

IF3070 Dasar Intelegensi Artifisial

Tugas Besar 2



Disusun oleh:

Kelompok 24 (2ez4us)

Rajendra Farras Rayhan / 18222105

Lina Azizah R.H. / 18222107

Gracya Tio Damena S. / 18222110

M. Kasyfil Aziz / 18222127

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

Daftar Isi	2
1. Implementasi K-Nearest Neighbour	3
2. Implementasi Gaussian Naive-Bayes	5
3. Cleaning and Preprocessing	11
3.1. Data Cleaning	11
3.2. Data Preprocessing	13
4. Perbandingan Hasil dan Analisis	18
4.1. Hasil Prediksi Algoritma K-Nearest Neighbour (KNN)	18
4.2. Hasil Prediksi Algoritma Gaussian Naive-Bayes	18
4.3. Perbandingan Hasil Algoritma KNN dengan Gaussian Naive-Bayes	18
5. Kontribusi Anggota	19
6. Referensi	20

1. Implementasi K-Nearest *Neighbour*

K-Nearest Neighbor (KNN) adalah algoritma *supervised learning* yang mengklasifikasikan objek berdasarkan kedekatannya dengan objek lain dalam *dataset*. Algoritma K-Nearest Neighbor bekerja dengan mencari sejumlah k tetangga terdekat (*Nearest Neighbors*) dari data yang akan diprediksi. Kemudian, menentukan kelas dari data tersebut berdasarkan mayoritas kelas dari tetangga terdekatnya.

Nama Fungsi	<pre>def __init__(self, n_neighbors=5, metric='euclidean'):</pre>
Deskripsi	<p>Konstruktor kelas yang digunakan untuk menginisialisasi variabel-variabel yang akan menyimpan parameter model.</p> <ul style="list-style-type: none">- <code>self.n_neighbors</code>: Menyimpan jumlah tetangga terdekat yang akan digunakan untuk prediksi.- <code>self.metric</code>: Menyimpan metode perhitungan jarak yang digunakan.- <code>self.X_train</code>: Menyimpan data fitur dari dataset latihan.- <code>self.y_train</code>: Menyimpan label dari dataset latihan.
Source Code	<pre>def __init__(self, n_neighbors=5, metric='euclidean'): self.n_neighbors = n_neighbors self.metric = metric self.X_train = None self.y_train = None</pre>

Nama Fungsi	<pre>def fit(self, X_train, y_train):</pre>
Deskripsi	<p>Fungsi ini bertujuan untuk melatih model dengan menyimpan data fitur (<code>X_train</code>) dan labelnya (<code>y_train</code>).</p>
Source Code	<pre>def fit(self, X_train, y_train): self.X_train = X_train</pre>

	<code>self.y_train = y_train</code>
--	-------------------------------------

Nama Fungsi	<code>def _calculate_distance(self, x1, x2):</code>
Deskripsi	Fungsi ini bertujuan untuk menghitung jarak antara dua data, yaitu data uji dan data latihan, berdasarkan metrik yang dipilih.
Source Code	<pre>def _calculate_distance(self, x1, x2): if self.metric == 'euclidean': return np.sqrt(np.sum((x1 - x2) ** 2)) elif self.metric == 'manhattan': return np.sum(np.abs(x1 - x2)) elif self.metric == 'minkowski': p = 3 return np.sum(np.abs(x1 - x2) ** p) ** (1 / p) else: raise ValueError("Unsupported metric. Choose 'euclidean', 'manhattan', or 'minkowski'.")</pre>

Nama Fungsi	<code>def predict(self, X_test):</code>
Deskripsi	<p>Fungsi ini bertujuan untuk memprediksi label dari data uji dengan langkah-langkah berikut:</p> <ul style="list-style-type: none"> - Menghitung jarak antara setiap data uji dengan seluruh data latihan. - Menentukan tetangga terdekat berdasarkan jarak terpendek. - Mengambil label dari tetangga-tetangga tersebut dan menentukan label mayoritas sebagai hasil prediksi.
Source Code	<pre>def predict(self, X_test): predictions = [] for x in X_test:</pre>

	<pre> distances = [self._calculate_distance(x, x_train) for x_train in self.X_train] nearest_indices = np.argsort(distances)[:self.n_neighbors] nearest_labels = [self.y_train[i] for i in nearest_indices] predictions.append (np.bincount(nearest_labels).argmax()) return np.array(predictions) </pre>
--	---

Nama Fungsi	def save_model(self, filepath):
Deskripsi	Fungsi ini bertujuan untuk menyimpan model KNN yang telah dilatih ke dalam file menggunakan format serialisasi dari pickle.
Source Code	<pre> def save_model(self, filepath): with open(filepath, 'wb') as f: pickle.dump(self, f) </pre>

Nama Fungsi	def load_model(filepath):
Deskripsi	Fungsi ini bertujuan untuk memuat model KNN yang telah disimpan sebelumnya dari file menggunakan pickle.
Source Code	<pre> def load_model(filepath): with open(filepath, 'rb') as f: return pickle.load(f) </pre>

2. Implementasi Gaussian Naive-Bayes

Naive Bayes merupakan algoritma klasifikasi berbasis probabilitas yang didasarkan oleh Teorema Bayes dengan mengasumsikan bahwa fitur-fitur dalam data bersifat independen satu sama lain (*naive assumption*). Teorema Bayes mendefinisikan hubungan antara probabilitas *posterior* dan probabilitas *prior* sebagai berikut:

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

Keterangan: $P(C | X)$: Probabilitas data B termasuk dalam kelas A (probabilitas *posterior*)

$P(X | C)$: Probabilitas data A termasuk dalam kelas B (*likelihood*)

$P(C)$: Probabilitas awal kelas A (probabilitas *prior*)

$P(X)$: Probabilitas data B (*evidence*)

Gaussian Naive Bayes adalah varian dari Naive Bayes dimana setiap fitur pada data memiliki distribusi Gaussian (normal). *Likelihood* dihitung sebagai:

$$P(B | A) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(B-\mu)^2}{2\sigma^2}}$$

Keterangan: $P(B | A)$: Probabilitas data A termasuk dalam kelas B (*likelihood*)

μ : Mean fitur untuk kelas tertentu

σ^2 : Variansi fitur untuk kelas tertentu

Nama Fungsi	<code>def __init__(self):</code>
--------------------	----------------------------------

Deskripsi	<p>Konstruktor kelas yang digunakan untuk menginisialisasi variabel-variabel yang akan menyimpan parameter model.</p> <ul style="list-style-type: none"> - <code>self.classes</code>: Menyimpan daftar kelas unik dalam data target. - <code>self.mean</code>: Menyimpan rata-rata (<i>mean</i>) dari setiap fitur untuk setiap kelas. - <code>self.variance</code>: Menyimpan variansi dari setiap fitur untuk setiap kelas. - <code>self.priors</code>: Menyimpan probabilitas <i>prior</i> untuk setiap kelas.
Source Code	<pre>def __init__(self): self.classes = None self.mean = None self.var = None self.priors = None</pre>

Nama Fungsi	<code>def fit(self, features, labels)</code>
Deskripsi	<p>Fungsi ini bertujuan untuk melatih model dengan melakukan penginisiasian mean, variansi, dan probabilitas <i>prior</i> serta menghitung parameter (mean dan variansi) serta probabilitas <i>prior</i> setiap kelas dalam data. Pada fungsi ini juga dilakukan pengidentifikasian kelas unik yang ada dalam data.</p>
Source Code	<pre>def fit(self, features, labels) self.classes = np.unique(labels) n_classes = len(self.classes) n_samples, n_features = features.shape # inisiasi mean, variansi, dan priors self.mean = np.zeros((n_classes, n_features), dtype=np.float64)</pre>

	<pre> self.var = np.zeros((n_classes, n_features), dtype=np.float64) self.priors = np.zeros(n_classes, dtype=np.float64) for class_idx, class_label in enumerate(self.classes): class_samples = features[labels == class_label] self.mean [class_idx, :] = class_samples.mean(axis=0) self.var [class_idx, :] = class_samples.var(axis=0) self.priors [class_idx] = class_samples.shape[0] / float(n_samples) </pre>
--	--

Nama Fungsi	<pre> def gaussian_pdf (self, class_idx, feature_value): </pre>
Deskripsi	<p>Fungsi ini menghitung nilai <i>Probability Density Function</i> dari distribusi Gaussian untuk fitur suatu kelas. Fungsi ini akan mengembalikan hasil probabilitasnya dengan menggunakan rumus</p> $P(B A) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(B-\mu)^2}{2\sigma^2}}$ <p>Fungsi ini membutuhkan parameter indeks kelas yang akan dihitung probabilitas gaussian fiturnya dan nilai fiturnya.</p>
Source Code	<pre> def gaussian_pdf (self, class_idx, feature_value): mean = self.mean [class_idx] var = self.var [class_idx] exp = np.exp(-((feature_value - mean) ** 2) / (2 * var)) </pre>

	<pre>return (1 / np.sqrt(2 * np.pi * var)) * exp</pre>
--	--

Nama Fungsi	def posterior (self, sample, class_idx):
Deskripsi	<p>Fungsi ini bertujuan untuk menghitung probabilitas posterior untuk <i>sample</i> tertentu dari suatu kelas. Perhitungannya menggunakan logaritma untuk menghindari masalah <i>underflow</i>. Perhitungan dilakukan dengan menghitung <i>log prior</i> dan <i>log likelihood</i> lalu akan mengembalikan hasil probabilitas <i>posterior</i> dengan menggunakan rumus:</p> $\log P(A B) = \log P(A) + \sum_{i=1}^n \log P(x_i A)$
Source Code	<pre>def posterior (self, sample, class_idx): log_prior = np.log(self.priors [class_idx]) log_likelihood = np.sum(np.log([self.gaussian_pdf(class_idx, feature_value) for feature_value in sample])) return log_prior + log_likelihood</pre>

Nama Fungsi	<code>def predict(self, samples):</code>
Deskripsi	Fungsi ini bertujuan untuk memprediksi kelas untuk tiap sampel dan setelah itu melakukan evaluasi untuk mendapatkan kelas dengan probabilitas <i>posterior</i> terbesar menggunakan dengan menggunakan fungsi <code>argmax</code> .
Source Code	<pre>def predict(self, samples): predictions = [] # menghitung posterior untuk tiap kelas dan cari # posterior terbesar for sample in samples: posteriors = [self.posterior(sample, idx) for idx in range(len(self.classes))] predicted_class_idx = np.argmax(posteriors) predictions.append (self.classes[predicted_class_idx]) return np.array(predictions)</pre>

Nama Fungsi	<code>def score(self, test_features, true_labels):</code>
Deskripsi	Fungsi ini digunakan untuk menghitung akurasi model dengan membandingkan label prediksi dengan label asli. Fungsi ini akan mengembalikan persentase prediksi yang benar.
Source Code	<pre>def score(self, test_features, true_labels): predicted_labels = self.predict(test_features) accuracy = np.sum(predicted_labels == true_labels) / len(true_labels) return accuracy</pre>

3. *Cleaning and Preprocessing*

3.1. *Data Cleaning*

3.1.1. *Handling Missing Data*

Nama Metode	Column (Feature) Deletion
Deskripsi	Pada kolom FILENAME dilakukan column deletion karena fitur ini tidak berpengaruh secara signifikan terhadap label sehingga bisa diabaikan atau dihapus untuk mengurangi noise atau kompleksitas algoritma.
Source Code	<pre># Menghapus kolom FILENAME df = df.drop(columns=['FILENAME'])</pre>

Nama Metode	Domain Knowledge
Deskripsi	Semua yang berkaitan dengan URL dapat diisi missing valuenya dengan saling melengkapi informasi yang tersedia. Fitur-fitur tersebut antara lain URL, URLLength, Domain, DomainLength, IsDomainIP, TLD, CharContinuationRate, TLDLength, NoOfSubDomain, NoOfLettersInURL, LetterRatioInURL, NoOfDigitsInURL, DigitRatioInURL, NoOfEqualsInURL, NoOfQMarkInURL, NoOfAmpersandInURL, NoOfOtherSpecialCharsInURL, SpacialCharRatioInURL, IsHTTPS, DomainTitleMatchScore, dan URLTitleMatchScore
Langkah-langkah	URL bisa direkonstruksi dari Domain dan IsHTTPS, jika ada informasi yang tidak tersedia akan diisi 'unknown'. URLLength diambil dari panjang URL. Domain bisa diambil dari Domain pada URL. DomainLength adalah panjang dari Domain. IsDomainIP diambil dari Domain itu IP atau bukan. TLD diambil dari TLD pada Domain. CharContinutatioRate diambil dari rumus

	$\frac{(\text{panjang-segmen-domain-terpanjang} + 1)}{(\text{panjang-domain-tanpa-TLD})}$ TLDLength diambil dari panjang TLD.
--	---

Nama Metode	Mean, Median, Mode Imputation
Deskripsi	Mode Imputation dilakukan pada kolom 'HasObfuscation', 'NoOfObfuscatedChar', 'ObfuscationRatio', 'LineOfCode', 'LargestLineLength', 'HasFavicon', 'Robots', 'IsResponsive', 'NoOfURLRedirect', 'NoOfSelfRedirect', 'HasDescription', 'NoOfPopup', 'NoOfiFrame', 'HasExternalFormSubmit', 'HasSocialNet', 'HasSubmitButton', 'HasHiddenFields', 'HasPasswordField', 'Bank', 'Pay', 'Crypto', 'HasCopyrightInfo', 'NoOfImage', 'NoOfCSS', 'NoOfJS', 'NoOfSelfRef', 'NoOfEmptyRef', 'NoOfExternalRef', sedangkan Mean Imputation dilakukan pada kolom TLDLegitimateProb. Lalu Median Imputation dilakukan pada URLCharProb
Source Code	<pre> # Hitung rata-rata TLDLegitimateProb berdasarkan TLD tld_prob_mean = df.groupby('TLD')['TLDLegitimateProb'].mean() # Rata-rata global untuk fallback global_mean_tld_prob = df['TLDLegitimateProb'].mean() # Fungsi untuk mengisi TLDLegitimateProb def fill_tld_legitimate_prob(row): if pd.isnull(row['TLDLegitimateProb']): # Jika TLDLegitimateProb kosong if row['TLD'] == 'unknown': # Jika TLD adalah 'unknown' </pre>

```

        return 0 # Isi dengan 0
        elif row['TLD'] in tld_prob_mean and not
pd.isnull(tld_prob_mean[row['TLD']]): # Jika TLD
memiliki rata-rata
            return tld_prob_mean[row['TLD']] #
Isi dengan rata-rata TLD
        elif row['TLD'] in tld_prob_mean and
pd.isnull(tld_prob_mean[row['TLD']]): # Jika TLD
ada tapi rata-rata null
            return global_mean_tld_prob # Isi
dengan rata-rata global sebagai fallback
        else:
            return global_mean_tld_prob #
Fallback jika semua kondisi gagal
            return row['TLDLegitimateProb'] # Biarkan
nilai asli jika sudah ada

# Terapkan fungsi ke DataFrame
df['TLDLegitimateProb'] =
df.apply(fill_tld_legitimate_prob, axis=1)

# Verifikasi hasil
missing_tld_legitimate_prob =
df['TLDLegitimateProb'].isnull().sum()

# Tampilkan contoh data untuk verifikasi
print("Jumlah nilai null pada TLDLegitimateProb
setelah imputasi:", missing_tld_legitimate_prob)
print(df[['TLD', 'TLDLegitimateProb']].head(20))

# Hitung median dari URLCharProb
median_url_char_prob = df['URLCharProb'].median()
# Hitung median

```

```

# Isi missing value pada URLCharProb dengan
median
df['URLCharProb'] =
df['URLCharProb'].fillna(median_url_char_prob)

# Verifikasi hasil
missing_url_char_prob =
df['URLCharProb'].isnull().sum()

# Tampilkan hasil verifikasi
print("Jumlah nilai null pada URLCharProb setelah
imputasi:", missing_url_char_prob)

# Fungsi untuk mengisi missing value menggunakan
mode untuk banyak kolom
def fill_missing_with_modus(df, columns):
    for column in columns:
        if column in df.columns: # Periksa
apakah kolom ada di DataFrame
            mode_value = df[column].mode()[0] #
Hitung mode dari kolom
            df[column] =
df[column].fillna(mode_value) # Isi missing
value dengan mode
    return df

# Daftar kolom yang ingin diisi missing value-nya
menggunakan mode
sis_a_kolom = ['HasObfuscation',
'NoOfObfuscatedChar', 'ObfuscationRatio',
'LineOfCode', 'LargestLineLength', 'HasFavicon',
'Robots', 'IsResponsive', 'NoOfURLRedirect',
'NoOfSelfRedirect', 'HasDescription',
'NoOfPopup', 'NoOfiFrame',

```

	<pre> 'HasExternalFormSubmit', 'HasSocialNet', 'HasSubmitButton', 'HasHiddenFields', 'HasPasswordField', 'Bank', 'Pay', 'Crypto', 'HasCopyrightInfo', 'NoOfImage', 'NoOfCSS', 'NoOfJS', 'NoOfSelfRef', 'NoOfEmptyRef', 'NoOfExternalRef'] # Terapkan fungsi ke DataFrame df = fill_missing_with_modus(df, sisa_kolom) # Verifikasi hasil missing_values_setelah_imputasi = df[sisa_kolom].isnull().sum() # Tampilkan hasil verifikasi print("Jumlah nilai null setelah imputasi:") print(missing_values_setelah_imputasi) </pre>
--	---

3.1.2. *Dealing with Outliers*

Nama Metode	Hapus Outlier Ekstrem
Deskripsi	Oleh karena ada fitur seperti URLLength dan CharContinuationRate apabila bernilai nol maka baris tersebut tidak valid
Source Code	<pre> def remove_zero_values(data, columns): for column in columns: data = data[data[column] != 0] return data # Kolom yang ingin diperiksa dan dihapus jika # nilainya 0 columns_to_check = ['URLLength', 'CharContinuationRate'] </pre>

	<pre> # Hapus baris dengan nilai 0 pada kolom-kolom tersebut untuk train_set dan val_set train_set = remove_zero_values(train_set, columns_to_check) val_set = remove_zero_values(val_set, columns_to_check) plot_distribution(train_set, 'URLLength') plot_distribution(train_set, 'CharContinuationRate') </pre>
--	---

Nama Metode	Capping dengan nilai IQR
Deskripsi	Capping dilakukan di 'URLCharProb', 'LineOfCode', 'LargestLineLength', 'NoOfiFrame', 'NoOfImage', 'NoOfCSS', 'NoOfJS', 'NoOfSelfRef', 'NoOfEmptyRef', dan 'NoOfExternalRef'
Source Code	<pre> # Fungsi untuk melakukan capping outliers menggunakan metode IQR def cap_outliers_iqr(data, columns): for column in columns: Q1 = data[column].quantile(0.25) Q3 = data[column].quantile(0.75) IQR = Q3 - Q1 lower_bound = Q1 - 1.5 * IQR upper_bound = Q3 + 1.5 * IQR # Capping nilai yang berada di luar batas data[column] = data[column].apply(lambda x: lower_bound if x < lower_bound else (upper_bound if x > upper_bound else x) </pre>

	<pre>) return data # Daftar kolom untuk diterapkan metode IQR columns_to_clean = ['URLCharProb', 'LineOfCode', 'LargestLineLength', 'NoOfiFrame', 'NoOfImage', 'NoOfCSS', 'NoOfJS', 'NoOfSelfRef', 'NoOfEmptyRef', 'NoOfExternalRef'] # Bersihkan outliers dengan capping untuk train_set dan val_set train_set = cap_outliers_iqr(train_set, columns_to_clean) val_set = cap_outliers_iqr(val_set, columns_to_clean) </pre>
--	--

Nama Metode	Capping dengan nilai Persentil
Deskripsi	Capping dilakukan di 'URLCharProb', 'LineOfCode', 'LargestLineLength', 'NoOfiFrame', 'NoOfImage', 'NoOfCSS', 'NoOfJS', 'NoOfSelfRef', 'NoOfEmptyRef', dan 'NoOfExternalRef'
Source Code	<pre> def cap_outliers_with_percentile(data, columns, percentile=0.9999): for column in columns: upper_bound = data[column].quantile(percentile) # Capping nilai yang berada di atas batas atas </pre>

	<pre> data[column] = data[column].apply(lambda x: upper_bound if x > upper_bound else x) return data # Kolom yang ingin dicapping dengan percentile tertentu columns_to_cap_percentile = ['NoOfObfuscatedChar', 'ObfuscationRatio', 'NoOfPopup'] # Terapkan capping dengan percentile untuk train_set dan val_set train_set = cap_outliers_with_percentile(train_set, columns_to_cap_percentile) val_set = cap_outliers_with_percentile(val_set, columns_to_cap_percentile) for column in columns_to_cap_percentile: plot_distribution(train_set, column) </pre>
--	--

3.1.3. Removing Duplicates

Deskripsi	Pada tahap ini, dilakukan pengecekan apakah terdapat duplikasi di dalam <i>dataset</i> . Hal ini dilakukan untuk menjaga <i>data integrity</i> dan memastikan keakuratan <i>dataset</i> .
Source Code	<pre> # print the number of duplicates print(f"Number of duplicates in training set: {train_set.duplicated().sum()}") print(f"Number of duplicates in validation set: {val_set.duplicated().sum()}") </pre>

3.1.4. Feature Engineering

Nama Kelas	Binning
Deskripsi	Binning digunakan untuk mengelompokkan nilai-nilai dalam kolom DomainLength menjadi kategori <i>short</i> , <i>average</i> , dan <i>long</i> , memudahkan analisis dan interpretasi data.
Source Code	<pre># Binning DomainLength bins = [0, 10, 50, 93] labels = ['short', 'average', 'long'] df['DomainLength_Binned'] = pd.cut(df['DomainLength'], bins=bins, labels=labels)</pre>

Nama Kelas	VarianceThresholdSelector()
Deskripsi	Metode ini digunakan untuk menghapus fitur yang memiliki nilai varians di bawah <i>threshold</i> tertentu. Fitur yang memiliki nilai varians rendah tidak memberikan informasi cukup untuk model sehingga dengan penghapusannya dapat menyederhanakan data serta meningkatkan performansi model.
Source Code	<pre>class VarianceThresholdSelector(BaseEstimator, TransformerMixin): def __init__(self, threshold=0.1): self.threshold = threshold self.selector = VarianceThreshold(threshold=threshold) def fit(self, X, y=None): return self.selector.fit(X) def transform(self, X): return self.selector.transform(X)</pre>

--	--

3.2. Data Preprocessing

3.2.1. Feature Scaling

Feature scaling adalah teknik *data preprocessing* untuk menstandarisasi skala yang dimiliki *feature-feature* dalam *dataset*. Pada tugas besar ini, kami menggunakan metode Standardization.

Standardization digunakan untuk menyesuaikan skala fitur numerik sehingga semuanya memiliki rata-rata 0 dan standar deviasi 1. Ini penting untuk algoritma seperti KNN yang mengandalkan perhitungan jarak, agar semua fitur memberikan pengaruh yang setara. Meski Naive Bayes tidak terlalu bergantung pada skala, menggunakan *Standardization* dapat membantu jika data memiliki rentang nilai yang sangat berbeda, sehingga perhitungan probabilitas menjadi lebih stabil.

Nama Metode	StandardScaler
Deskripsi	Metode ini mengubah <i>mean</i> di setiap feature menjadi 0 dan mengubah skala <i>feature</i> sehingga memiliki standar deviasi 1. Metode ini digunakan pada <i>dataset</i> yang mengikuti distribusi normal, namun sensitif terhadap <i>outlier</i> .
Source Code	<pre> class FeatureScaler(BaseEstimator, TransformerMixin): def __init__(self, scaling_method='standard'): self.scaling_method = scaling_method self.scaler = None def fit(self, X, y=None): self.numerical_columns = </pre>

```

X.select_dtypes(include=[np.number]).columns

        if self.scaling_method == 'standard':
            self.scaler = StandardScaler()
        else:
            raise ValueError("Invalid
scaling_method. Only 'standard' is supported.")

self.scaler.fit(X[self.numerical_columns])
return self

def transform(self, X):
    X_scaled = X.copy()
    X_scaled[self.numerical_columns] =
self.scaler.transform(X[self.numerical_columns])
    return X_scaled

scaler = FeatureScaler(scaling_method='standard')
X_train_scaled = scaler.fit_transform(train_set)

```

3.2.2. **Feature Encoding**

Feature encoding adalah teknik mengubah *feature categorical* (*non-numeric*) menjadi format numeric sehingga dapat digunakan dalam algoritma *machine learning* yang umumnya membutuhkan *input numeric*. Pada tugas besar ini, kami menggunakan metode *target encoding*.

Target Encoding mengubah kategori menjadi angka dengan cara menggantinya dengan rata-rata target untuk setiap kategori. Hal ini membantu algoritma seperti KNN memahami informasi dari kategori saat menghitung jarak antar data. Untuk Naive Bayes, metode ini juga berguna karena menghasilkan angka yang lebih berguna tanpa

menambah banyak kolom baru, seperti yang terjadi pada One-Hot Encoding.

Nama Metode	<i>Target Encoding</i>
Deskripsi	Metode ini akan menggantikan setiap <i>feature categorical</i> dengan nilai statistik yang dihitung dari <i>target variable</i> , seperti <i>mean</i> . Metode ini digunakan untuk menangani <i>feature</i> yang memiliki kardinalitas tinggi dan mengurangi dimensionalitas <i>dataset</i> . Namun, metode ini dapat menyebabkan <i>overfitting</i> jika diimplementasi pada dataset dengan jumlah sampel kecil.
Source Code	<pre># Target Encoding Class class TargetEncoder(BaseEstimator, TransformerMixin): def __init__(self): self.target_means = {} def fit(self, X, y): self.target_means = { col: X.groupby(col)[y.name].mean() for col in X.select_dtypes(include=['object', 'category']).columns } return self def transform(self, X): X_encoded = X.copy() for col, mapping in self.target_means.items(): X_encoded[col] = X_encoded[col].map(mapping).fillna(0) return X_encoded</pre>

	<pre> encoder = TargetEncoder() y_train = train_set['label'] X_train_encoded = encoder.fit(pd.DataFrame(train_set), y_train).transform(pd.DataFrame(X_train_scaled)) </pre>
--	---

3.2.3. *Handling Imbalanced Dataset*

Handling imbalanced dataset adalah proses mengatasi ketidakseimbangan dalam *dataset* akibat beberapa kelas dalam *dataset* memiliki jumlah sampel yang jauh lebih sedikit daripada kelas-kelas lainnya. Hal ini dilakukan untuk mencegah bias dan mengabaikan kelas-kelas minoritas yang dapat mengurangi akurasi prediksi. Pada tugas besar ini, kami menggunakan *Synthetic Minority Oversampling Technique*.

SMOTE (*Synthetic Minority Oversampling Technique*) membantu menyeimbangkan data dengan menambahkan contoh baru untuk kelas minoritas, sehingga data menjadi lebih rata. Pada KNN, ini membuat *neighbor* terdekat lebih representatif untuk kelas minoritas, dan meningkatkan akurasi prediksi. Untuk Naive Bayes, SMOTE membantu model memahami pola dari kelas minoritas dengan lebih baik, sehingga prediksi untuk kelas tersebut tidak kalah dengan kelas mayoritas.

Nama Metode	Synthetic Minority Oversampling Technique (SMOTE)
Deskripsi	Metode ini akan menghasilkan data sintetis pada kelas minoritas dengan membuat sampel baru berdasarkan interpolasi dari sampel asli, sehingga jumlah data di setiap kelas menjadi lebih seimbang.
Source Code	<pre> class SMOTEHandler: def __init__(self, random_state=42): self.random_state = random_state </pre>

	<pre> self.smote = SMOTE(random_state=self.random_state) def fit_resample(self, X, y): return self.smote.fit_resample(X, y) smote_handler = SMOTEHandler(random_state=42) X_train_resampled, y_train_resampled = smote_handler.fit_resample(X_train_encoded, y_train) </pre>
--	--

3.2.4. *Compile Preprocessing Pipeline*

Deskripsi	
Source Code	

4. Perbandingan Hasil dan Analisis

4.1. Hasil Prediksi Algoritma K-Nearest Neighbour (KNN)

Hasil Algoritma <i>Scratch</i>	<pre>Accuracy: 0.9829065916455967 Classification Report: precision recall f1-score support 0 0.98 0.80 0.88 2152 ... accuracy 0.98 0.98 0.98 28081 macro avg 0.98 0.90 0.93 28081 weighted avg 0.98 0.98 0.98 28081</pre>
Hasil Algoritma Scikit-learn	<pre>Accuracy: 0.9829065916455967 Classification Report: precision recall f1-score support 0 0.98 0.80 0.88 2152 ... accuracy 0.98 0.98 0.98 28081 macro avg 0.98 0.90 0.93 28081 weighted avg 0.98 0.98 0.98 28081</pre>

Dengan menggunakan algoritma *scratch*, skor akurasi algoritma K-Nearest Neighbour (KNN) sebesar 0.98. Sedangkan, ketika menggunakan algoritma Scikit-learn, skor akurasi yang dicapai adalah 0.98 dengan waktu eksekusi 5.7s. Maka, algoritma dengan menggunakan library Scikit-learn berjalan dengan lebih efektif dan efisien.

Library scikit-learn dapat mengoptimalkan algoritma KNN dengan teknik yang lebih canggih dan teruji, sehingga mampu memberikan performa yang lebih cepat dan akurasi yang lebih tinggi. Sementara itu, implementasi algoritma KNN *from scratch* dapat memberikan pemahaman yang lebih mendalam terhadap proses kerja algoritma ini. Namun, memerlukan waktu dan usaha yang lebih besar serta berpotensi menghadapi kesalahan implementasi.

4.2. Hasil Prediksi Algoritma Gaussian Naive-Bayes

Hasil Algoritma Scratch	<pre>from GaussianNaiveBayes import GaussianNaiveBayes import numpy as np # Assuming 'X_train', 'X_test', 'y_train', 'y_test' are defined as numpy arrays # Initialize the custom Gaussian Naive Bayes model gnb = GaussianNaiveBayes() # Fit the model to the training data gnb.fit(X_train, y_train) # Make predictions on the test data y_pred = gnb.predict(X_test) # Calculate the accuracy score print(gnb.score(X_test, y_test)) # Call score() with the test data ✓ 30.6s c:\Users\lilinh\Downloads\test tubes dai\GaussianNaiveBayes.py:38: RuntimeWarning: divide by zero encountered in log log_likelihood = np.sum(np.log([self.gaussian_pdf(class_idx, feature_value) for feature_value in sample])) 0.07517538549196966</pre>
Hasil Algoritma Scikit-learn	<pre>> ✓ # Importing necessary libraries for Gaussian Naive Bayes from sklearn.naive_bayes import GaussianNB from sklearn.metrics import accuracy_score, classification_report, confusion_matrix # Assuming 'X_train', 'X_test', 'y_train', 'y_test' are preprocessed and defined # Initialize the Gaussian Naive Bayes model gnb = GaussianNB() # Fit the model to the training data gnb.fit(X_train, y_train) # Make predictions on the test data y_pred = gnb.predict(X_test) # Calculate the accuracy score accuracy = accuracy_score(y_test, y_pred) # Ensure y_test is defined print(f'Accuracy: {accuracy:.2f}') 38] ✓ 0.1s ... Accuracy: 0.95</pre>

Dengan menggunakan algoritma *scratch*, skor akurasi algoritma Gaussian Naive-bayes sebesar 0.95 dengan waktu eksekusi 0.1s. Sedangkan, ketika menggunakan algoritma Scikit-learn, skor akurasi yang dicapai adalah 0.075 dengan waktu eksekusi 30.6s. Maka, algoritma dengan menggunakan library Scikit-learn berjalan dengan lebih efektif dan efisien.

Dari hasil yang didapatkan antara algoritma from scratch dengan algoritma yang menggunakan library didapatkan bahwa hasil akurasi algoritma from scratch lebih rendah. Hal ini memang mungkin terjadi karena scikit-learn sudah menggunakan optimisasi yang lebih matang dan lebih teruji secara luas. Selain hasil akurasi, performa algoritma scikit-learn memang lebih cepat karena diimplementasikan menggunakan metode yang lebih efisien.

Untuk tujuan pembelajaran dan eksplorasi, pengimplementasian algoritma from scratch bisa memberikan lebih banyak kontrol dalam memahami algoritma

Gaussian Naive Bayes lebih mendalam, tetapi cukup rentan terhadap kesalahan. Di lain sisi, algoritma dengan library scikit-learn biasanya efektif dan efisien digunakan untuk skala besar.

5. Kontribusi Anggota

No.	Nama	NIM	Kegiatan
1.	Rajendra Farras Rayhan	18222105	- Data Cleaning
2.	Lina Azizah R.H.	18222107	- Naive Bayes - Data Cleaning
3.	Gracya Tio Damena S.	18222110	- Naive Bayes
4.	M. Kasyfil Aziz	18222127	- Data Preprocessing - KNN