IF3070 Dasar Artificial Intelligence

Tugas Besar 2

Dipersiakan oleh

Kelompok 15

Akmal Gali Aji Sugmo Seno	18222046
Muhammad Kevinza Faiz	18222072
Raizan Iqbal Resi	18222068
Muhammad Raihan Arrifianto	18222092

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

Bab I. Implementasi Algoritma Pembelajaran Mesin

1.1. Implementasi Algoritma KNN

K-Nearest Neighbor atau KNN adalah algoritma *machine learning* yang digunakan untuk melakukan klasifikasi ataupun regresi nilai pada suatu *dataset.* Prinsip kerja algoritma KNN didasarkan pada aspek similaritas data, di mana sebuah data baru diklasifikasikan atau diprediksi nilainya berdasarkan kedekatan dengan data lain di sekitarnya. Berikut adalah gambaran besar mengenai kerja KNN:

- Parameter (K) atau jumlah tetangga terdekat ditentukan. Besarnya nilai (K) berdampak pada algoritma, dimana nilai (K) yang kecil akan membuat model lebih spesifik dan sensitif terhadap *noise*, yaitu data yang tidak relevan yang dapat menyebabkan kesalahan dalam analisis ataupun prediksi, sedangkan nilai (K) yang besar bisa menyebabkan pengabaian terhadap pola.
- Algoritma menghitung jarak antara sebuah data dengan data lain dalam suatu *dataset* menggunakan metrik yang ditentukan, antara lain jarak Euclidean, jarak Manhattan, dll.
- Algoritma akan mengambil (K) tetangga terdekat yang memiliki jarak terpendek ke data yang di analisis.
- Untuk tujuan klasifikasi, algoritma akan melakukan "*voting"* dari label-label tetangga terdekat untuk kelas yang sesuai.
- Algoritma terakhir melakukan prediksi hasil dengan mengklasifikasikan data baru berdasarkan hasil langkah sebelumnya.

Perlu diketahui bahwa algoritma KNN memiliki karakteristik seperti *lazy learning*, di mana KNN tidak melakukan proses *training* secara eksplisit, melainkan melakukan penyimpanan seluruh data dan digunakan kembali saat melakukan prediksi. Hal ini dapat menyebabkan waktu prediksi menjadi lambat dan kebutuhan memori yang lebih besar. Untuk menangani waktu prediksi lambat, digunakan linear *data structure* sehingga dapat menekan kompleksitas waktu hingga $O(nlogn)$.

Dalam implementasinya, algoritma KNN dibuat dalam sebuah Kelas Object yang menerima input dataset, jumlah *neighbour*, dan pemilihan metrik. Berikut adalah *method* yang ada pada algoritma KNN yang dibuat:

- **fit (X, y):** adalah *method* yang digunakan untuk penyimpanan data yang menerima dua input parameter, yakni *training* (X_train) dan label training (y_train). Selain itu, *method* ini mengimplementasikan KD-Tree dalam prosesnya.
- **predict (X):** adalah *method* yang digunakan untuk memprediksi hasil dari data yang menerima input parameter, yakni X_test. *Method* ini akan *return* sebuah array yang berisi prediksi yang telah dihasilkan.

- **euclidean_distance (x1, x2):** adalah *private method* yang menerima input dua parameter, yakni x1 dan x2, yang digunakan untuk menghitung jarak *euclidean* antara 2 data. Berikut adalah rumus yang digunakan:

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

- **manhattan_distance (x1, x2):** adalah *private method* yang menerima input dua parameter, yakni x1 dan x2, yang digunakan untuk menghitung jarak *manhattan* antara 2 data. Berikut adalah rumus yang digunakan:

$$d(x, y) = \sum_{i=1}^{n}|x_i - y_i|$$

- **minkowski_distance (x1, x2):** adalah *private method* yang menerima input dua parameter, yakni x1 dan x2, yang digunakan untuk menghitung jarak *minkowski* antara 2 data. Berikut adalah rumus yang digunakan:

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

1.2. Implementasi Algoritma Gaussian Naive-Bayes

Gaussian Naive Bayes adalah sebuah metode klasifikasi yang digunakan dalam *machine learning* untuk memprediksi sebuah nilai. Algoritma ini memanfaatkan beberapa pendekatan utama dalam melakukan pemodelan, yaitu metode *probability* dan distribusi Gaussian atau distribusi normal. Distribusi Gaussian mengasumsikan bahwa seluruh fitur yang digunakan dalam pemodelan bersifat independen dalam memprediksi suatu nilai. Gaussian Naive Bayes mengasumsikan bahwa *likelihood* P(Xi | Y) mengikuti distribusi

Gaussian untuk setiap xi dalam Yk. Berikut adalah rumus distribusi normal yang mendefinisikan *probability* sebuah observasi berada pada salah satu kelas.

$$P(x_i|y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Dengan μ adalah rata-rata dan σ adalah standar deviasi.

Untuk mengklasifikasikan setiap data baru x, algoritma mencari nilai maksimum dari probabilitas *posterior* untuk setiap kelas dan menetapkan data tersebut ke kelas dengan probabilitas tertinggi.

Selanjutnya, untuk memahami lebih jauh terkait cara kerja algoritma Gaussian Naive Bayes, perlu dipahami beberapa konsep matematika di baliknya.

- ***Probability Density Function*** **:**
  Fungsi ini digunakan untuk menghitung *likelihood* atau kemungkinan suatu fitur berdasarkan kelas tertentu.

- ***Conditional Probability*** **:**
  Perhitungan probabilitas dari setiap kelas pada label dataset berdasarkan fitur masukkan yang diberikan.

- ***Bayes' Theorem in Gaussian Naive Bayes*** **:**
  Teorema yang digunakan dalam menghitung probabilitas *posterior* suatu kelas berdasarkan fitur *input*.

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(B|A) \cdot P(A)}{P(B)} \end{aligned}$$

  - P (A|B) : probabilitas *posterior* yang menyatakan probabilitas terjadinya A apabila B telah terjadi
  - P (A) : probabilitas *prior*
  - P (B) : probabilitas terjadinya peristiwa B
  - P (B|A) : probabilitas terjadinya B dengan asumsi bahwa A sudah terjadi

Dalam pengimplementasian algoritma Gaussian Naive Bayes, model disimpan dalam sebuah kelas *object* yang di dalamnya terdapat 3 buah *method*. Penjelasan lebih lanjut untuk setiap *method* adalah berikut :

- **fit (X, y)**
  Fungsi ini digunakan untuk melatih model dengan menghitung dan menyimpan parameter statistik untuk setiap kelas. Parameter statistik tersebut antara lain adalah berikut :
  1. Mean : Rata-rata fitur dalam setiap kelas
  2. Varians : Variansi fitur untuk setiap kelas
  3. *Prior* : Probabilitas *prior* dari setiap kelas, dihitung sebagai proporsi data dari suatu kelas

  Setiap parameter tersebut akan disimpan dalam sebuah *dictionary* self.parameters.

- **probability_density (x, mean, var)**
  Fungsi ini digunakan untuk menghitung *Probability Density Gaussian* (PDF) untuk nilai x berdasarkan mean dan varians suatu fitur. Fungsi ini menggunakan perhitungan dari rumus distribusi normal.

- **predict (x)**
  Fungsi ini digunakan untuk memprediksi kelas untuk setiap baris dalam dataset x. Proses yang dilakukan dalam *method* ini adalah sebagai berikut :
  1. Menghitung *prior* dan *likelihood* untuk setiap kelas.
  2. Menjumlahkan *prior* dan likelihood menjadi *posterior log-probability*.
  3. Memilih kelas dengan nilai *posterior* terbesar.
  4. Apabila nilai *posterior* adalah sama untuk kedua kelas, maka dilakukan pemilihan kelas secara acak.

Selain itu, dilakukan penambahan *logarithmic scaling* dengan menggunakan fungsi log dalam menghitung *prior* dan *likelihood* untuk mencegah permasalahan numerik akibat perkalian dengan probabilitas sangat kecil, yaitu menuju -∞.

Bab II. Tahap Cleaning dan Preprocessing

2.1. Tahap Cleaning

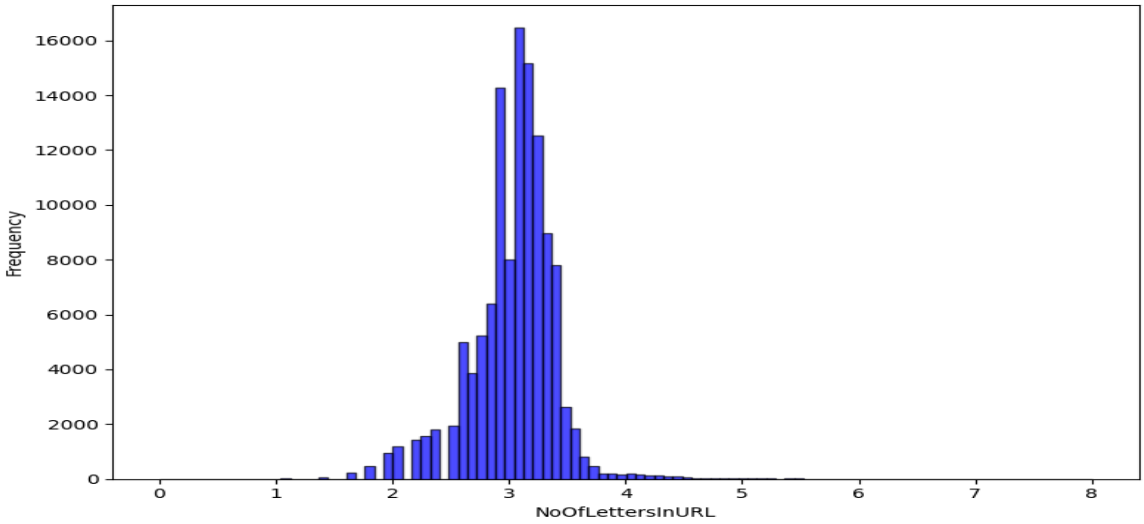
Proses *cleaning* adalah tahapan membersihkan data mentah untuk memastikan kualitas dan konsistensi *dataset* yang akan digunakan dalam analisis ataupun model *machine learning*. Proses ini melibatkan identifikasi data, penanganan data yang tidak lengkap, duplikasi data, inkonsistensi, dan eliminasi *outlier*. Hal ini bertujuan untuk meningkatkan akurasi dan validitas data sehingga dapat mencerminkan sebuah pola atau hubungan yang sebenarnya.

2.1.1. Missing Values

Untuk menangani missing value, pertama-tama dataset dibagi terlebih dahulu berdasarkan jenis atributnya, yaitu numerik dan kategorikal. Untuk feature yang memiliki kaitan dengan URL dilakukan imputasi menggunakan *value* yang didapat dari kolom URL. Sedangkan untuk atribut yang dianggap tidak dapat dilakukan imputasi dari URL, digunakan metode yang berbeda untuk tiap jenisnya. Atribut numerik yang kontinu diisi menggunakan nilai rata-rata, sedangkan atribut yang kategorikal diisi dengan menggunakan metode KNN imputer.

2.1.2. Outlier

Kami melakukan analisis terhadap data yang sudah diimputasi untuk menangani missing values dan menemukan bahwa sebagian besar data memiliki outlier yang menyebabkan *skewness*. Oleh karena itu, diperlukan metode untuk menangani outlier yang menyebabkan *skewness* tersebut



2.1.2.1 Outlier Clipping

Pada bagian menangani outlier, kami menggunakan metode clipping pada value yang berada di luar 95% nilai pada atribut tersebut. Sehingga, nilai yang berada di luar rentang tersebut akan di-*clipping* ke *upper bound* atau *lower bound*

2.1.2.2 Log Transformation

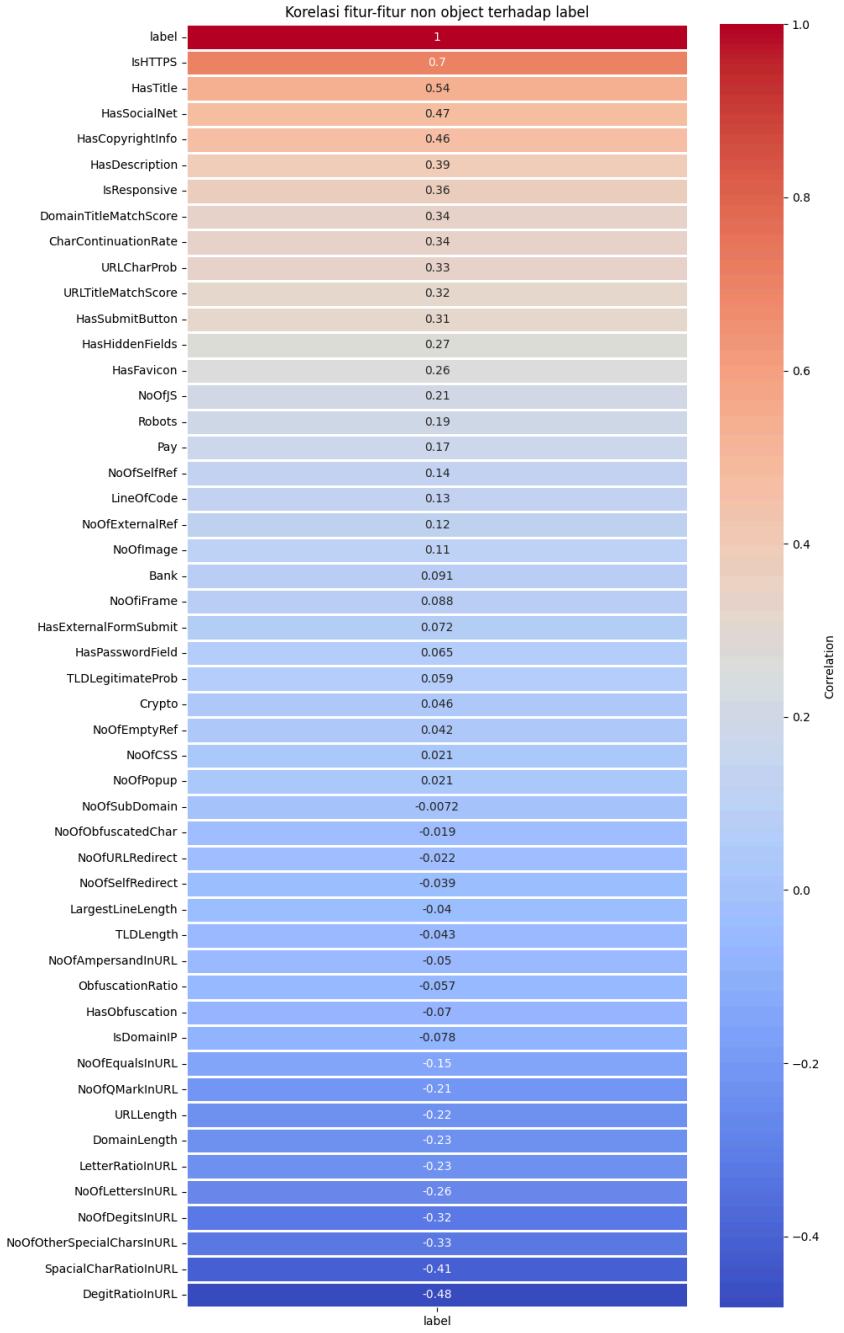
Kami menggunakan metode Log Transformation terhadap atribut yang memiliki skewness terhadap datanya setelah imputasi. Metode ini bekerja dengan menghitung nilai ln pada tiap nilai di atribut tersebut. Hal ini selain mengurangi skewness data dan menangani outlier juga mempermudah interpretasi data tersebut

2.1.3. Removing Duplicates

Setelah dilakukan analisa, kami tidak menemukan adanya data yang terdeteksi sebagai duplikat. Namun, kami menambahkan kode untuk drop data yang terdeteksi sebagai duplikat dan melakukan drop terhadap row tersebut

2.1.4. Feature Engineering.

Setelah melakukan *exploratory data analysis* (EDA), dilakukan penghapusan fitur yang memiliki korelasi rendah dengan label. Fitur yang dihapus adalah fitur yang memiliki nilai korelasi >-0.3 dan <0.3, pemilihan batas nilai korelasi ditentukan dengan mementingkan juga jumlah fitur yang ingin disimpan, sehingga menyisakan 10-15 fitur.



Pengurangan jumlah fitur menghilangkan fitur-fitur yang tidak relevan atau redundan, sehingga model dapat lebih fokus pada pola yang benar-benar signifikan dalam data. Hal ini tidak hanya mengurangi risiko overfitting, tetapi juga meningkatkan kecepatan pemrosesan dan efisiensi model, karena data yang diproses memiliki dimensi yang lebih kecil. Selain itu, model yang dibangun dengan lebih sedikit fitur menjadi lebih sederhana dan mudah dipahami, memungkinkan interpretasi yang lebih baik terhadap hasil prediksi. Dengan menghilangkan fitur yang

kurang bermakna, model juga menjadi lebih andal dalam menghadapi data baru, karena hanya menggunakan informasi yang paling relevan untuk membuat prediksi. Pendekatan ini penting untuk meningkatkan performa dan generalisasi model dalam berbagai aplikasi machine learning. Selain itu, dilakukan juga uji untuk pengecekan *multicollinearity* dimana fitur yang memiliki korelasi dengan fitur yang disimpan juga ikut dihilangkan.

2.2. Tahap Preprocessing

2.2.1. Feature Scaling

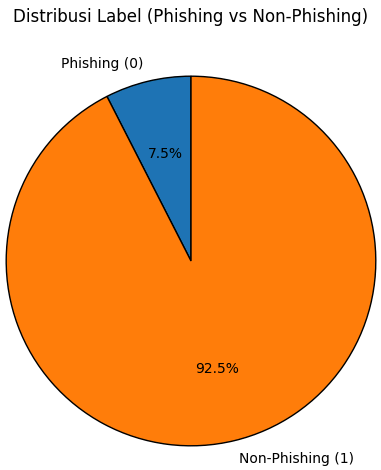
Feature scaling di sini kami lakukan terhadap atribut numerik kontinu yang memiliki *skewness* yang cukup signifikan. Pertama, kami mencari atribut yang memiliki *skewness* dengan visualisasi distribusi nilai tiap atribut dalam histogram. Kami mendapati bahwa kecuali atribut NoOfLettersInURL dan URLLength semua atribut numerik yang kontinu memiliki right-skewness. Dengan insight ini, kami menggunakan metode Robust Scaling terhadap seluruh atribut numerik yang kontinu

2.2.2. Feature Encoding

Kami melakukan encoding terhadap atribut TLD agar lebih mudah melakukan analisis korelasi nya dengan atribut numerik lainnya. Kami menggunakan metode One-Hot encoding karena metode ini mempermudah analisis korelasi dan tetap memperlakukan tiap kemungkinan value di TLD sebagai hal yang berbeda

2.2.3. Handling Imbalanced Dataset

Dalam penanganan *imbalanced dataset*, terdapat beberapa opsi seperti melakukan *oversampling* dan *undersampling*. *Oversampling* menambahkan sampel dari kelas minoritas sehingga jumlahnya menjadi seimbang dengan kelas mayoritas, salah satu contohnya menggunakan metode seperti SMOTE (Synthetic Minority Oversampling Technique). Sebaliknya, *undersampling* mengurangi jumlah sampel dari kelas mayoritas agar sesuai dengan kelas minoritas, dengan tujuan mengurangi bias terhadap kelas mayoritas.



Dari distribusi kelas pada label, dapat dilihat bahwa kelas phising (label = 1) hanya merepresentasikan 7,5% dari keseluruhan dataset, hal ini dapat menyebabkan bias pada *modelling* dan pengurangan akurasi untuk prediksi data *phising*.

Penanganan dilakukan dengan melakukan *oversampling* terhadap dataset melalui SMOTE-ENN. ENN (Edited Nearest Neighbour) di sini berperan dalam mengurangi *noise* pada dataset, yakni dengan menghapus data mayoritas yang salah klasifikasi. Metode ini dapat mengurangi bias sekaligus meningkatkan presisi prediksi.

Bab III. Pembahasan

3.1. Perbandingan Hasil Algoritma

3.1.1. KNN

Berikut adalah perbedaan kinerja berdasarkan metrik jarak antara implementasi mesin KNN kami dan implementasi Scikit-learn:

1. Jarak Minkowski
    - Gap Akurasi: -0.085%
    - Gap Presisi (Kelas 0): -0.408%
    - Gap Recall (Kelas 0): 0.000% (Cocok sempurna)
    - Gap F1-Score (Kelas 0): -0.204%
    - Gap Presisi (Kelas 1): 0.000% (Cocok sempurna)
    - Gap Recall (Kelas 1): -0.396%
    - Gap F1-Score (Kelas 1): -0.201%
    - Gap Rata-rata F1-Score: -0.203%
2. Jarak Manhattan
    - Gap Akurasi: -0.027%
    - Gap Presisi (Kelas 0): -0.013%
    - Gap Recall (Kelas 0): 0.000% (Cocok sempurna)
    - Gap F1-Score (Kelas 0): -0.007%
    - Gap Presisi (Kelas 1): -0.024%
    - Gap Recall (Kelas 1): 0.000% (Cocok sempurna)
    - Gap F1-Score (Kelas 1): -0.012%
    - Gap rata-rata F1-Score: -0.009%
3. Jarak Euclidean
    - Perbedaan Akurasi: -0.023%
    - Perbedaan Presisi (Kelas 0): -0.122%
    - Perbedaan Recall (Kelas 0): 0.000% (Cocok sempurna)
    - Perbedaan F1-Score (Kelas 0): -0.061%
    - Perbedaan Presisi (Kelas 1): 0.000% (Cocok sempurna)
    - Perbedaan Recall (Kelas 1): -0.101%
    - Perbedaan F1-Score (Kelas 1): -0.051%
    - Perbedaan Rata-rata Makro F1-Score: -0.057%

Gap pada mesin yang dibuat dari *scratch* dengan mesin Scikit-learn's memiliki beberapa poin yang dapat diperhatikan pada tiap metrik:

1. Jarak Manhattan

Jarak Manhattan, yang menghitung jarak absolut antara koordinat (L1-norm), menunjukkan kemiripan terbesar dengan hasil Scikit-learn. Hal ini karena perhitungan jarak Manhattan lebih sederhana dibandingkan metrik lainnya. Jarak ini tidak melibatkan operasi perpangkatan ataupun akar kuadrat, sehingga:

a. Kurang sensitif terhadap kesalahan pembulatan: Perhitungan hanya berupa penjumlahan nilai absolut, yang cenderung stabil secara numerik.
b. Efek *noise* data lebih kecil: Perubahan kecil dalam koordinat tidak berdampak besar pada perhitungan jarak.

Karena kesederhanaannya, implementasi kami paling mendekati hasil Scikit-learn tanpa banyak perbedaan numerik dibanding metrik lainnya.

2. Jarak Minkowski

Dengan parameter p=3, jarak Minkowski merupakan metrik yang lebih kompleks. Jarak ini menggunakan perpangkatan p dan akar pangkat-p, yang membuat perhitungan lebih:

a. Sensitif terhadap kesalahan pembulatan: Ketika data mengandung nilai dengan skala yang besar atau kecil, perpangkatan dan pengakaran dapat memperbesar kesalahan.
b. Tidak stabil untuk dimensi tinggi: Pada dataset dengan banyak fitur, efek jarak Minkowski cenderung lebih rentan terhadap pengaruh dimensi tertentu yang memiliki nilai ekstrem.

Akibatnya, pemilihan tetangga dalam implementasi kami dapat berbeda dari Scikit-learn, terutama pada kasus *borderline* (contoh: titik yang hampir sama jaraknya ke beberapa tetangga).

3. Jarak Euclidean

Sebagai kasus khusus Minkowski dengan p=2, jarak Euclidean menggunakan akar kuadrat untuk menghitung jarak langsung (L2-norm). Meskipun jarak Euclidean lebih stabil dibandingkan Minkowski dengan p>2, ia tetap melibatkan operasi akar kuadrat, yang:

a. Memengaruhi presisi: Kesalahan kecil dalam perhitungan akar dapat menyebabkan peringkat tetangga berubah, terutama jika beberapa titik memiliki jarak hampir sama.

b. Menurunkan presisi untuk Kelas 0: Pada dataset dengan distribusi yang tidak merata antara kelas, jarak Euclidean dapat mengakibatkan bias dalam memilih tetangga dari kelas mayoritas.

Gap pada hasil mesin pembelajaran KNN yang kami buat dapat disebabkan oleh hal-hal berikut:

1. Presisi Numerik

   Scikit-learn menggunakan pustaka numerik yang dioptimalkan seperti Cython dan NumPy, yang dirancang untuk menangani perhitungan presisi tinggi. Implementasi kami, yang mungkin menggunakan float bawaan Python, lebih rentan terhadap:

   a. Pembulatan nilai kecil: Misalnya, $\sqrt{a^2 + b^2}$ dapat menghasilkan perbedaan nilai signifikan jika hasil $a^2 + b^2$ sangat besar atau sangat kecil.
   b. Efek *chaining* error: Pada dataset dengan banyak fitur, kesalahan kecil dapat terakumulasi ketika menghitung jarak total.
2. Efisiensi KD-Tree

   KD-Tree pada *Scikit-learn* dirancang untuk mempartisi data secara optimal, dengan pembagian yang seimbang untuk meminimalkan waktu pencarian tetangga. Perbedaan efisiensi KD-Tree dapat memengaruhi:

   a. Pencarian tetangga dekat: Pada implementasi kami, pemilihan titik-batas atau jarak-sama diproses kurang optimal.
   b. Kasus tepi: Scikit-learn memiliki algoritma tambahan untuk menangani poin-poin yang jaraknya sama ke beberapa tetangga, sedangkan implementasi kami mengandalkan pengurutan default tanpa aturan tambahan.
3. Implementasi Metrik Jarak

   Minkowski (p=3) memerlukan perhitungan lebih kompleks, terutama pada tahap:

   a. Eksponen: Perhitungan $\sum \left|x_i - y_i\right|^p$ memperbesar efek dari perbedaan kecil pada nilai fitur tertentu.

b. Akar pangkat-p: Operasi ini sensitif terhadap presisi numerik, sehingga hasilnya dapat berbeda, terutama pada dataset yang memiliki banyak dimensi atau nilai ekstrem.

4. Parallelisasi dan Efisiensi

Scikit-learn menggunakan teknik paralel untuk mempercepat:

a. Pencarian KD-Tree: Operasi ini dipecah menjadi beberapa thread untuk menangani dimensi yang berbeda secara bersamaan.
b. Voting hasil akhir: Paralelisasi dalam menghitung jumlah tetangga dari setiap kelas mengurangi waktu eksekusi dan memastikan hasil lebih konsisten.

Implementasi kami, yang bergantung pada pemrosesan sekuensial, membutuhkan lebih banyak waktu untuk memproses titik-titik tertentu. Hal ini dapat menyebabkan perbedaan kecil dalam hasil akhir karena urutan pengolahan data.

3.1.2. Gaussian Naive-Bayes

Berikut adalah kesenjangan kinerja antara implementasi Gaussian Naive Bayes (GNB) kami dan Scikit-learn:

Kelas 0:

- *Gap* Presisi: -0,001% (hampir identik)
- *Gap* Recall: -0,824% (sedikit lebih buruk dalam mengenali true positive untuk Kelas 0)
- *Gap* F1-Score: -0,430%

Kelas 1:

- *Gap* Presisi: -0,666% (lebih banyak false positive untuk Kelas 1 dibandingkan Scikit-learn)
- *Gap* Recall: 0,000% (identik)
- *Gap* F1-Score: -0,345%

Rata-rata Makro

- *Gap* Presisi: -0,320%
- *Gap* Recall: -0,393%
- *Gap* F1-Score: -0,384%

Rata-rata *Weighted*

- *Gap* Presisi: -0,326%
- *Gap* Recall: -0,384%
- *Gap* F1-Score: -0,398%

Akurasi: -0,384%

Observasi

1. Presisi Kelas 1 memiliki kesenjangan terbesar (-0,666%), yang kemungkinan disebabkan oleh perbedaan dalam penanganan *false positive*. Hal ini mengindikasikan bahwa implementasi kami cenderung lebih sering salah memprediksi kelas 1.
2. Recall kelas 1 identik dengan Scikit-learn (0,000%), menunjukkan bahwa implementasi kami memiliki kemampuan yang setara dalam mengenali true positive untuk kelas 1.
3. Recall kelas 0 lebih rendah sebesar -0,824%, yang berdampak pada kinerja keseluruhan, termasuk skor F1 untuk Kelas 0 dan rata-rata makro.

Kemungkinan Penyebab

1. Presisi Numerik

   Scikit-learn kemungkinan memiliki pengelolaan stabilitas numerik yang lebih baik, terutama dalam menangani kasus ekstrem seperti log(0) atau varians yang sangat kecil. Implementasi kami mungkin rentan terhadap kesalahan numerik dalam skenario probabilitas ekstrem.

2. *Smoothing*

   Scikit-learn menerapkan *smoothing Laplace*, yang membantu mengurangi jumlah *false positive*, terutama untuk kelas 1. *Smoothing* ini memberikan stabilitas tambahan ketika menghadapi data dengan distribusi probabilitas yang tidak merata.

3. Penanganan Kasus Ekstrem

   Peringatan *runtime* dalam implementasi kami, seperti "*divide by zero*," menunjukkan adanya potensi masalah dalam menangani kasus ekstrem, seperti varians nol atau fitur yang hilang. Hal ini dapat memengaruhi prediksi, terutama dalam kondisi data yang tidak ideal.

Bab IV. Penutup

4.1. Github Repository: https://github.com/Qibaal/TubesDAI-2

Pembagian Tugas

Kegiatan	Nama (NIM)
- Melakukan data cleaning - Melakukan data preprocessing - Membuat error Analysis - Membuat laporan mengenai pembahasan	Akmal Galih Aji Sugmo Seno (18222046)
- Membuat Data Preprocessing - Membuat Data Pipeline - Pembuatan KNN - Membuat Laporan KNN - Membuat Laporan Data Preprocessing	Raizan Iqbal Resi (18222068)
- Melakukan data cleaning - Membuat Implementasi Algoritma Gaussian Naive-Bayes - Membuat laporan Gaussian Naive-Bayes	Muhammad Kevinza Faiz (18222072)
- Membuat Data Cleaning -Membuat Data Preprocessing - Membuat Laporan Data Cleaning dan Preprocessing	Muhammad Raihan Ariffianto (18222092)

Referensi

GeeksforGeeks. *K-Nearest Neighbor (KNN) Algorithm*. Diakses dari https://www.geeksforgeeks.org/k-nearest-neighbours/

Medium. KD-TREE AND BALL TREE IN KNN ALGORITHM. Diakses dari https://medium.com/@narasimharaodevisetti14/kd-tree-and-ball-tree-in-knn-algorithm-09a86d1bc6e6

Medium. Gaussian Naive Bayes: Understanding the Basics and Applications. Diakses dari https://medium.com/@kashishdafe0410/gaussian-naive-bayes-understanding-the-basics-and-applications-52098087b963

GeeksforGeeks. Detect and Remove the Outliers using Python. Diakses dari

Detect and Remove the Outliers using Python - GeeksforGeeks