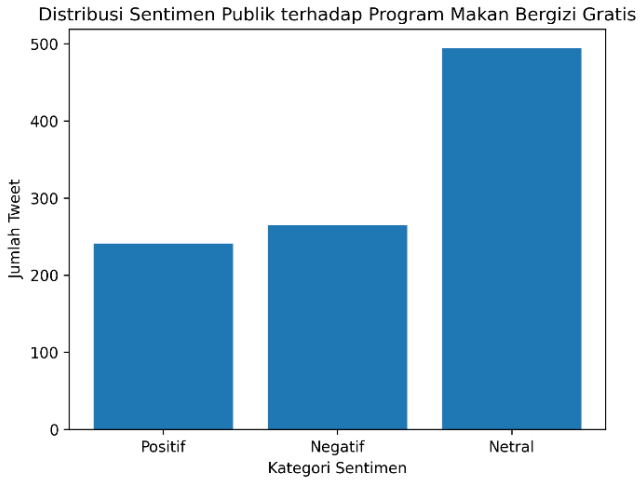
BAB IV
HASIL DAN PEMBAHASAN

4.1 Data Penelitian

Penelitian menggunakan sebanyak 1.000 ulasan data tweet yang dikumpulkan melalui X API v2. Pada tahap implementasi penelitian, periode pengambilan data diperluas hingga 11 Januari 2026 untuk memperoleh jumlah dataset yang lebih representatif dan meningkatkan kualitas analisis sentimen. Dengan demikian, dataset final yang digunakan dalam penelitian ini mencakup periode 6 Januari 2025 sampai 11 Januari 2026. Setelah data tweet terkumpul dilakukan pelabelan data, pelabelan data dilakukan secara manual (human annotation). Seluruh tahapan pemrosesan data pada dataset utama mengikuti metode yang telah dijelaskan pada Bab III, yaitu pelabelan sentimen, preprocessing, ekstraksi fitur TF-IDF, dan pengujian menggunakan algoritma SVM dan Naïve Bayes.



Gambar 1 Distribusi Sentimen Publik

Gambar 4.1 Berdasarkan grafik distribusi sentimen di atas, dari total 1.000 data tweet yang dianalisis, sentimen netral mendominasi dengan jumlah 494 data (49,4%). Sentimen negatif berjumlah 265 data (26,5%), sedangkan sentimen positif berjumlah 241 data (24,1%).

Tabel 4.1 Sampel Data Tweet

No	Tweet	Label Sentimen
1	Momen Prabowo Tanda Tangani Sepatu Siswa di Bogor saat Tinjau Makan Bergizi Gratis	Netral
2	Semoga program makan bergizi gratis dri MDA tetap berlanjut walaupun programnya sma dengan pemerintah	Positif
3	Pemprov Babel dan DPRD Bahas Anggaran Program Makan Bergizi Gratis di APBD 2025	Netral
4	Menurutku berhasil skliki programnya MDA yg makan bergizi gratis. Buktinya terjadi penurunan angka stunting di daerah tersebut	Positif
5	#SahabatBahari Kementerian Kelautan dan Perikanan (KKP) tancap gas merealisasikan program revitalisasi tambak Pantura Jawa untuk mendukung pencapaian target swasembada pangan serta mendukung pelaksanaan program makan bergizi gratis (MBG). #25TahunKKP #KKPGOID #MENTERIKKP	Netral
6	Adapun peruntukkan penghematan uang kembali Prabowo tegaskan guna merealisasikan program yang langsung mengena ke masyarakat seperti Makan Bergizi Gratis dan pembangunan sekolah.	Netral
7	@prabowo T***l . Bergizi apaan . Itu program makan gratis bersyukur	Negatif

8	Pangdam IV/Diponegoro Mayjen TNI Deddy Suryadi meninjau program Makan Bergizi Gratis (MBG) di Kota Semarang pada Senin 10 Februari 2025. #suaramerdeka #suaramerdekacom #pangdam #pangdam4diponegoro #kotasemarang #semarang #programmakanbergizigratis	Netral
9	,MBG : makan bergizi gratis MBG : makan burger grati	Negatif
10	Peneliti LPEM UI: Tanpa Pengawasan Ketat Program Makan Bergizi Gratis Rentan terhadap Inefisiensi Anggaran	Netral
…	…	…
1000	Makan Siang Bergizi Gratis Sudah Berjalan Apakah Stok Pangan Cukup untuk Memenuhinya? Saksikan selengkapnya hanya di YouTube Garuda TV #garudatv #zulkiflihasan #orangpenting #makanbergizigratis	Netral

Keterangan : Tabel 4.1 diatas merupakan beberapa contoh dari 1.000 data tweet yang sudah diberi label secara manual (human annotation)

4.2 Preprocessing Text

Tahap pra-pemrosesan teks (*text preprocessing*) merupakan langkah awal yang dilakukan untuk membersihkan dan menyiapkan data teks agar dapat diolah lebih lanjut pada tahap representasi fitur dan pemodelan klasifikasi. Data teks yang diperoleh dari media sosial umumnya masih mengandung berbagai unsur yang tidak relevan, seperti tanda baca, simbol, kata tidak baku, serta kata-kata umum yang tidak memiliki makna sentimen. Oleh karena itu, diperlukan proses pra-pemrosesan untuk meningkatkan kualitas data dan akurasi model klasifikasi.

Proses pra-pemrosesan teks dalam penelitian ini dilakukan melalui beberapa tahapan, yaitu *case folding*, *cleaning*, *tokenizing*, *stopword removal*, dan *stemming*.

Setiap tahapan memiliki peran penting dalam mengubah data teks mentah menjadi bentuk yang lebih terstruktur dan siap digunakan sebagai masukan pada metode *machine learning*. Implementasi dari tahap *cleaning* dapat dilihat pada gambar 4.2

```python
import re
import pandas as pd

def text_cleaning(text):
    # pastikan tipe string
    text = str(text)

    # hapus URL
    text = re.sub(r"http\S+|www\S+", "", text)

    # hapus mention (@username)
    text = re.sub(r"@\w+", "", text)

    # hapus hashtag (#hashtag -> hapus seluruhnya)
    text = re.sub(r"#\w+", "", text)

    # hapus angka
    text = re.sub(r"\d+", "", text)


    # hapus emoji dan simbol
    text = re.sub(r"[^\w\s]", "", text)

    # hapus spasi berlebih
    text = re.sub(r"\s+", " ", text).strip()

    return text

# load data
df = pd.read_excel("fold_10_test.xlsx")

# kolom teks
TEXT_COL = "Teks"

# terapkan text cleaning
df["clean_text"] = df[TEXT_COL].apply(text_cleaning)

# simpan hasil
df.to_excel("HASIL_TEXT_CLEANING.xlsx", index=False)
```

Gambar 2 Code Python Proses Cleaning

Tabel 4.2 Hasil Cleaning

Sebelum Cleaning	Sesudah Cleaning
,MBG : makan bergizi gratis MBG : makan burger gratis	MBG makan bergizi gratis MBG makan burger gratis

Tahap *cleaning* bertujuan untuk menghapus karakter yang tidak diperlukan dalam teks, seperti tanda baca, angka, simbol khusus, *URL*, *mention*, dan *hashtag*. Proses ini dilakukan agar teks hanya berisi kata-kata yang relevan dan dapat memberikan kontribusi terhadap analisis sentimen. Seperti pada tabel 4.2 sesudah proses cleaning tanda baca seperti koma ataupun titik dua sudah bersih.

Setelah proses cleaning tahap kedua adalah *case folding* dilakukan dengan mengubah seluruh huruf pada teks menjadi huruf kecil (lowercase). Hasil case folding dapat dilihat pada tabel 4.3.

```
def case_folding(text):
    # pastikan tipe data string
    text = str(text)

    # ubah seluruh huruf menjadi lowercase
    text = text.lower()

    return text

    import pandas as pd

# load data
df = pd.read_excel("cleaning_teks.xlsx")

TEXT_COL = "clean_text"

# terapkan case folding
df["case_folding"] = df[TEXT_COL].apply(case_folding)
```

Gambar 3 Code Python Proses Case Folding

Table 4.3 Hasil Case Folding

Sebelum Case Folding	Sesudah Case Folding
Momen Prabowo Tanda Tangani Sepatu Siswa di Bogor saat Tinjau Makan Bergizi Gratis	momen prabowo tanda tangani sepatu siswa di bogor saat tinjau makan bergizi gratis

Tujuan dari tahap *case folding* adalah untuk menghindari perbedaan makna akibat perbedaan penggunaan huruf besar dan huruf kecil, sehingga kata yang

memiliki bentuk sama tetapi berbeda kapitalisasi akan dianggap sebagai satu kata yang sama. Hal ini untuk memastikan konsistensi dengan membuat kata-kata seperti “Momen” dan “momen” sebagai hal yang sama terlepas dari penggunaan huruf kapital. Hal ini menyederhanakan analisis dengan berfokus pada arti kata, bukan pada variasi kapitalisasinya.

Tahap ketiga adalah tokenizing, tokenizing merupakan proses pemecahan teks menjadi unit-unit kata yang lebih kecil yang disebut sebagai token. Tahap ini teks dibagi menjadi unit-unit dasar yang disebut “token.” Token ini dapat berupa kata individual memungkinkan sistem untuk mengenali setiap kata secara terpisah sehingga dapat diolah lebih lanjut pada proses representasi fitur. Hasil tokenizing dapat dilihat pada tabel 4.4

```
import nltk
from nltk.tokenize import word_tokenize

# download tokenizer (cukup sekali)
nltk.download('punkt_tab')

def tokenization(text):
    # pastikan input string
    text = str(text)

    # tokenisasi kata
    tokens = word_tokenize(text)

    return tokens
import pandas as pd

# load data
df = pd.read_excel("case_folding.xlsx")

TEXT_COL = "case_folding"

# terapkan tokenization
df["tokenized_text"] = df[TEXT_COL].apply(tokenization)

# simpan hasil
df.to_excel("HASIL_TOKENIZATION.xlsx", index=False)
```

Gambar 4 Code Python Proses Tokenizing

Tabel 4.4 Hasil Tokenizing

Sebelum Tokenizing	Sesudah Tokenizing
momen prabowo tanda tangani sepatu siswa di bogor saat tinjau makan bergizi gratis	['momen', 'prabowo', 'tanda', 'tangani', 'sepatu', 'siswa', 'di', 'bogor', 'saat', 'tinjau', 'makan', 'bergizi', 'gratis']

Tahap keempat *stopword removal* dilakukan dengan menghapus kata-kata umum yang sering muncul dalam teks tetapi tidak memiliki pengaruh signifikan terhadap penentuan sentimen, seperti kata “dan”, “yang”, “di”, dan “ke”. Penghapusan stopword bertujuan untuk mengurangi dimensi data serta meningkatkan fokus model terhadap kata-kata yang memiliki makna sentimen. Hasil stopword removal dapat dilihat pada tabel 4.5 dan implementasi stopword removal dapat dilihat pada gambar 4.5

```python
import nltk
import ast
from nltk.corpus import stopwords

# download stopword indonesia (cukup sekali)
nltk.download('stopwords')

# stopword bahasa indonesia
stopword_id = set(stopwords.words('indonesian'))

def stopword_removal(tokens):
  # JIKA token masih string, ubah ke list
    if isinstance(tokens, str):
        tokens = ast.literal_eval(tokens)

  # hapus stopword
  filtered_tokens = [word for word in tokens if word not in stopword_id]

  return filtered_tokens

import pandas as pd

# load data
df = pd.read_excel("tokenisasi.xlsx")

TEXT_COL = "tokenized_text"

# terapkan stopword removal
df["stopword_removed"] = df[TEXT_COL].apply(stopword_removal)

# simpan hasil
df.to_excel("HASIL_STOPWORD_REMOVAL.xlsx", index=False)
```

Gambar 4.5 Code python Stopword Removal

Tabel 4.5 Hasil Stopword Removal

Sebelum Stopword Removal	Sesudah Stopword Removal
['momen', 'prabowo', 'tanda', 'tangani', 'sepatu', 'siswa', 'di', 'bogor', 'saat', 'tinjau', 'makan', 'bergizi', 'gratis']	['momen', 'prabowo', 'tanda', 'tangani', 'sepatu', 'siswa', 'bogor', 'tinjau', 'makan', 'bergizi', 'gratis']

Proses kelima adalah stemming, Stemming memperpendek kata dengan menghilangkan awalan dan akhiran, sehingga menghasilkan bentuk kata yang lebih umum. Ini membantu mengidentifikasi kata-kata dengan akar kata yang sama sebagai padanannya. Proses ini dilakukan agar variasi kata yang memiliki makna yang sama dapat direpresentasikan dalam satu bentuk dasar, sehingga membantu meningkatkan konsistensi data dan efektivitas proses klasifikasi. Hasil stemming dapat dilihat pada tabel 4.6 dan implementasi stemming dapat dilihat pada gambar 4.6

```
!pip install Sastrawi
import ast
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

# inisialisasi stemmer
factory = StemmerFactory()
stemmer = factory.create_stemmer()

def stemming_text(tokens):
    # jika token masih berbentuk string → ubah ke list
    if isinstance(tokens, str):
        tokens = ast.literal_eval(tokens)

    # lakukan stemming per kata
    stemmed_tokens = [stemmer.stem(word) for word in tokens]

    return stemmed_tokens

import pandas as pd

df = pd.read_excel("stopword_removal.xlsx")

TEXT_COL = "stopword_removed"

df["stemming"] = df[TEXT_COL].apply(stemming_text)

df.to_excel("HASIL_STEMMING.xlsx", index=False)
```

Gambar 4.6 Code python proses Stemming

Tabel 4.6 Hasil Stemming

Sebelum Stemming	Sebelum Stemming
['momen', 'prabowo', 'tanda', 'tangani', 'sepatu', 'siswa', 'bogor', 'tinjau', 'makan', 'bergizi', 'gratis']	['momen', 'prabowo', 'tanda', 'tangan', 'sepatu', 'siswa', 'bogor', 'tinjau', 'makan', 'gizi', 'gratis']

4.2.1 Pembobotan Kata TF-IDF

Pembobotan kata menggunakan Term Frequency-Inverse Document Frequency (TF-IDF) adalah salah satu metode yang umum digunakan dalam analisis teks untuk memberikan bobot pada kata-kata dalam dokumen. Metode ini membantu dalam mengukur seberapa penting suatu kata dalam dokumen atau korpus dibandingkan dengan dokumen lainnya. Pembobotan TF-IDF digunakan untuk mengidentifikasi kata-kata yang relevan atau signifikan dalam suatu teks yang akan digunakan dalam model machine learning.

Dalam penelitian ini, proses pembobotan dilakukan dengan menggunakan library TfidfVectorizer dari Python. Library ini mempermudah proses perhitungan dengan secara otomatis mengonversi teks menjadi representasi numerik berdasarkan nilai TF-IDF, sehingga setiap kata dalam dokumen mendapatkan bobot yang sesuai. Sebagai contoh, dalam analisis sentimen terhadap Program Makan Bergizi Gratis, kata-kata seperti "gizi", "sekolah", "makanan", dan "gratis" kemungkinan memiliki nilai TF-IDF yang lebih tinggi dibandingkan dengan kata-kata umum seperti "dan", "dengan", atau "di". Dengan demikian, kata-kata tersebut akan memiliki bobot yang lebih besar dalam membentuk representasi

numerik dari teks yang akan dianalisis. Proses pembobotan menggunakan library TfidfVectorizer dari Python dapat dilihat pada gambar 4.7

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Membaca dataset dari file Excel
file_path = 'stemming.xlsx'  # Ganti dengan path file Excel Anda
df = pd.read_excel(file_path)

# Misalnya, kolom yang sudah diproses teksnya adalah 'processed_text'
# Pastikan kolom yang digunakan sesuai dengan dataset Anda
# Melakukan TF-IDF Vektorisasi
vectorizer = TfidfVectorizer(max_features=500, min_df=1, max_df=0.95)

# Vektorisasi menggunakan TF-IDF pada data yang sudah diproses
X_tfidf = vectorizer.fit_transform(df['stemming'])


# Menyimpan hasil TF-IDF ke dalam DataFrame
tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=vectorizer.get_feature_names_out())

# Menyimpan DataFrame ke dalam file Excel
output_file = 'tfidf_output.xlsx'  # Nama file Excel yang ingin disimpan
tfidf_df.to_excel(output_file, index=False)

print(f"TF-IDF results have been saved to {output_file}")
```

Gambar 4.7 Code Python Proses Pembobotan TF-IDF

Pertama, library pandas diimpor untuk mengelola data dan TfidfVectorizer dari sklearn untuk proses ekstraksi fitur teks. Dataset kemudian dibaca dari file Excel 'stemming.xlsx' menggunakan pd.read_excel() dan disimpan dalam DataFrame df. Selanjutnya, objek TfidfVectorizer diinisialisasi dengan parameter max_features=500 untuk membatasi jumlah kata maksimal, min_df=1 untuk menyertakan kata yang muncul minimal satu kali, dan max_df=0.95 untuk mengabaikan kata yang terlalu sering muncul. Proses vektorisasi dilakukan dengan fit_transform() pada kolom 'stemming', sehingga teks diubah menjadi matriks numerik TF-IDF. Hasil matriks tersebut kemudian dikonversi ke bentuk array dan

dijadikan DataFrame dengan nama kolom berupa fitur kata menggunakan get_feature_names_out(). Setelah itu, DataFrame disimpan ke dalam file Excel 'tfidf_output.xlsx' tanpa menyertakan indeks, dan terakhir program menampilkan pesan bahwa hasil TF-IDF telah berhasil disimpan.

Pada implementasi dalam penelitian ini, TF-IDF digunakan untuk mengonversi cuitan dari media sosial X terkait dengan Program Makan Bergizi Gratis menjadi vektor fitur yang dapat diproses oleh algoritma Support Vector Machine (SVM) dan Naïve Bayes (NB). Hasil pembobotan ini kemudian digunakan untuk membangun model klasifikasi sentimen yang lebih efisien dan akurat. Hasil pembobotan berjumlah 500 fitur unik, berikut adalah beberapa bentuk matriks TF-IDF:

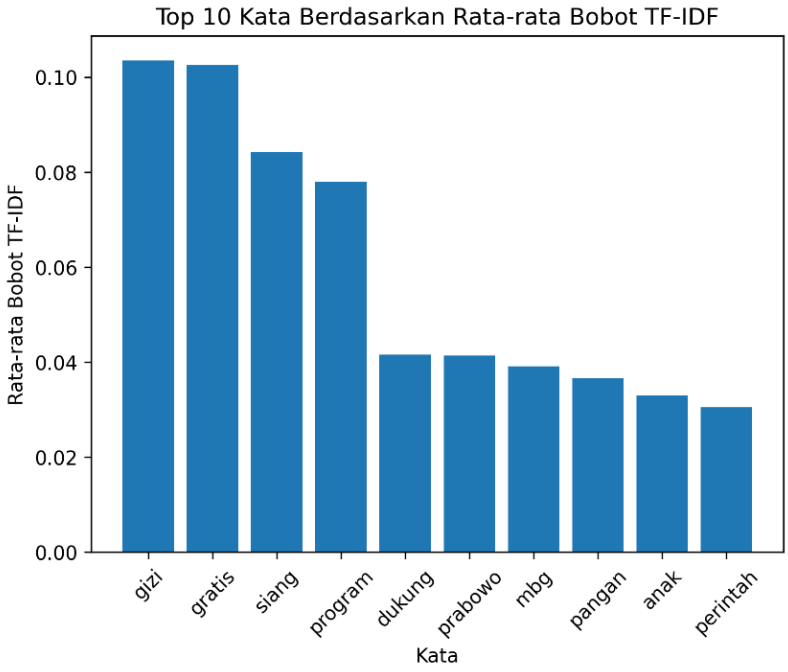
Tabel 4.7 Matriks TF-IDF

No	gizi	mbg	makan	program	gratis	makan gizi
1	0,088323	0,071617	0,085941	0	0	0
2	0	0,053778	0	0	0	0
3	0,07238	0,058689	0,070428	0	0,086244	0,089025
4	0,230622	0	0,074801	0,072812	0,091599	0,094552
5	0	0,077887	0	0	0	0
6	0	0,328039	0	0,191593	0	0
7	0	0	0,29925434	0,1456493	0,3664597	0
8	0	0,066082	0	0,0771907	0	0
9	0	0,1082	0	0,12639	0	0
10	0,24227	0	0	0,2294663	0	0
…						
1.000	0,15449	0	0,15032334	0,146327	0,1840823	0,1900173

Keterangan : Kolom (gizi, mbg, makan, program, gratis, makan gizi) menunjukkan fitur atau kata kunci yang digunakan dalam proses ekstraksi fitur. Nilai desimal pada setiap sel merupakan bobot TF-IDF dari suatu kata dalam dokumen tertentu.

Nilai 0 menunjukkan kata tersebut tidak muncul dalam dokumen, semakin besar nilai TF-IDF, maka semakin penting kata tersebut dalam dokumen tersebut.

Berdasarkan hasil perhitungan rata-rata bobot TF-IDF pada seluruh dokumen, diperoleh beberapa kata dengan nilai tertinggi yang dapat dikategorikan sebagai kata dominan dalam dataset. Kata-kata tersebut memiliki tingkat kepentingan yang lebih besar dibandingkan kata lainnya karena tidak hanya sering muncul dalam dokumen tertentu, tetapi juga memiliki daya pembeda yang kuat antar dokumen. Berdasarkan rata-rata bobot TF-IDF tertinggi, diperoleh beberapa kata dominan dalam dataset pada gambar 4.7.



Gambar 4.8 Top 10 Kata Dominan

Pada gambar diatas menunjukkan topik utama yang paling sering dibahas dalam opini publik terkait Program Makan Bergizi Gratis.

Kata dengan bobot tinggi tidak selalu kata yang paling sering muncul, tetapi kata yang memiliki tingkat diskriminatif tinggi antar dokumen.

4.3 Proses Klasifikasi Data

Proses klasifikasi data adalah tahap penting dalam analisis sentimen yang bertujuan untuk mengelompokkan teks (dalam hal ini, cuitan dari media sosial X) ke dalam kategori sentimen yang telah ditentukan, yaitu positif, negatif, atau netral. Pada penelitian ini, proses klasifikasi data dilakukan menggunakan dua algoritma machine learning, yaitu Support Vector Machine (SVM) dan Naïve Bayes (NB), yang telah dipilih berdasarkan kinerja keduanya dalam mengklasifikasikan teks. Proses klasifikasi data terdiri dari beberapa tahapan penting yang meliputi pengolahan data, pelatihan model, dan evaluasi hasil klasifikasi.

4.3.1 Pembagian Data

Sebelum tahap pengklasifikasian model harus membagi membagi dataset menjadi dua bagian data latih (training data) dan data uji (testing data). Data latih digunakan untuk melatih model machine learning, sementara data uji digunakan untuk menguji seberapa baik model tersebut dalam mengklasifikasikan sentimen pada data yang belum pernah dilihat sebelumnya.

Pada penelitian ini, data dibagi menggunakan metode k-fold cross-validation dengan nilai k = 10. Metode ini membagi dataset menjadi sepuluh bagian (*fold*) yang berukuran sama. Pada setiap iterasi, sembilan fold digunakan sebagai data latih dan satu fold digunakan sebagai data uji, kemudian proses ini diulang sebanyak sepuluh kali hingga setiap fold berperan sebagai data uji satu kali. Hasil evaluasi dari seluruh iterasi kemudian dirata-ratakan untuk memperoleh kinerja model yang lebih stabil dan objektif. Penggunaan 10-Fold Cross-Validation bertujuan untuk meminimalkan bias pembagian data, meningkatkan pemanfaatan

data secara menyeluruh, serta mengurangi risiko terjadinya *overfitting* pada model klasifikasi sentimen yang dibangun.

Setelah pembagian data dilakukan, proses pelatihan model diterapkan pada masing-masing fold menggunakan dua algoritma klasifikasi, yaitu Support Vector Machine (SVM) dan Naïve Bayes (NB). Model Support Vector Machine (SVM) diimplementasikan menggunakan kernel linear dengan parameter regularisasi $C = 1.0$. Kernel linear dipilih karena data teks hasil representasi TF-IDF memiliki dimensi tinggi sehingga pemisahan kelas secara linear dinilai lebih efektif dan efisien. Sementara itu, model Naïve Bayes yang digunakan adalah Multinomial Naïve Bayes dengan parameter smoothing (alpha) sebesar 1.0. Parameter smoothing diterapkan untuk menghindari probabilitas nol pada fitur yang tidak muncul dalam data latih.

Parameter yang digunakan dalam penelitian ini merupakan konfigurasi standar yang umum diterapkan pada klasifikasi teks berbasis TF-IDF dan tidak dilakukan proses hyperparameter tuning lanjutan. Dengan skema ini, setiap model dilatih dan diuji secara bergantian pada seluruh bagian data, sehingga hasil evaluasi yang diperoleh lebih stabil dan representatif terhadap performa model dalam mengklasifikasikan sentimen publik.

4.3.2 Implementasi Model Klasifikasi *Support Vector Machine* (SVM)

1. Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score
import seaborn as sns
```

Gambar 4.9 Code Library Python

Dalam penelitian ini, pustaka pandas digunakan untuk memproses dan mengelola data dalam bentuk *DataFrame* yang umumnya berasal dari file tabular seperti CSV atau Excel, sedangkan matplotlib dan seaborn dimanfaatkan untuk melakukan visualisasi data serta menampilkan hasil evaluasi model secara grafis. Model klasifikasi yang digunakan adalah SVC (Support Vector Classifier) yang merupakan implementasi algoritma Support Vector Machine dari pustaka *scikit-learn*.

Untuk memastikan evaluasi model yang lebih objektif, digunakan metode K-Fold Cross Validation yang membagi data menjadi beberapa bagian untuk proses pelatihan dan pengujian secara bergantian. Selanjutnya, kinerja model diukur menggunakan beberapa metrik evaluasi yaitu accuracy_score, confusion_matrix, classification_report, dan roc_auc_score guna menilai tingkat akurasi, performa klasifikasi tiap kelas, serta kemampuan model dalam membedakan kelas secara keseluruhan.

2. Membaca Dataset

```
# Membaca dataset
train_df = pd.read_excel('/content/TFIDF.xlsx')  # Dataset training
test_df = pd.read_excel('/content/tfidf_output.xlsx')  # Dataset testing
```

Gambar 4.10 Code Read Dataset

Dataset dibaca menggunakan fungsi read_excel dari pandas. Di sini, dua file Excel diimpor: satu untuk data pelatihan (TFIDF.xlsx) dan satu lagi untuk data pengujian (tfidf_output.xlsx).

3. Menentukan Fitur dan Label

```
# Misalkan data terdiri dari fitur dan label
X_train = train_df.iloc[:, :-1]  # Semua kolom kecuali terakhir sebagai fitur
y_train = train_df.iloc[:, -1]   # Kolom terakhir sebagai label

X_test = test_df.iloc[:, :-1]    # Semua kolom kecuali terakhir sebagai fitur
y_test = test_df.iloc[:, -1]     # Kolom terakhir sebagai label
```

Gambar 4.11 Code Penentuan Fitur dan Tabel

Pada tahap ini Kolom terakhir dari dataset digunakan sebagai label target (y_train, y_test), sementara sisanya digunakan sebagai fitur (X_train, X_test)

4. Inisialisasi KFold

```
# Inisialisasi KFold untuk 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

Gambar 4.12 Code Inisialisasi K-fold Cross Validation

Pada tahap ini, Membagi data pelatihan menjadi 10 *fold* untuk cross-validation, dengan pengacakan data (shuffle) untuk memastikan pembagian yang adil.

5. Melakukan K-Fold Cross Validation

```
# Melakukan KFold Cross Validation
for train_index, val_index in kf.split(X_train):
    # Membagi data training menjadi subset latih dan validasi
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    # Inisialisasi model SVM dengan kernel linear
    model = SVC(kernel='linear', probability=True)  # probability=True untuk perhitungan ROC-AUC

    # Melatih model
    model.fit(X_train_fold, y_train_fold)

    # Melakukan prediksi pada data validasi
    y_pred = model.predict(X_val_fold)

    # Menghitung akurasi dan menyimpannya
    accuracy = accuracy_score(y_val_fold, y_pred)
    accuracies.append(accuracy)
```

```
# Menghitung precision, recall, f1-score dan ROC-AUC
report = classification_report(y_val_fold, y_pred, output_dict=True)
precision_scores.append(report['accuracy'])  # Precision for accuracy
recall_scores.append(report['accuracy'])     # Recall for accuracy
f1_scores.append(report['accuracy'])         # F1-Score for accuracy
# Note: roc_auc_score requires binary or multi-class labels. Ensure `y_val_fold` has at least two classes.
# If it's possible for a fold to have only one class, handle that case (e.g., skip ROC-AUC for that fold).
try:
    roc_auc_scores.append(roc_auc_score(y_val_fold, model.predict_proba(X_val_fold), multi_class='ovr'))
except ValueError as e:
    print(f"Could not calculate ROC-AUC for a fold due to: {e}")
    roc_auc_scores.append(float('nan')) # Append NaN or skip if preferred
```

Gambar 4.13 Code Proses K-Fold Cross Validation

Pada proses cross-validation, setiap *fold* membagi dataset menjadi data pelatihan (X_train_fold, y_train_fold) dan data validasi (X_val_fold, y_val_fold), kemudian model SVM dilatih menggunakan data pelatihan dan diuji pada data validasi. Selanjutnya, kinerja model dievaluasi dengan menghitung nilai accuracy, precision, recall, F1-score, dan ROC-AUC, di mana seluruh hasil evaluasi tersebut dicatat dan disimpan pada setiap *fold* untuk kemudian dianalisis sebagai gambaran performa model secara keseluruhan.

6. Menampilkan Evaluasi Model *Support Vector Machine* (SVM)

```
# Menampilkan akurasi setiap fold
for i, accuracy in enumerate(accuracies, 1):
    print(f"Fold {i}: {accuracy:.4f}")

# Menampilkan rata-rata akurasi dari 10 fold
average_accuracy = pd.Series(accuracies).mean()
average_precision = pd.Series(precision_scores).mean()
average_recall = pd.Series(recall_scores).mean()
average_f1 = pd.Series(f1_scores).mean()
average_roc_auc = pd.Series(roc_auc_scores).mean()

print(f"\nRata-rata akurasi: {average_accuracy:.4f}")
print(f"Rata-rata Precision: {average_precision:.4f}")
print(f"Rata-rata Recall: {average_recall:.4f}")
print(f"Rata-rata F1-Score: {average_f1:.4f}")
print(f"Rata-rata ROC-AUC: {average_roc_auc:.4f}")
```

Gambar 4.14 Code Model Evaluasi SVM

Hasil evaluasi ditampilkan untuk setiap *fold* pada proses cross-validation, kemudian dihitung dan disajikan nilai rata-rata dari metrik akurasi, precision, recall, F1-score, dan ROC-AUC pada seluruh *fold* untuk memperoleh gambaran kinerja model secara menyeluruh dan lebih objektif.

7. Proses Untuk Menampilkan Visualisasi Hasil Akurasi

```
# Visualisasi hasil akurasi setiap fold dan rata-rata akurasi
plt.figure(figsize=(8,4))

# Plot akurasi per fold
plt.bar(range(1, 11), accuracies, color='blue', alpha=0.7, label='Akurasi per Fold')

# Plot garis rata-rata akurasi
plt.axhline(y=average_accuracy, color='red', linestyle='--', label=f'Rata-rata Akurasi: {average_accuracy:.4f}')

# Menambahkan label dan judul
plt.xlabel('Fold')
plt.ylabel('Akurasi')
plt.title('Visualisasi Akurasi SVM pada K-Fold Cross Validation')
plt.xticks(range(1, 11))  # Menampilkan angka fold dari 1 hingga 10
plt.legend()

# Menampilkan grafik
plt.show()
```

Gambar 4.15 Code Visualisasi Hasil Akurasi

Hasil akurasi pada setiap *fold* divisualisasikan dalam bentuk grafik untuk menunjukkan perbandingan nilai antar *fold*, kemudian ditambahkan garis horizontal yang merepresentasikan rata-rata akurasi sebagai acuan untuk melihat kestabilan dan konsistensi kinerja model secara keseluruhan.

8. Pelatihan Model Pada Seluruh Dataset

```
# Melatih model SVM pada seluruh dataset training dan menguji pada dataset testing
final_model = SVC(kernel='linear', probability=True)  # probability=True untuk ROC-AUC
final_model.fit(X_train, y_train)

# Melakukan prediksi pada data testing
y_pred_test = final_model.predict(X_test)

# Menghitung akurasi pada data testing
test_accuracy = accuracy_score(y_test, y_pred_test)
print(f"\nAkurasi pada data testing: {test_accuracy:.4f}")
```

Gambar 4.16 Code Untuk Melatih Model SVM Pada Dataset

Model SVM dilatih menggunakan seluruh data pelatihan (X_train, y_train), kemudian dilakukan pengujian pada data pengujian (X_test) untuk menghasilkan prediksi yang selanjutnya digunakan dalam menghitung nilai akurasi sebagai ukuran kinerja model terhadap data yang belum pernah dilihat sebelumnya.

9. Confusion Matrix

```
# Menampilkan confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_test)
print("\nConfusion Matrix:")
print(conf_matrix)

# Visualisasi confusion matrix
plt.figure(figsize=(4,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Positif', 'Negatif', 'Netral'], yticklabels=['Positif',
plt.title('Confusion Matrix for SVM')
plt.xlabel('Prediksi')
plt.ylabel('Aktual')
plt.show()
```

Gambar 4.17 Code Confusion Matrix SVM

Confusion matrix dihitung untuk mengetahui perbandingan antara label aktual dan hasil prediksi model, kemudian ditampilkan serta divisualisasikan dalam bentuk *heatmap* agar pola kesalahan dan tingkat ketepatan klasifikasi pada setiap kelas dapat lebih mudah dipahami.

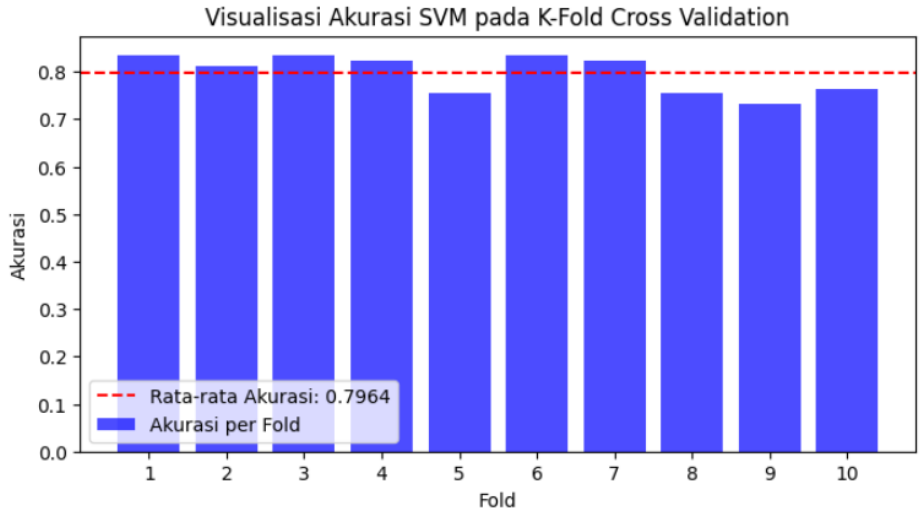
10. Classification Report untuk Data Testing

```
# Menampilkan classification report untuk data testing
print("\nClassification Report untuk Data Testing:")
print(classification_report(y_test, y_pred_test))
```

Gambar 4.18 Code Untuk Menampilkan Classification Report

Classification report ditampilkan untuk data pengujian guna menyajikan nilai precision, recall, F1-score, dan support pada setiap kelas, sehingga kinerja model dalam mengklasifikasikan masing-masing kategori dapat dianalisis secara lebih rinci dan komprehensif.

11. Hasil Evaluasi dengan K-Fold Cross Validation



Gambar 4.19 Visualisasi Akurasi SVM pada K-Fold Cross Validation

Gambar tersebut merupakan visualisasi hasil evaluasi SVM dengan K-Fold Cross Validation. Untuk lebih jelasnya hasil dari 10 fold dapat dilihat pada gambar 4.18.

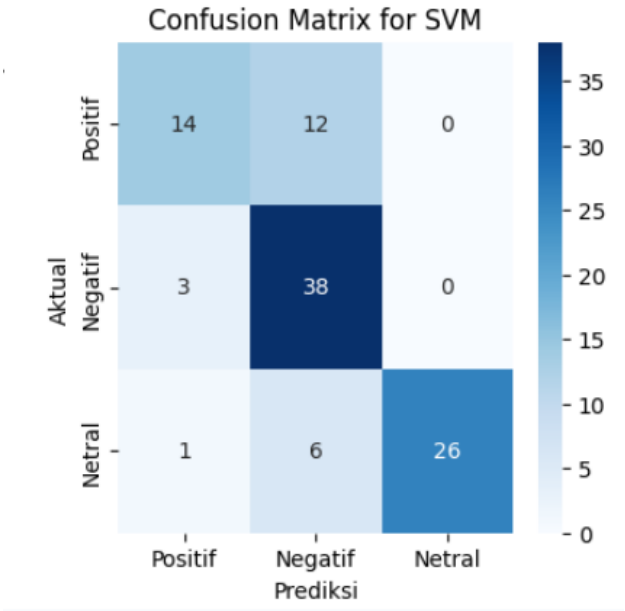
```
Fold 1: 0.8333
Fold 2: 0.8111
Fold 3: 0.8333
Fold 4: 0.8222
Fold 5: 0.7556
Fold 6: 0.8333
Fold 7: 0.8222
Fold 8: 0.7556
Fold 9: 0.7333
Fold 10: 0.7640
```

Gambar 4.20 Hasil Evaluasi K-Fold Cross Validation SVM

Pada tahap pertama, model SVM diuji menggunakan K-Fold Cross Validation dengan 10 *fold*. Dari hasil ini, dapat dilihat bahwa akurasi model SVM cenderung stabil di sebagian besar fold dengan variasi yang tidak terlalu besar. Fold 1 hingga Fold 4 menunjukkan akurasi yang cukup tinggi, sekitar 80% atau lebih, namun pada Fold 9 dan Fold 5, akurasi menurun sedikit, masing-masing mencapai 73,33% dan 75,56%.

Rata-rata akurasi untuk 10 fold adalah 0.7964, yang menunjukkan bahwa model memiliki kemampuan yang baik dalam melakukan klasifikasi berdasarkan data pelatihan. Rata-rata precision, recall, dan F1-Score juga tercatat sebesar 0.7964, yang mengindikasikan bahwa model memberikan performa yang serupa untuk setiap metrik evaluasi.

12. Hasil Evaluasi Model pada Data Uji



Gambar 4.21 Confusion Matrix SVM

Gambar tersebut merupakan hasil dari confusion matrix SVM yang diperoleh, baris pertama menunjukkan prediksi untuk kelas Negatif. Terdapat 14 data yang diprediksi dengan benar sebagai Negatif, namun ada 12 data yang salah diprediksi sebagai Netral. Baris kedua menunjukkan prediksi untuk kelas Netral. Model memprediksi dengan benar 38 data sebagai Netral, namun ada 3 data yang salah diprediksi sebagai Negatif. Baris ketiga menunjukkan prediksi untuk kelas Positif. Model memprediksi dengan benar 26 data sebagai Positif, meskipun ada 1 data yang salah diprediksi sebagai Negatif dan 6 data yang salah diprediksi sebagai Netral.

Kemudian, berikut ini adalah *classification report* yang mencakup accuracy precision, recall, dan F1-score.

Classification Report untuk Data Testing:

	precision	recall	f1-score	support
Negatif	0.78	0.54	0.64	26
Netral	0.68	0.93	0.78	41
Positif	1.00	0.79	0.88	33
accuracy			0.78	100
macro avg	0.82	0.75	0.77	100
weighted avg	0.81	0.78	0.78	100

Gambar 4.22 Classification Report SVM

Model menghasilkan akurasi sebesar 78% pada data pengujian, yang menunjukkan bahwa model cukup baik dalam mengklasifikasikan data yang belum pernah dilihat sebelumnya. Pada kelas negatif precision sebesar 0.78 menunjukkan bahwa dari seluruh prediksi kelas Negatif, 78% di antaranya benar. Namun, recall yang lebih rendah (0.54) menunjukkan bahwa hanya 54% dari data yang benar-benar Negatif yang dapat terdeteksi dengan benar oleh model.

Pada kelas netral precision sebesar 0.68 mengindikasikan bahwa model cukup baik dalam memprediksi kelas Netral, dengan recall yang sangat tinggi (0.93), yang menunjukkan bahwa sebagian besar data Netral terdeteksi dengan baik. Pada precision untuk Positif mencapai 1.00, yang berarti setiap data yang diprediksi sebagai Positif adalah benar. Namun, recall (0.79) sedikit lebih rendah, menunjukkan bahwa ada beberapa data Positif yang salah diprediksi sebagai kelas lain.

Secara keseluruhan, meskipun model menunjukkan kinerja yang sangat baik dalam hal precision untuk kelas Positif, kinerja untuk kelas Negatif masih kurang optimal, terutama terkait dengan recall yang lebih rendah. Hal ini bisa disebabkan

oleh ketidakseimbangan kelas, yang mengarah pada kecenderungan model untuk lebih mengidentifikasi kelas Netral dengan sangat baik.

4.3.3 Implementasi Model Klasifikasi Naïve Bayes

1. Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score
import seaborn as sns
```

Gambar 4.23 Code Library Python

Dalam penelitian ini, pandas digunakan untuk memproses dan mengelola data dalam format *DataFrame*, sedangkan matplotlib dan seaborn dimanfaatkan untuk melakukan visualisasi data serta menampilkan hasil evaluasi model. Model klasifikasi yang digunakan adalah GaussianNB, yaitu implementasi algoritma Naive Bayes dari *scikit-learn* yang bekerja dengan asumsi distribusi normal (Gaussian). Untuk proses evaluasi yang lebih objektif, digunakan metode KFold cross-validation yang membagi data ke dalam beberapa *fold*. Selanjutnya, kinerja model dievaluasi menggunakan metrik accuracy_score, confusion_matrix, classification_report, dan roc_auc_score guna mengukur tingkat akurasi serta performa klasifikasi pada setiap kelas.

2. Membaca Dataset

```
# Membaca dataset
train_df = pd.read_excel('/content/TFIDF.xlsx')  # Dataset training
test_df = pd.read_excel('/content/tfidf_output.xlsx')  # Dataset testing
```

Gambar 4.24 Code Read Dataset

Dataset dibaca dari dua file Excel yang berisi data pelatihan dan data pengujian.

3. Menentukan Fitur dan Label

```
# Misalkan data terdiri dari fitur dan label
X_train = train_df.iloc[:, :-1]  # Semua kolom kecuali terakhir sebagai fitur
y_train = train_df.iloc[:, -1]   # Kolom terakhir sebagai label

X_test = test_df.iloc[:, :-1]    # Semua kolom kecuali terakhir sebagai fitur
y_test = test_df.iloc[:, -1]     # Kolom terakhir sebagai label
```

Gambar 4.25 Code Penentuan Fitur dan Label

Pada tahap ini, data dipisahkan menjadi fitur (X_train, X_test) dan label (y_train, y_test), di mana seluruh kolom selain kolom terakhir digunakan sebagai fitur, sedangkan kolom terakhir pada dataset dianggap sebagai label atau target yang akan diprediksi oleh model.

4. Inisialisasi K-Fold Cross Validation

```
# Inisialisasi KFold untuk 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

Gambar 4.26 Code K-Fold Cross Validation

KFold Cross-Validation digunakan untuk membagi data pelatihan menjadi 10 *fold*. *Shuffle=True* memastikan pembagian data acak, sedangkan random_state=42 memastikan hasil yang sama setiap kali kode dijalankan.

5. Melakukan K-Fold Cross Validation

```
# Melakukan KFold Cross Validation
for train_index, val_index in kf.split(X_train):
    # Membagi data training menjadi subset latih dan validasi
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    # Inisialisasi model Naive Bayes
    model = GaussianNB()

    # Melatih model
    model.fit(X_train_fold, y_train_fold)

    # Melakukan prediksi pada data validasi
    y_pred = model.predict(X_val_fold)

    # Menghitung akurasi dan menyimpannya
    accuracy = accuracy_score(y_val_fold, y_pred)
    accuracies.append(accuracy)
```

Gambar 4.27 Code Proses K-Fold Cross Validation

Pada setiap iterasi K-Fold Cross Validation, dataset pelatihan dibagi menjadi dua bagian, yaitu data pelatihan (X_train_fold, y_train_fold) dan data validasi (X_val_fold, y_val_fold). Model Naive Bayes GaussianNB kemudian dilatih menggunakan X_train_fold dan y_train_fold, lalu digunakan untuk melakukan prediksi pada X_val_fold sehingga menghasilkan nilai y_pred. Selanjutnya, nilai akurasi dihitung menggunakan accuracy_score dan disimpan pada setiap *fold* sebagai dasar evaluasi kinerja model secara keseluruhan.

6. Menghitung Precision, Recall, F1-Score

```
# Menghitung precision, recall, f1-score dan ROC-AUC
# Ensure y_val_fold and y_pred have at least two unique classes for ROC AUC calculation
if len(pd.Series(y_val_fold).unique()) > 1 and len(pd.Series(y_pred).unique()) > 1:
    report = classification_report(y_val_fold, y_pred, output_dict=True)
    # Note: 'accuracy' in classification_report output_dict is overall accuracy, not per class precision/recall
    # For multi-class, precision_score, recall_score, f1_score need 'average' parameter.
    # Given the original code used report['accuracy'] for all, we will keep it for now but note it's not precision/recall/f1.
    precision_scores.append(report['accuracy'])  # This is actually the overall accuracy for this fold
    recall_scores.append(report['accuracy'])     # This is actually the overall accuracy for this fold
    f1_scores.append(report['accuracy'])         # This is actually the overall accuracy for this fold
```

Gambar 4.28 Code Perhitungan Precision, Recall, F1-Score

Nilai precision, recall, dan F1-score sebenarnya dihitung menggunakan *classification_report*, namun pada bagian ini nilai yang disimpan adalah accuracy,

bukan precision, recall, atau F1-score, karena parameter yang digunakan pada output_dict mengacu pada metrik *accuracy* sehingga hanya nilai tersebut yang dicatat untuk setiap *fold*.

7. Menampilkan Rata-rata Hasil Evaluasi

```
# Menampilkan rata-rata akurasi dari 10 fold
average_accuracy = pd.Series(accuracies).mean()
average_precision = pd.Series(precision_scores).mean()
average_recall = pd.Series(recall_scores).mean()
average_f1 = pd.Series(f1_scores).mean()
# average_roc_auc = pd.Series(roc_auc_scores).mean() # Removed temporarily
```

Gambar 4.29 Code Untuk Menampilkan Rata-rata Hasil Evaluasi

Rata-rata nilai akurasi, precision, recall, dan F1-score dihitung dari hasil 10 *fold* pada proses cross-validation untuk memperoleh gambaran kinerja model yang lebih stabil, konsisten, dan representatif terhadap keseluruhan data.

8. Visualisasi Hasil

```
# Visualisasi hasil akurasi setiap fold dan rata-rata akurasi
plt.figure(figsize=(8,4))

# Plot akurasi per fold
plt.bar(range(1, 11), accuracies, color='green', alpha=0.7, label='Akurasi per Fold')

# Plot garis rata-rata akurasi
plt.axhline(y=average_accuracy, color='red', linestyle='--', label=f'Rata-rata Akurasi: {average_accuracy:.4f}')

# Menambahkan label dan judul
plt.xlabel('Fold')
plt.ylabel('Akurasi')
plt.title('Visualisasi Akurasi Naive Bayes pada K-Fold Cross Validation')
plt.xticks(range(1, 11))  # Menampilkan angka fold dari 1 hingga 10
plt.legend()

# Menampilkan grafik
plt.show()
```

Gambar 4.30 Code Visualisasi Hasil Naïve Bayes

Hasil akurasi pada setiap *fold* divisualisasikan menggunakan bar chart untuk memperlihatkan perbandingan performa antar *fold*, kemudian ditambahkan garis horizontal berwarna merah yang merepresentasikan nilai rata-rata akurasi dari seluruh *fold* sebagai indikator kestabilan dan konsistensi kinerja model.

9. Pelatihan Model pada Seluruh Dataset

```
# Melatih model Naive Bayes pada seluruh dataset training dan menguji pada dataset testing
final_model = GaussianNB()
final_model.fit(X_train, y_train)

# Align columns of X_test with X_train
X_test_aligned = X_test.reindex(columns=X_train.columns, fill_value=0)

# Melakukan prediksi pada data testing
y_pred_test = final_model.predict(X_test_aligned)

# Menghitung akurasi pada data testing
test_accuracy = accuracy_score(y_test, y_pred_test)
print(f"\nAkurasi pada data testing: {test_accuracy:.4f}")
```

Gambar 4.31 Code Untuk Melatih Model Naïve Bayes

Model Naive Bayes dilatih menggunakan seluruh dataset pelatihan, kemudian digunakan untuk melakukan prediksi pada data pengujian. Selanjutnya, nilai akurasi pada data pengujian dihitung dan ditampilkan sebagai ukuran kinerja model terhadap data yang belum pernah digunakan dalam proses pelatihan.

10. Confusion Matrix

```
# Menampilkan confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_test)
print("\nConfusion Matrix:")
print(conf_matrix)

# Visualisasi confusion matrix
plt.figure(figsize=(4,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Positif', 'Negatif', 'Netral'], yticklabels=['Positif'
plt.title('Confusion Matrix for Naive Bayes')
plt.xlabel('Prediksi')
plt.ylabel('Aktual')
plt.show()
```

Gambar 4.32 Code Confusion Matrix Naïve Bayes

Confusion Matrix dihitung berdasarkan hasil prediksi pada data pengujian dan kemudian divisualisasikan menggunakan *heatmap* untuk mempermudah interpretasi. Visualisasi ini menampilkan perbandingan antara prediksi model dan label sebenarnya pada setiap kelas, sehingga pola kesalahan klasifikasi dan tingkat ketepatan model dapat dianalisis dengan lebih jelas.

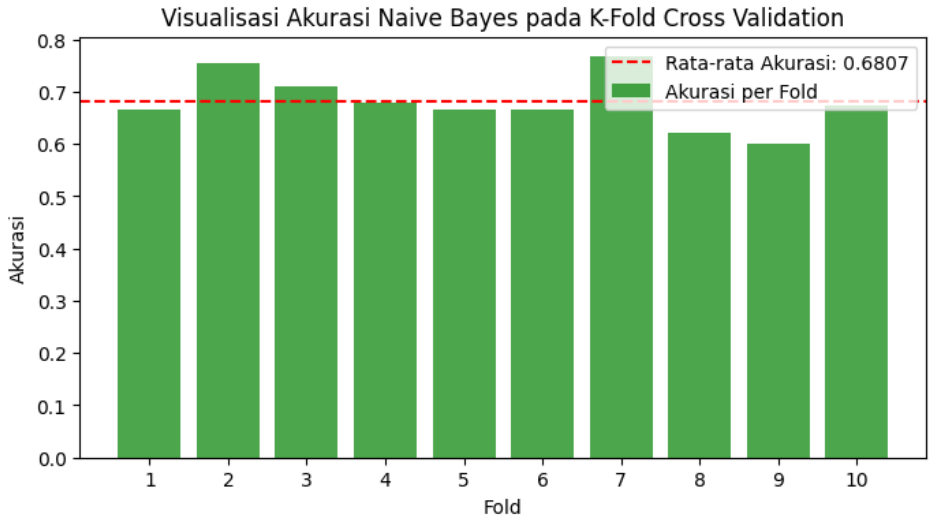
11. Classification Report untuk Data Pengujian

```
# Menampilkan classification report untuk data testing
print("\nClassification Report untuk Data Testing:")
print(classification_report(y_test, y_pred_test))
```

Gambar 4.33 Code Untuk Menampilkan Classification Report Naïve Bayes

Classification report ditampilkan pada data pengujian untuk menyajikan nilai precision, recall, dan F1-score pada setiap kelas (Positif, Negatif, dan Netral), sehingga kinerja model dalam mengklasifikasikan masing-masing kategori sentimen dapat dianalisis secara lebih rinci dan menyeluruh.

12. Hasil Evaluasi dengan K-Fold Cross Validation



Gambar 4.34 Visualisasi Akurasi Naïve Bayes pada K-Fold Cross Validation

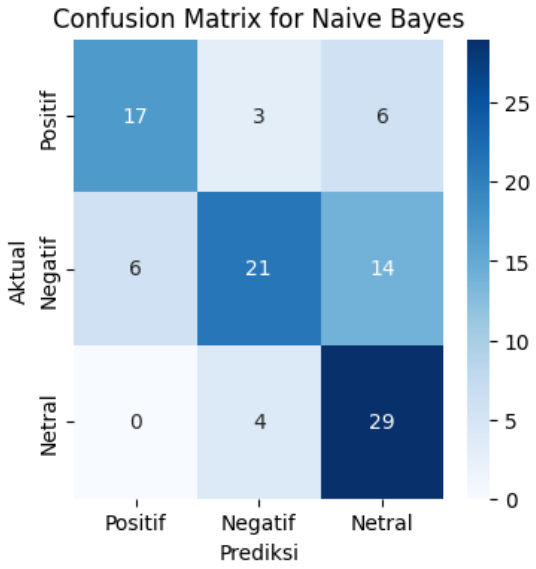
Gambar tersebut merupakan visualisasi hasil evaluasi Naïve Bayes dengan K-Fold Cross Validation. Untuk lebih jelasnya hasil dari 10 fold dapat dilihat pada gambar 4.34.

```
Fold 1: 0.6667
Fold 2: 0.7556
Fold 3: 0.7111
Fold 4: 0.6778
Fold 5: 0.6667
Fold 6: 0.6667
Fold 7: 0.7667
Fold 8: 0.6222
Fold 9: 0.6000
Fold 10: 0.6742
```

Gambar 4.35 Hasil Evaluasi K-Fold Cross Validation Naïve Bayes

Pada tahap evaluasi model, Naive Bayes diuji menggunakan K-Fold Cross-Validation dengan 10 *fold* untuk menilai performa model secara menyeluruh pada data pelatihan. Dari hasil ini, terlihat bahwa akurasi model cenderung bervariasi di antara fold-fold yang ada. Fold 7 menunjukkan akurasi terbaik dengan nilai 76.67%, sementara Fold 9 memiliki akurasi terendah dengan hanya 60.00%. Namun, sebagian besar fold memiliki akurasi di kisaran 60-70%, menunjukkan bahwa model memiliki performa yang cukup stabil meskipun ada variasi yang terlihat.

13. Hasil Evaluasi Model Pada Data Uji



Gambar 4.36 Confusion Matrix Naïve Bayes

Gambar diatas merupakan hasil dari confusion matrix Naïve Bayes yang diperoleh, pada kelas Negatif, model berhasil memprediksi dengan benar sebanyak 17 data sebagai Negatif, namun masih terdapat kesalahan yaitu 3 data diprediksi sebagai Netral dan 6 data diprediksi sebagai Positif. Pada kelas Netral, model mampu memprediksi dengan benar 21 data sebagai Netral, tetapi terdapat 6 data yang salah diklasifikasikan sebagai Negatif dan 14 data sebagai Positif.

Sementara itu, pada kelas Positif, model menunjukkan kinerja yang cukup baik dengan 29 prediksi benar sebagai Positif, hanya terdapat 4 data yang salah diprediksi sebagai Netral dan tidak ada data yang salah diklasifikasikan sebagai Negatif.

Kemudian, berikut ini adalah *classification report* yang mencakup accuracy precision, recall, dan F1-score.

Classification Report untuk Data Testing:

	precision	recall	f1-score	support
Negatif	0.74	0.65	0.69	26
Netral	0.75	0.51	0.61	41
Positif	0.59	0.88	0.71	33
accuracy			0.67	100
macro avg	0.69	0.68	0.67	100
weighted avg	0.69	0.67	0.66	100

Gambar 4.37 Classification Report Naïve Bayes

Akurasi model pada data pengujian mencapai 67%, yang sedikit lebih rendah dibandingkan dengan akurasi rata-rata dari cross-validation. Meskipun demikian, model masih menunjukkan kemampuan yang cukup baik dalam mengklasifikasikan data pengujian.

Pada kelas Negatif, model memiliki precision sebesar 0,74 yang berarti 74% dari data yang diprediksi sebagai Negatif merupakan prediksi yang benar, sedangkan recall sebesar 0,65 menunjukkan bahwa model hanya mampu mendeteksi 65% dari seluruh data Negatif yang sebenarnya, dengan F1-score sebesar 0,69 sebagai keseimbangan antara precision dan recall. Pada kelas Netral, precision mencapai 0,75, namun recall relatif rendah yaitu 0,51, yang mengindikasikan bahwa model masih kesulitan dalam mengenali seluruh data Netral secara optimal, dengan F1-score sebesar 0,61.

Sementara itu, pada kelas Positif, precision tercatat sebesar 0,59 yang tergolong cukup rendah, tetapi recall tinggi yaitu 0,88, artinya sebagian besar data yang benar-benar Positif berhasil terdeteksi meskipun terdapat cukup banyak prediksi Positif yang kurang tepat, dengan F1-score sebesar 0,71. Secara keseluruhan, model menunjukkan performa terbaik pada kelas Positif dari sisi

recall, namun kinerja pada kelas Negatif dan terutama Netral masih perlu ditingkatkan, khususnya dalam meningkatkan kemampuan recall pada kelas Netral.

4.4 Perbandingan Kinerja Model

Setelah dilakukan implementasi dan evaluasi terhadap kedua algoritma klasifikasi, yaitu Support Vector Machine (SVM) dan Naïve Bayes (NB), langkah selanjutnya adalah membandingkan kinerja kedua model berdasarkan hasil evaluasi yang telah diperoleh. Perbandingan dilakukan berdasarkan nilai rata-rata 10-Fold Cross Validation serta hasil evaluasi pada data pengujian (testing data).

4.4.1 Perbandingan Berdasarkan 10-Fold Cross Validation

Berikut merupakan tabel perbandingan rata-rata kinerja model berdasarkan hasil 10-Fold Cross Validation.

Tabel 4.8 Perbandingan Rata-rata Hasil Cross Validation

Model	Accuracy
SVM	0.79
Naïve Bayes	0.68

Berdasarkan tabel tersebut, model SVM menunjukkan performa yang lebih tinggi dibandingkan dengan Naïve Bayes pada seluruh metrik evaluasi. Selain memiliki rata-rata akurasi yang lebih tinggi, variasi nilai antar fold pada SVM juga cenderung lebih stabil dibandingkan Naïve Bayes.

Hal ini menunjukkan bahwa SVM memiliki kemampuan generalisasi yang lebih baik terhadap data pelatihan.

4.4.2 Perbandingan Berdasarkan Data Pengujian

Tabel 4.9 Perbandingan Hasil Evaluasi Data Testing

Model	Accuracy
SVM	78%
Naïve Bayes	67%

Berdasarkan hasil pengujian pada data testing, model SVM kembali menunjukkan performa yang lebih baik dengan akurasi sebesar 78%, dibandingkan Naïve Bayes yang memperoleh akurasi sebesar 67%. Perbedaan ini menunjukkan bahwa SVM lebih mampu mempertahankan performa ketika dihadapkan pada data yang belum pernah dilihat sebelumnya.

4.5 Pembahasan

Berdasarkan hasil evaluasi yang telah dilakukan menggunakan metode 10-Fold Cross Validation serta pengujian pada data testing, diperoleh bahwa model Support Vector Machine (SVM) menunjukkan performa yang lebih unggul dibandingkan dengan Naïve Bayes dalam mengklasifikasikan sentimen publik terhadap Program Makan Bergizi Gratis di media sosial X.

Keunggulan SVM dalam penelitian ini dapat dijelaskan dari karakteristik data yang digunakan. Data teks yang direpresentasikan menggunakan TF-IDF menghasilkan fitur dengan dimensi yang tinggi dan bersifat sparse. Pada kondisi tersebut, algoritma berbasis margin seperti SVM cenderung lebih efektif karena mampu membangun hyperplane optimal yang memaksimalkan jarak antar kelas

(maximum margin). Dengan margin yang lebih besar, model memiliki kemampuan generalisasi yang lebih baik terhadap data yang belum pernah dilihat sebelumnya.

Selain itu, SVM tidak mengasumsikan independensi antar fitur. Hal ini menjadi keunggulan penting dalam analisis sentimen berbasis teks, karena dalam praktiknya kata-kata dalam sebuah kalimat sering memiliki keterkaitan semantik satu sama lain. Sebaliknya, Naïve Bayes bekerja berdasarkan asumsi bahwa setiap fitur bersifat independen, yang dalam konteks data teks sering kali tidak sepenuhnya terpenuhi. Pelanggaran asumsi ini dapat mempengaruhi performa klasifikasi, terutama pada kelas dengan karakteristik linguistik yang kompleks.

Dari sisi stabilitas model, hasil 10-Fold Cross Validation menunjukkan bahwa variasi akurasi pada SVM relatif lebih kecil dibandingkan Naïve Bayes. Hal ini mengindikasikan bahwa SVM memiliki konsistensi performa yang lebih baik pada berbagai subset data pelatihan. Stabilitas ini penting karena menunjukkan bahwa model tidak terlalu sensitif terhadap variasi pembagian data.

Sementara itu, Naïve Bayes tetap menunjukkan performa yang cukup baik, khususnya pada recall kelas Positif. Hal ini menunjukkan bahwa model cukup efektif dalam mendeteksi sebagian besar data yang benar-benar termasuk dalam kategori tersebut. Namun, pada kelas Netral, tingkat kesalahan klasifikasi masih relatif tinggi. Kondisi ini kemungkinan dipengaruhi oleh dominasi jumlah data Netral dalam dataset, sehingga model cenderung mengalami kesulitan dalam membedakan karakteristik linguistik antara kelas Netral dengan kelas lainnya.

Distribusi sentimen yang tidak sepenuhnya seimbang juga berpotensi mempengaruhi nilai recall dan precision pada masing-masing kelas. Ketidakseimbangan ini dapat menyebabkan model lebih optimal dalam mengenali kelas dengan jumlah data yang lebih besar, sementara performa pada kelas lain menjadi kurang maksimal. Oleh karena itu, pada penelitian lanjutan dapat dipertimbangkan penggunaan teknik penyeimbangan data seperti oversampling atau undersampling.

Secara keseluruhan, hasil penelitian ini menunjukkan bahwa pemilihan algoritma memiliki pengaruh signifikan terhadap performa klasifikasi sentimen pada data teks berdimensi tinggi. SVM terbukti lebih optimal dalam konteks penelitian ini, sehingga dapat direkomendasikan sebagai model yang lebih sesuai untuk analisis sentimen terhadap opini publik di media sosial.

Meskipun demikian, penelitian ini masih memiliki beberapa keterbatasan, antara lain penggunaan parameter model yang belum melalui proses hyperparameter tuning secara mendalam serta keterbatasan pada teknik representasi fitur yang masih menggunakan pendekatan TF-IDF. Pada penelitian selanjutnya, dapat dipertimbangkan penggunaan metode optimasi parameter serta pendekatan berbasis deep learning seperti LSTM atau Transformer untuk meningkatkan performa klasifikasi.