TUGAS BESAR 2
IF3070 – DASAR INTELIGENSI ARTIFISIAL

Algoritma *Machine Learning* Menggunakan KNN dan Naive Bayes

Disusun Oleh Kelompok 8:

Daffari Adiyatma - 18222003 (K01)

Aththariq Lisan Q. D. S. - 18222013 (K01)

Muhammad Rafly - 18222067 (K01)

Alvin Fadhilah Akmal - 18222079 (K01)

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB I
DESKRIPSI PERSOALAN

PhiUSIIL Phishing URL Dataset merupakan sebuah dataset berisi URL phishing dan URL legitimate yang bertujuan untuk penelitian. Pada tugas besar kali ini diberikan dataset dari PhiUSIIL tersebut yang sudah dimodifikasi untuk pembelajaran *machine learning*.

Machine learning sendiri merupakan salah satu cabang dari AI yang memungkinkan sistem untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit. Pada tugas kali ini, kami diminta untuk memprediksi legitimasi URL berdasarkan label menggunakan KNN dan Gaussian Naive Bayes. Untuk spesifikasi lengkapnya adalah sebagai berikut:

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).
5. [Bonus] Kaggle Submission pada link berikut.

BAB II
IMPLEMENTASI KNN

K-nearest neighbor adalah algoritma *surpervised learning* yang menggunakan kedekatan data untuk membuat klasifikasi atau prediksi. Implementasi algoritma KNN yang kami gunakan dalam bentuk kelas python.

```
class KNearestNeighbors:
    def __init__(self, k=3, metric='euclidean', p=2):
        """
        :param k: Jumlah tetangga terdekat.
        :param metric: Metrik jarak yang digunakan ('euclidean',
'manhattan', 'minkowski').
        :param p: Parameter untuk Minkowski distance (default
p=2 untuk Euclidean).
        """
        self.k = k
        self.metric = metric
        self.p = p

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
```

Kelas KNN memiliki atribut k (jumlah tetangga terdekat yang dikonsiderasi), metric (metode pengukuran jarak), p (parameter *minkowski distance*), x_train (*input* data latih) , dan y_train (*output* data latih).

```
    def predict(self, X):
        X = np.array(X)
        predictions = [self._predict_one(x) for x in X]
        return np.array(predictions)
```

Metode predict digunakan untuk melakukan prediksi suatu array data di mana setiap data akan dilakukan prediksi dengan metode _predict_one dan hasilnya disimpan dalam suatu array.

```
    def _predict_one(self, x):
        # Hitung jarak berdasarkan metrik yang dipilih
        if self.metric == 'euclidean':
            distances = np.sqrt(np.sum((self.X_train - x) ** 2,
axis=1))
        elif self.metric == 'manhattan':
            distances = np.sum(np.abs(self.X_train - x), axis=1)
        elif self.metric == 'minkowski':
            distances = np.sum(np.abs(self.X_train - x) **
self.p, axis=1) ** (1 / self.p)
        else:
            raise ValueError("Metrik tidak valid. Gunakan
'euclidean', 'manhattan', atau 'minkowski'.")

        # Ambil k tetangga terdekat
        k_indices = np.argsort(distances)[:self.k]
        k_labels = self.y_train[k_indices]
        # Voting mayoritas
        most_common = Counter(k_labels).most_common(1)
        return most_common[0][0]
```

Metode _predict_one akan melakukan perhitungan jarak suatu data dengan setiap data pada data latih berdasarkan metrik yang sudah ditentukan model pada instansiasi. Array hasil perhitungan jarak akan diurut dari paling kecil. Seteelah disimpan k jumlah tetangga terdekat, akan dicari hasil yang paling banyak (mode), dan mengembalikan hasil tersebut.

Cara pemakaian fungsi sebagai berikut.

```
# Latih model menggunakan euclidean
knn_scratch_euclidean = KNearestNeighbors(k=5,
metric='euclidean')
knn_scratch_euclidean.fit(X_train_resampled, y_train_resampled)
y_pred_knn_scratch_euclidean =
knn_scratch_euclidean.predict(X_val_preprocessed)

# Latih model menggunakan manhattan
knn_scratch_manhattan = KNearestNeighbors(k=5,
metric='manhattan')
knn_scratch_manhattan.fit(X_train_resampled, y_train_resampled)
y_pred_knn_scratch_manhattan =
knn_scratch_manhattan.predict(X_val_preprocessed)

#menggunakan Minkowski dengan p=3
knn_scratch_minkowski = KNearestNeighbors(k=5,
metric='minkowski', p=3)
knn_scratch_minkowski.fit(X_train_resampled, y_train_resampled)
y_pred_knn_scratch_minkowski =
knn_scratch_minkowski.predict(X_val_preprocessed)
```

BAB III
IMPLEMENTASI NAIVE-BAYES

Kelas ini terdiri dari dua fungsi utama, yaitu `fit` untuk melatih model dan `predict` untuk melakukan prediksi. Ada pula fungsi privat `_predict_one` untuk menghitung probabilitas posterior bagi satu sampel.

```
class NaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.class_priors = {cls: np.mean(y == cls) for cls in
self.classes}
        self.class_means = {cls: X[y == cls].mean(axis=0) for
cls in self.classes}
        self.class_variances = {cls: X[y == cls].var(axis=0) +
1e-9 for cls in self.classes}
```

Fungsi fit melatih model dengan menghitung parameter distribusi Gaussian untuk setiap kelas dalam data latih.

- Parameter classes kelas unik dari label `y` menggunakan `np.unique(y)`.
- Parameter class_priors menghitung probabilitas prior untuk setiap kelas.
- Parameter class_means menghitung rata-rata (mean) dari setiap fitur untuk setiap kelas.
- Parameter class_variances menghitung varians dari setiap fitur untuk setiap kelas. Untuk variansi sendiri dijumlahkan dengan 1e-9 agar terhindar dari pembagian dengan 0 yang dapat mengurangi performa model.

```
  def predict(self, X):
        X = np.array(X)
        predictions = [self._predict_one(x) for x in X]
        return np.array(predictions)
```

Fungsi predict ini digunakan untuk melakukan prediksi pada data baru. Pertama akan mengiterasi setiap sampel dalam `X` dan menggunakan fungsi `_predict_one` untuk menghitung probabilitas posterior setiap kelas. Akhirnya algoritma akan memilih kelas dengan probabilitas tertinggi dan mengembalikannya dalam array.

```
    def _predict_one(self, x):
        posteriors = []
        for cls in self.classes:
            prior = np.log(self.class_priors[cls])
            likelihood = -0.5 * np.sum(np.log(2 * np.pi *
self.class_variances[cls]))
            likelihood -= 0.5 * np.sum(((x -
self.class_means[cls]) ** 2) / (self.class_variances[cls] +
1e-9))
            posteriors.append(prior + likelihood)
        return self.classes[np.argmax(posteriors)]
```

Fungsi _predict_one(self, x) menghitung probabilitas posterior untuk satu sampel `x`.

Melatih Model

```
nb_scratch = NaiveBayes()
nb_scratch.fit(X_train_resampled, y_train_resampled)
```

Fungsi fit melatih model menggunakan data `X_train_resampled` (fitur) dan `y_train_resampled` (label) serta menghitung prior, mean, dan varians untuk setiap kelas.

Cara pemakaian fungsi sebagai berikut.

```
y_pred_nb_scratch = nb_scratch.predict(X_val_preprocessed)
```

BAB IV
TAHAP CLEANING DAN PREPROCESSING

Pada tahap ini kami menggunakan sampel berupa 20% data dari dataset asli. Pengambilan sampel dilakukan karena penggunaan sampel dapat membuat komputasi lebih cepat. Penggunaan sampel mengurangi jumlah data yang harus di komputasikan dan. Data sampel juga masih cukup representatif untuk menggambarkan distribusi data. Pengambilan sampel diimplementasikan sebagai berikut

```
df_sample = df.sample(frac=0.2, random_state=42)
df_sample.shape
```

A. Cleaning

a. Handling Missing Value

```
#Memeriksa Missing Value
missing_count = train_set.isnull().sum()

missing_percentage = (missing_count / len(train_set)) * 100

missing_data = pd.DataFrame({
    'Missing Count': missing_count,
    'Percentage': missing_percentage
})

missing_data_sorted = missing_data.sort_values(by='Missing Count', ascending=False)

print(missing_data_sorted[missing_data_sorted['Missing Count'] > 0])
```

Implementasi kode handling missing value yang pertama adalah kode menganalisis jumlah dan persentase missing values dalam dataset menggunakan isnull().sum(). Ini dilakukan agar kita mengetahui seberapa banyak data yang hilang dan menyusun strategi penanganannya.

```
#Drop Kolom yang missing valuenya di atas 49%
columns_to_drop = ['LineOfCode', 'NoOfExternalRef', 'Domain']

train_set.drop(columns=columns_to_drop, axis=1, inplace=True)
val_set.drop(columns=columns_to_drop, axis=1, inplace=True)

print("Columns dropped:", columns_to_drop)
```

Selanjutnya, kode menghapus kolom-kolom yang memiliki missing values di atas 49% ('LineOfCode', 'NoOfExternalRef', 'Domain'). Alasannya sederhana yaitu kolom dengan terlalu banyak missing values bisa mengurangi kualitas model karena terlalu banyak data yang harus diimputasi.

```
#Buat Imputasi Nanti
# Identifikasi kolom numerik
numerical_columns = train_set.select_dtypes(include=['float64', 'int64']).columns

# Identifikasi kolom biner
binary_columns = [col for col in numerical_columns if set(train_set[col].dropna().unique()) <= {0, 1}]

# Identifikasi kolom non-biner
non_binary_columns = [col for col in numerical_columns if col not in binary_columns]

# Hitung skewness untuk kolom numerik
skewness = train_set[numerical_columns].skew()
print(skewness)

# Klasifikasiin kolom berdasarkan skewness
mean_imputation_columns = skewness[skewness.abs() < 0.5].index.tolist()
median_imputation_columns = skewness[skewness.abs() >= 0.5].index.tolist()

print("Kolom dengan imputasi mean:", mean_imputation_columns)
print("Kolom dengan imputasi median:", median_imputation_columns)

# Imputasi untuk kolom biner
binary_imputer = SimpleImputer(strategy='most_frequent')
train_set[binary_columns] = binary_imputer.fit_transform(train_set[binary_columns])

# Imputasi mean untuk kolom dengan skewness rendah
mean_imputer = SimpleImputer(strategy='mean')
train_set[mean_imputation_columns] = mean_imputer.fit_transform(train_set[mean_imputation_columns])

# Imputasi median untuk kolom dengan skewness tinggi
median_imputer = SimpleImputer(strategy='median')
train_set[median_imputation_columns] = median_imputer.fit_transform(train_set[median_imputation_columns])

# Terapkan transformasi ke validation set menggunakan parameter dari train set
val_set[binary_columns] = binary_imputer.transform(val_set[binary_columns])
val_set[mean_imputation_columns] = mean_imputer.transform(val_set[mean_imputation_columns])
val_set[median_imputation_columns] = median_imputer.transform(val_set[median_imputation_columns])
```

Kode kemudian mengidentifikasi dan memisahkan kolom-kolom berdasarkan jenisnya:

- Kolom numerik (float64 dan int64)
- Kolom biner (hanya berisi 0 dan 1)
- Kolom non-biner (numerik selain biner)

Untuk imputasi, strategi berbeda digunakan menyesuaikan karakteristik data:

- Kolom biner diimputasi menggunakan most_frequent (modus)
- Kolom numerik dengan skewness < 0.5 diimputasi menggunakan mean
- Kolom numerik dengan skewness ≥ 0.5 diimputasi menggunakan median

Pendekatan ini diterapkan karena:

- Mean cocok untuk data yang terdistribusi normal (skewness rendah)
- Median lebih robust untuk data yang skewed (menceng)
- Most frequent cocok untuk data biner karena mempertahankan proporsi kelas yang dominan

```
#Drop kolom kategorikal karena tidak akan diimputasi
categorical_columns = train_set.select_dtypes(include=['object']).columns

train_set.drop(columns=categorical_columns, axis=1, inplace=True)
val_set.drop(columns=categorical_columns, axis=1, inplace=True)

print("Columns dropped (categorical):", categorical_columns.tolist())
```

Kolom kategorikal dihapus karena tidak akan diimputasi, mengingat imputasi pada data kategorikal bisa menghasilkan bias jika tidak dilakukan

dengan hati-hati.

```
#Cek missing value lagi setelah imputasi
missing_train = train_set.isnull().sum()
missing_train_percentage = (missing_train / len(train_set)) * 100

print("Missing values di Train Set setelah imputasi:")
print(pd.DataFrame({
    'Missing Count': missing_train[missing_train > 0],
    'Percentage': missing_train_percentage[missing_train > 0]
}))

missing_val = val_set.isnull().sum()
missing_val_percentage = (missing_val / len(val_set)) * 100

print("\nMissing values di Validation Set setelah imputasi:")
print(pd.DataFrame({
    'Missing Count': missing_val[missing_val > 0],
    'Percentage': missing_val_percentage[missing_val > 0]
}))
print("Total missing values di Train Set:", train_set.isnull().sum().sum())
print("Total missing values di Validation Set:", val_set.isnull().sum().sum())
```

Terakhir, kode memeriksa hasil imputasi pada training dan validation set untuk memastikan tidak ada missing values yang tersisa untuk verifikasi bahwa proses imputasi telah berhasil dilakukan dengan sempurna.

b. Handling Outlier

```python
# Cek dulu kondisi outlier masing2 kolom
numerical_columns = train_set.select_dtypes(include=['float64', 'int64']).columns

outlier_analysis = {}

for col in numerical_columns:

    Q1 = train_set[col].quantile(0.25)
    Q3 = train_set[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR


    z_scores = zscore(train_set[col].dropna())
    z_outliers = np.abs(z_scores) > 3

    outlier_analysis[col] = {
        "IQR Lower Bound": lower_bound,
        "IQR Upper Bound": upper_bound,
        "Z-Score Outliers Count": z_outliers.sum(),
        "IQR Outliers Count": ((train_set[col] < lower_bound) | (train_set[col] > upper_bound)).sum(),
        "Skewness": train_set[col].skew()
    }


    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.hist(train_set[col].dropna(), bins=30, color='blue', alpha=0.7)
    plt.title(f'Histogram of {col}')

    plt.subplot(1, 2, 2)
    plt.boxplot(train_set[col].dropna(), vert=False, patch_artist=True)
    plt.title(f'Boxplot of {col}')

    plt.tight_layout()
    plt.show()

outlier_df = pd.DataFrame(outlier_analysis).T
print(outlier_df)
```

Pada handling outlier, kami menerapkan analisis awal terhadap outlier untuk setiap kolom numerik dengan menggunakan dua metode: IQR (Interquartile Range) dan Z-score. Untuk metode IQR, kode menghitung Q1 (kuartil pertama), Q3 (kuartil ketiga), dan IQR (Q3-Q1), kemudian menentukan batas bawah (Q1 - 1.5*IQR) dan batas atas (Q3 + 1.5*IQR). Untuk metode Z-score, data yang memiliki nilai absolut z-score lebih dari 3 dianggap sebagai outlier. Kode juga menghitung skewness untuk memahami kemiringan distribusi data. Hasil analisis divisualisasikan menggunakan histogram dan boxplot untuk setiap kolom, memberikan gambaran visual tentang distribusi data dan posisi outlier.

```python
# Imputasi Outliernya
numerical_columns = train_set.select_dtypes(include=['float64', 'int64']).columns

binary_columns = [col for col in numerical_columns if set(train_set[col].dropna().unique()) <= {0, 1}]

parameters = {}

for col in numerical_columns:
    if col in binary_columns:
        continue

    Q1 = train_set[col].quantile(0.25)
    Q3 = train_set[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    parameters[col] = {'lower_bound': lower_bound, 'upper_bound': upper_bound}

    train_set[col] = train_set[col].clip(lower=lower_bound, upper=upper_bound)

for col in numerical_columns:
    if col in binary_columns:
        continue

    lower_bound = parameters[col]['lower_bound']
    upper_bound = parameters[col]['upper_bound']


    val_set[col] = val_set[col].clip(lower=lower_bound, upper=upper_bound)
```

Selanjutnya, kode melakukan penanganan outlier dengan metode clipping (pemotongan). Proses ini dimulai dengan mengidentifikasi kolom-kolom numerik dan memisahkan kolom biner (yang hanya berisi nilai 0 dan 1) karena kolom biner tidak perlu ditangani outliernya. Untuk setiap kolom non-biner, kode menerapkan metode clipping dengan menggunakan batas atas dan bawah yang telah dihitung sebelumnya. Nilai-nilai yang berada di luar batas tersebut "dipotong" ke nilai batas terdekat. Parameter batas yang digunakan untuk training set juga diterapkan ke validation set untuk menjaga konsistensi.

```python
# Cek lagi kondisi outlier masing2 kolom
numerical_columns = train_set.select_dtypes(include=['float64', 'int64']).columns


outlier_analysis = {}

for col in numerical_columns:

    Q1 = train_set[col].quantile(0.25)
    Q3 = train_set[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR


    z_scores = zscore(train_set[col].dropna())
    z_outliers = np.abs(z_scores) > 3


    outlier_analysis[col] = {
        "IQR Lower Bound": lower_bound,
        "IQR Upper Bound": upper_bound,
        "Z-Score Outliers Count": z_outliers.sum(),
        "IQR Outliers Count": ((train_set[col] < lower_bound) | (train_set[col] > upper_bound)).sum(),
        "Skewness": train_set[col].skew()
    }

    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.hist(train_set[col].dropna(), bins=30, color='blue', alpha=0.7)
    plt.title(f'Histogram of {col}')

    plt.subplot(1, 2, 2)
    plt.boxplot(train_set[col].dropna(), vert=False, patch_artist=True)
    plt.title(f'Boxplot of {col}')

    plt.tight_layout()
    plt.show()

outlier_df = pd.DataFrame(outlier_analysis).T
print(outlier_df)
```

Terakhir, kode melakukan analisis ulang untuk memverifikasi hasil penanganan outlier dengan menggunakan metode dan visualisasi yang sama seperti analisis awal. Pendekatan ini dipilih karena clipping merupakan metode yang lebih ‘menjaga’ data dibandingkan dengan menghapus outlier, sehingga tidak mengurangi jumlah data yang tersedia untuk training model. Selain itu, metode ini juga mempertahankan tren dan pola dalam data sambil mengurangi dampak nilai-nilai ekstrim yang mungkin dapat mempengaruhi performa model.

c. Handling Duplicate

```
train_duplicates = train_set.duplicated().sum()
print(f"Jumlah duplikasi pada train_set: {train_duplicates}")

val_duplicates = val_set.duplicated().sum()
print(f"Jumlah duplikasi pada val_set: {val_duplicates}")
```

Pada implementasi ini, Fungsi duplicated() mengembalikan Series boolean yang menandai setiap baris apakah merupakan duplikat dari baris sebelumnya, kemudian sum() menghitung total nilai True yang ditemukan, sehingga memberikan jumlah total baris duplikat. Pengecekan duplikasi dilakukan karena data duplikat dapat menyebabkan bias dalam model machine learning. Jika baris yang sama muncul berkali-kali dalam dataset training, model bisa menjadi overfitting terhadap pola tersebut. Selain itu, duplikasi dalam validation set juga dapat memberikan evaluasi performa model yang tidak akurat.

d. Feature Engineering

```
#Feature Selection. Kami akan drop salah satu kolom yang memiliki korelasi tinggi antar kolomnya.
import pandas as pd
import numpy as np

correlation_threshold = 0.65

correlation_matrix = train_set.corr()

upper_triangle = np.triu(np.ones(correlation_matrix.shape), k=1)
high_correlation_pairs = np.where((correlation_matrix > correlation_threshold) & (upper_triangle == 1))

columns_to_drop = set()
for i, j in zip(high_correlation_pairs[0], high_correlation_pairs[1]):
    col1 = correlation_matrix.columns[i]
    col2 = correlation_matrix.columns[j]
    columns_to_drop.add(col2)

train_set.drop(columns=columns_to_drop, inplace=True)
val_set.drop(columns=columns_to_drop, inplace=True)

print("Matriks Korelasi:\n", correlation_matrix)
```

Pertama, ditetapkan threshold korelasi sebesar 0.65, yang berarti kolom-kolom yang memiliki korelasi di atas nilai ini akan dianggap memiliki korelasi tinggi.

Kemudian, dibuat matriks korelasi menggunakan train_set.corr() untuk melihat hubungan antar semua kolom numerik.

Selanjutnya, kode menggunakan np.triu untuk membuat matriks segitiga atas (upper triangle) dengan nilai 1, yang membantu menghindari penghitungan korelasi yang duplikat karena matriks korelasi bersifat simetris. Fungsi np.where digunakan untuk menemukan pasangan kolom yang memiliki korelasi di atas threshold (0.65) dan berada di segitiga atas matriks.

Untuk setiap pasangan kolom yang teridentifikasi memiliki korelasi tinggi, kode akan menambahkan salah satu kolom (col2) ke dalam set columns_to_drop. Pendekatan ini dilakukan karena kolom-kolom yang sangat berkorelasi cenderung memberikan informasi yang redundan, yang bisa menyebabkan masalah multikolinearitas dalam model machine learning. Terakhir, kolom-kolom yang terpilih untuk dihapus dikeluarkan dari kedua dataset (train_set dan val_set) menggunakan fungsi drop().

Penerapan features engineering dapat meningkatkan efisiensi model dengan mengurangi dimensi data dan menghindari redundansi informasi, serta mencegah masalah overfitting yang bisa terjadi ketika menggunakan fitur-fitur yang sangat berkorelasi.

B. Preprocessing

a. Feature Scaling

```python
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.q1 = None
        self.q3 = None
        self.iqr = None

    def fit(self, X, y=None):
        X = np.array(X)
        self.q1 = np.percentile(X, 25, axis=0)
        self.q3 = np.percentile(X, 75, axis=0)
        self.iqr = self.q3 - self.q1
        return self

    def transform(self, X):
        X = np.array(X)
        return (X - self.q1) / np.where(self.iqr == 0, 1, self.iqr)
```

Kode ini mendefinisikan class FeatureScaler yang merupakan implementasi custom scaler untuk melakukan normalisasi fitur menggunakan metode IQR (Interquartile Range). Class ini mewarisi dari BaseEstimator dan TransformerMixin dari scikit-learn, yang memungkinkannya untuk digunakan dalam pipeline preprocessing scikit-learn.

Class ini memiliki tiga metode utama:

1. **init**: Menginisialisasi variabel untuk menyimpan nilai Q1 (persentil 25), Q3 (persentil 75), dan IQR (Q3-Q1)
2. **fit**: Menghitung nilai Q1, Q3, dan IQR dari data training
3. **transform**: Melakukan normalisasi data dengan formula (X - Q1) / IQR, dengan penanganan khusus untuk kasus IQR = 0

Alasan penggunaan normalisasi berbasis IQR ini adalah:

- Lebih robust terhadap outlier dibandingkan dengan StandardScaler atau MinMaxScaler karena menggunakan kuartil
- Membantu menstandarkan skala fitur tanpa terpengaruh nilai ekstrim
- Formula normalisasi ini menghasilkan distribusi di mana Q1 akan menjadi 0 dan Q3 akan menjadi 1, membuat skala data lebih mudah didefinisikan
- Penggunaan np.where pada transform memastikan tidak ada pembagian dengan nol saat IQR = 0

Implementasi ini diterapkan untuk preprocessing data sebelum dimasukkan ke model machine learning, terutama untuk dataset yang memiliki outlier atau distribusi yang tidak normal.

b. Feature Encoding

Pada preprocessing ini kami tidak menerapkan feature encoding dengan beberapa alasan berikut:

1. Berdasarkan analisis dataset yang diberikan, fitur-fitur kategorikal (bertipe object) yang ada seperti Domain sudah dihapus sebelumnya karena memiliki missing value yang tinggi (>49%). Ini mengurangi kebutuhan untuk melakukan encoding.
2. Dataset PhiUSIIL Phishing URL sudah memiliki format yang cukup terstruktur di mana kebanyakan fitur sudah dalam bentuk numerik, termasuk fitur-fitur yang sebenarnya bersifat kategorikal seperti:
    - IsDomainIP (0/1)
    - HasObfuscation (0/1)
    - IsHTTPS (0/1)
    - HasTitle (0/1) dan fitur biner lainnya
3. Fitur-fitur yang ada sudah merepresentasikan hasil ekstraksi dari URL dan halaman web, di mana nilai-nilai kategorikal sudah dikonversi menjadi nilai numerik saat proses ekstraksi fitur dilakukan, seperti yang dijelaskan dalam metadata dataset.

4. Model yang digunakan (KNN dan Gaussian Naive Bayes) bekerja dengan baik untuk data numerik, dan dataset ini sudah dalam format yang sesuai tanpa memerlukan encoding tambahan.

c. Feature Imbalance Class Handling

```python
from imblearn.over_sampling import SMOTE
from sklearn.base import BaseEstimator, TransformerMixin

class SMOTEWrapper:
    def __init__(self, random_state=42):
        self.smote = SMOTE(random_state=random_state)

    def fit_resample(self, X, y):
        return self.smote.fit_resample(X, y)
```

Kode ini mendefinisikan sebuah class wrapper bernama SMOTEWrapper yang mengenkapsulasi fungsionalitas SMOTE (Synthetic Minority Over-sampling Technique) dari library imlearn. Class ini memiliki dua metode utama: **init** yang menginisialisasi objek SMOTE dengan random_state yang dapat diatur (default 42 untuk reproducibility), dan fit_resample yang memanggil metode fit_resample dari objek SMOTE untuk melakukan oversampling pada data.

SMOTE sendiri adalah teknik yang digunakan untuk menangani imbalanced dataset dengan cara membuat sampel sintetis dari kelas minoritas. Penggunaan wrapper ini memudahkan integrasi SMOTE ke dalam pipeline preprocessing data, terutama ketika dataset memiliki distribusi kelas yang tidak seimbang. Random state diset ke nilai tetap (42) untuk memastikan hasil yang konsisten setiap kali kode dijalankan, yang penting untuk reproducibility eksperimen.

BAB V

PERBANDINGAN HASIL PREDIKSI

Hasil KNN from scratch:

```
KNN (Scratch) Euclidan Accuracy: 0.9343065693430657
KNN (Scratch) Euclidean Classification Report:
              precision    recall  f1-score   support

         0.0       0.54      0.90      0.67       420
         1.0       0.99      0.94      0.96      5197

    accuracy                           0.93      5617
   macro avg       0.76      0.92      0.82      5617
weighted avg       0.96      0.93      0.94      5617

KNN (Scratch) Manhattan Accuracy: 0.9554922556524835
KNN (Scratch) Manhattan Classification Report:
              precision    recall  f1-score   support

         0.0       0.65      0.86      0.74       420
         1.0       0.99      0.96      0.98      5197

    accuracy                           0.96      5617
   macro avg       0.82      0.91      0.86      5617
weighted avg       0.96      0.96      0.96      5617

KNN (Scratch) Minkowski Accuracy: 0.9170375645362293
KNN (Scratch) Minkowski Classification Report:
              precision    recall  f1-score   support

         0.0       0.47      0.88      0.61       420
         1.0       0.99      0.92      0.95      5197
```

Hasil KNN dari Scikit Learn:

```
KNN (Scikit-Learn) Accuracy: 0.9343065693430657
KNN (Scikit-Learn) Classification Report:
              precision    recall  f1-score   support

         0.0       0.54      0.90      0.67       420
         1.0       0.99      0.94      0.96      5197

    accuracy                           0.93      5617
   macro avg       0.76      0.92      0.82      5617
weighted avg       0.96      0.93      0.94      5617
```

Dari hasil ini dapat dilihat bahwa hasil implementasi KNN manual memiliki hasil akurasi yang mirip dengan penggunaan library Scikit-Learn. Namun dapat juga dilihat berdasarkan parameter yang digunakan, algoritma dari scratch memiliki variasi dalam akurasinya.

Hasil Naive Bayes from scratch:

```
Naive Bayes (Scratch) Accuracy: 0.959408937155065
Naive Bayes (Scratch) Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.46      0.63       420
         1.0       0.96      1.00      0.98      5197

    accuracy                           0.96      5617
   macro avg       0.97      0.73      0.80      5617
weighted avg       0.96      0.96      0.95      5617
```

Hasil Naive Bayes dari Scikit Learn:

```
Naive Bayes (Scikit-Learn) Accuracy: 0.9601210610646252
Naive Bayes (Scikit-Learn) Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      0.47      0.64       420
         1.0       0.96      1.00      0.98      5197

    accuracy                           0.96      5617
   macro avg       0.97      0.74      0.81      5617
weighted avg       0.96      0.96      0.95      5617
```

Dari hasil ini dapat dilihat bahwa hasil implementasi Naive Bayes manual memiliki hasil akurasi yang mirip, sedikit lebih buruk, dengan penggunaan library Scikit-Learn.

Dari perbandingan yang didapat terkait KNN dan Naive Bayes from scratch dan menggunakan sklearn, didapatkan bahwa implementasi KNN dan Naive Bayes yang kami buat sudah hampir optimal, meskipun masih dapat dilakukan *improvement*.

PEMBAGIAN TUGAS

Nama	**NIM**	**Tugas**
Daffari Adiyatma	18222003	Membuat laporan
		Data Cleaning
		Data Preprocessing
		Error Analysis
Aththariq Lisan Q. D. S.	18222013	Membuat laporan
		Data Cleaning
		Data Preprocessing
		Membuat repository github
		Submission Kaggle
Muhammad Rafly	18222067	Implementasi KNN
		Feature Engineering
		Implementasi Bayes
		Membuat laporan

Alvin Fadhilah Akmal	18222079	Implementasi KNN
		Optimasi Naive Bayes
		Model *save and load*
		Membuat laporan dan analisis error

REFERENSI

PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning. https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558#se0060

Scikit-learn.org https://scikit-learn.org/stable/modules/naive_bayes.html

https://pandas.pydata.org/docs/ (dokumentasi pandas)

https://www.geeksforgeeks.org/k-nearest-neighbours/

https://www.w3schools.com/python/python_ml_knn.asp