BAB 3
METODOLOGI PENELITIAN

Dalam melakukan penelitian untuk merancang Analisis Sentimen Pengguna Social Media X terhadap Program Makan Bergizi Gratis menggunakan Hard Voting dan Soft Voting Ensemble dengan Naive Bayes dan SVM, memiliki beberapa tahapan perancangan.

3.1 Metodologi Penelitian

Penelitian ini terdiri dari beberapa tahapan utama yang dilakukan secara sistematis untuk mencapai tujuan penelitian. Berikut adalah tahapan yang dilakukan:

Penelitian ini dilakukan secara sistematis melalui beberapa tahapan berikut:

1. **Studi Literatur:** Tahap awal dilakukan dengan meninjau berbagai literatur yang relevan untuk memahami dasar teori mengenai analisis sentimen, preprocessing teks, algoritma klasifikasi (Naive Bayes, SVM, Random Forest), serta teknik ensemble learning seperti Stacking.

2. **Crawling Data:** Data dikumpulkan dari media sosial X (Twitter) menggunakan Tweet Harvester. Proses ini mencakup pemilihan sumber data, kata kunci (*keyword*), bahasa yang digunakan, serta rentang waktu pengambilan data.

3. **Preprocessing dan Labeling Data:** Data yang diperoleh kemudian dibersihkan melalui tahapan preprocessing seperti case folding, cleaning, tokenizing, stopword removal, stemming, dan normalisasi. Setelah itu, data diberi label sentimen menggunakan model *IndoBERT*.

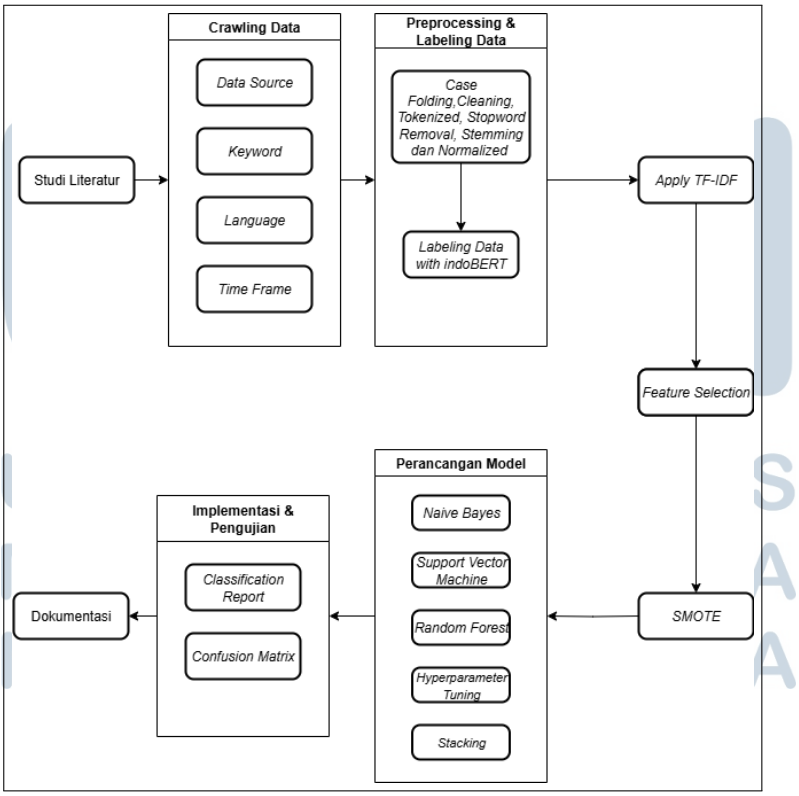
4. **Ekstraksi dan Seleksi Fitur:** Data teks dilanjutkan ke tahap ekstraksi fitur menggunakan metode TF-IDF. Setelah itu, dilakukan proses seleksi fitur untuk memilih fitur yang paling relevan dalam mendukung proses klasifikasi.

5. **Balancing Data:** Teknik SMOTE (Synthetic Minority Over-sampling Technique) diterapkan hanya pada data pelatihan untuk mengatasi ketidakseimbangan kelas yang mungkin terjadi dalam dataset.

6. **Perancangan Model:** Model baseline dibangun menggunakan tiga algoritma klasifikasi yaitu Naive Bayes, SVM, dan Random Forest. Masing-masing model kemudian dituning menggunakan hyperparameter tuning untuk meningkatkan performa, dan akhirnya digabungkan menggunakan teknik Stacking Ensemble.
7. **Implementasi dan Evaluasi:** Model yang telah dirancang kemudian diuji dengan data uji. Hasil klasifikasi dievaluasi menggunakan Confusion Matrix serta metrik evaluasi seperti akurasi, precision, recall, dan F1-score.
8. **Dokumentasi:** Seluruh proses dan hasil yang diperoleh didokumentasikan sebagai bagian dari laporan penelitian untuk dianalisis lebih lanjut dan menjadi dasar rekomendasi di penelitian selanjutnya.

3.2 Alur Metodologi Penelitian

Penelitian ini mengikuti metodologi atau tahapan yang disajikan dalam bentuk diagram alur (flowchart). Pada Gambar 3.1 merupakan gambar *flowchart* dari metodologi penelitian yang dilakukan pada penelitian ini.



Gambar 3.1. *Pipeline metodologi penelitian*

Pada alur metodologi di Gambar 3.1, langkah seleksi fitur menggunakan metode *Chi-Square* dilakukan setelah proses *preprocessing* selesai dan sebelum data dibagi menjadi data latih dan data uji. Seleksi fitur bertujuan untuk mengurangi fitur yang tidak relevan dan meningkatkan kinerja model.

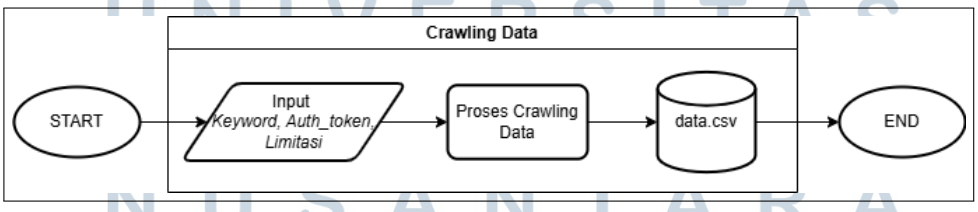
3.2.1 Studi Literatur

Studi literatur dilakukan dengan meninjau berbagai penelitian sebelumnya yang membahas *analisis sentimen*, *ensemble learning*, serta penggunaan algoritma *Naive Bayes* dan *Support Vector Machine (SVM)* dalam klasifikasi teks. Studi ini bertujuan untuk memahami pendekatan yang telah digunakan serta menemukan *research gap* yang mendukung penggunaan metode *Hard Voting* dan *Soft Voting Ensemble* dalam penelitian ini.

3.2.2 Crawling Data

Data yang digunakan dalam penelitian ini diperoleh dari platform *X (Twitter)* menggunakan *Tweet Harvest*. Pengumpulan data dilakukan dengan ketentuan berikut:

- **Sumber Data**: Tweet yang membahas Program Makan Bergizi Gratis.
- **Kata Kunci**: *"makan gratis lang:id since:2024-12-01 until:2025-02-24"* dan variasi ejaannya.
- **Bahasa**: Hanya tweet yang berbahasa Indonesia.
- **Periode Pengambilan Data**: Mulai 1 Desember 2024 sampai 24 Februari 2025 dengan filter.



Gambar 3.2. *Flowchart* pengumpulan data dengan metode *crawling*

Pada Gambar 3.2 merupakan gambar *flowchart* pengambilan data dengan metode *crawling data*. Proses tersebut meliputi memasukkan input berupa

keywords, auth_token, dan limitasi, proses crawling dilakukan menggunakan *Google Colab*, dan menyimpan hasil dari crawling pada file data.csv.

3.2.3 Preprocessing & Labeling Data

Setelah data dikumpulkan, dilakukan tahapan preprocessing untuk membersihkan teks dan labeling otomatis untuk menentukan sentimen dari tweet.

A Konfigurasi dan Pengambilan Data

Bagian ini menjelaskan proses awal sebelum pengambilan data dilakukan, meliputi konfigurasi token autentikasi Twitter dan instalasi dependensi penting seperti `pandas` untuk pengolahan data dan `Node.js` untuk menjalankan alat pengambil data *tweet-harvest*. Token autentikasi digunakan untuk memberikan akses API Twitter. Instalasi mencakup pembaruan daftar paket sistem, instalasi dependensi umum, hingga konfigurasi repositori Node.js versi 20 dan pengecekan versi untuk memastikan keberhasilan instalasi.

```
SET twitter_auth_token = "your_auth_token"

# Instalasi dependensi sistem dan Python
INSTALL package "pandas"
UPDATE system package list
INSTALL packages: ca-certificates, curl, gnupg
CREATE folder "/etc/apt/keyrings" if not exists
DOWNLOAD NodeSource GPG key
SAVE GPG key to "/etc/apt/keyrings/nodesource.gpg"
ADD Node.js source repository (versi 20)
UPDATE system package list again
INSTALL Node.js
CHECK Node.js version
```

Kode 3.1: Konfigurasi token dan instalasi tweet harvest

Setelah konfigurasi selesai, proses pengambilan data dilakukan menggunakan `tweet-harvest`. Kata kunci pencarian yang digunakan adalah ``makan gratis'' dengan filter bahasa Indonesia dan rentang tanggal tertentu. Hasil pencarian dibatasi hingga 10.000 tweet dan disimpan ke dalam file bernama `data.csv`.

```
# Crawling data tweet
SET filename = "data.csv"
SET search_keyword = "makan gratis lang:id since:2024-12-01 until
    :2025-02-24"
SET limit = 10000

RUN tweet-harvest with:
    - Output file: filename
    - Search keyword: search_keyword
    - Tab: LATEST
    - Limit: limit
```

```
    - Token: twitter_auth_token
```

Kode 3.2: Menentukan keyword dan menyimpan hasil crawling

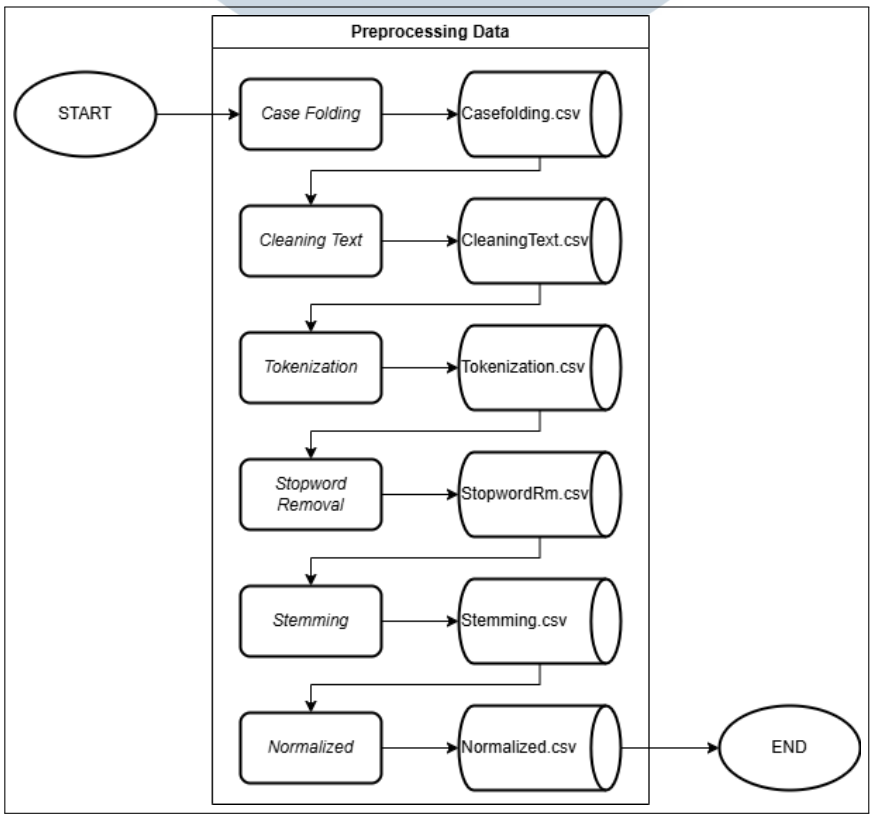
Setelah data berhasil diambil, file CSV hasil *crawling* dimuat ke dalam `pandas DataFrame` untuk diproses lebih lanjut. Data kemudian ditampilkan sebagai tabel agar dapat dilihat dan dianalisis secara visual.

```
# Load data tweet ke DataFrame
SET file_path = "tweets-data/" + filename
READ file_path as CSV into DataFrame
TAMPILKAN isi DataFrame
```

Kode 3.3: Menampilkan data tweet ke dalam tabel

B Preprocessing Data

Pada Gambar 3.3 ditampilkan tahapan preprocessing data yang dilakukan dalam penelitian ini, yang disusun dalam bentuk flowchart untuk menggambarkan alur proses secara sistematis mulai dari input data mentah hingga data siap digunakan untuk pemodelan.



Gambar 3.3. *Flowchart preporcessing data*

- **Case Folding:** Mengubah seluruh teks menjadi huruf kecil agar tidak terjadi perbedaan antara huruf kapital dan huruf kecil. Proses ini dilakukan dengan membaca file data awal, lalu setiap elemen dalam kolom `full_text` diubah menjadi huruf kecil dan hasilnya disimpan ke file baru.

```
SET file_path TO "D:/SKRIPSI/Data/Data/data_skripsi_cleaned.csv"

LOAD CSV file from file_path INTO dataframe df

# Proses Case Folding
FOR each text IN df["full_text"]:
    UBAH text menjadi huruf kecil

SIMPAN dataframe df ke file baru:
    "D:/SKRIPSI/Data/Data/Preprocessing/data_skripsi_casefolded.csv"
    (tanpa menyimpan index)

TAMPILKAN isi dataframe df

```

Kode 3.4: Case Folding: Mengubah teks menjadi huruf kecil

- **Cleaning Teks:** Membersihkan teks dari URL, mention, hashtag, angka, karakter non-alfabet, serta spasi berlebih. Fungsi `clean_text` didefinisikan dan diterapkan ke seluruh teks dalam data.

```
SET file_path TO "D:/SKRIPSI/Data/Data/Preprocessing/data_skripsi_casefolded.csv"

LOAD CSV file dari file_path KE dataframe df

DEFINE fungsi clean_text(text):
    HAPUS URL (yang diawali dengan http atau www)
    HAPUS mention (@username) dan hashtag (#tag)
    HAPUS karakter non-alfabet (hanya sisakan huruf dan spasi)
    GANTI spasi berlebih dengan satu spasi, lalu HILANGKAN spasi di awal/akhir
    KEMBALIKAN hasil teks yang telah dibersihkan

APLIKASIKAN fungsi clean_text ke setiap nilai dalam kolom "full_text" pada df

SIMPAN dataframe df ke file baru:
    "D:/SKRIPSI/Data/Data/Preprocessing/data_skripsi_cleaned_text.csv"
    (tanpa menyimpan index)

TAMPILKAN isi dataframe df

```

Kode 3.5: Cleaning: Membersihkan simbol pada data

NUSANTARA

- **Tokenisasi:** Memisahkan teks menjadi kata-kata individual (token). Tiap kalimat dipecah menjadi kumpulan kata, yang kemudian disimpan dalam format daftar.

```
SET file_path TO "D:/SKRIPSI/Data/Data/Preprocessing/
    data_skripsi_cleaned_text.csv"

LOAD CSV file dari file_path KE dataframe df

UNTUK setiap teks dalam kolom "full_text" di df:
    BAGI teks menjadi token-token kata (menggunakan
    word_tokenize)
    SIMPAN hasil tokenisasi ke kolom yang sama

SIMPAN dataframe df ke file baru:
    "D:/SKRIPSI/Data/Data/Preprocessing/
    data_skripsi_tokenized.csv"
    (tanpa menyimpan index)

TAMPILKAN kolom "full_text" dari df

```

Kode 3.6: Tokenisasi: Memisahkan kata-kata dalam teks

- **Stopword Removal:** Menghapus kata-kata umum yang tidak memiliki makna penting seperti “yang”, “dan”, atau “di”. Stopword Bahasa Indonesia diambil dari pustaka NLTK, lalu digunakan untuk memfilter token.

```
IMPORT daftar stopword Bahasa Indonesia dari pustaka NLTK

UNDUH stopword jika belum tersedia

SET file_path TO "D:/SKRIPSI/Data/Data/Preprocessing/
    data_skripsi_tokenized.csv"

LOAD file CSV dari file_path KE dataframe df

UBAH setiap nilai pada kolom "full_text" menjadi list (dari
    string ke list Python)

AMBIL daftar stopword Bahasa Indonesia dan simpan ke dalam
    stopwords_indonesia

DEFINE fungsi remove_stopwords(text_tokens):
    UNTUK setiap word dalam text_tokens:
        JIKA word TIDAK termasuk dalam stopwords_indonesia:
            SIMPAN word ke hasil akhir
    KEMBALIKAN daftar hasil akhir

APLIKASIKAN fungsi remove_stopwords ke setiap baris pada
    kolom "full_text"

SIMPAN dataframe df ke file baru:
    "D:/SKRIPSI/Data/Data/Preprocessing/
    data_skripsi_stopwords_removed.csv"
    (tanpa menyimpan index)

TAMPILKAN kolom "full_text" dari df

```

Kode 3.7: Stopword Removal: Menghapus kata umum yang tidak penting

- **Stemming:** Mengubah kata menjadi bentuk dasarnya menggunakan algoritma Nazief & Adriani yang tersedia pada pustaka `Sastrawi`. Setiap kata dalam token distem untuk menyederhanakan variasi kata.

```
INSTALL library "Sastrawi" untuk melakukan stemming Bahasa Indonesia

IMPORT StemmerFactory dari Sastrawi
IMPORT modul ast untuk mengonversi string list ke objek list

SET file_path TO "D:/SKRIPSI/Data/Data/Preprocessing/data_skripsi_stopwords_removed.csv"

LOAD file CSV dari file_path KE dataframe df

UBAH setiap nilai dalam kolom "full_text" menjadi list Python menggunakan ast.literal_eval

BUAT objek stemmer menggunakan Sastrawi

DEFINE fungsi stemming_text(text_tokens):
    UNTUK setiap word dalam text_tokens:
        UBAH word menjadi bentuk dasar menggunakan stemmer
    KEMBALIKAN daftar hasil stemming

APLIKASIKAN fungsi stemming_text ke setiap baris dalam kolom "full_text"

SIMPAN dataframe df ke file baru:
    "D:/SKRIPSI/Data/Data/Preprocessing/data_skripsi_stemmed.csv"
    (tanpa menyimpan index)

TAMPILKAN kolom "full_text" dari df

```

Kode 3.8: Stemming: Mengubah kata ke bentuk dasar

- **Normalisasi:** Menghapus kata tidak baku atau singkatan seperti “yg”, “tdk”, dan “dmn”. Setiap token diperiksa dan difilter agar hanya menyisakan kata-kata formal atau yang memiliki panjang lebih dari tiga huruf.

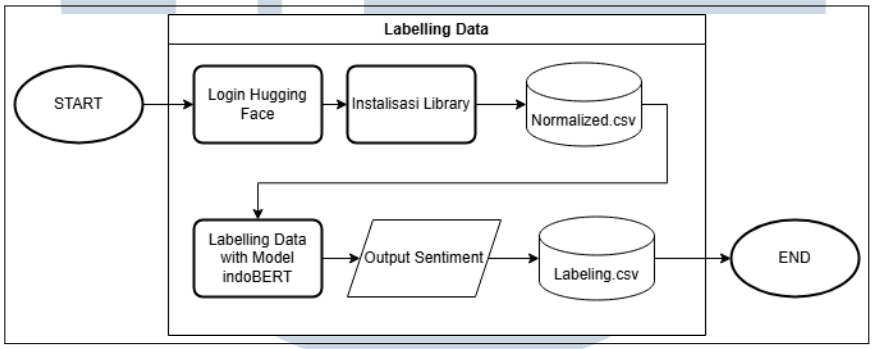
```
SET input_path TO "D:/SKRIPSI/Data/Data/Preprocessing/data_skripsi_stemmed.csv"

LOAD file CSV dari input_path KE dataframe df

BUAT daftar singkatan_tidak_diinginkan:
    {"dmn", "yg", "sm", "dll", "tdk", "krn", "gk", "ga", "aja"}

DEFINE fungsi hapus_singkatan(text_tokens):
    JIKA text_tokens berupa string:
        BERSIHKAN tanda kurung dan kutip dari string
        PECAH string menjadi list kata menggunakan koma dan spasi
    FILTER list dengan ketentuan:
        HANYA simpan kata yang TIDAK ada dalam daftar singkatan_tidak_diinginkan
        DAN panjang katanya lebih dari 3 huruf
    KEMBALIKAN list hasil filter

APLIKASIKAN fungsi hapus_singkatan ke setiap nilai di kolom "full_text"

SIMPAN dataframe df ke file baru:
    "D:/SKRIPSI/Data/Data/Preprocessing/data_skripsi_normalized.csv"
```

```
    (tanpa menyimpan index)

TAMPILKAN kolom "full_text" dari df

```

Kode 3.9: Normalisasi: Menghapus kata tidak baku atau slang

C Automatic Labeling dengan IndoBERT Sentiment Classifier

Pada Gambar 3.4 ditampilkan *flowchart* proses pelabelan data sentimen secara otomatis menggunakan model **IndoBERT** yang telah dilatih untuk klasifikasi sentimen dalam Bahasa Indonesia.



Gambar 3.4. *Flowchart labeling data* dengan *indobert*

Langkah-langkah pelabelan otomatis dilakukan sebagai berikut:

1. **Login Hugging Face**
   Pengguna perlu login terlebih dahulu ke akun Hugging Face dengan menjalankan perintah `huggingface-cli login` pada terminal, lalu memasukkan *access token* dari akun Hugging Face yang telah dibuat di situs `https://huggingface.co`.

2. **Instalasi Library yang Dibutuhkan**
   Beberapa library yang diperlukan adalah `transformers`, `torch`, dan `huggingface_hub`. Instalasi dilakukan dengan perintah `pip install transformers==4.30.2 torch==2.0.1 huggingface_hub==0.14.1`.

3. **Pemanggilan Model dan Tokenizer**
   `w11wo/indonesian-roberta-base-sentiment-classifier` merupakan model indoBERT yang digunakan. Tokenizer digunakan untuk mengubah kalimat menjadi format token numerik agar dapat diproses oleh model. Model

dan tokenizer diakses melalui fungsi `AutoTokenizer.from_pretrained()` dan `AutoModelForSequenceClassification.from_pretrained()`.

4. **Pelabelan Sentimen**

   Teks hasil preprocessing digunakan sebagai input ke tokenizer, kemudian ditransformasikan ke dalam tensor dan diproses oleh model. Output dari model berupa logits, kemudian diproses menggunakan `softmax` dan diklasifikasikan ke dalam tiga label sentimen, yaitu positif, netral dan negatif:

   Potongan kode berikut digunakan untuk melakukan pelabelan otomatis terhadap data sentimen menggunakan model *indoBERT* yang disediakan oleh HuggingFace.

```
IMPORT pustaka: torch, pandas, dan HuggingFace Transformers (AutoTokenizer, AutoModelForSequenceClassification)

SET model_name TO "W11wo/indonesian-roberta-base-sentiment-classifier"

MUAT tokenizer dan model dari HuggingFace berdasarkan model_name

SET file_path TO lokasi file "data_skripsi_convert_text.csv"
BACA file CSV ke dalam dataframe df

ISI nilai kosong di kolom "full_text" dengan string kosong

DEFINE fungsi batch_predict_sentiment(texts, batch_size):
    BUAT daftar label sentimen: ["Negatif", "Netral", "Positif"]
    INISIALISASI list kosong untuk hasil

    UNTUK setiap batch dari texts (dengan ukuran batch_size):
        Hitung probabilitas tiap label menggunakan softmax
        TENTUKAN label dengan skor tertinggi untuk tiap teks
        TAMBAHKAN label ke dalam list hasil

    KEMBALIKAN list hasil

KONVERSI kolom "full_text" ke list dan simpan ke text

JALANKAN fungsi batch_predict_sentiment untuk text
SIMPAN hasil ke kolom baru: "sentiment"

SIMPAN dataframe df ke file baru: "data_skripsi_labeled_indobert_new.csv" (tanpa index)
```

Kode 3.10: Labeling otomatis data menggunakan model indoBERT

Kode ini menggunakan model klasifikasi sentimen `indonesian-roberta-base-sentiment-classifier`. Data teks dibaca

dari file CSV dan diproses dalam bentuk batch menggunakan fungsi `batch_predict_sentiment`. Proses prediksi dilakukan dengan mengukur probabilitas setiap label sentimen, kemudian label dengan skor tertinggi ditentukan sebagai hasil klasifikasi. Label yang diperoleh disimpan ke dalam kolom baru bernama `sentiment` sebelum disimpan ke file CSV baru untuk keperluan analisis lanjutan.

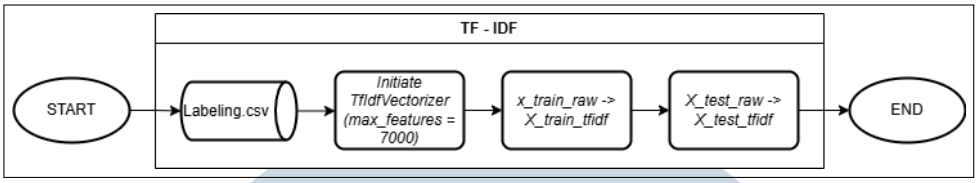
5. **Validasi Label Sentimen:**
   Proses pelabelan sentimen pada penelitian ini dilakukan secara otomatis menggunakan model*IndoBERT* dengan nama `W11wo/indonesian-roberta-base-sentiment-classifier`. Model telah melalui tahap *fine-tuning* dan evaluasi oleh pengembangnya menggunakan dataset sentimen berbahasa Indonesia, serta dipublikasikan melalui platform *HuggingFace*. Sehingga, validasi dilakukan secara tidak langsung dengan mengandalkan akurasi dan performa model sebagaimana disampaikan dalam dokumentasi model tersebut. Tidak dilakukan validasi manual oleh pakar karena pelabelan berbasis *machine-generated* dianggap cukup representatif untuk kebutuhan eksperimen klasifikasi sentimen.

3.2.4 Feature Engineering

Penelitian ini menggunakan metode **TF-IDF (Term Frequency–Inverse Document Frequency)** sebagai pendekatan utama dalam tahap *feature engineering*. TF-IDF digunakan untuk merepresentasikan data teks hasil preprocessing ke dalam bentuk vektor numerik berbasis bobot kata, yang dapat diproses oleh algoritma klasifikasi. Pemilihan metode ini didasarkan pada kemampuannya dalam menangkap pentingnya suatu kata dalam dokumen sekaligus mempertimbangkan frekuensinya di seluruh korpus.

A TF-IDF

Pada Gambar 3.5 ditunjukkan diagram alur dari proses pembobotan kata menggunakan *Term Frequency - Inverse Document Frequency (TF-IDF)*.



Gambar 3.5. *Flowchart* tf-idf

TF-IDF digunakan untuk mengekstraksi fitur dari teks dengan memberikan bobot pada setiap kata berdasarkan frekuensi kemunculan dan kepentingannya dalam dokumen secara keseluruhan. Pada potongan kode berikut menunjukkan kode untuk melakukan proses *feature engineering* yaitu TF-IDF.

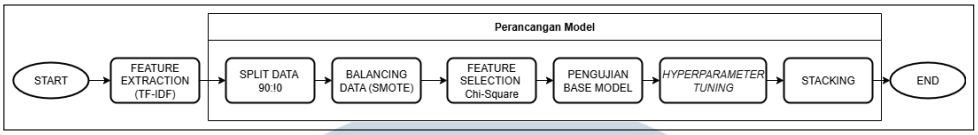
```
IMPORT TF-IDF Vectorizer dari pustaka scikit-learn

INISIALISASI TF-IDF Vectorizer dengan:
    - Jumlah maksimum fitur: 7000

# === Proses pada Data Training ===
LAKUKAN fit dan transformasi pada data training (X_train_raw)
HASILKAN representasi vektor TF-IDF dan simpan ke
X_train_tfidf

# === Proses pada Data Testing ===
LAKUKAN transformasi pada data testing (X_test_raw)
GUNAKAN vectorizer yang telah di-fit dari data training
SIMPAN hasilnya ke X_test_tfidf
```

Kode 3.11: pengaplikasian TF-IDF

3.2.5 Perancangan Model

Penelitian ini menggunakan pendekatan **ensemble learning** untuk meningkatkan performa klasifikasi sentimen terhadap data media sosial. Algoritma yang digunakan terdiri dari **Multinomial Naive Bayes (MNB)**, **Support Vector Machine (SVM)**, dan **Random Forest (RF)**, yang dikombinasikan menggunakan metode *ensemble learning* yaitu **Stacking Classifier**.

Pada Gambar 3.6 menunjukkan gambar perancangan *flowchart* untuk perancangan model pada penelitian ini. Tahapan perancangan model secara sistematis dijelaskan sebagai berikut:



Gambar 3.6. Flowchart perancangan model pengujian

1. ***Splitting Data***: Proses ini bertujuan untuk membagi data menjadi data latih dan data uji agar model dapat diuji secara objektif. Data dibagi dengan rasio 90:10 menggunakan metode *stratified split* untuk menjaga proporsi kelas tetap seimbang.

```
# === Encoding Label ===
UBAH label asli (y_raw) menjadi bentuk numerik
menggunakan LabelEncoder
SIMPAN hasilnya ke variabel y_encoded

# === Pembagian Data ===
BAGI data fitur (X_tfidf) dan label (y_encoded) menjadi:
    - Data latih: X_train, y_train
    - Data uji: X_test, y_test
    DENGAN:
    - Rasio data uji = 10%
    - Stratify = y_encoded (agar distribusi kelas
seimbang)
    - random_state = 42 (untuk hasil yang konsisten)

```

Kode 3.12: Pengaplikasian splitting data

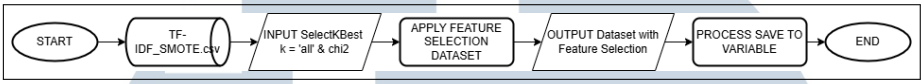
2. ***Balancing Data***: Untuk mengatasi ketidakseimbangan kelas, digunakan teknik SMOTE (Synthetic Minority Over-sampling Technique) pada data hasil ekstraksi TF-IDF.

```
INISIALISASI SMOTE dengan random_state = 42
TERAPKAN SMOTE pada X_train dan y_train
HASILKAN X_train_smote dan y_train_smote

```

Kode 3.13: Pengaplikasian SMOTE

SMOTE digunakan untuk menyeimbangkan jumlah data antar kelas. Proses ini dimulai dengan memisahkan fitur dan label, mengonversi label menjadi numerik, lalu menerapkan SMOTE untuk menghasilkan data sintetis pada kelas minoritas. Data hasil *oversampling* kemudian disimpan sebagai file baru untuk digunakan pada tahap pelatihan model klasifikasi.

3. ***Feature Selection***: Menggunakan metode **SelectKBest** dengan fungsi *chi-square* untuk memilih fitur yang paling relevan terhadap label sentimen. Pada Gambar 3.7 menunjukkan alur atau *flowchart* dari implementasi *feature selection*.



Gambar 3.7. Flowchart *feature selection* menggunakan SelectKBest

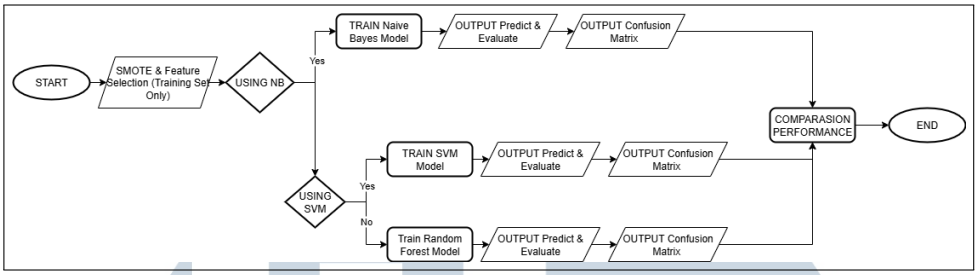
```
INISIALISASI SelectKBest dengan metode chi-square dan k = 'all' (atau nilai tertentu)
LAKUKAN fit_transform pada X_train_smote dan y_train_smote
TRANSFORM X_test menggunakan selector yang sama

```

Kode 3.14: Implementasi kode pengujian feature seleciton

Untuk menghindari *data leakage*, proses *balancing data* menggunakan SMOTE hanya diterapkan pada data latih yang telah dibagi sebelumnya. Setelah proses resampling selesai, dilakukan seleksi fitur menggunakan metode *Chi-Square* untuk mengurangi dimensi dan memilih fitur paling relevan. Tahapan ini kemudian diikuti dengan pelatihan model menggunakan pendekatan ensemble Stacking.

4. **Perancangan Baseline Model**: Model Naive Bayes, SVM, dan Random Forest digunakan sebagai baseline dengan **hyperparameter terbaik** hasil dari proses tuning.

   Pada Gambar 3.8 menunjukkan alur proses pemodelan *baseline* menggunakan tiga algoritma pembelajaran mesin, yaitu Naive Bayes, SVM, dan Random Forest. Dataset yang digunakan merupakan hasil dari tahap preprocessing, termasuk TF-IDF, pembagian data, SMOTE, dan seleksi fitur. Setiap model dilatih dan dievaluasi secara paralel untuk memperoleh metrik performa sebagai acuan baseline.



Gambar 3.8. Flowchart *pemodelan baseling* MNB, SVM, dan RF

Implementasi kode untuk perancangan dan pengujian salah satu *base model Support Vector Machine*.

```
# === Langkah 4: Inisialisasi dan Latih Model SVM Linear ===
TAMPILKAN "Support Vector Machine"
BUAT model SVM dengan kernel linear dan random_state = 42
LATIH model SVM menggunakan data latih (X_train, y_train)

# === Langkah 5: Prediksi dan Evaluasi Model ===
PREDIKSI hasil label menggunakan data uji (X_test)
SIMPAN hasil prediksi ke y_pred

TAMPILKAN "Confusion Matrix"
HITUNG dan TAMPILKAN confusion matrix antara y_test dan y_pred

TAMPILKAN "Classification Report"
HITUNG dan TAMPILKAN classification report (precision, recall, f1-score) antara y_test dan y_pred

HITUNG akurasi model dengan membandingkan y_test dan y_pred
TAMPILKAN akurasi dalam format desimal dan persentase

```

Kode 3.15: Implementasi kode pengujian SVM

Model *Support Vector Machine* (SVM) diimplementasikan dengan kernel linear dan nilai `random_state` sebesar 42 untuk memastikan reprodusibilitas. Model dilatih menggunakan data latih, kemudian dilakukan prediksi terhadap data uji. Evaluasi dilakukan menggunakan *confusion matrix*, *classification report*, serta akurasi model ditampilkan dalam bentuk desimal dan persentase.

Implementasi kode untuk perancangan dan pengujian salah satu *base model Random Forest*

```
# === Inisialisasi dan Latih Model Random Forest ===
TAMPILKAN "Random Forest Classifier"
BUAT model Random Forest dengan:
    - Jumlah pohon (n_estimators) = 100
    - random_state = 42
LATIH model menggunakan data latih (X_train, y_train)

# === Prediksi dan Evaluasi Model ===
PREDIKSI label data uji (X_test)
SIMPAN hasil prediksi ke y_pred_rf

TAMPILKAN "Confusion Matrix"
HITUNG dan TAMPILKAN confusion matrix antara y_test dan y_pred_rf

TAMPILKAN "Classification Report"
HITUNG dan TAMPILKAN classification report (precision, recall, f1-score)

HITUNG akurasi model berdasarkan perbandingan y_test dan y_pred_rf
TAMPILKAN akurasi dalam format desimal dan persentase

```

Kode 3.16: Implementasi kode pengujian Random Forest

Model *Random Forest* dikembangkan menggunakan 100 pohon keputusan dengan parameter `random_state = 42`. Setelah proses pelatihan pada data latih, model memprediksi data uji. Evaluasi dilakukan menggunakan metrik *confusion matrix*, *classification report*, dan akurasi.

Implementasi kode untuk perancangan dan pengujian salah satu *base model*, yaitu *Multinomial Naive Bayes*.

```
# === Inisialisasi dan Latih Model Naive Bayes ===
TAMPILKAN "Multinomial Naive Bayes"
BUAT model Naive Bayes bertipe Multinomial
LATIH model menggunakan data latih (X_train, y_train)

# === Prediksi dan Evaluasi Model ===
PREDIKSI label data uji (X_test)
SIMPAN hasil prediksi ke y_pred_mnb
```

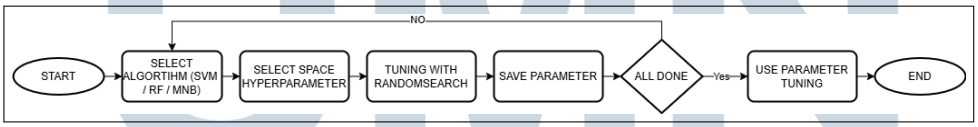
```
9
10      TAMPILKAN "Confusion Matrix"
11      HITUNG dan TAMPILKAN confusion matrix antara y_test dan
        y_pred_mnb
12
13      TAMPILKAN "Classification Report"
14      HITUNG dan TAMPILKAN classification report (precision,
        recall, f1-score)
15
16      HITUNG akurasi model berdasarkan perbandingan y_test dan
        y_pred_mnb
17      TAMPILKAN akurasi dalam format desimal dan persentase
18
```

Kode 3.17: Implementasi kode pengujian Multinomial Naive Bayes

Model *Multinomial Naive Bayes* digunakan sebagai pendekatan probabilistik berbasis frekuensi. Model dilatih pada data latih, lalu memprediksi data uji. Evaluasi dilakukan dengan *confusion matrix*, *classification report*, serta perhitungan akurasi.

5. ***Hyperparameter Tuning***: Dilakukan penyesuaian parameter pada masing-masing *base model* untuk meningkatkan performa dari masing-masing algoritma.

   Pada Gambar 3.9 menunjukkan alur atau *flowchart* dari rencana implementasi penggunaan *hyperparameter tuning*.



Gambar 3.9. Flowchart Hyperparameter Tuning

implementasi kode untuk penggunaan *hyperparameter tuning* terhadap *baseline model SVM*

```
1       IMPORT pustaka: RandomizedSearchCV, SVC dari scikit-learn
2       IMPORT numpy sebagai np
3
4       # === Definisikan Ruang Pencarian Parameter SVM ===
5       BUAT dictionary svm_params
6
7       # === Inisialisasi RandomizedSearchCV ===
```

```
    BUAT objek svm_random

    # === Latih RandomizedSearchCV dengan Data Training ===
    LATIH svm_random menggunakan X_train dan y_train

    # === Tampilkan Hasil Terbaik ===
    TAMPILKAN kombinasi parameter terbaik (best_params_)
    TAMPILKAN skor akurasi validasi silang terbaik (
    best_score_)

```

Kode 3.18: Implementasi hyperperameter tuning SVM

Tuning dilakukan dengan *RandomizedSearchCV* menggunakan ruang pencarian sebagai berikut:

- **C**: Nilai antara `0.01` hingga `100` (logaritmik), untuk mengeksplorasi kekuatan regularisasi dari kuat hingga lemah.
- **kernel**: Menggunakan `linear` dan `rbf`. Kernel linier cocok untuk data TF-IDF berdimensi tinggi, sedangkan RBF digunakan untuk pola nonlinier.
- **gamma**: Hanya untuk RBF, dicoba dengan nilai `'scale'`, `'auto'`, serta angka `0.001` hingga `1`, untuk mengatur pengaruh tiap sampel.
- **probability**: Diset `True` agar model menghasilkan probabilitas, dibutuhkan dalam teknik ensemble *stacking*.

Implementasi kode untuk penggunaan *hyperparameter tuning* terhadap *base model Random Forest*

```
    # === Definisikan Ruang Pencarian Parameter Random Forest
    ===
    BUAT dictionary rf_params

    # === Inisialisasi RandomizedSearchCV untuk Random Forest
    ===
    BUAT objek rf_random
    # === Latih RandomizedSearchCV dengan Data Training ===
    LATIH rf_random menggunakan X_train dan y_train

    # === Tampilkan Hasil Terbaik ===
    TAMPILKAN kombinasi parameter terbaik (best_params_)
    TAMPILKAN skor validasi silang terbaik (best_score_)
```

```
12
```

Kode 3.19: Implementasi hyperparameter tuning random forest

Tuning dilakukan menggunakan *RandomizedSearchCV* untuk mengeksplorasi kombinasi parameter berikut:

- **n_estimators**: Banyaknya pohon dalam hutan, diuji pada nilai `[100, 200, 300, 400]`. Semakin banyak pohon dapat meningkatkan performa, namun berdampak pada waktu komputasi.
- **max_depth**: Kedalaman maksimum pohon, diuji pada `[None, 10, 20, 30]`. Nilai `None` membiarkan pohon tumbuh hingga daun sempurna, sementara nilai lainnya digunakan untuk menghindari overfitting.
- **min_samples_split**: Jumlah minimum sampel yang dibutuhkan untuk memisahkan node, dengan nilai `[2, 5, 10]`. Nilai lebih besar digunakan untuk regularisasi agar pohon tidak terlalu kompleks.
- **min_samples_leaf**: Jumlah minimum sampel pada daun pohon, diuji pada `[1, 2, 4]` untuk mencegah daun dengan sampel terlalu sedikit yang rawan overfitting.
- **bootstrap**: Digunakan untuk menentukan apakah bootstrap sampling digunakan saat membangun pohon. `True` berarti menggunakan sampel acak (bootstrap), `False` menggunakan seluruh data.

Implementasi kode untuk penggunaan *hyperparameter tuning* terhadap *base model Multinomial Naive Bayes*.

```
1   IMPORT MultinomialNB dari scikit-learn
2   IMPORT np untuk membuat rentang nilai alpha
3
4   # === Definisikan Ruang Pencarian Parameter MNB ===
5   BUAT dictionary mnb_params berisi:
6       - alpha: 20 nilai dari 0.1 hingga 2.0 (menggunakan np
    .linspace)
7
8   # === Inisialisasi RandomizedSearchCV untuk MNB ===
9   BUAT objek mnb_random
10
11  # === Latih RandomizedSearchCV dengan Data Training ===
12  LATIH mnb_random menggunakan X_train dan y_train
```

```
13
14      # === Tampilkan Hasil Terbaik ===
15      TAMPILKAN nilai alpha terbaik (best_params_)
16      TAMPILKAN skor akurasi validasi silang terbaik (
        best_score_)
17
```

Kode 3.20: Implementasi hyperparameter tuning naive bayes

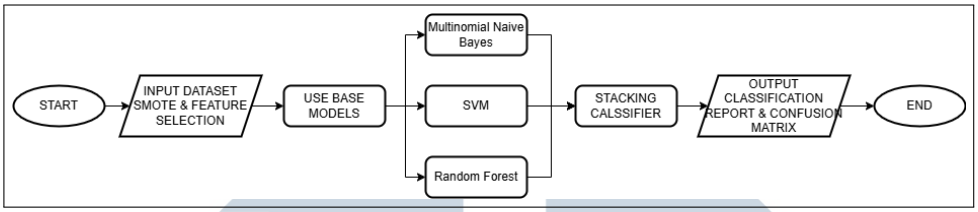
Tuning model Multinomial Naive Bayes dilakukan menggunakan `RandomizedSearchCV` dengan parameter `alpha` sebagai objek pencarian. Parameter `alpha` berperan sebagai smoothing Laplace yang membantu menghindari pembagian nol dalam probabilitas, khususnya pada fitur yang jarang muncul.

- **alpha**: Diuji pada 20 nilai yang dibentuk secara merata dari rentang `0.1` hingga `2.0` menggunakan fungsi `np.linspace`. Nilai `alpha` yang kecil memberikan bobot lebih tinggi terhadap fitur langka, sementara nilai besar cenderung mengurangi sensitivitas terhadap frekuensi kata. Rentang ini dipilih untuk mengeksplorasi keseimbangan antara overfitting dan underfitting dalam konteks data teks.

Proses tuning menggunakan `RandomizedSearchCV` dengan `n_iter = 10`, artinya hanya 10 kombinasi dipilih secara acak dari 20 kemungkinan nilai `alpha` yang tersedia. Jumlah ini dipilih untuk menjaga efisiensi waktu komputasi, mengingat model MNB bersifat ringan dan cepat untuk dilatih. Selain itu, karena hanya satu parameter yang diuji, eksplorasi 10 iterasi sudah dianggap cukup representatif untuk menemukan nilai optimal secara praktis.

6. **Ensemble Learning - Stacking**: Ketiga model individu dikombinasikan menggunakan teknik **Stacking Classifier**, dengan **Logistic Regression** sebagai meta-classifier untuk membuat prediksi akhir.

   Pada Gambar 3.10 menunjukkan alur atau *flowchart* dari pengujian metode *ensemble learning* dengan menggunakan *stacking classifier*. Dengan pendekatan ini, diharapkan performa klasifikasi sentimen meningkat secara signifikan dengan memanfaatkan keunggulan dari masing-masing algoritma dasar.



Gambar 3.10. *Flowchart* model stacking

Implementasi kode untuk melakukan pengujian menggunakan *ensemble learning stacking*.

```
# === Langkah 4: Definisikan Model Hasil Tuning ===
BUAT model SVM

BUAT model Random Forest

BUAT model Multinomial Naive Bayes

# === Langkah 5: Buat Stacking Classifier ===
BUAT objek stacking_clf

# === Langkah 6: Buat Pipeline dengan Seleksi Fitur ===
BUAT pipeline dengan langkah-langkah:
    1. SelectKBest dengan chi2 sebagai score_func dan k = 'all'
    2. classifier: stacking_clf

# === Langkah 7: Latih dan Evaluasi Pipeline ===
LATIH pipeline menggunakan X_train dan y_train

PREDIKSI label data uji (X_test) dan simpan ke y_pred

TAMPILKAN:
    - Confusion Matrix antara y_test dan y_pred
    - Classification Report (precision, recall, f1-score)
    - Accuracy Score (dalam format desimal dan persen)

```

Kode 3.21: Implementasi Kode Penggunaan Stacking Classifier

NUSANTARA

3.2.6 Implementasi dan Pengujian

Tahapan implementasi dan pengujian dilakukan untuk mengevaluasi performa setiap model klasifikasi sentimen yang telah dikembangkan, baik model individual maupun model ensemble. Model utama yang digunakan adalah **Naive Bayes**, **SVM**, **Random Forest**, serta pendekatan **Stacking Classifier**.

Langkah-langkah implementasi dan pengujian:

- **Model Individual:**
  - **Multinomial Naive Bayes (MNB)** digunakan karena cocok untuk data representasi *TF-IDF*. Model ini menggunakan parameter `alpha=0.1`, yang diperoleh melalui proses *hyperparameter tuning*. Nilai alpha yang kecil memungkinkan model lebih sensitif terhadap fitur yang jarang muncul.
  - **Support Vector Machine (SVM)** dengan `kernel='rbf'` dan parameter `C=35.938`. Nilai `C` yang besar memberikan penalti tinggi terhadap kesalahan klasifikasi, sehingga model mencoba memisahkan kelas dengan margin semaksimal mungkin. `probability=True` digunakan agar model mendukung probabilitas prediksi, penting untuk ensemble *stacking*.
  - **Random Forest (RF)** dengan parameter: `n_estimators=100`: jumlah pohon dalam ensemble. `min_samples_split=5`, `min_samples_leaf=1`: untuk mengontrol overfitting, `bootstrap=False`: pohon dibangun tanpa pengambilan sampel bootstrap, `max_depth=None`: kedalaman pohon tidak dibatasi, Parameter tersebut diperoleh dari hasil tuning menggunakan **GridSearchCV**.
- **Feature Selection:**
  - Digunakan **SelectKBest** dengan fungsi `chi2` (Chi-Squared Test), yang memilih fitur berdasarkan korelasinya terhadap label target.
  - Nilai parameter `k='all'` digunakan untuk memasukkan seluruh fitur. Dalam eksperimen tambahan, nilai `k` juga divariasikan (`k=500, 1000, 1500, 2000`) untuk menemukan jumlah fitur optimal terhadap akurasi.
- **Stacking Classifier:**

- **Base Learners**: SVM, Random Forest, dan MultinomialNB.
- **Meta-Classifier**: `Logistic Regression` digunakan untuk menggabungkan prediksi dari base models.
- **Passthrough=True**: Output dari base learners disatukan dengan input asli sebagai fitur tambahan ke meta-classifier.
- **cv=5**: Validasi silang 5-fold digunakan untuk membuat prediksi meta-training yang lebih stabil.

- **Pipeline:**
  - Digunakan `Pipeline()` dari Scikit-Learn agar seluruh proses (feature selection dan model training) dilakukan dalam satu jalur komputasi.
  - Pipeline memungkinkan replikasi eksperimen dan menjaga integritas proses training testing.

- **Evaluasi Model:**
  - Pengujian dilakukan dalam beberapa skenario menggunakan berbagai model klasifikasi, antara lain:
    * **Multinomial Naive Bayes (MNB)**
    * **Support Vector Machine (SVM)**
    * **Random Forest (RF)**
    * **Voting Classifier** (Hard Voting dan Soft Voting)
    * **Stacking Classifier**
  - Seluruh model diuji menggunakan data uji yang telah melalui proses **balancing** dengan **SMOTE** dan **feature selection** menggunakan metode *SelectKBest*.
  - Evaluasi performa model dilakukan menggunakan metrik:
    * **Accuracy**: Persentase prediksi yang benar dari seluruh data uji.
    * **Precision**: Tingkat ketepatan model dalam memprediksi kelas positif.
    * **Recall**: Kemampuan model dalam mengenali semua instance dari kelas tertentu.
    * **F1-Score**: Rata-rata harmonis dari precision dan recall.

    - **Confusion Matrix**: Matriks evaluasi untuk melihat distribusi prediksi benar dan salah pada masing-masing kelas (Positif, Negatif, Netral).
- Hasil evaluasi dibandingkan untuk menentukan metode klasifikasi dengan performa terbaik dalam analisis sentimen terhadap Program Makan Bergizi Gratis.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA