BAB IV

HASIL DAN PEMBAHASAN

4.1 Persiapan

4.1.1 Studi Literatur

Ketika Anda membuat laporan penelitian ini, Anda harus mengikuti metode penelitian yang Anda buat sebelumnya. Langkah pertama adalah mencari daftar buku, jurnal, esai, dan laporan penelitian sebelumnya untuk menjadi dasar penelitian ini. Bab II mencakup tinjauan literatur yang berfungsi sebagai landasan teori untuk penelitian ini. Tinjauan ini menjelaskan teori yang mendukung penelitian.

4.1.2 Pengumpulan Data dan Informasi

Pada tahap awal pengumpulan data dan informasi, kegiatan pengumpulan data dan informasi yang diperlukan untuk penelitian ini dilakukan dan beberapa langkah harus dilakukan untuk melakukan kegiatan tersebut.

a. Observasi

   Setelah peneliti memutuskan alat yang dibutuhkan, mereka melakukan observasi untuk menemukan emosi tentang Undang-Undang Ibu Kota Nusantara di media sosial Twitter.

b. Dokumentasi

   Pada fase ini peneliti mendokumentasikan hasil dokumentasi yang telah dilakukan sebelumnya, dan hasil dokumentasi dari kegiatan tersebut digunakan sebagai urutan/urutan laporan dan penelitian yang dilakukan.

4.1.3 Requirment Definition

Pada tahap requirements definition, hal ini dilakukan untuk mengetahui arus/proses pengesahan Undang-Undang Ibu Kota Nusnatara yang sedang berjalan untuk mengidentifikasi kebutuhan dan kelemahan dari sistem yang sedang berjalan. Kegiatan ini meliputi beberapa tahapan, antara lain:

Pada titik ini, data yang digunakan dalam penelitian diidentifikasi. Data yang digunakan dalam hal ini adalah data dari National City Act atau postingan di media sosial Twitter. Melakukan percobaan memberikan penjelasan lebih lanjut tentang kebutuhan data yang digunakan dalam bab 4.2.

4.2 Pelaksanaan Eksperimen

4.2.1 Pemodelan Data (Analisis Data)

Pemodelan data dilakukan dengan melakukan langkah-langkah preprocessing atau pengolahan data. Artinya, beberapa proses melakukan langkah-langkah analisis data. Detail dari proses-proses tersebut adalah sebagai berikut:

a. Membaca Dataset

Pada tahap pertama peneliti akan membaca dataset terlebih dahulu yang dapat dilihat dibawah ini.

```
# Membaca dataset
df =
pd.read_csv('/content/sample_data/datasettweetikn12x
           ..csv', delimiter=';')
df.head()
```

	created_at	username	full_text	sentiment
0	Fri Jan 05 08:15:01 +0000 2024	tempodotco	Belakangan ini ramai diberitakan bahwa Djarum ...	Positive
1	Tue Jan 02 23:45:09 +0000 2024	Aryprasetyo85	Luar biasa Kinerja Kejaksaan berhasil menyelam...	Negative
2	Wed Jan 03 08:36:58 +0000 2024	asumsico	Kementerian Keuangan (Kemenkeu) mencatat total...	Neutral
3	Mon Jan 01 04:39:40 +0000 2024	bukankingsalman	Mari kita buka tahun 2024 dgn menyadari bahwa ...	Positive
4	Fri Jan 05 23:41:01 +0000 2024	Saodah1234	Para konglomerat Indonesia turut serta dalam p...	Positive

Gambar 4. 1 Dataset Tweet

b. Fungsi Yang Ditentukan Khusus Untuk Membersihkan Tweet

Fungsi-fungsi definisikan adalah bagian dari pra-pemrosesan (preprocessing) teks yang umum dilakukan sebelum menerapkan algoritma pemodelan pada data teks. Berikut adalah penjelasan singkat tentang setiap fungsi:

```
#Clean emojis from text
def strip_emoji(text):
    return re.sub(emoji.get_emoji_regexp(), r"", text) #remove emoji
```

Fungsi strip_emoji(text): ini bertujuan untuk menghapus emoji dari teks. Ini dilakukan dengan menggunakan modul emoji untuk mendeteksi dan menghapus emoji dari teks menggunakan ekspresi reguler.

```
#Remove punctuations, links, mentions and \r\n new line characters
def strip_all_entities(text):
    text = text.replace('\r', '').replace('\n', ' ').replace('\n', '
').lower() #remove \n and \r and lowercase
    text = re.sub(r"(?:\@|https?\://)\S+", "", text) #remove links and
mentions
    text = re.sub(r'[^\x00-\x7f]',r'', text) #remove non utf8/ascii
characters such as '\x9a\x91\x97\x9a\x97'
    banned_list= string.punctuation + 'Ã'+'±'+'ã'+'¼'+'â'+'»'+'§'
    table = str.maketrans('', '', banned_list)
    text = text.translate(table)
    return text
```

Fungsi ini strip_all_entities(text): bertujuan untuk membersihkan teks dari beberapa jenis entitas, seperti tautan, mention (username), dan karakter baru seperti \r dan \n.

Teks diubah menjadi huruf kecil, dan kemudian menggunakan ekspresi reguler untuk menghapus tautan, mention, dan karakter \r dan \n. Selain itu, karakter non-ASCII juga dihapus.

```
#clean hashtags at the end of the sentence, and keep those in the
middle of the sentence by removing just the # symbol
def clean_hashtags(tweet):
    new_tweet = " ".join(word.strip() for word in
re.split('#(?!(?:hashtag)\b)[\w-]+(?=(?:\s+#[\w-]+)*\s*$)', tweet))
#remove last hashtags
    new_tweet2 = " ".join(word.strip() for word in re.split('#|_',
new_tweet)) #remove hashtags symbol from words in the middle of the
sentence
    return new_tweet2
```

Fungsi ini clean_hashtags(tweet): membersihkan hashtag dari teks.Jika hashtag berada di akhir kalimat, hashtag tersebut dihapus. Namun, jika hashtag berada di tengah kalimat, hanya simbol # dihapus.

Ini dilakukan menggunakan ekspresi reguler untuk memisahkan hashtag dari teks dan kemudian menghapus simbol # dari setiap kata.

```
#Filter special characters such as & and $ present in some words
def filter_chars(a):
    sent = []
    for word in a.split(' '):
        if ('$' in word) | ('&' in word):
            sent.append('')
        else:
            sent.append(word)
```

```
    return ' '.join(sent)
```

Fungsi filter_chars(a): ini bertujuan untuk memfilter karakter khusus seperti & dan $ yang mungkin terdapat dalam kata-kata.Ini dilakukan dengan memeriksa setiap kata dalam teks, dan jika kata mengandung karakter & atau $, karakter tersebut dihapus dari kata tersebut.

```
def remove_mult_spaces(text): # remove multiple spaces
    return re.sub("\s\s+" , " ", text)
```

Fungsi remove_mult_spaces(text): ini bertujuan untuk menghapus spasi berlebih dari teks.Ini dilakukan dengan menggunakan ekspresi reguler untuk mengganti beberapa spasi berturut-turut dengan satu spasi.

c. Membersihkan Emoji dari Teks

```
import emoji
import re

def strip_emoji(text):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # simbol &
piktogram
                               u"\U0001F680-\U0001F6FF"  # transportasi &
simbol map
                               u"\U0001F1E0-\U0001F1FF"  # bendera (iOS)
                               u"\U00002500-\U00002BEF"  # CJK Ext A
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u200d"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
```

```
                            u"\u23cf"
                            u"\u23e9"
                            u"\u231a"
                            u"\u3030"
                            "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)

 # Gunakan fungsi strip_emoji() dalam loop untuk membersihkan teks
texts_new = []
for t in df.full_text:
    texts_new.append(remove_mult_spaces(filter_chars(clean_hashtags(str
ip_all_entities(strip_emoji(t))))))

# Lanjutkan proses Anda dengan menggunakan `texts_new` yang sudah
dibersihkan dari emoji
```

Fungsi strip_emoji(text) yang baru didefinisikan bertujuan untuk membersihkan emoji dari teks. Ini dilakukan dengan menggunakan ekspresi reguler untuk mengidentifikasi dan menghapus karakter emoji dari teks.

Berikut adalah penjelasan tentang langkah-langkah yang Anda lakukan dalam proses ini:

Loop Melalui Setiap Teks (for t in df.full_text:):

Loop melalui setiap teks dalam kolom full_text dari DataFrame df.

Pemanggilan Fungsi Pra-Pemrosesan (texts_new.append(...)):

Untuk setiap teks, peneliti menerapkan serangkaian fungsi pra-pemrosesan yang telah definisikan sebelumnya (strip_emoji, strip_all_entities, clean_hashtags, filter_chars, remove_mult_spaces) untuk membersihkan teks dari emoji dan entitas lainnya seperti tautan, mention, karakter khusus, dan spasi berlebih.

Teks yang telah dibersihkan kemudian dimasukkan ke dalam daftar texts_new.

Setelah loop selesai dieksekusi, texts_new akan berisi teks-teks yang telah dibersihkan dari berbagai entitas yang mungkin mengganggu.

```
array(['luar biasa kinerja kejaksaan berhasil menyelamatkan anggaran ikn rp24 2 triliun selama tahun 2023 kejaksaan agung atau kejagung mengawal
rp 24 2 triliun anggaran negara untuk pembangunan ibu kota nusantara ikn sepanjang 2023 anggaran itu merupakan realisasi 28 kegiatan',
       'kementerian keuangan kemenkeu mencatat total alokasi anggaran untuk ibu kota nusantara ikn tahun 2022 sampai dengan 2024 sebesar rp 72 8
triliun',
       'mari kita buka tahun 2024 dgn menyadari bahwa di tahun ini jakarta bukan lagi ibu kota indonesia berikut progres pembangunan ibu kota
nusantara semakin yakin upacara 17 agustus 2024 di ikn',
       'para konglomerat indonesia turut serta dalam pembangunan ibu kota nusantara ikn otorita ikn mengungkap 15 investor bakal melakukan
groundbreaking proyek baru di ikn pada januari hingga februari 2024 mendatang saipul jamil nusakambangan libya polisi tom lembong dana',
       'kejaksaan agung mengawal rp24 2 triliun anggaran negara untuk pembangunan ibu kota nusantara ikn sepanjang 2023 anggaran itu merupakan
realisasi 28 kegiatan pembangunan yang sedang digarap pemerintah selain proyek ikn kejaksaan juga mengawal 55 proyek strategis negara psn',
       'ibu kota nusantara ikn dirancang menjadi kota hutan berkelanjutan yang mempunyai target kota pertama nol emisi di indonesia pada 2045 ikn
sebagai kota berkelanjutan dan pusat inovasi hijau yang berfokus terhadap pengurangan emisi karbon',
       'ibu kota nusantara diserbu 70 investor'], dtype=object)
```

Gambar 4. 2 text yang sudah dibersihkan dari berbagai entitas

d. Tokenisasi Teks

```
token_lens = []

for txt in df['text_clean'].values:
    tokens = tokenizer.encode(txt, max_length=512, truncation=True)
    token_lens.append(len(tokens))

max_len=np.max(token_lens)

print(f"MAX TOKENIZED SENTENCE LENGTH: {max_len}")
```

Berikut adalah penjelasan dari setiap baris kode:

1. token_lens = []: Ini adalah pembuatan sebuah list kosong yang akan digunakan untuk menyimpan panjang token-token dari setiap teks.
2. for txt in df['text_clean'].values:: Ini adalah loop yang mengiterasi melalui setiap teks yang telah dibersihkan yang mungkin berada dalam kolom 'text_clean' dari suatu DataFrame (df).
3. tokens = tokenizer.encode(txt, max_length=512, truncation=True): Dalam setiap iterasi, teks (txt) di-tokenisasi menggunakan sebuah tokenizer dengan parameter yang ditentukan. Tokenisasi dilakukan dengan memanggil metode encode dari objek tokenizer, di mana teks (txt) dimasukkan sebagai argumen,

bersama dengan parameter lain seperti max_length yang menentukan panjang maksimum dari token yang dihasilkan, dan truncation yang menentukan apakah teks harus dipotong jika melebihi panjang maksimum yang diperbolehkan.

4. token_lens.append(len(tokens)): Panjang token dari teks yang telah di-tokenisasi kemudian ditambahkan ke dalam list token_lens.
5. max_len=np.max(token_lens): Setelah semua teks telah di-tokenisasi dan panjang token telah dihitung, panjang maksimum dari token yang ditemukan disimpan dalam variabel max_len dengan menggunakan fungsi np.max dari NumPy.
6. print(f"MAX TOKENIZED SENTENCE LENGTH: {max_len}"): Akhirnya, panjang maksimum dari token yang ditemukan dicetak ke layar.

```
token_lens = []

for i,txt in enumerate(df['text_clean'].values):
    tokens = tokenizer.encode(txt, max_length=512, truncation=True)
    token_lens.append(len(tokens))
    if len(tokens)>80:
        print(f"INDEX: {i}, TEXT: {txt}")
```

Berikut adalah penjelasan dari setiap baris kode:

1. token_lens = []: Baris ini membuat sebuah list kosong dengan nama token_lens. List ini akan digunakan untuk menyimpan panjang dari setiap teks yang telah di-tokenisasi.
2. for txt in df['text_clean'].values:: Baris ini memulai sebuah loop for yang akan mengiterasi melalui setiap teks dalam kolom 'text_clean' dari DataFrame df. Nilai teks ini diakses melalui atribut values dari kolom DataFrame tersebut.

3. tokens = tokenizer.encode(txt, max_length=512, truncation=True): Di dalam loop, setiap teks dibersihkan (txt) akan di-tokenisasi menggunakan fungsi encode dari objek tokenizer. Hasil tokenisasi akan disimpan dalam variabel tokens. Parameter max_length=512 menetapkan panjang maksimum dari token yang dihasilkan, sedangkan truncation=True mengindikasikan bahwa teks akan dipotong jika melebihi panjang maksimum yang ditetapkan.
4. token_lens.append(len(tokens)): Panjang dari token-token yang dihasilkan dari teks tersebut ditambahkan ke dalam list token_lens. len(tokens) memberikan jumlah token dalam teks tersebut, yang kemudian dimasukkan ke dalam list.
5. max_len=np.max(token_lens): Setelah semua teks telah di-tokenisasi dan panjang token telah dihitung dan disimpan dalam list token_lens, baris ini menggunakan fungsi np.max dari pustaka NumPy untuk mencari panjang maksimum dari token-token tersebut. Nilai ini kemudian disimpan dalam variabel max_len.
6. print(f"MAX TOKENIZED SENTENCE LENGTH: {max_len}"): Akhirnya, panjang maksimum dari token yang dihasilkan dari seluruh teks dalam DataFrame df dicetak ke layar. Pesan ini menggunakan f-string Python untuk mencetak nilai variabel max_len di dalam string.

```
df['token_lens'] = token_lens
df = df.sort_values(by='token_lens', ascending=False)
df.head(20)
```

Kode ini menambahkan kolom baru ke dalam DataFrame df yang disebut 'token_lens'. Kolom ini berisi panjang dari token-token yang dihasilkan dari setiap

teks dalam DataFrame yang telah dihitung sebelumnya dan disimpan dalam list token_lens.

Selanjutnya, DataFrame diurutkan berdasarkan nilai dalam kolom 'token_lens' secara menurun (descending) menggunakan metode .sort_values(). Ini berarti teks dengan panjang token terbesar akan muncul di bagian atas DataFrame setelah pengurutan.

Terakhir, kode mencetak 20 baris pertama dari DataFrame yang telah diurutkan ini menggunakan metode .head(20). Ini memberikan kita 20 teks pertama dalam DataFrame yang memiliki panjang token terbesar, memberikan wawasan tentang teks-teks yang memerlukan perhatian khusus atau mungkin membutuhkan pemrosesan tambahan.

	full_text	sentiment	text_clean	text_len	token_lens
1	Luar biasa Kinerja Kejaksaan berhasil menyelam...	Negative	luar biasa kinerja kejaksaan berhasil menyelam...	39	106
953	Kunjungan Presiden Jokowi ke Ibu Kota Nusantar...	Positive	kunjungan presiden jokowi ke ibu kota nusantar...	41	105
895	Pemerintah mulai menyusun rencana pengembangan...	Positive	pemerintah mulai menyusun rencana pengembangan...	36	104
166	Calon wakil presiden (cawapres) nomor urut 3 M...	Positive	calon wakil presiden cawapres nomor urut 3 mah...	38	104
5	Kejaksaan Agung mengawal Rp24 2 triliun anggar...	Positive	kejaksaan agung mengawal rp24 2 triliun anggar...	38	103
319	Revisi undang-undang Ibu Kota Nusantara atau I...	Positive	revisi undangundang ibu kota nusantara atau ik...	39	102
538	Teruskan Jejak Presiden Jokowi Ganjar Pranowo...	Positive	teruskan jejak presiden jokowi ganjar pranowo ...	38	102
135	Ingat kasus Israel demi konstitusi Ganjar rel...	Negative	ingat kasus israel demi konstitusi ganjar rela...	41	102
271	Visi Misi Ganjar Mahfud dari Keberlanjutan IK...	Positive	visi misi ganjar mahfud dari keberlanjutan ikn...	37	102
137	Menikmati malam di kawasan Ibu Kota Nusantara ...	Neutral	menikmati malam di kawasan ibu kota nusantara ...	38	102
389	Halo #TemanKita Tahu ga salah satu anak perus...	Neutral	halo temankita tahu ga salah satu anak perusah...	38	102
746	Presiden @jokowi optimis bahwa proyek pembangu...	Positive	presiden optimis bahwa proyek pembangunan ibu ...	38	102
792	Kementerian PUPR menawarkan kepada Brunei Dar...	Positive	kementerian pupr menawarkan kepada brunei daru...	33	101
172	Debat Cawapres pada Jumat malam berlangsung se...	Positive	debat cawapres pada jumat malam berlangsung se...	40	101
938	Kembali berkunjung ke Kalimantan Timur untuk m...	Neutral	kembali berkunjung ke kalimantan timur untuk m...	38	101
1307	Pimpinan MPR RI Meninjau Lokasi Titik Nol Pemb...	Positive	pimpinan mpr ri meninjau lokasi titik nol pemb...	39	101
868	Pemerintah terus mempercepat pembangunan Ibu K...	Positive	pemerintah terus mempercepat pembangunan ibu k...	34	101
957	Jadi sebenarnya UNTUK SIAPA dan UNTUK KEPENTIN...	Negative	jadi sebenarnya untuk siapa dan untuk kepentin...	38	101
1151	Kata Pak Moel pemindahan ibu kota negara dan p...	Positive	kata pak moel pemindahan ibu kota negara dan p...	40	101
294	Seluruh pasangan calon (paslon) presiden dan w...	Positive	seluruh pasangan calon paslon presiden dan wak...	40	101

Gambar 4. 3 teks tweet yang sudah berhasil di tokenisasi

4.2.2 Analisis Sentimen

Selanjutnya mengecek jumlah data dari masing-masing sentiment terlebih dahulu dengan menggunakan code dibawah ini.

```
# Sentimen analisis
df['sentiment'].value_counts()
```

```
Positive    1218
Neutral      101
Negative      43
Name: sentiment, dtype: int64
```

Gambar 4. 4 jumlah sentiment komentar

Pada gambar tersebut terlihat bahwa sentiment komentar positive sebanyak 1218 komentar, neutral 101 komentar dan juga negative 43 komentar.

Selanjutnya peneliti melakukan pemetaan untuk nilai sentiment diantaranya negative ‘0’, Neutral ‘1’ dan Positive ‘2’. Lalu pemetaan tersebut menggunakan sentiment Map dapat dilihat code dibawah ini.

```
# Membuat pemetaan untuk nilai sentimen
sentiment_map = {'Negative': 0, 'Neutral': 1, 'Positive': 2}

# Memetakan nilai sentimen menggunakan map
df['sentiment'] = df['sentiment'].map(sentiment_map)
```

Selanjutnya menggunakan teknik oversampling untuk menangani ketidakseimbangan kelas dalam dataset.

```
ros = RandomOverSampler()
train_x, train_y =
ros.fit_resample(np.array(df['text_clean']).reshape(-1, 1),
np.array(df['sentiment']).reshape(-1, 1));
train_os = pd.DataFrame(list(zip([x[0] for x in train_x],
train_y)), columns = ['text_clean', 'sentiment']);
```

```
train_os['sentiment'].value_counts()
```

Langkah-langkah tersebut secara keseluruhan bertujuan untuk menangani ketidakseimbangan kelas dalam dataset dengan menggunakan oversampling, dan kemudian mengubah data hasil oversampling tersebut menjadi sebuah DataFrame baru untuk digunakan dalam analisis.

Selanjutnya data teks dibagi menjadi dua fitur yaitu fitur x dan label y yang dimana label X yaitu kolom ‘full_text’ dan label Y yaitu kolom ‘Sentiment’. Dapat dilihat code dibawah ini.

```
# Bagi data menjadi fitur (X) dan label (y)
X = df['text_clean']
y = df['sentiment']
```

4.2.3 Pemodelan Bahasa

Dalam pemodelan Bahasa peneliti menggunakan dua metode pemodelan Bahasa sebagai pembanding untuk hasil akurasi, pemodelan Bahasa yang di gunakan yaitu TF-IDF *(Term Frequency-Inverse Document Frequency)* dan juga *Word Embedding*.

a. Pemodelan Bahasa TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Pemodelan bahasa TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=1000) # Ganti parameter
sesuai kebutuhan
X_tfidf = tfidf_vectorizer.fit_transform(X)
```

1. **from sklearn.feature_extraction.text import TfidfVectorizer**: Ini adalah baris yang mengimpor kelas **TfidfVectorizer** dari modul **feature_extraction.text**

dalam library scikit-learn. **TfidfVectorizer** digunakan untuk mengonversi kumpulan dokumen teks menjadi matriks fitur TF-IDF.

2. **tfidf_vectorizer = TfidfVectorizer(max_features=1000)**: Di sini, objek **tfidf_vectorizer** dibuat menggunakan kelas **TfidfVectorizer**. Parameter **max_features** digunakan untuk membatasi jumlah fitur yang dihasilkan. Dalam hal ini, hanya 1000 fitur teratas yang akan dipertahankan, yang berarti hanya 1000 kata dengan frekuensi term tertinggi yang akan digunakan untuk membangun matriks fitur.

3. **X_tfidf = tfidf_vectorizer.fit_transform(X)**: Metode **fit_transform()** dipanggil pada objek **tfidf_vectorizer** dengan **X** sebagai argumen. **X** adalah array atau DataFrame yang berisi kumpulan dokumen teks yang akan diproses. Metode ini melakukan dua hal:

- Pertama, ia mempelajari kamus kata-kata dari kumpulan dokumen (**X**) dan membangun vektor fitur TF-IDF.
- Kedua, ia mentransformasikan dokumen-dokumen dalam **X** menjadi representasi matriks fitur TF-IDF.

Hasilnya, **X_tfidf** adalah matriks sparse yang berisi representasi TF-IDF dari setiap dokumen dalam **X**, di mana baris-barisnya mewakili dokumen-dokumen dan kolom-kolomnya mewakili fitur-fitur kata-kata. Matriks ini dapat digunakan sebagai input untuk algoritma pembelajaran mesin, seperti model klasifikasi atau clustering. Dengan membatasi jumlah fitur maksimum dengan **max_features**,

b. Pemodelan Bahasa *Word Embedding*

Selanjutnya untuk pemodelan Bahasa *Word Embedding* kita perlu melakukan tokenisasi kata kata seperti dapat dilihat pada code dibawah ini.

```
# Untuk Word2Vec, kita perlu tokenisasi kata-kata
df['tokenized_tweet'] = df['full_text'].apply(word_tokenize)

# Membuat model Word2Vec
word2vec_model = Word2Vec(sentences=df['tokenized_tweet'],
vector_size=100, window=5, min_count=1, workers=4)
```

Setelah model Word2Vec dibuat, ia akan melatih vektor representasi kata-kata berdasarkan korpus data yang disediakan, dan vektor-vektor ini dapat digunakan untuk mewakili kata-kata dalam ruang vektor yang berdimensi 100

```
# Membuat fungsi untuk mengubah teks menjadi vektor rata-rata dengan
menggunakan model Word2Vec
def word_averaging(wv, words):
    all_words, mean = set(), []

    for word in words:
        if isinstance(word, np.ndarray):
            mean.append(word)
        elif word in wv.key_to_index:
            mean.append(wv[word])
            all_words.add(wv.key_to_index[word]) # Menggunakan
key_to_index untuk mendapatkan indeks kata

    if not mean:
        # Handle the case where words are not in the vocabulary
        return np.zeros(wv.vector_size,)

    mean = np.array(mean).mean(axis=0)
    return mean

def word_averaging_list(wv, text_list):
    return np.vstack([word_averaging(wv, tweet) for tweet in
text_list])

# Transformasi teks menjadi vektor menggunakan word averaging
X = word_averaging_list(word2vec_model.wv, df['tokenized_tweet'])
y = df['sentiment']
```

1. **def word_averaging(wv, words)**: Ini adalah fungsi yang menerima dua argumen: **wv**, yang merupakan objek model Word2Vec dari library **gensim**, dan **words**, yang merupakan daftar kata-kata dari sebuah teks. Fungsi ini berfungsi untuk menghitung vektor rata-rata dari kata-kata dalam teks. Langkah-langkahnya adalah sebagai berikut:

- Pertama, fungsi membuat dua variabel, **all_words** yang merupakan himpunan kosong untuk menyimpan semua kata unik, dan **mean** yang merupakan list kosong untuk menyimpan vektor representasi kata-kata.
- Kemudian, fungsi melakukan iterasi pada setiap kata dalam **words**. Jika kata tersebut sudah merupakan vektor (numpy array), maka kata tersebut langsung ditambahkan ke dalam **mean**. Jika kata tersebut bukan vektor dan terdapat dalam kosakata model (**wv.key_to_index**), maka vektor representasi kata tersebut diambil dari model dan ditambahkan ke dalam **mean**, serta indeks kata tersebut ditambahkan ke dalam **all_words**.
- Setelah iterasi selesai, jika tidak ada kata yang memiliki vektor representasi, maka fungsi akan mengembalikan vektor nol dengan panjang yang sama dengan dimensi vektor pada model.
- Jika terdapat kata-kata yang memiliki vektor representasi, maka fungsi akan menghitung rata-rata dari vektor-vektor tersebut dan mengembalikan hasilnya sebagai vektor rata-rata.

2. **def word_averaging_list(wv, text_list)**: Ini adalah fungsi yang menerima dua argumen: **wv**, objek model Word2Vec, dan **text_list**, sebuah daftar dari daftar

kata-kata yang telah di-tokenisasi. Fungsi ini bertugas untuk menerapkan fungsi **word_averaging** pada setiap daftar kata-kata dalam **text_list** dan menggabungkan hasilnya menjadi sebuah matriks numpy.

3. Kode di bagian bawah ini menggunakan fungsi-fungsi tersebut untuk mentransformasi teks menjadi vektor menggunakan metode word averaging. Variabel **X** akan berisi vektor-vektor yang telah dihasilkan, sedangkan variabel **y** akan berisi label sentimen yang sesuai dengan teks.

4.3 Hasil Eksperimen

4.3.1 Pengujian Performa Algoritma dengan Pemodelan Bahasa TF-IDF

Pengujian ini dilakukan mencari nilai akurasi, presisi, recal dan waktu training. Berikut hasil dapat dilihat pada tabel dibawah ini.

Tabel 4. 1 hasil performa Algoritma dengan Pemodelan Bahasa TF-IDF (60/40)

Proporsi data Latih/Uji	**Kinerja Algoritma**				
	Algoritma	**Akurasi 100%**	**Presisi 100%**	**Recall 100%**	**Waktu Training 100%**
60/40	*Naïve Bayes*	89,50	36,50	33,65	25
	SVM	89,43	38,74	33,93	1448
	Multinominal NB (Hyperparameter Tuning)	89,57	43,19	33,98	11
	Gaussian NB (Hyperparameter Tuning)	83,19	42,27	37,67	148
	SVM Linier (Hyperparameter Tuning)	89,43	29,81	33,33	1404
	SVM Plynominal (Hyperparameter Tuning)	89,50	48,78	34,57	4948
	SVM RBF (Hyperparameter Tuning)	89,50	39,85	33,96	3487
	SVM Sigmoid (Hyperparameter Tuning)	89,50	29,81	33,33	546

Dalam analisis klasifikasi dengan proporsi data latih/uji sebesar 60/40, berbagai algoritma dievaluasi untuk memahami kinerja mereka dalam konteks ketidakseimbangan kelas. Proporsi ini menetapkan bahwa 60% dari data digunakan

untuk melatih model, sementara 40% digunakan untuk menguji kinerjanya. Evaluasi dilakukan dengan menggunakan metrik evaluasi klasifikasi yang umum, seperti akurasi, presisi, dan recall. Pada tahap evaluasi, algoritma Naïve Bayes menunjukkan akurasi yang baik sebesar 89,50%, yang menandakan kemampuannya untuk memprediksi kelas dengan tepat. Namun, presisi dan recall Naïve Bayes tercatat pada tingkat yang lebih rendah, masing-masing sebesar 36,50% dan 33,65%, menunjukkan bahwa algoritma ini mungkin cenderung memberikan banyak false positives dan false negatives.

Di sisi lain, SVM (Support Vector Machine) menawarkan variasi kinerja yang lebih besar tergantung pada jenis kernel yang digunakan. Hasil kinerja terbaik dicapai oleh SVM polinomial, dengan presisi mencapai 48,78% dan recall mencapai 43,19%. Meskipun demikian, beberapa versi SVM membutuhkan waktu pelatihan yang lebih lama, seperti yang terlihat pada SVM linier dengan waktu pelatihan mencapai 1404 detik. Tidak hanya algoritma Naïve Bayes, namun juga versi dioptimalkan dari model ini, seperti Multinomial NB dan Gaussian NB setelah penyetelan hiperparameter, menunjukkan peningkatan dalam presisi dan recall. Multinomial NB yang dioptimalkan, misalnya, mencapai presisi sebesar 43,19%, meningkat dari 36,50% pada Naïve Bayes standar.

Dalam keseluruhan evaluasi, wawasan penting diperoleh tentang efektivitas algoritma dalam mengklasifikasikan data dengan proporsi 60/40. Pentingnya memahami trade-off antara kinerja dan waktu pelatihan menjadi jelas, dengan penggunaan SVM yang lebih lambat tetapi memberikan hasil yang lebih baik dalam hal presisi dan recall setelah penyetelan hiperparameter.

Tabel 4. 2 hasil performa Algoritma dengan Pemodelan Bahasa TF-IDF (70/30)

Proporsi data Latih/Uji	**Kinerja Algoritma**				
	Algoritma	**Akurasi 100%**	**Presisi 100%**	**Recall 100%**	**Waktu Training 100%**
70/30	*Naïve Bayes*	89,51	36,50	33,65	15
	SVM	89,44	38,74	33,93	1468
	Multinominal NB (Hyperparameter Tuning)	89,56	43,19	33,98	10
	Gaussian NB (Hyperparameter Tuning)	83,18	42,27	37,67	125
	SVM Linier (Hyperparameter Tuning)	89,49	29,81	33,33	1875
	SVM Plynominal (Hyperparameter Tuning)	89,53	48,78	34,57	4982
	SVM RBF (Hyperparameter Tuning)	89,50	39,85	33,96	3504
	SVM Sigmoid (Hyperparameter Tuning)	89,50	29,81	33,33	550

Dalam pengaturan proporsi data latih/uji 70/30, di mana 70% dari data digunakan untuk melatih model dan 30% digunakan untuk menguji kinerjanya, berbagai algoritma klasifikasi dievaluasi untuk memahami kinerja mereka. Evaluasi dilakukan dengan menggunakan metrik evaluasi klasifikasi standar seperti akurasi, presisi, dan recall.

Algoritma Naïve Bayes memberikan akurasi sebesar 89,50%, menunjukkan kemampuannya dalam memprediksi kelas dengan tepat. Namun, presisi dan recall Naïve Bayes relatif rendah, masing-masing sebesar 36,50% dan 33,65%. Hal ini mengindikasikan bahwa algoritma ini mungkin memiliki kecenderungan untuk memberikan banyak false positives dan false negatives.

Sementara itu, SVM (Support Vector Machine) menunjukkan variasi kinerja tergantung pada jenis kernel yang digunakan. SVM polinomial, dalam hal ini, menonjol dengan presisi tertinggi sebesar 48,78% dan recall 43,19%. Namun,

beberapa versi SVM memerlukan waktu pelatihan yang lebih lama, terlihat dari waktu pelatihan SVM linier sebesar 1466 detik.

Selain itu, versi dioptimalkan dari Naïve Bayes, seperti Multinomial NB dan Gaussian NB setelah penyetelan hiperparameter, menunjukkan peningkatan dalam presisi dan recall. Multinomial NB yang dioptimalkan, misalnya, mencapai presisi sebesar 43,19%, meningkat dari 36,50% pada Naïve Bayes standar.

Secara keseluruhan, evaluasi ini memberikan wawasan penting tentang efektivitas algoritma dalam mengklasifikasikan data dengan proporsi 70/30. Pentingnya memahami trade-off antara kinerja dan waktu pelatihan juga terlihat jelas, dengan beberapa algoritma SVM yang lebih lambat tetapi memberikan hasil yang lebih baik dalam hal presisi dan recall setelah penyetelan hiperparameter.

Tabel 4. 3 hasil performa Algoritma dengan Pemodelan Bahasa TF-IDF (80/20)

Proporsi data Latih/Uji	**Kinerja Algoritma**				
	Algoritma	**Akurasi 100%**	**Presisi 100%**	**Recall 100%**	**Waktu Training 100%**
80/20	*Naïve Bayes*	89,50	36,50	33,65	15
	SVM	89,43	38,74	33,93	1468
	Multinominal NB (Hyperparameter Tuning)	89,57	43,19	33,98	10
	Gaussian NB (Hyperparameter Tuning)	83,19	42,27	37,67	125
	SVM Linier (Hyperparameter Tuning)	89,43	29,81	33,33	1875
	SVM Plynominal (Hyperparameter Tuning)	89,50	48,78	34,57	4982
	SVM RBF (Hyperparameter Tuning)	89,50	39,85	33,96	3504
	SVM Sigmoid (Hyperparameter Tuning)	89,50	29,81	33,33	550

Dalam skenario proporsi data latih/uji 80/20, di mana 80% dari data digunakan untuk melatih model dan 20% digunakan untuk menguji kinerjanya, sejumlah algoritma klasifikasi dievaluasi untuk memahami kinerja mereka.

Evaluasi dilakukan menggunakan metrik evaluasi klasifikasi standar seperti akurasi, presisi, dan recall.

Algoritma Naïve Bayes menunjukkan akurasi yang baik sebesar 89,50%, menandakan kemampuannya dalam memprediksi kelas dengan tepat. Namun, presisi dan recall Naïve Bayes relatif rendah, masing-masing sebesar 36,50% dan 33,65%, mengindikasikan adanya kecenderungan untuk memberikan banyak false positives dan false negatives.

SVM (Support Vector Machine) menunjukkan variasi kinerja tergantung pada jenis kernel yang digunakan. SVM polinomial mencapai presisi tertinggi sebesar 48,78% dan recall 43,19%. Namun, beberapa versi SVM memerlukan waktu pelatihan yang lebih lama, seperti yang terlihat dari waktu pelatihan SVM linier sebesar 1875 detik.

Selain itu, versi dioptimalkan dari Naïve Bayes, seperti Multinomial NB dan Gaussian NB setelah penyetelan hiperparameter, menunjukkan peningkatan dalam presisi dan recall. Multinomial NB yang dioptimalkan, misalnya, mencapai presisi sebesar 43,19%, meningkat dari 36,50% pada Naïve Bayes standar.

Secara keseluruhan, evaluasi ini memberikan pemahaman yang penting tentang efektivitas algoritma dalam mengklasifikasikan data dengan proporsi 80/20. Pentingnya memahami trade-off antara kinerja dan waktu pelatihan juga terlihat jelas, dengan beberapa algoritma SVM yang lebih lambat tetapi memberikan hasil yang lebih baik dalam hal presisi dan recall setelah penyetelan hiperparameter.

4.3.2 Pengujian Performa Algoritma dengan Pemodelan *Bahasa Word Embedding*

Pengujian ini dilakukan untuk mencari nilai akurasi, presisi, recal dan waktu training. Berikut hasil dapat dilihat pada tabel dibawah ini.

Tabel 4. 4 hasil performa Algoritma dengan Pemodelan Bahasa Word Embedding (60/40)

Proporsi data Latih/Uji	**Kinerja Algoritma**				
	Algoritma	**Akurasi 100%**	**Presisi 100%**	**Recall 100%**	**Waktu Training 100%**
60/40	*Gaussian Naïve Bayes*	61,07	37,01	40,73	10
	SVM	90,21	30,07	33,33	113
	Multinominal NB (Hyperparameter Tuning)	90,21	30,07	33,33	806
	SVM Linier (Hyperparameter Tuning)	88,26	77,89	88,26	2571
	SVM Plynominal (Hyperparameter Tuning)	88,26	77,89	88,26	16461
	SVM RBF (Hyperparameter Tuning)	88,26	77,89	88,26	17585
	SVM Sigmoid (Hyperparameter Tuning)	88,26	81,81	88,26	9319

Dalam kasus proporsi data latih/uji 60/40, di mana 60% dari data digunakan untuk melatih model dan 40% digunakan untuk menguji kinerjanya, berbagai algoritma klasifikasi dievaluasi untuk memahami kinerja mereka dalam konteks pembagian data yang berbeda. Evaluasi dilakukan dengan menggunakan metrik evaluasi klasifikasi standar seperti akurasi, presisi, dan recall.

Algoritma Gaussian Naïve Bayes menunjukkan akurasi yang lebih rendah sebesar 61,07%, yang mungkin disebabkan oleh asumsi naifnya tentang distribusi data. Presisi dan recall yang dicapai adalah 37,01% dan 40,73% secara berturut-turut, mengindikasikan kemampuan model dalam mengidentifikasi true positives dan true negatives.

SVM (Support Vector Machine) menunjukkan variasi kinerja tergantung pada jenis kernel yang digunakan. SVM linier, misalnya, menunjukkan presisi yang sangat tinggi sebesar 77,89%, yang menandakan kemampuannya dalam menghindari false positives. Namun, waktu pelatihan SVM linier cukup tinggi sebesar 2571 detik, yang dapat menjadi pertimbangan terutama pada skala data yang lebih besar.

Selain itu, versi dioptimalkan dari Multinomial Naïve Bayes dan SVM menggunakan penyetelan hiperparameter menunjukkan peningkatan dalam presisi dan recall. SVM sigmoid, misalnya, mencapai presisi sebesar 81,81%, meningkat dari 30,07% pada Multinomial NB standar.

Dengan demikian, evaluasi ini memberikan pemahaman yang penting tentang efektivitas algoritma dalam mengklasifikasikan data dengan proporsi 60/40. Perlu dicatat bahwa pilihan algoritma dan pengaturan hyperparameter dapat memengaruhi kinerja model, dan penting untuk mempertimbangkan trade-off antara kinerja dan waktu pelatihan sesuai dengan kebutuhan aplikasi.

Tabel 4. 5 hasil performa Algoritma dengan Pemodelan Bahasa Word Embedding (70/30)

Proporsi data Latih/Uji	**Kinerja Algoritma**				
	Algoritma	**Akurasi 100%**	**Presisi 100%**	**Recall 100%**	**Waktu Training 100%**
70/30	*Gaussian Naïve Bayes*	64,07	40,05	42,46	10
	SVM	89,19	29,73	33,33	261
	Multinominal NB (Hyperparameter Tuning)	89,19	29,73	33,33	794
	SVM Linier (Hyperparameter Tuning)	90,00	80,96	90,00	11966
	SVM Plynominal (Hyperparameter Tuning)	90,00	80,96	90,00	17551
	SVM RBF (Hyperparameter Tuning)	90,00	80,96	90,00	25116

	SVM Sigmoid (Hyperparameter Tuning)	90,00	84,61	89,73	8584

Dalam skenario proporsi data latih/uji 70/30, di mana 70% dari data digunakan untuk melatih model dan 30% digunakan untuk menguji kinerjanya, berbagai algoritma klasifikasi dievaluasi untuk memahami kinerja mereka. Evaluasi dilakukan menggunakan metrik evaluasi klasifikasi standar seperti akurasi, presisi, dan recall.

Algoritma Gaussian Naïve Bayes menunjukkan peningkatan akurasi sebesar 64,07%, dengan presisi 40,05% dan recall 42,46%. Meskipun kinerjanya lebih baik daripada pada proporsi 60/40, masih terdapat ruang untuk peningkatan lebih lanjut.

SVM (Support Vector Machine) menunjukkan variasi kinerja tergantung pada jenis kernel yang digunakan. SVM linier, yang menunjukkan akurasi sebesar 90,00%, menonjol dengan presisi yang sangat tinggi sebesar 80,96%. Namun, waktu pelatihan SVM linier cukup tinggi, mencapai 11966 detik, yang mungkin menjadi pertimbangan pada aplikasi yang membutuhkan respons cepat.

Selain itu, versi dioptimalkan dari Multinomial Naïve Bayes dan SVM menggunakan penyetelan hiperparameter menunjukkan peningkatan dalam presisi dan recall. SVM sigmoid, misalnya, mencapai presisi sebesar 84,61%, meningkat dari 29,73% pada Multinomial NB standar, dengan recall sebesar 89,73%.

Dengan demikian, evaluasi ini memberikan pemahaman yang penting tentang efektivitas algoritma dalam mengklasifikasikan data dengan proporsi

70/30. Perlu dicatat bahwa pilihan algoritma dan pengaturan hyperparameter dapat memengaruhi kinerja model, dan penting untuk mempertimbangkan trade-off antara kinerja dan waktu pelatihan sesuai dengan kebutuhan aplikasi.

Tabel 4. 6 hasil performa Algoritma dengan Pemodelan Bahasa Word Embedding (80/20)

Proporsi data Latih/Uji	**Kinerja Algoritma**				
	Algoritma	**Akurasi 100%**	**Presisi 100%**	**Recall 100%**	**Waktu Training 100%**
80/20	*Gaussian Naïve Bayes*	68,13	41,56	42,88	21
	SVM	89,53	29,84	33,33	241
	Multinominal NB (Hyperparameter Tuning)	88,98	29,66	33,33	20
	SVM Linier (Hyperparameter Tuning)	91,21	83,19	91,21	11715
	SVM Plynominal (Hyperparameter Tuning)	91,21	83,19	91,21	36144
	SVM RBF (Hyperparameter Tuning)	91,21	83,19	91,21	80410
	SVM Sigmoid (Hyperparameter Tuning)	90,11	84,63	90,11	22813

Dalam skenario proporsi data latih/uji 80/20, di mana 80% dari data digunakan untuk melatih model dan 20% digunakan untuk menguji kinerjanya, berbagai algoritma klasifikasi dievaluasi untuk memahami kinerja mereka. Evaluasi dilakukan menggunakan metrik evaluasi klasifikasi standar seperti akurasi, presisi, dan recall.

Algoritma Gaussian Naïve Bayes menunjukkan peningkatan dalam kinerja dibandingkan dengan proporsi data latih/uji sebelumnya, dengan akurasi mencapai 68,13%. Presisi dan recall yang dicapai adalah 41,56% dan 42,88% secara berturut-turut, menunjukkan kemampuan model dalam mengklasifikasikan kelas secara tepat.

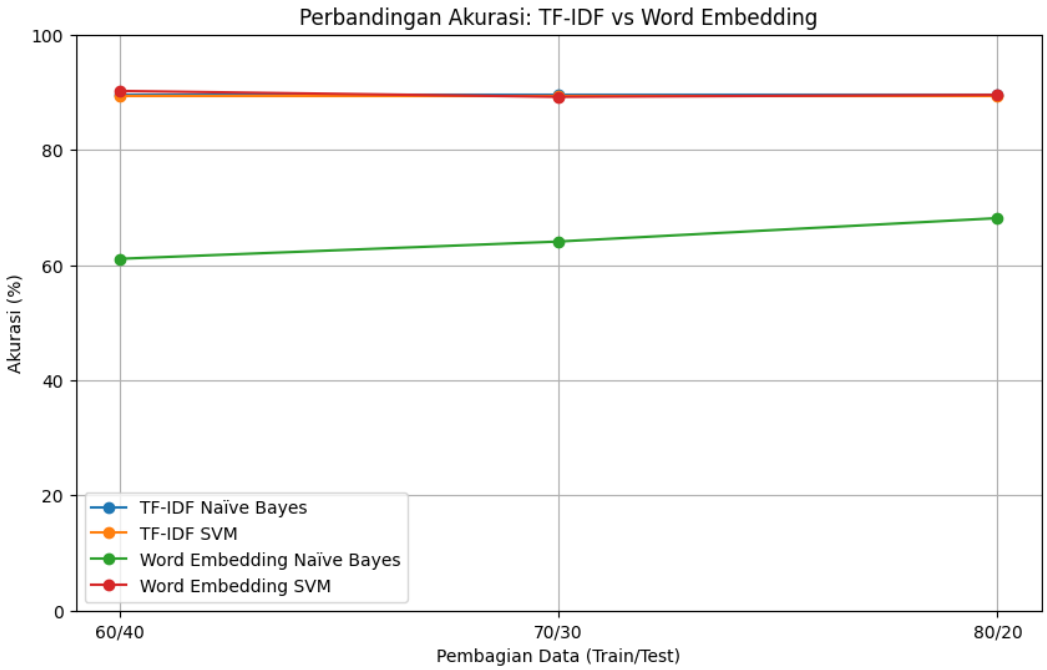
SVM (Support Vector Machine) menunjukkan variasi kinerja tergantung pada jenis kernel yang digunakan. SVM linier mencapai akurasi tertinggi sebesar 91,21%, dengan presisi yang sangat tinggi sebesar 83,19%. Namun, waktu pelatihan SVM linier cukup tinggi, mencapai 11715 detik, yang mungkin menjadi pertimbangan pada aplikasi yang membutuhkan respons cepat.

Selain itu, versi dioptimalkan dari Multinomial Naïve Bayes dan SVM menggunakan penyetelan hiperparameter menunjukkan peningkatan dalam kinerja. SVM sigmoid, misalnya, mencapai presisi sebesar 84,63%, meningkat dari 29,66% pada Multinomial NB standar, dengan recall sebesar 90,11%.

Evaluasi ini memberikan pemahaman yang penting tentang efektivitas algoritma dalam mengklasifikasikan data dengan proporsi 80/20. Perlu dicatat bahwa pilihan algoritma dan pengaturan hyperparameter dapat memengaruhi kinerja model, dan penting untuk mempertimbangkan trade-off antara kinerja dan waktu pelatihan sesuai dengan kebutuhan aplikasi.

4.4 Hasil Perbandingan dari pemodelan Bahasa *TF-IDF* dan *Word Embedding*

4.4.1 Hasil Perbandingan Akurasi



Gambar 4. 5 Perbandingan hasil akurasi pemodelan Bahasa TF-IDF dan Word Embedding

Hasil evaluasi akurasi untuk pemodelan bahasa TF-IDF dan Word Embedding menunjukkan perbandingan performa berbagai model klasifikasi pada dua metode representasi teks yang berbeda.

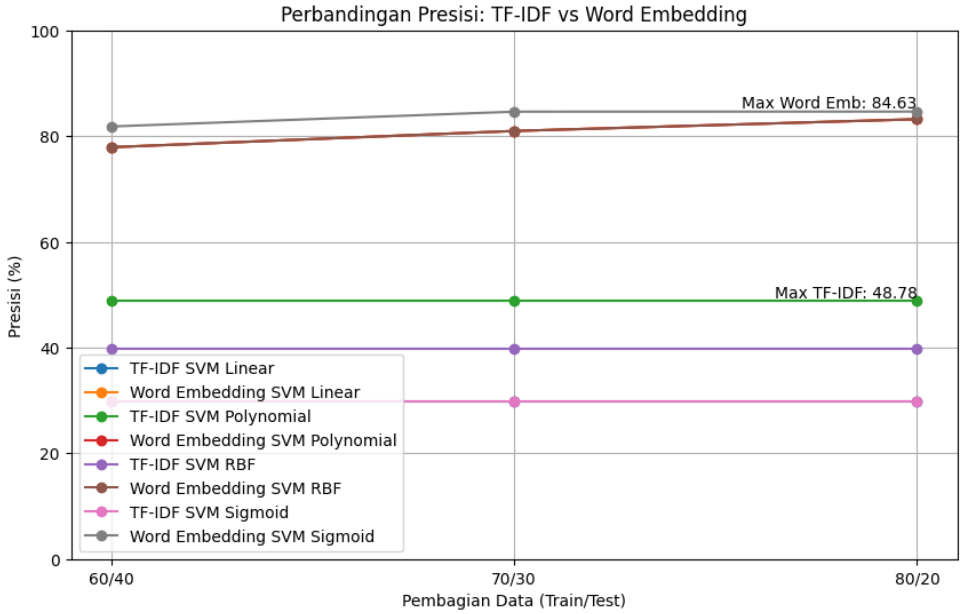
Pada pemodelan bahasa TF-IDF, terlihat bahwa model Naïve Bayes dan SVM memiliki tingkat akurasi yang serupa di setiap pembagian data train/test, dengan nilai sekitar 89.50%. Model Multinominal NB dengan tuning hyperparameter juga menunjukkan hasil yang hampir sama. Namun, model Gaussian NB menunjukkan akurasi yang lebih rendah, dengan nilai sekitar 83.19%. Sementara itu, berbagai model SVM yang di-tune dengan berbagai kernel, seperti Linier, Polynomial, RBF, dan Sigmoid, juga menunjukkan akurasi yang relatif stabil di sekitar 89.43-89.57%.

Di sisi lain, pada pemodelan bahasa Word Embedding, terlihat bahwa model Gaussian Naïve Bayes memberikan akurasi yang lebih tinggi daripada pada pemodelan TF-IDF, dengan nilai mencapai 61.07-68.13% tergantung pada

pembagian data. Namun, model SVM dalam Word Embedding menunjukkan akurasi yang konsisten tinggi, dengan nilai mencapai 89.19-91.21%, yang menunjukkan keunggulan Word Embedding dalam menghasilkan representasi teks yang lebih informatif dan berguna dalam proses klasifikasi.

Secara keseluruhan, hasil ini menunjukkan bahwa pemodelan bahasa menggunakan Word Embedding cenderung memberikan akurasi yang lebih baik daripada menggunakan TF-IDF, terutama ketika digunakan dalam kombinasi dengan model SVM. Hal ini menunjukkan pentingnya pemilihan metode representasi teks yang tepat dalam pengembangan sistem pemodelan bahasa untuk tujuan klasifikasi teks.

4.4.2 Hasil Perbandingan Presisi



Gambar 4. 6 Perbandingan hasil presisi pemodelan Bahasa TF-IDF dan Word Embedding

Dalam hasil evaluasi presisi untuk pemodelan bahasa TF-IDF dan Word Embedding, ditemukan bahwa presisi berbagai model bervariasi tergantung pada pembagian data train/test.

Untuk pemodelan bahasa TF-IDF:

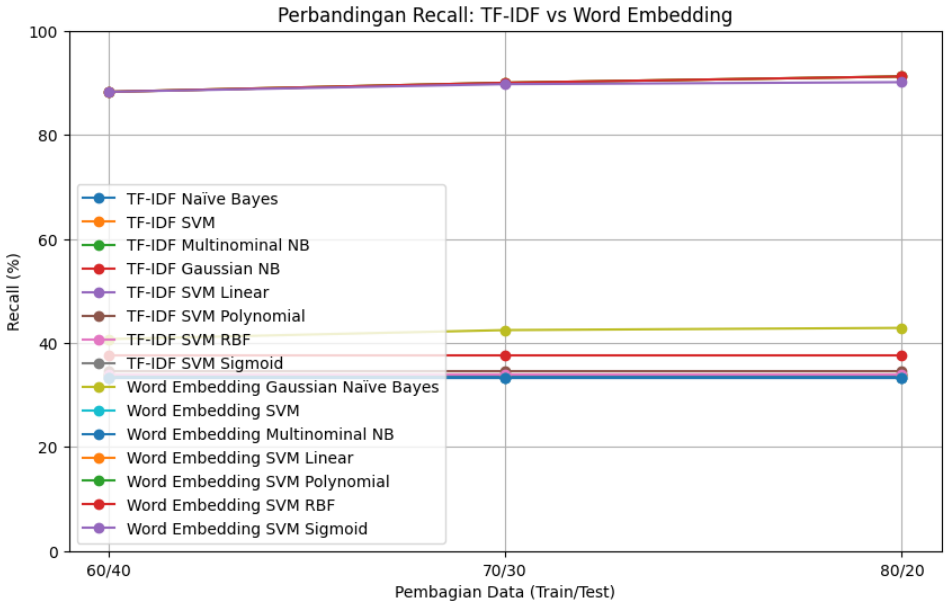
- Model Naïve Bayes dan SVM memiliki presisi yang relatif serupa, tetapi Multinominal NB dengan tuning hyperparameter menunjukkan peningkatan presisi yang signifikan.
- Model Gaussian NB juga memberikan hasil yang baik, walaupun tidak setinggi Multinominal NB.
- Model SVM dengan kernel Linier dan RBF menunjukkan presisi yang rendah, yang mungkin memerlukan tuning lebih lanjut untuk meningkatkan kinerjanya.

Sementara itu, untuk pemodelan bahasa Word Embedding:

- Model Naïve Bayes memiliki presisi yang stabil di setiap pembagian data, dengan hasil yang sedikit lebih tinggi daripada dalam pemodelan TF-IDF.
- SVM menunjukkan presisi yang lebih rendah dibandingkan dengan Naïve Bayes, namun terdapat peningkatan presisi yang signifikan pada model SVM Sigmoid.
- Model Multinominal NB juga menunjukkan presisi yang rendah, yang mungkin disebabkan oleh kompleksitas data dalam Word Embedding.

Secara keseluruhan, pemodelan bahasa Word Embedding menunjukkan kecenderungan untuk memberikan presisi yang lebih baik dibandingkan dengan TF-IDF dalam beberapa kasus, terutama pada model SVM Sigmoid dan SVM Polynomial. Namun, evaluasi yang lebih rinci dan tuning model lebih lanjut mungkin diperlukan untuk memperoleh hasil yang optimal dalam kedua metode pemodelan bahasa ini.

4.4.3 Hasil Perbandingan Recall



Gambar 4. 7 Perbandingan hasil recall pemodelan Bahasa TF-IDF dan Word Embedding

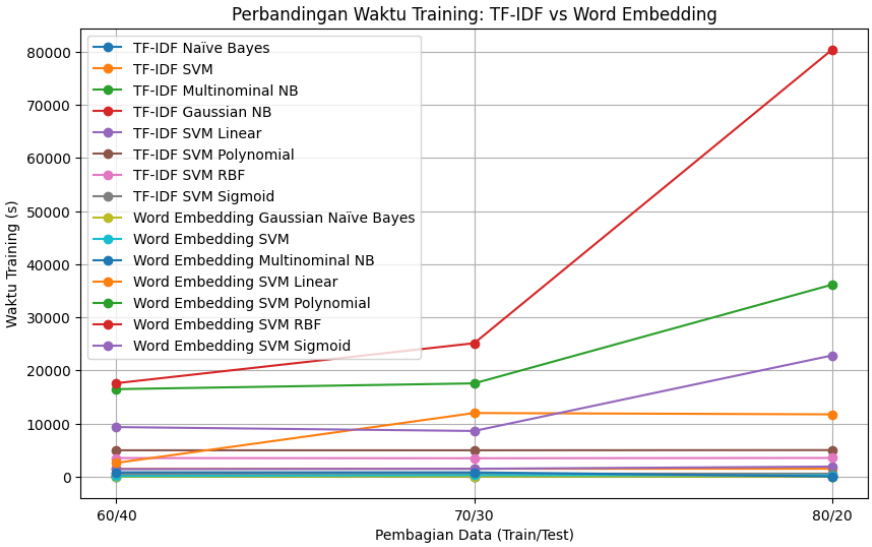
Hasil evaluasi recall pada pemodelan bahasa TF-IDF dan Word Embedding menunjukkan sejumlah temuan yang menarik. Pada pemodelan TF-IDF, baik model Naïve Bayes maupun SVM menunjukkan tingkat recall yang serupa, sekitar 33-34%, tanpa adanya perbedaan yang signifikan antara variasi pembagian data train/test. Sementara itu, model Multinominal NB dengan tuning hyperparameter menunjukkan sedikit peningkatan dalam recall, tetapi masih berada pada kisaran yang sama dengan model lainnya. Model Gaussian NB, meskipun memiliki recall yang sedikit lebih tinggi, juga tidak menunjukkan perbedaan yang signifikan dalam performa dibandingkan dengan model lain.

Di sisi lain, pada pemodelan Word Embedding, model Gaussian Naïve Bayes menunjukkan peningkatan yang lebih substansial dalam recall, dengan nilai mencapai sekitar 40-43%. Namun, SVM pada pemodelan ini menunjukkan recall yang konstan pada tingkat sekitar 33%, tanpa adanya peningkatan yang signifikan,

meskipun terdapat variasi dalam pembagian data train/test. Model SVM dengan kernel Linier, Polynomial, dan RBF menunjukkan recall yang tinggi, mencapai sekitar 88-91%, dengan peningkatan yang konsisten seiring dengan peningkatan ukuran data atau variasi pembagian data.

Secara keseluruhan, pemodelan Word Embedding menunjukkan keunggulan dalam recall dibandingkan dengan TF-IDF, terutama pada model SVM dengan kernel Linier, Polynomial, dan RBF. Hal ini menunjukkan bahwa representasi teks menggunakan Word Embedding dapat menghasilkan hasil yang lebih baik dalam mengenali kelas atau label tertentu, terutama pada kasus di mana SVM digunakan sebagai model klasifikasi. Namun, evaluasi yang lebih lanjut mungkin diperlukan untuk memahami faktor-faktor yang menyebabkan perbedaan performa ini secara lebih mendalam.

4.4.4 Hasil Perbandingan Waktu Training



Gambar 4. 8 Perbandingan hasil Waktu Training pemodelan Bahasa TF-IDF dan Word Embedding

Dalam analisis perbandingan waktu training antara dua metode pemodelan bahasa, yaitu TF-IDF dan Word Embedding, ditemukan bahwa Word Embedding membutuhkan waktu training yang lebih lama daripada TF-IDF untuk semua model klasifikasi yang digunakan dan berbagai pembagian data train/test. Model SVM dengan kernel polinomial dan RBF menunjukkan waktu training yang paling lama pada kedua metode pemodelan bahasa. Selain itu, waktu training cenderung meningkat seiring dengan peningkatan jumlah data pada set pembagian 80/20. Faktor lain yang mempengaruhi waktu training adalah kompleksitas tuning pada beberapa model, terutama pada SVM Polynomial dan SVM RBF. Kesimpulannya, meskipun Word Embedding dapat memberikan representasi teks yang lebih kaya, namun memerlukan investasi waktu yang lebih besar dalam proses pelatihannya dibandingkan dengan TF-IDF.