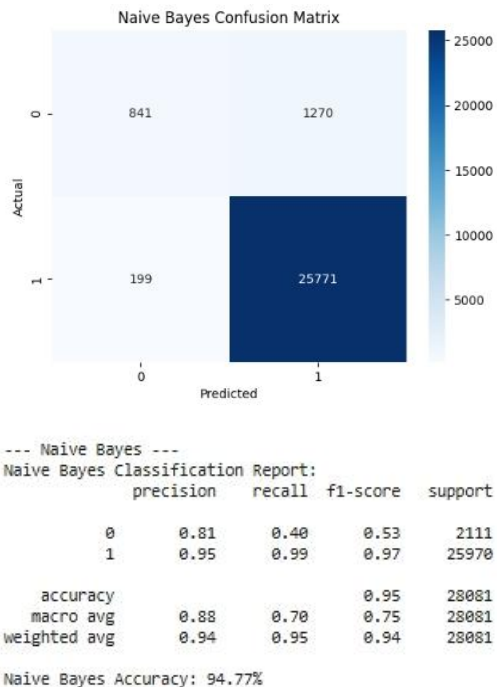
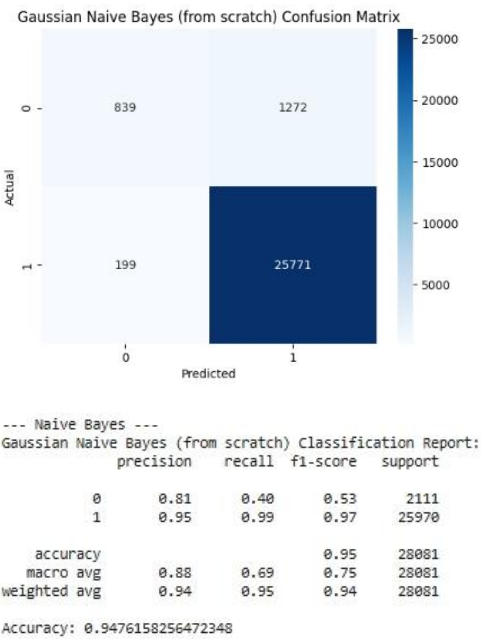
A. Perbandingan Model Sklearn VS Model From Scratch

Tabel 3.1 Perbandingan Model Sklearn vs Model *From Scratch*

Model	From Scratch	Using Library																																																																														
KNN Minkowski	<div><p>KNN Accuracy: 0.8612585021900929</p><p>KNN Confusion Matrix</p><table><tr><th>Actual \ Predicted</th><th>0</th><th>1</th></tr><tr><th>0</th><td>1903</td><td>208</td></tr><tr><th>1</th><td>3688</td><td>22282</td></tr></table><p>KNN (from scratch) Classification Report:</p><table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.34</td><td>0.90</td><td>0.49</td><td>2111</td></tr><tr><td>1</td><td>0.99</td><td>0.86</td><td>0.92</td><td>25970</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.86</td><td>28081</td></tr><tr><td>macro avg</td><td>0.67</td><td>0.88</td><td>0.71</td><td>28081</td></tr><tr><td>weighted avg</td><td>0.94</td><td>0.86</td><td>0.89</td><td>28081</td></tr></table><p>KNN Accuracy: 0.8612585021900929</p></div>	Actual \ Predicted	0	1	0	1903	208	1	3688	22282		precision	recall	f1-score	support	0	0.34	0.90	0.49	2111	1	0.99	0.86	0.92	25970	accuracy			0.86	28081	macro avg	0.67	0.88	0.71	28081	weighted avg	0.94	0.86	0.89	28081	<div><p>KNN Confusion Matrix</p><table><tr><th>Actual \ Predicted</th><th>0</th><th>1</th></tr><tr><th>0</th><td>1947</td><td>164</td></tr><tr><th>1</th><td>3821</td><td>22149</td></tr></table><p>--- K-Nearest Neighbors ---</p><p>KNN Classification Report:</p><table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.34</td><td>0.92</td><td>0.49</td><td>2111</td></tr><tr><td>1</td><td>0.99</td><td>0.85</td><td>0.92</td><td>25970</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.86</td><td>28081</td></tr><tr><td>macro avg</td><td>0.67</td><td>0.89</td><td>0.71</td><td>28081</td></tr><tr><td>weighted avg</td><td>0.94</td><td>0.86</td><td>0.89</td><td>28081</td></tr></table><p>KNN Accuracy: 85.81%</p></div>	Actual \ Predicted	0	1	0	1947	164	1	3821	22149		precision	recall	f1-score	support	0	0.34	0.92	0.49	2111	1	0.99	0.85	0.92	25970	accuracy			0.86	28081	macro avg	0.67	0.89	0.71	28081	weighted avg	0.94	0.86	0.89	28081
Actual \ Predicted	0	1																																																																														
0	1903	208																																																																														
1	3688	22282																																																																														
	precision	recall	f1-score	support																																																																												
0	0.34	0.90	0.49	2111																																																																												
1	0.99	0.86	0.92	25970																																																																												
accuracy			0.86	28081																																																																												
macro avg	0.67	0.88	0.71	28081																																																																												
weighted avg	0.94	0.86	0.89	28081																																																																												
Actual \ Predicted	0	1																																																																														
0	1947	164																																																																														
1	3821	22149																																																																														
	precision	recall	f1-score	support																																																																												
0	0.34	0.92	0.49	2111																																																																												
1	0.99	0.85	0.92	25970																																																																												
accuracy			0.86	28081																																																																												
macro avg	0.67	0.89	0.71	28081																																																																												
weighted avg	0.94	0.86	0.89	28081																																																																												
Insight KNN	Berdasarkan hasil analisis pada gambar di atas, algoritma <i>K-Nearest Neighbors</i> (KNN) yang diimplementasikan secara manual (" <i>From Scratch</i> ") menunjukkan tingkat akurasi sebesar 86.12%, yang hampir sama dengan akurasi KNN yang dihasilkan dari <i>library scikit-learn</i> (85.81%). Kedua metode ini juga memiliki distribusi matriks evaluasi lainnya, seperti <i>precision</i> , <i>recall</i> , dan <i>f1-score</i> , yang cukup konsisten, terutama untuk kelas mayoritas (<i>label 1</i>). Implementasi manual menghasilkan <i>precision</i> sebesar 0.34 dan <i>recall</i> sebesar 0.90 untuk kelas minoritas (<i>label 0</i>), yang menunjukkan bahwa model sangat baik dalam mengenali sebagian besar data kelas 0, tetapi cenderung menghasilkan <i>false positive</i> yang tinggi. Hal serupa juga terlihat pada model																																																																															

scikit-learn, dengan *precision* 0.34 dan *recall* 0.92 untuk label yang sama. Namun, untuk kelas mayoritas (label 1), kedua metode menunjukkan performa yang sangat baik dengan *precision* 0.99 dan *recall* 0.85. Perbedaan kecil dalam akurasi dan performa ini menunjukkan bahwa implementasi manual KNN berhasil mengikuti logika dasar algoritma yang digunakan oleh *scikit-learn*. Namun, penggunaan *library* memberikan efisiensi dalam hal waktu dan optimasi, sedangkan implementasi manual berguna untuk memahami detail teknis algoritma.

Naive Bayes



Insight Gaussian Naive Bayes

Berdasarkan hasil analisis pada gambar di atas, algoritma *Gaussian Naive Bayes* yang diimplementasikan secara manual ("*From Scratch*") menunjukkan tingkat akurasi sebesar 94.76%, yang hampir sama dengan akurasi *Gaussian Naive Bayes* yang dihasilkan dari library *scikit-learn* (94.77%). Kedua metode ini memiliki distribusi matriks evaluasi lainnya, seperti *precision*, *recall*, dan *f1-score*, yang cukup serupa, terutama untuk kelas mayoritas (label 1). Implementasi manual menghasilkan *precision* sebesar 0.81 dan *recall* sebesar 0.40 untuk kelas minoritas (label 0), yang menunjukkan bahwa model

	<p>mengalami kesulitan dalam mengenali kelas ini. Hal serupa juga terlihat pada model <i>scikit-learn</i>, dengan <i>precision</i> 0.81 dan <i>recall</i> 0.40 untuk label yang sama. Namun, untuk kelas mayoritas (label 1), kedua metode menunjukkan performa yang sangat baik dengan <i>precision</i> 0.95 dan <i>recall</i> 0.99. Perbedaan kecil dalam hasil ini menunjukkan bahwa implementasi manual <i>Gaussian Naive Bayes</i> berhasil mengikuti logika dasar algoritma yang digunakan oleh <i>scikit-learn</i>. Penggunaan <i>library</i> memberikan keunggulan dalam hal efisiensi waktu dan optimasi, sedangkan implementasi manual berguna untuk memahami detail teknis algoritma dan penerapan matematisnya.</p>
--	---

Pembagian Tugas

NIM	Nama	Pembagian Tugas
18222026	Tamara Mayranda Lubis	- Preprocessing, Laporan
18222074	Kayla Dyara	- Model KNN, Laporan
18222078	Monica Angela Hartono	- Model Naive-Bayes, Laporan
18222094	Yovanka Sandrina Maharaja	- Cleaning, Laporan

Referensi

- GeeksforGeeks. (n.d.). *Gaussian Naive Bayes*. GeeksforGeeks. Retrieved December 22, 2024, from <https://www.geeksforgeeks.org/gaussian-naive-bayes/>
- Scikit-learn. (n.d.). *Naive Bayes*. Scikit-learn. Retrieved December 20, 2024, from https://scikit-learn.org/1.5/modules/naive_bayes.html
- Scikit-learn. (n.d.). *Neighbors-based learning*. Scikit-learn. Retrieved December 20, 2024, from <https://scikit-learn.org/1.5/modules/neighbors.html>
- StackAbuse. Cássia Sampaio. *Guide to the K-Nearest Neighbors Algorithm in Python and Scikit-Learn*. StackAbuse. Retrieved December 20, 2024, from <https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/>