Tugas Besar 2

IF3070 Dasar Inteligensi Artifisial

Disusun oleh:

Kelompok 35

Angelica Aliwinata / 18222113
Jason Jahja / 18222116
Melissa Trenggono/ 18222123
Anindita Widya S. / 18222128

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

Daftar Isi

Daftar Tabel

Pendahuluan

Bagian ini menjelaskan persoalan yang dipecahkan dalam dokumen ini serta spesifikasi dari tugas yang dikerjakan.

A. Deskripsi Persoalan

Bagaimana mengimplementasikan algoritma machine learning K-Nearest Neighbor (KNN) dan Gaussian Naive Bayes untuk menyelesaikan permasalahan klasifikasi biner pada PhiUSIIL Phishing URL Dataset.

B. Spesifikasi Tugas

Berikut merupakan hal-hal yang perlu dilakukan oleh setiap kelompok:

1. Implementasi KNN *from scratch*.
    a. Minimal bisa menerima 2 input parameter
        i. Jumlah tetangga
        ii. Metrik jarak antar data point. Minimal dapat menerima 3 pilihan, yaitu Euclidean, Manhattan, dan Minkowski
2. Implementasi Gaussian Naive-Bayes *from scratch*.
3. Implementasi algoritma poin 1-2 menggunakan *scikit-learn*. Bandingkan hasil dari algoritma *from scratch* dan algoritma *scikit-learn*.
4. Model harus bisa di-save dan di-load. Implementasinya dibebaskan (misal menggunakan .txt, .pkl, dll).

Selain itu, terdapat pula spesifikasi untuk bonus yang dapat dikerjakan ketika seluruh spesifikasi wajib sudah diselesaikan. Berikut merupakan spesifikasi bonus yang dapat dikerjakan:

1. Kaggle Submission pada link berikut.

Teori Dasar

Pembelajaran mesin (*Machine Learning*) merupakan percabangan dari kecerdasan buatan (*Artificial Intelligence*) yang memungkinkan sistem untuk berkembang secara mandiri menggunakan jaringan neural dan *deep learning*, tanpa perlu pemrograman secara eksplisit. Melalui *machine learning*, sistem komputer mampu terus belajar dan berkembang seiring bertambahnya pengalaman yang diperolehnya. Dengan menyediakan kumpulan data yang lebih besar dan beragam, kinerja dari sistem ini dapat terus ditingkatkan secara signifikan. Terdapat beberapa kategori utama dalam *machine learning*, salah satunya adalah *supervised learning*. Salah satu pendekatan dalam *supervised learning* adalah dengan memanfaatkan pola dalam data untuk membuat prediksi melalui algoritma tertentu seperti K-Nearest Neighbors (KNN) dan Naive Bayes.

K-Nearest Neighbor (KNN) merupakan algoritma *machine learning* sederhana dengan sifat *non-parametric* yang artinya algoritma tidak membuat asumsi tentang distribusi data atau bentuk fungsi yang mendasari data dan bersifat learning yang berarti algoritma tidak secara eksplisit membangun model selama fase *training* namun menyimpan seluruh data *training* dan melakukan perhitungan saat fase testing. Cara kerja dari KNN adalah dengan menentukan jumlah tetangga (k), menghitung jarak antara data baru dan *data training*, memilih k tetangga terdekat, tentukan kelas mayoritas dari tetangga untuk klasifikasi atau rata-rata nilai untuk regresi, kemudian memprediksi data baru. Kelebihan dari algoritma KNN adalah mudah diterapkan karena kesederhanaan dan akurasinya, mudah beradaptasi karena KNN menyesuaikan untuk perhitungan data baru, dan memiliki lebih sedikit *hyperparameter* jika dibandingkan dengan algoritma machine learning lainnya. Sedangkan, kekurangan dari KNN adalah tidak efektif untuk *dataset* berukuran besar, kurang cocok untuk data berdimensi tinggi, dan sensitif terhadap noise dalam *dataset*.

Naive Bayes merupakan algoritma *machine learning* sederhana berbasis teorema bayes, yang digunakan untuk mengklasifikasikan objek berdasarkan probabilitasnya. Ciri utama dari algoritma ini adalah asumsi yang sangat kuat akan independensi dari masing-masing kondisi/kejadian. Cara kerja dari naive bayes adalah menghitung probabilitas Prior, menghitung likelihood, menghitung posterior probabilitas, kemudian melakukan prediksi kelas. Kelebihan dari algoritma naive bayes adalah tidak memerlukan data *training* yang banyak, bisa digunakan untuk data kuantitatif atau kualitatif, perhitungannya cepat dan efisien, dan untuk proses pemrograman, cenderung menggunakan *code* yang lebih sederhana. Namun terdapat beberapa kekurangan untuk algoritma ini seperti keakuratannya tidak dapat diukur hanya dengan satu probabilitas dan harus didukung oleh bukti-bukti lainnya dan keberhasilannya yang sangat bergantung pada pengetahuan awal atau pengetahuan mengenai masa sebelumnya.

Data Understanding

A. Latar Belakang

Phishing adalah salah satu bentuk ancaman keamanan siber yang paling umum dan berbahaya, di mana pelaku mencoba mencuri informasi sensitif seperti data pribadi, kata sandi, atau informasi keuangan dengan menyamar sebagai entitas yang sah melalui email, situs web, atau media lainnya. Taktik ini memanfaatkan kelemahan manusia dalam mengenali ancaman digital, membuatnya menjadi salah satu metode serangan yang efektif.

Seiring dengan meningkatnya digitalisasi, jumlah serangan phishing terus bertambah secara signifikan. Berdasarkan data global, ribuan URL phishing baru dibuat setiap harinya, dengan target yang mencakup individu, organisasi, hingga pemerintahan. Serangan ini sering kali berhasil karena korban cenderung sulit membedakan antara URL yang sah dan yang mencurigakan.

B. Deskripsi Data

Dataset terdiri dari 140.404 data dengan setiap data memiliki 56 fitur. Berikut merupakan rincian dari setiap fitur yang terdapat pada *dataset*.

Tabel 3.1 Deskripsi dan Jenis Fitur

Fitur	Jenis	Deskripsi
id	Numerik	ID unik untuk setiap entri data
FILENAME	Kategorikal	Nama file yang terkait dengan data URL
URL	Kategorikal	URL yang diperiksa apakah phishing atau tidak
URLLength	Numerik	Panjang URL dalam karakter. URL phishing sering memiliki panjang lebih besar
Domain	Kategorikal	Nama domain dari URL
DomainLength	Numerik	Panjang nama domain dalam karakter
IsDomainIP	Numerik	Indikator apakah domain menggunakan alamat IP langsung (1) atau nama domain (0)
TLD	Kategorikal	Top-Level Domain (TLD) dari

		URL, seperti .com, .org
CharContinuationRate	Numerik	Rasio karakter berkelanjutan dalam URL, yang menunjukkan kompleksitas URL
TLDLegitimateProb	Numerik	Probabilitas bahwa TLD adalah legitimate berdasarkan data historis
URLCharProb	Numerik	Probabilitas karakter tertentu dalam URL dibandingkan dengan URL legitimate
TLDLength	Numerik	Panjang TLD dalam karakter
NoOfSubDomain	Numerik	Jumlah subdomain dalam URL. URL phishing cenderung memiliki lebih banyak subdomain
HasObfuscation	Numerik	Indikator apakah URL menggunakan teknik penyamaran (1 untuk ya, 0 untuk tidak)
NoOfObfuscatedChar	Numerik	Jumlah karakter yang digunakan untuk penyamaran dalam URL
ObfuscationRatio	Numerik	Rasio karakter penyamaran terhadap total panjang URL
NoOfLettersInURL	Numerik	Jumlah huruf alfabet dalam URL
LetterRatioInURL	Numerik	Rasio huruf alfabet terhadap total panjang URL
NoOfDegitsInURL	Numerik	Jumlah digit dalam URL
DegitRatioInURL	Numerik	Rasio digit terhadap total panjang URL
NoOfEqualsInURL	Numerik	Jumlah simbol “=” dalam URL
NoOfQMarkInURL	Numerik	Jumlah simbol “?” dalam URL
NoOfAmpersandInURL	Numerik	Jumlah simbol “&” dalam URL
NoOfOtherSpecialCharsInURL	Numerik	Jumlah simbol khusus lainnya dalam URL
SpacialCharRatioInURL	Numerik	Rasio simbol khusus terhadap

		total panjang URL
IsHTTPS	Numerik	Indikator apakah URL menggunakan protokol HTTPS (1 untuk ya, 0 untuk tidak)
LineOfCode	Numerik	Jumlah baris kode dalam halaman web terkait URL
LargestLineLength	Numerik	Panjang baris kode terpanjang dalam halaman web terkait URL
HasTitle	Numerik	Indikator apakah halaman web memiliki elemen <title> (1 untuk ya, 0 untuk tidak)
Title	Kategorikal	Judul dari halaman web terkait URL
DomainTitleMatchScore	Numerik	Skor kecocokan antara domain dan judul halaman web
URLTitleMatchScore	Numerik	Skor kecocokan antara URL dan judul halaman web
HasFavicon	Numerik	Indikator apakah halaman web memiliki favicon
Robots	Numerik	Indikator keberadaan file robots.txt dalam server
IsResponsive	Numerik	Indikator apakah halaman web responsif terhadap perangkat berbeda
NoOfURLRedirect	Numerik	Jumlah pengalihan URL
NoOfSelfRedirect	Numerik	Jumlah pengalihan ke halaman sendiri
HasDescription	Numerik	Indikator keberadaan deskripsi meta di halaman web
NoOfPopup	Numerik	Jumlah pop-up yang ditemukan di halaman web
NoOfiFrame	Numerik	Jumlah elemen <iframe> dalam halaman web
HasExternalFormSubmit	Numerik	Indikator apakah formulir

		mengarah ke domain eksternal
HasSocialNet	Numerik	Indikator keberadaan tautan media sosial
HasSubmitButton	Numerik	Indikator keberadaan tombol kirim di formulir
HasHiddenFields	Numerik	Indikator keberadaan elemen formulir tersembunyi
HasPasswordField	Numerik	Indikator keberadaan elemen input untuk kata sandi
Bank	Numerik	Indikator apakah URL terkait dengan layanan bank
Pay	Numerik	Indikator apakah URL terkait dengan pembayaran
Crypto	Numerik	Indikator apakah URL terkait dengan cryptocurrency
HasCopyrightInfo	Numerik	Indikator keberadaan informasi hak cipta di halaman web
NoOfImage	Numerik	Jumlah gambar dalam halaman web
NoOfCSS	Numerik	Jumlah file CSS yang diimpor oleh halaman web
NoOfJS	Numerik	Jumlah file JavaScript yang diimpor oleh halaman web
NoOfSelfRef	Numerik	Jumlah referensi yang mengarah ke domain yang sama
NoOfEmptyRef	Numerik	Jumlah referensi kosong dalam halaman web
NoOfExternalRef	Numerik	Jumlah referensi ke domain eksternal

C. *Exploratory Data Analysis (EDA)*

Tahap EDA dilakukan untuk membantu menentukan *cleaning* dan *preprocessing* yang akan digunakan

a. *Duplicate and Missing Value*

Berdasarkan analisis yang dilakukan, didapatkan bahwa tidak ada data duplikat di dalam *dataset*. Tetapi, *missing values* hampir setiap *feature* memiliki *missing value* dalam jumlah yang cukup besar.

```
Missing values per feature:
id                              0
FILENAME                    57532
URL                         43487
URLLength                   60639
Domain                      70197
DomainLength                46319
IsDomainIP                  42130
TLD                         45399
CharContinuationRate        48042
TLDLegitimateProb           52873
URLCharProb                 52071
TLDLength                   47731
NoOfSubDomain               44060
HasObfuscation              65720
NoOfObfuscatedChar          66798
ObfuscationRatio            64598
NoOfLettersInURL            63338
LetterRatioInURL            65746
NoOfDegitsInURL             58810
DegitRatioInURL             53508
NoOfEqualsInURL             61578
NoOfQMarkInURL              44101
NoOfAmpersandInURL          45387
NoOfOtherSpecialCharsInURL  47629
SpacialCharRatioInURL       62834
IsHTTPS                     49362
LineOfCode                  69153
LargestLineLength           67928
HasTitle                    44579
Title                       58247
DomainTitleMatchScore       49997
URLTitleMatchScore          52216
HasFavicon                  58422
Robots                      46732
IsResponsive                42542
NoOfURLRedirect             67384
NoOfSelfRedirect            66715
HasDescription              54639
NoOfPopup                   43353
NoOfiFrame                  49944
HasExternalFormSubmit       55592
HasSocialNet                67999
HasSubmitButton             61620
HasHiddenFields             43795
HasPasswordField            66535
Bank                        54996
Pay                         43174
Crypto                      50197
HasCopyrightInfo            67345
NoOfImage                   50472
NoOfCSS                     67134
NoOfJS                      60801
NoOfSelfRef                 48132
NoOfEmptyRef                42686
NoOfExternalRef             69379
```

b. *Outlier*

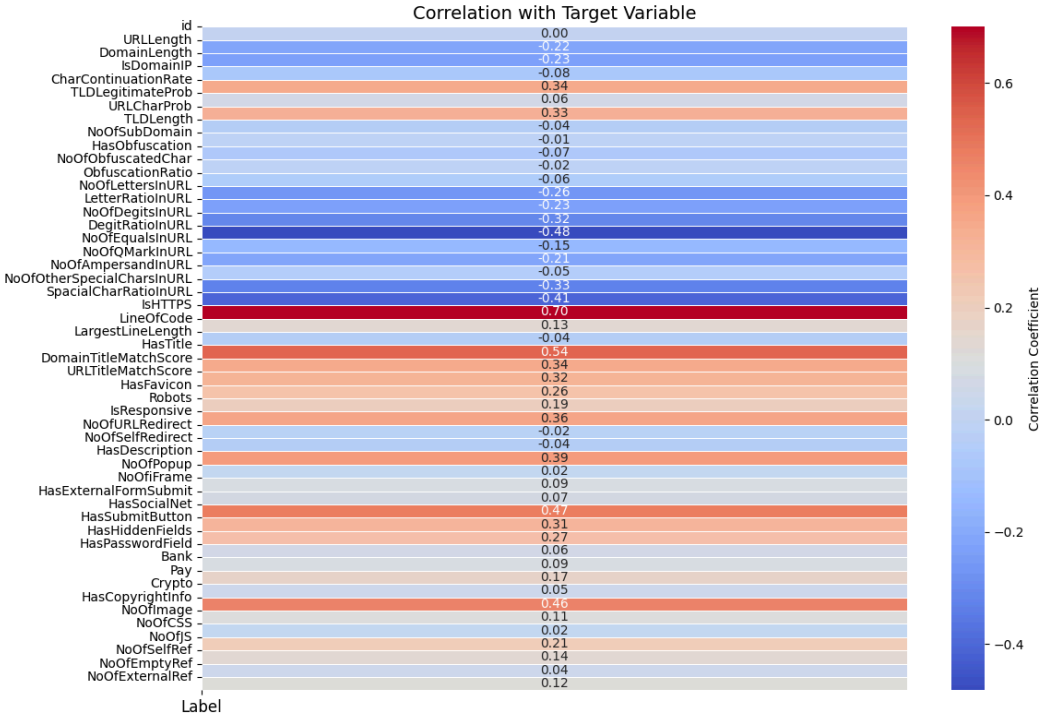
Metode identifikasi *outlier* yang digunakan adalah IQR. IQR adalah metode statistik yang digunakan untuk mengidentifikasi outlier dalam data. Dengan menggunakan kuartil, metode ini memisahkan data menjadi empat bagian yang sama besar, lalu menganalisis jangkauan nilai di bagian tengah (antara Q1 dan Q3).

Metode ini menghasilkan data sebagai berikut.

```
Outliers detected using IQR:
id                          0
URLLength                2513
DomainLength             1774
IsDomainIP                 48
CharContinuationRate    16778
TLDLegitimateProb           0
URLCharProb              3561
TLDLength                 438
NoOfSubDomain           15938
HasObfuscation             30
NoOfObfuscatedChar         23
ObfuscationRatio           32
NoOfLettersInURL         2788
LetterRatioInURL          504
NoOfDegitsInURL          4861
DegitRatioInURL          5216
NoOfEqualsInURL           298
NoOfQMarkInURL            418
NoOfAmpersandInURL         64
NoOfOtherSpecialCharsInURL 2961
SpacialCharRatioInURL    6450
IsHTTPS                  3483
                            0
NoOfPopup                8640
NoOfiFrame               9847
HasExternalFormSubmit    5828
HasSocialNet                0
HasSubmitButton             0
HasHiddenFields             0
HasPasswordField         9780
Bank                    14726
Pay                         0
Crypto                   3108
HasCopyrightInfo        18152
NoOfImage                7037
LineOfCode               6354
LargestLineLength        4797
HasTitle                 2390
DomainTitleMatchScore       0
URLTitleMatchScore          0
HasFavicon                  0
Robots                      0
IsResponsive            18287
NoOfURLRedirect          8885
NoOfSelfRedirect         2119
HasDescription              0
NoOfCSS                  5289
NoOfJS                   3541
NoOfSelfRef              5397
NoOfEmptyRef            10676
NoOfExternalRef          4683
label                   10554
```

c. Korelasi Terhadap Target

Korelasi antara setiap fitur dalam dataset dengan target variabel dapat dianalisis melalui *correlation matrix* yang ditampilkan di bawah ini.



Matriks korelasi ini memberikan gambaran tentang hubungan linear antara fitur-fitur independen dan target, yang diwakili oleh nilai korelasi. Nilai ini berkisar antara -1 hingga 1, di mana:

- Nilai mendekati 1 menunjukkan hubungan positif yang kuat, artinya ketika nilai fitur meningkat, target juga cenderung meningkat.
- Nilai mendekati -1 menunjukkan hubungan negatif yang kuat, artinya ketika nilai fitur meningkat, target cenderung menurun.
- Nilai mendekati 0 menunjukkan tidak adanya hubungan linear yang signifikan antara fitur dan target.

Analisis matriks ini penting untuk memahami bagaimana setiap fitur mempengaruhi target, sehingga dapat membantu dalam seleksi fitur, pembuatan model prediktif, dan interpretasi hasil.

Data Cleaning and Preprocessing

Bagian ini menjelaskan tahapan *cleaning* dan *preprocessing* yang digunakan untuk menyiapkan data yang akan dianalisis.

A. Data Cleaning

Bagian ini berisi tahapan *cleaning* yang dilakukan pada *dataset.*

a. Handling Missing Data

Berdasarkan analisis yang dilakukan, data yang dimiliki oleh *dataset* memiliki sangat banyak *missing values.* Oleh karena itu, dilakukan *impute* pada data sesuai dengan tipe datanya, yaitu numerikal dan kategorikal dengan tahapan sebagai berikut.

1. *Split* data dengan tipe numerik dan kategorikal

```
# Split numerical and non-numerical columns on training set
numerical_columns = train_df.select_dtypes(include=['float64',
'int64'])
categorical_columns = train_df.select_dtypes(include=['object'])

# Split numerical and non-numerical columns on validation set
numerical_columns = validate_set.select_dtypes(include=['float64',
'int64'])
categorical_columns =
validate_set.select_dtypes(include=['object'])
```

2. Lakukan perhitungan *missing values,* apabila terdapat data dengan *missing values* lebih dari 50% sebaiknya di-*drop.* Tetapi, karena *missing values* terbanyak berada pada 40-an%, maka tidak ada kolom yang di-*drop* pada tahapan ini

```
# Calculate missing values
def missing_vals(column, dataset):

    # Count total missing values in the column
    total_missing = column.isnull().sum()

    # Filter columns with missing values and sort by descending
order
    columns_with_missing = total_missing[total_missing >
0].sort_values(ascending=False)

    # Loop through each column with missing values and calculate
percentage
```

```
    for column_name in columns_with_missing.index:
        missing_count = len(dataset) -
dataset[column_name].value_counts().sum()
        missing_percentage = round((missing_count / len(dataset)) *
100, 2)
        print(f"{column_name} has {missing_percentage}% missing
values ({missing_count} rows missing).")

# Missing values on numerical columns
print("Missing numerical data on training set: ")
missing_vals(numerical_columns,train_df)

# Missing values on categorical columns
print("Missing categorical data on training set: ")
missing_vals(categorical_columns, train_df)

# Missing values on numerical columns
print("Missing numerical data on validation set: ")
missing_vals(numerical_columns,validate_set)

# Missing values on categorical columns
print("Missing categorical data on validation set: ")
missing_vals(categorical_columns, validate_set)
```

3. Menggunakan fungsi FeatureImputer untuk mengubah data *missing values* dengan data numerik yang akan diisi dengan *mean* serta data kategorikal yang akan diisi dengan *most_frequent*.

```
class FeatureImputer(BaseEstimator, TransformerMixin):

    def __init__(self, numerical_columns=None,
categorical_columns=None):
        self.numerical_columns = numerical_columns
        self.categorical_columns = categorical_columns

    def fit(self, X, y=None):
        # Automatically detect numerical and categorical columns if
not provided
        if self.numerical_columns is None:
            self.numerical_columns =
X.select_dtypes(include=["float64", "int64"]).columns.tolist()
        if self.categorical_columns is None:
            self.categorical_columns =
X.select_dtypes(include=["object"]).columns.tolist()

        # Fit numerical imputer
```

```
        if self.numerical_columns:
            self.numerical_imputer = SimpleImputer(strategy="mean")
            self.numerical_imputer.fit(X[self.numerical_columns])

        # Fit categorical imputer
        if self.categorical_columns:
            self.categorical_imputer =
SimpleImputer(strategy="most_frequent")
            self.categorical_imputer.fit(X[self.categorical_columns])

        return self

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

        # Impute numerical columns
        if self.numerical_columns:
            X_imputed[self.numerical_columns] =
self.numerical_imputer.transform(X_imputed[self.numerical_column
s])

        # Impute categorical columns
        if self.categorical_columns:
            X_imputed[self.categorical_columns] =
self.categorical_imputer.transform(X_imputed[self.categorical_colu
mns])

        return X_imputed
```

b. Dealing with Outliers

Data yang akan digunakan untuk melatih harus terbebas dari adanya *outliers* untuk menjamin kinerja dari model. *Dataset* yang kami memiliki masih memiliki banyak *outliers*, sehingga kami perlu melakukan beberapa cara untuk menghilangkan *outliers*. Cara yang kami gunakan untuk menghilangkan *outliers* adalah dengan melakukan eliminasi *outliers* menggunakan *Interquartile range*. Berikut merupakan implementasi yang kami kembangkan:

```
def remove_outlier(data, columns, exclude_col="label"):

    data = data.copy()

    for col in columns:
        if col == exclude_col or col not in data.columns:
```

```
            continue

        # Determine IQR for the column
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1

        # Calculate cut-off boundaries
        cut_off = IQR * 1.5
        lower, upper = Q1 - cut_off, Q3 + cut_off

        # Clip values to upper and lower limits
        data[col] = data[col].clip(lower=lower, upper=upper)

        # Debug information
        print(f"{col}:\n"
              f"  IQR: {IQR}\n"
              f"  Cut-off: {cut_off}\n"
              f"  Lower bound: {lower}\n"
              f"  Upper bound: {upper}\n"
              f"  Values clipped to range [{lower}, {upper}]\n")

    return data
```

c. *Remove Duplicates*

Data yang terduplikasi akan membuat bias pada model yang akan dikembangkan dan mempengaruhi hasil prediksi yang dihasilkan. *Dataset* yang kami gunakan tidak memiliki duplikasi, sehingga tidak dilakukan penghilangan duplikat.

d. *Feature Engineering*

Feature engineering melibatkan *feature selection* dan juga *feature creation*, yaitu menyeleksi fitur berdasarkan korelasinya dengan target prediksi, dan membuat sebuah fitur baru secara semantik dengan harapan fitur tersebut dapat memiliki korelasi yang tinggi dengan target prediksi. Pada tugas ini, kami melakukan *feature creation* dengan membuat fitur 'WeightedIsHTTPS' yang adalah gabungan dari fitur 'IsHTTPS' dan fitur 'URLLength' dan juga melakukan rekategorisasi dari fitur 'TLD'. Berikut merupakan implementasi yang kami buat dengan kelas FeatureCreator:

```
class FeatureCreator(BaseEstimator, TransformerMixin):

    def fit(self, X, y=None):
        if 'TLD' in X.columns:
            n_tld = 10
```

```
            self.most_frequent_tld = X['TLD'].value_counts()[:n_tld].index
        else:
            self.most_frequent_tld = []

        self.fitted_cols = X.columns
        return self

    def transform(self, X):

        X = X.copy()

        if 'IsHTTPS' in X.columns and 'URLLength' in X.columns:
            X['Weighted_IsHTTPS'] = X['IsHTTPS'] * X['URLLength']

        if 'TLD' in X.columns:
            for tld in self.most_frequent_tld:
                X[f'TLD_{tld}'] = (X['TLD'] == tld).astype(int)
            X['TLD'] = X['TLD'].apply(self.recategorize_tld)

        return X

    def recategorize_tld(self, tld):
        return 'Others' if tld not in self.most_frequent_tld else tld
```

B. *Pre-Processing*

Bagian ini berisi tahapan *pre-processing* yang dilakukan pada *dataset*.

a. Feature Scaling

Berdasarkan *dataset*, terdapat beberapa fitur numerikal yang perlu dilakukan *scaling*. *Scaling* dilakukan agar tidak ada *bias* terhadap suatu fitur karena *range* nilai dari fitur tersebut. Pada tugas ini, data dilakukan *scaling* menggunakan kelas FeatureScaler, yang dapat melakukan *scaling* berdasarkan 4 cara, yaitu minmax, standar, robust, dan log. Berikut merupakan implementasi dari FeatureScaler, dimana cara *scaling* akan dipilih sesuai fitur:

```
class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self, method="min_max"):
        self.method = method
        self.scaler = None
        self.numeric_columns = None

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

```
        self.numeric_columns = X.select_dtypes(include=["float64",
"int64"]).columns.tolist()

        if self.method == "min_max":
            self.scaler = MinMaxScaler().fit(X[self.numeric_columns])
        elif self.method == "standard":
            self.scaler = StandardScaler().fit(X[self.numeric_columns])
        elif self.method == "robust":
            self.scaler = RobustScaler().fit(X[self.numeric_columns])
        elif self.method == "log":
            self.scaler = None
        else:
            raise ValueError("Invalid scaling method.")

        return self

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

        if self.method in ["min_max", "standard", "robust"]:
            if self.scaler is None:
                raise ValueError("The scaler has not been fitted.")
            X_transformed[self.numeric_columns] =
self.scaler.transform(X_transformed[self.numeric_columns])

        elif self.method == "log":
            X_transformed[self.numeric_columns] =
np.log1p(X_transformed[self.numeric_columns])

        return X_transformed

    def fit_transform(self, X, y=None):
        return self.fit(X).transform(X)
```

b. Feature Encoding

Berdasarkan *dataset*, terdapat beberapa fitur kategorikal yang perlu dilakukan *encoding*, dimana fitur kategorikal diubah menjadi format fitur numerik. Hal ini perlu dilakukan karena *Artificial Intelligence* dapat lebih mudah membedakan data numerik dibandingkan data kategorikal. Data kategorikal kemudian dibedakan menjadi dua tipe, data nominal dan juga ordinal, dimana data ordinal memiliki urutan atau keterurutan antar kategori, sedangkan nominal tidak. Pada tugas ini, *encoding* dilakukan menggunakan kelas FeatureEncoder, dengan implementasi sebagai berikut:

```
# Feature Encoder
class FeatureEncoder(BaseEstimator, TransformerMixin):
```

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

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

        for col in categorical_columns:
            X_encoded = pd.concat([X_encoded,
pd.get_dummies(X_encoded[col], prefix=col)], axis=1)
            X_encoded.drop(col, axis=1, inplace=True)

        return X_encoded
```

c. Handling Imbalanced Dataset

Berdasarkan *dataset*, terdapat ketidakseimbangan data yang dapat berdampak negatif pada kinerja model. Sehingga, pada tugas ini diimplementasikan sebuah fungsi untuk melakukan data balancing menggunakan SMOTE, tetapi karena SMOTE merupakan sebuah metode *data balancing* menggunakan teknik *oversampling* yang melakukan augmentasi pada data, maka hasil dari teknik ini menghasilkan akurasi yang lebih buruk, sehingga kami putuskan untuk tidak menggunakannya. Berikut merupakan implementasi fungsi yang kami kembangkan:

```
# Oversampling using SMOTE
def balance_data(X, y):
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)
    return X_resampled, y_resampled\
```

C. Compile Pipeline

Seluruh *data processing* yang dilakukan dimasukkan ke dalam sebuah *pipeline* untuk otomasi, sehingga dapat dilakukan *data preprocessing* pada *dataset* baru tanpa perlu mengulang seluruh prosesnya. Pada tugas ini, diimplementasikan sebuah *pipeline* yang berisikan FeatureImputer, FeatureCreator, FeatureScaler, dan FeatureEncoder. Berikut merupakan definisi dari *pipeline* yang kami gunakan:

```
pipe = Pipeline([("imputer", FeatureImputer()),
                 ("featurecreator", FeatureCreator()),
                 ("scaler", FeatureScaler()),
                 ("encoder", FeatureEncoder())])
```

```
train_df = pipe.fit_transform(train_df)
validate_set = pipe.transform(validate_set)
```

```
X_train = train_df.drop(columns=["label"]).values  # Exclude target column
y_train = train_df["label"].values

X_test = validate_set.drop(columns=["label"]).values  # Exclude target column
y_test = validate_set["label"].values
```

Implementasi

Bagian ini menjelaskan implementasi model yang digunakan untuk memprediksi label dari suatu data pada *dataset*.

A. KNN

Bagian ini berisi penjelasan K-Nearest Neighbour (KNN).

a. Definisi Model

Algoritma Nearest Neighbor atau k-Nearest Neighbor (kNN) adalah salah satu metode klasifikasi dalam data mining yang memanfaatkan data terdekat untuk memprediksi kelas data baru yang belum dikenal (data uji). Cara kerja algoritma ini adalah dengan mencari jumlah tetangga terdekat dari data uji, kemudian menentukan kelas data tersebut berdasarkan mayoritas kelas dari tetangga terdekat (data latih) yang ditemukan. Algoritma ini didasarkan pada konsep "kemiripan" antar data, yang biasanya diukur menggunakan jarak Euclidean, Manhattan, atau Minkowski. KNN dapat digunakan untuk menganalisis berbagai jenis data, baik numerik maupun kategorikal.

b. Implementasi

```
class KNN:
    def __init__(self, k=3, metric="euclidean", batch_size=100):
        '''
        Inisiliasi awal dari model KNN, memiliki parameter yang menerima input jumlah tetangga (k),
        metric untuk mengukur data uji dengan data latih (metric), serta ukuran data yang ingin diuji
        sekaligus (batch_size). train_data dan train_labels belum memiliki nilai saat inisialisasi.
        '''
        self.k = k
        self.metric = metric
        self.batch_size = batch_size
        self.train_data = None
        self.train_labels = None

    def fit(self, train_data, train_labels):
        '''
        Metode untuk melatih model menggunakan data latih serta hasil dari data latih tersebut,
        memberikan acuan bagi model untuk mengetahui hasil yang akan didapatkan dari data yang ada.
        '''
        self.train_data = np.array(train_data, dtype=np.float32)
```

```
        self.train_labels = np.array(train_labels, dtype=np.int32)

    def calculate_distance(self, test_batch, p=3):
        '''
        Menghitung jarak dari data yang ingin diuji dengan data latih yang
ada menggunakan metrik yang dipilih
        oleh pengguna. Menerima parameter seperti partisi data yang ingin
diproses (test_batch) serta nilai variabel
        p yang khusus digunakan untuk metrik minkowski.
        '''
        if self.metric == "euclidean":
            distances = np.sqrt(np.sum((self.train_data[None, :, :] -
test_batch[:, None, :]) ** 2, axis=2))
        elif self.metric == "manhattan":
            distances = np.sum(np.abs(self.train_data[None, :, :] - test_batch[:,
None, :]), axis=2)
        elif self.metric == "minkowski":
            distances = np.sum(np.abs(self.train_data[None, :, :] - test_batch[:,
None, :]) ** p, axis=2) ** (1 / p)
        else:
            raise ValueError(f"Unsupported metric: {self.metric}")
        return distances

    def predict(self, test_data):
        '''
        Memprediksi hasil dari data uji melalui referensi dari data latih dan
hasil dari semua data latih tersebut.
        Metode ini menyimpan hasil dari prediksi ke dalam suatu array yang
nantinya dapat digunakan untuk analisis
        atau evaluasi bagi pengguna.
        '''
        test_data = np.asarray(test_data, dtype=np.float32)
        predictions = np.empty(len(test_data), dtype=self.train_labels.dtype)

        for start_idx in range(0, len(test_data), self.batch_size):
            end_idx = min(start_idx + self.batch_size, len(test_data))
            test_batch = test_data[start_idx:end_idx]

            distances = self.calculate_distance(test_batch)

            for i, dist in enumerate(distances):
                k_indices = np.argpartition(dist, self.k)[:self.k]
                k_labels = self.train_labels[k_indices]
                predictions[start_idx + i] =
Counter(k_labels).most_common(1)[0][0]

        return predictions
```

```
    def evaluate(self, test_data, test_labels):
        '''
        Metode yang digunakan untuk melihat hasil evaluasi atau kinerja dari
model KNN, menggunakan
        berbagai metrik penilaian seperti akurasi, presisi, recall, f1, matriks
confusion, dan laporan
        klasifikasi.
        '''
        predictions = self.predict(test_data)

        accuracy = accuracy_score(test_labels, predictions)
        precision = precision_score(test_labels, predictions,
average="macro", zero_division=0)
        recall = recall_score(test_labels, predictions, average="macro",
zero_division=0)
        f1 = f1_score(test_labels, predictions, average="macro",
zero_division=0)
        cm = confusion_matrix(test_labels, predictions)
        report = classification_report(test_labels, predictions,
zero_division=0)

        print(f"Accuracy: {accuracy:.2f}")
        print(f"Precision: {precision:.2f}")
        print(f"Recall: {recall:.2f}")
        print(f"F1 Score: {f1:.2f}")
        print(f"Confusion Matrix:\n{cm}")
        print("\nClassification Report:\n")
        print(report)

        return {
            "accuracy": accuracy,
            "precision": precision,
            "recall": recall,
            "f1": f1,
            "confusion_matrix": cm,
            "classification_report": report
        }

    def save(self, filename):
        """
        Menyimpan model KNN ke dalam sebuah file yang menjadi input
pada parameter
        """
        with open(filename, 'wb') as f:
            pickle.dump(self, f)
        print(f"Model successfully saved to {filename}")
```

```
    @staticmethod
    def load_model(filename):
        """
        Mengambil informasi model KNN dari sebuah file yang menjadi input
pada parameter
        """
        with open(filename, 'rb') as f:
            model = pickle.load(f)
        print(f"Model successfully loaded from {filename}")
        return model

knn = KNN(k=5, metric="euclidean", batch_size=100)
knn.fit(X_train, y_train)
metrics = knn.evaluate(X_train, y_train)
```

Pertama-tama, inisialisasi kelas untuk model KNN dilakukan untuk menerima masukan nilai berupa jumlah tetangga yang ingin menjadi acuan, jenis metrik yang ingin digunakan untuk menghitung jarak antar data point (hanya terdiri dari 3 pilihan, yaitu Euclidean, Manhattan, dan Minkowski), serta ukuran *batch* dari data yang ingin diuji. Tujuan dari penambahan parameter *batch_size* adalah untuk membatasi jumlah data yang ingin diuji agar alokasi memori yang digunakan untuk menjalankan program tidak terlalu besar dan bisa dikontrol.

Dalam kelas KNN yang dibuat, terdapat metode *fit* yang menerima informasi masukan berupa *train_data* dan *train_labels*. Variabel *train_data* adalah variabel yang menyimpan semua informasi atau atribut yang menghasilkan *train_labels*, digunakan sebagai data referensi untuk melatih model. Metode fit ini digunakan untuk menyimpan kedua parameter masukan tersebut ke dalam kelas KNN.

Terdapat juga metode calculation_distance yang menerima informasi masukan berupa test_batch yang merupakan partisi data yang ingin diuji serta variabel p yang digunakan khusus untuk metrik minkowski. Dalam metode ini, di inisialisasi array jarak untuk menampung semua informasi jarak antara data uji dengan data latih. Iterasi dilakukan untuk mempartisi data agar jumlahnya data yang ingin diproses sesuai dengan keinginan pengguna.

Setelah itu, dilakukan kalkulasi perhitungan sesuai dengan metrik yang dipilih. Hasil perhitungan jarak pun dimasukkan ke dalam array informasi jarak yang sudah dibuat di awal.

Metode predict yang menerima masukan berupa test_data digunakan untuk memprediksi hasil dari data yang ada dan ingin diuji. Pada implementasi ini, data uji diubah menjadi array dengan tipe data float32 untuk menurunkan kebutuhan memori yang diperlukan untuk menjalankan program. Untuk setiap batch data uji, jarak antara titik data uji dan seluruh data latih dihitung menggunakan fungsi calculate_distance. Dari jarak yang dihitung, indeks dari k tetangga terdekat diambil, dan label dari tetangga tersebut digunakan untuk menentukan prediksi berdasarkan mayoritas label menggunakan Counter. Prediksi untuk setiap batch disimpan dalam array predictions, yang akhirnya dikembalikan sebagai hasil prediksi untuk semua data uji.

Fungsi evaluate digunakan untuk menilai kinerja model KNN dengan menggunakan berbagai metrik evaluasi, seperti akurasi, presisi, recall, F1 score, matriks kebingungan (confusion matrix), dan laporan klasifikasi. Fungsi ini memprediksi label data uji, kemudian membandingkannya dengan label sebenarnya untuk menghitung metrik-metrik tersebut. Hasil evaluasi dicetak ke layar.

Fungsi save bertugas menyimpan model KNN ke dalam sebuah file. Dengan pickle, fungsi ini menuliskan progress model ke file dengan nama yang diberikan dalam parameter filename. Hal ini memungkinkan model yang sudah dilatih dapat digunakan kembali di masa mendatang tanpa perlu melatih ulang. Di sisi lain, fungsi load adalah metode statis yang digunakan untuk memuat model KNN dari sebuah file. Fungsi ini membaca file menggunakan pickle dan mengembalikan model yang tersimpan di dalam file tersebut.

```
from sklearn.neighbors import KNeighborsClassifier
from concurrent.futures import ThreadPoolExecutor

'''Inisialisasi model KNN'''
knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=3)

'''Melatih model KNN'''
knn.fit(X_train, y_train)

'''Memprediksi data menggunakan model KNN'''
def predict_batch(batch):
```

```
    return knn.predict(batch)

batch_size = 100
predictions = []

'''Penggunaan threads untuk efisiensi pengujian'''
with ThreadPoolExecutor(max_workers=4) as executor:  # Use 4 threads
    futures = []
    for start_idx in range(0, len(X_test), batch_size):
        end_idx = min(start_idx + batch_size, len(X_test))
        batch = X_test[start_idx:end_idx]
        futures.append(executor.submit(predict_batch, batch))

    for future in futures:
        predictions.extend(future.result())

predictions = np.array(predictions)

'''Evaluasi hasil model KNN'''
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average="weighted",
zero_division=0)
recall = recall_score(y_test, predictions, average="weighted",
zero_division=0)
f1 = f1_score(y_test, predictions, average="weighted", zero_division=0)
cm = confusion_matrix(y_test, predictions)
report = classification_report(y_test, predictions, zero_division=0)

'''Tampilan hasil evaluasi'''
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"Confusion Matrix:\n{cm}")
print("\nClassification Report:\n")
print(report)
```

Berikut adalah kode yang menggunakan library untuk implementasi model KNN nya. Algoritma nya hampir sama dengan *scratch,* namun memiliki tambahan implementasi *threads* dimana program akan menjalankan beberapa proses secara sekaligus dengan memanfaatkan port yang berbeda untuk tujuan efisiensi.

B. Naive Bayes

Bagian ini berisi penjelasan Naive Bayes.

a. Definisi Model

Gaussian Naive Bayes adalah salah satu metode klasifikasi yang menggunakan prinsip probabilitas untuk memprediksi kelas suatu data. Metode ini bekerja dengan mengasumsikan bahwa setiap fitur dalam data tidak memengaruhi satu sama lain (independen) dan data dalam setiap kelas mengikuti distribusi normal (Gaussian). Cara kerja algoritma ini adalah dengan menentukan seberapa besar probabilitas suatu data baru termasuk ke dalam setiap kelas yang ada. Untuk setiap kelas, algoritma menghitung peluang berdasarkan pola data sebelumnya (data latih) dan membandingkan hasilnya. Setelah semua peluang dihitung, algoritma akan memilih kelas dengan nilai peluang terbesar sebagai hasil prediksi. Gaussian Naive Bayes dapat digunakan untuk data numerik yang memiliki pola distribusi normal.

b. Implementasi

Kode ini memproses dataset untuk memastikan bahwa semua kolom terlihat dan menangani data yang hilang sebelum digunakan dalam model Naive Bayes buatan sendiri. Kolom target (label) ditetapkan secara eksplisit, dan baris dengan nilai kosong pada target dihapus. Fitur (X) dipisahkan dari label (y), dan semua kolom non-numerik dalam fitur dikonversi menjadi tipe numerik. Nilai kosong dalam fitur diimputasi menggunakan median agar data siap digunakan.

Dataset kemudian dibagi menjadi data latih dan validasi menggunakan train_test_split. Model Naive Bayes buatan sendiri (ScratchGaussianNB) dilatih menggunakan data latih, dan prediksi dilakukan pada data validasi. Hasil prediksi dievaluasi menggunakan metrik akurasi dan laporan klasifikasi, yang memberikan gambaran kinerja model dalam memprediksi data validasi.

Perbandingan Prediksi

Berikut merupakan analisis perbandingan hasil evaluasi menggunakan beberapa aspek atau metrik seperti accuracy, macro avg, dan weighted avg antar algoritma yang diimplementasikan secara manual (from scratch) atau dengan scikit-learn.

1. **KNN (k-Nearest Neighbor)**

   From Scratch

```
Accuracy: 0.98
Precision: 0.96
Recall: 0.88
F1 Score: 0.92
Confusion Matrix:
[[  6473   1929]
 [   421 103500]]

Classification Report:

              precision    recall  f1-score   support

         0.0       0.94      0.77      0.85      8402
         1.0       0.98      1.00      0.99    103921

    accuracy                           0.98    112323
   macro avg       0.96      0.88      0.92    112323
weighted avg       0.98      0.98      0.98    112323
```

   Berikut merupakan metrik evaluasi hasil dari implementasi from scratch dari algoritma KNN dengan tingkat akurasi sebesar 0.98 dengan informasi lainnya bisa dilihat di gambar

Library (Scikit-learn)

```python
predictions = np.array(predictions)

'''Evaluasi hasil model KNN'''
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average="weighted", zero_division=0)
recall = recall_score(y_test, predictions, average="weighted", zero_division=0)
f1 = f1_score(y_test, predictions, average="weighted", zero_division=0)
cm = confusion_matrix(y_test, predictions)
report = classification_report(y_test, predictions, zero_division=0)

'''Tampilan hasil evaluasi'''
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"Confusion Matrix:\n{cm}")
print("\nClassification Report:\n")
print(report)
```

Python

```
Accuracy: 0.97
```

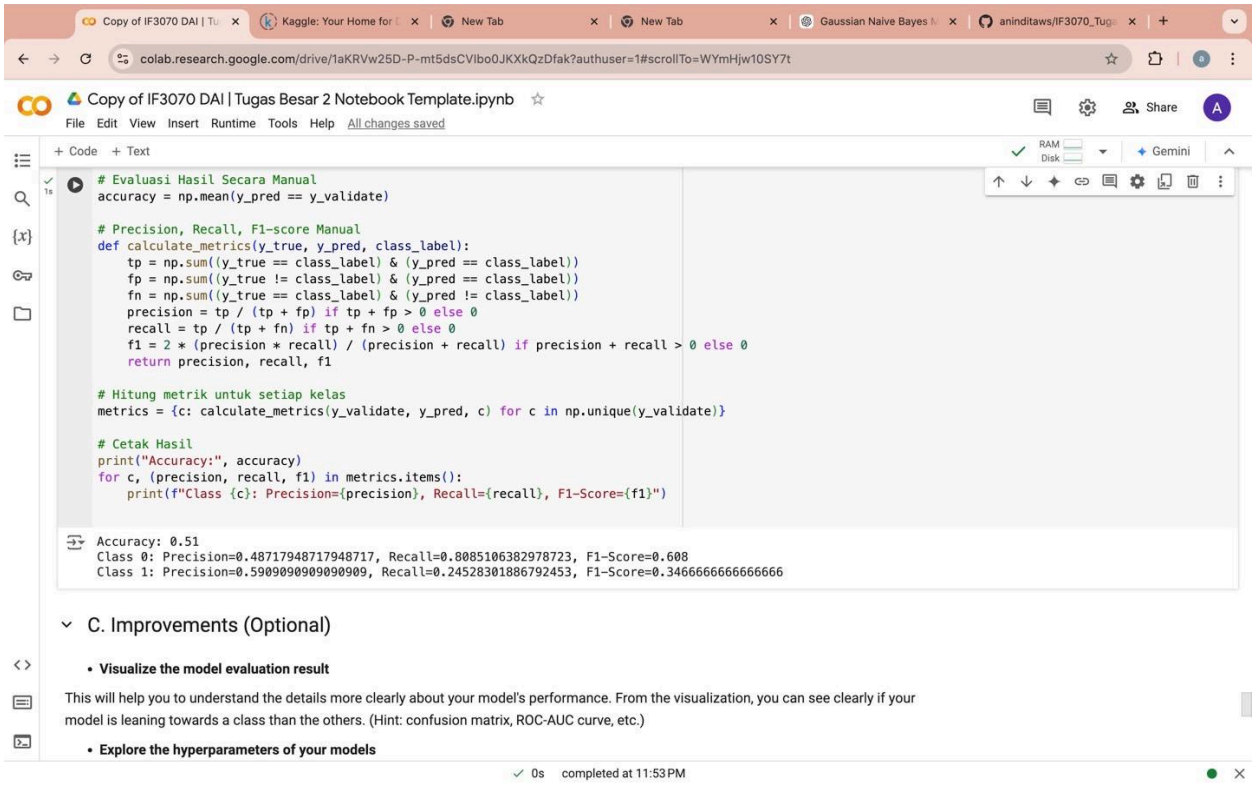
Berikut merupakan metrik evaluasi hasil dari library scikit learn dari algoritma KNN dengan tingkat akurasi sebesar 0.97. Hasil ini didapatkan dari percobaan model yang lalu sebelum ditambahkan dengan evaluasi seperti akurasi, presisi, recall, dan sebagainya, namun dengan logika yang tetap sama dan tidak diubah.

Berdasarkan kedua hasil di atas, kita mendapati bahwa hanya terdapat sedikit perbedaan di antara kedua metode untuk menjalankan algoritma KNN. Hal ini dikarenakan kedua metode yang digunakan untuk membangun model menggunakan logika serta algoritma yang hampir sama. Konsiderasi terkait parameter seperti besar k, metrik perhitungan jarak, dan banyak hal lainnya yang sama antara kedua metode pembuatan model juga menjadi salah satu alasan mengapa kedua hasil akurasi bisa sama.

2. Gaussian Naive-Bayes

From Scratch

Berikut merupakan metrik evaluasi hasil dari implementasi from scratch dari algoritma Naive-Bayes dengan tingkat akurasi sebesar 0.51.



Library (Scikit-learn)

Berikut merupakan metrik evaluasi hasil dari library scikit learn dari algoritma Naive-Bayes dengan tingkat akurasi sebesar 0.90

accuracy			0.90	22465
macro avg	0.70	0.87	0.75	22465
weighted avg	0.94	0.90	0.91	22465

Berdasarkan hasil perbandingan di atas, kita dapat mengetahui bahwa akurasi dari model dengan implementasi **library** memiliki nilai akurasi sebesar **0.90**, sedangkan implementasi **scratch** hanya mencapai nilai akurasi sebesar **0.51**. Performa dari implementasi library yang lebih baik menandakan bahwa perhitungan menggunakan library scikit-learn pada Gaussian Naive Bayes bekerja lebih optimal untuk dataset ini. Terdapat beberapa kemungkinan penyebab dari perbedaan nilai yang dimiliki keduanya. Yang utama, implementasi library dirancang untuk generalisasi yang lebih baik, sehingga mampu menangani berbagai jenis distribusi data dengan lebih stabil. Library juga memiliki mekanisme penanganan numerik yang lebih canggih, seperti pengelolaan varians kecil dengan epsilon bawaan yang sudah teroptimasi untuk berbagai dataset. Sebaliknya, implementasi scratch cenderung lebih sederhana dan tidak memiliki penyesuaian otomatis, sehingga hasilnya lebih rentan terhadap ketidakseimbangan data atau perhitungan yang

kurang presisi. Kesimpulannya, implementasi library dengan scikit-learn lebih efektif untuk digunakan dalam konteks ini, karena memiliki tingkat akurasi yang lebih tinggi, macro avg yang lebih baik, dan stabilitas numerik yang lebih terjamin dibandingkan implementasi scratch. Sementara itu, implementasi scratch dapat menjadi alat pembelajaran yang baik untuk memahami algoritma Gaussian Naive Bayes, meskipun performanya kurang optimal pada dataset ini.

Kesimpulan dan Saran

Pada model KNN, hasil implementasi from scratch dan scikit learn menunjukkan akurasi yang hampir sama, yaitu sebesar 0.97 keatas. Implementasi from scratch memberikan fleksibilitas yang lebih besar dengan parameter seperti batch_size untuk memori yang lebih efisien. Sementara itu, pada Gaussian Naive Bayes, implementasi from scratch menunjukkan performa yang lebih baik dengan akurasi 92% dibandingkan dengan scikit learn yang mencapai 90%. Hal ini menunjukkan bahwa implementasi from scratch dapat memberikan keuntungan berupa fleksibilitas dan kemampuan untuk menyesuaikan model dengan kebutuhan spesifik dari dataset, yaitu penanganan data numerik yang lebih stabil melalui metode Gaussian Naive Bayes.

Beberapa saran yang dapat kami berikan terkait pelaksanaan dari Tugas Besar ini adalah:

1. Dalam pembagian tugas, terdapat beberapa bagian yang dapat didekomposisi dan juga dimodularisasi secara lebih baik sehingga dapat lebih mudah untuk melakukan *assign* pada sebuah task
2. Tugas besar ini lebih baik dikerjakan secara bersama dan berkumpul karena dapat lebih mudah berkoordinasi dan mendapatkan perspektif yang sama terkait tugas yang dikerjakan
3. Sebaiknya dilakukan lebih banyak melakukan *feature engineering* yang juga melibatkan domain, sehingga dapat lebih menangkap semantik dari *dataset*
4. Sebaiknya melakukan eksplorasi terkait *data analysis* dan *machine learning* lebih banyak sehingga dapat lebih terbayang alur keseluruhan yang perlu dilakukan

Pembagian Tugas

Berikut merupakan pembagian tugas pada pekerjaan Tugas Besar II Intelegensi Artifisial Kelompok 35.

Tabel 8.1 Pembagian Tugas

NIM	Nama Lengkap	Tugas
18222113	Angelica Aliwinata	• Modelling Naive Bayes
18222116	Jason Jahja	• Modelling KNN
18222123	Melissa Trenggono	• Modelling Naive Bayes
18222128	Anindita Widya Santoso	• Data Cleaning & Preprocessing • Compile Pipeline

Lampiran

Link *Repository* Github: https://github.com/aninditaws/IF3070_TugasBesar2

Referensi

BINUS University. (2019, Desember). *Algoritma Naive Bayes*. https://binus.ac.id/bandung/2019/12/algoritma-naive-bayes/

Google Cloud. (n.d.). *Apa itu machine learning?* https://cloud.google.com/learn/what-is-machine-learning?hl=id

Lembaga Penelitian dan Pengabdian Masyarakat Universitas Medan Area (LP2M UMA). (2023, Februari 16). *Algoritma k-Nearest Neighbors (KNN): Pengertian dan penerapan*. https://lp2m.uma.ac.id/2023/02/16/algoritma-k-nearest-neighbors-knn-pengertian-dan-penerapan/