TUGAS BESAR 2
IF3170 - INTELIGENSI BUATAN
IMPLEMENTASI ALGORITMA

Kelompok 27 :

Michael Leon Putra Widhi	(13521108)
Muhammad Zaki Amanullah	(13521146)
Mohammad Rifqi Farhansyah	(13521166)
Nathan Tenka	(13521172)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2023

1. Deskripsi Singkat

Pada tugas besar ini, Kami melakukan implementasi algoritma pembelajaran mesin KNN dan *Naive-Bayes* (sesuai dengan cakupan materi kuliah IF3170 - Intelegensi Buatan). Data yang digunakan pada implementasi ini sama seperti data tugas kecil 2. Kami melakukan proses pelatihan model menggunakan data latih yang terdapat pada pranala tersebut, kemudian dilakukan validasi hasil dengan menggunakan data validasi untuk mendapatkan *insight* seberapa baik model melakukan generalisasi.

Tahap selanjutnya adalah melakukan perbandingan hasil implementasi algoritma KNN dan *Naive-Bayes* kelompok Kami dengan algoritma milik pustaka eksternal *scikit-learn.* Parameter perbandingan yang digunakan, antara lain: *precision, recall, f1-score, support, accuracy, macro avg,* dan *weighted avg.* Hasil perbandingan akan dibahas kemudian pada bab selanjutnya.

Pada bagian implementasi, kami merealisasikannya sesuai dengan *tree-structure* berikut ini :

```
README.md

data
    data_train.csv
    data_validation.csv
    full_data.csv
    test.csv

result
    predictions-knn.csv
    predictions-naive-bayes.csv

src
    knn.ipynb
    naive.ipynb

    algorithm
        knn.py
        naiveBayes.py
        weightedKnn.py

    models
        knn_model.pkl
        naive_bayes_model.pkl

    utils
        scaler.py
```

Gambar 1.1. *Tree-Structure* Implementasi.

Folder *data* berisi data yang digunakan selama melakukan pembangunan dan validasi model, folder *result* berisi hasil prediksi berdasarkan algoritma KNN dan *Naive-Bayes*, sementara folder *src* berisi implementasi algoritma KNN dan *Naive-Bayes*.

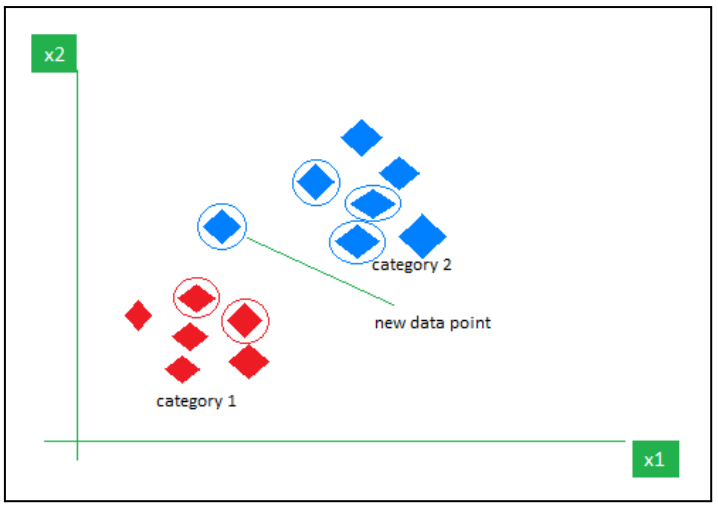
Folder *algorithm* berisi implementasi dari algoritma KNN dan *Naive-Bayes*, folder *models* berisi hasil model yang dibangun, serta folder *utils* berisi kelas yang dibangun untuk melakukan normalisasi terhadap data yang akan digunakan. Komponen-komponen tersebut kemudian akan digunakan untuk melakukan tahapan-tahapan analisis yang tercakup pada file knn.ipynb dan naive.ipynb. Pada kedua file tersebut, akan dilakukan proses analisis mulai dari pemanggilan tiap fungsi hasil implementasi, hasil pemrosesan terhadap data yang digunakan, perbandingan hasil implementasi dan algoritma pustaka eksternal, serta beberapa analisis lainnya.

2. Algoritma *K- Nearest Neighbors* (KNN)

1. Definisi Singkat

K-Nearest Neighbors (KNN) adalah salah satu algoritma klasifikasi yang paling dasar namun penting dalam pembelajaran mesin. KNN merupakan metode pembelajaran mesin intuitif yang seringkali digunakan untuk mengatasi permasalahan klasifikasi dan regresi. Dengan memanfaatkan konsep kemiripan, KNN memprediksi label atau nilai dari titik data baru dengan mempertimbangkan K tetangga terdekatnya dalam *train dataset.*

Algoritma ini seringkali digunakan karena bersifat non-parametrik, artinya algoritma KNN tidak membuat asumsi mendasar tentang distribusi data (berbeda dengan algoritma lain seperti GMM, yang mengasumsikan distribusi Gaussian dari data yang diberikan). KNN hanya menerima beberapa data sebelumnya (atau yang biasa disebut sebagai *train data*), yang mengklasifikasikan koordinat tertentu ke dalam kelompok yang diidentifikasi oleh suatu atribut.



Gambar 2.1.1 Visualisasi Langkah Kerja Algoritma KNN

Algoritma *K-Nearest Neighbors* (KNN) juga seringkali digunakan karena dapat menangani data numerik maupun data kategorikal, sehingga dapat menjadi pilihan yang fleksibel untuk berbagai jenis dataset dalam proses klasifikasi atau regresi. Kelemahan yang dimiliki KNN adalah kurangnya sensitifitas terhadap pencilan jika dibandingkan dengan algoritma lain.

2. Alur Algoritma

Algoritma KNN bekerja dengan mencari K tetangga terdekat dari suatu titik data berdasarkan metrik jarak, seperti metode *Euclidean*, *Manhattan*, dan lain sebagainya. Kelas atau nilai dari titik data kemudian ditentukan oleh mayoritas kelas dari K tetangga tersebut. Pendekatan ini memungkinakan algoritma untuk beradaptasi dengan pola-pola yang berbeda dan membuat prediksi berdasarkan struktur lokal dari data.

Pada algoritma ini, terdapat beberapa variasi metrik jarak, antara lain:

a. *Euclidean Distance*

Metode ini sebenarnya merupakan jarak kartesian antara dua titik yang berada dalam bidang. *Euclidean distance* juga dapat divisualisasikan sebagai panjang garis lurus yang menghubungkan dua titik. Berikut ini adalah rumusan dari *euclidean distance*.

$$distance(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{ij})^2}$$

b. *Manhattan Distance*

Metrik *manhattan distance* umumnya digunakan ketika kita tertarik pada total jarak yang ditempuh oleh objek daripada perpindahannya. Metrik ini dihitung dengan menjumlahkan selisih absolut antara koordinat titik-titik dalam n dimensi. Berikut ini adalah rumusan dari *manhattan distance*.

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

c. *Minkowski Distance*

Metode ini sebenarnya merupakan bentuk umum dari *manhattan dan euclidean distance,* atau dengan kata lain, keduanya merupakan kasus khusus dari *minkowski distance*. Berikut ini adalah rumusan dari *minkowski distance.*

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

Berdasarkan rumusan di atas, kita dapat mengatakan bahwa ketika $p = 2$, maka rumusan itu sama dengan rumus untuk *euclidean distance*, serta apabila $p = 1$, maka kita memperoleh rumus untuk *manhattan distance*.

Pemilihan nilai K (jumlah tetangga terdekat) juga merupakan aspek penting/krusial dalam algoritma KNN. Nilai K dalam algoritma *K-Nearest Neighbors* sebaiknya dipilih berdasarkan data input. Apabila data input memiliki lebih banyak pencilan atau *noise,* nilai k yang lebih tinggi akan lebih baik. Selain itu, disarankan untuk memilih nilai K yang ganjil untuk menghindari hasil klasifikasi yang seimbang.

3. Prosedur

Algoritma *K-Nearest Neighbors* (KNN) beroperasi berdasarkan prinsip kemiripan, metode ini akan memprediksi label atau nilai dari titik data baru dengan mempertimbangkan label atau nilai dari K tetangga terdekatnya dalam *train dataset*. Untuk membuat prediksi, algoritma menghitung jarak antara setiap titik data baru dalam dataset uji dan semua titik dalam *train dataset*. *Euclidean distance* adalah metrik jarak yang umum digunakan dalam KNN, tetapi metrik jarak lainnya, seperti *Manhattan distance* atau *Minkowski distance* juga dapat digunakan tergantung pada masalah dan data yang ada. Setelah jarak antara titik data baru dan semua titik data

dalam *train dataset* dihitung, algoritma melanjutkan untuk menemukan K tetangga terdekat berdasarkan jarak tersebut. Metode khusus untuk memilih tetangga terdekat dapat bervariasi, tetapi pendekatan yang umum digunakan adalah dengan mengurutkan jarak secara berurutan dan memilih K titik data dengan jarak terpendek.

Algoritma kemudian mengidentifikasi prediksi berdasarkan label atau nilai yang terkait dengan tetangga-tetangga tersebut. Pada proses klasifikasi, kelas mayoritas di antara K tetangga tersebut ditetapkan sebagai label yang diprediksi untuk titik data baru. Sementara pada proses regresi, rata-rata dari nilai K tetangga terdekat ditetapkan sebagai nilai yang diprediksi.

Misalkan *X* adalah dataset pelatihan dengan *n* titik data, serta setiap titik data direpresentasikan oleh vektor fitur X_i berdimensi d dan *Y* adalah label atau nilai yang sesuai untuk setiap titik data di *X*. Diberikan titik data baru *x*, maka algoritma akan menghitung jarak antara *x* dan setiap titik data X_i dalam *X* menggunakan metrik jarak tertentu, misalnya *Euclidean distance*.

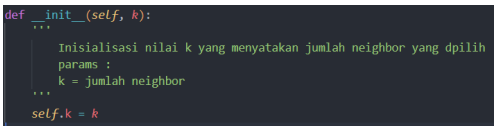
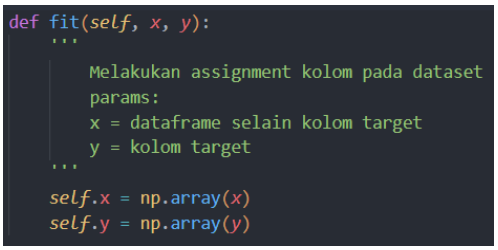
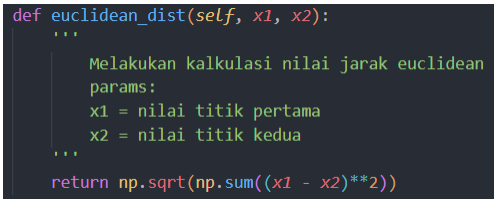
$$distance(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{ij})^2}$$

Algoritma selanjutnya memilih K titik data dari *X* yang memiliki jarak terpendek ke *x*. Pada proses klasifikasi, algoritma menetapkan label y yang paling sering muncul di antara K tetangga terdekat *x*, Sementara pada proses regresi, algoritma menghitung rata-rata dari nilai *y* milik K tetangga terdekat dan menetapkannya sebagai nilai yang diprediksi untuk *x*.

4. Implementasi

Pada file knn.py dalam direktori *src/algorithm,* terdapat kelas yang merupakan implementasi algoritma KNN kelompok kami. Implementasi dimulai dengan mendefinisikan kelas KNNAlgorithm yang berisi fungsi empat fungsi, yaitu: __init__, fit, euclidean_dist, dan predict. Berikut ini adalah penjelasan lebih detail terkait dengan implementasi kelas KNNAlgorithm.

Tabel 2.4.1. Penjelasan Kode Implementasi KNN

No.	Fungsi	Penjelasan
1		Konstruktor dari kelas KNNAlgorithm yang akan dipanggil ketika saat membuat objek baru dari kelas ini. Konstruktor tersebut menerima satu parameter k yang digunakan untuk menginisialisasi atribut self.k dengan nilai yang diberikan.
2		Fungsi yang akan digunakan untuk melakukan assignment kolom pada dataset dengan menginisialisasi atribut self.x dan self.y dengan nilai-nilai yang diberikan.
3		Fungsi yang akan digunakan untuk menghitung jarak antara dua titik dalam ruang n-dimensi menggunakan metode *euclidean*.
4		Fungsi yang akan digunakan untuk melakukan prediksi berdasarkan algoritma K-Nearest Neighbors (KNN) dengan menggunakan sampel data yang diberikan.

3. Algoritma *Naive-Bayes*

1. Deskripsi Singkat

Naive-Bayes adalah algoritma pembelajaran mesin berbasis probabilitas yang digunakan untuk tugas klasifikasi. Algoritma ini mengasumsikan bahwa keberadaan suatu fitur dalam suatu kelas tidak terkait dengan keberadaan fitur lainnya, sebuah konsep yang dikenal sebagai independensi bersyarat. Algoritma ini sangat efektif dan mudah diimplementasikan sehingga membuatnya populer untuk klasifikasi teks, penyaringan spam, dan banyak tugas lain yang memerlukan pembelajaran mesin.

2. Alur Algoritma

Algoritma *Naive-Bayes* beroperasi dengan menggunakan prinsip teorema Bayes untuk memprediksi kelas sebuah objek berdasarkan probabilitas kondisional atributnya. Metode ini mengasumsikan independensi antar atribut dalam sebuah kelas, yang sering disebut sebagai "*naive*". Berikut adalah beberapa komponen utama dalam prosedur *Naive-Bayes*:

a. Probabilitas Kondisional

   Ini adalah ukuran probabilitas suatu atribut, 'A', diketahui kelasnya, 'C'. Dalam *Naive-Bayes*, probabilitas kondisional dihitung untuk setiap atribut secara terpisah. Ini serupa dengan menghitung "jarak" dalam KNN, dimana "jarak" di sini adalah probabilitas atribut terjadi dalam kelas tertentu.

b. Probabilitas Prior

   Probabilitas prior adalah probabilitas murni dari sebuah kelas tanpa mempertimbangkan atribut lainnya. Ini serupa dengan "nilai K" dalam KNN yang menentukan jumlah tetangga terdekat yang akan dipertimbangkan, dimana dalam *Naive-Bayes*, probabilitas prior menetapkan dasar sebelum atribut lain diperhitungkan.

c. Prediksi / Probabilitas Posterior

   Dalam *Naive-Bayes*, prediksi dibuat dengan mengalikan semua probabilitas kondisional atribut yang diamati dengan probabilitas prior kelas, dan kelas dengan nilai terbesar dipilih sebagai prediksi. Ini dapat dianggap analog dengan memilih kelas dengan jarak terdekat dalam KNN.

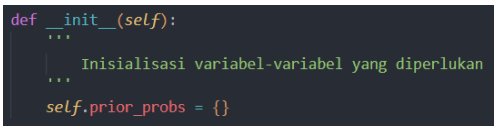
3. Prosedur

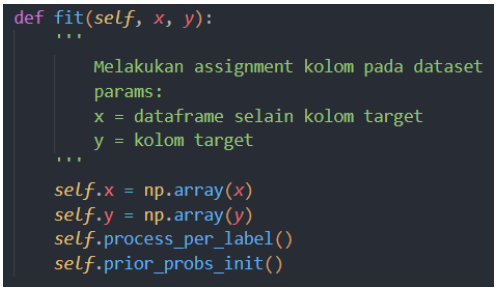
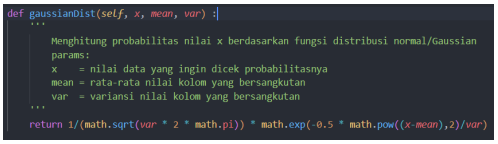
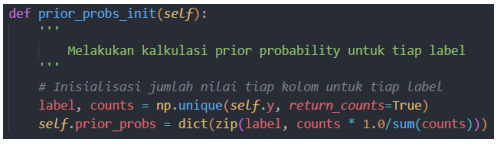
- Algoritma pertama-tama mengukur frekuensi setiap nilai atribut untuk kelas tertentu dan frekuensi setiap kelas itu sendiri. Ini adalah langkah penting karena probabilitas kondisional dan probabilitas prior dari kelas tersebut akan bergantung pada nilai-nilai ini.
- Menentukan probabilitas kondisional ‘P(ai | vj)’ yang merepresentasikan probabilitas atribut ‘ai’ muncul jika diberikan kelas ‘vj’. Ini mengacu pada probabilitas suatu fitur/faktor tertentu terjadi berdasarkan kelasnya.
- Menentukan probabilitas prior ‘P(vj)’ dari kelas ‘vj’ yang merepresentasikan probabilitas umum dari kelas ‘vj’ terjadi dalam dataset.
- Dibangun model probabilitas berdasarkan probabilitas-probabilitas yang telah dihitung sebelumnya, yang akan digunakan untuk membuat prediksi.
- Tahap Klasifikasi (Menentukan data yang tidak terlihat)
  - Menghitung proyeksi probabilitas atribut dari data yang tidak terlihat (query), dihitung untuk setiap kelas dengan mengalikan probabilitas kondisional setiap atribut ‘P(ai | vj)’.
  - Hasil proyeksi tersebut kemudian dikalikan dengan probabilitas kelas yang bersesuaian untuk mendapatkan ‘P(vj | a1, a2, …, an)’, yang merupakan probabilitas dari kelas ‘vj’ jika diketahui deret atribut ‘a1, a2, …, an’.
  - Kelas dengan nilai probabilitas maksimum ‘P(vj | a1, a2, …, an)’ dipilih sebagai prediksi kelas untuk *instance* data yang tidak terlihat.

4. Implementasi

Algoritma *Naive-Bayes* diimplementasikan pada kelas NaiveBayes pada file naiveBayes.py sebagai berikut.

Tabel 3.4.1. Penjelasan Kode Implementasi *Naive-Bayes*

No.	Fungsi	Penjelasan
1		Konstruktor dari kelas NaiveBayes yang akan membuat sebuah *dictionary* ‘prior_probs’ untuk menyimpan probabilitas *prior*.

2		Metode ini menerima 'x' (fitur) dan 'y' (label target) sebagai parameter dan berfungsi untuk memproses data per label dan menginisialisasi probabilitas *prior* dengan metode process_per_label dan prior_probs_init. Metode ini juga melakukan proses assignment kolom pada dataset.
3		Metode yang berfungsi untuk menghitung nilai probabilitas berdasarkan distribusi *Gaussian*.
4		Metode ini digunakan untuk mengelompokkan data berdasarkan label. Metode ini juga berfungsi untuk menyiapkan struktur data untuk menyimpan informasi tersebut.
5		Metode yang berfungsi untuk menghitung probabilitas prior berdasarkan frekuensi label dalam data latih.

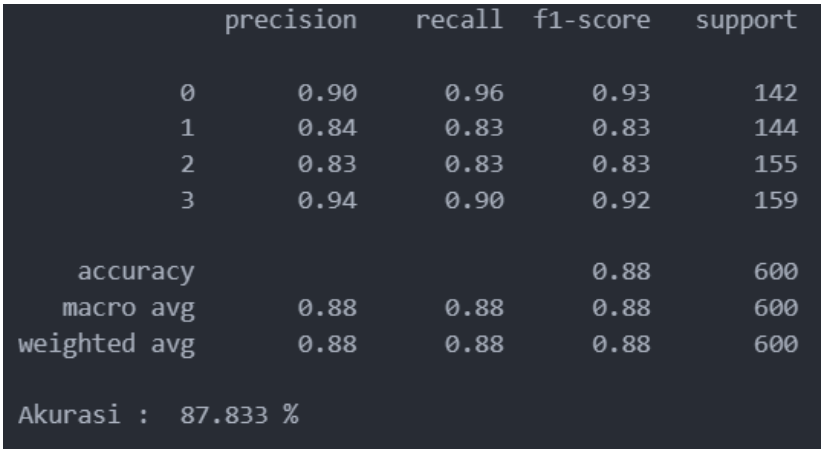
4. Perbandingan Algoritma KNN dengan Pustaka Eksternal

Hasil implementasi algoritma KNN kelompok kami kemudian dibandingkan dengan algoritma KNN milik pustaka eksternal scikit-learn. Berikut ini merupakan nilai hasil pengujian untuk kedua algoritma tersebut:

	precision	recall	f1-score	support
0	0.90	0.96	0.93	142
1	0.84	0.83	0.83	144
2	0.83	0.83	0.83	155
3	0.94	0.90	0.92	159
accuracy			0.88	600
macro avg	0.88	0.88	0.88	600
weighted avg	0.88	0.88	0.88	600

Akurasi : 87.833 %

Gambar 4.1. Nilai Pengujian Algoritma KNN Hasil Implementasi



	precision	recall	f1-score	support
0	0.90	0.96	0.93	142
1	0.84	0.83	0.83	144
2	0.83	0.83	0.83	155
3	0.94	0.90	0.92	159
accuracy			0.88	600
macro avg	0.88	0.88	0.88	600
weighted avg	0.88	0.88	0.88	600

Akurasi : 87.833 %

Gambar 4.2. Nilai Pengujian Algoritma KNN dari Pustaka Eksternal

Hasil pengujian di atas menunjukkan bahwa nilai pengujian yang diperoleh untuk algoritma KNN hasil implementasi dan algoritma KNN dari pustaka eksternal adalah sama. Akan tetapi, hal tersebut tidak dapat dijadikan sebagai tolak ukur bahwa model yang dibangun merupakan alternatif solusi terbaik. Proses kalkulasi untuk mendapatkan model dengan akurasi yang tinggi memerlukan pemilihan nilai k yang baik. Nilai pengujian di atas diperoleh dari percobaan dengan nilai k = 5. Pada percobaan tersebut, diperoleh nilai akurasi sebesar 87.833%. Angka ini tentu saja masih dapat ditingkatkan dengan melakukan pemilihan nilai k yang dapat memaksimalkan nilai akurasi.

Pemilihan nilai terbaik untuk *hyperparameter* ‘k’ pada algoritma KNN melibatkan proses yang disebut sebagai *hyperparameter tuning*. Salah satu pendekatan yang umum dilakukan adalah dengan menggunakan *cross-validation*. Pendekatan tersebut terdiri atas beberapa langkah, antara lain:

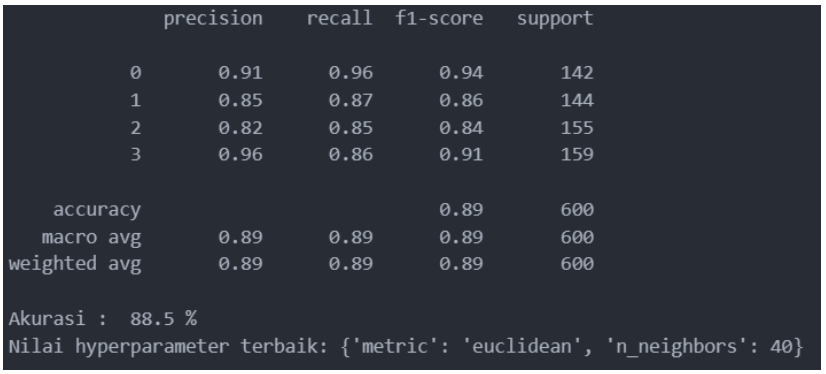
1. **Pemilihan jumlah subset (*Fold*)**

   Dataset akan dibagi menjadi beberapa subset yang disebut sebagai ‘*fold’*. Setiap bagian ‘*fold’* akan digunakan sebagai subset pengujian satu kali, sementara sisanya akan digunakan sebagai subset pelatihan.
2. **Pelatihan dan Pengujian Berulang**

   Model pembelajaran mesin dilatih pada subset pelatihan dan diuji pada subset pengujian untuk setiap iterasi *cross-validation*. Proses ini dilakukan sebanyak jumlah *fold* yang telah ditentukan.
3. **Perhitungan Metrik Kinerja**

   Metrik kinerja seperti akurasi, presisi, *recall,* atau *F1-Score* dihitung untuk setiap iterasi *cross-validation*. Metrik ini memberikan gambaran tentang seberapa baik kinerja model pada berbagai subset data.

Kelompok kami kemudian menggunakan pendekatan *cross-validation* untuk meningkatkan akurasi model kami. *Cross-validating* dilakukan dengan menggunakan pustaka eksternal milik *scikit-learn.* Secara umum, proses yang dilakukan adalah *Grid-Search* dengan rentang nilai 1 hingga 500. Hasil nilai K dan model kemudian digunakan untuk melakukan pengujian kembali terhadap algoritma KNN yang telah diimplementasikan sebelumnya.



	precision	recall	f1-score	support
0	0.91	0.96	0.94	142
1	0.85	0.87	0.86	144
2	0.82	0.85	0.84	155
3	0.96	0.86	0.91	159
accuracy			0.89	600
macro avg	0.89	0.89	0.89	600
weighted avg	0.89	0.89	0.89	600

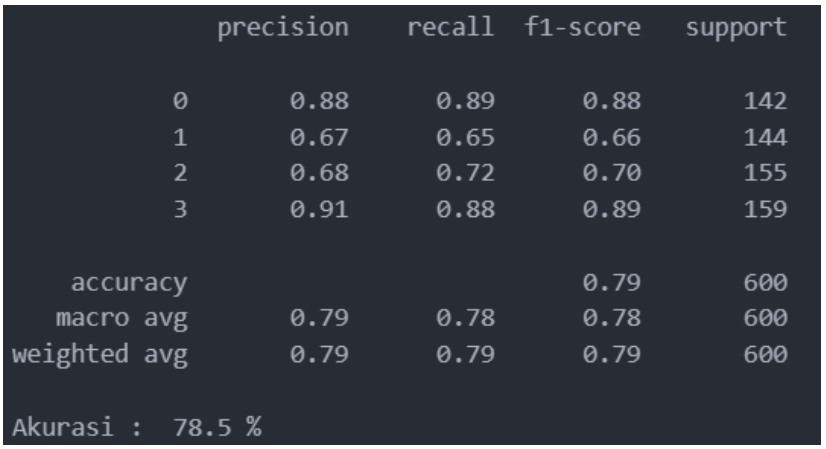
Akurasi : 88.5 %
Nilai hyperparameter terbaik: {'metric': 'euclidean', 'n_neighbors': 40}

Gambar 4.3. Nilai Pengujian Algoritma KNN setelah Optimasi

Pada gambar 4.3, terlihat bahwa dengan melakukan *cross-validation* pada rentang nilai k dari 1 hingga 500, diperoleh nilai k terbaik adalah 40 dengan akurasi sebesar 88.5%, dengan kata lain, nilai akurasi mengalami peningkatan sebesar 0.667%.

5. Perbandingan Algoritma *Naive-Bayes* dengan Pustaka Eksternal

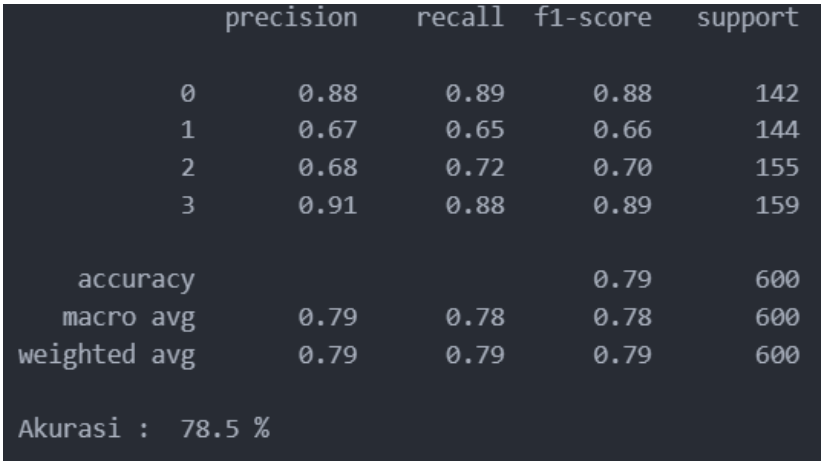
Hasil implementasi algoritma *Naive-Bayes* kemudian akan dibandingkan performanya dengan algoritma *Naive-Bayes* pada pustaka *scikit-learn*. Hasil dari analisis perbandingan tersebut adalah sebagai berikut.



	precision	recall	f1-score	support
0	0.88	0.89	0.88	142
1	0.67	0.65	0.66	144
2	0.68	0.72	0.70	155
3	0.91	0.88	0.89	159
accuracy			0.79	600
macro avg	0.79	0.78	0.78	600
weighted avg	0.79	0.79	0.79	600

Akurasi : 78.5 %

Gambar 5.1. Nilai Pengujian Algoritma *Naive-Bayes* Hasil Implementasi



	precision	recall	f1-score	support
0	0.88	0.89	0.88	142
1	0.67	0.65	0.66	144
2	0.68	0.72	0.70	155
3	0.91	0.88	0.89	159
accuracy			0.79	600
macro avg	0.79	0.78	0.78	600
weighted avg	0.79	0.79	0.79	600

Akurasi : 78.5 %

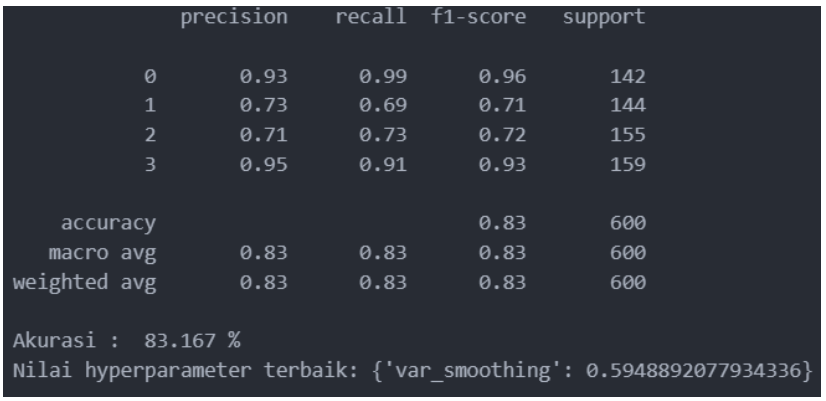
Gambar 5.2. Nilai Pengujian Algoritma *Naive-Bayes* dari Pustaka Eksternal

Berdasarkan hasil tersebut, kedua model tampaknya berkinerja cukup baik dan secara signifikan mirip, yang menunjukkan bahwa implementasi mandiri berhasil mencapai hasil yang sebanding dengan algoritma yang sudah ada di pustaka eksternal. Semua kelas tampaknya memiliki jumlah sampel yang seimbang (*support*), yang menguntungkan dalam pengujian model karena tidak ada bias yang berlebihan terhadap satu kelas tertentu. Meskipun kelas 0 dan 3 memiliki kinerja yang baik, kelas 1 dan 2 memiliki ruang untuk peningkatan, terutama dalam *precision* dan *recall*.

Pendekatan optimasi yang diambil untuk meningkatkan model *Naive-Bayes* adalah dengan menggunakan teknik *Grid-Search* untuk menemukan nilai optimal dari parameter var_smoothing yang merupakan bagian dari implementasi *Gaussian Naive-Bayes* dari pustaka serta implementasi mandiri. Grid Search merupakan teknik yang sistematis untuk menelusuri berbagai kombinasi parameter untuk menemukan yang terbaik untuk model tertentu. Dalam kasus ini ‘GridSearchCV’ dari scikit-learn digunakan untuk menelusuri 400 nilai berbeda dari ‘var_smoothing’, yang berkisar antara 1 hingga 10^-9 secara eksponensial. Parameter ‘var_smoothing’ merupakan parameter yang menambahkan nilai kecil ke varians nol, yang dapat menyebabkan pembagian dengan nol dalam kalkulasi probabilitas.

Dengan menyesuaikan parameter ini, model menjadi lebih *robust* terhadap fitur dengan varian yang sangat rendah. Digunakan metode cross-validation menggunakan 10-fold cross-validation (‘cv=10’) untuk memastikan bahwa penyesuaian parameter tidak hanya baik untuk satu subset data tetapi secara konsisten baik di seluruh data. Untuk dapat melakukan proses grid search secara paralel digunakan sebuah teknik paralelisasi untuk menggunakan semua core CPU yang tersedia untuk mempercepat pencarian. Hasil

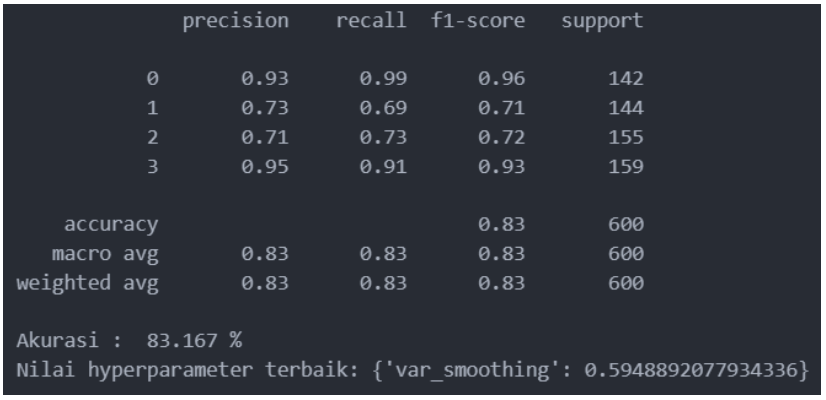
dari optimasi menggunakan metode grid search untuk menemukan nilai ‘var_smoothing’ yang optimal adalah sebagai berikut.



	precision	recall	f1-score	support
0	0.93	0.99	0.96	142
1	0.73	0.69	0.71	144
2	0.71	0.73	0.72	155
3	0.95	0.91	0.93	159
accuracy			0.83	600
macro avg	0.83	0.83	0.83	600
weighted avg	0.83	0.83	0.83	600

Akurasi : 83.167 %
Nilai hyperparameter terbaik: {'var_smoothing': 0.5948892077934336}

Gambar 5.3. Optimasi algoritma *Naive-Bayes* dari pustaka



	precision	recall	f1-score	support
0	0.93	0.99	0.96	142
1	0.73	0.69	0.71	144
2	0.71	0.73	0.72	155
3	0.95	0.91	0.93	159
accuracy			0.83	600
macro avg	0.83	0.83	0.83	600
weighted avg	0.83	0.83	0.83	600

Akurasi : 83.167 %
Nilai hyperparameter terbaik: {'var_smoothing': 0.5948892077934336}

Gambar 5.4. Optimasi algoritma *Naive-Bayes* implementasi mandiri

Berdasarkan output yang didapat, akurasi model dari pustaka meningkat menjadi 83.167% dari 78.5% sedangkan model implementasi mandiri meningkat menjadi 83.167% dari 78.5%. Secara spesifik, kelas 1 dan 2 pada implementasi mandiri dan implementasi dari pustaka memiliki peningkatan terbesar dari optimasi dengan peningkatan pada semua metrik (*precision*, *recall*, *f1-score*). Sedangkan nilai *precision, recall,* dan *f1-score* juga memiliki peningkatan pada level 0 dan 3 walaupun tidak se-signifikan level 1 dan 2.

Secara keseluruhan dapat ditunjukkan bahwa implementasi mandiri, ketika dikalibrasi dengan parameter yang tepat, dapat bersaing dengan implementasi dari pustaka yang sudah ada. Kinerja model menunjukkan bahwa tuning parameter yang tepat sangat penting dalam model yang signifikan dalam kinerja model.

6. Optimalisasi Algoritma KNN

1. Ide Awal

Implementasi algoritma sudah berhasil dilakukan, tetapi tentu saja, untuk menghasilkan model yang memiliki kualitas baik diperlukan sebuah pemrosesan terhadap prosedur yang telah didefinisikan dan dilakukan sebelumnya.

Optimasi model KNN yang akan dilakukan dibuat melalui beberapa langkah, meliputi pemilihan fitur, penentuan nilai K terbaik, hingga modifikasi sederhana terhadap algoritma KNN yang telah dibuat.

a. Pemilihan fitur

Melalui hasil eksplorasi yang kami lakukan, tidak semua fitur (dalam hal ini kolom) memiliki kontribusi yang besar atau bahkan memiliki kontribusi terhadap penentuan nilai pada kolom target. Oleh sebab itu, penting untuk melakukan pemilihan fitur yang memiliki kontribusi besar terhadap proses *clustering* kolom target. Berdasarkan hasil *Exploratory Data Analysis* yang kami lakukan pada Tugas Kecil 2, didapati bahwa kolom yang memiliki koefisien korelasi yang lebih besar dari 0.10 dengan kolom target adalah kolom ram (0.918), battery_power (0.184), px_width (0.178), dan px_height (0.158). Sisa kolom selain keempat kolom tersebut tidak kami gunakan dalam proses pembangunan model.

b. Penentuan nilai K terbaik

Seperti yang telah dibahas pada bagian analisis, bahwa penentuan nilai K memiliki pengaruh yang besar terhadap kualitas model. Nilai K tidak boleh terlalu besar maupun terlalu kecil untuk menghasilkan *clustering* yang baik oleh sebab itu, dilakukan penentuan nilai K terbaik menggunakan metode *elbow curve* dengan nilai setiap titik pada kurva menyatakan nilai *error* dari masing-masing nilai K yang mungkin.

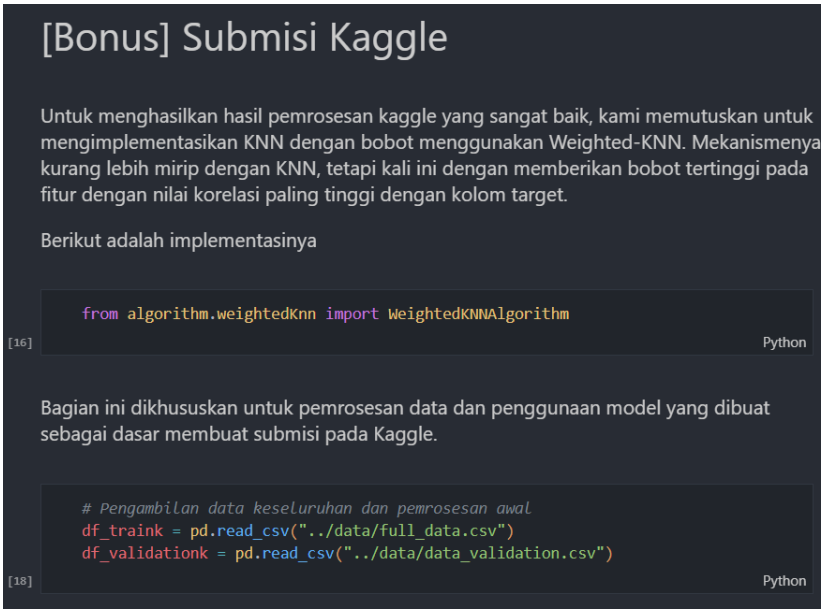
c. Modifikasi sederhana algoritma KNN

Kami menyadari bahwa kontribusi dari masing-masing fitur tidak sepenuhnya dapat dikatakan terdistribusi merata. Hal ini dapat dilihat melalui nilai koefisien korelasi kolom ram (korelasi tertinggi) bernilai jauh lebih besar dari kolom kedua terbesar, battery_power. Oleh sebab itu, penting untuk melakukan pembobotan terhadap fitur yang memiliki nilai korelasi yang lebih tinggi untuk menghasilkan hasil yang lebih akurat. Dalam hal ini, pembobotan dilakukan

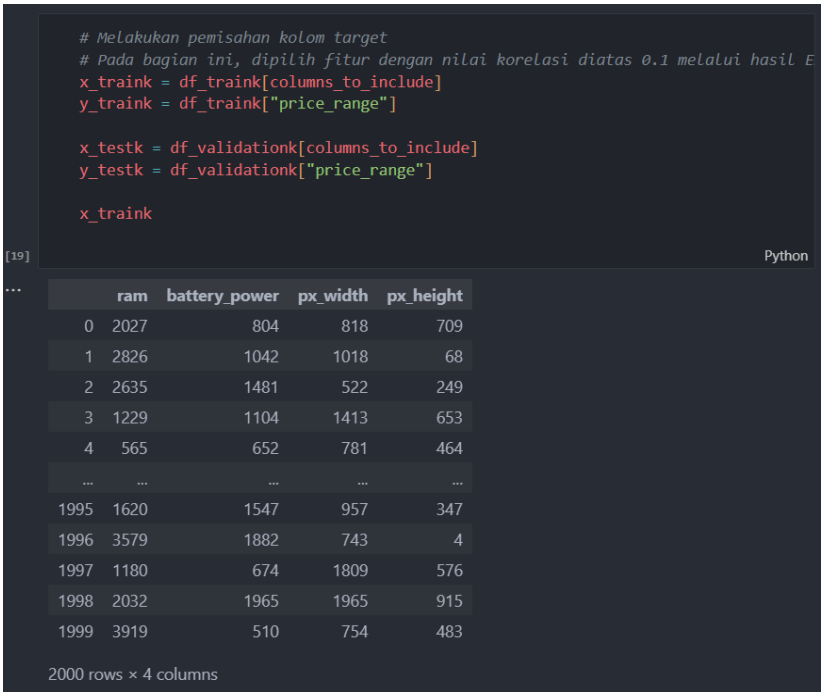
dengan nilai 0.7 untuk fitur dengan nilai korelasi tertinggi dengan kolom target dan sisa tiga kolom dengan nilai 0.1.

2. Pemrosesan

Berikut adalah seluruh implementasi yang dilakukan sebagai usaha optimalisasi algoritma KNN yang telah dibentuk.



Gambar 6.1. Pemrosesan optimalisasi KNN: *import* pustaka dan pengambilan data



Gambar 6.2. Pemrosesan optimalisasi KNN: Pemisahan kolom target

```
# Dilakukan standarisasi dengan scaler yang dibuat mandiri
from utils.scaler import Scaler

scaler = Scaler()
x_traink = scaler.fit_transform(x_traink)
x_testk = scaler.transform(x_testk)

x_traink
```

[20] Python

	ram	battery_power	px_width	px_height
0	0.473276	0.202405	0.212283	0.361735
1	0.686799	0.361389	0.345794	0.034694
2	0.635756	0.654643	0.014686	0.127041
3	0.260021	0.402806	0.609479	0.333163
4	0.082576	0.100868	0.187583	0.236735
...	...	...	...	...
1995	0.364511	0.698731	0.305073	0.177041
1996	0.888028	0.922512	0.162216	0.002041
1997	0.246927	0.115564	0.873832	0.293878
1998	0.474613	0.977956	0.977971	0.466837
1999	0.978888	0.006012	0.169559	0.246429

2000 rows × 4 columns

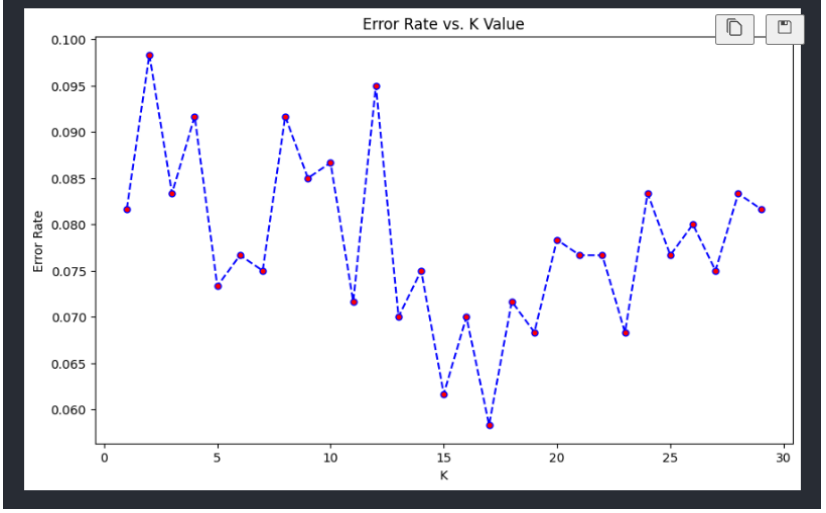
Gambar 6.3. Pemrosesan optimalisasi KNN: Standarisasi

```
# Menggunakan nilai weighthing yang tinggi pada korelasi tinggi
weights = [0.7, 0.1, 0.1, 0.1]

# Menggunakan elbow diagram
error_rate = []
for i in range(1,30):
    knn = WeightedKNNAlgorithm(k=i, weights=weights)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))

# Plotting elbow diagram
plt.figure(figsize=(10,6))
plt.plot(range(1,30),error_rate,color='blue', linestyle='dashed', marker='o', mar
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[22] Python

Gambar 6.4. Pemrosesan optimalisasi KNN: weighting dan plotting

```
Text(0, 0.5, 'Error Rate')
```



Gambar 6.5. Pemrosesan optimalisasi KNN: Elbow diagram

```python
# Penggunaan hasil pada data test
test_data = pd.read_csv('../data/test.csv')
features = test_data[columns_to_include]

# Proses normalisasi data
scaler_final = Scaler()
feature_test = scaler.fit_transform(features)

feature_test
```

[23] Python

	ram	battery_power	px_width	px_height
0	0.197690	0.384461	0.543434	0.064990
1	0.827558	0.541192	0.794613	0.784067
2	0.952995	0.946417	0.839731	0.137841
3	0.782971	0.056932	0.057912	0.065514
4	0.117916	0.685867	0.740067	0.077044
...	...	...	...	...
1995	0.370132	0.896852	0.789226	0.685535
1996	0.333602	0.376423	0.141414	0.053459
1997	0.950846	0.633624	0.097643	0.179245
1998	0.165995	0.390489	0.268013	0.061845
1999	0.929089	0.795044	0.562290	0.128407

2000 rows × 4 columns

Gambar 6.6. Pemrosesan optimalisasi KNN: Normalisasi

```python
# Gunakan model KNN terbaik yang sebelumnya dibangun
knn_kaggle = WeightedKNNAlgorithm(k=17, weights=weights)  # Memilih menggunaka

# Lakukan fit model
knn_kaggle.fit(x_traink, y_traink)

# Lakukan prediksi model
predictions = knn_kaggle.predict(feature_test)

# Membuat dataframe hasil
result_df = pd.DataFrame({'id': test_data['id'], 'price_range': predictions})

# Menyimpan dataframe dalam csv
result_df.to_csv('../result/predictions-knn.csv', index=False)
```

Gambar 6.7. Pemrosesan optimalisasi KNN: Penggunaan model KNN dan penyimpanan *dataframe* dalam format *csv*

```python
import numpy as np

class WeightedKNNAlgorithm:
    '''
    Kelas yang mengimplementasikan Algoritma Weighted KNN from scrat
    '''
    def __init__(self, k, weights):
        '''
            Inisialisasi nilai k yang menyatakan jumlah neighbor yan
            params :
            k = jumlah neighbor
        '''
        self.k = k
        self.weights = weights

    def fit(self, x, y):
        '''
            Melakukan assignment kolom pada dataset
            params:
            x = dataframe selain kolom target
            y = kolom target
        '''
        self.x = np.array(x)
        self.y = np.array(y)

    def euclidean_dist(self, x1, x2):
        '''
            Melakukan kalkulasi nilai jarak euclidean
            params:
            x1 = nilai titik pertama
            x2 = nilai titik kedua
        '''
        return np.sqrt(np.sum(self.weights * (x1 - x2)**2, axis=1))
```

Gambar 6.8. Pemrosesan optimalisasi KNN: Kelas WeightedKNNAlgorithm

```python
    def predict(self, x):
        '''
            Melakukan prediksi berdasarkan Algoritma KNN
            params:
            x = sampel data
        '''
        x = np.array(x)
        y_pred = []

        # Proses kalkulasi utama
        for sample in x:            # Untuk setiap sampel data
            distances = []          # Untuk menyimpan nilai jarak

            # Lakukan kalkulasi jarak antara sampel dengan setiap data tra
            for train_sample, train_label in zip(self.x, self.y):
                distance = self.euclidean_dist(sample, train_sample)
                distances.append((train_sample, train_label, distance))

            distances.sort(key=lambda x: x[2])      # Urutkan berdasarkan
            neighbors = distances[:self.k]          # Ambil k tetangga ter

            # Ambil nilai label dari setiap tetangga tersebut
            labels = [neighbor[1] for neighbor in neighbors]
            unique_labels, counts = np.unique(labels, return_counts=True)

            # Pilih yang memiliki frekuensi paling tinggi
            y_pred.append(unique_labels[np.argmax(counts)])

        # Kembalikan nilai prediksi
        return y_pred
```

Gambar 6.9. Pemrosesan optimalisasi KNN: Metode predict pada kelas WeightedKNNAlgorithm

7. Optimalisasi Algoritma *Naive-Bayes*

1. Ide Awal

Implementasi algoritma sudah berhasil dilakukan. Akan tetapi, masih perlu dilakukan optimasi algoritma *Naive-Bayes*. Optimasi yang dilakukan meliputi pemilihan fitur dan penentuan nilai var_smoothing terbaik.

a. Pemilihan fitur

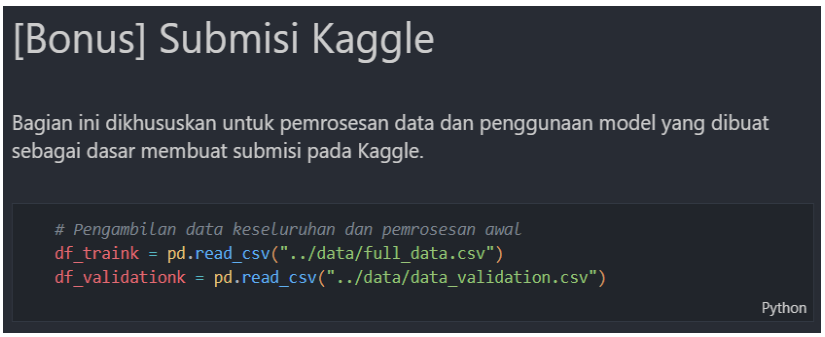
Pemilihan fitur sama dengan pada pemilihan fitur algoritma KNN.

b. Penentuan nilai var_smoothing terbaik

Seperti penjelasan pada bagian analisis, penentuan nilai var_smoothing bisa memiliki pengaruh yang cukup besar terhadap kualitas model. Nilai var_smoothing yang tepat bisa meningkatkan akurasi model secara signifikan. Nilai var_smoothing dilakukan menggunakan *10-fold cross-validation* pada 400 nilai yang terdistribusi dalam rentang 1 hingga 10^-9 secara eksponensial.

2. Pemrosesan

Berikut adalah seluruh implementasi yang dilakukan sebagai usaha optimalisasi algoritma *Naive-Bayes* yang telah dibentuk.



Gambar 7.1. Pemrosesan algoritma *Naive-Bayes* untuk submisi *kaggle*: pengambilan data

```python
# Melakukan pemisahan kolom target
# Pada bagian ini, dipilih fitur dengan nilai korelasi diatas 0.1 melalui hasil El
x_traink = df_traink[columns_to_include]
y_traink = df_traink["price_range"]

x_testk = df_validationk[columns_to_include]
y_testk = df_validationk["price_range"]

x_traink
```

Python

	ram	battery_power	px_width	px_height
0	2027	804	818	709
1	2826	1042	1018	68
2	2635	1481	522	249
3	1229	1104	1413	653
4	565	652	781	464
...	...	...	...	...
1995	1620	1547	957	347
1996	3579	1882	743	4
1997	1180	674	1809	576
1998	2032	1965	1965	915
1999	3919	510	754	483

2000 rows × 4 columns

Gambar 7.2. Pemrosesan algoritma *Naive-Bayes* untuk submisi *kaggle*: pemisahan kolom target

```python
# Dilakukan standarisasi dengan scaler yang dibuat mandiri
from utils.scaler import Scaler

scaler = Scaler()
x_traink = scaler.fit_transform(x_traink)
x_testk = scaler.transform(x_testk)

x_traink
```

Python

	ram	battery_power	px_width	px_height
0	0.473276	0.202405	0.212283	0.361735
1	0.686799	0.361389	0.345794	0.034694
2	0.635756	0.654643	0.014686	0.127041
3	0.260021	0.402806	0.609479	0.333163
4	0.082576	0.100868	0.187583	0.236735
...	...	...	...	...
1995	0.364511	0.698731	0.305073	0.177041
1996	0.888028	0.922512	0.162216	0.002041
1997	0.246927	0.115564	0.873832	0.293878
1998	0.474613	0.977956	0.977971	0.466837
1999	0.978888	0.006012	0.169559	0.246429

2000 rows × 4 columns

Gambar 7.3. Pemrosesan algoritma *Naive-Bayes* untuk submisi *kaggle*: standarisasi

```python
# Penggunaan hasil pada data test
test_data = pd.read_csv('../data/test.csv')
features = test_data[columns_to_include]

# Proses normalisasi data
scaler_final = Scaler()
feature_test = scaler.fit_transform(features)

feature_test
```

Python

	ram	battery_power	px_width	px_height
0	0.197690	0.384461	0.543434	0.064990
1	0.827558	0.541192	0.794613	0.784067
2	0.952995	0.946417	0.839731	0.137841
3	0.782971	0.056932	0.057912	0.065514
4	0.117916	0.685867	0.740067	0.077044
...	...	...	...	...
1995	0.370132	0.896852	0.789226	0.685535
1996	0.333602	0.376423	0.141414	0.053459
1997	0.950846	0.633624	0.097643	0.179245
1998	0.165995	0.390489	0.268013	0.061845
1999	0.929089	0.795044	0.562290	0.128407

2000 rows × 4 columns

Gambar 7.4. Pemrosesan algoritma *Naive-Bayes* untuk submisi *kaggle*: penggunaan hasil data pada data test dan normalisasi

```python
# Gunakan model terbaik yang sebelumnya dibangun
nb_kaggle = NaiveBayes()

# Lakukan fit model
nb_kaggle.fit(x_traink, y_traink)

# Lakukan prediksi dengan data validation
predictions = nb_kaggle.predict(feature_test, best_param)

# Membuat dataframe hasil
result_df = pd.DataFrame({'id': test_data['id'], 'price_range': predictions})

# Menyimpan dataframe dalam csv
result_df.to_csv('../result/predictions-naive-bayes.csv', index=False)
```

Gambar 7.5. Pemrosesan algoritma *Naive-Bayes* untuk submisi *kaggle*: *fitting*, pembuatan prediksi, serta pembuatan file *csv*.

8. Submisi *Kaggle* [Bonus]

a. Penggunaan Data *Test*

Data yang akan diprediksi merupakan data *test.csv* yang terdapat dalam *Kaggle*. Data tersebut terdiri atas 2.000 baris dengan atribut yang sama kecuali memiliki kolom id dan tidak memiliki kolom target price_range. Sebelum data yang akan diprediksi digunakan, dilakukan pemrosesan dengan urutan prosedur yang sama dengan data yang dijadikan sebagai data latihan, mulai dari pemilihan fitur, hingga normalisasi.

b. Persiapan csv submisi

Berdasarkan sampel contoh penulisan data submisi yang terdapat pada *Kaggle*, maka submisi akan disesuaikan dengan format tersebut. Terdapat kolom id dan price_target, kolom id diperoleh dari data *test* sedangkan kolom price_target diperoleh dari hasil prediksi kluster dari masing-masing algoritma. Adapun penyusunan CSV dilakukan dengan membuat sebuah *dataframe* baru yang kemudian disimpan dengan format nama predictions-knn.csv untuk algoritma KNN dan predictions-naive-bayes.csv untuk algoritma *Naive-Bayes*.

9. Penyimpanan Model

Penyimpanan model dilakukan menggunakan pustaka pickle. Model dituliskan ke dalam file knn_model .pkl dan naive_bayes_model.pkl. Kode pembuatan dan pengujian model yang telah disimpan sebagai berikut :

```python
import pickle

# Menyimpan model dalam pkl
model_pkl_file = "models/naive_bayes_model.pkl"

with open(model_pkl_file, 'wb') as file:
    pickle.dump(nb_scratch, file)
```

```python
# Load kembali model dari pkl
with open(model_pkl_file, 'rb') as file:
    model = pickle.load(file)

# Melakukan prediksi dengan model tersebut
y_pickle = model.predict(x_test, best_param)

# Menguji hasil akurasi model
print(classification_report(y_test, y_pickle))
```

```
              precision    recall  f1-score   support

           0       0.93      0.99      0.96       142
           1       0.73      0.69      0.71       144
           2       0.71      0.73      0.72       155
           3       0.95      0.91      0.93       159

    accuracy                           0.83       600
   macro avg       0.83      0.83      0.83       600
weighted avg       0.83      0.83      0.83       600
```

Gambar 9.1. dan 9.2. Penyimpanan dan pengujian model *Naive-Bayes*

Kode yang dicantumkan hanya untuk *Naive-Bayes* karena penyimpanan model dan pengujian model juga sama untuk algoritma KNN.

10. Kesimpulan

Berdasarkan hasil laporan di atas, kesimpulan yang dapat diambil adalah bahwa dengan pemahaman yang mendalam tentang algoritma dan teknik optimasi parameter yang tepat, implementasi mandiri dapat mencapai kinerja yang kompatibel dengan *pustaka* eksternal yang sudah mapan. Ini menunjukkan kematangan dalam pemahaman konsep dan aplikasi praktis dalam bidang pembelajaran mesin khususnya *supervised learning*.

11. Kontribusi Anggota

Tabel 11.1. Kontribusi Tiap Anggota

NIM	Nama	Algoritma	Kontribusi
13521108	Michael Leon Putra Widhi	KNN	Implementasi
13521146	Muhammad Zaki Amanullah	*Naive-Bayes*	Laporan
13521166	Mohammad Rifqi Farhansyah	KNN	Laporan
13521172	Nathan Tenka	*Naive-Bayes*	Implementasi

Repository : https://github.com/mikeleo03/Supervised-Learning-Algorithm