BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Magang telah dilaksanakan pada IT System & Development Department di divisi Data Analyst (DA). Divisi tersebut terlibat dalam proyek yang berhubungan dengan *machine learning*, *database*, dan *deep learning*. *Data Scientist* dan *Data Engineer* menjadi peran utama dalam divisi DA. *Data Engineer* memiliki kewajiban dalam pengelolaan *database* perusahaan, termasuk pembuatan, pembaruan, manipulasi, dan penghabusan *database*, data, atau tabel. Selain itu, *Data Engineer* juga harus menjaga integritas *database* dan mengatur akses-akses perusahaan bagi rekan kerja dengan hak akses tertentu. *Data Scientist* bertanggung jawab dalam melakukan pengambilan data dari berbagai sumber, termasuk *database* melalui *Structured Query Language* (SQL), serta melakukan analisis mendalam terhadap beragam proyek. Tugas ini mencakup pembuatan visualisasi data untuk mempermudah interpretasi hasil analisis, serta pengembangan model berbasis *machine learning* atau *deep learning* untuk mengotomatisasi proses tertentu. Selain itu, *Data Scientist* juga berperan dalam melakukan evaluasi terhadap berbagai model sehingga menghasilkan sebuah strategi yang ideal dan efisien. Kedudukan kegiatan magang ini berada pada peran *Data Scientist*.

Struktur koordinasi pada magang terdiri dari beberapa tahap. Pada tahap pertama, proyek yang diberikan akan dibimbing oleh seorang mentor yang berkewajiban untuk memberikan arahan, membantu jika terjadi kendala, memberikan masukan terhadap pekerjaan, dan memastikan hasil dari pekerjaan sudah sesuai standar yang diinginkan. Setelah itu, hasil yang ditempuh dalam pengerjaan proyek yang sudah diselesaikan akan dipresentasikan kepada *supervisor*, yaitu asisten manajer dalam divisi DA, untuk mengevaluasi proyek serta menentukan tindakan lanjutan. *Supervisor* berhak untuk menyampaikan persetujuan akhir (ACC) terhadap proyek setelah evaluasi menyeluruh sebelum implementasikan dalam skala luas.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama masa magang adalah merancang dan klasifikasi sentimen konsumen berdasarkan ulasan toko Gramedia

JABODETABEK. Berdasarkan pembahasan pada Bab 1, sistem analisis tersebut dirancang dengan menggunakan model mBERT sebagai model prediksi. Model ini menerapkan strategi klasifikasi dan memerlukan *dataset* yang telah memiliki label. Namun, karena *dataset* yang tersedia belum memiliki label, maka perlu dikembangkan sebuah sistem pelabelan otomatis pada *dataset* berukuran besar melalui proses *sentiment labelling* berbasis *lexicon*, sebagaimana telah dijelaskan pada Bab 1.

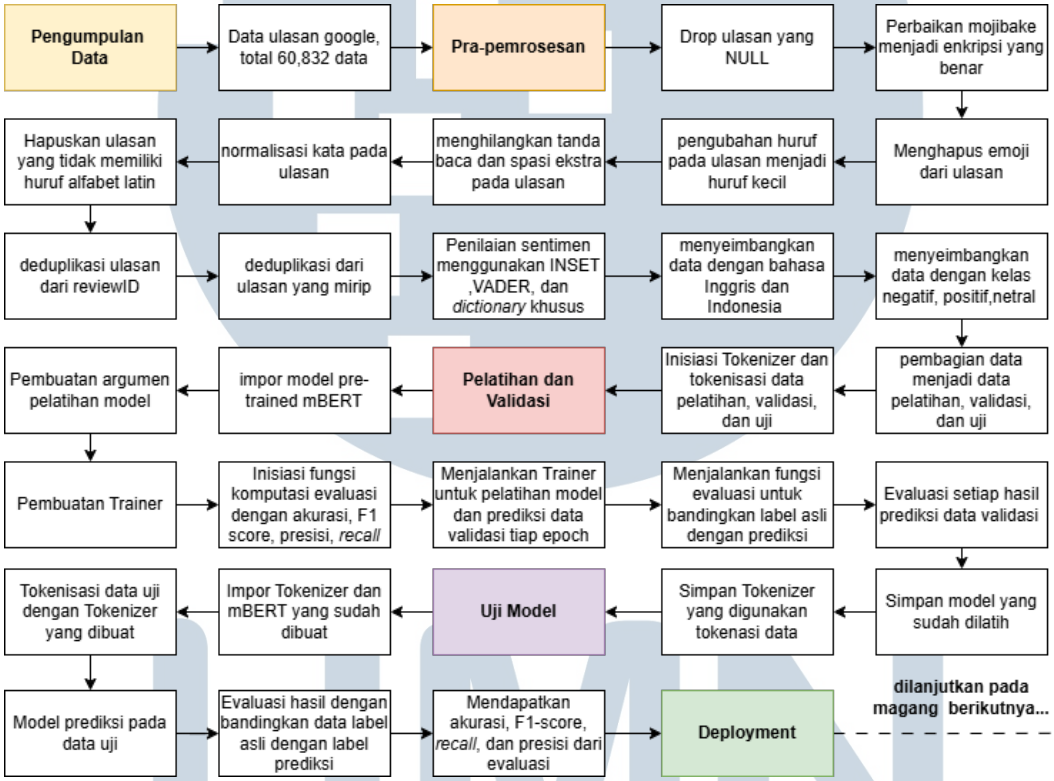
Pemilihan *Multilingual BERT* didasarkan pada karakteristik pelanggan Gramedia JABODETABEK yang umumnya menggunakan bahasa Indonesia dan bahasa Inggris dalam ulasannya. Berdasarkan hasil diskusi dengan tim *Data Analyst* yang telah melakukan riset terhadap toko-toko tersebut, disimpulkan bahwa analisis sentimen harus mampu memahami kedua bahasa tersebut secara akurat. Oleh karena itu, model BERT tidak dapat digunakan secara sembarangan, melainkan harus menggunakan model yang telah melalui proses *pre-trained* dengan korpus berbahasa Indonesia dan Inggris, sehingga *Multilingual BERT* menjadi pilihan yang paling sesuai.

Selain itu, sistem *sentiment labelling* yang dikembangkan juga harus mampu memproses teks dalam Bahasa Indonesia dan Bahasa Inggris. *Dataset* yang digunakan diperoleh dari ulasan *Google Review* untuk seluruh toko Gramedia JABODETABEK. Ulasan tersebut terdiri dari teks dalam kedua bahasa tersebut dengan tujuan sistem yang dikembangkan dapat memberikan hasil analisis sentimen yang akurat dan relevan, baik untuk teks berbahasa Indonesia maupun Inggris.

3.3 Uraian Pelaksanaan Magang

Dalam perancangan dan pembuatan sistem klasifikasi sentimen konsumen, terdapat empat tahap utama, yaitu tahap pengumpulan data, tahap pra-pemrosesan data, tahap pelatihan, validasi, dan pengujian model, dan tahap *deployment*. Tahap pengumpulan data dilakukan untuk memperoleh data yang relevan, baik dari sumber internal perusahaan maupun dari sumber eksternal, yang nantinya digunakan untuk melatih model sesuai kebutuhan proyek. Tahap pra-pemrosesan mencakup pembersihan, penyesuaian format, dan transformasi struktur data agar data tersebut lebih konsisten dan mudah diproses pada tahap pelatihan model. Tahap pelatihan, validasi, dan pengujian bertujuan untuk membangun model, mengevaluasi kinerjanya, serta memastikan bahwa model tersebut memenuhi standar performa yang diharapkan sebelum digunakan. Evaluasi pada tahap

validasi dan tahap pengujian dengan cara membandingkan label prediksi dengan label asli pada data validasi dan uji. Terakhir, pada tahap *deployment*, walau akan dilaksanakan pada magang selanjutnya dan tidak akan terlampirkan pada laporan, model yang telah lolos pengujian diintegrasikan ke dalam sistem produksi agar dapat digunakan secara langsung dalam operasional. Bagan keseluruhan perancangan dan pembuatan sistem klasifikasi sentimen konsumen telah dilampirkan pada Gambar 3.1.



Gambar 3.1. Bagan proyek perancangan dan pembuatan sistem klasifikasi sentimen konsumen

Dalam pengerjaan pembuatan sistem klasifikasi sentimen konsumen dirancang beberapa strategi, bukan hanya satu. Strategi yang dicantumkan dalam laporan merupakan strategi yang paling optimal dan telah diimplementasikan. Proses pengerjaan juga berlangsung dengan waktu yang tidak konsisten karena adanya proyek lain yang harus diselesaikan selama masa pelaksanaan kerja magang. Uraian pelaksanaan proyek sistem analisis tersebut disajikan pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Tanggal	Pekerjaan yang dilakukan
8 September 2025	Diskusi dengan tim DA dalam perancangan dan pembangunan sistem pembuatan sistem klasifikasi sentimen konsumen
15-19 September 2025	Mempelejari strategi sentimen analisis, preparasi pembuatan model, dan pembuatan model
24-26 & 29-30 September 2025	Mencari cara meningkatkan model sentimen analisis, bisa dari *hypertuning*, mengganti strategi label data, tambah data.
3 Oktober 2025	Mencari cara meningkatkan model sentimen analisis, bisa dari *hypertuning*, mengganti strategi label data, tambah data.
23 Oktober 2025	Diskusi kembali dengan mentor strategi baru untuk meingkatkan kembali hasil dari model sentimen analisis
31 Oktober 2025	Diskusi dengan tim DA dan evaluasi strategi baru yang telah didiskusikan dengan mentor
3-5 November 2025	Modifikasi beberapa kode, seperti label data, *splitting* data, dan pemilihan data serta melatih kembali model sentimen analisis

3.3.1 Pengumpulan Data

Data dalam proyek ini diperoleh dari ulasan Google seluruh toko Gramedia yang berada di wilayah JABODETABEK. Berdasarkan diskusi bersama Tim DA, telah disepakati bahwa pembangunan pembuatan sistem klasifikasi sentimen konsumen toko Gramedia JABODETABEK dapat menggunakan data ulasan dari Google sebagai fondasi utama model. Ruang lingkup data dibatasi hanya pada JABODETABEK karena model ini dirancang untuk memahami karakteristik penulisan dan gaya linguistik pelanggan secara lebih spesifik pada wilayah tersebut, sehingga data dari luar wilayah ini dianggap tidak diperlukan. Sesuai arahan mentor, penambahan data dari luar JABODETABEK justru dikhawatirkan akan mengubah tujuan model, karena model dapat terdorong untuk mempelajari pola linguistik pelanggan Gramedia secara nasional dan bukan khusus untuk Gramedia JABODETABEK. Data yang didapatkan bertotal 60,832 baris. Struktur tabel

dataset serta penjelasannya telah disertakan pada Tabel 3.2.

Tabel 3.2. Penjelasan untuk column di *dataset* Sentiment Analisis Toko Gramedia Jalma

Column Name	Context
Location	Lokasi toko Gramedia yang berada dalam area JABODETABEK.
Author	Penulis ulasan google untuk toko Gramedia.
Rating	Nilai yang diberikan terhadap toko Gramedia.
Review	Tulisan ulasan yang diberikan oleh penulis.
reviewID	ID yang telah tercantum di ulasan google, jadi setiap ulasan memiliki reviewID yang unik.
languageMode	Review tersebut diterjamahkan bahasa Indonesia atau English, sesuai pilihan.

3.3.2 Tahap pra-pemrosesan data

Dataset yang diperoleh masih memerlukan proses *cleansing* dan pra-pemrosesan sebelum dapat digunakan pada tahap pelatihan, validasi, dan pengujian model. Kebutuhan ini muncul karena terdapat beberapa permasalahan pada data, seperti keberadaan emoji, adanya toko yang tidak termasuk toko Gramedia, serta ketiadaan label pada *dataset* awal sehingga proses pelabelan harus dilakukan secara manual. Permasalahan tersebut akan ditangani melalui kombinasi antara pra-pemrosesan manual menggunakan Microsoft Excel dan pra-pemrosesan otomatis menggunakan Python untuk memastikan kualitas data lebih konsisten sebelum masuk ke tahap pemodelan.

A Proses manual

Dalam pemeriksaan *dataset*, khususnya pada kolom 'location', ditemukan beberapa entri yang bukan merupakan toko Gramedia. Sebagai contoh, pada Gambar 3.2 terlihat adanya ulasan untuk "Holland Bakery–Gramedia Matraman" yang sebenarnya tidak terdaftar sebagai toko Gramedia. Selain itu, terdapat juga lokasi yang memang termasuk dalam grup Gramedia, namun tidak merepresentasikan keseluruhan toko dari Gramedia JABODETABEK. Sebagai ilustrasi, JanSport merupakan salah satu merek yang berada di bawah naungan Gramedia dan berfokus pada perlengkapan olahraga, namun tidak merepresentasikan keseluruhan toko Gramedia.

Daftar nama toko yang dieksklusi beserta tipe tokonya telah disertakan pada Tabel 3.3. Setelah dieliminasi toko-toko tersebut, data yang telah disimpan bertotal 59,190 baris.

Location	Author	Rating
Holland Bakery - Gramedia Matraman	Muhamme	5 stars
Holland Bakery - Gramedia Matraman	A s	4 stars
Holland Bakery - Gramedia Matraman	Bejo Wiyor	4 stars
Holland Bakery - Gramedia Matraman	Oki Pratan	5 stars
Holland Bakery - Gramedia Matraman	NAYAMUL	5 stars
Holland Bakery - Gramedia Matraman	Anindiatie	4 stars
Holland Bakery - Gramedia Matraman	cevy dwiba	5 stars
Holland Bakery - Gramedia Matraman	Seriyanilau	1 star
Holland Bakery - Gramedia Matraman	Emyl Idris	5 bintang
Holland Bakery - Gramedia Matraman	Moon Light	4 bintang
Holland Bakery - Gramedia Matraman	Indah Wije	2 bintang
Holland Bakery - Gramedia Matraman	milo cokel	5 bintang
Gramedia	Nia Putri A	5 stars
Gramedia	Ruth Amm	5 stars
Gramedia	RIZKI SANJ	5 stars
Gramedia	catur fajar	5 stars
Gramedia	ulil awaliya	5 stars
Gramedia	Suha Salsa	5 stars
Gramedia	Yudha Fad	4 stars

Gambar 3.2. Holland backery terdapat di *dataset* yang seharusnya tidak ada karena harusnya cuman toko Gramedia.

B Pra-pemrosesan dengan Python

Setelah melakukan proses eksklusi terhadap beberapa lokasi pada *dataset*, tahap pra-pemrosesan dilanjutkan dengan *cleansing* tambahan dan pelabelan menggunakan pendekatan *lexicon-based*. Sebelum pelabelan, dilakukan normalisasi teks, perbaikan Mojibake, penghapusan emoji, deduplikasi berdasarkan kolum ’reviewID’ dan isi teks yang telah dinormalisasi, serta penghapusan ulasan kosong atau tidak valid. Pendeteksian bahasa tidak menggunakan kolom ’LanguageMode’ bawaan *dataset*, melainkan dilakukan menggunakan *library* Lingua untuk memastikan setiap ulasan teridentifikasi sebagai bahasa Indonesia atau bahasa Inggris. Sistem pelabelan sentimen memanfaatkan kombinasi InSet lexicon *dataset* dan Python *dictionary* ’`INDO_CUSTOM_PHRASES`’ untuk bahasa Indonesia, serta gabungan antara Python *dictionary* ’`EN_CUSTOM_PHRASE`’ dan skor polaritas VADER untuk ulasan berbahasa Inggris. Kinerja dari *lexicon* bisa dicontohkan dengan berikut:

Tabel 3.3. Lokasi yang Dikecualikan dan Tipe Lokasi

Lokasi di Eksklusi	**Tipe Lokasi**
KOMPLEK KOMPAS 2 GRAMEDIA KOMPAS	Komplek Perumahan
MG Sports & Music - Gramedia Matraman	Peralatan olahraga dan musik
KOMPLEK KOMPAS NIRMALA GRAMEDIA KOMPAS	Komplek Perumahan
IMOO & Olike GRAMEDIA GRANDMETRO BEKASI	Barang Elektronik
MG Sports & Music Gramedia Bintaro	Peralatan olahraga dan musik
MG Sports & Music Gramedia Emerald Bintaro	Toko MG Sports & Music
TOKO OLAHRAGA & MUSIK MG GRAMEDIA METROPOLITAN MALL CILEUNGSI	Peralatan olahraga dan musik
Musholla Gramedia World BSD - SB Floor	Fasilitas Musholla
Gramedia Pustaka Utama (GPU)	Kantor Penerbit
Jansport Gramedia Karawaci	Peralatan olahraga
Holland Bakery - Gramedia Matraman	Toko Roti
TOKO OLAHRAGA & MUSIK MG GRAMEDIA MATRAMAN	Peralatan olahraga dan musik
SONIDO MUSICA INDONESIA (SMI GRAMEDIA GRAND INDONESIA)	Toko Peralatan Musik

1. Pertama, disiapkan kalimat yang ingin dilabel. Pada contoh ini, ada enam kalimat yang akan dilampirkan, yaitu:

   (a) Tempat ini rapi.

   (b) Tempat ini berantakan.

   (c) Tempat ini berantakan, tapi bukunya lengkap.

   (d) Tempat ini rapi, tapi bukunya dikit.

   (e) Tempat ini sebagian rapi sebagian berantakan.

   (f) Saya baca buku.

2. Kedua, informasi konteks sentimen pada setiap kata dengan skor yang diberikan di *dictionary* akan digunakan untuk perhitungan skor sentimen. Contoh *dictionary* konteks sentimen terlampirkan pada Kode 3.1. Jika kata tersebut memiliki konteks sentimen yang positif, maka skor yang akan diberikan adalah angka positif yang dimulai dari 1 sampai angka maximum

yang telah disepakati. Jika kata tersebut memiliki konteks sentimen yang negatif, maka skor yang akan diberikan adalah angka negatif yang dimulai dari -1 sampai angka maximum yang telah disepakati.

```
lexicon = {
    "rapi": 3,
    "lengkap": 5,
    "dikit": -5,
    "berantakan": -3
}
```

Kode 3.1: Contoh isi dari *dictionary* untuk *lexicon*

3. Ketiga, setiap kalimat yang sudah disiapkan akan dicarikan kata yang memiliki konteks sentimen atau terdapat di *dictionary* pada Kode 3.1 dan akan diakumulasikan. Perhitungan dan penjelasan rinci untuk setiap perhitungan sebagai berikut:

   (a) Kalimat 'Tempat ini rapi.'

   Terdapat pada kalimat 'Tempat ini rapi.' bahwa 'rapi' adalah kata yang memiliki konteks sentimen positif. Terdapat juga pada Kode 3.1 bahwa skor yang diberikan untuk kata 'rapi' adalah 3. Maka akumulasi sentimen skor untuk kalimat tersebut adalah 3.

   (b) Kalimat 'Tempat ini berantakan.'

   Terdapat pada kalimat 'Tempat ini berantakan.' bahwa 'berantakan' adalah kata yang memiliki konteks sentimen negatif. Terdapat juga pada Kode 3.1 bahwa skor yang diberikan untuk kata 'berantakan' adalah -3. Maka akumulasi sentimen skor untuk kalimat tersebut adalah -3.

   (c) Kalimat 'Tempat ini berantakan, tapi bukunya lengkap.'

   Pada kalimat 'Tempat ini berantakan, tapi bukunya lengkap.' memiliki kata 'berantakan' yang bersifat negatif dengan skor -3 dan kata 'lengkap' yang bersifat positif dengan skor 5 sesuai dengan Kode 3.1. Jika diakumulasikan pada kalimat tersebut, maka akan didapatkan skor sentimen bertotal 2.

   (d) Kalimat 'Tempat ini rapi, tapi bukunya dikit.'

   Pada kalimat 'Tempat ini rapi, tapi bukunya dikit.' memiliki kata 'dikit' yang bersifat negatif dengan skor -5 dan kata 'rapi' yang bersifat positif

dengan skor 3 sesuai dengan Kode 3.1. Jika diakumulasikan pada kalimat tersebut, maka akan didapatkan skor sentimen bertotal -2.

(e) Kalimat 'Tempat ini sebagian rapi sebagian berantakan.'

Pada kalimat 'Tempat ini sebagian rapi sebagian berantakan.' memiliki kata 'berantakan' yang bersifat negatif dengan skor -3 dan kata 'rapi' yang bersifat positif dengan skor 3 sesuai dengan Kode 3.1. Jika diakumulasikan pada kalimat tersebut, maka akan didapatkan skor sentimen bertotal 0.

(f) Kalimat 'Saya baca buku.'

Pada kalimat 'Saya baca buku.', tidak ada kata yang memiliki konteks sentimen. Maka total skor sentimen pada kalimat tersebut adalah 0.

4. Terakhir setiap kalimat akan dilabelkan sesuai dengan skor sentimen yang diakumulasikan dengan peraturan pelabelan sebagai berikut:

   (a) Jika skor sentimen diatas 0 atau skor sentimen > 0, maka kalimat akan dilabelkan sentimen positif.

   (b) Jika skor sentimen dibawah 0 atau skor sentimen < 0, maka kalimat akan dilabelkan sentimen negatif.

   (c) Jika skor sentimen sama dengan 0 atau skor sentimen = 0, maka kalimat akan dilabelkan sentimen netral.

   Maka untuk kalimat contoh sudah dibuatkan, maka hasil perlabelan dilampirkan pada Tabel 3.4.

Tabel 3.4. Hasil *lexicon* untuk kalimat contoh yang telah dibuat

Kalimat	**Skor Sentimen**	**Label**
Tempat ini rapi.	3	Positif
Tempat ini berantakan.	-3	Negatif
Tempat ini berantakan, tapi bukunya lengkap.	2	Positif
Tempat ini rapi, tapi bukunya dikit.	-2	Negatif
Tempat ini sebagian rapi sebagian berantakan.	0	Netral
Saya baca buku.	0	Netral

Setelah seluruh ulasan memperoleh label sentimen (positif, negatif, atau netral), *dataset* diseimbangkan sehingga setiap kelas memiliki jumlah sampel yang

sama, sekaligus menjaga proporsi data bahasa Indonesia dan bahasa Inggris tetap seimbang. Berikut adalah kode dengan penjelasan untuk setiap bagian:

1. *Library* yang di impor

   Library yang digunakan terlampirkan pada Kode 3.2 untuk *cleansing* dan pelabelan data dengan penjabaran *library* sebagai berikut:

```
import re
import pandas as pd
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from tqdm import tqdm
from lingua import Language, LanguageDetectorBuilder
import random
import json
```

Kode 3.2: Kode impor library yang dibutuhkan

   (a) `re`

   Library `re` digunakan untuk melakukan manipulasi dan pendeteksian pola pada sebuah *string*. *Library* ini sangat membantu pada tahap *cleansing*. Dalam kode, penggunaan `re` dapat ditemukan pada fungsi-fungsi seperti `re.sub` untuk menghapus tanda baca dan mengurangi spasi berlebih. Selain itu, `re.findall` dan `re.search` juga digunakan untuk mendeteksi token dan memastikan bahwa sebuah teks mengandung karakter. Semua fungsi ini berperan dalam proses normalisasi dan pembersihan data sebelum analisis lebih lanjut.

   (b) `pandas`

   Library ini digunakan untuk membaca dan manipulasi dataset. Pada kode program, penggunaan `pandas` dapat ditemukan pada beberapa operasi penting seperti `pd.read_csv` untuk membaca *dataset*, `df.dropna` untuk menghapus baris yang kosong, `df.sample` untuk mengacak data, `df.drop_duplicates` untuk menghilangkan duplikasi, `df.apply` untuk menerapkan fungsi pembersihan teks, serta `df.to_csv` untuk menyimpan hasil akhir ke file.

   (c) `nltk` dan `from nltk.sentiment.vader import SentimentIntensityAnalyzer`

Library `nltk` digunakan karena modul VADER yang dipakai dalam proyek ini berasal dari `nltk.sentiment.vader`. Pada kode, `nltk` dipanggil untuk mengunduh *lexicon* VADER melalui perintah `nltk.download("vader_lexicon")` dan kemudian digunakan untuk membuat objek `SentimentIntensityAnalyzer`. Dengan demikian, seluruh proses analisis sentimen bahasa Inggris di dalam pipeline bergantung pada *library* `nltk`.

(d) `tqdm`

Library `tqdm` digunakan untuk menampilkan *progress bar* ketika program menjalankan suatu proses. Pada kode ini, `tqdm` digunakan bersama `pandas` melalui `progress_apply` untuk menunjukkan perkembangan proses penilaian sentimen pada setiap ulasan. *Library* ini digunakan karena prosesnya memakan waktu cukup lama, sehingga *progress bar* membantu memastikan apakah lambatnya proses disebabkan oleh kesalahan pada kode atau memang karena prosesnya yang secara alami membutuhkan waktu.

(e) `lingua`

Library `lingua` digunakan untuk mendeteksi bahasa dari setiap teks ulasan. Pada Kode ini, `lingua` dikonfigurasi hanya dengan dua pilihan bahasa, yaitu *English* dan *Indonesian*, sehingga setiap ulasan akan diklasifikasikan ke salah satu dari kedua bahasa tersebut. Jika sebuah ulasan tidak dapat diklasifikasikan ke dalam salah satu dari dua bahasa tersebut, maka kategorinya akan ditetapkan sebagai *unknown*. *Library* ini membantu memastikan bahwa proses penilaian sentimen dilakukan dengan metode yang sesuai untuk masing-masing bahasa.

(f) `random`

Library `random` digunakan untuk menghasilkan nilai acak yang konsisten melalui pengaturan `seed`. Dalam Kode ini, `random.seed()` digunakan agar proses seperti pengacakan data (*shuffling*) atau pemilihan sampel berjalan akan memberikan hasil yang sama setiap kali menjalankan program, sehingga memastikan reprodusibilitas eksperimen dan validitas analisis.

(g) `json`

Library `json` digunakan untuk berinteraksi dengan berkas berformat JSON. Dalam konteks ini, *library* tersebut hanya dipakai untuk

membaca berkas `normalize_dict.json` yang akan digunakan pada proses normalisasi teks pada ulasan.

2. Konfigurasi awal

   Konfigurasi awal program telah terlampirkan pada Kode 3.3. Bagian ini menjadi landasan untuk penentuan nama variabel yang digunakan di seluruh program, termasuk pengaturan nilai `random.seed()` yang diperlukan untuk memastikan hasil proses dapat direplikasi. Variabel `INPUT_CSV` berfungsi sebagai penentu nama *dataset* masukan yang akan dibaca, yaitu "`gramedia_jabodetabek_reviews.csv`". Selain itu, terdapat variabel `OUT_CSV` yang menentukan nama keluaran berkas yang akan dihasilkan, yaitu "`dataset_balanced_random_lang_final_dedup_review.csv`". Variabel `POS_TSV` dan `NEG_TSV` masing-masing merujuk pada berkas yang berisi daftar kata beserta skor sentimen dari *lexicon* InSet dengan `POS_TSV` menyimpan kata-kata bernada positif dan `NEG_TSV` menyimpan kata-kata bernada negatif. Terakhir, variabel `RANDOM_SEED` digunakan untuk menetapkan nilai seed pada proses pengacakan sehingga seluruh eksperimen dapat dilakukan ulang dengan hasil yang konsisten. Dalam kasus ini, nilainya ditetapkan menjadi 42.

```
# ======================================================
# CONFIGURATION
# ======================================================
INPUT_CSV = "gramedia_jabodetabek_reviews.csv"
POS_TSV = "positive.tsv"
NEG_TSV = "negative.tsv"
OUT_CSV = "dataset_balanced_random_lang_final_dedup_review.
    csv"

RANDOM_SEED = 42
```

Kode 3.3: Kode konfigurasi awal penamaan dan pengaturan nilai `random.seed()`

3. Inisiasi pengaturan

   Inisiasi untuk mengunduh *file* serta pengaturan fungsi dari *library* yang telah diimpor terlampirkan pada Kode 3.4. `nltk.download("vader_lexicon", quiet=True)` digunakan untuk mengunduh kamus sentimen milik VADER tanpa menampilkan log karena adanya parameter `quiet=True`. Objek `SentimentIntensityAnalyzer()` kemudian diinisiasi sebagai `vader`

sehingga proses analisis sentimen dapat dilakukan secara langsung. Terakhir, `random.seed(RANDOM_SEED)` diatur menggunakan variabel `RANDOM_SEED` yang telah ditentukan pada Kode 3.3. `with open` digunakan untuk membuka berkas `"normalize_dict.json"` dalam mode baca (`"r"`) dengan pengaturan `encoding="utf-8"`. Setelah berkas berhasil dibuka, `json.load(f)` digunakan untuk memuat isi berkas tersebut ke dalam variabel `NORMALIZE_MAP`. Berkas `normalize_dict.json` berisi *dictionary* yang disusun secara manual berdasarkan pengecekan kata satu per satu dari *dataset*. Daftar ini digunakan untuk menormalkan variasi penulisan tidak baku seperti pengulangan huruf berlebih, misalnya atau "bagussssssssss" menjadi "bagus". Dengan pendekatan ini, seluruh kata tidak baku dapat dinormalisasi secara konsisten, tidak hanya sebagian yang terdeteksi oleh metode otomatis.

```
# ================================================
# INITIAL SETUP
# ================================================
nltk.download("vader_lexicon", quiet=True)
vader = SentimentIntensityAnalyzer()
random.seed(RANDOM_SEED)
with open("normalize_dict.json", "r", encoding="utf-8") as f:
    NORMALIZE_MAP = json.load(f)
```

Kode 3.4: Kode inisiasi pengaturan pra-pemrosesan dan *cleansing*

4. Inisiasi fungsi normalisasi teks

   Fungsi normalisasi pada Kode 3.5 dimulai dengan inisiasi melalui `def normalize_text(text)`. Pertama, `if pd.isna(text)` digunakan untuk mengecek apakah nilai yang diterima adalah *null*. Jika benar, fungsi langsung mengembalikan `""`. Setelah itu, `text = str(text).lower()` mengonversi seluruh karakter menjadi huruf kecil, diikuti `re.sub(r"[^\w\s]", "", text)` yang menghapus seluruh tanda baca. Spasi berlebih kemudian disederhanakan menggunakan `re.sub(r"\s+", " ", text)` dan `text.strip()` memastikan tidak ada spasi di awal maupun akhir kalimat. Sebagai contoh, teks seperti `"  Hello world  "` akan berubah menjadi `"hello world"` setelah tahap ini.

   Setelah pembersihan dasar selesai, teks dipecah menjadi daftar kata menggunakan `words = text.split()`. Daftar `NORMALIZE_MAP`,

yang telah diinisiasi pada Kode 3.4, berfungsi sebagai pemetaan kata tidak baku ke bentuk yang benar. Proses ini dilakukan melalui `[NORMALIZE_MAP.get(w, w) for w in words]` yang akan mengganti setiap kata yang ditemukan dalam kamus tersebut. Misalnya, kata seperti `"kereeeeeeen"` akan dinormalisasi menjadi `"keren"`. Terakhir, kata-kata yang telah dinormalisasi digabung kembali menggunakan `" ".join(words)` sehingga menghasilkan teks akhir yang bersih dan seragam.

```
def normalize_text(text):
    if pd.isna(text):
        return ""
    text = str(text).lower()
    text = re.sub(r"[^\w\s]", "", text)
    text = re.sub(r"\s+", " ", text)
    text = text.strip()

    words = text.split()
    words = [NORMALIZE_MAP.get(w, w) for w in words]
    return " ".join(words)
```

Kode 3.5: Kode inisiasi normalisasi teks

5. Inisiasi untuk perbaikan teks Mojibake.

   Perbaikan teks Mojibake terlampirkan pada Kode 3.6. Mojibake merupakan karakter-karakter yang tidak dapat dibaca (seperti teks “kerÃ«n”, “jelÃ«k”, atau “kurÃ¥ng” yang seharusnya terbaca sebagai “keren”, “jelek”, dan “kurang”) dan muncul akibat kesalahan proses *encoding* pada suatu berkas, baik itu CSV maupun Excel. Fungsi `fix_mojibake(text)` digunakan untuk memperbaiki masalah tersebut dengan menerima sebuah variabel `text` sebagai input. Baris `if not isinstance(text, str)` digunakan untuk memeriksa apakah nilai `text` merupakan sebuah *string*. Jika bukan, fungsi akan langsung melakukan `return text` tanpa perubahan. Selanjutnya, di dalam blok `try`, perintah `raw_bytes = text.encode("latin-1", errors="ignore")` dan `fixed = raw_bytes.decode("utf-8", errors="ignore")` digunakan untuk melakukan *encoding* teks ke Latin-1 dan kemudian *decoding* kembali ke UTF-8, karena hasil inspeksi *dataset* menunjukkan adanya teks yang awalnya terbaca sebagai Latin-1 tetapi sebenarnya dimaksudkan sebagai UTF-8. Jika proses perbaikan berhasil, fungsi akan mengembalikan nilai melalui `return fixed`, namun jika proses

encoding atau *decoding* mengalami kegagalan, blok `except Exception` akan menangani kondisi tersebut dan fungsi akan mengembalikan `text` dalam bentuk aslinya tanpa perbaikan.

```
def fix_mojibake(text):
    if not isinstance(text, str):
        return text
    try:
        raw_bytes = text.encode("latin-1", errors="ignore")
        fixed = raw_bytes.decode("utf-8", errors="ignore")
        return fixed
    except Exception:
        return text
```

Kode 3.6: Kode inisiasi perbaikan teks yang ada Mojibake

6. Inisiasi fungsi untuk eliminasi emoji

Fungsi untuk menghilangkan emoji terlampirkan pada Kode 3.7. Setelah melakukan diskusi dengan tim DA, disarankan untuk menghapus emoji dari ulasan karena fokus analisis adalah pada tanggapan konsumen secara lisan, sehingga keberadaan emoji tidak memberikan nilai tambahan. Fungsi `remove_emojis(text)` digunakan untuk menginisiasi proses tersebut dengan menerima variabel `text` sebagai input. `if not isinstance(text, str)` berfungsi untuk memeriksa apakah `text` merupakan sebuah *string*. Jika bukan, maka fungsi langsung melakukan `return text` tanpa perubahan. Selanjutnya, `emoji_pattern = re.compile(...)` digunakan untuk membentuk objek ekspresi reguler dari *library* `re` yang memungkinkan pencarian dan penghapusan emoji secara efisien. Rentang Unicode seperti `\U0001F600-\U0001F64F` merepresentasikan kelompok emoji tertentu, dan beberapa rentang lain pada kode digunakan untuk mencakup berbagai jenis emoji. Tanda `+` di dalam `re.compile` menunjukkan bahwa pola tersebut akan mencocokkan satu atau lebih karakter emoji secara berturut-turut. Setelah pola selesai dibangun, perintah `emoji_pattern.sub("", text)` akan menggantikan seluruh emoji yang ditemukan dengan *string* kosong, sehingga menghasilkan teks akhir tanpa emoji.

```
def remove_emojis(text):
    if not isinstance(text, str):
        return text
    emoji_pattern = re.compile(
```

```
        "["
        u"\U0001F600-\U0001F64F"
        u"\U0001F300-\U0001F5FF"
        u"\U0001F680-\U0001F6FF"
        u"\U0001F1E0-\U0001F1FF"
        u"\U00002500-\U00002BEF"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        "]+"
    )
    return emoji_pattern.sub("", text)
```

Kode 3.7: Kode inisiasi fungsi untuk eliminasi emoji

7. Inisialisasi fungsi untuk memeriksa kemunculan kata dalam teks

   Fungsi ini terlampirkan pada Kode 3.8. Pemeriksaan terhadap kemunculan kata diperlukan karena meskipun teks telah melalui proses pembersihan, beberapa elemen seperti angka saja atau Mojibake masih dapat lolos. Kode ini membantu memastikan bahwa karakter yang tidak diperlukan tidak dihapus secara keseluruhan. Fungsi `def has_words(text)` digunakan sebagai inisiasi dengan menerima variabel `text`. Baris `if not isinstance(text, str)` berfungsi untuk memeriksa apakah variabel yang diberikan merupakan sebuah *string*. Jika bukan, maka `return False` dijalankan untuk menyaring dan mengeliminasi data yang tidak bertipe *string*. Sementara itu, `return bool(re.search(r"[a-zA-Z]", text))` akan mengembalikan nilai `True` atau `False`. Nilai `True` diberikan jika ditemukan setidaknya satu karakter huruf Latin (a–z atau A–Z) dalam variabel `text` dan `False` jika tidak ditemukan.

```
def has_words(text):
    if not isinstance(text, str):
        return False
    return bool(re.search(r"[a-zA-Z]", text))
```

Kode 3.8: Kode memeriksa kemunculan kata dalam teks

8. Pembacaan *dataset* InSet

   Kode untuk pembacaan informasi sentimen dari InSet terlampirkan pada Kode 3.9. Fungsi `def read_inset(pos_path, neg_path)` digunakan sebagai inisiasi proses pembacaan data InSet dengan

mewajibkan dua variabel, yaitu `pos_path` dan `neg_path`. Baris `pos = pd.read_csv(pos_path, sep="\t", header=0)` dan `neg = pd.read_csv(neg_path, sep="\t", header=0)` memiliki fungsi yang sama, yakni membaca *dataset* sentimen positif dan negatif dengan kondisi `sep="\t"` menunjukkan bahwa pemisahan kolom menggunakan *tab* dan `header=0` menyatakan bahwa baris pertama merupakan nama kolom. Variabel `d = {}` digunakan untuk menginisiasi *dictionary* yang nantinya menyimpan pasangan kata dan nilai sentimennya. Dua perulangan, yaitu `for w, wt in zip(pos["word"], pos["weight"])` dan `for w, wt in zip(neg["word"], neg["weight"])`, digunakan untuk memasukkan kata (`word`) beserta nilai sentimen (`weight`) ke dalam *dictionary* dengan terlebih dahulu mengubah kata menjadi *string* berhuruf kecil dan mengonversi nilai sentimen menjadi tipe data `float`. Proses ini ditempatkan dalam blok `try` dan `except` sehingga apabila terjadi kesalahan saat pengisian, eksekusi akan dilewati dengan `continue`. Setelah seluruh data diproses, fungsi `read_inset` memberikan hasil melalui `return d`. Sebagai contoh penggunaannya, baris `INSET_LEXICON = read_inset(POS_TSV, NEG_TSV)` memanggil fungsi `read_inset` dengan memberikan `POS_TSV` dan `NEG_TSV` sebagai argumen, dan karena kedua *dataset* tersebut telah ditentukan sebelumnya pada Kode 3.3, maka tidak diperlukan lagi penjelasan tambahan mengenai lokasi atau nama berkas tersebut.

```
# ================================================
# LEXICONS & PHRASES
# ================================================
def read_inset(pos_path, neg_path):
    pos = pd.read_csv(pos_path, sep="\t", header=0)
    neg = pd.read_csv(neg_path, sep="\t", header=0)
    d = {}
    for w, wt in zip(pos["word"], pos["weight"]):
        try:
            d[str(w).lower()] = float(wt)
        except:
            continue
    for w, wt in zip(neg["word"], neg["weight"]):
        try:
            d[str(w).lower()] = float(wt)
        except:
            continue
    return d
```

```
19
20 INSET_LEXICON = read_inset(POS_TSV, NEG_TSV)
```

Kode 3.9: Kode membaca *dataset* INSET dan inisiasi dalam variabel `INSET_LEXICON`

9. Inisialisasi Python *dictionary* untuk kata dan nilai sentimen khusus

   Pembuatan Python *dictionary* yang memuat kata dan nilai sentimen khusus sesuai dengan toko Gramedia JABODETABEK terlampirkan pada Kode 3.10. Sesuai kebutuhan, dibuat Python *dictionary* dengan kata atau kalimat khusus yang telah dikumpulkan dari toko Gramedia JABODETABEK. Pemilihan kata dan penentuan nilai sentimen telah melalui diskusi dengan tim DA. Variabel `INDO_CUSTOM_PHRASES` dan `EN_CUSTOM_PHRASES` berfungsi sebagai Python *dictionary* untuk *lexicon* khusus. Sintaks `{k.lower(): float(v) for k, v in {}.items()}` digunakan untuk membangun *dictionary* ini dengan `k.lower()` memastikan semua kata menjadi huruf kecil dan `float(v)` mengonversi semua nilai sentimen menjadi tipe data `float`. Metode `.items()` memungkinkan iterasi dalam sebuah perulangan untuk memperoleh pasangan *key-value* dari *dictionary*, yaitu kata dan nilai sentimen.

```
1  INDO_CUSTOM_PHRASES = {k.lower(): float(v) for k, v in {
2      "pelayanan bagus": 3.0, "pelayanan baik": 3.0, "pelayanan
       cepat": 3.0,
3      "pelayanan ramah": 3.0, "pelayanan memuaskan": 3.0, "
      pelayanan sopan": 3.0,
4      "pelayanan profesional": 3.0, "pelayanan buruk": -3.0, "
      pelayanan jelek": -3.0,
5      "pelayanan lambat": -3.0, "layanan lambat": -3.0, "
      layanan kurang": -3.0,
6      "lama tunggu": -3.0, "lama menunggu": -3.0, "buku lengkap
      ": 3.0,
7      "buku banyak": 3.0, "buku sedikit": -3.0, "koleksi buku
      lengkap": 3.0,
8      "koleksi lengkap": 3.0, "koleksi kurang": -3.0,
9      "stok lengkap": 3.0, "stok habis": -3.0, "barang habis":
      -3.0,
10     "tidak lengkap": -3.0, "kurang lengkap": -3.0, "lengkap
      sekali": 3.0,
11     "sudah lengkap": 3.0, "toko buku lengkap": 3.0, "toko
      rapi": 3.0, "toko bersih": 3.0,
12     "tempat nyaman": 3.0, "nyaman": 3.0, "suka mampir": 3.0,
```

```
    "tempat luas": 3.0, "tempat sempit": -3.0, "parkir luas": 3.0,
    "tempat parkir luas": 3.0, "parkir susah": -3.0, "tempat panas": -3.0,
    "tempat berantakan": -3.0,
}.items()}

EN_CUSTOM_PHRASES = {k.lower(): float(v) for k, v in {
 "good service": 3.0, "nice service": 3.0, "fast service": 3.0,
 "friendly service": 3.0, "satisfying service": 3.0, "polite service": 3.0,
 "professional service": 3.0, "bad service": -3.0, "poor service": -3.0,
 "slow service": -3.0, "slow response": -3.0, "insufficient service": -3.0,
 "long wait": -3.0, "waiting too long": -3.0, "complete books": 3.0,
 "many books": 3.0, "few books": -3.0, "complete collection": 3.0,
 "complete stock": 3.0, "out of stock": -3.0, "no items": -3.0,
 "not complete": -3.0, "less complete": -3.0, "very complete": 3.0,
 "already complete": 3.0, "tidy store": 3.0, "clean store": 3.0,
 "comfortable place": 3.0, "comfortable": 3.0, "spacious place": 3.0,
 "cramped place": -3.0, "spacious parking": 3.0, "large parking area": 3.0,
 "hard to park": -3.0, "hot place": -3.0, "messy place": -3.0
}.items()}
```

Kode 3.10: Python *dictionary* kata dan nilai sentimen khusus

10. Pembuatan fungsi pemjumlahan nilai-nilai sentimen

    Fungsi untuk penjumlahan nilai-nilai sentimen terlampirkan pada Kode 3.11. Kode ini digunakan untuk menjumlahkan nilai sentimen dari kata-kata yang diperoleh dari Python *dictionary* khusus pada Kode 3.10. Fungsi `def phrase_score(text_lower, phrase_dict)` digunakan untuk menginisiasi proses penjumlahan tersebut dengan menerima variabel `text_lower` dan `phrase_dict`. Fungsi ini hanya menjalankan `sum(w`

`for k, w in phrase_dict.items() if k in text_lower)`, yang menjumlahkan semua nilai `w` dari kata `k` yang terdapat dalam `phrase_dict` apabila kata tersebut ada dalam `text_lower`.

```
def phrase_score(text_lower, phrase_dict):
    return sum(w for k, w in phrase_dict.items() if k in
    text_lower)
```

Kode 3.11: Kode penjumlahan nilai-nilai sentimen

11. Pembuatan fungsi deteksi bahasa dengan variabel pendukung

    Kode untuk deteksi bahasa beserta pemanggilan *library* yang dibutuhkan terlampirkan pada Kode 3.12. Fungsi ini akan digunakan dalam proses *sentiment scoring* untuk mendeteksi bahasa ulasan dan menentukan apakah ulasan tersebut berbahasa Indonesia atau Inggris. Variabel `detector` diinisialisasi menggunakan `LanguageDetectorBuilder`, yang secara khusus dikonfigurasi untuk mendukung bahasa Inggris dan Indonesia melalui `from_languages(Language.ENGLISH, Language.INDONESIAN).build()` dan diperoleh dari *library* `lingua` yang telah dimuat sebelumnya pada Kode 3.2. Fungsi `def detect_language(text)` memanggil deteksi bahasa dengan memberikan teks melalui variabel `text`. Di dalam fungsi tersebut, blok `try` dan `except` memastikan bahwa `return detector.detect_language_of(text)` akan mengembalikan bahasa yang terdeteksi, sedangkan `return None` akan dijalankan jika bahasa tidak dapat dikenali.

```
# ================================================
# LANGUAGE DETECTOR
# ================================================
detector = (LanguageDetectorBuilder
            .from_languages(Language.ENGLISH, Language.
    INDONESIAN)
            .build())

def detect_language(text):
    try:
        return detector.detect_language_of(text)
    except:
        return None
```

Kode 3.12: Kode Pembuatan fungsi deteksi bahasa dengan variabel pendukung

12. Pembuatan fungsi penilaian sentimen

Fungsi penilaian sentimen telah terlampirkan pada Kode 3.13 mengakhiri fungsi dengan `sentiment` dan `lang_label` sebagai hasilnya. Fungsi `def score_review(text)` menginisiasi proses dengan menerima variabel `text`. Baris `tokens = re.findall(r"\b\w+\b", text)` digunakan untuk memecah kalimat panjang menjadi token, misalnya kalimat "Aku suka buku" menjadi `["aku"], ["suka"], ["buku"]`. Selanjutnya, `lang = detect_language(text)` akan mendeteksi bahasa ulasan seperti yang dijelaskan pada Kode 3.12.

Jika ulasan berbahasa Indonesia (`if lang == Language.INDONESIAN`), perhitungan skor dilanjutkan dengan penambahan nilai pada variabel `score` melalui `phrase_score(text, INDO_CUSTOM_PHRASES)`, seperti yang terdapat pada Kode 3.11, ditambah dengan akumulasi nilai sentimen dari `INSET_LEXICON` melalui `sum(INSET_LEXICON.get(t, 0.0) for t in tokens)`. Variabel `sentiment` kemudian ditentukan sebagai "*Positive*" jika `score` lebih dari 0, "*Negative*" jika `score` kurang dari 0, atau "*Netral*" jika `score` sama dengan 0. Variabel `lang_label` diberikan nilai "*Indonesian*".

Jika ulasan berbahasa Inggris (`elif lang == Language.ENGLISH`), variabel `score`, `sentiment`, dan `lang_label` diproses serupa, kecuali perhitungan `score` menggunakan `phrase_score(text, EN_CUSTOM_PHRASES)` ditambah dengan `vader.polarity_scores(text)["compound"]`, yang menghitung nilai sentimen berdasarkan kata-kata dalam *dataset* VADER. Variabel `lang_label` diberikan nilai "*English*". Sementara `sentiment` mengikuti kondisi yang sama seperti sebelumnya.

Jika bahasa ulasan bukan Indonesia maupun Inggris, deteksi bahasa tidak dapat menentukan bahasa ulasan tersebut. Dalam kondisi ini, `sentiment` akan diberikan nilai "*Mixed*". Demikian juga, `lang_label` akan diatur menjadi "*Mixed*". Ini bisa terjadi jika fungsi deteksi bahasa tidak bisa mengetehui jika hasil tersebut adalah bahasa Ingrris atau Indonesia.

```
# ========================================
# SENTIMENT SCORING
# ========================================
def score_review(text):
```

```
    tokens = re.findall(r"\b\w+\b", text)
    lang = detect_language(text)
    if lang == Language.INDONESIAN:
        score = phrase_score(text, INDO_CUSTOM_PHRASES) + sum
    (INSET_LEXICON.get(t, 0.0) for t in tokens)}
        sentiment = "Positive" if score > 0 else "Negative"
    if score < 0 else "Neutral"
        lang_label = "Indonesian"
    elif lang == Language.ENGLISH:
        score = phrase_score(text, EN_CUSTOM_PHRASES) + vader
    .polarity_scores(text)["compound"]
        sentiment = "Positive" if score > 0 else "Negative"
    if score < 0 else "Neutral"
        lang_label = "English"
    else:
        sentiment = "Mixed"
        lang_label = "Mixed"
    return sentiment, lang_label
```

Kode 3.13: Kode pembuatan fungsi penilaian sentimen

13. *Pipeline* utama: membaca *dataset*

    Bagian awal dari kinerja program terlampirkan pada Kode 3.14. Sebelum menjalankan proses apa pun, diperlukan *input dataset* yang akan melalui tahap pra-pemrosesan. Baris `df = pd.read_csv(INPUT_CSV, dtype=str)` merupakan bagian yang membaca file CSV yang akan dipra-pemrosesan. Variabel `INPUT_CSV` telah diinisialisasi pada Kode 3.3 dengan tipe data yang dibaca sebagai `string`. Sementara itu, `print(f"Initial dataset: len(df) rows")` hanya berfungsi untuk menampilkan log jumlah baris data yang tersedia sebelum proses *cleansing* dilakukan.

```
# ================================================
# MAIN PIPELINE
# ================================================
df = pd.read_csv(INPUT_CSV, dtype=str)
print(f"Initial \texit{dataset}: {len(df)} rows")
```

Kode 3.14: Kode *Pipeline* utama: membaca *dataset*

14. *Pipeline* utama: *drop* ulasan kosong

    Eliminasi ulasan kosong terlampirkan pada Kode 3.15. Ulasan yang tidak

memiliki isi dapat mengganggu proses pelatihan model, sehingga sangat disarankan agar tidak terdapat ulasan kosong dalam *dataset*. Perintah `df.dropna(subset=["Review"])` digunakan untuk menghapus entri yang bernilai `NULL` atau kosong. Selain itu, `print(f"After dropping NaN reviews: {len(df)} rows")` digunakan untuk menampilkan log jumlah baris data yang tersisa setelah proses eliminasi ulasan kosong dilakukan.

```
# Step 1: Drop empty reviews
df = df.dropna(subset=["Review"])
print(f"After dropping NaN reviews: {len(df)} rows")
```

Kode 3.15: Kode *Pipeline* utama: *drop* ulasan kosong

15. *Pipeline* utama: teks *cleansing*

Teks *cleansing* telah terlampirkan pada Kode 3.16. Setelah eliminasi data yang memiliki *null*, maka akan dilakukan data *cleansing* pada kolum 'Review' dengan `apply(fix_mojibake)`, `apply(remove_emojis)` dan juga `apply(normalize_text)`. Fungsi `fix_mojibake` telah diinisiasi pada Kode 3.6, `remove_emojis` telah diinisiasi pada Kode 3.7, dan `normalize_text` telah diinisiasi pada Kode 3.5. Setelah itu dengan `df[df["Review"].apply(has_words)]` akan menyimpan data yang memiliki sebuah kata latin dan bukan berupa nomor atau tanda baca. Fungsi `has_words` telah diinisiasi pada Kode 3.20. Setelah selesai, maka akan di log sisa data setelah *cleansing*.

```
# Step 2: Clean text
df["Review"] = df["Review"].apply(fix_mojibake).apply(
    remove_emojis).apply(normalize_text)
df = df[df["Review"].apply(has_words)]
print(f" After cleaning & non-word removal: {len(df)} rows")
```

Kode 3.16: Kode *Pipeline* utama: text *cleansing*

16. *Pipeline* utama: *shuffle* dan deduplikasi

Proses pengacakan data untuk pemilihan sampel dan deduplikasi terlampirkan pada Kode 3.17. Pada kode tersebut, deduplikasi dilakukan berdasarkan kolom 'reviewID' pada *dataset*, karena beberapa data memiliki 'reviewID' yang sama namun berbeda bahasa. Misalnya, satu 'reviewID' dapat muncul dalam dua baris dengan bahasa yang

berbeda. Baris `df = df.sample(frac=1, random_state=RANDOM_SEED)` digunakan untuk mengacak seluruh dataset, dengan `frac=1` menyatakan bahwa seluruh baris diacak dan `RANDOM_SEED` telah diinisiasi sebelumnya pada Kode 3.4. Pengacakan dilakukan karena perintah `drop_duplicates(subset=["reviewID"], keep="first")` akan menyimpan kemunculan pertama dari setiap 'reviewID' akibat penggunaan opsi `keep="first"`. Terakhir, log dicetak melalui `f"After dedup by reviewID: {len(df)} rows (removed {before_dedup - len(df)})"` yang menunjukkan jumlah baris setelah deduplikasi serta jumlah baris yang dihapus.

```
# Step 3: Shuffle & dedup
df = df.sample(frac=1, random_state=RANDOM_SEED)
before_dedup = len(df)
df = df.drop_duplicates(subset=["reviewID"], keep="first")
print(f"After dedup by reviewID: {len(df)} rows (removed {
    before_dedup - len(df)})")
```

Kode 3.17: Kode *pipeline* utama: *shuffle* dan deduplikasi

17. *Pipeline* utama: deduplikasi secara isi ulasan

Deduplikasi berdasarkan isi ulasan terlampirkan pada Kode 3.18. Meskipun deduplikasi berdasarkan kolum 'reviewID' sudah dilakukan, beberapa data masih memiliki isi ulasan yang sama walaupun 'reviewID' berbeda. Variabel `before_text_dedup = len(df)` digunakan untuk menyimpan jumlah baris sebelum proses deduplikasi. Perintah `df.drop_duplicates(subset=["Review"], keep="first")` dijalankan untuk menghapus baris dengan isi ulasan yang identik berdasarkan kolom 'Review', dan setelah proses deduplikasi selesai, urutan baris dikembalikan menggunakan `reset_index(drop=True)`. Terakhir, log `f"After dedup by normalized text: {len(df)} rows (removed {before_text_dedup - len(df)})"` digunakan untuk menampilkan jumlah baris setelah deduplikasi serta jumlah baris yang dihapus.

```
# Step 4: Dedup by review text (cleaned text already in df["
    Review"])
before_text_dedup = len(df)
df = df.drop_duplicates(subset=["Review"], keep="first").
    reset_index(drop=True)
```

```
print(f"After dedup by normalized text: {len(df)} rows (
    removed {before_text_dedup - len(df)})")
```

Kode 3.18: Kode *pipeline* utama: *drop* ulasan kosong

18. *Pipeline* utama: penilaian sentimen ulasan

    Penilaian sentimen terhadap ulasan pada `dataset` pada Kode 3.19 dilakukan melalui beberapa tahap. Proses ini dimulai dengan menjalankan `tqdm.pandas(desc="Scoring reviews")` untuk memantau progres penilaian sentimen serta memperkirakan waktu yang dibutuhkan. Selanjutnya, perintah `df["Review"].progress_apply(score_review)` dijalankan dan disimpan dalam variabel `results`, yang memanggil fungsi pada Kode 3.13 dengan memberikan data pada kolom `Review`. Hasil keluaran berupa `Sentiment` dan `LangDetected` kemudian dimasukkan kembali ke dalam `dataframe` menggunakan `pd.DataFrame(results.tolist(), index=df.index)` agar tersusun sesuai urutan aslinya. Terakhir, log `print("Sentiment distribution (raw):")` dan `print(df["Sentiment"].value\_counts(), "\n")` digunakan untuk menampilkan distribusi sentimen yang diperoleh setelah seluruh proses penilaian selesai.

```
# Step 5: Sentiment scoring
tqdm.pandas(desc="Scoring reviews")
results = df["Review"].progress_apply(score_review)
df[["Sentiment", "LangDetected"]] = pd.DataFrame(results.
    tolist(), index=df.index)

print("Sentiment distribution (raw):")
print(df["Sentiment"].value_counts(), "\n")
```

Kode 3.19: Kode pipeline utama: penilaian sentimen ulasan

19. *Pipeline* utama: filter data berdasarkan bahasa

    Tahap filter bahasa hanya mempertahankan data berbahasa Inggris dan Indonesia, sebagaimana terlampirkan pada Kode 3.20. Setelah proses *sentiment scoring*, data kembali difilter sehingga hanya ulasan dengan bahasa Inggris atau Indonesia yang diambil, berdasarkan nilai pada kolom 'LangDetected' yang dihasilkan dari Kode 3.13. Kolom 'LangDetected' dapat berisi label *English*, *Indonesian*, atau *mixed*. Label *mixed* diberikan

ketika sistem tidak dapat mendeteksi satu bahasa yang dominan, sehingga ulasan tersebut tidak diberikan nilai sentimen. Dengan menggunakan perintah `["LangDetected"].isin(["English", "Indonesian"])`, seluruh ulasan dengan label *mixed* otomatis dikeluarkan dari *dataset*.

```
# Step 6: Filter valid languages only
df = df[df["LangDetected"].isin(["English", "Indonesian"])]
```

Kode 3.20: Kode *Pipeline* utama: filter data bahasa

20. *Pipeline* utama: penyeimbangan data berdasarkan kelas sentimen terkecil

    Penyeimbangan data berdasarkan kelas sentimen terlampirkan pada Kode 3.21. Jika diperlukan hasil pemodelan yang lebih baik dan akurasi prediksi yang seimbang antar kelas, maka distribusi jumlah data yang proporsional pada setiap kelas sentimen harus dilakukan. Proses ini diawali dengan inisialisasi `balanced = []` yang akan digunakan untuk menyimpan *dataframe* dari masing-masing kelas sentimen yang telah diproses.

    Selain berdasarkan kelas sentimen, penyeimbangan juga dilakukan secara terpisah berdasarkan bahasa. Hal ini dilakukan melalui perulangan `for lang in ["English", "Indonesian"]` yang memastikan setiap bahasa diproses secara mandiri sehingga jumlah data untuk setiap sentimen di dalam masing-masing bahasa menjadi seimbang. Variabel `df_lang` digunakan untuk mempermudah pemisahan *dataset*, karena melalui perintah `df[df["LangDetected"] == lang]` seluruh data yang memiliki nilai `lang` tertentu langsung dikelompokkan. Dengan demikian, setiap kelas sentimen akan diseimbangkan hanya terhadap data dari bahasa yang sama, tanpa bercampur dengan bahasa lainnya.

    Selanjutnya, `counts` menyimpan jumlah total baris untuk setiap kelas sentimen melalui perintah `df["Sentiment"].value_counts().to_dict()`, yang mengubah hasil perhitungan menjadi sebuah Python *dictionary*. Dari struktur tersebut, ditentukan kelas dengan jumlah data paling sedikit menggunakan `min(sentiment_counts.get(s, 0) for s in ["Positive", "Neutral", "Negative"])` dalam `min_count`. Nilai ini kemudian ditampilkan melalui log `(f"\nSmallest sentiment class has {min_count} samples...`

untuk memberikan informasi mengenai ukuran kelas terkecil yang akan dijadikan acuan.

Proses penyeimbangan dilakukan melalui perulangan `for sentiment in ["Positive", "Neutral", "Negative"],` yang memproses setiap kelas secara berurutan di dalam setiap bahasa. Pada tahap ini, `subset` tidak lagi diambil dari `df` secara keseluruhan, melainkan dari `df_lang`, yaitu *dataframe* yang telah dipisahkan berdasarkan bahasa. Dengan demikian, setiap kelas sentimen diproses hanya terhadap data yang berasal dari bahasa yang sama.

Setiap kelas dibuat *subset* data menggunakan `subset = df_lang[df_lang["Sentiment"] == sentiment]` yang berisi baris dengan sentimen tertentu dalam bahasa yang sedang diproses. Jumlah baris dalam *subset* tersebut dihitung melalui `available = len(subset)`. Jika jumlah data melebihi `min_count_lang`, yakni ukuran kelas sentimen terkecil dalam bahasa tersebut, maka dilakukan pengambilan sampel secara acak menggunakan `subset.sample()` dengan `n=min_count_lang` yang memastikan bahwa jumlah data yang diambil disesuaikan dengan kelas yang paling sedikit dalam bahasa tersebut dan proses pengacakan diatur menggunakan `random_state=RANDOM_SEED`, dengan `RANDOM_SEED` telah diinisiasi pada Kode 3.3, sehingga hasilnya konsisten dan dapat direplikasi. Jika jumlah data dalam *subset* sudah sama atau lebih kecil daripada `min_count_lang`, maka data digunakan apa adanya tanpa modifikasi. Hasil pemrosesan untuk setiap kelas kemudian ditambahkan ke dalam daftar `balanced_parts` melalui `balanced_parts.append(sampled)`.

Pada tahap akhir, seluruh *subset* yang telah diseimbangkan digabungkan menjadi satu *dataframe* baru menggunakan `pd.concat(balanced).reset_index(drop=True)`. Proses ini sekaligus melakukan penyelarasan kembali indeks sehingga membentuk *dataset* akhir yang telah seimbang berdasarkan kelas sentimen dan telah dipastikan seimbang pula untuk setiap bahasa.

```
# Step 7: Balance by smallest sentiment class

balanced_parts = []

for lang in ["English", "Indonesian"]:
    df_lang = df[df["LangDetected"] == lang]
```

```
7
8       # Count per sentiment inside this language
9       counts = df_lang["Sentiment"].value_counts().to_dict()
10      print(f"\n===== {lang} Class Counts =====")
11      print(counts)
12
13      # Pick the minimum count *within this language*
14      min_count_lang = min(counts.get(s, 0) for s in ["Positive
        ", "Neutral", "Negative"])
15
16      print(f"{lang}: min class size = {min_count_lang}")
17
18      # Sample equally per sentiment
19      for sentiment in ["Positive", "Neutral", "Negative"]:
20          subset = df_lang[df_lang["Sentiment"] == sentiment]
21
22          if len(subset) > min_count_lang:
23              sampled = subset.sample(n=min_count_lang,
        random_state=RANDOM_SEED)
24          else:
25              sampled = subset
26
27          balanced_parts.append(sampled)
28
29  df_balanced = pd.concat(balanced_parts).reset_index(drop=True
        )
```

Kode 3.21: Kode *pipeline* utama: penyeimbangan data

21. Keluaran dan ringkasan

    Log dan ringkasan untuk hasil dari program telah terlampirkan pada Kode 3.22. `df_balanced.to_csv(OUT_CSV, index=False)` akan disimpan dengan nama keluaran yang telah ditentukan melalui variabel `OUT_CSV` yang sudah diinisiasi pada Kode 3.3. Secara singkat, Kode akan menuliskan *dataset* yang sudah seimbang ke dalam sebuah berkas CSV, kemudian menampilkan ringkasan akhir yang mencakup lokasi penyimpanan berkas, jumlah total baris yang dihasilkan, serta distribusi jumlah data untuk setiap kelas sentimen dan bahasa melalui `df_balanced['Sentiment'].value_counts()` dan `df_balanced['LangDetected'].value_counts()`. Ringkasan ini memastikan bahwa proses penyeimbangan telah berjalan sesuai harapan dan

semua kelas memiliki jumlah sampel yang seragam.

Selain itu, bagian akhir program juga menampilkan ringkasan tambahan berupa *language-by-sentiment summary*. Ringkasan ini dihasilkan melalui perulangan bertingkat yang memproses setiap sentimen pada daftar `sentiments` dan memeriksa jumlah data untuk masing-masing bahasa pada daftar `languages`. Setiap sentimen, terlebih dahulu dibuat *subset* `df_s` yang hanya memuat baris dengan sentimen tertentu menggunakan `df_balanced` terlebih dahulu difilter berdasarkan sentimen dengan memilih baris yang memiliki nilai `"Sentiment"` sesuai variabel `sentiment`. Hasil penyaringan ini disimpan dalam `df_s`. Setelah itu, untuk setiap bahasa, jumlah baris dihitung menggunakan `len(df_s[df_s["LangDetected"] == lang])` untuk memperoleh jumlah data pada sentimen dan bahasa tertentu, yang kemudian ditampilkan dalam format yang mudah dibaca.

```
# ============================================
# OUTPUT & SUMMARY
# ============================================
df_balanced.to_csv(OUT_CSV, index=False)

print("\n DONE      Balanced dataset saved to:", OUT_CSV)
print(f" Final shape: {len(df_balanced)} rows")
print(" Sentiment counts:\n", df_balanced['Sentiment'].
    value_counts())
print(" Language counts:\n", df_balanced['LangDetected'].
    value_counts())


print("\n================ LANGUAGEBYSENTIMENT SUMMARY
    ================\n")

sentiments = ["Positive", "Neutral", "Negative"]
languages = ["English", "Indonesian"]

for sentiment in sentiments:
    print(f"Sentiment: {sentiment}")
    df_s = df_balanced[df_balanced["Sentiment"] == sentiment]

    for lang in languages:
        count = len(df_s[df_s["LangDetected"] == lang])
        print(f"   {lang:<12}: {count}")
    print()
```

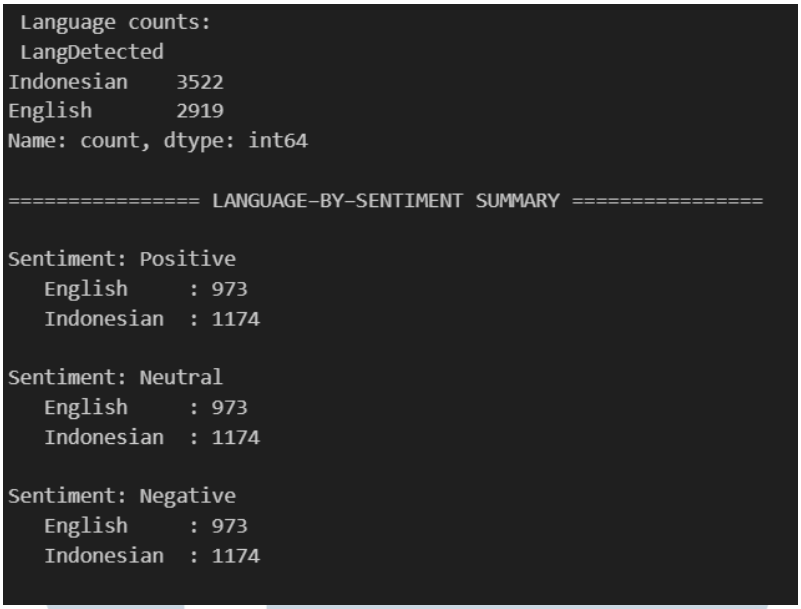
Kode 3.22: Kode Keluaran dan Ringkasan

Jika program *cleansing* dan pelabelan *dataset* dijalankan, keluaran *logging* yang dihasilkan dapat dilihat pada Gambar 3.3. Terlihat bahwa setiap kelas sentimen memiliki 2,147 baris data, yang disesuaikan dengan jumlah data pada kelas dan bahasa dengan jumlah paling sedikit agar seluruh kelas berada dalam kondisi seimbang. Sebagai contoh, pada bahasa Inggris, sentimen negatif merupakan kelas dengan jumlah data paling sedikit, yaitu 973 baris. Oleh karena itu, setiap kelas sentimen pada bahasa Inggris disamakan menjadi 973 baris data.

Selain itu, seperti dilampirkan pada Gambar 3.4, *dataset* terdiri atas 2,919 baris data berbahasa Inggris dan 3,522 baris data berbahasa Indonesia secara keseluruhan, dengan masing-masing kelas memiliki 973 data berbahasa Inggris dan 1,174 data berbahasa Indonesia. Jumlah data bahasa Inggris dan Indonesia perlu dijaga agar tetap *seimbang*, karena berdasarkan penelitian oleh Vincent Jung dan Lonneke van der Plas ditemukan bahwa ketidakseimbangan data antarbahasa dapat menurunkan kinerja model dibandingkan dengan kondisi ketika distribusi bahasa berada dalam keadaan seimbang [28].

```
Initial dataset: 59190 rows
 After dropping NaN reviews: 59190 rows
 After cleaning & non-word removal: 58857 rows
After dedup by reviewID: 35463 rows (removed 23394)
After dedup by normalized text: 28453 rows (removed 7010)
 Scoring reviews: 100%|██████████| 28453/28453 [00:04<00:00, 6078.30it/s]
 Sentiment distribution (raw):
Sentiment
Negative    12069
Positive    11595
Neutral      4789
Name: count, dtype: int64


===== English Class Counts =====
{'Positive': 8547, 'Neutral': 3615, 'Negative': 973}
English: min class size = 973

===== Indonesian Class Counts =====
{'Negative': 11096, 'Positive': 3048, 'Neutral': 1174}
Indonesian: min class size = 1174

 DONE — Balanced dataset saved to: dataset_balanced_random_lang_final_dedup_review.csv
 Final shape: 6441 rows
 Sentiment counts:
 Sentiment
Positive    2147
Neutral     2147
Negative    2147
Name: count, dtype: int64
```

Gambar 3.3. Keluaran setelah menjalankan program cleansing dan perlabelan dataset



Gambar 3.4. Keluaran untuk distribusi bahasa dan kelas sentiment dalam *dataset*

Setelah memperoleh *dataset* yang telah dilabel, proses dapat dilanjutkan dengan melakukan *splitting dataset* dan menyimpannya ke dalam satu folder. *Splitting dataset* berarti membagi *dataset* yang tersedia menjadi tiga bagian, yaitu latihan, validasi, dan uji. Pembagian data akan dilakukan sekali saja dengan mengikuti teknik validasi *holdout method* [29] sebab selama rangkaian magang, waktu lebih difokuskan pada perbaikan dan perancangan strategi pra-pemrosesan, pelatihan, validasi, serta pengujian model, sehingga efisiensi pembagian data menjadi aspek yang krusial. Selain itu, data yang ada perlu melalui proses tokenisasi menggunakan model mBERT. Berikut adalah kode untuk melakukan *setup* tersebut:

1. *Library* di impor

   Library yang perlu impor untuk menjalankan *splitting* dataset dan tokenisasi terlampirkan pada Kode 3.23 dengan penjabaran *library* sebagai berikut:

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from datasets import Dataset
from transformers import AutoTokenizer
```

Kode 3.23: Kode *library* di impor untuk membagi dan *tokenize*

(a) `os`

Library `os` merupakan modul yang menyediakan berbagai fungsi untuk berinteraksi dengan sistem operasi. Pada konteks ini, *library* tersebut hanya akan digunakan pada fungsi `os.makedirs()` untuk membuat sebuah folder secara lokal dan `os.path.join` sebagai fungsi untuk simpan sebuah file atau folder dalam direktori yang sudah dibuat.

(b) `pandas`

Library ini digunakan untuk membaca dan memanipulasi *dataset*. Pada kode program, penggunaan `pandas` dapat ditemukan pada beberapa operasi penting seperti `pd.read_csv` untuk membaca *dataset*, `df.sample` untuk mengacak data atau mengambil sebagian data dari keseluruhan *dataset*, dan `.value.count()` untuk melihat jumlah data pada sebuah variabel.

(c) `sklearn.model_selection import train_test_split`

Library `train_test_split` yang terdapat pada `sklearn` diperlukan untuk membagikan *dataset* menjadi beberapa bagian. Khusus pada program ini, fungsi `train_test_split()` akan digunakan untuk membagi *dataset* menjadi tiga bagian, yaitu data latih, data validasi, dan data uji.

(d) `datasets import Dataset`

Library `Dataset` digunakan untuk mengubah data hasil olahan `pandas` menjadi format yang siap digunakan oleh model. `Dataset.from_pandas` memiliki fungsi untuk mengonversi DataFrame menjadi *dataset* HuggingFace, `dataset.map` untuk melakukan tokenisasi secara *batch*, `dataset.remove_columns` untuk menghapus kolom yang tidak dibutuhkan model, dan `dataset.set_format("torch")` untuk mengubah data menjadi tensor sehingga dapat diproses oleh PyTorch saat training.

(e) `transformers import AutoTokenizer`

Library `transformers` melalui `AutoTokenizer` digunakan untuk melakukan tokenisasi pada data teks. `AutoTokenizer.from_pretrained` digunakan untuk memuat tokenizer bawaan model, sedangkan fungsi tokenisasi yang dipetakan dengan `dataset.map` berfungsi untuk mengubah kolom 'Review' menjadi token yang siap diproses oleh model.

(f) `random`

Library `random` digunakan untuk menghasilkan nilai acak yang konsisten melalui pengaturan `seed`. Dalam Kode ini, `random.seed()` digunakan agar proses seperti pengacakan data (`random_state()`) atau pemilihan sampel berjalan akan memberikan hasil yang sama setiap kali menjalankan program, sehingga eksperimen bisa diulang dengan hasil yang konsisten dan analisis tetap terpercaya.

2. Konfigurasi awal

Konfigurasi awal terlampirkan pada Kode 3.24. Variabel `GRAMEDIA_FILE` digunakan untuk menyimpan lokasi data yang telah dilabelkan. Variabel `SAVE_DIR` menyimpan lokasi folder atau direktori yang akan digunakan sebagai tempat penyimpanan model, log, serta *dataset* yang telah dibagi menjadi tiga bagian. Perintah `os.makedirs(SAVE_DIR, exist_ok=True)` digunakan untuk membuat direktori sesuai nilai `SAVE_DIR`, dengan `exist_ok=True` yang memastikan bahwa direktori tidak akan dibuat ulang jika sudah ada sebelumnya. Variabel `VAL_FRAC` menentukan persentase dari keseluruhan data yang dialokasikan sebagai *dataset* validasi, sedangkan `TEST_FRAC` menentukan persentase data yang digunakan sebagai data uji. Terakhir, `RANDOM_STATE` digunakan untuk memastikan bahwa setiap proses yang melibatkan pengacakan dapat direproduksi kembali menggunakan nilai acak yang sama.

```
# ========================
# Config
# ========================

GRAMEDIA_FILE = "making dataset/
    dataset_balanced_random_lang_final_dedup_review.csv"
SAVE_DIR = "./
    mbert_sentiment_gramedia_only_balanced_final_dedup_review"
os.makedirs(SAVE_DIR, exist_ok=True)

VAL_FRAC = 0.15
TEST_FRAC = 0.15
RANDOM_STATE = 42
```

Kode 3.24: Kode Konfigurasi awal untuk membagi dan dan *tokenize*

3. Membaca dan persiapan *dataset*

Persiapan dan membaca *dataset* terlampir pada Kode 3.25. `pd.read_csv(GRAMEDIA_FILE)` akan membaca *dataset* dalam program dengan `GRAMEDIA_FILE` telah diinisiasi pada Kode 3.24. `df["Sentiment_Lang"]` akan diinisiasi dengan `df["Sentiment"] + "_" + df["LangDetected"]` buat kolum baru yang bernama 'Sentiment_Lang' yang gabung kolum 'Sentiment' dan "LangDetected". Ini akan berguna pada tahap membagikan *dataset*. Setelah itu akan di log berapa baris data yang telah dibaca dan distribusi data untuk setiap kelas sentiment.

```
# ========================
# 1. Load Dataset
# ========================
df = pd.read_csv(GRAMEDIA_FILE)
df["Sentiment_Lang"] = df["Sentiment"] + "_" + df["
    LangDetected"]
print(f" Loaded dataset: {len(df)} rows")
print("Class distribution:\n", df["Sentiment"].value_counts()
    , "\n")
```

Kode 3.25: Kode Membaca dan persiapan *dataset*

4. Membagi *dataset*

Pembagian atau *split dataset* ditunjukkan pada Kode 3.26. Langkah pertama adalah membagi data awal menggunakan `train_test_split` berdasarkan nilai `VAL_FRAC` untuk menghasilkan `val_df` dan `rest_df`. Pada tahap ini, `stratify=df["Sentiment_Lang"]` digunakan agar proporsi kelas tetap konsisten berdasarkan kombinasi sentimen dan bahasa. Nilai `VAL_FRAC` telah diinisiasi sebelumnya pada Kode 3.24. Selanjutnya, pembagian kedua dilakukan untuk memperoleh `train_df` dan `test_df` dari `rest_df`. Nilai `test_size_relative` dihitung agar ukuran `test_df` sesuai dengan proporsi `TEST_FRAC` terhadap keseluruhan data awal. Variabel `TEST_FRAC` juga diinisiasi pada Kode 3.24. Pada pembagian kedua ini, proses *stratified split* kembali diterapkan melalui `stratify=rest_df["Sentiment_Lang"]` untuk menjaga distribusi kelas tetap seragam. Baik pada pembagian pertama maupun kedua, parameter `random_state` digunakan agar proses pembagian data bersifat deterministik dan dapat direplikasi. Nilai `RANDOM_STATE` telah ditentukan pada Kode 3.24. Setelah seluruh proses selesai, program

menampilkan jumlah data pada setiap *subset* serta distribusi kelas pada *dataset* latihan sebagai bagian dari pengecekan awal.

```
# ========================
# 2. Stratified split -> train, val, test
# ========================
rest_df, val_df = train_test_split(
    df,
    test_size=VAL_FRAC,
    random_state=RANDOM_STATE,
    stratify=df["Sentiment_Lang"],
)
test_size_relative = TEST_FRAC / (1 - VAL_FRAC)
train_df, test_df = train_test_split(
    rest_df,
    test_size=test_size_relative,
    random_state=RANDOM_STATE,
    stratify=rest_df["Sentiment_Lang"],
)

print(" Dataset sizes -> train:", len(train_df), "val:", len(
    val_df), "test:", len(test_df))
print("Train class distribution:\n", train_df["Sentiment"].
    value_counts(), "\n")
```

Kode 3.26: Kode *split dataset*

5. Simpan data latih, data validasi, dan data uji

   Melakukan penyimpanan *dataset* terlampirkan pada Kode 3.27. Penyimpanan *dataset* `train_df`, `val_df`, dan `test_df` dilakukan menggunakan fungsi yang sama, yaitu `to_csv`. Perintah `os.path.join` digunakan untuk menentukan lokasi penyimpanan berdasarkan `SAVE_DIR` yang telah diinisiasi pada Kode 3.24. Setiap *dataset* disimpan dengan nama berkas yang sesuai tanpa menyertakan indeks karena penggunaan `index=False`.

```
# ========================
# 3. Save CSVs for reproducibility
# ========================
train_df.to_csv(os.path.join(SAVE_DIR, "train.csv"), index=
    False)
val_df.to_csv(os.path.join(SAVE_DIR, "val.csv"), index=False)
```

```
test_df.to_csv(os.path.join(SAVE_DIR, "test.csv"), index=
    False)
```

Kode 3.27: Kode Simpan *dataset* latihan, validasi, dan uji

6. Tokenisasi data dan preparasi untuk model

   Proses tokenisasi dan persiapan data untuk model mBERT terlampirkan pada Kode 3.28. Pertama, dibuat sebuah `label2id` untuk mengubah label teks menjadi nilai numerik, kemudian kolom 'label' ditambahkan pada setiap bagian *dataset*. Selanjutnya, `Dataset.from_pandas` digunakan untuk mengonversi *DataFrame* menjadi format *dataset* yang kompatibel dengan HuggingFace. *Tokenizer* mBERT dimuat menggunakan `AutoTokenizer.from_pretrained`. Fungsi `tokenize` kemudian digunakan untuk mengubah kolom `Review` menjadi token dengan pemotongan (*truncation*) dan *padding* otomatis. Proses tokenisasi diterapkan melalui `dataset.map` pada data latih, validasi, dan uji. Setelah itu, kolom yang tidak lagi dibutuhkan seperti `Review` dan `Sentiment` dihapus menggunakan `remove_columns`. Terakhir, setiap *dataset* diatur menjadi format tensor melalui `set_format("torch")` agar dapat diproses oleh model mBERT saat pelatihan.

```

# ========================
# 4. Prepare for BERT fine-tuning
# ========================
label2id = {"Positive": 0, "Negative": 1, "Neutral": 2}
for df_ in [train_df, val_df, test_df]:
    df_["label"] = df_["Sentiment"].map(label2id)

train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
test_dataset = Dataset.from_pandas(test_df)

tokenizer = Autotokenizer.from_pretrained("bert-base-
    multilingual-cased")

def tokenize(batch):
    return tokenizer(batch["Review"], truncation=True,
    padding="max_length", max_length=128)

train_dataset = train_dataset.map(tokenize, batched=True)
```

```
val_dataset = val_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.map(tokenize, batched=True)

remove_cols = ["Review", "Sentiment"]
train_dataset = train_dataset.remove_columns([c for c in
    remove_cols if c in train_dataset.column_names])
val_dataset = val_dataset.remove_columns([c for c in
    remove_cols if c in val_dataset.column_names])
test_dataset = test_dataset.remove_columns([c for c in
    remove_cols if c in test_dataset.column_names])

train_dataset.set_format("torch")
val_dataset.set_format("torch")
test_dataset.set_format("torch")
```

Kode 3.28: Kode tokenisasi data dan preparasi untuk model

Setelah menjalankan ini semua, maka bisa terlihat hasil pembagian *dataset* pada Gambar 3.5. Total data pelatihan mencapai 4,507, data validasi mencapai 967 baris data, dan data pengujian mencapai total data 967. Data pengujian untuk kelas netral dan positif adalah 1,503 baris data dan untuk data negatif mencapai 1,502 baris data.

```
✅ Loaded dataset: 6441 rows
Class distribution:
 Sentiment
Positive    2147
Neutral     2147
Negative    2147
Name: count, dtype: int64

📊 Dataset sizes -> train: 4507 val: 967 test: 967
Train class distribution:
 Sentiment
Positive    1503
Negative    1502
Neutral     1502
Name: count, dtype: int64
```

Gambar 3.5. Hasil membagikan *dataset* ke data pelatihan, validasi, dan pengujian

3.3.3 Tahap Pelatihan dan Validasi Model

Setelah data sudah di pra-pemrosesan, maka pelatihan model bisa dilaksanakan. Sebelum melatih model, akan diperlukan dipilihkan *hyperparameter*, yaitu parameter yang bisa diotak-atik sesuai dengan keperluan, seperti meningkatkan model atau mempercepat proses pelatihan, yang akan diberikan terhadap kode *trainer*. *Trainer* akan nanti bantu melakukan pelatihan model dengan data latih dan akan berikan log dan hasil terhadap model yang telah melakukan prediksi ke data validasi. Hasil tersebut bisa digunakan untuk evaluasi jika model siap digunakan untuk tahap selanjutnya. Berikut adalah penjabaran kode tahap pelatihan dan validasi model:

1. *Library* yang diimpor

   Library yang digunakan terlampirkan pada Kode 3.29 untuk pelatihan model dengan penjabaran *library* sebagai berikut:

```
1 import os
2 from transformers import AutoModelForSequenceClassification,
    TrainingArguments, Trainer
3 from sklearn.metrics import accuracy_score,
    precision_recall_fscore_support
```

   Kode 3.29: Kode *library* yang diimpor untuk pelatihan dan validasi model

   (a) `os`

   Library `os` merupakan modul yang menyediakan berbagai fungsi untuk berinteraksi dengan sistem operasi. Pada konteks ini, *library* tersebut hanya akan digunakan pada fungsi `os.path.join` sebagai fungsi untuk simpan sebuah file atau folder dalam direktori yang sudah dibuat.

   (b) `transformers import TrainingArguments, Trainer`

   Library yang diimpor dari `transformers` mencakup beberapa komponen penting. Pertama, `AutoModelForSequenceClassification` berfungsi untuk memanggil dan mencantumkan model speisifik kedalam program. Selanjutnya, `TrainingArguments` berfungsi untuk menginisialisasi berbagai *hyperparameter* yang diperlukan selama proses pelatihan. Terakhir, `Trainer` membantu menjalankan proses pelatihan dan validasi

model dengan memanfaatkan parameter serta komponen yang telah disediakan.

(c) `sklearn.metrics import accuracy_score, precision_recall_fscore_support`

Fungsi-fungsi yang diimpor dari `sklearn.metrics` ini digunakan untuk membantu proses evaluasi model setelah pelatihan selesai dilakukan. `accuracy_score` digunakan untuk menghitung tingkat akurasi prediksi model, sedangkan `precision_recall_fscore_support` digunakan untuk memperoleh nilai presisi, *recall*, dan F1-score sebagai metrik evaluasi.

2. Model yang diimpor

Model yang akan diimpor untuk pelatihan terlampir pada Kode 3.30. Pemilihan model telah dilakukan dengan fungsi `AutoModelForSequenceClassification` dengan `from_pretrained()` yang menunjukkan bahwa model yang telah dipilih sudah melalui tahap *pre-training*. Model `"bert-base-multilingual-cased"` diberikan parameter `num_labels=3`, sesuai dengan tiga label pada proyek ini, yaitu positif, negatif, dan netral. Parameter `id2label` diperlukan tahap evaluasi untuk mengubah ID, yaitu 0,1 atau 2, menjadi label aslinya (contoh ID 1 merepresentasikan kategori positif dalam model).`{v: k for k, v in label2id.items()}` adalah kode pembuatan sendiri dengan `k` adalah nama label dan `v` adalah angka label dengan fungsi `v in label2id.items()` untuk memberikan konsistensi pada memberikan relasi `v` terhadap `k`. `label2id` akan digunakan untuk mengubah sebuah label menjadi ID yang telah diberikan (contoh label positif akan direpresentasikan ID 1) supaya model perlabelan yang ada.

```
model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-multilingual-cased",
    num_labels=3,
    id2label={v: k for k, v in label2id.items()},
    label2id=label2id
)
```

Kode 3.30: Kode model yang diimpor untuk pelatihan model

3. Menginisiasi argumen pelatihan model

Inisiasi argumen pelatihan model telah terlampirkan pada Kode 3.31. `training_args` akan menyimpan informasi argumen pelatihan dan *hyperparameter* untuk nanti diberikan kepada pelatihan model. `TrainingArguments()` adalah fungsi yang akan bantu menginisiasi *hyperparameter* dan argumen pelatihan yang diperlukan dengan penjabaran sebagai berikut:

(a) `output_dir=SAVE_DIR`

`output_dir` adalah salah satu argumen untuk menyatakan direktori atau folder untuk menyimpan model yang sudah dilatih dengan tuntas. `SAVE_DIR` sudah diiniasi pada Kode 3.24.

(b) `eval_strategy="epoch"`

`eval_strategy` adalah argumen yang menentukan apakah proses evaluasi akan dilakukan, serta kapan waktu evaluasinya. Argumen ini memiliki tiga pilihan, yaitu `no` yang berarti evaluasi tidak dijalankan sama sekali, `steps` yaitu evaluasi dilakukan setiap sejumlah langkah pelatihan tertentu dan tidak menunggu satu *epoch* penuh, serta `epoch` yang berarti evaluasi dilakukan setiap kali satu *epoch* selesai. Pada kasus ini, argumen yang digunakan adalah `epoch`.

(c) `save_strategy="epoch"`

`save_strategy` adalah argumen yang menentukan apakah ingin melakukan penyimpanan *checkpoint*, serta kapan *checkpoint* tersebut dibuat. Argumen ini memiliki empat pilihan, yaitu `no` yang berarti tidak menyimpan sama sekali, `steps` yaitu menyimpan setiap sejumlah langkah pelatihan tertentu, `epoch` yaitu menyimpan setiap kali satu *epoch* selesai, dan `best` yaitu hanya menyimpan ketika model mencapai hasil terbaik pada evaluasi. Pada program ini menggunakan `epoch` untuk `save_strategy`.

(d) `learning_rate=5e-5`

`learning_rate=5e-5` adalah argumen dan *hyperparameter* untuk inisiasi `learning_rate` yang digunakan pada *optimzer* AdamW, dengan *optimzer* digunakan untuk mengatur besarnya pembaruan parameter model [30]. Variabel yang diinisiasi adalah 5e-5.

(e) `per_device_train_batch_size=32` dan `per_device_eval_batch_size=32`

Argumen ini menentukan jumlah data yang diproses dalam satu kelompok oleh sebuah perangkat, seperti CPU atau GPU. Nilai `per_device_train_batch_size` digunakan pada tahap pelatihan dan nilai `per_device_eval_batch_size` digunakan pada tahap validasi. Dengan perangkat 1 GPU saja, maka 32 baris data yang akan dikirim pada tahap pelatihan dan validasi.

(f) `num_train_epochs=5`

`num_train_epochs` merupakan argumen yang menentukan berapa banyak *epoch* yang akan dijalankan selama pelatihan. Satu *epoch* berarti model telah melihat seluruh data pelatihan sekali. Dengan nilai 5, proses training akan dilakukan sebanyak lima kali putaran penuh terhadap seluruh dataset.

(g) `weight_decay=0.05`

`weight_decay` adalah regularisasi mencegah kasus *overfitting* dengan memberikan penalti kecil pada bobot parameter *weight* pada setiap lapisan, selain lapisan normalisasi, yang terlalu besar [31]. Nilai 0.05 akan diberikan pada kode ini.

(h) `logging_dir=os.path.join(SAVE_DIR, "logs")`

`logging_dir` menentukan direktori tempat file log pelatihan disimpan. Log ini berisi informasi seperti *loss*, *learning rate*, dan metrik lain yang dicatat selama proses training. `SAVE_DIR` telah diinisiasi pada Kode 3.24.

(i) `logging_steps=50`

`logging_steps` menentukan seberapa sering proses training mencatat informasi ke dalam log. Dengan nilai 50, Trainer akan melakukan log setiap 50 langkah pelatihan.

(j) `load_best_model_at_end=True`

`load_best_model_at_end` menginstruksikan Trainer untuk memuat model terbaik berdasarkan metrik evaluasi pada akhir pelatihan. Ini memastikan model akhir yang digunakan adalah model dengan performa evaluasi terbaik.

```
training_args = TrainingArguments(
    output_dir=SAVE_DIR,
    eval_strategy="epoch",
    save_strategy="epoch",
```

```
    learning_rate=5e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    num_train_epochs=5,
    weight_decay=0.05,
    logging_dir=os.path.join(SAVE_DIR, "logs"),
    logging_steps=50,
    load_best_model_at_end=True
)
```

Kode 3.31: Kode menginisiasi argumen pelatihan model

4. Inisiasi komputasi metrik

   Inisiasi komputasi metrik telah terlampirkan pada Kode 3.32. Fungsi `compute_metrics` digunakan oleh `Trainer` untuk menghitung nilai metrik setiap kali proses evaluasi dijalankan. Parameter `eval_pred` merupakan nilai yang akan diberikan `Trainer`, yaitu `logits` dan `labels`. `logits` adalah keluaran mentah prediksi dari model dan `labels` adalah label benar dari data validasi. `preds` digunakan untuk memperoleh prediksi aktual dari model dengan mengambil indeks kelas yang memiliki nilai probabilitas tertinggi menggunakan fungsi `np.argmax` pada `logits`. Setelah itu, beberapa metrik dihitung, yaitu akurasi dengan `accuracy_score` serta presisi, *recall*, dan F1-score menggunakan `precision_recall_fscore_support` dengan rata-rata `"macro"` untuk memberikan bobot yang sama pada setiap kelas. Fungsi kemudian mengembalikan sebuah *dictionary* yang berisi seluruh metrik tersebut agar dapat dicatat dan ditampilkan selama proses evaluasi berlangsung.

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    acc = accuracy_score(labels, preds)
    p, r, f1, _ = precision_recall_fscore_support(labels,
    preds, average="macro")
    return {"accuracy": acc, "precision_macro": p, "
    recall_macro": r, "f1_macro": f1}
```

Kode 3.32: Kode inisiasi komputasi metrik

5. Inisiasi trainer

   Kode 3.33 menunjukkan proses inisiasi `Trainer` yang digunakan untuk

mengelola seluruh tahapan pelatihan dan evaluasi model. Pada inisiasi ini, beberapa komponen penting diberikan kepada `Trainer`, yaitu `model` sebagai arsitektur yang akan dilatih, `training_args` yang berisi seluruh konfigurasi pelatihan yang telah diinisiasikan pada Kode 3.31 serta `train_dataset` dan `eval_dataset` sebagai data pelatihan dan evaluasi. Selain itu, fungsi `compute_metrics` juga dilampirkan agar `Trainer` dapat menghitung metrik evaluasi secara otomatis pada setiap proses evaluasi.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)
```

Kode 3.33: Kode Inisiasi Trainer

6. Menjalankan trainer dan menyimpan model

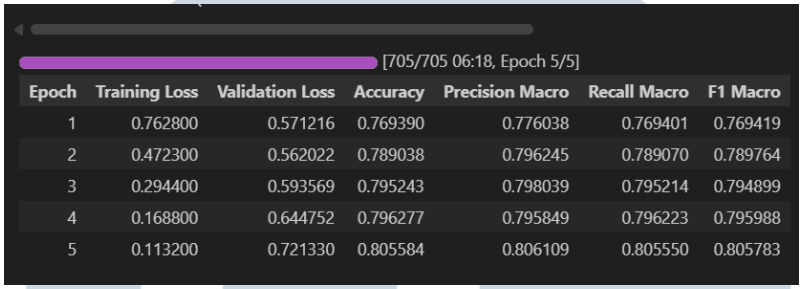
   Kode 3.34 menunjukkan proses menjalankan pelatihan menggunakan perintah `trainer.train()`. Setelah proses pelatihan selesai, model disimpan menggunakan `model.save_pretrained` pada direktori `best_model` agar dapat digunakan kembali tanpa perlu melakukan pelatihan ulang. *tokenizer* yang digunakan juga disimpan melalui `tokenizer.save_pretrained` untuk memastikan proses pemrosesan teks tetap konsisten saat model digunakan pada tahap inferensi. Dengan demikian, baik model maupun tokenizer yang telah dilatih tersimpan secara aman dan siap digunakan.

```
trainer.train()

model.save_pretrained(os.path.join(SAVE_DIR, "best_model"))
tokenizer.save_pretrained(os.path.join(SAVE_DIR, "best_model"
    ))
print("    Done     model and tokenizer saved. No rebalancing
    or cleaning applied.")
)
```

Kode 3.34: Kode menjalankan trainer dan menyimpan model

Jika program telah dijalankan maka keluaran yang akan tersedia adalah pada Gambar 3.6. Pada gambar, bisa terlihat bahwa setiap *epoch train loss* selalu

menurun, namun *validation loss* tetap masih tinggi. Namun, akurasi dan F1-score mencapai 80%. Dengan begitu, akan tetap dilanjutkan untuk melihat model dalam menghadapi data pengujian.



[705/705 06:18, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy	Precision Macro	Recall Macro	F1 Macro
1	0.762800	0.571216	0.769390	0.776038	0.769401	0.769419
2	0.472300	0.562022	0.789038	0.796245	0.789070	0.789764
3	0.294400	0.593569	0.795243	0.798039	0.795214	0.794899
4	0.168800	0.644752	0.796277	0.795849	0.796223	0.795988
5	0.113200	0.721330	0.805584	0.806109	0.805550	0.805783

Gambar 3.6. Hasil model setelah pelatihan dan evaluasi dengan data validasi

3.3.4 Tahap Uji Model

Setelah pelatihan model dan evaluasi validasi sudah memandai, maka bisa dilanjutkan dengan tahap berikutnya, yaitu tahap uji. Tahap ini akan evaluasi model dengan *dataset* uji yang telah dipisahkan pada tahap pra-pemrosesan dan melihat apakah model tersebut sudah siap dipake untuk proyek. Kode pengujian model terlampirkan sebagai berikut:

1. *Library* yang diimpor

   Library yang perlu impor untuk uji model Kode 3.35 dengan penjabaran *library* sebagai berikut:

```
import os
import pandas as pd
import torch
import numpy as np
from tqdm import tqdm
from sklearn.metrics import confusion_matrix,
    classification_report, accuracy_score
from transformers import AutoTokenizer,
    AutoModelForSequenceClassification
```

Kode 3.35: Kode *library* yang diimpor untuk pengujian model

   (a) `os`

   Library `os` merupakan modul yang menyediakan berbagai fungsi untuk berinteraksi dengan sistem operasi. *Library* tersebut hanya akan

digunakan pada fungsi `os.path.join` sebagai fungsi untuk simpan sebuah berkas atau folder dalam direktori yang sudah dibuat.

(b) `pandas`

Library ini digunakan untuk membaca dan manipulasi *dataset*. Pada kode program, `pandas` digunakan untuk memuat berkas CSV test *dataset* melalui `pd.read_csv`, memetakan label teks menjadi angka dengan `df["label"] = df["Sentiment"].map(label2id)`, melihat distribusi kelas menggunakan `df["Sentiment"].value_counts()`, mengambil *batch* teks untuk prediksi dengan `df["Review"].iloc[i:i+BATCH_SIZE].tolist()`, dan mengekstrak label sebenarnya menjadi array NumPy menggunakan `df["label"].values` untuk evaluasi.

(c) `torch`

Pada kode program, `torch` digunakan untuk menonaktifkan perhitungan gradien selama prediksi dengan `torch.no_grad()`, menghitung prediksi kelas dengan `torch.argmax`, dan mengonversi hasil prediksi dari *tensor* ke *array* NumPy menggunakan `.cpu().numpy()` agar dapat digunakan untuk evaluasi.

(d) `numpy`

Library ini digunakan untuk operasi numerik dan manipulasi *array*. Pada kode program, `numpy` digunakan untuk mengubah *list* hasil prediksi menjadi *array* NumPy dengan `np.array`, sehingga memudahkan perhitungan metrik evaluasi seperti *confusion matrix*, *classification report*, dan akurasi keseluruhan.

(e) `tqdm`

Library `tqdm` digunakan untuk menampilkan *progress bar* ketika program menjalankan suatu proses. Pada kode ini, `tqdm` digunakan untuk menampilkan perkembangan prediksi sebuah kelompok pada data uji dengan menambahkan `tqdm(..., desc="Predicting")` pada pengulangan.

(f) `sklearn.metrics import confusion_matrix, classification_report, accuracy_score`

Fungsi-fungsi yang diimpor dari `sklearn.metrics` digunakan untuk melakukan evaluasi model setelah prediksi selesai dilakukan.

`confusion_matrix` digunakan untuk membuat matriks kebingungan yang menunjukkan jumlah prediksi benar dan salah untuk setiap kelas, `classification_report` digunakan untuk menghitung dan menampilkan metrik evaluasi seperti presisi, *recall*, dan F1-score untuk setiap kelas, sedangkan `accuracy_score` digunakan untuk menghitung akurasi keseluruhan model.

(g) `transformers import AutoTokenizer, AutoModelForSequenceClassification`

Library `transformers` digunakan untuk memuat model dan Tokenizer *pre-trained* yang siap digunakan untuk klasifikasi teks. `AutoTokenizer` berfungsi untuk mengubah teks ulasan menjadi token numerik yang dapat dipahami oleh model, sementara `AutoModelForSequenceClassification` digunakan untuk memuat model klasifikasi yang telah dilatih sebelumnya dan menyiapkannya untuk melakukan prediksi. Fungsi `from_pretrained` digunakan untuk memuat Tokenizer dan model dari direktori model yang telah disimpan sehingga dapat langsung digunakan pada dataset test.

2. Konfigurasi pengujian model

Konfigurasi untuk pengujian model ditampilkan pada Kode 3.36. `SAVE_DIR` yang sudah diinisiasikan pada Kode 3.24 digunakan sebagai direktori utama untuk menyimpan hasil evaluasi model. `MODEL_DIR` yang juga sudah diinisiasikan pada Kode 3.24 menyimpan lokasi model yang siap digunakan untuk pengujian dengan bantuan fungsi `os.path.join()` dari library `os`, yang mengarah ke folder `"best_model"`. `TEST_FILE` menyimpan lokasi data pengujian dengan nama `"test.csv"`, juga dengan bantuan fungsi `os.path.join` dari `SAVE_DIR`. Selanjutnya, `label2id` memetakan label teks `"Positive"`, `"Negative"`, dan `"Neutral"` menjadi representasi numerik masing-masing 0, 1, dan 2 agar dapat digunakan dalam evaluasi model. `id2label` merupakan kebalikan dari `label2id`, yaitu konversi kembali dari prediksi numerik model menjadi label teks yang dapat dibaca. `BATCH_SIZE` menentukan jumlah sampel yang diproses dalam satu iterasi saat prediksi, membantu mengatur penggunaan memori sekaligus mempercepat proses. Sedangkan `MAX_LENGTH` menetapkan panjang maksimum token untuk setiap ulasan, sehingga teks yang lebih panjang dipangkas dan teks yang lebih

pendek diberi *padding* agar semua input memiliki panjang seragam sebelum dimasukkan ke model.

```
# ==============================
# CONFIG
# ==============================
SAVE_DIR = "./
    mbert_sentiment_gramedia_only_balanced_final_dedup_review"
MODEL_DIR = os.path.join(SAVE_DIR, "best_model")
TEST_FILE = os.path.join(SAVE_DIR, "test.csv")

label2id = {"Positive": 0, "Negative": 1, "Neutral": 2}
id2label = {v: k for k, v in label2id.items()}

BATCH_SIZE = 32
MAX_LENGTH = 128
```

Kode 3.36: Kode konfigurasi pengujian model

3. Membaca *dataset*

Proses pembacaan *dataset* yang digunakan ditampilkan pada Kode 3.37. Variabel `test_df` menyimpan data pengujian yang dimuat menggunakan `pd.read_csv()` dengan input dari variabel `TEST_FILE` yang telah diinisialisasi pada Kode 3.36. Selanjutnya, pada `test_df` dibuat kolom baru bernama `"label"` yang berisi hasil pemetaan label teks menjadi representasi numerik menggunakan fungsi `map(label2id)`, dengan `label2id` yang telah didefinisikan pada Kode 3.36 untuk kolom `"Sentiment"`. Setelah itu, program menampilkan log berupa jumlah total baris data pengujian serta distribusi jumlah baris untuk setiap kelas sentimen, yang dibantu oleh fungsi `value_counts()`.

```
# ==============================
#  Load Test Dataset
# ==============================
test_df = pd.read_csv(TEST_FILE)
test_df["label"] = test_df["Sentiment"].map(label2id)

print(f"    Loaded test dataset: {len(test_df)} samples")
print("Class distribution:\n", test_df["Sentiment"].
    value_counts(), "\n")
```

Kode 3.37: Kode membaca *dataset* pengujian

4. Membaca model dan *tokenizer*

   Pembacaan model dan *tokenizer* terlampirkan pada Kode 3.38. Variabel `MODEL_DIR` yang telah diinisialisasi pada Kode 3.36 digunakan untuk memuat model dan *tokenizer pre-trained* yang tersimpan. Fungsi `AutoTokenizer.from_pretrained` digunakan mengubah teks ulasan menjadi token numerik yang dapat diproses oleh model. Sedangkan `AutoModelForSequenceClassification.from_pretrained` digunakan untuk memuat model klasifikasi yang telah dilatih sebelumnya, sedangkan penggunaan `to(device)` memastikan model dan data berada pada perangkat yang sama sehingga mencegah error pada PyTorch. Selanjutnya, metode `model.eval()` dipanggil untuk menempatkan model dalam mode evaluasi sehingga prediksi dapat dilakukan tanpa menghitung gradient.

```
# ============================
# Load Model + Tokenizer
# ============================
tokenizer = AutoTokenizer.from_pretrained(MODEL_DIR)
model = AutoModelForSequenceClassification.from_pretrained(
    MODEL_DIR).to(device)
model.eval()
```

Kode 3.38: Kode membaca model dan *tokenizer*

5. Prediksi data pengujian

   Prediksi pada data pengujian terlampirkan pada Kode 3.39. `with torch.no_grad():` digunakan kerana semasa pengujian, model tidak perlu dilatih atau mengemas kini parameter. Tanpa arahan ini, PyTorch akan mengira gradien seperti dalam proses latihan. Proses prediksi dilakukan secara batch menggunakan `BATCH_SIZE` yang telah diinisiasikan pada Kode 3.36 untuk mengatur jumlah sampel yang diproses sekaligus, sehingga meminimalkan penggunaan memori dan mempercepat proses. Perulangan `for` digunakan untuk mengambil setiap batch dari kolom `"Review"` pada `test_df`, yang kemudian dikonversi menjadi token numerik oleh `tokenizer` dengan pengaturan `truncation=True`, `padding=True`, panjang maksimum `MAX_LENGTH` yang telah diinisiasikan pada Kode 3.36, dan `return_tensor` yang akan menghasilkan *torch.tensor*. Model menghasilkan `outputs` melalui pemanggilan `model(**enconding)`, yang berupa vektor skor untuk setiap kelas. Prediksi kelas kemudian diperoleh dengan memilih indeks

kelas dengan nilai tertinggi menggunakan `torch.argmax`. Hasil prediksi setiap sampel kemudian dikonversi ke *array* NumPy dan dikumpulkan dalam `all_preds` untuk evaluasi selanjutnya. *Progress bar* dari `tqdm` ditampilkan untuk menunjukkan sejauh mana proses prediksi telah berjalan.

```
# ============================
# Predict in Batches
# ============================
all_preds = []

with torch.no_grad():
    for i in tqdm(range(0, len(test_df), BATCH_SIZE), desc="Predicting"):
        batch_texts = test_df["Review"].iloc[i:i+BATCH_SIZE].tolist()
        encodings = tokenizer(
            batch_texts,
            truncation=True,
            padding=True,
            max_length=MAX_LENGTH,
            return_tensors="pt"
        ).to(device)

        outputs = model(**encodings)
        preds = torch.argmax(outputs.logits, dim=1)
        all_preds.extend(preds.cpu().numpy())
```

Kode 3.39: Kode prediksi data pengujian

6. Penjabaran Evaluasi

Penjabaran proses evaluasi setelah prediksi data pengujian ditunjukkan pada Kode 3.41. Pada tahap ini, nilai `y_true` diambil dari kolom `"label"` pada `test_df`, sedangkan `y_pred` merupakan hasil prediksi yang telah dikumpulkan dalam `all_preds`. Evaluasi dimulai dengan pembuatan *confusion matrix* menggunakan fungsi `confusion_matrix`, yang kemudian dikonversi ke dalam bentuk *DataFrame* agar mudah dibaca, dengan indeks dan kolom yang dipetakan menggunakan `id2label`. Matriks ini memberikan gambaran visual mengenai jumlah prediksi benar dan salah untuk setiap kelas sentimen. Selanjutnya, laporan klasifikasi dihasilkan melalui fungsi `classification_report`, yang menyajikan metrik presisi, *recall*, dan F1-score untuk masing-masing kelas *Positive*, *Negative*, dan *Neutral*. Tahap

evaluasi ditutup dengan perhitungan akurasi keseluruhan menggunakan `accuracy_score`, yang menunjukkan persentase prediksi yang tepat dari keseluruhan sampel uji.

```
# ============================
# 4        Evaluation
# ============================
y_true = test_df["label"].values
y_pred = np.array(all_preds)

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
cm_df = pd.DataFrame(cm, index=id2label.values(), columns=
    id2label.values())
print("\nConfusion Matrix (rows = true, cols = predicted):\n"
    )
print(cm_df)

# Classification Report
print("\nClassification Report:\n")
report = classification_report(y_true, y_pred, target_names=[
    "Positive", "Negative", "Neutral"], digits=4)
print(report)

# Overall Accuracy
acc = accuracy_score(y_true, y_pred)
print(f"\ n   Overall Accuracy: {acc * 100:.2f}%")
```

Kode 3.40: Kode penjabaran Evaluasi

7. Menyimpan hasil evaluasi uji model

Proses penyimpanan hasil evaluasi setelah pengujian model ditunjukkan pada Kode 3.41. Pada tahap ini, dua berkas utama dihasilkan, yaitu `classification_report.txt` dan `confusion_matrix.csv`. Dua berkas ini akan disimpan di direktori `SAVE_DIR` yang sudah diinisiasikan pada Kode 3.36. Variabel `report_path` dan `cm_path` digunakan untuk menentukan lokasi penyimpanan kedua berkas tersebut.

Berkas `classification_report.txt` dibuat dengan `with open(report_path, "w", encoding="utf-8")` yang membuka berkas menggunakan mode tulis dan menuliskan beberapa komponen evaluasi, seperti *confusion matrix*, laporan klasifikasi, serta nilai akurasi

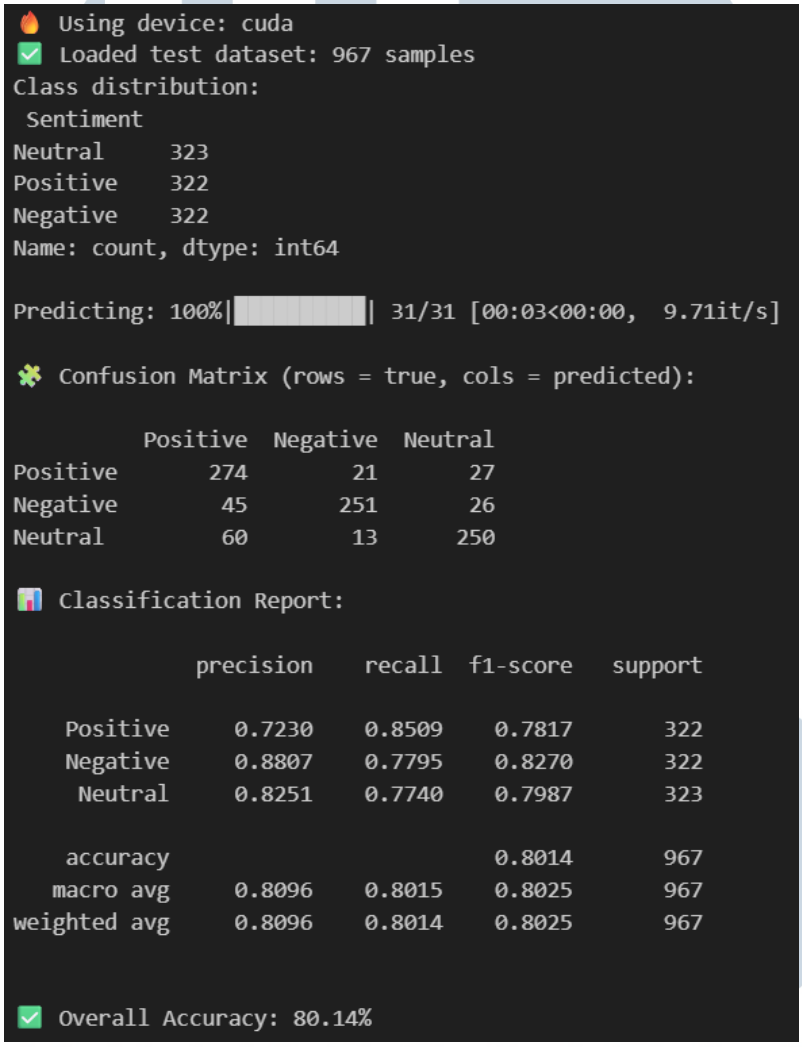
keseluruhan. Sementara itu, `confusion_matrix.csv` dihasilkan dengan menyimpan DataFrame `cm_df` ke dalam format CSV menggunakan fungsi `to_csv()`. Pada akhir proses, program menampilkan lokasi penyimpanan kedua berkas tersebut sebagai konfirmasi bahwa hasil evaluasi telah berhasil disimpan.

```
# ============================
# 5        Save Evaluation Results
# ============================
report_path = os.path.join(SAVE_DIR, "classification_report.
    txt")
cm_path = os.path.join(SAVE_DIR, "confusion_matrix.csv")

with open(report_path, "w", encoding="utf-8") as f:
    f.write("Confusion Matrix:\n")
    f.write(cm_df.to_string())
    f.write("\n\nClassification Report:\n")
    f.write(report)
    f.write(f"\n\nOverall Accuracy: {acc * 100:.2f}%")

cm_df.to_csv(cm_path)

print(f"\n Reports saved to:\n - {report_path}\n - {cm_path}"
    )
```

Kode 3.41: Kode menyimpan hasil evaluasi uji model

Setelah program evaluasi data pengujian model dijalankan, hasil keluarannya ditampilkan pada Gambar 3.7. Berdasarkan hasil tersebut, akurasi model mencapai sekitar 80%, yang dapat dianggap cukup baik. Dengan distribusi kelas, yaitu netral, positif, dan negatif, pada data uji yang berada dalam kondisi seimbang, kemampuan model dapat direpresentasikan dengan baik melalui nilai akurasi. Namun, jika ditinjau menggunakan metrik evaluasi lainnya, pada metrik presisi terlihat bahwa kelas negatif memiliki nilai tertinggi sebesar 88%, diikuti oleh kelas netral sebesar 82% dan kelas negatif sebesar 72%. Hal ini menunjukkan bahwa model memiliki kemampuan yang baik dan dapat divalidasi bahwa prediksi pada kelas negatif cenderung benar-benar termasuk dalam kategori negatif, sedangkan pada kelas positif masih terdapat beberapa data yang diprediksi sebagai positif namun sebenarnya tidak. Pada metrik *recall*, performa didominasi oleh kategori positif dengan nilai sebesar 85%, diikuti oleh kategori netral dan

negatif sebesar 77%. Hal ini menunjukkan bahwa model lebih mudah mengenali data pada kategori positif dibandingkan kategori negatif dan netral, yang relatif lebih sulit untuk teridentifikasi [32]. Jika metrik presisi dan *recall* digabungkan, maka F1-score terbaik diperoleh oleh kelas negatif sebesar 83%, diikuti oleh kelas netral sebesar 79%, dan kelas positif sebesar 78%.



Gambar 3.7. Hasil evaluasi data pengujian

3.4 Kendala yang Ditemukan

Dalam pelaksanaan magang, terdapat dua kendala yang menghalang efisiensi dan kinerja proyek yang terlampirkan sebagai berikut:

1. lambat pelatihan model

Pada pelatihan kedua model, model diperlukan 2 jam lebih untuk menyelesaikannya. Namun pasa iterasi yang pertama, model bisa dilatihkan dan selesai cukup menunggu 20 menit. Kebingunan ini juga berlanjut sebab selama pelatihan GPU yang digunakan dan bukan CPU. Seharusnya dengan GPU, pelatihan cuman perlu minimal dalam interval menit bukan berjam-jam.

2. Proyek harus diselesaikan dalam 2 minggu

   Walau pada Tabel 3.1 telah dilampirkan bahwa pengerjaan proyek ini berlangsung sampai November, namun pada bulan September, pertama kali diberikan informasi tentang proyek ini, diberikan 2 minggu untuk menyelesaikannya. Jika lebih spesifik batas waktu adalah akhir bulan September, tapi pada 2 minggu awal September telah memberikan waktu untuk proyek lainnya. Maka setelah tugas-tugas yang perlu dikerjakan sudah selesai, maka baru bisa lanjut proyek pembuatan sistem klasifikasi sentimen konsumen.

3.5 Solusi atas Kendala yang Ditemukan

Walau kendalanya menghambat pelaksanaan proyek, 2 solusi telah ditemukan dan pada akhirnya melancarkan kinerja yang terlampirkan sebagai berikut:

1. Mulai ulang Kernel setiap menyelesaikan pelatihan model

   Setelah riset lebih dalam permasalahan pelatihan yang lambat, ditemukan bahwa permasalahan terjadi pada Kernel Python. Pada forum PyTorch banyak orang telah menemuman kendala sama yang membuat pelatihan berikutnya melambat setiap menjalankan program pelatihan model [33]. Pada situs mindfulchase.com telah terlampirkan beberapa solusi, salah satunya adalah mulai ulang Kernel untuk eliminasi memori yang tidak diperlukan pada Kernel [34]. Setelah dimulai ulang, pelatihan berjalan secara normal lagi. Ini akan tetap lakukan jika perlambatan mulai terjadi lagi.

2. Minggu pertama mempelajari tugas, minggu kedua pelaksanaan

   Selama pelaksanaan magang, pelajari sebuah proyek dan mencari informasi strategi-strategi yang bisa dilakukan sebelum pelaksanaan pembuatan model lebih memandai dibanding sebaliknya. Maka pada proyek ini telah mengikuti prosedur tersebut. Dikarenakan memberikan waktu 1 minggu

untuk mempelajari lebih dalam tentang proyek, maka pada minggu kedua lebih mudah dalam pelaksanaan proyek dan bisa diselesaikan pada akhir September. Walau setelah memeriksa lagi pada bulan kedepannya ada yang masih belum memandai, pada akhir batas waktu, model telah disetujui dan sudah bisa dipakai dengan waktu yang diberikan.