pr-day24-welly-oktariana

April 10, 2024

1 PR DAY 24

- Buatlah sebuah model analisis sentiment untuk sebuah film yang baru dirilis.
- Tugas dikumpulkan dalam bentuk PDF, yang di dalamnya meliputi: Step by step Jawaban atas pertanyaan:

1. Latar belakang pemilihan algoritma yang digunakan
2. Hasil evaluasi
3. Langkah-langkah untuk meningkatkan akurasi model
4. Link repository github (code, dataset, dan model)

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import requests
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[20]:
```
data = pd.read_excel(r'/content/drive/MyDrive/Machine learning AI batch 2/DAY24/
↪assignment/review_172days.xlsx')
data.head()
```

[20]:

	Text_Tweet	Sentiment
0	172 days bener bener ya definisi abis di naiki…	Netral
1	Udah siap dibikin nangis nonton cerita Amer &a…	Netral
2	172 DAYS udah tayang di Netflix! Mengangkat ki…	Netral
3	AAAAAA SIAPA BUAT CERITA 172 DAYS NI https://t…	Positif
4	emosi nya I tgk 172 days ni	Netral

1.1 Text Processing

1.1.1 Cleaning Text and Lower Case

[5]:
```
def cleaning_text(text):
    # remove url
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    text = url_pattern.sub(r'', text)

    # remove hashtags
    # only removing the hash # sign from the word
    text = re.sub(r'#', '', text)

    # remove mention handle user (@)
    text = re.sub(r'@[\w]*', ' ', text)

    # remove emojis
    emoji_pattern = re.compile(
        '['
        '\U0001F600-\U0001F64F'  # emoticons
        '\U0001F300-\U0001F5FF'  # symbols & pictographs
        '\U0001F680-\U0001F6FF'  # transport & map symbols
        '\U0001F700-\U0001F77F'  # alchemical symbols
        '\U0001F780-\U0001F7FF'  # Geometric Shapes Extended
        '\U0001F800-\U0001F8FF'  # Supplemental Arrows-C
        '\U0001F900-\U0001F9FF'  # Supplemental Symbols and Pictographs
        '\U0001FA00-\U0001FA6F'  # Chess Symbols
        '\U0001FA70-\U0001FAFF'  # Symbols and Pictographs Extended-A
        '\U00002702-\U000027B0'  # Dingbats
        '\U000024C2-\U0001F251'
        ']+',
        flags=re.UNICODE
    )
    text = emoji_pattern.sub(r'', text)

    # remove punctuation
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
    for x in text.lower():
        if x in punctuations:
            text = text.replace(x, " ")

    # remove extra whitespace
    text = ' '.join(text.split())

    # lowercase
    text = text.lower()
    return text
```

1.1.2 Remove Stopword

[6]:
```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')

# CONSTRUCT STOPWORDS
rama_stopword = "https://raw.githubusercontent.com/ramaprakoso/
 ↪analisis-sentimen/master/kamus/stopword.txt"
yutomo_stopword = "https://raw.githubusercontent.com/yasirutomo/
 ↪python-sentianalysis-id/master/data/feature_list/stopwordsID.txt"
fpmipa_stopword = "https://raw.githubusercontent.com/onlyphantom/elangdev/
 ↪master/elang/word2vec/utils/stopwords-list/fpmipa-stopwords.txt"
sastrawi_stopword = "https://raw.githubusercontent.com/onlyphantom/elangdev/
 ↪master/elang/word2vec/utils/stopwords-list/sastrawi-stopwords.txt"
aliakbar_stopword = "https://raw.githubusercontent.com/onlyphantom/elangdev/
 ↪master/elang/word2vec/utils/stopwords-list/aliakbars-bilp.txt"
pebahasa_stopword = "https://raw.githubusercontent.com/onlyphantom/elangdev/
 ↪master/elang/word2vec/utils/stopwords-list/pebbie-pebahasa.txt"
elang_stopword = "https://raw.githubusercontent.com/onlyphantom/elangdev/master/
 ↪elang/word2vec/utils/stopwords-id.txt"
nltk_stopword = stopwords.words('indonesian')

# create path url for each stopword
path_stopwords = [rama_stopword, yutomo_stopword, fpmipa_stopword,␣
 ↪sastrawi_stopword,
                  aliakbar_stopword, pebahasa_stopword, elang_stopword]

# combine stopwords
stopwords_l = nltk_stopword
for path in path_stopwords:
    response = requests.get(path)
    stopwords_l += response.text.split('\n')

custom_st = '''
yg yang dgn ane smpai bgt gua gwa si tu ama utk udh btw
ntar lol ttg emg aj aja tll sy sih kalo nya trsa mnrt nih
ma dr ajaa tp akan bs bikin kta pas pdahl bnyak guys abis tnx
bang banget nang mas amat bangettt tjoy hemm haha sllu hrs lanjut
bgtu sbnrnya trjadi bgtu pdhl sm plg skrg
'''

# create dictionary with unique stopword
st_words = set(stopwords_l)
custom_stopword = set(custom_st.split())
```

```
# result stopwords
stop_words = st_words | custom_stopword
print(f'Stopwords: {list(stop_words)[:5]}')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
Stopwords: ['seingat', 'mana', 'jelaskan', 'tersebut', 'sebenarnya']
```

[7]:
```
# remove stopwords
from nltk import word_tokenize, sent_tokenize

def remove_stopword(text, stop_words=stop_words):
    word_tokens = word_tokenize(text)
    filtered_sentence = [w for w in word_tokens if not w in stop_words]
    return ' '.join(filtered_sentence)
```

1.1.3 Stemming / Lemmatization

[9]:
```
!pip install sastrawi -q
```

```
                                        209.7/209.7
kB 4.4 MB/s eta 0:00:00
```

[10]:
```
# stemming and lemmatization
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

def stemming_and_lemmatization(text):
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()
    return stemmer.stem(text)
```

1.1.4 Tokenization

[12]:
```
# tokenization
def tokenize(text):
    return word_tokenize(text)
```

[13]:
```
# example
text = 'Agak Laen ini emang agak lain, ya.. Bisa-bisanya jadi film Indonesia␣
 ↪kedua terlaris   #respect https://x.com/alnurulg/status/1761301921846140991?
 ↪s=20'
print(f'Original text: \n{text}\n')
```

```
# cleaning text and lowercase
text = cleaning_text(text)
print(f'Cleaned text: \n{text}\n')

# remove stopwords
text = remove_stopword(text)
print(f'Removed stopword: \n{text}\n')

# stemming and lemmatization
text = stemming_and_lemmatization(text)
print(f'Stemmed and lemmatized: \n{text}\n')

# tokenization
text = tokenize(text)
print(f'Tokenized: \n{text}')
```

```
Original text:
Agak Laen ini emang agak lain, ya.. Bisa-bisanya jadi film Indonesia kedua
terlaris   #respect https://x.com/alnurulg/status/1761301921846140991?s=20

Cleaned text:
agak laen ini emang agak lain ya bisa bisanya jadi film indonesia kedua terlaris
respect

Removed stopword:
laen bisanya film indonesia terlaris respect

Stemmed and lemmatized:
laen bisa film indonesia laris respect

Tokenized:
['laen', 'bisa', 'film', 'indonesia', 'laris', 'respect']
```

[16]:
```
# pipeline preprocess
def preprocess(text):
    # cleaning text and lowercase
    output = cleaning_text(text)

    # remove stopwords
    output = remove_stopword(output)

    # # stemming and lemmatization
    # output = stemming_and_lemmatization(output)

    # # tokenization
    # output = tokenize(output)
```

```
    return output
```

[18]:
```
data
```

[18]:
```
                                          full_text Unnamed: 1
0    172 days bener bener ya definisi abis di naiki…     Netral
1    Udah siap dibikin nangis nonton cerita Amer &a…     Netral
2    172 DAYS udah tayang di Netflix! Mengangkat ki…     Netral
3    AAAAAA SIAPA BUAT CERITA 172 DAYS NI https://t…    Positif
4                          emosi nya I tgk 172 days ni      Netral
..                                                 …        …
135  @whosjaygf AWCH salting deh hari ini aku senen…     Netral
136               @sweetchcco ternyata 172 days baguss    Positif
137  kan baru sadar aku rumah di santri pilihan bun…     Netral
138              @sendalkukus gan coba nonton 172 days     Netral
139      @WatchmenID Bukan… 172 days (later) ya Min?     Netral

[140 rows x 2 columns]
```

[23]:
```
preprocessed_data = data.copy()
preprocessed_data['Text Tweet'] = data['Text_Tweet'].map(preprocess)
```

[24]:
```
preprocessed_data.tail()
```

[24]:
```
                                        Text_Tweet Sentiment  \
135  @whosjaygf AWCH salting deh hari ini aku senen…    Netral
136               @sweetchcco ternyata 172 days baguss   Positif
137  kan baru sadar aku rumah di santri pilihan bun…    Netral
138              @sendalkukus gan coba nonton 172 days    Netral
139      @WatchmenID Bukan… 172 days (later) ya Min?    Netral

                                        Text Tweet
135  awch salting seneng nonton film smkeluarga kar…
136                                  172 days baguss
137       sadar rumah santri pilihan bunda ck 172 days
138                           gan coba nonton 172 days
139                                 172 days later min
```

[25]:
```
preprocessed_data['Text Tweet'][0]
```

[25]:
```
'172 days bener bener definisi naikin tingginya dijatuhin sejatuh jatuhnya'
```

[29]:
```
df = preprocessed_data[['Text Tweet', 'Sentiment']]
```

[36]:
```
df.shape
```

[36]:
```
(140, 2)
```

1.2 LSTM

Dalam model sentiment analysis ini menggunakan model LSTM. LSTM mampu menangani sequence yang panjang dan kompleks, LSTM juga memiliki kemampuan untuk mengingat informasi jangka panjang.

[30]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense

# Assuming the 'Text Tweet' column contains the text data and 'Sentiment'␣
 ↪contains labels
texts = df['Text Tweet'].tolist()
labels = df['Sentiment'].tolist()

# Tokenize the text data
max_words = 10000  # Adjust based on your dataset size
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

# Pad sequences to make them of equal length
max_sequence_length = 100  # Adjust based on your dataset and sequence length
data = pad_sequences(sequences, maxlen=max_sequence_length)

# Convert labels to one-hot encoding
labels = pd.get_dummies(labels)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.
 ↪2, random_state=42)

# Build the LSTM model
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=100,␣
 ↪input_length=max_sequence_length))
model.add(LSTM(units=64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(units=len(labels.columns), activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test,␣
 ↪y_test))
```

```python

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

```
Epoch 1/100
4/4 [==============================] - 5s 327ms/step - loss: 1.0877 - accuracy:
0.4911 - val_loss: 1.0827 - val_accuracy: 0.4643
Epoch 2/100
4/4 [==============================] - 1s 163ms/step - loss: 1.0282 - accuracy:
0.5893 - val_loss: 1.0675 - val_accuracy: 0.4643
Epoch 3/100
4/4 [==============================] - 1s 168ms/step - loss: 0.9611 - accuracy:
0.5893 - val_loss: 1.1703 - val_accuracy: 0.4643
Epoch 4/100
4/4 [==============================] - 1s 180ms/step - loss: 0.9295 - accuracy:
0.5893 - val_loss: 1.1187 - val_accuracy: 0.4643
Epoch 5/100
4/4 [==============================] - 1s 177ms/step - loss: 0.9028 - accuracy:
0.5893 - val_loss: 1.0615 - val_accuracy: 0.4643
Epoch 6/100
4/4 [==============================] - 1s 157ms/step - loss: 0.8792 - accuracy:
0.5982 - val_loss: 1.0495 - val_accuracy: 0.4643
Epoch 7/100
4/4 [==============================] - 1s 161ms/step - loss: 0.8500 - accuracy:
0.6250 - val_loss: 1.0687 - val_accuracy: 0.4643
Epoch 8/100
4/4 [==============================] - 1s 253ms/step - loss: 0.8222 - accuracy:
0.6250 - val_loss: 1.0798 - val_accuracy: 0.4643
Epoch 9/100
4/4 [==============================] - 1s 325ms/step - loss: 0.7819 - accuracy:
0.6339 - val_loss: 1.0370 - val_accuracy: 0.4643
Epoch 10/100
4/4 [==============================] - 1s 305ms/step - loss: 0.7379 - accuracy:
0.7054 - val_loss: 1.0243 - val_accuracy: 0.4643
Epoch 11/100
4/4 [==============================] - 1s 226ms/step - loss: 0.6797 - accuracy:
0.7232 - val_loss: 1.0277 - val_accuracy: 0.4643
Epoch 12/100
4/4 [==============================] - 1s 172ms/step - loss: 0.6185 - accuracy:
0.7500 - val_loss: 0.9942 - val_accuracy: 0.5357
Epoch 13/100
4/4 [==============================] - 1s 156ms/step - loss: 0.5432 - accuracy:
0.8125 - val_loss: 0.9996 - val_accuracy: 0.5714
Epoch 14/100
4/4 [==============================] - 1s 161ms/step - loss: 0.4781 - accuracy:
0.8036 - val_loss: 0.9632 - val_accuracy: 0.5357
```

```
Epoch 15/100
4/4 [==============================] - 1s 156ms/step - loss: 0.4188 - accuracy:
0.9018 - val_loss: 0.9356 - val_accuracy: 0.5357
Epoch 16/100
4/4 [==============================] - 1s 184ms/step - loss: 0.3680 - accuracy:
0.9107 - val_loss: 1.0088 - val_accuracy: 0.6071
Epoch 17/100
4/4 [==============================] - 1s 170ms/step - loss: 0.3175 - accuracy:
0.9643 - val_loss: 0.9370 - val_accuracy: 0.5714
Epoch 18/100
4/4 [==============================] - 1s 149ms/step - loss: 0.2824 - accuracy:
0.9464 - val_loss: 0.8856 - val_accuracy: 0.5357
Epoch 19/100
4/4 [==============================] - 1s 163ms/step - loss: 0.2428 - accuracy:
0.9821 - val_loss: 0.8610 - val_accuracy: 0.6071
Epoch 20/100
4/4 [==============================] - 1s 152ms/step - loss: 0.2015 - accuracy:
0.9821 - val_loss: 0.8803 - val_accuracy: 0.5000
Epoch 21/100
4/4 [==============================] - 1s 162ms/step - loss: 0.1655 - accuracy:
0.9821 - val_loss: 0.9934 - val_accuracy: 0.5357
Epoch 22/100
4/4 [==============================] - 1s 156ms/step - loss: 0.1250 - accuracy:
0.9732 - val_loss: 0.9503 - val_accuracy: 0.5357
Epoch 23/100
4/4 [==============================] - 1s 153ms/step - loss: 0.1038 - accuracy:
0.9911 - val_loss: 0.9699 - val_accuracy: 0.5000
Epoch 24/100
4/4 [==============================] - 1s 174ms/step - loss: 0.0918 - accuracy:
0.9821 - val_loss: 0.9855 - val_accuracy: 0.5000
Epoch 25/100
4/4 [==============================] - 1s 175ms/step - loss: 0.0769 - accuracy:
1.0000 - val_loss: 0.9654 - val_accuracy: 0.5357
Epoch 26/100
4/4 [==============================] - 1s 169ms/step - loss: 0.0599 - accuracy:
0.9911 - val_loss: 0.9979 - val_accuracy: 0.5000
Epoch 27/100
4/4 [==============================] - 1s 273ms/step - loss: 0.0548 - accuracy:
0.9821 - val_loss: 0.9983 - val_accuracy: 0.5714
Epoch 28/100
4/4 [==============================] - 1s 275ms/step - loss: 0.0589 - accuracy:
0.9821 - val_loss: 0.9816 - val_accuracy: 0.5000
Epoch 29/100
4/4 [==============================] - 1s 317ms/step - loss: 0.0367 - accuracy:
1.0000 - val_loss: 1.0032 - val_accuracy: 0.5000
Epoch 30/100
4/4 [==============================] - 1s 185ms/step - loss: 0.0335 - accuracy:
0.9911 - val_loss: 1.0643 - val_accuracy: 0.5714
```

```
Epoch 31/100
4/4 [==============================] - 1s 185ms/step - loss: 0.0289 - accuracy:
1.0000 - val_loss: 1.1793 - val_accuracy: 0.5000
Epoch 32/100
4/4 [==============================] - 1s 174ms/step - loss: 0.0237 - accuracy:
1.0000 - val_loss: 1.1738 - val_accuracy: 0.5357
Epoch 33/100
4/4 [==============================] - 1s 169ms/step - loss: 0.0221 - accuracy:
1.0000 - val_loss: 1.1833 - val_accuracy: 0.4643
Epoch 34/100
4/4 [==============================] - 1s 170ms/step - loss: 0.0260 - accuracy:
0.9911 - val_loss: 1.2237 - val_accuracy: 0.4643
Epoch 35/100
4/4 [==============================] - 1s 175ms/step - loss: 0.0220 - accuracy:
1.0000 - val_loss: 1.2735 - val_accuracy: 0.5000
Epoch 36/100
4/4 [==============================] - 1s 168ms/step - loss: 0.0191 - accuracy:
1.0000 - val_loss: 1.2914 - val_accuracy: 0.5714
Epoch 37/100
4/4 [==============================] - 1s 183ms/step - loss: 0.0188 - accuracy:
1.0000 - val_loss: 1.3161 - val_accuracy: 0.5000
Epoch 38/100
4/4 [==============================] - 1s 166ms/step - loss: 0.0125 - accuracy:
1.0000 - val_loss: 1.2225 - val_accuracy: 0.6071
Epoch 39/100
4/4 [==============================] - 1s 154ms/step - loss: 0.0158 - accuracy:
1.0000 - val_loss: 1.1759 - val_accuracy: 0.5714
Epoch 40/100
4/4 [==============================] - 1s 158ms/step - loss: 0.0128 - accuracy:
1.0000 - val_loss: 1.1544 - val_accuracy: 0.5714
Epoch 41/100
4/4 [==============================] - 1s 167ms/step - loss: 0.0109 - accuracy:
1.0000 - val_loss: 1.1658 - val_accuracy: 0.5714
Epoch 42/100
4/4 [==============================] - 1s 155ms/step - loss: 0.0118 - accuracy:
1.0000 - val_loss: 1.2089 - val_accuracy: 0.5357
Epoch 43/100
4/4 [==============================] - 1s 159ms/step - loss: 0.0144 - accuracy:
1.0000 - val_loss: 1.2621 - val_accuracy: 0.5357
Epoch 44/100
4/4 [==============================] - 1s 160ms/step - loss: 0.0104 - accuracy:
1.0000 - val_loss: 1.3007 - val_accuracy: 0.5357
Epoch 45/100
4/4 [==============================] - 1s 272ms/step - loss: 0.0080 - accuracy:
1.0000 - val_loss: 1.3072 - val_accuracy: 0.5714
Epoch 46/100
4/4 [==============================] - 1s 274ms/step - loss: 0.0080 - accuracy:
1.0000 - val_loss: 1.3391 - val_accuracy: 0.5714
```

```
Epoch 47/100
4/4 [==============================] - 1s 311ms/step - loss: 0.0073 - accuracy:
1.0000 - val_loss: 1.3703 - val_accuracy: 0.5357
Epoch 48/100
4/4 [==============================] - 1s 214ms/step - loss: 0.0133 - accuracy:
0.9911 - val_loss: 1.4552 - val_accuracy: 0.5000
Epoch 49/100
4/4 [==============================] - 1s 197ms/step - loss: 0.0077 - accuracy:
1.0000 - val_loss: 1.3723 - val_accuracy: 0.5000
Epoch 50/100
4/4 [==============================] - 1s 169ms/step - loss: 0.0049 - accuracy:
1.0000 - val_loss: 1.3362 - val_accuracy: 0.5714
Epoch 51/100
4/4 [==============================] - 1s 151ms/step - loss: 0.0076 - accuracy:
1.0000 - val_loss: 1.3355 - val_accuracy: 0.5714
Epoch 52/100
4/4 [==============================] - 1s 165ms/step - loss: 0.0078 - accuracy:
1.0000 - val_loss: 1.3394 - val_accuracy: 0.5714
Epoch 53/100
4/4 [==============================] - 1s 158ms/step - loss: 0.0086 - accuracy:
1.0000 - val_loss: 1.3523 - val_accuracy: 0.5714
Epoch 54/100
4/4 [==============================] - 1s 167ms/step - loss: 0.0066 - accuracy:
1.0000 - val_loss: 1.4079 - val_accuracy: 0.5000
Epoch 55/100
4/4 [==============================] - 1s 183ms/step - loss: 0.0047 - accuracy:
1.0000 - val_loss: 1.4508 - val_accuracy: 0.5000
Epoch 56/100
4/4 [==============================] - 1s 179ms/step - loss: 0.0055 - accuracy:
1.0000 - val_loss: 1.3680 - val_accuracy: 0.5357
Epoch 57/100
4/4 [==============================] - 1s 164ms/step - loss: 0.0043 - accuracy:
1.0000 - val_loss: 1.3528 - val_accuracy: 0.6071
Epoch 58/100
4/4 [==============================] - 1s 159ms/step - loss: 0.0050 - accuracy:
1.0000 - val_loss: 1.3653 - val_accuracy: 0.6071
Epoch 59/100
4/4 [==============================] - 1s 168ms/step - loss: 0.0050 - accuracy:
1.0000 - val_loss: 1.3783 - val_accuracy: 0.5714
Epoch 60/100
4/4 [==============================] - 1s 166ms/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 1.4113 - val_accuracy: 0.5357
Epoch 61/100
4/4 [==============================] - 1s 157ms/step - loss: 0.0035 - accuracy:
1.0000 - val_loss: 1.4335 - val_accuracy: 0.5357
Epoch 62/100
4/4 [==============================] - 1s 150ms/step - loss: 0.0047 - accuracy:
1.0000 - val_loss: 1.4600 - val_accuracy: 0.5000
```

```
Epoch 63/100
4/4 [==============================] - 1s 239ms/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 1.4728 - val_accuracy: 0.5000
Epoch 64/100
4/4 [==============================] - 1s 308ms/step - loss: 0.0031 - accuracy:
1.0000 - val_loss: 1.4674 - val_accuracy: 0.5000
Epoch 65/100
4/4 [==============================] - 1s 317ms/step - loss: 0.0027 - accuracy:
1.0000 - val_loss: 1.4697 - val_accuracy: 0.5357
Epoch 66/100
4/4 [==============================] - 1s 250ms/step - loss: 0.0029 - accuracy:
1.0000 - val_loss: 1.4876 - val_accuracy: 0.5000
Epoch 67/100
4/4 [==============================] - 1s 170ms/step - loss: 0.0032 - accuracy:
1.0000 - val_loss: 1.5110 - val_accuracy: 0.5000
Epoch 68/100
4/4 [==============================] - 1s 196ms/step - loss: 0.0022 - accuracy:
1.0000 - val_loss: 1.5361 - val_accuracy: 0.5714
Epoch 69/100
4/4 [==============================] - 1s 160ms/step - loss: 0.0041 - accuracy:
1.0000 - val_loss: 1.5497 - val_accuracy: 0.5714
Epoch 70/100
4/4 [==============================] - 1s 154ms/step - loss: 0.0025 - accuracy:
1.0000 - val_loss: 1.5547 - val_accuracy: 0.5000
Epoch 71/100
4/4 [==============================] - 1s 162ms/step - loss: 0.0033 - accuracy:
1.0000 - val_loss: 1.5571 - val_accuracy: 0.5000
Epoch 72/100
4/4 [==============================] - 1s 175ms/step - loss: 0.0028 - accuracy:
1.0000 - val_loss: 1.5051 - val_accuracy: 0.5357
Epoch 73/100
4/4 [==============================] - 1s 172ms/step - loss: 0.0022 - accuracy:
1.0000 - val_loss: 1.4745 - val_accuracy: 0.5357
Epoch 74/100
4/4 [==============================] - 1s 154ms/step - loss: 0.0025 - accuracy:
1.0000 - val_loss: 1.4604 - val_accuracy: 0.5714
Epoch 75/100
4/4 [==============================] - 1s 170ms/step - loss: 0.0028 - accuracy:
1.0000 - val_loss: 1.4631 - val_accuracy: 0.5357
Epoch 76/100
4/4 [==============================] - 1s 177ms/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 1.4873 - val_accuracy: 0.5000
Epoch 77/100
4/4 [==============================] - 1s 168ms/step - loss: 0.0024 - accuracy:
1.0000 - val_loss: 1.5322 - val_accuracy: 0.5000
Epoch 78/100
4/4 [==============================] - 1s 172ms/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 1.5586 - val_accuracy: 0.5000
```

```
Epoch 79/100
4/4 [==============================] - 1s 172ms/step - loss: 0.0018 - accuracy:
1.0000 - val_loss: 1.5860 - val_accuracy: 0.5000
Epoch 80/100
4/4 [==============================] - 1s 155ms/step - loss: 0.0020 - accuracy:
1.0000 - val_loss: 1.5812 - val_accuracy: 0.5000
Epoch 81/100
4/4 [==============================] - 1s 253ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 1.5674 - val_accuracy: 0.5000
Epoch 82/100
4/4 [==============================] - 1s 281ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 1.5632 - val_accuracy: 0.5000
Epoch 83/100
4/4 [==============================] - 1s 314ms/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 1.5663 - val_accuracy: 0.5000
Epoch 84/100
4/4 [==============================] - 1s 246ms/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 1.5860 - val_accuracy: 0.4643
Epoch 85/100
4/4 [==============================] - 1s 152ms/step - loss: 0.0015 - accuracy:
1.0000 - val_loss: 1.6022 - val_accuracy: 0.4643
Epoch 86/100
4/4 [==============================] - 1s 185ms/step - loss: 0.0018 - accuracy:
1.0000 - val_loss: 1.6057 - val_accuracy: 0.5000
Epoch 87/100
4/4 [==============================] - 1s 171ms/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 1.6125 - val_accuracy: 0.5357
Epoch 88/100
4/4 [==============================] - 1s 173ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 1.6239 - val_accuracy: 0.5357
Epoch 89/100
4/4 [==============================] - 1s 195ms/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 1.6364 - val_accuracy: 0.5357
Epoch 90/100
4/4 [==============================] - 1s 170ms/step - loss: 0.0019 - accuracy:
1.0000 - val_loss: 1.6496 - val_accuracy: 0.5000
Epoch 91/100
4/4 [==============================] - 1s 153ms/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 1.6585 - val_accuracy: 0.5000
Epoch 92/100
4/4 [==============================] - 1s 173ms/step - loss: 0.0010 - accuracy:
1.0000 - val_loss: 1.6702 - val_accuracy: 0.5000
Epoch 93/100
4/4 [==============================] - 1s 188ms/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 1.6866 - val_accuracy: 0.5000
Epoch 94/100
4/4 [==============================] - 1s 164ms/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 1.6911 - val_accuracy: 0.5000
```

```
Epoch 95/100
4/4 [==============================] - 1s 171ms/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 1.6856 - val_accuracy: 0.5000
Epoch 96/100
4/4 [==============================] - 1s 175ms/step - loss: 0.0015 - accuracy:
1.0000 - val_loss: 1.6921 - val_accuracy: 0.5357
Epoch 97/100
4/4 [==============================] - 1s 151ms/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 1.7091 - val_accuracy: 0.5357
Epoch 98/100
4/4 [==============================] - 1s 166ms/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 1.7161 - val_accuracy: 0.5357
Epoch 99/100
4/4 [==============================] - 1s 295ms/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 1.7160 - val_accuracy: 0.5000
Epoch 100/100
4/4 [==============================] - 1s 303ms/step - loss: 0.0010 - accuracy:
1.0000 - val_loss: 1.7125 - val_accuracy: 0.5000
1/1 [==============================] - 0s 43ms/step - loss: 1.7125 - accuracy:
0.5000
Test Loss: 1.7125, Test Accuracy: 0.5000
```

Model ini memilik akurasi 0.5, bisa dikatakan model masih kurang bagus dalam melakukan prediksi. Hal ini bisa jadi disebabkan oleh data yang digunakan masih terlalu sedikit, dalam processing ini data yang digunakan adalahh 140 data. Keterbatasan data dapat menyebabkan model tidak dapat memprediksi dengan presisi

[31]:
```python
# Save the model to a file
model.save('lstm_sentiment_model.h5')

# Optionally, save the tokenizer as well for later use during inference
import pickle

with open('tokenizer.pkl', 'wb') as tokenizer_file:
    pickle.dump(tokenizer, tokenizer_file)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

[37]:
```python
from keras.models import load_model
import pickle

# Load the model
loaded_model = load_model('lstm_sentiment_model.h5')
```

```
# Load the tokenizer
with open('tokenizer.pkl', 'rb') as tokenizer_file:
    loaded_tokenizer = pickle.load(tokenizer_file)

# New sentence for testing
test_sentence = 'nangis banget nontonnya'
new_sentence = preprocess(test_sentence)
# Tokenize and pad the new sentence
new_sequence = loaded_tokenizer.texts_to_sequences([new_sentence])
new_data = pad_sequences(new_sequence, maxlen=max_sequence_length)

# Predict sentiment for the new sentence
predictions = loaded_model.predict(new_data)

# Get the predicted sentiment label
predicted_sentiment_label = labels.columns[predictions.argmax(axis=1)[0]]
print(f"preprocessing: {new_sentence}")
print(f"new_sequence: {new_sequence}")
print(f"new_data: {new_data}")
print(f"Predicted Sentiment: {predicted_sentiment_label}")
```

```
1/1 [==============================] - 0s 253ms/step
preprocessing: nangis nontonnya
new_sequence: [[5]]
new_data: [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5]]
Predicted Sentiment: Positif
```

[38]:
```
# New sentence for testing
test_sentence = 'cringe banget filmnya'
new_sentence = preprocess(test_sentence)
# Tokenize and pad the new sentence
new_sequence = loaded_tokenizer.texts_to_sequences([new_sentence])
new_data = pad_sequences(new_sequence, maxlen=max_sequence_length)

# Predict sentiment for the new sentence
predictions = loaded_model.predict(new_data)

# Get the predicted sentiment label
predicted_sentiment_label = labels.columns[predictions.argmax(axis=1)[0]]
print(f"preprocessing: {new_sentence}")
print(f"new_sequence: {new_sequence}")
print(f"new_data: {new_data}")
print(f"Predicted Sentiment: {predicted_sentiment_label}")
```

```
1/1 [==============================] - 0s 30ms/step
```

```
preprocessing: cringe filmnya
new_sequence: [[17, 352]]
new_data: [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  17 352]]
Predicted Sentiment: Negatif
```

dilihat dari test tes di atas, model masih mampu memprediksi dengan benar

1.3 Link github

Find code and data on github

https://github.com/wellyokt/ML2_DAY24.git