# Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial

# Implementasi Algoritma Pembelajaran Mesin

i


Oleh

**Kelompok 7 (Teletubbies)**

Bryan P. Hutagalung (18222130)

Ardra Rafif Sahasika (18222134)

Timothy Haposan Simanjuntak (18222137)

Yusril Fazri Mahendra (18222141)

**Program Studi Sistem dan Teknologi Informasi**

**Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung**

**Jl. Ganesha 10, Bandung 40132**

**2024**

# Daftar Isi

# Daftar Gambar

# BAB I

# Deskripsi Persoalan

Tugas Besar 2 pada kuliah IF3070 Dasar Inteligensi Buatan bertujuan untuk memberikan pengalaman langsung kepada peserta kuliah dalam menerapkan algoritma pembelajaran mesin pada permasalahan nyata. Pembelajaran mesin merupakan salah satu cabang dari kecerdasan buatan yang memungkinkan sistem untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit. Pada tugas ini, Anda diminta untuk mengimplementasikan algoritma pembelajaran mesin yang telah kalian pelajari di kuliah, yaitu KNN dan Gaussian Naive-Bayes pada dataset PhiUSIIL Phising URL Dataset.

# BAB II

# Implementasi KNN

Algoritma K-Nearest Neighbor (KNN) adalah algoritma machine learning yang bersifat non-parametric dan lazy learning. Metode yang bersifat non-parametric memiliki makna bahwa metode tersebut tidak membuat asumsi apa pun tentang distribusi data yang mendasarinya. Dengan kata lain, tidak ada jumlah parameter atau estimasi parameter yang tetap dalam model, terlepas data tersebut berukuran kecil ataupun besar. jarak). Jika, Lazy learning artinya, proses pembelajaran tidak ada sampai data test diberikan. Algoritma hanya menyimpan data training dan melakukan perhitungan ketika diminta untuk melakukan prediksi. KNN akan bekerja dengan membandingkan train set dengan test set. Prediksi dibuat berdasarkan data baru dan data training. Pada implementasi KNN di tugas ini, terdapat tiga algoritma penentuan jarak, yaitu : Euclidean, Manhattan, dan Minkowski.

a. Rumus Euclidean:

$$\mathrm{dist}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Gambar 2.1. Rumus Euclidean

b. Rumus Manhattan:

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

Gambar 2.2. Rumus Manhattan

c. Rumus Minkowski:

$$\text{Minkowski Distance} = \left(\sum_{i=1}^{n} |x_i - y_i|\right)^{1/p}$$

Gambar 2.3. Rumus Minkowski

Kami membuat algoritma untuk menghitung KNN seperti di bawah ini.

1. __init__: Fungsi ini untuk menginisialisasi parameter model K-Nearest Neighbors (KNN), seperti jumlah tetangga terdekat (k), jenis matrik jarak (metric), metode pembobotan (weights), jumlah pekerjaan paralel (n_jobs), ukuran batch (batch_size), dan tingkat keluaran verbose.
2. _compute_distances: Fungsi ini untuk menghitung jarak antara sampel dalam dataset uji (X1) dan dataset latih (X2) menggunakan matrik jarak yang dipilih (manhattan, euclidean, atau minkowski). Dirancang untuk efisiensi memori dengan menghitung jarak secara iteratif.
3. fit: Fungsi ini untuk menyimpan data latih (fitur dan label) yang akan digunakan selama prediksi. Fungsi ini hanya memuat data ke dalam atribut internal tanpa pelatihan aktual karena KNN adalah model berbasis memori (lazy learning).
4. _predict_batch: Fungsi ini untuk melakukan prediksi untuk satu batch data uji. Fungsi ini menghitung jarak ke tetangga terdekat, memilih k tetangga, dan menentukan label berdasarkan mode (frekuensi tertinggi) atau pembobotan jarak.
5. predict: Fungsi ini untuk melakukan prediksi untuk dataset uji penuh dengan memproses data secara batch untuk menghindari masalah memori. Jarak dihitung secara paralel jika memungkinkan, dan hasil prediksi dikembalikan untuk semua sampel.
6. save: Fungsi ini untuk menyimpan model yang telah diinisialisasi ke dalam file menggunakan format pickle sehingga bisa digunakan di masa depan tanpa perlu inisialisasi ulang.
7. load: Fungsi ini untuk memuat model yang disimpan sebelumnya dari file pickle. Fungsi ini memulihkan objek model ke kondisi saat disimpan untuk prediksi atau penggunaan lebih lanjut.

Berikut adalah kode untuk KNN.py:

```
import numpy as np
import pandas as pd
import pickle
```

```
import concurrent.futures
from os import cpu_count
from tqdm import tqdm
import time
from scipy.stats import mode

class knn:
    def __init__(self, k=5, n_jobs=1, metric='minkowski', p=2,
weights='uniform',
                 verbose=True, batch_size=100):  # Reduced batch size
        # Input validation
        if k < 1 or not isinstance(k, int):
            raise ValueError("k must be a positive integer.")
        if metric not in ['manhattan', 'euclidean', 'minkowski']:
            raise ValueError("Invalid metric. Choose from 'manhattan',
'euclidean', or 'minkowski'.")
        if p < 1 or not isinstance(p, (int, float)):
            raise ValueError("p must be a positive number.")
        if weights not in ['uniform', 'distance']:
            raise ValueError("weights must be either 'uniform' or
'distance'.")
        if n_jobs < 1 and n_jobs != -1 or not isinstance(n_jobs, int):
            raise ValueError("n_jobs must be a positive integer or -1.")

        self.k = k
        self.verbose = verbose
        self.metric = metric
        self.weights = weights
        self.p = p if metric == 'minkowski' else (1 if metric ==
'manhattan' else 2)
        self.n_jobs = cpu_count() if n_jobs == -1 else n_jobs
        self.batch_size = batch_size

    def _compute_distances(self, X1, X2):
        """
        Memory-efficient distance computation
        """
        if self.metric == 'euclidean':
            # Compute distances without creating large intermediate
```

```
arrays
            distances = np.zeros((X1.shape[0], X2.shape[0]))
            for i in range(X1.shape[0]):
                distances[i] = np.sqrt(np.sum((X2 - X1[i]) ** 2,
axis=1))
            return distances
        elif self.metric == 'manhattan':
            distances = np.zeros((X1.shape[0], X2.shape[0]))
            for i in range(X1.shape[0]):
                distances[i] = np.sum(np.abs(X2 - X1[i]), axis=1)
            return distances
        else:  # minkowski
            distances = np.zeros((X1.shape[0], X2.shape[0]))
            for i in range(X1.shape[0]):
                distances[i] = np.power(np.sum(np.power(np.abs(X2 -
X1[i]), self.p), axis=1), 1/self.p)
            return distances

    def fit(self, X_train, y_train):
        self.X_train = X_train.values.astype(np.float32) if
isinstance(X_train, pd.DataFrame) else X_train.astype(np.float32)
        self.y_train = np.array(y_train)
        return self

    def _predict_batch(self, X_batch):
        # Compute distances for the batch
        distances = self._compute_distances(X_batch, self.X_train)

        # Get indices of k nearest neighbors
        nearest_neighbor_indices = np.argpartition(distances, self.k,
axis=1)[:, :self.k]

        # Get corresponding labels
        nearest_labels = self.y_train[nearest_neighbor_indices]

        if self.weights == 'uniform':
            predictions = mode(nearest_labels, axis=1)[0].flatten()
        else:
            k_distances = np.take_along_axis(distances,
```

```
nearest_neighbor_indices, axis=1)
            weights = 1 / (k_distances + 1e-5)
            weights /= np.sum(weights, axis=1, keepdims=True)

            predictions = np.zeros(X_batch.shape[0], dtype=int)
            for i in range(X_batch.shape[0]):
                predictions[i] = np.bincount(nearest_labels[i],
                                             weights=weights[i],

minlength=len(np.unique(self.y_train))).argmax()

        return predictions

    def predict(self, X_test):
        if self.verbose:
            print(f"Using {self.n_jobs} {'core' if self.n_jobs == 1 else
'cores'} for predictions.")

        X_test = X_test.values.astype(np.float32) if isinstance(X_test,
pd.DataFrame) else X_test.astype(np.float32)

        # Process in smaller chunks sequentially to avoid memory issues
        n_samples = X_test.shape[0]
        predictions = np.zeros(n_samples, dtype=int)

        for i in tqdm(range(0, n_samples, self.batch_size), disable=not
self.verbose):
            end_idx = min(i + self.batch_size, n_samples)
            batch = X_test[i:end_idx]
            predictions[i:end_idx] = self._predict_batch(batch)

        return predictions

    def save(self, path):
        with open(path, 'wb') as f:
            pickle.dump(self, f)

    @staticmethod
    def load(path):
```

```
        with open(path, 'rb') as f:
            return pickle.load(f)
```

# BAB III

# Implementasi Naive-Bayes

Gaussian Naive Bayes (GNB) adalah teknik klasifikasi dalam machine learning yang menggunakan pendekatan probabilitas dan distribusi Gaussian. GNB mengasumsikan bahwa setiap parameter, yang disebut juga fitur atau prediktor, memiliki kemampuan independen untuk memprediksi variabel output. Kombinasi prediksi dari semua parameter tersebut menghasilkan prediksi akhir berupa probabilitas yang menunjukkan kemungkinan variabel tergantung masuk ke dalam setiap kelompok. Klasifikasi akhir akan diberikan pada kelompok dengan probabilitas tertinggi. Teorema Bayes dapat dilihat sebagai berikut.

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

Gambar 3.1. Rumus Teorema Bayes

- P(A|B): Probabilitas bersyarat A yang diberikan oleh B.
- P(B|A): Probabilitas bersyarat B yang diberikan oleh A.
- P(A): Probabilitas kejadian A.
- P(B): Probabilitas kejadian B.

Distribusi Gaussian juga dikenal sebagai distribusi normal. Distribusi normal adalah model statistik yang menggambarkan sebaran variabel acak kontinu di alam dan ditandai oleh kurva berbentuk lonceng. Dua karakteristik utama dari distribusi normal adalah rata-rata (μ) dan simpangan baku (σ). Rata-rata (mean) menunjukkan nilai rata-rata dari distribusi, sedangkan simpangan baku menggambarkan "lebar" distribusi di sekitar rata-rata. Sebuah variabel (X) yang berdistribusi normal memiliki nilai yang tersebar secara kontinu (variabel kontinu) dari $-\infty < X < +\infty$, dengan total luas area di bawah kurva model sama dengan 1. Distribusi normal memiliki persamaan matematika yang menentukan probabilitas suatu pengamatan berada dalam salah satu kelompok. Rumusnya sebagai berikut.

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

Gambar 3.2. Distribusi Normal

Kami membuat algoritma untuk menghitung GNB seperti di bawah ini.

1. __init__: Fungsi ini adalah konstruktor untuk inisialisasi objek Gaussian Naive Bayes (gnb). Di sini, parameter awal seperti jumlah pekerjaan paralel (n_jobs), ukuran batch, prior kelas, rata-rata, varians, dan atribut lainnya diatur untuk digunakan saat pelatihan dan prediksi.
2. fit: Fungsi ini melatih model Gaussian Naive Bayes menggunakan data masukan (X) dan (y).
3. _compute_log_likelihood_batch: Fungsi ini menghitung log likelihood untuk batch data tertentu. Dengan menggunakan operasi vektor, fungsi ini memperkirakan seberapa besar kemungkinan suatu data untuk setiap kelas berdasarkan distribusi Gaussian, sehingga menjadi dasar untuk prediksi kelas.
4. predict: Fungsi ini digunakan untuk memprediksi kelas dari data masukan. Data diproses dalam batch untuk efisiensi, terutama pada dataset besar, dan log likelihood dihitung untuk setiap kelas sebelum mengembalikan prediksi berdasarkan kemungkinan tertinggi.
5. save: Fungsi ini menyimpan model yang telah dilatih ke dalam file menggunakan format pickle atau jika dilihat pada file berupa (.pkl), sehingga dapat digunakan kembali tanpa harus melatih ulang.
6. load: Fungsi ini memuat model yang telah disimpan sebelumnya dari file. Hal ini memungkinkan model dilanjutkan atau digunakan untuk prediksi di masa depan tanpa harus.

Berikut adalah kode untuk GNB.py:

```
import numpy as np
from collections import defaultdict
import pickle
import concurrent.futures
```

```
from tqdm import tqdm

class gnb:
    """
    Gaussian Naive Bayes classifier implementation from scratch
    Optimized for large datasets using vectorization and parallel
processing
    """
    def __init__(self, n_jobs=-1, batch_size=1000):
        # Initialize model parameters
        self.class_priors = None
        self.means = None
        self.variances = None
        self.classes = None
        self.n_features = None
        self.n_jobs = n_jobs
        self.batch_size = batch_size

        # Pre-computed constants for Gaussian density
        self._const = None
        self._log_const = None

    def fit(self, X, y):
        """
        Fit Gaussian Naive Bayes classifier
        Vectorized implementation for faster training
        """
        # Convert input to numpy array if needed
        X = X.values if hasattr(X, 'values') else np.array(X)
        y = np.array(y)

        # Store dimensions
        self.n_features = X.shape[1]
        self.classes = np.unique(y)

        # Compute class priors
        class_counts = np.bincount(y)
        self.class_priors = class_counts / len(y)
```

```
        # Initialize parameters
        self.means = np.zeros((len(self.classes), self.n_features))
        self.variances = np.zeros((len(self.classes), self.n_features))

        # Compute means and variances for each class
        for i, c in enumerate(self.classes):
            X_c = X[y == c]
            self.means[i] = X_c.mean(axis=0)
            self.variances[i] = X_c.var(axis=0) + 1e-9  # Add small
constant for numerical stability

        # Pre-compute constants for Gaussian density
        self._const = 1 / np.sqrt(2 * np.pi * self.variances)
        self._log_const = np.log(self._const)

        return self

    def _compute_log_likelihood_batch(self, X_batch):
        """
        Compute log likelihood for a batch of samples
        Vectorized implementation for faster prediction
        """
        # Initialize log likelihood matrix
        log_likelihood = np.zeros((X_batch.shape[0], len(self.classes)))

        # Add log priors
        log_likelihood += np.log(self.class_priors)

        # Compute log likelihood for each class
        for i, _ in enumerate(self.classes):
            # Vectorized computation of Gaussian density
            diff = X_batch - self.means[i]
            exponent = -0.5 * np.sum(diff ** 2 / self.variances[i],
axis=1)
            log_likelihood[:, i] += np.sum(self._log_const[i]) +
exponent

        return log_likelihood
```

```
    def predict(self, X):
        """
        Predict class labels for samples in X
        Uses batch processing and parallel computation for large
datasets
        """
        # Convert input to numpy array if needed
        X = X.values if hasattr(X, 'values') else np.array(X)

        # Process data in batches
        n_samples = X.shape[0]
        n_batches = (n_samples + self.batch_size - 1) // self.batch_size
        predictions = np.zeros(n_samples, dtype=int)

        for i in tqdm(range(n_batches)):
            start_idx = i * self.batch_size
            end_idx = min((i + 1) * self.batch_size, n_samples)
            X_batch = X[start_idx:end_idx]

            # Compute log likelihoods for batch
            log_likelihood = self._compute_log_likelihood_batch(X_batch)

            # Get predictions for batch
            predictions[start_idx:end_idx] =
self.classes[np.argmax(log_likelihood, axis=1)]

        return predictions

    def save(self, path):
        """Save model to disk"""
        with open(path, 'wb') as f:
            pickle.dump(self, f)

    @staticmethod
    def load(path):
        """Load model from disk"""
        with open(path, 'rb') as f:
            return pickle.load(f)
```

# BAB IV

# Tahap Cleaning dan Preprocessing

## 4. 1 Data Cleaning

### 4.1.1. Handling Missing Data - Data Imputation

Untuk mengurus nilai data yang hilang, kami menggunakan data imputation yang merupakan proses mengganti nilai-nilai yang hilang (missing values) dalam suatu dataset dengan nilai-nilai yang diperkirakan atau diproduksi berdasarkan informasi yang tersedia. Kami menggunakan metode basic imputation dengan mean, median, dan modus. Modus digunakan untuk kolom kategorikal untuk mendapat nilai yang sering muncul dan mean atau median digunakan untuk kolom numerik.

**Implementasi**

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

class BasicImputationHandler(BaseEstimator,
TransformerMixin):
    def __init__(self, num_feats, cat_feats, method='mean'):
        self.num_feats = num_feats
        self.cat_feats = cat_feats
        self.method = method
        self.fill_values = {}

    def fit(self, X, y=None):
        for column in X.columns:
            if column in self.cat_feats:
                self.fill_values[column] =
X[column].mode().iloc[0]
            elif column in self.num_feats:
                if self.method == 'mean':
                    self.fill_values[column] =
X[column].mean()
```

```
                elif self.method == 'median':
                    self.fill_values[column] =
X[column].median()
        return self

    def transform(self, X):
        X = X.copy()
        for column, value in self.fill_values.items():
            X[column] = X[column].fillna(value)
        return X
```

### 4.1.2. Dealing with Outliers - Clipping

Outliers data adalah data yang memiliki nilai yang berbeda secara signifikan dengan mayoritas dari data yang ada. Data tersebut dapat sangat tinggi dan rendah yang tidak sesuai dengan pola dari dataset sisanya sehingga berpengaruh terhadap performa dari model yang dimiliki. Kami menggunakan metode clipping yang memotong data yang berada di luar batas minimum dan maksimum

**Implementasi**

```
class ClippingOutlierHandler(BaseEstimator,
TransformerMixin):
    def __init__(self, num_feats, threshold=1.5):
        self.num_feats = num_feats
        self.threshold = threshold
        self.bounds = {}

    def fit(self, X, y=None):
        for col in self.num_feats:
            Q1 = X[col].quantile(0.25)
            Q3 = X[col].quantile(0.75)
            IQR = Q3 - Q1
            self.bounds[col] = {
                'lower': Q1 - self.threshold * IQR,
                'upper': Q3 + self.threshold * IQR
```

```
            }
        return self

    def transform(self, X):
        X = X.copy()
        for col in self.num_feats:
            X[col] = X[col].clip(
                lower=self.bounds[col]['lower'],
                upper=self.bounds[col]['upper']
            )
        return X
```

### 4.1.3. Remove Duplicates

Menghilangkan data duplikat merupakan langkah krusial karena dapat memengaruhi integritas data yang berpengaruh pada akurasi analisis dan insight. Kami menganalisis keberadaan duplikat pada tiga dataset terpisah, yaitu train_set, val_set, dan test_set serta menghapusnya.

**Implementasi**

```
def handle_duplicates(train_set, val_set, test_set,
verbose=True):
   if verbose:
        print("Initial shapes:")
        print(f"Train set: {train_set.shape}")
        print(f"Validation set: {val_set.shape}")
        print(f"Test set: {test_set.shape}")
        print("\nChecking for duplicates...")

        train_dups = train_set.duplicated().sum()
        val_dups = val_set.duplicated().sum()
        test_dups = test_set.duplicated().sum()

        print(f"\nDuplicates found:")
        print(f"Train set: {train_dups}")
        print(f"Validation set: {val_dups}")
```

```
        print(f"Test set: {test_dups}")


    train_set_clean = train_set.drop_duplicates(keep='first')
    val_set_clean = val_set.drop_duplicates(keep='first')
    test_set_clean = test_set.drop_duplicates(keep='first')

    if verbose:

        print("\nFinal shapes after removing duplicates:")
        print(f"Train set: {train_set_clean.shape}")
        print(f"Validation set: {val_set_clean.shape}")
        print(f"Test set: {test_set_clean.shape}")


        train_reduction = (1 -
len(train_set_clean)/len(train_set)) * 100
        val_reduction = (1 - len(val_set_clean)/len(val_set))
* 100
        test_reduction = (1 -
len(test_set_clean)/len(test_set)) * 100

        print(f"\nReduction percentages:")
        print(f"Train set: {train_reduction:.2f}%")
        print(f"Validation set: {val_reduction:.2f}%")
        print(f"Test set: {test_reduction:.2f}%")

    return train_set_clean, val_set_clean, test_set_clean

train_set, val_set, test_set = handle_duplicates(train_set,
val_set, test_set)
```

#### 4.1.4. Feature Engineering

Feature engineering merupakan pembuatan fitur baru atau transformasi fitur yang sudah ada untuk meningkatkan kinerja model machine learning. Tujuan dari feature engineering adalah untuk meningkatkan kemampuan model dalam mempelajari pola dan membuat prediksi yang akurat

berdasarkan data. Kami menggunakan Feature Selection, menghilangkan fitur yang tidak relevan atau redundan dengan berbagai metode, seperti korelasi, mutual information, dan ambang batas varians, dan Feature Creator, penciptaan fitur baru dari data yang ada dengan pendekatan utama, yaitu *polynomial features, key ratios, dan interaction terms*.

**Implementasi Feature Selection**

```
class FeatureSelector(BaseEstimator, TransformerMixin):
    def __init__(self, num_feats, cat_feats,
method='correlation', threshold=0.1, target_col='label'):
        self.num_feats = num_feats
        self.cat_feats = cat_feats
        self.method = method
        self.threshold = threshold
        self.selected_features = None
        self.target_col = target_col

        self.drop_columns = ['id', 'FILENAME', 'URL',
'Domain', 'TLD', 'Title']

    def fit(self, X, y=None):
                X_temp = X.copy()
        if y is not None:
            X_temp[self.target_col] = y
        self.columns_to_drop = [col for col in
self.drop_columns if col in X_temp.columns]
        X_temp = X_temp.drop(columns=self.columns_to_drop)

        if self.method == 'correlation':

            numeric_cols =
X_temp.select_dtypes(include=[np.number]).columns
            corr_matrix = X_temp[numeric_cols].corr()

            if self.target_col in corr_matrix.columns:

                target_corr =
```

```
abs(corr_matrix[self.target_col])

                self.selected_features =
target_corr[target_corr > self.threshold].index
            else:
                self.selected_features = numeric_cols

        elif self.method == 'mutual_info':

            selector =
SelectKBest(score_func=mutual_info_classif, k=20)
            selector.fit(X_temp[self.num_feats], y)
            self.selected_features =
X_temp[self.num_feats].columns[selector.get_support()]


        self.num_feats = [col for col in self.num_feats if
col not in self.columns_to_drop]
        self.cat_feats = [col for col in self.cat_feats if
col not in self.columns_to_drop]

        return self

    def transform(self, X):

        X_transformed = X.drop(columns=self.columns_to_drop)


        if self.selected_features is not None:
            features_to_keep = [col for col in
self.selected_features if col != self.target_col]
            X_transformed = X_transformed[features_to_keep]

        return X_transformed
```

**Implementasi  Feature Creator**

```
class FeatureCreator(BaseEstimator, TransformerMixin):
   def __init__(self, num_feats, max_features=50):
        self.num_feats = num_feats
        self.max_features = max_features
        self.important_features = [
            'URLLength',
            'DomainLength',
            'NoOfLettersInURL',
            'NoOfDigitsInURL',
            'CharContinuationRate'
        ]

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

    def transform(self, X):
        X = X.copy()
        feature_count = len(X.columns)


        if feature_count < self.max_features:
            for col in self.important_features:
                if col in X.columns:
                    X[f'{col}_squared'] = X[col] ** 2
                    feature_count += 1
                    if feature_count >= self.max_features:
                        return X


        important_ratios = [
            ('URLLength', 'DomainLength'),
            ('NoOfLettersInURL', 'URLLength'),
            ('NoOfDigitsInURL', 'URLLength'),
            ('NoOfObfuscatedChar', 'URLLength'),
            ('NoOfSpecialCharsInURL', 'URLLength')
        ]

        if feature_count < self.max_features:
```

```
            for num, denom in important_ratios:
                if num in X.columns and denom in X.columns:
                    X[f'{num}_{denom}_ratio'] = X[num] /
(X[denom] + 1e-6)
                    feature_count += 1
                    if feature_count >= self.max_features:
                        return X


        important_interactions = [
            ('URLLength', 'CharContinuationRate'),
            ('DomainLength', 'NoOfSubDomain'),
            ('NoOfLettersInURL', 'NoOfDigitsInURL'),
            ('NoOfSpecialCharsInURL', 'ObfuscationRatio')
        ]

        if feature_count < self.max_features:
            for feat1, feat2 in important_interactions:
                if feat1 in X.columns and feat2 in
X.columns:
                    X[f'{feat1}_{feat2}_interaction'] =
X[feat1] * X[feat2]
                    feature_count += 1
                    if feature_count >= self.max_features:
                        return X

        return X
```

## 4. 2 Data Preprocessing

### 4.2.1. Feature Scaling

Feature scaling adalah teknik praproses data yang penting dalam machine learning untuk memastikan bahwa semua fitur memiliki kontribusi yang setara selama proses pelatihan. Metode feature scaling yang kami gunakan adalah standardscaler, metode standardization (Z-score scaling) yang

menskalakan fitur sehingga memiliki rata-rata 0 dan standar deviasi 1. Formula yang kami gunakan adalah

$$X' = \frac{X - \mu}{\sigma}$$

X: nilai asli fitur.

μ: rata-rata dari fitur.

σ: standar deviasi dari fitur

**Implementasi**

```
class StandardScalerTransformer(BaseEstimator,
TransformerMixin):
    def __init__(self, num_feats):
        self.num_feats = num_feats
        self.scaler = StandardScaler()

    def fit(self, X, y=None):
        self.existing_num_feats = [col for col in
self.num_feats
                                   if col in X.columns and
X[col].dtype.kind in 'biufc']
        if self.existing_num_feats:
            self.scaler.fit(X[self.existing_num_feats])
        return self

    def transform(self, X):
        X = X.copy()
        if self.existing_num_feats:
            data = X[self.existing_num_feats].values
            X[self.existing_num_feats] =
self.scaler.transform(data)
        return X
```

### 4.2.2. Feature Encoding

Feature encoding (atau categorical encoding) adalah proses mengubah data kategorikal (data non-numerik) menjadi format numerik agar dapat digunakan dalam algoritma machine learning. Hal ini disebabkan algoritma membutuhkan data numerik untuk pelatihan dan prediksi. Metode feature encoding yang kami gunakan adalah *One-Hot Encoding*, pengisian kolom nilai biner yang menunjukan kehadiran kategori pada data serta melakukan penambahan kolom baru untuk setiap kategori.

**Implementasi**

```
class LabelEncodingTransformer(BaseEstimator,
TransformerMixin):
    """
    Label encoding: Convert categories to integer labels
    Suitable for ordinal data
    """
    def __init__(self, cat_feats):
        self.cat_feats = cat_feats
        self.label_maps = {}

    def fit(self, X, y=None):
        # Create mapping for each categorical feature
        for col in self.cat_feats:
            if col in X.columns:
                unique_values = X[col].unique()
                self.label_maps[col] = {val: idx for idx, val
in enumerate(unique_values)}
        return self

    def transform(self, X):
        X = X.copy()
        for col, mapping in self.label_maps.items():
            if col in X.columns:
                X[col] = X[col].map(mapping)
        return X
```

### 4.2.3. Handling Imbalanced Dataset

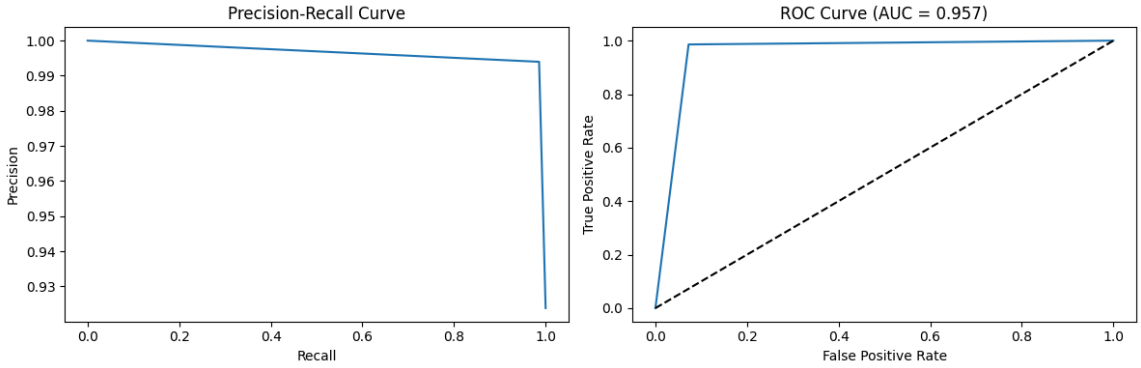
*Handling Imbalanced Dataset* adalah langkah penting sebab data yang tidak seimbang dapat menyebabkan berbagai masalah yang memengaruhi kinerja dan akurasi model machine learning. Metode yang kami gunakan adalah metode resampling dengan menambah jumlah sampel pada kelas minoritas berupa data sintetis berdasarkan interpolasi atau disebut SMOTE (Synthetic Minority Oversampling Technique)

**Implementasi**

```
class ResamplingHandler(BaseEstimator, TransformerMixin):

    def __init__(self, method='smote',
sampling_strategy='auto', random_state=42):
        self.method = method
        self.sampling_strategy = sampling_strategy
        self.random_state = random_state

        if method == 'smote':
            self.sampler = SMOTE(
                sampling_strategy=sampling_strategy,
                random_state=random_state
            )
        elif method == 'undersample':
            self.sampler = RandomUnderSampler(
                sampling_strategy=sampling_strategy,
                random_state=random_state
            )
        elif method == 'combine':
            self.sampler =
SMOTEENN(random_state=random_state)

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

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

```
        if y is not None:
            X_resampled, y_resampled =
self.sampler.fit_resample(X, y)
            return X_resampled, y_resampled
        return X

    def fit_transform(self, X, y):
        return self.fit(X, y).transform(X, y)
```

## 4. 3 Precision-Recall Curve dan ROC Curve




Gambar 4.1. Plot Precision-Recall dan ROC Curve

Melalui plot dari precision-recall, kita dapat melihat bahwa model memiliki precision dan recall yang baik dengan precision mendekati 1 pada sebagian besar recall. Hal ini menandakan model bekerja dengan baik dalam mendeteksi hasil positif tanpa banyak prediksi salah. Penurunan di akhir menandakan bahwa dataset memiliki sedikit kasus kelas positif yang sulit diprediksi. Di sisi kanan, ROC Curve menunjukan nilai Area Under Curve sebesar 0.947 yang mendekati 1. Hal ini berarti model secara akurat memisahkan kelas minoritas dari kelas mayoritas.Lalu, Kurva yang hampir menyentuh sudut kiri atas menunjukkan bahwa model memiliki performa sangat baik dengan keseimbangan yang baik antara false positive dan true positive.

# BAB V

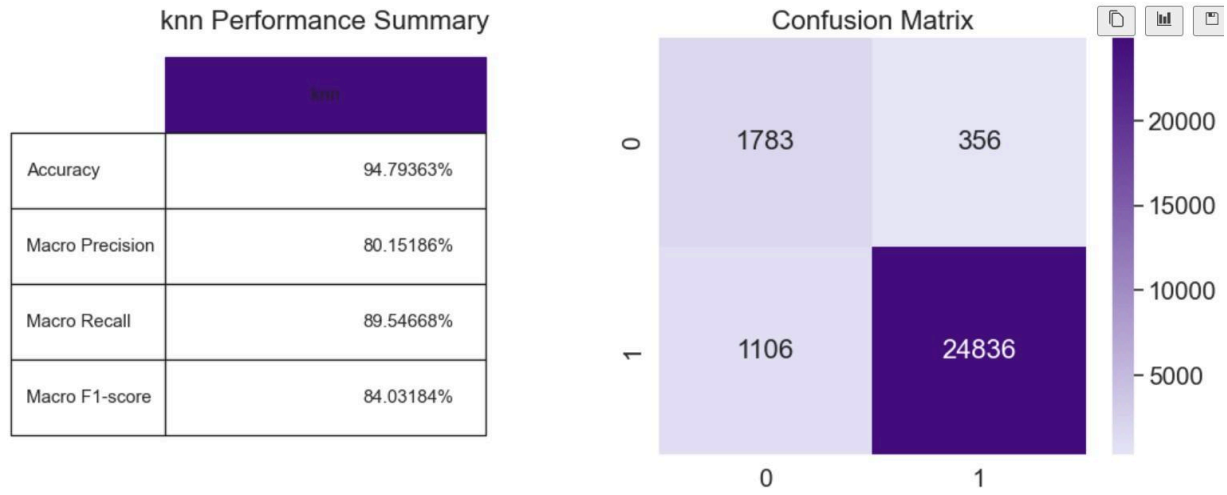
# Perbandingan Hasil Prediksi

## 5.1. KNN :

Menggunakan library :


Gambar 5.1. Gambar performance library

Menggunakan algoritma from scratch :




Gambar 5.2. Performance metrics for KNN from scratch

Berdasarkan hasil tersebut, dapat disimpulkan bahwa perbedaan akurasi antara kedua metode sangat kecil yaitu sekitar 0.9%. Hal ini menunjukkan bahwa implementasi from the scratch dari algoritma KNN sudah akurat dan sesuai dengan hasil yang dihasilkan oleh *library standard*. Perbedaan ini dapat disebabkan oleh beberapa faktor:

a. **Presisi Perhitungan:**

   Perbedaan akurasi dapat terjadi karena perbedaan presisi bilangan desimal pada perhitungan jarak dan proses pengambilan keputusan (voting).

b. **Test case adaptability:**

   Test case telah dibuat sesuai dengan kebutuhan dan variasi pemakaian yang ekstensif sehingga mampu lebih adaptif. Sehingga memberikan performa yang konsisten bagi dataset yang variatif.

c. **Optimasi Library:**

   Library yang digunakan umumnya telah dioptimasi untuk menangani perhitungan sehingga lebih efisien, namun hasil akhirnya tetap mendekati atau sama dengan implementasi manual jika dilakukan dengan benar.

Perbandingan berdasarkan matriks yang telah ditentukan sebagai standar validitas akurasi suatu analisis :

**a. Akurasi :**

Dengan skor akurasi scratch 94.79% dan skor presisi library 94.79% menunjukan bahwa pengaplikasian scratch dan library dapat menghasilkan akurasi yang sama.

**b. Precision :**

Dengan skor presisi scratch 80.15% dan skor presisi library 80.15% menunjukan bahwa pengaplikasian scratch dan library menghasilkan presisi data yang sebanding.

**c. Recall :**

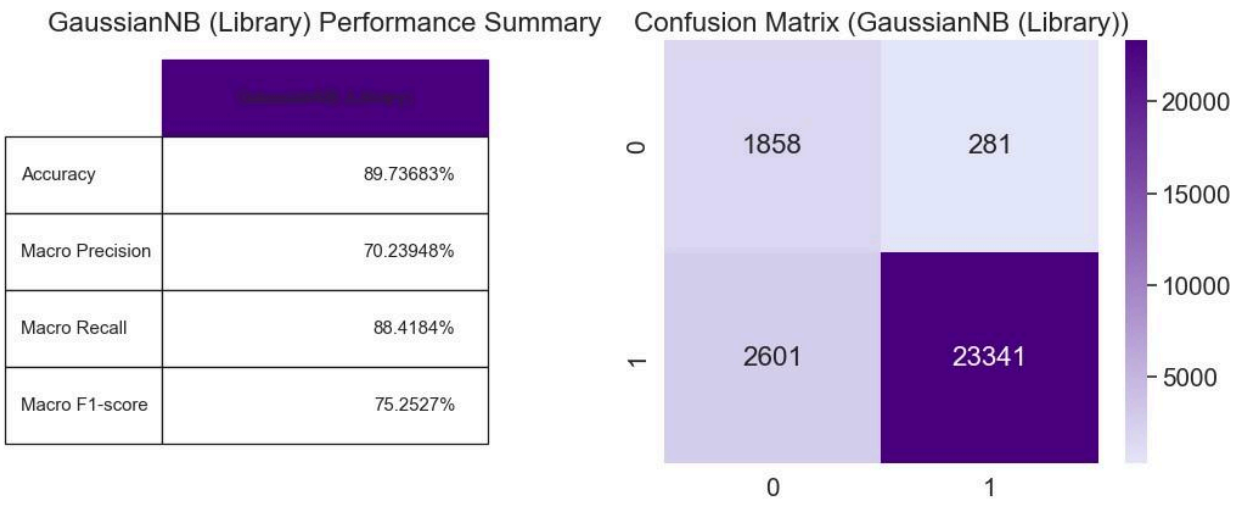
Dengan skor recall scratch 89.54% dan skor recall library 89.54% menunjukan bahwa pengaplikasian scratch dan library dapat menghasilkan hasil yang konsisten, sehingga data sebanding.

**d. F1 Score :**

Dengan skor F1 scratch 84% dan F1 score library 84% menunjukan bahwa pengaplikasian scratch dan library dapat menghasilkan hasil yang konsisten dapat menghasilkan hasil yang konsisten, sehingga data sebanding.

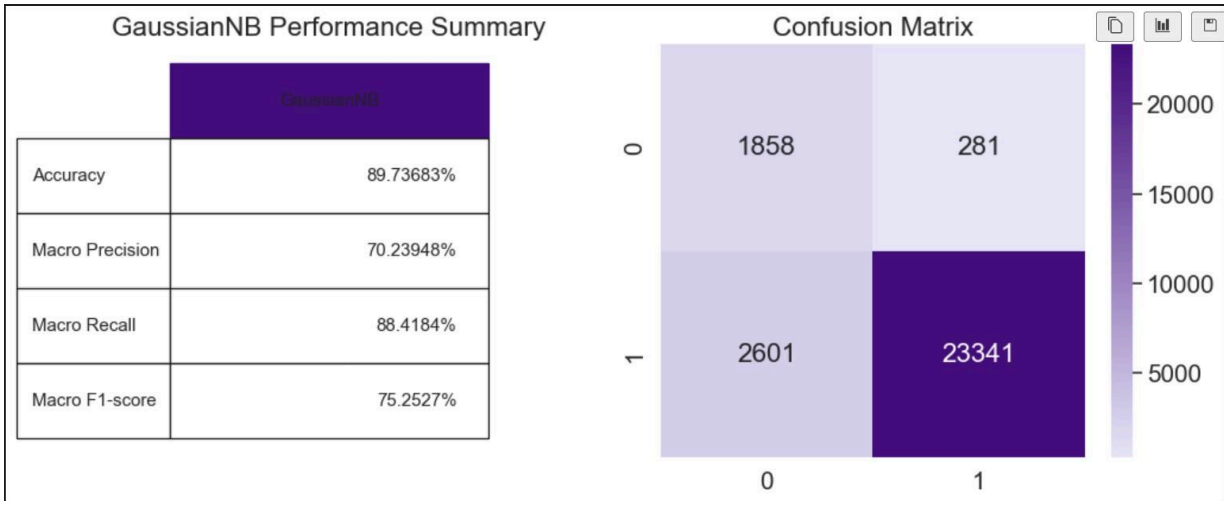
**5.2. GNB :**

Menggunakan library :




Gambar 5.3. Performa GNB library

Menggunakan algoritma from scratch :




Gambar 5.4. Performa GNB scratch

Berdasarkan hasil tersebut, dapat disimpulkan bahwa perbedaan akurasi antara kedua metode juga kecil yaitu sekitar 0.7%. Hal ini menunjukkan bahwa implementasi from the scratch dari algoritma GNB sudah akurat dan sesuai dengan hasil yang dihasilkan oleh *library standard*. Perbedaan ini dapat disebabkan oleh beberapa faktor:

1. **Presisi Perhitungan:**
   Perbedaan akurasi dapat terjadi karena perbedaan presisi bilangan desimal pada perhitungan jarak dan proses pengambilan keputusan (voting).

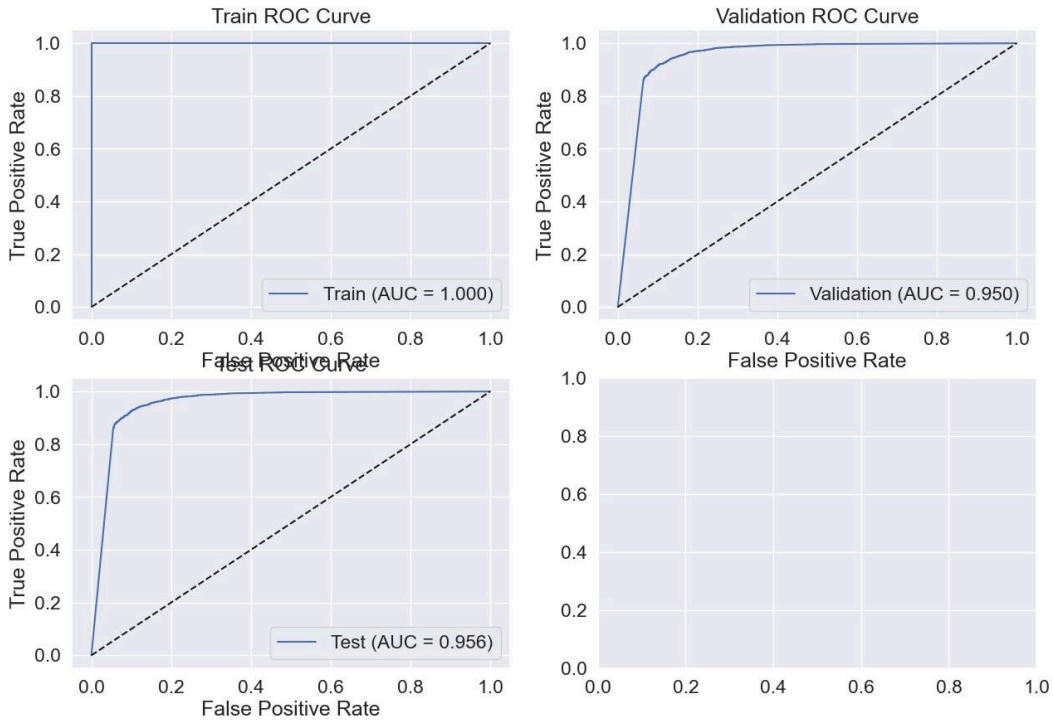
2. **Test case adaptability:**
   Test case telah dibuat sesuai dengan kebutuhan dan variasi pemakaian yang ekstensif sehingga mampu lebih adaptif. Sehingga memberikan performa yang konsisten bagi dataset yang variatif.
3. **Optimasi Library:**
   Library yang digunakan umumnya telah dioptimasi untuk menangani perhitungan sehingga lebih efisien, namun hasil akhirnya tetap mendekati atau sama dengan implementasi manual jika dilakukan dengan benar.

Perbandingan berdasarkan matriks yang telah ditentukan sebagai standar validitas akurasi suatu analisis :

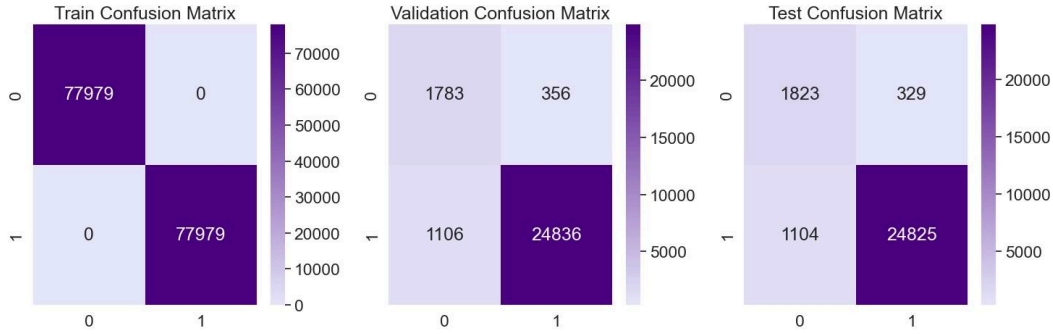
a. **Akurasi :**
   Dengan skor akurasi scratch 89.73% dan skor presisi library 89.73% menunjukan bahwa pengaplikasian scratch dan library dapat menghasilkan akurasi yang sama.
b. **Precision :**
   Dengan skor presisi scratch 70.24% dan skor presisi library 70.24% menunjukan bahwa pengaplikasian scratch dan library menghasilkan presisi data yang sebanding.
c. **Recall :**
   Dengan skor recall scratch 88.42% dan skor recall library 88.42% menunjukan bahwa pengaplikasian scratch dan library menghasilkan recall data yang sebanding.
d. **F1 Score :**
   Dengan skor F1 scratch 75.25% dan F1 score library 75.25% menunjukan bahwa pengaplikasian scratch dan library menghasilkan F1 Score data yang sebanding.

## 5.3. Improvement

**KNN :**




Gambar 5.5. Kumpulan evaluasi grafik KNN dengan ROC Plot




Gambar 5.6. Kumpulan evaluasi confusion matrix KNN oleh 3 dataset yang berbeda

Cross-validation results (5 folds):

Accuracy: 0.978 (+/- 0.002)

Precision: 0.979 (+/- 0.002)

Recall: 0.978 (+/- 0.002)

F1-score: 0.978 (+/- 0.002)

**ROC Curves:**

1. **Train ROC**:

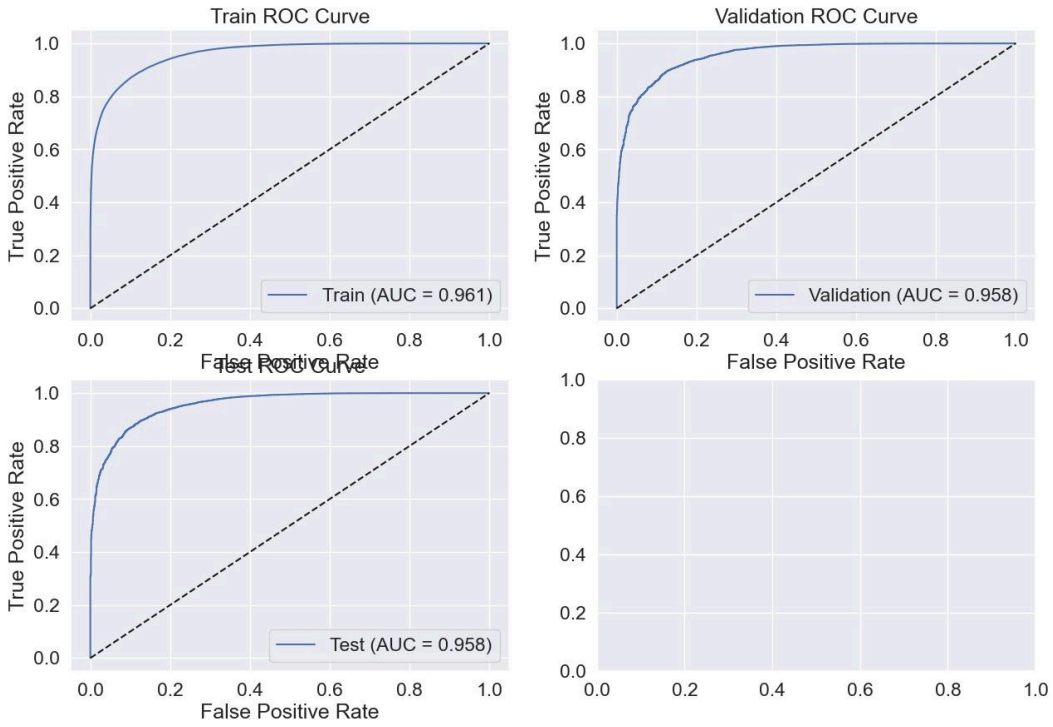
   AUC = 1.000 menunjukkan performa sempurna pada data latih, namun ini bisa mengindikasikan overfitting.

2. **Validation ROC**:
   AUC = 0.950 menunjukkan model bekerja sangat baik pada data validasi.
3. **Test ROC**:
   AUC = 0.956 menunjukkan model mampu melakukan generalisasi dengan baik pada data uji.

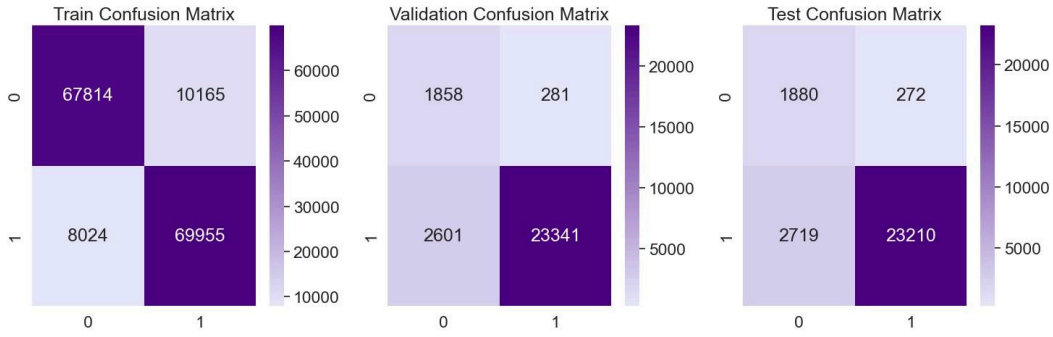
**Confusion Matrix:**

1. **Train**:
   Tidak ada kesalahan prediksi (overfitting). Semua data diprediksi benar.
2. **Validation**:
   - 1.106 **false negatives** (positif salah diprediksi negatif).
   - 356 **false positives** (negatif salah diprediksi positif).
     Kinerja tetap sangat baik.
3. **Test**:
   Mirip dengan validasi:
   - 1.104 **false negatives**.
   - 329 **false positives**.

**GNB :**




Gambar 5.7. Kumpulan evaluasi grafik GNB dengan ROC Plot




Gambar 5.8. Kumpulan evaluasi confusion matrix GNB oleh 3 dataset yang berbeda

Cross-validation results (5 folds):

Accuracy: 0.883 (+/- 0.003)

Precision: 0.884 (+/- 0.003)

Recall: 0.883 (+/- 0.003)

F1-score: 0.883 (+/- 0.003)

**ROC Curve**

Grafik ROC menunjukkan kinerja model pada dataset **Train, Validation**, dan **Test**.

- **AUC (Area Under Curve):**
    - **Train:** 0.961
    - **Validation:** 0.958
    - **Test:** 0.958

Nilai AUC mendekati 1, menunjukkan model memiliki performa yang sangat baik dalam membedakan kelas positif dan negatif di ketiga dataset.

**Confusion Matrix**

Matriks ini menunjukkan jumlah prediksi benar (True Positive/Negative) dan salah (False Positive/Negative) pada dataset **Train, Validation**, dan **Test**.

- Contoh Interpretasi (Validation):
    - **True Negative (1858):** Prediksi "negatif" benar.
    - **False Positive (281):** Prediksi "positif" salah.
    - **False Negative (2601):** Prediksi "negatif" salah.
    - **True Positive (23341):** Prediksi "positif" benar.

# Pembagian Tugas

| NIM | Nama | Pembagian Tugas |
|---|---|---|
| 18222130 | Bryan P. Hutagalung | KNN scratch, sklearn, dan mengerjakan laporan |
| 18222134 | Ardra Rafif Sahasika | GNB scratch, sklearn, dan mengerjakan laporan |
| 18222137 | Timothy Haposan Simanjuntak | KNN scratch, sklearn, dan mengerjakan laporan |
| 18222141 | Yusril Fazri Mahendra | GNB scratch, sklearn, dan mengerjakan laporan |

# Referensi


1. UCI Machine Learning Repository. (n.d.). *Phishing URL dataset*. Retrieved December 22, 2024, from https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset
2. ScienceDirect. (2023). *Phishing URL dataset study*. Retrieved December 22, 2024, from https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub
3. Scikit-learn. (2024). *Neighbors-based classification and regression*. Retrieved December 22, 2024, from https://scikit-learn.org/1.5/modules/neighbors.html
4. Scikit-learn. (2024). *Naive Bayes classification*. Retrieved December 22, 2024, from https://scikit-learn.org/1.5/modules/naive_bayes.html
5. Built In. (n.d.). *Gaussian Naive Bayes: A guide to this machine learning classification technique*. Retrieved December 22, 2024, from https://builtin.com/artificial-intelligence/gaussian-naive-bayes#:~:text=Gaussian%20Naive%20Bayes%20is%20a%20machine%20learning%20classification%20technique%20based,class%20follows%20a%20normal%20distribution