Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Dipersiapkan oleh

Traienta

Kelompok 30

18222035	Lydia Gracia
18222049	Willhelmina Rachel Silalahi
18222070	Favian Izza Diasputra
18222100	Ervina Limka

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

Daftar Gambar

Daftar Tabel

BAB I
Implementasi Algoritma

1.1. Implementasi Algoritma KNN

Algoritma K-Nearest Neighbors (KNN) adalah metode pembelajaran mesin yang bekerja dengan cara mengidentifikasi titik tetangga terdekat dari data baru. Setelah itu, akan ditentukan kelas atau nilai berdasarkan rata-rata ataupun mayoritas dari tetangga tersebut. KNN bersifat *lazy learning* yang menyebabkan proses pelatihan dan perhitungan dilakukan saat prediksi. Dalam KNN, pengukuran jarak antara titik data berguna untuk menentukan tetangga terdekat.

Untuk mengimplementasi algoritma KNN tersebut, terdapat beberapa metrik jarak yang digunakan antara lain.

1. Jarak Euclidean merupakan pengukuran jarak lurus antara dua titik dalam ruang multidimensi dengan rumus sebagai berikut.

   X = (x1, x2, … , xn) dan *Y = (y1, y2, … , yn)*

   $$d(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2. Jarak Manhattan merupakan pengukuran jarak antara dua titik dengan menjumlahkan perbedaan dari koordinat masing-masing dalam ruang berdimensi tinggi menggunakan rumus berikut.

   $$d(X,Y) = \sum_{i=1}^{n} |X_i - y_i|$$

3. Jarak Minkowski merupakan generalisasi dari jarak Euclidean dan Manhattan menggunakan parameter orde *p* untuk menentukan jenis jarak dengan rumus sebagai berikut.

   $$d(X,Y) = (\sum_{i=1}^{n} \left|x_i - y_i\right|^p)^{\frac{1}{p}}$$

4. Jarak ChebyShev merupakan pengukuran jarak terjauh antara koordinat dari dua titik yang digunakan dalam situasi pergerakan yang dapat dilakukan ke segala arah dengan rumus berikut.

$$d(X, Y) = max_i \left| x_i - y_i \right|$$

Langkah-langkah implementasi dari kode dimulai dengan memilih kolom yang relevan terlebih dahulu yaitu **id, DegitRatioInURL, NoOfOtherSpecialCharsInURL**, dan **DomainTitleMatchScore**. Setelah itu, akan dilakukan pengambilan 500 sampel pertama untuk pelatihan dan validasi. Kemudian, akan dibuat pipeline agar dapat mengotomatisasi proses preprocessing dan pelatihan model. Pipeline yang dilakukan terdiri dari imputasi data hilang menggunakan **SimpleImputer** dengan *strategy* **mean** untuk mengganti nilai yang hilang dengan nilai rata-rata fitur, normalisasi data menggunakan **MinMaxScaler()** untuk menormalisasi fitur dengan rentang 0 hingga 1, menggunakan implementasi KNN dengan jarak Manhattan untuk mengukur kedekatan.

Pipeline yang telah dibuat akan dilatih menggunakan subset data pelatihan dan memprediksi label pada data validasi. Hasil prediksi akan dibandingkan dengan label aktual menggunakan **classification_report** yang terdiri dari:

- Precision: Akurasi prediksi positif
- Recall: kemampuan model mendeteksi semua label positif
- F1-score: rata-rata dari precision dan recall
- Support: jumlah sampel di setiap kelas

Model pipeline yang sudah dilatih akan disimpan ke dalam file menggunakan **joblib** agar dapat digunakan untuk memuat model kembali.

1.2. Implementasi Algoritma Gaussian Naive-Bayes

Algoritma Gaussian Naive-Bayes adalah algoritma dengan asumsi independensi antar fitur berdasarkan Teorema Bayes. Algoritma ini menganggap bahwa ada tidaknya suatu fitur dalam suatu kelas tidak bergantung pada fitur lain sehingga cenderung efektif untuk data kontinu. Implementasi algoritma ini direpresentasikan oleh kelas *GaussianNaiveBayes* pada file naive_bayes.py. Langkah-langkah implementasi dari kode *GaussianNaiveBayes* adalah sebagai berikut.

1. Inisialisasi Kelas

   Pada tahap inisialisasi, terdapat parameter **var_smoothing** yang digunakan untuk menambahkan *smoothing* ke nilai variansi agar dapat menghindari pembagian terhadap nol dan varian yang kecil. Selain itu, terdapat atribut lain yang harus disimpan yaitu:

   - classes_: label unik dalam data
   - class_priors_: probabilitas prior pada setiap kelas
   - features_: jumlah fitur dalam data
   - means_: nilai rata-rata setiap fitur pada kelas
   - vars_: nilai varians setiap fitur pada kelas

```
def __init__(self, var_smoothing: float = 1e-9):
    self.var_smoothing: float = var_smoothing
    self.classes_: np.ndarray = None
    self.class_priors_: np.ndarray = None
    self.features_: int = None
    self.means_: np.ndarray = None
    self.vars_: np.ndarray = None
```

Gambar 1.2.1 Cuplikan Code Inisialisasi Kelas Gaussian Naive Bayes

2. Metode fit

   Metode ini digunakan untuk melatih model menggunakan data train (X, y) dengan langkah-langkah berikut.

   a. Identifikasi kelas unik menggunakan np.unique(y) untuk mendapatkan label kelas unik.
   b. Menginisialisasi atribut **class_priors_** sebagai array untuk menyimpan probabilitas prior setiap kelas dan atribut **means_** & **vars_** sebagai matriks untuk menyimpan nilai rata-rata dan variansi setiap fitur.
   c. Menghitung *smoothing* dengan mengalikan **var_smoothing** dengan varians maksimum setiap fitur untuk stabilitas perhitungan.
   d. Melakukan perhitungan untuk setiap kelas berupa *masking data*, menghitung probabilitas, mean dan varians fitur untuk kelas tersebut.

```python
def fit(self, X: np.ndarray, y: np.ndarray):
    # Get unique class labels
    self.classes_ = np.unique(y)
    n_classes = len(self.classes_)

    # Get number of features
    self.features_ = X.shape[1]

    # Initialize arrays
    self.class_priors_ = np.zeros(n_classes)
    self.means_ = np.zeros((n_classes, self.features_))
    self.vars_ = np.zeros((n_classes, self.features_))

    # Calculate maximum variance for each feature
    var_smoothing_factor = self.var_smoothing * np.max(np.var(X, axis=0))

    # For every class
    for idx, c in enumerate(self.classes_):
        mask = (y == c)
        X_c = X[mask]

        # Compute the prior probability
        self.class_priors_[idx] = np.mean(mask)

        # Compute mean and variance (with smoothing factor) for each feature
        self.means_[idx] = np.mean(X_c, axis=0)
        self.vars_[idx] = np.var(X_c, axis=0) + var_smoothing_factor
```

Gambar 1.2.2 Cuplikan Code Metode fit() Dalam Algoritma Gaussian Naive Bayes

3. Metode predict

Metode ini digunakan untuk memprediksi data baru menggunakan model yang telah dilatih. Prediksi dilakukan dengan melakukan validasi model dan data terlebih dahulu untuk memastikan **classes_** tidak kosong. Kemudian, melakukan pencocokan jumlah fitur data baru dengan data pelatihan. Terakhir, melakukan prediksi per sampel dengan menghitung probabilitas untuk setiap sampel dalam X.

```python
def predict(self, X: np.ndarray) -> np.ndarray:
    # Check if the model has been trained
    if self.classes_ is None:
        raise ValueError("Model has not been trained yet. Please call the fit method first.")

    # Check if the number of features in X is the same as in the training data
    if X.shape[1] != self.features_:
        raise ValueError("Number of features in X does not match the number of features in the training data.")

    y_pred = [self._predict(x) for x in X]
    return np.array(y_pred)
```

Gambar 1.2.3 Cuplikan Code Metode predict() Dalam Algoritma Gaussian Naive Bayes

4. Metode _predict

Metode ini digunakan untuk memprediksi kelas dari satu sampel data. Langkah-langkah dimulai dengan menghitung log prior dan menambahkan probabilitas prior setiap kelas ke log-likelihood. Setelah itu, akan dihitung log-likelihood menggunakan **_log_pdf** untuk mendapatkan probabilitas likelihood setiap fitur dalam logaritma. Kelas dengan log-likelihood tertinggi akan dipilih sebagai prediksi.

```python
def _predict(self, x: np.ndarray) -> int:
    """Predict the class label for a single sample"""
    # Initialize the list of posterior likelihood
    log_likelihoods = np.zeros(len(self.classes_))

    # Compute the posterior probability for each class
    for idx in range(len(self.classes_)):
        # Add log prior probability
        log_likelihoods[idx] = np.log(self.class_priors_[idx] + 1e-9)  # epsilon

        # Add log likelihood
        log_likelihoods[idx] += np.sum(self._log_pdf(idx, x))

    # Return the class with the highest posterior probability
    return self.classes_[np.argmax(log_likelihoods)]
```

Gambar 1.2.4 Cuplikan Code Metode _predict() Dalam Algoritma Gaussian Naive Bayes

5. Metode _log_pdf

Metode ini digunakan untuk menghitung fungsi probabilitas densitas logaritmik sebagai distribusi Gaussian dengan menggunakan rumus sebagai berikut.

$$log\_pdf(x) = -0.5 \cdot (log(variance) + log(2\pi) + \frac{(x-mean)^2}{variance})$$

Setelah itu, akan dilakukan penambahan *smoothing* untuk mencegah pembagian dengan nol.

```python
def _log_pdf(self, idx: int, x: np.ndarray) -> np.ndarray:
    epsilon = 1e-9

    var = self.vars_[idx] + epsilon
    mean = self.means_[idx]

    return -0.5 * (np.log(var) + np.log(2 * np.pi) + (x - mean) ** 2 / var)
```

Gambar 1.2.5 Cuplikan Code Metode _log_pdf() Dalam Algoritma Gaussian Naive Bayes

BAB II

Cleaning and Preprocessing Data

2.1. Data Cleaning

a. Handling Missing Data

Data yang hilang dapat berdampak buruk pada performa dan akurasi model pembelajaran mesin. Oleh karena itu, diperlukan untuk penanganan *missing data* sebelum membuat model. Pengisian *missing data* dilakukan dengan **penyesuaian terhadap fitur-fitur yang saling berkaitan terlebih dahulu** dan apabila masih ada data yang kosong akan di-*handle* dengan **SimpleImputer**. Pada fungsi SimpleImputer terdapat *strategy* seperti *mean, median,* dan *most_frequent*. Untuk fitur numerik yang tidak memiliki keterkaitan dengan fitur lain, kami menggunakan **fungsi *median* untuk mengisi data numerik** seperti NoOfURLRedirect, NoOfImage, NoOfCSS, NoOfExternalRef, dan NoOfJS. Selain itu, kami juga menggunakan ***safe_mode*** untuk menangani kolom yang tidak memiliki nilai selain NaN dengan mengembalikan nilai 0. Strategi penanganan yang lain juga dilakukan untuk mengisi *missing data* berdasarkan korelasinya terhadap fitur lain yaitu ***predict_based_on_correlation***.

Untuk fitur numerik yang memiliki kaitan dengan fitur lain seperti URLLength, IsDomainIP, DomainLength, TLDLength, IsHTTPS, HasTitle, DomainTitleMatchScore, dan URLTitleMatchScore akan dilakukan penyesuain terhadap URL, Domain dan Title. Untuk fitur kategorik, kami mengisi *missing data* sesuai dengan kaitannya dengan fitur lain. Contohnya pengisian URL menggunakan IsHTTPS dan Domain (jika IsHTTPS bernilai 1 maka URL akan diisi dengan https:// yang diikuti dengan domainnya, namun jika IsHTTPS bernilai 0 maka URL akan diisi dengan http:// yang diikuti dengan domainnya). Hal ini juga berlaku untuk pengisian fitur Domain, TLD, Title, serta fitur-fitur lain terkait URL dan Domain.

b. Dealing with Outliers

Pencilan (atau *outlier*) adalah persebaran data yang sangat berbeda dari sebagian besar data. *Outlier* dapat berupa nilai yang luar biasa tinggi atau rendah yang tidak mengikuti pola data lainnya. Pencilan dapat mempengaruhi kinerja model secara signifikan, maka penting untuk menangani *outlier* dengan benar agar tidak terjadi kesalahan dalam analisis data. Di sini, kami menggunakan metode transformasi data dengan cara mengganti nilai-nilai yang berada di luar batas deteksi *outlier* dengan nilai batas (***clipping***).

1. Jika *method='**zscore**'* maka setiap nilai dalam data akan dipotong sesuai dengan batas bawah dan atasnya.
   Batas bawah: *mean - (threshold x std)*
   Batas atas: *mean + (threshold x std)*
2. Jika *method='**iqr**'* maka nilai akan dipotong berdasarkan rentang atas dan bawah sesuai dengan ketentuan.
   Batas bawah: q25 - (1.5 x IQR)
   Batas atas: q75 + (1.5 x IQR)

Dealing with outlier hanya dilakukan bagi data continuous. Perlakuan ini tidak diberikan pada data boolean (binary) karena kekhawatiran akan misinterpretasi data.

c. Removing Duplicates

Tujuan dari penanganan data duplikat adalah untuk menjaga kualitas, efisiensi, dan keakuratan data. Data duplikat dapat muncul lebih dari satu kali dalam suatu dataset apabila kita menggunakan SimpleImputer dalam menangani *missing data* setelah data *unique* yang tidak digunakan di-*drop*. Penanganan data duplikat dilakukan dengan fungsi ***remove_duplicates***. Penggunaan fungsi ini akan menghapus semua data duplikat yang ada. Namun, karena pengimputan juga dikombinasikan dengan *domain knowledge* atau formula khusus bagi setiap fitur, **tidak dijumpai data duplikat** sehingga langkah ini dapat ditinggalkan.

d. Feature Engineering

Feature engineering dibutuhkan untuk meningkatkan kualitas model, terutama pada data yang memiliki dimensionalitas tinggi seperti data yang digunakan pada tugas ini. Feature engineering yang kami gunakan adalah dimensionality reduction dengan langsung men-***drop*** **kolom** yang tampaknya tidak begitu berelasi dengan fitur atau kolom yang sudah terwakili sepenuhnya oleh kolom-kolom lain, misalnya URL, Domain, dan Title.

Selain itu, kami juga menggunakan dimensionality reduction **Principal Component Analysis (PCA)**. PCA adalah metode yang digunakan untuk menyederhanakan dataset yang kompleks dan mengurangi korelasi antar fitur, hal ini terutama bagus untuk algoritma Gaussian Naive Bayes yang mengasumsikan fitur satu tidak berkaitan erat dengan fitur lainnya. PCA juga dapat diatur agar menjaga variansi dataset yang asli, kami menggunakan n_components = 0.95.

2.2. Data Preprocessing

a. Feature Scaling

Feature Scaling adalah teknik *preprocessing* yang digunakan dalam pembelajaran mesin untuk menstandarisasi rentang variabel independen. Tujuan utama *feature scaling* adalah untuk memastikan bahwa semua fitur berkontribusi secara setara pada data *train* dan algoritma pembelajaran mesin dapat bekerja secara efektif dengan data. *Method* yang digunakan adalah *normalization* dengan membuat kelas *Normalization* dan *standardization* dengan membuat kelas *Standardization*. Kelas ***Normalization*** digunakan untuk mengubah data sehingga berada dalam rentang tertentu yaitu 0 hingga 1 sedangkan kelas ***Standardization*** digunakan untuk mengubah data sehingga memiliki nilai *mean* = 0 dan standar deviasi = 1. Feature scaling terutama dibutuhkan untuk algoritma yang sensitif terhadap jarak dan/atau skala seperti KNN, tetapi Gaussian Naive Bayes juga dapat sedikit dioptimasi dengan *scaling*. Sams seperti *dealing with outlier*, *feature scaling* hanya dilakukan untuk data kontinuous.

b. Feature Encoding (Categorical Encoding)

Feature Encoding dilakukan untuk mengubah data kategorik (data non-numerik) ke dalam format numerik agar dapat di-*input* oleh algoritma pembelajaran mesin. Pada tahap ini, kami menggunakan *method* ***One-Hot Encoding*** untuk menangani nilai di luar daftar teratas dengan label khusus seperti "others". Kelas ***Top5TLDOneHotEncoder*** meng*inherit* BaseEstimator dan TransformerMixin dari *scikit-learn* untuk 5 nilai dengan frekuensi tertinggi dan nilai lainnya dikategorikan sebagai 'others'.

c. Handling Imbalanced Classes

Terdapat ketidakseimbangan kelas pada dataset yang disediakan, yaitu label 1 yang mendominasi dengan 25970 data dan label 0 yang jauh lebih sedikit dengan 2111 data saja. *Imbalanced classes* dapat ditangani dengan menggunakan library **imblearn** over_sampling dan under_sampling. Over_sampling dengan SMOTE dan under_sampling dengan RandomUnderSampler. Namun, model tidak menunjukkan perubahan kinerja yang signifikan, bahkan cenderung menurun, setelah mengalami over_sampling maupun under_sampling sehingga akhirnya kedua metode tersebut tidak digunakan pada submisi.

BAB III
Perbandingan Hasil Prediksi

3.1. Perbandingan Hasil Prediksi Algoritma KNN Scratch dengan Pustaka

a. Algoritma KNN dengan Jarak Euclidean

Tabel 3.1.1. Perbandingan Jarak Euclidean (Scratch dengan Pustaka)

Scratch	Pustaka
classification_report for euclidean precision recall f1-score support 0 0.46 0.34 0.39 50 1 0.93 0.96 0.94 450 accuracy 0.89 500 macro avg 0.69 0.65 0.67 500 weighted avg 0.88 0.89 0.89 500	Testing KNN with metric: euclidean Validation Metrics for euclidean: precision recall f1-score support 0 0.98 0.72 0.83 2111 1 0.98 1.00 0.99 25970 accuracy 0.98 28081 macro avg 0.98 0.86 0.91 28081 weighted avg 0.98 0.98 0.98 28081 Validation F1 Score (Macro) for euclidean: 0.9104

Perbandingan hasil antara algoritma KNN dengan jarak Euclidean yang diimplementasikan sendiri dan menggunakan pustaka sklearn menunjukkan perbedaan signifikan. Berdasarkan data pustaka sklearn, akurasi mencapai 98% dengan F1 Score (Macro Average) sebesar 0.91, sementara implementasi manual menghasilkan akurasi 88% dengan F1 Score (Macro Average) 0.67. Perbedaan ini mengindikasikan bahwa implementasi manual memiliki kekurangan dalam logika perhitungan jarak Euclidean atau pemilihan tetangga terdekat (k=5). Meskipun demikian, hasil implementasi manual menunjukkan performa yang cukup baik, terutama pada kelas dengan dukungan data yang lebih besar. Insight yang dapat diambil adalah pentingnya validasi menyeluruh untuk memastikan konsistensi antara algoritma manual dan pustaka dalam menghadirkan hasil yang akurat.

b. Algoritma KNN dengan Jarak Manhattan

Tabel 3.1.2. Perbandingan Jarak Manhattan (Scratch dengan Pustaka)

Scratch	Pustaka

	precision	recall	f1-score	support
0	0.45	0.42	0.43	50
1	0.94	0.94	0.94	450
accuracy			0.89	500
macro avg	0.69	0.68	0.69	500
weighted avg	0.89	0.89	0.89	500

```
Testing KNN with metric: manhattan
Validation Metrics for manhattan:
              precision    recall  f1-score   support

           0       0.98      0.83      0.90      2111
           1       0.99      1.00      0.99     25970

    accuracy                           0.99     28081
   macro avg       0.98      0.91      0.94     28081
weighted avg       0.99      0.99      0.98     28081

Validation F1 Score (Macro) for manhattan: 0.9436
```

Perbandingan hasil antara algoritma KNN dengan jarak Manhattan yang diimplementasikan sendiri dan menggunakan pustaka sklearn menunjukkan perbedaan signifikan pada performa. Berdasarkan data pada versi implementasi pustaka sklearn (Tabel 3.1.2), akurasi mencapai 99% dengan F1 Score Macro Average sebesar 0,94, sedangkan implementasi mandiri menghasilkan akurasi 89% dengan F1 Score Macro Average 0,69. Hasil ini menunjukkan bahwa meskipun logika perhitungan jarak Manhattan dan pemilihan tetangga terdekat (k=5) pada implementasi mandiri sudah berjalan sesuai, ada kelemahan pada skala performa, kemungkinan akibat optimasi yang belum maksimal dibandingkan pustaka. Insight yang didapat adalah pustaka sklearn memberikan performa lebih stabil dan unggul untuk skala data besar, sementara implementasi mandiri tetap valid untuk skala kecil tetapi perlu perbaikan lebih lanjut.

c. Algoritma KNN dengan Jarak Minkowski

Tabel 3.1.3. Perbandingan Jarak Minkowski (Scratch dengan Pustaka)

Scratch	Pustaka

```
classification_report for minkowski
              precision    recall  f1-score   support

           0       0.46      0.34      0.39        50
           1       0.93      0.96      0.94       450

    accuracy                           0.89       500
   macro avg       0.69      0.65      0.67       500
weighted avg       0.88      0.89      0.89       500
```

```
Testing KNN with metric: minkowski
Validation Metrics for minkowski:
              precision    recall  f1-score   support

           0       0.98      0.72      0.83      2111
           1       0.98      1.00      0.99     25970

    accuracy                           0.98     28081
   macro avg       0.98      0.86      0.91     28081
weighted avg       0.98      0.98      0.98     28081

Validation F1 Score (Macro) for minkowski: 0.9104
```

Perbandingan hasil antara algoritma KNN dengan jarak Minkowski yang diimplementasikan sendiri dan menggunakan pustaka sklearn menunjukkan perbedaan signifikan. Berdasarkan data pada implementasi pustaka sklearn (Tabel 3.1.3), akurasi mencapai 98% dengan F1 Score Macro Average sebesar 0,91, sedangkan implementasi mandiri menghasilkan akurasi 89% dengan F1 Score Macro Average sebesar 0,67. Hasil ini menunjukkan bahwa logika perhitungan jarak Minkowski dan pemilihan tetangga terdekat (k=5) pada implementasi mandiri sudah berjalan sesuai namun masih ada kelemahan pada skala performa dibandingkan pustaka karena optimasi yang kurang maksimal. Insight yang didapat adalah implementasi mandiri dan pustaka harus dilakukan validasi menyeluruh untuk mendapatkan hasil yang lebih bagus.

3.2. Perbandingan Hasil Prediksi Algoritma Naive Bayes Scratch dengan Pustaka

Tabel 3.2.1. Perbandingan Naive Bayes (Scratch dengan Pustaka)

Scratch

	precision	recall	f1-score	support
0	0.73	0.89	0.80	2111
1	0.99	0.97	0.98	25970
accuracy			0.97	28081
macro avg	0.86	0.93	0.89	28081
weighted avg	0.97	0.97	0.97	28081

Pustaka

	precision	recall	f1-score	support
0	0.84	0.77	0.80	2111
1	0.98	0.99	0.98	25970
accuracy			0.97	28081
macro avg	0.91	0.88	0.89	28081
weighted avg	0.97	0.97	0.97	28081

Perbandingan hasil antara algoritma Naive Bayes yang diimplementasikan secara mandiri dan yang menggunakan pustaka menunjukkan bahwa akurasi dan F1 Score (weighted average) yang dihasilkan hampir sama, yaitu 97% untuk keduanya. Namun, terdapat sedikit perbedaan pada precision dan recall (macro average), di mana implementasi mandiri menghasilkan precision 86% dan recall 89%, sedangkan pustaka menghasilkan precision 84% dan recall 77%. Hal ini menunjukkan bahwa meskipun logika dasar Naive Bayes dalam implementasi mandiri sudah sesuai, performa pustaka sedikit lebih unggul dalam konsistensi pada data yang lebih kompleks. Insight yang

didapat adalah implementasi mandiri dapat bekerja dengan baik, tetapi penggunaan pustaka menawarkan stabilitas dan efisiensi yang lebih tinggi.

BAB IV
Penutup

4.1. Kesimpulan

Implementasi algoritma KNN dan Gaussian Naive-Bayes *from scratch* menunjukkan performa yang cukup baik namun masih memiliki beberapa kelemahan dibandingkan dengan pustaka seperti sklearn, terutama pada aspek akurasi dan F1 Score yang lebih rendah pada implementasi manual untuk KNN. Hal ini mengindikasikan perlunya optimasi lebih lanjut dalam logika perhitungan jarak dan pemilihan tetangga terdekat pada KNN. Sebaliknya, implementasi Gaussian Naive-Bayes secara mandiri menunjukkan hasil yang kompetitif dengan pustaka, meskipun pustaka tetap unggul dalam efisiensi dan stabilitas pada data yang kompleks. Secara keseluruhan, penggunaan pustaka lebih disarankan untuk skala data besar karena menawarkan keandalan, konsistensi, dan optimasi yang lebih baik dibandingkan dengan implementasi manual.

4.2. Pembagian Tugas

Tabel 4.2.1. Pembagian Tugas

Anggota	Pekerjaan
Lydia Gracia (18222035)	Pipeline Development, Data Processing, Model Implementation, Documentation
Willhelmina Rachel Silalahi (18222049)	Machine Learning Pipeline, Data Analysis, Model Development, Technical Documentation
Favian Izza Diasputra (18222070)	Data Engineering, Model Architecture, Pipeline Integration, Documentation
Ervina Limka (18222100)	Feature Engineering, Model Optimization, Pipeline Development, Documentation Support

Referensi

Lembaga Penelitian dan Pengabdian Masyarakat Universitas Medan Area. (2023, Februari 16). *Algoritma K-Nearest Neighbors (KNN): Pengertian dan penerapan*. Diakses dari https://lp2m.uma.ac.id/2023/02/16/algoritma-k-nearest-neighbors-knn-pengertian-dan-penerapan/

Murni, A. P., & Sibarani, S. L. (2021). *Implementasi K-Nearest Neighbors (KNN) untuk Klasifikasi Data Mahasiswa*. Jurnal Sistem dan Aplikasi Teknologi Informasi, 4(1), 49-57. Diakses dari https://e-journals.unmul.ac.id/index.php/jsakti/article/view/9241

Javed, Aisha. "Unfolding Naïve Bayes from Scratch ! | by Aisha Javed." *Towards Data Science*, Medium, 25 August 2018, https://towardsdatascience.com/unfolding-na%C3%AFve-bayes-from-scratch-2e86dcae4b01#08ef.

Tokuç, A. Aylin. "How to Improve Naive Bayes Classification Performance?" *naive-bayes-classification-performance*, Baeldung, 18 March 2024, https://www.baeldung.com/cs/naive-bayes-classification-performance.