LAPORAN TUGAS BESAR 2

Implementasi Algoritma Pembelajaran Mesin KNN dan Gaussian Naive-Bayes pada dataset PhiUSIIL Phising URL Dataset

Disusun Oleh:

Kelompok 19 (Jamaah AI)

Yasra Zhafirah	(18222002)
Benedicta Eryka Santosa	(18222031)
Kerlyn Deslia Andeskar	(18222090)
Dahayu Ramaniya Aurasindu	(18222099)

IF3070 Dasar Inteligensi Artifisial

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024/2025

DAFTAR ISI

PENJELASAN IMPLEMENTASI KNN

KNN atau *K-Nearest Neighbors* adalah salah satu algoritma dalam *supervised machine learning* yang memanfaatkan jarak untuk membuat klasifikasi atau prediksi tentang pengelompokkan dari *data point*. Metrik jarak antar data point yang tersedia pada kode ini adalah Euclidean, Manhattan, dan Minkowski. User dapat memilih salah satu dari ketiga metrik ini untuk diterapkan pada modeling KNN. Selain itu, user juga dapat memasukkan jumlah tetangga/*neighbors* paling dekat (k) yang diinginkan. Berikut adalah tahapan umum dalam melakukan algoritma KNN.

1. Menerima masukan parameter k yang diinginkan
2. Menghitung jarak (Euclidean, Manhattan, atau Minkowski) suatu *data point* dari data yang ingin diprediksi (data *testing*) terhadap seluruh data *training* yang digunakan dengan masing-masing rumus untuk metrik jarak yang diinginkan
3. Mengurutkan data *training* berdasarkan jarak dari *data point* yang telah dihitung, mulai dari yang terdekat hingga terjauh
4. Mengambil sebanyak k data terdekat yang sudah diurutkan, lalu lihat kategori atau label dari masing-masing data tersebut. Kategori ini adalah hasil klasifikasi dari data tetangga terdekat
5. Dari kategori-kategori yang ditemukan pada tahap 4, kategori yang paling sering muncul (mayoritas) akan digunakan sebagai hasil prediksi untuk data *testing*

Berikut adalah kode yang kami pakai untuk membuat class KNN *from scratch* di file py yang terpisah dengan file notebook utama.

```
class KNN:
    def __init__(self, k: int = 3, distance_metric: str = 'euclidean'):
        """
        Initialize KNN with more flexible distance metric options.

        Parameters:
        -----------
            k (int, optional): Number of nearest neighbors to consider (default=3).
                distance_metric (str, optional): Distance metric to use for neighbor
calculation (default='euclidean', 'manhattan', 'minkowski').
        """
```

```
        self.k = k
        if distance_metric == "manhattan":
            self.distance_metric = "cityblock"
        else:
            self.distance_metric = distance_metric

    def _ensure_ndarray(self, X):
        """
        Ensure the input data is converted to a NumPy ndarray.

        Parameters:
        -----------
            X (pd.DataFrame, csr_matrix, or ndarray): Input data to convert.

        Returns:
        --------
            np.ndarray: Converted data as an ndarray.
        """
        if isinstance(X, pd.DataFrame):
            return X.values
        elif isinstance(X, csr_matrix):
            return X.toarray()
        elif isinstance(X, np.ndarray):
            return X
        if isinstance(X, pd.Series):
            return X.values
        else:
            raise TypeError("Input data must be a DataFrame, csr_matrix, or ndarray.")

    def fit(self, X_train, y_train):
        """
        Store training data after ensuring it's an ndarray.

        Parameters:
        -----------
                X_train (pd.DataFrame, csr_matrix, or ndarray): Feature matrix for
training.
            y_train (pd.Series, or ndarray): Target labels for training.
        """
        self.X_train = self._ensure_ndarray(X_train)
        self.y_train = np.array(y_train)
```

```
    def predict(self, X_test) -> np.ndarray:
        """
        Predict labels for test points in a memory-efficient manner.

        Parameters:
        -----------
            X_test (pd.DataFrame, csr_matrix, or ndarray): Feature matrix for testing

        Returns:
        --------
            np.ndarray: Predicted labels for test points
        """
        X_test = self._ensure_ndarray(X_test)
        predictions = []

        for test_point in X_test:
            distances = cdist([test_point], self.X_train, metric=self.distance_metric)

            k_indices = np.argsort(distances[0])[:self.k]

            k_nearest_labels = self.y_train[k_indices]

            unique_labels, counts = np.unique(k_nearest_labels, return_counts=True)
            most_common_label = unique_labels[np.argmax(counts)]

            predictions.append(most_common_label)

        return np.array(predictions)
```

Berikut adalah kode yang kami pakai untuk mengimplementasikan class KNN di file notebook utama.

```
def test_knn_metrics():
    """Test KNN dengan berbagai metrik jarak"""

    # Load data
    print("Loading data...")
    df = pd.read_csv('../train/train.csv')  # Sesuaikan dengan path file Anda

    # Select features (sesuaikan dengan dataset Anda)
    numerical_features = [
        'URLLength', 'DomainLength', 'CharContinuationRate',
```

```
        'TLDLegitimateProb', 'URLCharProb', 'NoOfSubDomain'
    ]

    X = df[numerical_features]
    y = df['label']

    # Handle missing values
    X = X.fillna(X.mean())

    # Split data
        X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)

    # Convert to DataFrame
    X_train_transformed = pd.DataFrame(X_train_scaled, columns=numerical_features)
    X_val_transformed = pd.DataFrame(X_val_scaled, columns=numerical_features)

    # Test different distance metrics
    metrics = ['euclidean', 'manhattan', 'minkowski']
    results = {}

    for metric in metrics:
        print(f"\nTesting KNN with {metric} distance:")

        # Initialize and train KNN
        knn_model = KNN(k=5, distance_metric=metric)
        knn_model.fit(X_train_transformed, y_train)

        # Save model
        model_filename = f"trained_knn_{metric}.pkl"
        with open(model_filename, 'wb') as f:
            pickle.dump(knn_model, f)
        print(f"Model saved as {model_filename}")

        # Make predictions
        y_val_pred = knn_model.predict(X_val_transformed)
```

```
        # Calculate metrics
        f1_macro = f1_score(y_val, y_val_pred, average='macro')

        print("\nClassification Report:")
        print(classification_report(y_val, y_val_pred))
        print(f'F1 Score (Macro Average) on Validation Set: {f1_macro}')

        results[metric] = {
            'f1_macro': f1_macro,
            'predictions': y_val_pred
        }

    # Compare results
    print("\nComparison of F1 Scores:")
    for metric, result in results.items():
        print(f"{metric}: {result['f1_macro']:.4f}")

    return results

if __name__ == "__main__":
    results = test_knn_metrics()
```

PENJELASAN IMPLEMENTASI NAIVE-BAYES

Naive bayes adalah salah satu algoritma dalam *supervised machine learning* yang menggunakan metode pengklasifikasian berdasarkan probabilitas dan distribusi normal. Berikut adalah tahapan-tahapan dalam implementasinya:

1. Menentukan parameter var smoothing untuk menghindari pembagian dengan nilai variansi yang terlalu kecil
2. Memisahkan data berdasarkan kelas
3. Menghitung probabilitas prior untuk setiap kelas
4. Untuk setiap data uji, diperlukan perhitungan joint log likelihood untuk setiap kelas berdasarkan rata-rata dan variansi yang telah dihitung.
5. Prediksi yang dipakai adalah kelas dengan log likelihood tertinggi

Berikut adalah kode yang kami pakai untuk mengimplementasikan Naive-Bayes *from scratch* yang dibuat terpisah dari file .ipynb utama:

```
import numpy as np
import pandas as pd
from typing import Union

class NaiveBayes:
    def __init__(self, var_smoothing: float = 1e-9):
        """
        Initialize Gaussian Naive Bayes Classifier.

        Parameters:
        -----------
                var_smoothing (float): Portion of the largest variance of all
features added to variances for stability.
        """
        self.var_smoothing = var_smoothing
        self.classes_: np.ndarray = None
        self.class_prior_: np.ndarray = None
        self.theta_: np.ndarray = None
        self.sigma_: np.ndarray = None

    def fit(self, X: Union[pd.DataFrame, np.ndarray], y: Union[pd.Series,
np.ndarray]):
        """
```

```
    Fit Gaussian Naive Bayes according to X, y.

    Parameters:
    -----------
        X (DataFrame or ndarray): Training vectors
        y (Series or ndarray): Target values

    Returns:
    --------
        self: Fitted estimator
    """
    X = X.values if isinstance(X, pd.DataFrame) else X
    y = y.values if isinstance(y, pd.Series) else y

    self.classes_ = np.unique(y)
    n_classes = len(self.classes_)
    n_features = X.shape[1]

    self.theta_ = np.zeros((n_classes, n_features))
    self.sigma_ = np.zeros((n_classes, n_features))
    self.class_prior_ = np.zeros(n_classes)

    for i, c in enumerate(self.classes_):
        X_c = X[y == c]
        self.class_prior_[i] = X_c.shape[0] / X.shape[0]

        self.theta_[i, :] = X_c.mean(axis=0)

        var = X_c.var(axis=0)

        var_max = np.max(var) if len(var) > 0 else 1.0
        self.sigma_[i, :] = var + self.var_smoothing * var_max

    return self

def _joint_log_likelihood(self, X: np.ndarray) -> np.ndarray:
    """
    Calculate joint log likelihood.

    Parameters:
    -----------
        X (ndarray): Input samples
```

```
            Returns:
            --------
                ndarray: Joint log likelihood
            """
            joint_log_likelihood = []
            for i in range(len(self.classes_)):
                jointi = np.log(self.class_prior_[i])

                n_ij = -0.5 * np.sum(np.log(2. * np.pi * self.sigma_[i, :]))
                        n_ij -= 0.5 * np.sum(((X - self.theta_[i, :]) ** 2) /
(self.sigma_[i, :]), axis=1)

                joint_log_likelihood.append(jointi + n_ij)

            joint_log_likelihood = np.array(joint_log_likelihood).T
            return joint_log_likelihood

        def predict(self, X: Union[pd.DataFrame, np.ndarray]) -> np.ndarray:
            """
            Perform classification on an array of test vectors X.

            Parameters:
            -----------
                X (DataFrame or ndarray): Input samples

            Returns:
            --------
                ndarray: Predicted class label for X
            """
            X = X.values if isinstance(X, pd.DataFrame) else X

            jll = self._joint_log_likelihood(X)

            return self.classes_[np.argmax(jll, axis=1)]
```

Dan berikut adalah implementasi Naive Bayes di *notebook* utama:

```
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
```

```
import pickle

from NaiveBayes import NaiveBayes

def test_naive_bayes():
    """Test Naive Bayes implementations (from scratch vs sklearn)"""

    # Load data
    print("Loading data...")
    df = pd.read_csv('../train/train.csv')

    # Select features
    numerical_features = [
        'URLLength', 'DomainLength', 'CharContinuationRate',
        'TLDLegitimateProb', 'URLCharProb', 'NoOfSubDomain'
    ]

    X = df[numerical_features]
    y = df['label']

    # Handle missing values
    X = X.fillna(X.mean())

    # Split data
      X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)

    # Convert to DataFrame
                    X_train_transformed     =     pd.DataFrame(X_train_scaled,
columns=numerical_features)
    X_val_transformed = pd.DataFrame(X_val_scaled, columns=numerical_features)

    print("\nTesting Our Naive Bayes Implementation:")
    # Our implementation
    nb_scratch = NaiveBayes()
    nb_scratch.fit(X_train_transformed, y_train)
```

```
# Save model
with open("trained_nb_scratch.pkl", 'wb') as f:
    pickle.dump(nb_scratch, f)
print("Model saved as trained_nb_scratch.pkl")

# Make predictions
y_pred_scratch = nb_scratch.predict(X_val_transformed)

# Calculate metrics
f1_macro_scratch = f1_score(y_val, y_pred_scratch, average='macro')

print("\nClassification Report (Our Implementation):")
print(classification_report(y_val, y_pred_scratch))
print(f'F1 Score (Macro Average) on Validation Set: {f1_macro_scratch}')

print("\nTesting Scikit-learn Naive Bayes:")
# Scikit-learn implementation
nb_sklearn = GaussianNB()
nb_sklearn.fit(X_train_transformed, y_train)

# Make predictions
y_pred_sklearn = nb_sklearn.predict(X_val_transformed)

# Calculate metrics
f1_macro_sklearn = f1_score(y_val, y_pred_sklearn, average='macro')

print("\nClassification Report (Scikit-learn):")
print(classification_report(y_val, y_pred_sklearn))
print(f'F1 Score (Macro Average) on Validation Set: {f1_macro_sklearn}')

# Compare results
print("\nComparison of F1 Scores:")
print(f"Our Implementation: {f1_macro_scratch:.4f}")
print(f"Scikit-learn: {f1_macro_sklearn:.4f}")

results = {
    'scratch': {
        'f1_macro': f1_macro_scratch,
        'predictions': y_pred_scratch
    },
    'sklearn': {
        'f1_macro': f1_macro_sklearn,
```

```
            'predictions': y_pred_sklearn
        }
    }

    return results

if __name__ == "__main__":
    results = test_naive_bayes()
```

PENJELASAN TAHAP CLEANSING DAN PREPROCESSING

Data cleansing adalah salah satu tahapan dalam *machine learning* yang bertujuan untuk memperbaiki kesalahan, ketidakakuratan, atau ketidakkonsistenan data. Dalam proses ini, terdapat beberapa rangkaian pengolahan data, yaitu:

1. Menangani Data yang Hilang

   Untuk menangani data-data yang hilang (*missing values*), kami menggunakan strategi berdasarkan jenis kolom yang ada.

   a. Untuk menangani kolom yang berhubungan dengan Domain, IsDomainIP diasumsikan 0 apabila tidak terdapat informasi lain.
   b. Untuk kolom yang berisi data biner 0 dan 1, nilai yang hilang akan diisi dengan nilai yang sering muncul (*most frequent*).
   c. Untuk menangani kolom yang berisi data numerik, nilai yang hilang akan diisi dengan median dari masing-masing kolom.
   d. Untuk menangani kolom yang berhubungan dengan Title, kolom yang berisi judul kosong akan diisi dengan “unknown”, dan kolom HasTitle akan diisi 1 apabila kolom Title tidak “unknown”.
   e. Untuk menangani kolom yang berisi data kategorikal, nilai yang hilang akan diisi dengan nilai yang sering muncul.

2. Menangani Data *Outliers*

   Untuk mendeteksi *outlier* data, kami menggunakan metode IQR (Interquartile Range) untuk mengetahui data yang berada di luar rentang batas bawah (Q3+(1.5×IQR)) dan batas atas (Q1−(1.5×IQR)). Nilai yang lebih kecil dari batas bawah akan diisi dengan nilai batas bawah, dan nilai yang lebih besar dari batas atas akan diisi dengan batas atas.

3. Menghilangkan Data Duplikat

   Data duplikat dihapus dari *dataset* menggunakan fungsi drop_duplicates().

4. Rekayasa Fitur (*Features Engineering*)

   Untuk mengurangi redundansi, kami memangkas fitur-fitur yang memiliki korelasi tinggi.

Data preprocessing adalah proses lanjutan dari *data cleansing*, dimana setelah data dibersihkan, data akan diproses agar formatnya dapat diterima oleh *machine learning*. Dalam proses ini, terdapat beberapa rangkaian pengolahan data, yaitu:

1. *Feature Scaling*

   Feature Scaling adalah teknik *preprocessing* yang digunakan untuk menstandarisasikan rentang variabel independen sehingga dapat memastikan bahwa kolom yang berisi data numerik memiliki *scale* yang sama.

2. *Encoding Categorical Variables*

   Model *machine learning* biasanya dapat bekerja dengan baik apabila menggunakan data numerik, sehingga kolom yang berisi data kategorikal perlu di-*encoded*. Kami menggunakan *One Hot Encoder* karena pada data yang kami pakai tidak terdapat urutan atau tatanan tertentu antar kategori.

3. *Handling Imbalanced Classes*

   Dalam menangani kelas yang *imbalanced*, kami menggunakan teknik *undersampling*, yaitu mengurangi jumlah baris dalam kelas mayoritas untuk menyeimbangkan dataset.

PERBANDINGAN HASIL PREDIKSI DARI ALGORITMA *FROM SCRATCH* DENGAN HASIL YANG DIDAPATKAN DARI SCIKIT-LEARN

1. Hasil Prediksi KNN

```
=== Testing Our KNN Implementation ===
Our model saved as knn_scratch.pkl

Classification Report (Our Implementation):
              precision    recall  f1-score   support

           0       0.83      0.45      0.58      2152
           1       0.96      0.99      0.97     25929

    accuracy                           0.95     28081
   macro avg       0.89      0.72      0.78     28081
weighted avg       0.95      0.95      0.94     28081

F1 Score (Macro Average): 0.778526

=== Testing Scikit-learn KNN ===

Classification Report (Scikit-learn):
              precision    recall  f1-score   support

           0       0.83      0.45      0.58      2152
           1       0.96      0.99      0.97     25929

    accuracy                           0.95     28081
   macro avg       0.89      0.72      0.78     28081
weighted avg       0.95      0.95      0.94     28081

F1 Score (Macro Average): 0.778303
```

From Scratch **: 0.778526**

scikit-learn **: 0.778303**

Perbandingan hasil antara algoritma KNN secara keseluruhan, yang menggunakan rata-rata jarak Euclidean, Manhattan, dan Minkowski pada implementasi manual dan *library* sklearn, menunjukkan hasil yang serupa dengan akurasi 0.95 dan F1 Score (Macro Average) 0.78. Hal ini membuktikan bahwa perhitungan yang dipakai pada jarak Euclidean, Manhattan, dan Minkowski dan pemilihan tetangga terdekat (k=5) pada implementasi sudah sesuai dengan pustaka. Dari sini, dapat disimpulkan bahwa implementasi KNN yang kita buat sendiri berjalan sedikit **lebih baik**.

2. **Hasil Prediksi Naive-Bayes:**

```
Testing Our Naive Bayes Implementation:
Model saved as trained_nb_scratch.pkl

Classification Report (Our Implementation):
              precision    recall  f1-score   support

           0       0.51      0.42      0.46      2152
           1       0.95      0.97      0.96     25929

    accuracy                           0.92     28081
   macro avg       0.73      0.69      0.71     28081
weighted avg       0.92      0.92      0.92     28081

F1 Score (Macro Average) on Validation Set: 0.7081338877910482

Testing Scikit-learn Naive Bayes:

Classification Report (Scikit-learn):
              precision    recall  f1-score   support

           0       0.51      0.42      0.46      2152
           1       0.95      0.97      0.96     25929

    accuracy                           0.92     28081
   macro avg       0.73      0.69      0.71     28081
weighted avg       0.92      0.92      0.92     28081

F1 Score (Macro Average) on Validation Set: 0.7081338877910482

Comparison of F1 Scores:
Our Implementation: 0.7081
Scikit-learn: 0.7081
```

From Scratch **: 0.7081**

scikit-learn **: 0.7081**

Perbandingan hasil antara Naive Bayes yang diimplementasikan dari nol (scratch) dengan menggunakan library scikit-learn, yang ditunjukkan pada gambar di atas, mengindikasikan bahwa akurasi dan F1 Score (Macro Average) kedua model tersebut sama. Akurasi untuk model scratch adalah 0.92, sama dengan model library, sementara nilai F1 Score Macro Average juga identik, yaitu 0.7081 untuk keduanya. Insight yang

diperoleh menunjukkan bahwa performa kedua model konsisten, terutama dalam mengidentifikasi kelas mayoritas seperti "Generic" dan "Normal". Perbedaan kecil yang ada mungkin disebabkan oleh perbedaan dalam implementasi teknis, seperti penanganan probabilitas kecil dan smoothing pada model library GaussianNB. Secara keseluruhan, implementasi Naive Bayes berjalan dengan baik dan menghasilkan hasil yang sebanding dengan *library*.

KONTRIBUSI ANGGOTA KELOMPOK

NIM	Nama	Pembagian Tugas
18222002	Yasra Zhafirah	Data cleaning, Data preprocessing, Data modelling, Laporan
18222031	Benedicta Eryka Santosa	Data cleaning, Data preprocessing, Data modelling, Laporan
18222090	Kerlyn Deslia Andeskar	Data cleaning, Data preprocessing, Data modelling, Laporan
18222099	Dahayu Ramaniya Aurasindu	Data cleaning, Data preprocessing, Data modelling, Laporan

REFERENSI

1.9. Naive Bayes — scikit-learn 1.5.2 documentation. (n.d.). Scikit-learn. Retrieved December 16, 2024, from https://scikit-learn.org/1.5/modules/naive_bayes.html

1.6. Nearest Neighbors — scikit-learn 1.5.2 documentation. (n.d.). Scikit-learn. Retrieved December 16, 2024, from https://scikit-learn.org/1.5/modules/neighbors.html

Prasad, A., & Chandra, S. (2024, January). *PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning*. Retrieved December 16, 2024, from https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub

UC Irvine Machine Learning Repository. (2024, March 3). *PhiUSIIL Phishing URL (Website)*. Retrieved December 16, 2024, from https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset