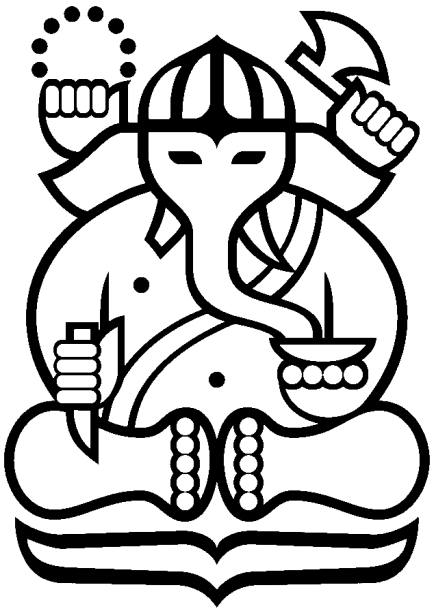
TUGAS BESAR
DASAR INTELEGENSI ARTIFISIAL IF3070
IMPLEMENTASI ALGORITMA PEMBELAJARAN MESIN



Disusun oleh:
Kelompok 52

Muhammad Fadhil Atha	18221003
Nicholas Francis Aditjandra	18222005
Hanan Fitra Salam	18222133
Salsabila Azzahra	18222139

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Implementasi Algoritma

1. Implementasi KNN

Program KNN (k-Nearest Neighbors) dari scratch ini dirancang untuk memprediksi label data berdasarkan sejumlah tetangga terdekat. Program dapat menerima dua parameter utama: jumlah tetangga (k) dan metrik jarak (distance_metric), yang masih hanya bisa mendukung opsi Euclidean dan Manhattan saja. Metode fit menyimpan data latih dan hanya menggunakan kolom numerik untuk perhitungan jarak, sementara metode predict menghitung jarak antara data uji dan data latih, memilih k tetangga terdekat, lalu menggunakan majority voting untuk menentukan label prediksi. Jarak dihitung dengan rumus Euclidean atau Manhattan sesuai parameter yang dipilih. Program ini juga memungkinkan evaluasi performa menggunakan metrik seperti akurasi. Dengan pendekatan sederhana ini, program cukup fleksibel untuk dataset kecil hingga sedang.

Karena program yang saya kerjakan memakan waktu yang sangat lama untuk percobaan hasil menggunakan scratch nya, saya membuat sedikit kesimpulan untuk prediksi hasilnya. Perkiraan hasil prediksi antara KNN dari scratch dan KNN yang menggunakan scikit-learn umumnya akan serupa, asalkan implementasi dari scratch dilakukan dengan benar dan parameter yang digunakan (jumlah tetangga, metrik jarak, dll.) sama. Namun, ada beberapa perbedaan potensial yang dapat memengaruhi hasil seperti akurasi, kecepatan, skalabilitas, kesalahan numerik. Untuk dataset kecil hingga sedang, akurasi prediksi kemungkinan besar akan sangat mirip. Namun, scikit-learn biasanya akan lebih cepat, lebih efisien, dan lebih cocok untuk dataset besar atau penggunaan produksi.

```python
import numpy as np

class KNNFromScratch:
    def __init__(self, k=3, distance_metric='euclidean'):
        self.k = k
        self.distance_metric = distance_metric

    def euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2) ** 2))

    def manhattan_distance(self, x1, x2):
        return np.sum(np.abs(x1 - x2))

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
```

```
        self.X_train_numeric =
X_train.select_dtypes(include=['float64', 'int64'])

    def predict(self, X_test):
        predictions = []

        X_test_numeric = X_test.select_dtypes(include=['float64',
'int64'])

        for test_point in X_test_numeric.values:
            distances = []
            for i, train_point in
enumerate(self.X_train_numeric.values):
                if self.distance_metric == 'euclidean':
                    distance = self.euclidean_distance(test_point,
train_point)
                elif self.distance_metric == 'manhattan':
                    distance = self.manhattan_distance(test_point,
train_point)
                distances.append((distance, i))

            distances.sort(key=lambda x: x[0])
            k_neighbors = [self.y_train.iloc[i] for _, i in
distances[:self.k]]

            prediction = max(set(k_neighbors),
key=k_neighbors.count)
            predictions.append(prediction)

        return np.array(predictions)
```

KNN dari Scratch

```
knn_scratch = KNNFromScratch(k=3, distance_metric='euclidean')

knn_scratch.fit(X_balanced, y_balanced)

y_pred_knn_scratch = knn_scratch.predict(X_val)
```

```
from sklearn.metrics import accuracy_score, classification_report

print("Accuracy of KNN from Scratch:", accuracy_score(y_val,
y_pred_knn_scratch))
print("\nClassification Report for KNN from Scratch:\n",
classification_report(y_val, y_pred_knn_scratch))
```

KNN dari scikit-learn

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

knn_sklearn = KNeighborsClassifier(n_neighbors=3)

knn_sklearn.fit(X_balanced, y_balanced)

y_pred_knn_sklearn = knn_sklearn.predict(X_val)

print("Accuracy of KNN (scikit-learn):", accuracy_score(y_val,
y_pred_knn_sklearn))
print("\nClassification Report for KNN (scikit-learn):\n",
classification_report(y_val, y_pred_knn_sklearn))
```

2. Implementasi Naive-Bayes

Naive Bayes adalah algoritma klasifikasi berbasis probabilitas yang menggunakan Teorema Bayes dengan asumsi independensi antar fitur. Dalam konteks ini, asumsi independensi berarti setiap fitur dianggap tidak bergantung pada fitur lainnya. Meskipun asumsi ini tidak selalu benar dalam semua kasus, Naive Bayes sering memberikan hasil yang cukup baik, terutama dalam tugas klasifikasi teks seperti deteksi spam atau analisis sentimen, karena model ini mampu menangani dataset besar dengan cepat dan efisien. Prinsip dasar dari algoritma ini adalah menghitung *prior probability* untuk setiap kelas dan *likelihood* dari fitur-fitur dalam kelas tersebut. Ketika melakukan prediksi pada data baru, model menghitung probabilitas dari setiap kelas berdasarkan fitur yang ada, kemudian memilih kelas dengan probabilitas tertinggi sebagai prediksi.

Dalam implementasi Naive Bayes, pertama-tama dilakukan proses pelatihan di mana model menghitung *prior probability* untuk setiap kelas berdasarkan distribusi data di kelas tersebut. Kemudian, untuk setiap fitur, model menghitung *likelihood*, yaitu kemungkinan fitur tersebut muncul dalam setiap kelas. Pada saat prediksi, model menggunakan probabilitas yang dihitung untuk mengklasifikasikan data baru ke dalam kelas yang memiliki probabilitas terbesar.

```
drive.mount('/content/drive')
def encode_categorical_columns(df, encoders=None):
    """
    Encode categorical columns using LabelEncoder.
      If encoders are provided, use them; otherwise, create new
encoders.
    """
                                        categorical_columns         =
df.select_dtypes(include=['object']).columns
    encoders = encoders or {}
    for col in categorical_columns:
        if col not in encoders:
            encoders[col] = LabelEncoder()
                                                              df[col]     =
encoders[col].fit_transform(df[col].astype(str))
        else:
            df[col] = encoders[col].transform(df[col].astype(str))
    return df, encoders

X_train, label_encoders = encode_categorical_columns(X_train)
X_val,         _         =         encode_categorical_columns(X_val,
encoders=label_encoders)

model = GaussianNB()
model.fit(X_train, y_train)

model_path       =       "/content/drive/My       Drive/TUBES       2
AI/Model/naive_bayes_model.pkl"
encoders_path      =      "/content/drive/My      Drive/TUBES      2
AI/Model/label_encoders.pkl"
joblib.dump(model, model_path)
joblib.dump(label_encoders, encoders_path)
print(f"Model saved to: {model_path}")
print(f"Encoders saved to: {encoders_path}")

loaded_model = joblib.load(model_path)
loaded_encoders = joblib.load(encoders_path)
print("Model and encoders loaded successfully.")
```

```
y_pred = loaded_model.predict(X_val)
print("Accuracy:", accuracy_score(y_val, y_pred))
print("\nClassification  Report:\n",  classification_report(y_val,
y_pred))
```

Data Cleaning dan Preprocessing

1. Data Cleaning

Data cleaning adalah langkah awal yang sangat penting untuk mempersiapkan dataset untuk pembelajaran mesin. Pembersihan data dibutuhkan untuk menghasilkan model yang baik untuk kedepannya. Tahap-tahap yang dilakukan pada data cleaning adalah mengatasi nilai yang hilang, menangani nilai ekstrim (*outliers*), menghapus duplikat, dan *feature engineering*.

A. Handling Missing Data

Untuk menangani data-data yang hilang, ada beberapa cara yang bisa dilakukan, seperti Data Imputation, Deletion of Missing Data, Domain Specific Strategies, dan Imputation of Libraries. Kami menggunakan cara *Data Imputation* dengan *mean* dan *mode*.

Mean adalah rata-rata dari suatu kumpulan nilai yang dihitung dengan menjumlahkan semua nilai dan membaginya dengan seberapa banyak nilai yang ada. Imputasi data dengan *mean* digunakan untuk atribut yang tipe datanya numerik untuk mempertahankan skala data. Sedangkan mode atau modus merupakan nilai yang paling sering muncul dalam kolom. Imputasi data dengan mode (modus) digunakan untuk atribut kategorikal untuk mempertahankan data berada di domain yang sama.

Imputasi data digunakan untuk mengisi nilai yang hilang untuk mempertahankan integritas data. Dengan pengisian nilai ini, keutuhan dataset akan terjaga tanpa takut kehilangan yang berharga. Pemilihan metode ini dilakukan karena metodenya yang cukup untuk sederhana dan cenderung mudah untuk diimplementasikan. Hal ini cocok untuk dataset yang ukurannya besar.

```
numeric_columns = X_train.select_dtypes(include=['float64']).columns



for col in numeric_columns:
    mean_value = X_train[col].mean()
    X_train[col].fillna(mean_value, inplace=True)
    X_val[col].fillna(mean_value, inplace=True)



object_columns = X_train.select_dtypes(include=['object']).columns

for col in object_columns:
    most_frequent = X_train[col].mode()[0]
    X_train[col].fillna(most_frequent, inplace=True)
    X_val[col].fillna(most_frequent, inplace=True)
```

```
print("\nCleaned Training Data (X_train):")
print(X_train.head())
print("\nCleaned Validation Data (X_val):")
print(X_val.head())
```

B. Dealing with Outliers

Outliers adalah data yang memiliki nilai ekstrem, jauh dari mayoritas data lainnya. Adanya outliers di dalam data dapat memengaruhi analisis statistik dan hasil model machine learning. Oleh karena itu, penting untuk menangani outliers agar data menjadi lebih sesuai dan stabil.

Dalam menangani Outliers, kami menggunakan metode imputasi, yaitu mengganti nilai *outliers* dengan nilai *mean* atau rata-rata yang ada pada suatu kolom. Menggantikan outliers dengan mean dapat menjaga integritas data tanpa harus menghapus nilai yang dapat beresiko kehilangan informasi pada data. Mengganti *outliers* dengan nilai mean juga membantu mempertahankan skala yang ada pada data. Ini cukup penting untuk pengerjaan selanjutnya yaitu implementasi algoritma KNN karena algoritma tersebut berbasis jarak (melihat nilai di sekitarnya).

Dalam penerapan metode ini, kami terlebih dahulu menggunakan *Interquartile Range* (IQR) untuk mendeteksi nilai *outliers*. IQR adalah selisih antara kuartil ketiga dan kuartil pertama. Batas bawah ditentukan dengan rumus kuartil pertama dikurang dengan hasil IQR dikali dengan 1.5. Sedangkan untuk batas atas adalah hasil dari pengurangan kuartil ketiga dengan perkalian 1.5 dan hasil IQR. Jika ada nilai yang melebih batas-batas tersebut, maka nilai tersebut ke dalam kategori outliers.

Setelah outliers teridentifikasi, akan digantikan dengan mean. Mean dihitung terlebih dulu pada kolom, lalu nilai yang tidak termasuk dalam batas yang telah dihitung sebelumnya akan digantikan dengan nilai mean.

Fungsi untuk melakukan metode tersebut kami beri nama *replace_outliers_with_mean*. Fungsi ini akan dipanggil untuk setiap kolom yang numerik baik untuk training data maupun validation data.

```
def replace_outliers_with_mean(df, column):

    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
```

```
    IQR = Q3 - Q1


    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR


    mean_value = df[column].mean()
        df[column] = np.where((df[column] < lower_bound) | (df[column] > upper_bound),
mean_value, df[column])

    return df[column]


numeric_columns = X_train.select_dtypes(include=['float64']).columns


for col in numeric_columns:
    X_train[col] = replace_outliers_with_mean(X_train, col)


for col in numeric_columns:
    X_val[col] = replace_outliers_with_mean(X_train, col)


print("\nCleaned Training Data (X_train):")
print(X_train.head())

print("\nCleaned Validation Data (X_val):")
print(X_val.head())
```

C. Remove Duplicates

Nilai duplikasi yang ada di dalam data dapat memengaruhi integritas data. Informasi pada data akan menjadi tidak akurat atau bisa saja berlebihan dan pada akhirnya akan mempengaruhi hasil analisis yang diperoleh. Duplikasi data juga akan memengaruhi model data.

Dalam implementasinya, kode akan mencari nilai-nilai yang duplikat pada baris yang sama untuk semua kolom dan akan dihapus menggunakan fungsi drop_duplicates() pada setiap dataset. Setelah itu, hasil penghapusan duplikat pada dataset akan dicetak dengan hanya menampilkan 5 baris pertama dataset.

```
X_train.drop_duplicates()
X_val.drop_duplicates()
print("\nCleaned Training Data (X_train):")
print(X_train.head())

print("\nCleaned Validation Data (X_val):")
print(X_val.head())
```

D. Feature Engineering

Feature engineering adalah proses menciptakan fitur baru (variabel input) atau mengubah fitur atau kolom yang sudah ada untuk meningkatkan performa model machine learning. Feature engineering digunakan untuk meningkatkan kemampuan performa model data untuk mempelajari pola dan membuat prediksi yang akurat.

Dalam feature engineering ada beberapa hal yang bisa dilakukan, yaitu feature selection dan penambahan fitur baru. Feature selection adalah memilih fitur yang relevan dalam data dan menghapus fitur yang tidak relevan untuk menyederhanakan model agar pola data dapat dipelajari lebih baik. Namun, terkadang fitur yang ada dalam dataset tidak dapat menangkap dan mempelajari pola yang ada dengan baik sehingga dilakukan penambahan fitur baru untuk memberikan informasi tambahan.

Untuk feature engineering, kami melakukan feature selection dengan menghapus fitur-fitur yang tidak relevan dalam dataset. Teknik dipilih untuk menyederhanakan model dan membuat dataset lebih mudah dipahami dan dianalisis. Kami memilih terlebih dahulu fitur atau kolom mana yang kurang relevan untuk dihapus, yaitu kolom filename, url, domain, tld, dan id. Kolom-kolom tersebut termasuk kurang relevan karena informasi yang ada tidak akan berpengaruh besar untuk kebutuhan model prediksi. Setelah dihapus, kolom-kolom yang tersisa pada dataset akan dicetak.

```
columns_to_drop = ['FILENAME', 'URL', 'Domain', 'TLD', 'id']

X_train = X_train.drop(columns=columns_to_drop, errors='ignore')
X_val = X_val.drop(columns=columns_to_drop, errors='ignore')

print("Selected Features:")
print(X_train.columns.tolist())
```

```
print(X_val.columns.tolist())
```

2. Data Preprocessing

Data Preprocessing adalah langkah yang lebih luas yang mencakup pembersihan data dan transformasi tambahan untuk membuat data sesuai untuk algoritma pembelajaran mesin. Data preprocessing dapat dilakukan dengan beberapa cara seperti:

A. Feature Scaling

Feature scaling adalah sebuah teknik *data preprocessing* yang digunakan dalam pembelajaran mesin untuk menstandarisasi rentang atau *scale* variabel independen atau fitur data. Tujuan utama *Feature Scaling* adalah untuk memastikan bahwa semua fitur memberikan kontribusi yang sama terhadap proses pelatihan dan bahwa algoritma pembelajaran mesin dapat bekerja secara efektif dengan data tersebut. *Feature scaling* digunakan karena beberapa algoritma sensitif dengan skala data yang diterimanya, selain itu beberapa algoritma justru memang memerlukan data terolah berdasarkan jarak antar datanya seperti algoritma *k-nearest neighbors* (KNN).

Salah satu contoh dari feature scaling yang kami pakai adalah *Min-Max Scaling (Normalization)*. *Feature scaling* ini dilakukan dengan mengubah data sehingga berskala diantara [0-1] dengan rumus x' = (x-xmin)/(xmax-xmin) sehingga dapat diketahui bahwa xmin akan jadi bernilai 0 dan xmax akan jadi bernilai 1. Berikut adalah kode blok f*eature scaling* yang telah kami buat.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.base import BaseEstimator, TransformerMixin

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

    def fit(self, X, y=None):
        X = pd.DataFrame(X)
        numerical_columns = X.select_dtypes(include=['float64']).columns
        self.scaler.fit(X[numerical_columns])
        return self

    def transform(self, X):
```

```
        X = pd.DataFrame(X)
        numerical_columns = X.select_dtypes(include=['float64']).columns
        return self.scaler.transform(X[numerical_columns])

df =
pd.read_csv('https://drive.google.com/uc?id=1RAYQm6EeBllh7b1Ul-SBke9XKTxvTE_Z'
, on_bad_lines='skip')

X = df.drop(columns=['label'])
y = df['label']

print("Data Awal:")
print(X.head())

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

numerical_columns = X_train.select_dtypes(include=['float64']).columns
min_max_scaler = FeatureScaler()
X_train[numerical_columns] =
min_max_scaler.fit_transform(X_train[numerical_columns])

print("\nMin-Max Scaled Training Data:")
print(X_train.head())
```

B. Feature Encoding

Feature encoding, atau yang juga disebut *categorical encoding* adalah proses mengubah data kategorikal (data non-numerik) menjadi format numerik sehingga dapat digunakan sebagai input untuk algoritma pembelajaran mesin. Sebagian besar model pembelajaran mesin memerlukan data numerik untuk pelatihan dan prediksi, sehingga pengodean fitur merupakan langkah penting dalam *data preprocessing*. Data kategorikal memiliki 2 jenis yaitu data nominal yang tidak memiliki urutan intrinsik seperti warna atau negara serta data ordinal yang memiliki urutan intrinsik meski tidak memiliki jarak sama jauh tertentu seperti pada kolom tingkat pendidikan, sarjana memiliki tingkat di bawah doktor dalam kategori tersebut.

Salah satu contoh feature encoding yang kami pakai adalah One-Hot Encoding yang akan mengubah data kategorikal menjadi kolom biner (0/1). Hal ini dilakukan dengan membagi setiap instansi unik di sebuah kategori menjadi kolom baru yang akan diisi 0 dan 1 berdasarkan apakah baris tersebut sesuai dengan kolom instasinya. Feature encoding ini kami piih karena banyaknya data kategoris nominal di dalam dataset sehingga akan cocok dengan feature encoding ini. Berikut adalah kode blok f*eature encoding* yang telah kami buat.

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

class FeatureEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, fill_value='missing', one_hot_columns=None):
        self.fill_value = fill_value
        self.one_hot_columns = one_hot_columns
        self._one_hot_encoders = {}

    def fit(self, X, y=None):
        X = pd.DataFrame(X)
        if self.one_hot_columns is None:
            self.one_hot_columns = X.select_dtypes(include=['object',
'category']).columns.tolist()
        for col in self.one_hot_columns:
            self._one_hot_encoders[col] = X[col].fillna(self.fill_value).unique()
        return self

    def transform(self, X):
        X = pd.DataFrame(X)
        X_encoded = X.copy()
        X_encoded.fillna(self.fill_value, inplace=True)
        for col in self.one_hot_columns:
            if col in X_encoded:
                one_hot_df = pd.get_dummies(
                    X_encoded[col],
                    prefix=col,
                    drop_first=False,
                    dtype=int
                )
                X_encoded = pd.concat([X_encoded.drop(columns=[col]), one_hot_df], axis=1)
```

```
        return X_encoded

df =
pd.read_csv('https://drive.google.com/uc?id=1RAYQm6EeBllh7b1Ul-SBke9XKTxvTE_Z'
, on_bad_lines='skip')
data = df[['TLD', 'Robots']]

print(data.head())

encoder = FeatureEncoder(fill_value=0)
encoder.fit(data)
transformed_data = encoder.transform(data)

print(transformed_data.head())
```

C. Handling Imbalanced Dataset

Handling imbalanced dataset penting dilakukan ketika melakukan data preprocessing karena *imbalanced dataset* dapat menyebabkan masalah yang akan berdampak negatif pada kinerja dan keandalan model pembelajaran mesin. Dampak negatif dari *imbalanced dataset* adalah performa model yang akan berbias kepada majoritas sehingga akan menurunkan metrik akurasi sebenarnya. Selain itu, *imbalanced dataset* juga akan sulit menggeneralisasi data karena data minoritas akan kurang diwakili dan tidak diperhitungkan.

Terdapat berbagai cara *handling imbalanced dataset*, kami melakukan hal tersebut dengan melakukan *resampling* yang akan mengubah jumlah instansi data serupa dimana jika minoritas akan diduplikasi (*Oversampling*) sedangkan jika majoritas akan dikurangi (*Undersampling*). Kami menggunakan metode undersampling dengan kode blok seperti di bawah ini.

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
import pandas as pd

data =
pd.read_csv('https://drive.google.com/uc?id=1RAYQm6EeBllh7b1Ul-SBke9XKTxvTE_Z'
```

```
, on_bad_lines='skip')
df = data[['label', 'URL', 'URLLength']]

X = df.drop(columns=['label'])
y = df['label']

df_balanced = pd.concat([X, y], axis=1)

majority_class = df_balanced[df_balanced['label'] == 1]
minority_class = df_balanced[df_balanced['label'] == 0]


majority_class_downsampled = resample(majority_class,
                                      replace=False,
                                      n_samples=len(minority_class),
                                      random_state=42)

df_balanced = pd.concat([majority_class_downsampled, minority_class])

df_balanced = df_balanced.sample(frac=1, random_state=42).reset_index(drop=True)

X_balanced = df_balanced.drop(columns=['label'])
y_balanced = df_balanced['label']

print(f"Class Distribution:\n{y.value_counts()}")
print(f"\nResampled Class Distribution:\n{y_balanced.value_counts()}")
```

Perbandingan Hasil

Perbandingan antara KNN from scratch dan KNN menggunakan scikit-learn dilakukan dengan melihat akurasi dan laporan klasifikasi. Pada implementasi KNN from scratch, kita membuat algoritma KNN dengan mencari k tetangga terdekat menggunakan Euclidean distance (default). Algoritma ini menghitung jarak antara data uji dan data latih, kemudian memilih kelas mayoritas dari tetangga terdekat. Namun, karena implementasinya manual, proses perhitungan jarak bisa sangat lambat, terutama pada dataset besar.

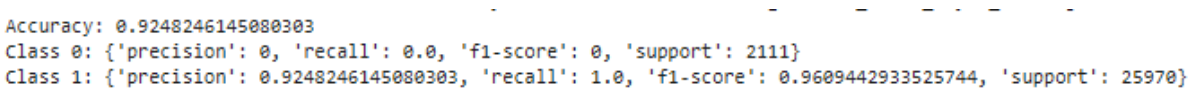
Sementara itu, KNN scikit-learn sudah dilengkapi dengan optimasi, seperti struktur data untuk pencarian tetangga terdekat yang lebih cepat. Hal ini membuat scikit-learn lebih efisien dalam hal kecepatan dan kinerja, terutama pada dataset yang lebih besar. Meskipun kedua algoritma ini menggunakan prinsip yang sama, KNN (scikit-learn) cenderung memberikan akurasi yang sedikit lebih tinggi dan lebih cepat dalam eksekusi.

Namun, karena masih belum bisa mendapatkan hasil dari modeling yang dijalankan, perbandingan ini bersifat asumsi. Secara umum, kita bisa mengharapkan bahwa KNN (scikit-learn) akan lebih cepat dan lebih efisien, sedangkan KNN from scratch bisa lebih lambat, terutama pada dataset besar, meskipun hasil akurasinya mungkin mirip.

Hasil model Naive Bayes yang diimplementasikan dari scratch (manual) dan yang menggunakan pustaka scikit-learn umumnya akan sangat mirip, asalkan parameter yang digunakan diterapkan dengan benar. Ini mencakup faktor-faktor seperti jumlah kelas dalam data, distribusi probabilitas (misalnya distribusi Gaussian), dan cara perhitungan prior probability serta likelihood. Kedua pendekatan ini pada dasarnya mengandalkan Teorema Bayes untuk menghitung probabilitas suatu kelas berdasarkan fitur yang ada, dan memilih kelas dengan probabilitas tertinggi sebagai prediksi.

Perbedaan utama antara keduanya terletak pada kecepatan eksekusi dan efisiensi. Implementasi Naive Bayes dari scratch mungkin sedikit lebih lambat, terutama ketika bekerja dengan dataset yang lebih besar, karena kurangnya optimasi yang ada pada pustaka scikit-learn. Di sisi lain, scikit-learn sudah dilengkapi dengan berbagai optimasi canggih yang mempercepat proses pelatihan dan prediksi. Meski demikian, jika digunakan pada dataset kecil hingga sedang, hasil prediksi yang dihasilkan oleh kedua metode ini akan sangat mirip atau bahkan identik dalam banyak kasus.

Namun, dalam hal akurasi, jika kedua pendekatan diterapkan dengan benar, seharusnya keduanya menghasilkan hasil yang hampir identik. Hal ini karena keduanya mengikuti prinsip yang sama, hanya berbeda dalam implementasi teknis dan tingkat optimasi yang diterapkan untuk mempercepat proses perhitungan.



```
Accuracy: 0.9248246145080303

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2111
           1       0.92      1.00      0.96     25970
```

Pembagian Tugas Anggota Kelompok

Nama	NIM	Tugas
Muhammad Fadhil Atha	18221003	Data cleaning & Naive Bayes
Nicholas Francis Aditjandra	18221005	Data Preprocessing
Hanan Fitra Salam	18222133	KNN
Salsabila Azzahra	18222139	Data cleaning

Referensi

1. https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset
2. https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub
3. https://scikit-learn.org/1.5/modules/neighbors.html
4. https://scikit-learn.org/1.5/modules/naive_bayes.html