

**Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial**  
***Implementasi Algoritma Pembelajaran Mesin***



**Disusun oleh:**  
**Kelompok K1-G27**

Alfandito Rais Akbar	/ 18222037
Givari Alfachri	/ 18222045
M. Rafi Dhiyaulhaq	/ 18222069
Muhammad Nurul Hakim	/ 18222097

**Program Studi Sistem dan Teknologi Informasi**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2024**

# Daftar Isi

<b>Daftar Isi.....</b>	<b>2</b>
<b>1. Implementasi Algoritma.....</b>	<b>3</b>
1.1 Implementasi K-Nearest Neighbors (KNN).....	3
1.2 Implementasi Gaussian Naive-Bayes.....	3
<b>2. Data Cleaning dan Preprocessing.....</b>	<b>5</b>
2.1 Data Cleaning.....	5
2.2 Data Preprocessing.....	5
<b>3. Perbandingan Hasil Prediksi Algoritma.....</b>	<b>7</b>
3.1 K-Nearest Neighbors (KNN).....	7
3.2 Gaussian Naive-Bayes.....	8
<b>Kontribusi.....</b>	<b>10</b>
<b>Referensi.....</b>	<b>11</b>

# 1. Implementasi Algoritma

## 1.1 Implementasi *K-Nearest Neighbors* (KNN)

K-Nearest Neighbors (KNN) adalah salah satu algoritma pembelajaran mesin berbasis *instance* yang digunakan untuk klasifikasi dan regresi. KNN bekerja dengan prinsip bahwa data yang mirip cenderung berdekatan dalam ruang fitur. Algoritma ini sangat sederhana namun efektif, karena memanfaatkan jarak antar data untuk membuat prediksi.

Langkah langkah umum algoritma KNN sebagai berikut:

- Menentukan parameter  $k$  (jumlah tetangga terdekat) yang akan digunakan untuk menentukan hasil klasifikasi atau prediksi. Nilai  $k$  ditentukan berdasarkan eksperimen untuk memastikan performa terbaik.
- Menghitung jarak untuk setiap data baru yang ingin diprediksi. Metrik yang umum digunakan yaitu *Euclidean Distance*, *Manhattan Distance*, dan *Minkowski Distance*.
- Menentukan  $k$  tetangga terdekat dengan memilih  $k$  data dari training set yang memiliki jarak terdekat/minimum dengan data baru.

Pada implementasi algoritma ini, KNN dioptimalkan dengan menggunakan struktur data KD-Tree. KD-Tree adalah struktur data yang digunakan untuk mempartisi ruang multidimensi dan mempercepat proses pencarian tetangga terdekat. Cara kerja program ini dimulai dengan membangun KD-Tree menggunakan data *training*. Data diurutkan berdasarkan sumbu tertentu, dan median data digunakan sebagai simpul. Pembagian dilakukan secara rekursif hingga seluruh data diproses. Untuk setiap titik data baru yang akan diprediksi, program mencari  $k$  tetangga terdekat dengan traversal KD-Tree, sehingga pencarian hanya difokuskan pada bagian yang relevan. Selanjutnya, program menentukan label mayoritas dari tetangga terdekat tersebut untuk klasifikasi atau menghitung rata-rata untuk regresi.

## 1.2 Implementasi Gaussian Naive-Bayes

*Gaussian Naive-Bayes* adalah algoritma klasifikasi berbasis probabilitas yang menggunakan asumsi sederhana bahwa setiap fitur bersifat independen satu sama lain dan memiliki distribusi normal. Algoritma ini cocok untuk data kontinu, di mana distribusi fitur dapat diasumsikan mengikuti distribusi normal. Kesederhanaan algoritma ini membuatnya sangat cepat dan efisien untuk digunakan dalam tugas klasifikasi. Formula distribusi normal untuk menghitung probabilitas *likelihood*  $P(x|y)$  sebagai berikut.

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma^2_y}} * \exp\left(-\frac{(x_i - \mu_y)^2}{(2\sigma^2_y)}\right)$$

dimana  $\mu_y$  adalah *mean* dan  $\sigma^2_y$  adalah *variance* dari fitur  $x_i$  untuk kelas  $y$ .

Langkah langkah umum algoritma *Gaussian Naive-Bayes* sebagai berikut:

a. Menghitung parameter distribusi berupa:

- *Prior probability* ( $P(y)$ ): Probabilitas awal setiap kelas dihitung berdasarkan frekuensi kemunculan kelas dalam data training.
- *Mean* ( $\mu_y$ ) : Nilai rata-rata untuk setiap fitur dihitung pada masing-masing kelas.
- *Variance* ( $\sigma^2_y$ ): Variansi setiap fitur dihitung sebagai rata-rata kuadrat selisih nilai fitur dengan mean kelasnya.

b. Melakukan prediksi dengan formula Naive Bayes berupa:

- Probabilitas dihitung menggunakan formula dasar.

$$P(y|X) \propto P(y) * \prod P(x_i|y)$$

- Untuk mencegah underflow, probabilitas dikonversi ke domain algoritmik.

$$\log P(y|X) = \log P(y) + \sum \log P(x_i|y)$$

- Perhitungan log *likelihood*.

$$\log P(x_i|y) = \frac{-0.5 \log(2\pi\sigma^2_y) - (x_i - \mu_y)^2}{2\sigma^2_y}$$

- Prediksi akhir dipilih pada probabilitas tertinggi.

Pada implementasi algoritma ini, *Gaussian Naive-Bayes* bekerja secara sistematis melalui beberapa tahap sesuai dengan yang dijelaskan sebelumnya. Program melatih model menggunakan data *training* dengan menghitung parameter distribusi seperti mean dan variansi untuk setiap fitur dalam masing-masing kelas. Selain itu, probabilitas untuk setiap kelas dihitung berdasarkan distribusi kelas dalam data.

Selanjutnya, program menghitung log *likelihood* untuk setiap kelas menggunakan formula distribusi Gaussian. Setelah semua probabilitas dihitung, program menentukan kelas dengan probabilitas tertinggi sebagai prediksi akhir. Langkah terakhir adalah evaluasi model, di mana akurasi prediksi diuji terhadap label sebenarnya dalam data testing. Seluruh proses ini memastikan efisiensi dan stabilitas numerik dalam menghasilkan prediksi yang akurat.

## **2. Data Cleaning dan Preprocessing**

### **2.1 Data Cleaning**

Dalam menangani data yang kosong, kami melakukan imputasi median terhadap data numerik menggunakan SimpleImputer dengan strategi 'median'. Median dipilih karena tidak banyak terpengaruh oleh outlier dan menjaga distribusi asli data, sehingga cocok untuk data yang tidak berdistribusi normal. Untuk data kategorikal, kami melakukan imputasi konstan dengan fill value berupa 'Unknown' menggunakan SimpleImputer dengan strategi 'constant'. Hal ini akan mempermudah interpretasi data kosong dan memastikan konsistensi dalam dataset.

Dalam menangani outlier, kami mengubah nilai outlier menjadi nilai rata-rata dari keseluruhan data pada fitur numerik menggunakan pendekatan z-score. Nilai yang memiliki z-score lebih besar dari 3 dianggap sebagai outlier dan digantikan dengan nilai rata-rata. Rata-rata dipilih sebagai representasi umum dari data, sehingga membantu menjaga konsistensi dan mencegah bias yang disebabkan oleh nilai ekstrim. Pendekatan ini juga memastikan data tetap berada dalam skala yang wajar tanpa menghilangkan sampel.

Kami menangani data duplikat dengan cara menghapusnya menggunakan fungsi `drop_duplicates()` dari pandas. Penghapusan duplikat dilakukan untuk menghindari pengaruh negatif pada model pembelajaran mesin, seperti overfitting atau bias. Penghapusan duplikat juga membantu menjaga keakuratan dan kualitas dataset.

Kami melakukan transformasi logaritmik pada fitur-fitur numerik, seperti 'URLLength', 'DomainLength', 'NoOfImage', 'NoOfCSS', dan 'NoOfJS'. Transformasi logaritmik membantu menyeimbangkan distribusi data yang memiliki skewness tinggi. Selain itu, kami juga membuat fitur interaksi dengan menghitung rasio dan selisih antara fitur-fitur tertentu, seperti rasio antara 'URLLength' dan 'DomainLength', selisih antara 'NoOfImage' dan 'NoOfCSS', dan rasio antara 'NoOfSelfRef' dan 'NoOfEmptyRef'. Fitur interaksi ini dapat memberikan informasi tambahan tentang hubungan antar fitur yang relevan.

### **2.2 Data Preprocessing**

Dilakukan scaling pada data menggunakan MinMaxScaler untuk memastikan bahwa data numerik berada dalam rentang nilai antara 0 dan 1. Hal ini penting karena banyak algoritma pembelajaran mesin sangat sensitif terhadap skala fitur. MinMaxScaler membantu meningkatkan efisiensi komputasi dengan mereduksi nilai data ke rentang yang lebih kecil, sehingga algoritma dapat berjalan lebih cepat dan stabil.

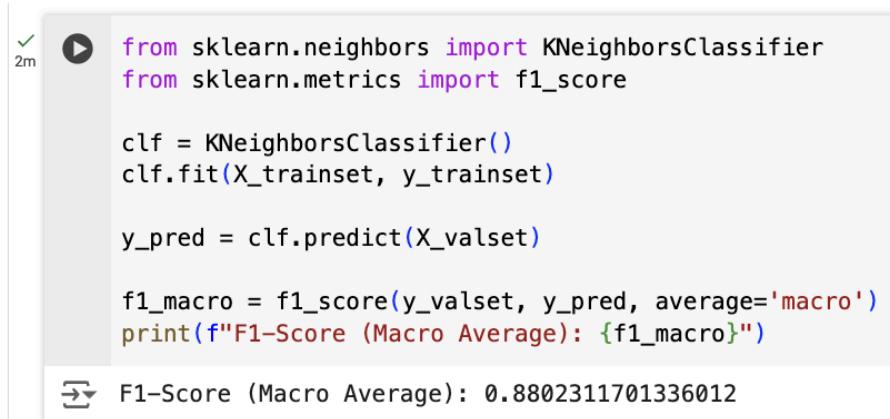
Penggunaan OneHotEncoder dilakukan pada data kategorial untuk mengubah fitur kategorial menjadi representasi biner. OneHotEncoder dipilih karena fitur kategorial pada data ini dianggap tidak memiliki bobot atau urutan yang signifikan. Setiap nilai unik dari fitur kategorial direpresentasikan sebagai vektor biner, di mana hanya satu elemen yang bernilai 1 sementara sisanya bernilai 0. Pendekatan ini memastikan bahwa tidak ada nilai yang dianggap lebih besar atau lebih kecil dari nilai lainnya, sehingga algoritma pembelajaran mesin dapat memperlakukan setiap kategori secara setara.

Imputasi median dan konstan membantu menangani data kosong, penghapusan outlier dengan nilai rata-rata menjaga data tetap dalam skala yang wajar, penghapusan duplikat menjaga keakuratan dataset, feature engineering mengekstraksi informasi yang lebih bermakna, serta scaling dan encoding mempersiapkan data untuk digunakan dalam algoritma.

### 3. Perbandingan Hasil Prediksi Algoritma

#### 3.1 K-Nearest Neighbors (KNN)

Berdasarkan hasil perbandingan yang diperoleh dari prediksi antara algoritma KNN yang diimplementasikan:

A screenshot of a Jupyter Notebook cell. On the left, there is a green checkmark and a play button icon, with '2m' indicating execution time. The code cell contains the following Python code: 

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score

clf = KNeighborsClassifier()
clf.fit(X_trainset, y_trainset)

y_pred = clf.predict(X_valset)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

 Below the code, the output is displayed: 

```
F1-Score (Macro Average): 0.8802311701336012
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score

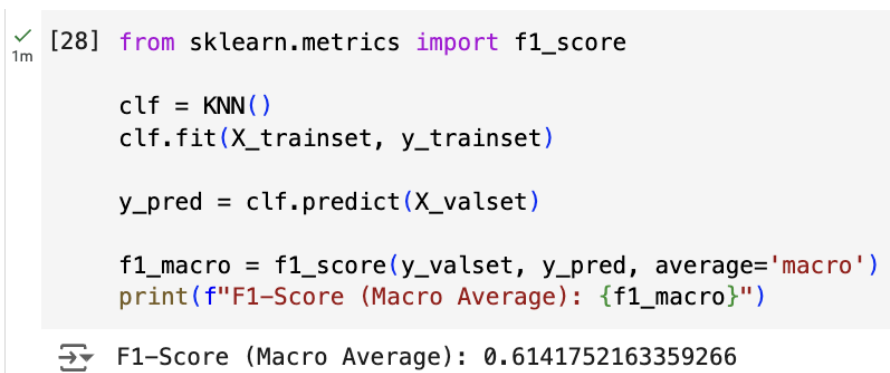
clf = KNeighborsClassifier()
clf.fit(X_trainset, y_trainset)

y_pred = clf.predict(X_valset)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

```
F1-Score (Macro Average): 0.8802311701336012
```

Gambar 1

A screenshot of a Jupyter Notebook cell. On the left, there is a green checkmark and a play button icon, with '1m' indicating execution time. The code cell contains the following Python code: 

```
[28] from sklearn.metrics import f1_score

clf = KNN()
clf.fit(X_trainset, y_trainset)

y_pred = clf.predict(X_valset)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

 Below the code, the output is displayed: 

```
F1-Score (Macro Average): 0.6141752163359266
```

```
[28] from sklearn.metrics import f1_score

clf = KNN()
clf.fit(X_trainset, y_trainset)

y_pred = clf.predict(X_valset)

f1_macro = f1_score(y_valset, y_pred, average='macro')
print(f"F1-Score (Macro Average): {f1_macro}")
```

```
F1-Score (Macro Average): 0.6141752163359266
```

Gambar 2

Perbedaan F1-Score antara kedua implementasi KNN menunjukkan bahwa implementasi dari library scikit-learn memberikan performa yang lebih baik. Hal ini mungkin disebabkan oleh optimasi dan fitur-fitur tambahan yang terdapat pada implementasi scikit-learn, serta perbedaan dalam detail implementasi algoritma.

Meskipun demikian, nilai F1-Score sebesar 0.6141752163359266 masih menunjukkan performa yang cukup baik dalam klasifikasi, meskipun tidak sebaik implementasi dari library.

## 3.2 Gaussian Naive-Bayes

Berdasarkan hasil prediksi algoritma Gaussian Naive Bayes yang diimplementasikan, diperoleh hasil:



```
✓ 1s ▶ from sklearn.naive_bayes import GaussianNB
      from sklearn.metrics import f1_score

      clf = GaussianNB()
      clf.fit(X_trainset, y_trainset)

      y_pred = clf.predict(X_valset)

      f1_macro = f1_score(y_valset, y_pred, average='macro')
      print(f"F1-Score (Macro Average): {f1_macro}")

⇒ F1-Score (Macro Average): 0.14662086218742756

✓ 1s [31] from sklearn.naive_bayes import GaussianNaiveBayes
      clf = GaussianNaiveBayes(var_smoothing=1e-1)
      clf.fit(X_trainset, y_trainset)

      y_pred = clf.predict(X_valset)

      f1_macro = f1_score(y_valset, y_pred, average='macro')
      print(f"F1-Score (Macro Average): {f1_macro}")

⇒ F1-Score (Macro Average): 0.9430940484878532
```

Gambar 3

Perbedaan F1-Score yang signifikan antara kedua implementasi Gaussian Naive Bayes menunjukkan bahwa implementasi dengan `var_smoothing` yang sesuai memberikan performa yang jauh lebih baik dibandingkan dengan implementasi dari library `scikit-learn`.

Nilai F1-Score sebesar 0.9430940484878532 pada implementasi menunjukkan performa yang sangat baik dalam klasifikasi, dengan akurasi dan presisi yang tinggi. Hal ini disebabkan oleh pengaturan `var_smoothing` yang optimal pada implementasi, yang membantu mengatasi masalah estimasi probabilitas yang tidak stabil ketika jumlah sampel pada suatu kelas sangat kecil.

**Secara keseluruhan**, berdasarkan hasil F1-Score yang diberikan, dapat disimpulkan bahwa:

1. Implementasi KNN dari library `scikit-learn` memberikan performa yang lebih baik dengan nilai F1-Score yang lebih tinggi.



2. Implementasi Gaussian Naive Bayes dari scratch dengan `var_smoothing` memberikan performa yang jauh lebih baik dibandingkan dengan implementasi dari library `scikit-learn`, dengan nilai F1-Score yang sangat tinggi.
3. Dalam konteks dataset yang digunakan, algoritma Gaussian Naive Bayes dan pengaturan yang optimal menunjukkan performa terbaik dibandingkan dengan algoritma KNN dan implementasi Gaussian Naive Bayes dari library `scikit-learn`.

## Kontribusi

Nama Lengkap - NIM	Deskripsi Tugas
Alfandito Rais Akbar - 18222037	<ul style="list-style-type: none"><li>- Ikut dalam diskusi bersama</li><li>- Mengerjakan bersama dalam data cleaning &amp; preprocessing</li><li>- Mengerjakan bersama dalam modelling</li><li>- Mengerjakan bersama pada laporan</li></ul>
Givari Al Fachri- 18222045	<ul style="list-style-type: none"><li>- Ikut dalam diskusi bersama</li><li>- Mengerjakan bersama dalam data cleaning &amp; preprocessing</li><li>- Mengerjakan bersama dalam modelling</li><li>- Mengerjakan bersama pada laporan</li></ul>
Muhammad Rafi Dhiyaulhaq - 18222069	<ul style="list-style-type: none"><li>- Ikut dalam diskusi bersama</li><li>- Mengerjakan bersama dalam data cleaning &amp; preprocessing</li><li>- Mengerjakan bersama dalam modelling</li><li>- Mengerjakan bersama pada laporan</li></ul>
Muhammad Nurul Hakim - 18222069	<ul style="list-style-type: none"><li>- Ikut dalam diskusi bersama</li><li>- Mengerjakan bersama dalam data cleaning &amp; preprocessing</li><li>- Mengerjakan bersama dalam modelling</li><li>- Mengerjakan bersama pada laporan</li></ul>

## Referensi

<https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset>

<https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub>

<https://scikit-learn.org/1.5/modules/neighbors.html>

[https://scikit-learn.org/1.5/modules/naive\\_bayes.html](https://scikit-learn.org/1.5/modules/naive_bayes.html)