Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial
Implementasi Algoritma Pembelajaran Mesin

Disusun oleh Kelompok 5:

Adinda Khairunnisa Indraputri	18222104
Chairul Nur Wahid	18222132

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

2024

DAFTAR ISI

BAB I
DESKRIPSI PERSOALAN

Tugas ini bertujuan untuk membuat algoritma *machine learning*, yaitu *K-Nearest Neighbors* (KNN) dan *Gaussian Naive Bayes*, untuk memprediksi apakah sebuah URL termasuk kategori aman (*legitimate*) atau berbahaya (*phishing*). Dataset yang digunakan adalah PhiUSIIL Phishing URL Dataset, yang berisi fitur-fitur dari URL dan source code halaman web.

Algoritma dibuat dari awal (*from scratch*) agar lebih memahami cara kerjanya. Hasilnya akan dibandingkan dengan implementasi menggunakan pustaka scikit-learn untuk mengevaluasi performa kedua pendekatan tersebut.

BAB II
PENJELASAN SINGKAT IMPLEMENTASI KNN.

K-Nearest Neighbors (KNN) adalah algoritma pembelajaran mesin yang menentukan kelas suatu data berdasarkan kelas mayoritas dari sejumlah tetangga terdekatnya. Dalam algoritma ini, langkah pertama adalah memilih nilai k, yang menentukan jumlah tetangga terdekat yang akan digunakan untuk prediksi. Selain itu, metrik jarak seperti Euclidean, Manhattan, atau Minkowski juga harus ditentukan untuk menghitung jarak antar data.

Setelah parameter tersebut ditentukan, jarak antara data uji dan semua data pelatihan dihitung menggunakan metrik yang dipilih. Data uji kemudian diklasifikasikan ke dalam kelas yang paling sering muncul di antara k tetangga terdekatnya. Proses ini diulangi untuk setiap data uji dalam dataset sehingga semua data memiliki prediksi kelas masing-masing. Algoritma ini sederhana tetapi efektif, terutama untuk dataset dengan struktur non-linear.

```python
# KNN Implementation
class KNN:
    def __init__(self, k=3, distance_metric='euclidean'):
        self.k = k
        self.distance_metric = distance_metric
        self.X_train = None
        self.y_train = None

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

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

    def _predict(self, x):
        distances = self._calculate_distances(x)
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

    def _calculate_distances(self, x):
        if self.distance_metric == 'euclidean':
            distances = np.sqrt(np.sum((self.X_train - x) ** 2, axis=1))
        elif self.distance_metric == 'manhattan':
            distances = np.sum(np.abs(self.X_train - x), axis=1)
        elif self.distance_metric == 'minkowski':
            p = 3  # Default for Minkowski distance
            distances = np.sum(np.abs(self.X_train - x) ** p, axis=1) ** (1 / p)
        else:
            raise ValueError("Unsupported distance metric. Choose 'euclidean', 'manhattan', or
'minkowski'.")
        return distances

    def save_model(self, file_path):
        with open(file_path, 'wb') as f:
            pickle.dump(self, f)

    @staticmethod
    def load_model(file_path):
        with open(file_path, 'rb') as f:
            return pickle.load(f)

# Accuracy Calculation
def calculate_accuracy(y_true, y_pred):
    correct = sum(yt == yp for yt, yp in zip(y_true, y_pred))
    return correct / len(y_true)

# Example Pipeline
if 'engineered_df' not in globals():
    raise ValueError("Variabel 'engineered_df' belum didefinisikan. Pastikan dataset Anda sudah ada di
variabel ini.")

# Preprocessing
engineered_df_processed, encoders, scaler = preprocess_data(engineered_df)

# Split dataset
X = engineered_df_processed.drop(columns=['label']).values  # Features
y = engineered_df_processed['label'].values  # Labels
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Use a subset for faster testing
subset_size = 500  # Adjust this size based on your system's speed
X_train_small, y_train_small = X_train[:subset_size], y_train[:subset_size]
X_test_small, y_test_small = X_test[:subset_size], y_test[:subset_size]

# Instantiate KNN
knn = KNN(k=3, distance_metric='euclidean')

# Train and Predict
knn.fit(X_train_small, y_train_small)
y_pred_small = knn.predict(X_test_small)

# Evaluate Accuracy
accuracy_small = calculate_accuracy(y_test_small, y_pred_small)
print(f"Accuracy on subset: {accuracy_small * 100:.2f}%")
```

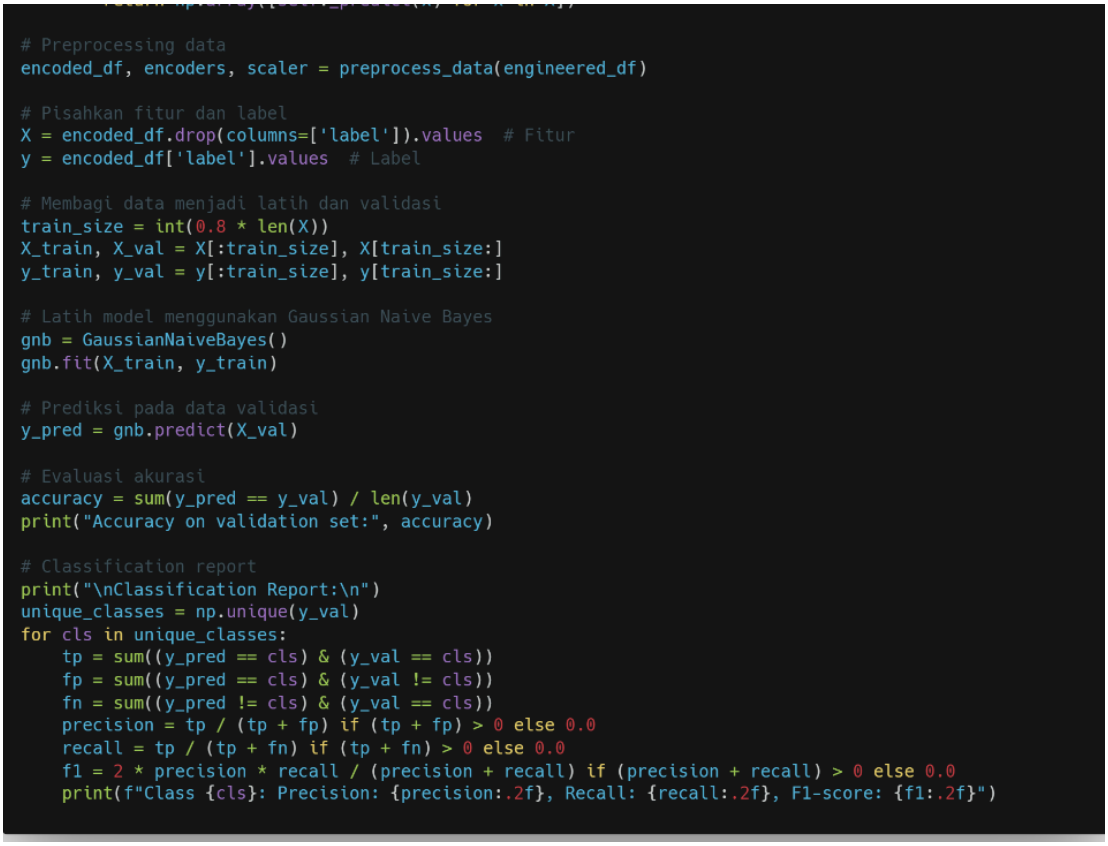
Implementasi KNN diinisialisasi dengan parameter k untuk menentukan jumlah tetangga terdekat dan distance_metric untuk memilih metode penghitungan jarak, seperti Euclidean, Manhattan, atau Minkowski. Data train disimpan menggunakan metode fit tanpa proses pelatihan eksplisit, karena KNN hanya memanfaatkan data pelatihan saat melakukan prediksi. Prediksi dilakukan dengan menghitung jarak setiap sampel uji ke semua sampel pelatihan, memilih k tetangga terdekat berdasarkan jarak, dan menentukan kelas mayoritas dari tetangga tersebut. Evaluasi performa model dilakukan dengan menghitung akurasi, yaitu perbandingan antara label asli dan hasil prediksi. Selain itu, implementasi ini mendukung pengujian pada subset data menggunakan parameter subset_size, yang memungkinkan pengujian cepat dengan data yang lebih kecil tanpa memproses seluruh dataset.

BAB III
PENJELASAN SINGKAT IMPLEMENTASI NAIVE-BAYES

Gaussian Naive Bayes (GNB) adalah algoritma klasifikasi yang menggunakan prinsip Teorema Bayes dengan asumsi bahwa setiap fitur bersifat independen satu sama lain. Algoritma ini sangat berguna untuk memprediksi probabilitas suatu data masuk ke dalam suatu kelas tertentu berdasarkan data pelatihan. Langkah pertama dalam implementasi GNB adalah menghitung statistik dasar seperti mean dan variansi untuk setiap fitur dalam masing-masing kelas. Statistik ini digunakan untuk membentuk distribusi Gaussian yang mewakili setiap fitur dalam kelas tertentu.

Setelah distribusi Gaussian terbentuk, algoritma menghitung probabilitas posterior untuk setiap kelas. Probabilitas ini didapatkan dengan menggabungkan probabilitas dari setiap fitur menggunakan distribusi Gaussian yang sesuai. Dengan menghitung probabilitas ini, algoritma dapat menentukan kelas dengan probabilitas tertinggi sebagai prediksi untuk data uji.

```python
class GaussianNaiveBayes:
    def __init__(self):
        self.classes = None
        self.mean = None
        self.var = None
        self.priors = None

    def fit(self, X, y):
        """
        Menghitung rata-rata, varians, dan prior probabilities untuk setiap kelas.
        """
        if len(X) != len(y):
            raise ValueError("Length of X and y must be the same.")

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

        # Initialize mean, variance, and priors
        self.mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self.var = np.zeros((n_classes, n_features), dtype=np.float64)
        self.priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self.classes):
            X_c = X[y == c]
            self.mean[idx, :] = X_c.mean(axis=0)
            self.var[idx, :] = X_c.var(axis=0)
            self.priors[idx] = X_c.shape[0] / float(X.shape[0])

    def _gaussian_density(self, class_idx, x):
        """
        Menghitung probabilitas densitas Gaussian.
        """
        mean = self.mean[class_idx]
        var = self.var[class_idx]
        numerator = np.exp(- (x - mean) ** 2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator

    def _predict(self, x):
        """
        Menghitung log posterior probabilities dan mengembalikan kelas dengan nilai tertinggi.
        """
        posteriors = []

        for idx, c in enumerate(self.classes):
            prior = np.log(self.priors[idx]) if self.priors[idx] > 0 else -np.inf
            class_conditional = np.sum(np.log(self._gaussian_density(idx, x) + 1e-9))  # Add epsilon to
avoid log(0)
            posterior = prior + class_conditional
            posteriors.append(posterior)

        return self.classes[np.argmax(posteriors)]

    def predict(self, X):
        """
        Melakukan prediksi pada data baru.
        """
        return np.array([self._predict(x) for x in X])
```



```python
# Preprocessing data
encoded_df, encoders, scaler = preprocess_data(engineered_df)

# Pisahkan fitur dan label
X = encoded_df.drop(columns=['label']).values  # Fitur
y = encoded_df['label'].values  # Label

# Membagi data menjadi latih dan validasi
train_size = int(0.8 * len(X))
X_train, X_val = X[:train_size], X[train_size:]
y_train, y_val = y[:train_size], y[train_size:]

# Latih model menggunakan Gaussian Naive Bayes
gnb = GaussianNaiveBayes()
gnb.fit(X_train, y_train)

# Prediksi pada data validasi
y_pred = gnb.predict(X_val)

# Evaluasi akurasi
accuracy = sum(y_pred == y_val) / len(y_val)
print("Accuracy on validation set:", accuracy)

# Classification report
print("\nClassification Report:\n")
unique_classes = np.unique(y_val)
for cls in unique_classes:
    tp = sum((y_pred == cls) & (y_val == cls))
    fp = sum((y_pred == cls) & (y_val != cls))
    fn = sum((y_pred != cls) & (y_val == cls))
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0.0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0.0
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0
    print(f"Class {cls}: Precision: {precision:.2f}, Recall: {recall:.2f}, F1-score: {f1:.2f}")
```

Implementasi Gaussian Naive Bayes diinisialisasi dengan atribut untuk menyimpan rata-rata (mean), varians (var), kelas unik (classes), dan probabilitas prior (priors) untuk setiap kelas. Pada tahap pelatihan, metode fit menghitung statistik dasar seperti rata-rata dan varians setiap fitur untuk masing-masing kelas, serta probabilitas prior berdasarkan proporsi data pada kelas tersebut. Probabilitas densitas Gaussian dihitung menggunakan rumus distribusi normal untuk setiap fitur, yang berfungsi sebagai probabilitas kondisi. Prediksi dilakukan dengan metode _predict, yang menghitung log-probabilitas posterior untuk setiap kelas dengan menjumlahkan log-prior dan log-probabilitas kondisi. Kelas dengan nilai posterior tertinggi dipilih sebagai hasil prediksi. Metode predict kemudian digunakan untuk memproses semua data validasi. Evaluasi dilakukan dengan menghitung akurasi prediksi, serta menghasilkan laporan klasifikasi berupa precision, recall, dan F1-score untuk setiap kelas. Pendekatan ini memastikan model mampu menangani data yang telah melalui preprocessing dengan baik dan memberikan prediksi yang konsisten serta akurat.

BAB IV
PENJELASAN TAHAP CLEANING DAN PREPROCESSING

1. Handling Missing Data

Data yang hilang diatasi dengan beberapa strategi:

- Imputasi berdasarkan relasi dengan fitur lain:
- Jika kolom URL tersedia, Domain diisi dengan domain yang diekstrak dari URL.
- Jika Domain tersedia tetapi URL kosong, URL dibentuk berdasarkan Domain dan protokol HTTPS atau HTTP.
- Kolom lainnya, seperti DomainLength, URLLength, TLD, dan fitur terkait URL, diisi menggunakan nilai yang dihitung berdasarkan data yang tersedia.
- Untuk nilai kosong di kolom FILENAME, nama file unik dihasilkan secara acak.
- Imputasi menggunakan SimpleImputer:
- Kolom numerik diisi dengan median
- Kolom kategori diisi dengan modus

2. Dealing with Outliers

penanganan outlier dilakukan menggunakan metode Clipping, yang memotong nilai-nilai ekstrem berdasarkan batas bawah (persentil ke-1) dan batas atas (persentil ke-99). Sebelum penanganan, beberapa fitur seperti URLLength dan CharContinuationRate memiliki jumlah outlier yang signifikan, masing-masing sebanyak 2702 dan 2423 outlier. Setelah clipping, sebagian besar fitur berhasil ditangani dengan jumlah outlier menjadi 0, menunjukkan efektivitas metode ini dalam mengurangi pengaruh nilai ekstrem tanpa mengubah distribusi data asli. Namun, beberapa fitur seperti URLCharProb masih memiliki outlier karena distribusi yang sangat ekstrem, yang mungkin memerlukan metode tambahan untuk penyempurnaan.

3. Removing Duplicates

Data duplikat dihapus untuk memastikan setiap baris dalam dataset adalah unik. Pada dataset ini, tidak ditemukan baris yang duplikat.

4. Feature Engineering

Menambah fitur URL_Domain_Ratio, URLLength_Square, URLLength_Cube, ,URLLength_Bin dengan keterangan <20 adalah *short*, 20 diantara 40 adalah *medium*, dan diatas 40 adalah *long*, dan SecureKeyword yang mengecek apakah Domain mengandung kata-kata terkait keamanan, seperti "secure", "login", "bank", atau "safe".

5. Feature Scaling

Pendekatan ini menggunakan kelas FeatureScaler, sebuah transformer kustom yang mendukung standard scaling dan Min-Max scaling. Dalam tugas ini, metode standard scaling dipilih karena lebih efektif dalam menangani data dengan distribusi yang tidak terbatas (misalnya, nilai yang bisa menjadi sangat besar atau kecil).

6. Feature Encoding

Feature encoding dilakukan untuk mengubah data kategorikal menjadi format numerik yang dapat diproses oleh model pembelajaran mesin. Dalam tugas ini, digunakan one-hot encoding untuk fitur nominal seperti TLD dan URLLength_Bin, karena fitur ini tidak memiliki urutan tertentu, sehingga encoding ini mencegah asumsi hubungan antar kategori. Selain itu, label encoding diterapkan pada fitur ordinal seperti SecureKeyword, yang memiliki urutan tertentu. Encoding ini dipilih karena lebih sederhana dan efektif untuk fitur dengan hierarki yang jelas. Proses encoding ini bertujuan untuk memastikan semua fitur, baik numerik maupun kategorikal, dapat digunakan oleh algoritma pembelajaran mesin tanpa bias terhadap kategori tertentu.

7. Handling Imbalanced Dataset

Handling imbalanced datasets dilakukan untuk memastikan model tidak bias terhadap kelas mayoritas. Dalam tugas ini, digunakan tiga metode utama: oversampling untuk menambah data pada kelas minoritas, undersampling untuk mengurangi data kelas mayoritas, dan SMOTE untuk membuat data sintetis pada kelas minoritas. SMOTE dipilih karena lebih efektif menyeimbangkan data tanpa duplikasi langsung, sehingga model dapat mempelajari pola kedua kelas dengan lebih baik dan meningkatkan performa pada prediksi kelas minoritas.

BAB V
PERBANDINGAN HASIL PREDIKSI

1. **KNN**

Berikut adalah hasil prediksi dari KNN *from scratch*

```
Model saved to knn_model.pkl
Accuracy on subset: 97.50%

Precision, Recall, F1-Score, and Support per class:
Class -3: Precision: 0.96, Recall: 0.72, F1-Score: 0.82, Support: 64
Class 0: Precision: 0.98, Recall: 1.00, F1-Score: 0.99, Support: 736
```

Sedangkan, berikut merupakan hasil dari library scikit-learn.

```
Accuracy with scikit-learn: 98.53%

Classification Report:
              precision    recall  f1-score   support

          -3       0.96      0.85      0.90      1721
           0       0.99      1.00      0.99     20744

    accuracy                           0.99     22465
   macro avg       0.97      0.92      0.95     22465
weighted avg       0.98      0.99      0.98     22465
```

2. **Naive-Bayes**

Berikut adalah hasil prediksi dari Naive-Bayes *from scratch*

```
  numerator = np.exp(- (x - mean) ** 2 / (2 * var))
Accuracy on validation set: 0.07660805697752059

Classification Report:

Class -3.5076606912319: Precision: 0.08, Recall: 1.00, F1-score: 0.14
Class 0.28509028894946975: Precision: 0.00, Recall: 0.00, F1-score: 0.00
```

Sedangkan, berikut merupakan hasil dari library scikit-learn.

```
Accuracy on validation set: 0.9692410416202982

Classification Report:

              precision    recall  f1-score   support

           0       0.79      0.82      0.80      1721
           2       0.99      0.98      0.98     20744

    accuracy                           0.97     22465
   macro avg       0.89      0.90      0.89     22465
weighted avg       0.97      0.97      0.97     22465
```

KONTRIBUSI ANGGOTA

NIM	Nama	Kontribusi
18222104	Adinda Khairunnisa Indraputri	Laporan, KNN, Preprocessing
18222132	Chairul Nur Wahid	Laporan, Data Cleaning