BINAR DSC#25 | Platinum Challenge

Building a Sentiment Analysis Engine & API

By:
Isumi Karina, Siti Aisyah, Nur’Adilah, Devan AA

Table of contents

01 Introduction

Background & Supporting Data

02 Methodology

Cleansing Process, Statistical & EDA Methods

03 Visualization

Visualization Process

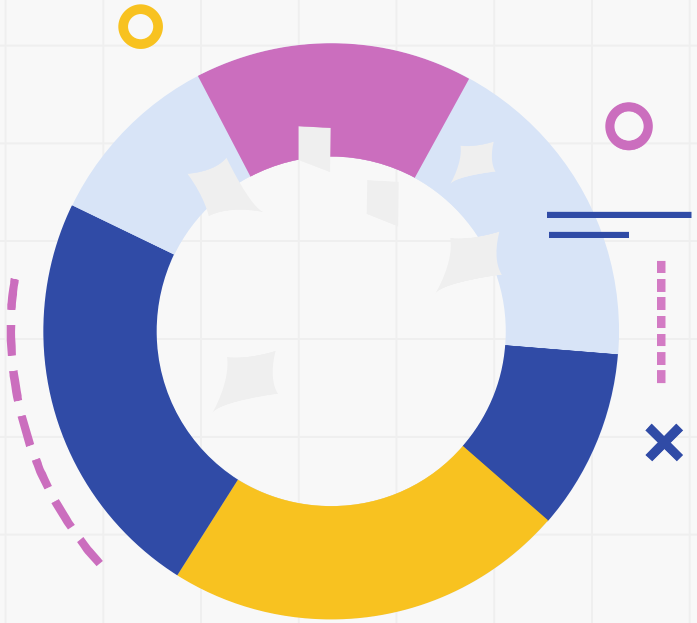
04 Results & Conclusion

Results, Conclusion, and Recommendation

01

Introduction

Background & Supporting Data



Background

Why Sentiment Analysis?

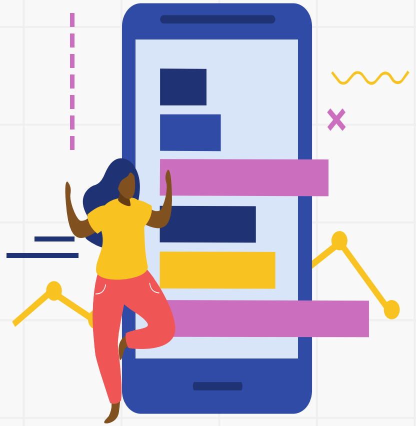
- In the digital age, millions of user-generated texts (e.g., comments, reviews) are shared daily.
- Understanding the sentiment behind these texts is crucial for businesses, content moderation, and user engagement strategies.

Problem Statement:

- Informal language, slang, and abusive words in user-generated content pose a challenge for traditional sentiment analysis models.
- There is a need for an engine/API capable of classifying text into positive, neutral, and negative sentiments effectively.

Project Purpose
To develop a robust API that:

1. Processes informal Indonesian text data, including slang and abusive words.
2. Classifies user inputs into **positive**, **neutral**, and **negative** sentiments.
3. Provides APIs for developers to integrate sentiment analysis into their applications.



Supporting Data

In 2024, approximately 5.44 billion individuals worldwide were internet users, accounting for about two-thirds of the global population (Statista). This vast online presence generates an immense volume of user-generated content daily, including comments, reviews, and social media posts.Understanding the sentiment within this content is crucial for businesses and platforms. Notably, 95% of consumers read online reviews before making a purchase, highlighting the significant impact of sentiment on consumer behavior (WiserNotify).

However, analyzing sentiment in informal text presents challenges due to the use of slang, abbreviations, and colloquial expressions. For instance, in Indonesian online communication, terms like "gue" (I), "lo" (you), and "alay" (over the top) are prevalent. Additionally, the presence of abusive language complicates sentiment analysis, necessitating robust preprocessing techniques. Addressing these challenges is essential for accurate sentiment analysis, enabling businesses to gain valuable insights from user-generated content and make informed decisions.

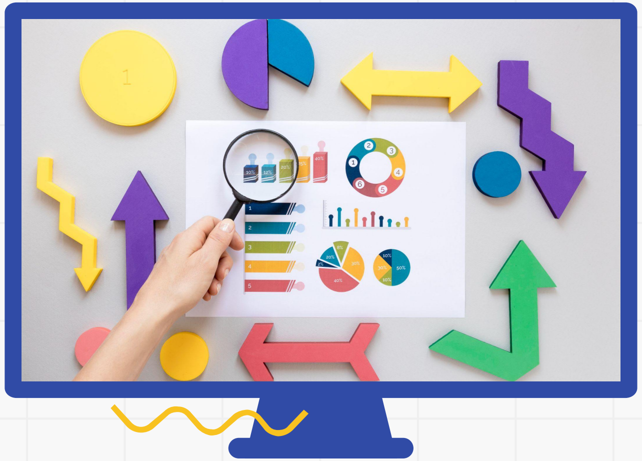
For this analysis, we used three specific files from the challenge dataset:

1. **dataset.csv** (here)
2. **abusive.csv**: A collection of abusive words used in Indonesian tweets, stored in a single column.
3. **new_kamusalay.csv**: A two-column file containing slang or "alay" words and their normalized equivalents, aiding in the data cleansing process to better understand the context of the speech.

02

Methodology

Sentiment Analysis Workflow, and Statistical & EDA Methods



Sentiment Analysis Workflow

Data Preparation:

- Cleaning data using `new_kamusalay.csv` and `abusive.csv`.
- Storing cleaned data in clean.csv

Feature Extraction:

- Converting text into numeric features suitable for machine learning.

Model Training:

- Using Neural Network and LSTM models.

Evaluation:

- Assessing model performance.

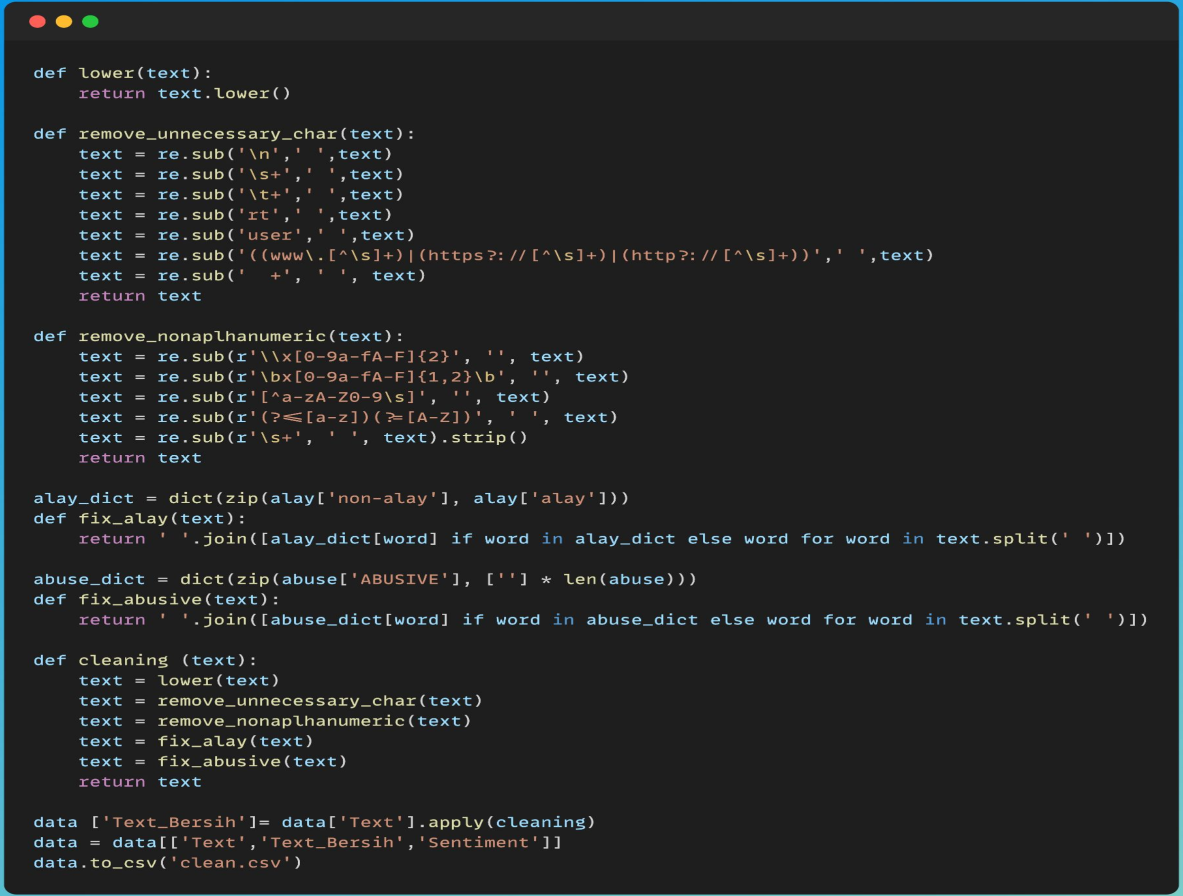
Prediction:

- Predicting sentiment labels (positive, neutral, negative) for new inputs.

Cleansing Process

Given the need to analyze the content, **data cleansing** plays a vital role in ensuring accurate analysis. The cleansing process also takes advantage of the use of **Regex** and other functions, including:

- **Lowercasing**: Converting the entire tweet text to lowercase ensures that all text is treated uniformly, which simplifies the cleansing process.
- **Removing Escape Characters and Unnecessary Whitespaces**: This step strips the text of any unwanted escape characters such as `\n`, `\t`, or hexadecimal sequences like `\xF0`. This ensures that the text is more readable and analyzable.
- **Removing Numeric Sequences**: Irrelevant numeric sequences are often present in tweets, especially in noisy data like social media text. This regex removes them while preserving meaningful text.
- **Replacing Slang Words (Alay)**: Using the `new_kamusalay.csv` dataset, slang words are normalized to standard Indonesian. This makes the text easier to interpret during analysis.
- **Removing Abusive Words**: The `abusive.csv` file contains a list of abusive words, and this step removes any such words found in the tweet.
- **Synonym replacement for augmenting training data**.



```python
def lower(text):
    return text.lower()

def remove_unnecessary_char(text):
    text = re.sub('\n',' ',text)
    text = re.sub('\s+',' ',text)
    text = re.sub('\t+',' ',text)
    text = re.sub('rt',' ',text)
    text = re.sub('user',' ',text)
    text = re.sub('((www\.[^\s]+)|(https?://[^\s]+)|(http?://[^\s]+))',' ',text)
    text = re.sub('  +', ' ', text)
    return text

def remove_nonaplhanumeric(text):
    text = re.sub(r'\\x[0-9a-fA-F]{2}', '', text)
    text = re.sub(r'\bx[0-9a-fA-F]{1,2}\b', '', text)
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    text = re.sub(r'(?<=[a-z])(?=[A-Z])', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

alay_dict = dict(zip(alay['non-alay'], alay['alay']))
def fix_alay(text):
    return ' '.join([alay_dict[word] if word in alay_dict else word for word in text.split(' ')])

abuse_dict = dict(zip(abuse['ABUSIVE'], [''] * len(abuse)))
def fix_abusive(text):
    return ' '.join([abuse_dict[word] if word in abuse_dict else word for word in text.split(' ')])

def cleaning (text):
    text = lower(text)
    text = remove_unnecessary_char(text)
    text = remove_nonaplhanumeric(text)
    text = fix_alay(text)
    text = fix_abusive(text)
    return text

data ['Text_Bersih']= data['Text'].apply(cleaning)
data = data[['Text','Text_Bersih','Sentiment']]
data.to_csv('clean.csv')
```

Feature Extraction Process

Why Feature Extraction?

- **Purpose**: Transform raw text data into numerical representations for machine learning models.
- **Importance**: Enables models to understand text features and relationships effectively.
- **Approach**: Utilize tokenization, vectorization, and statistical techniques to represent text data.

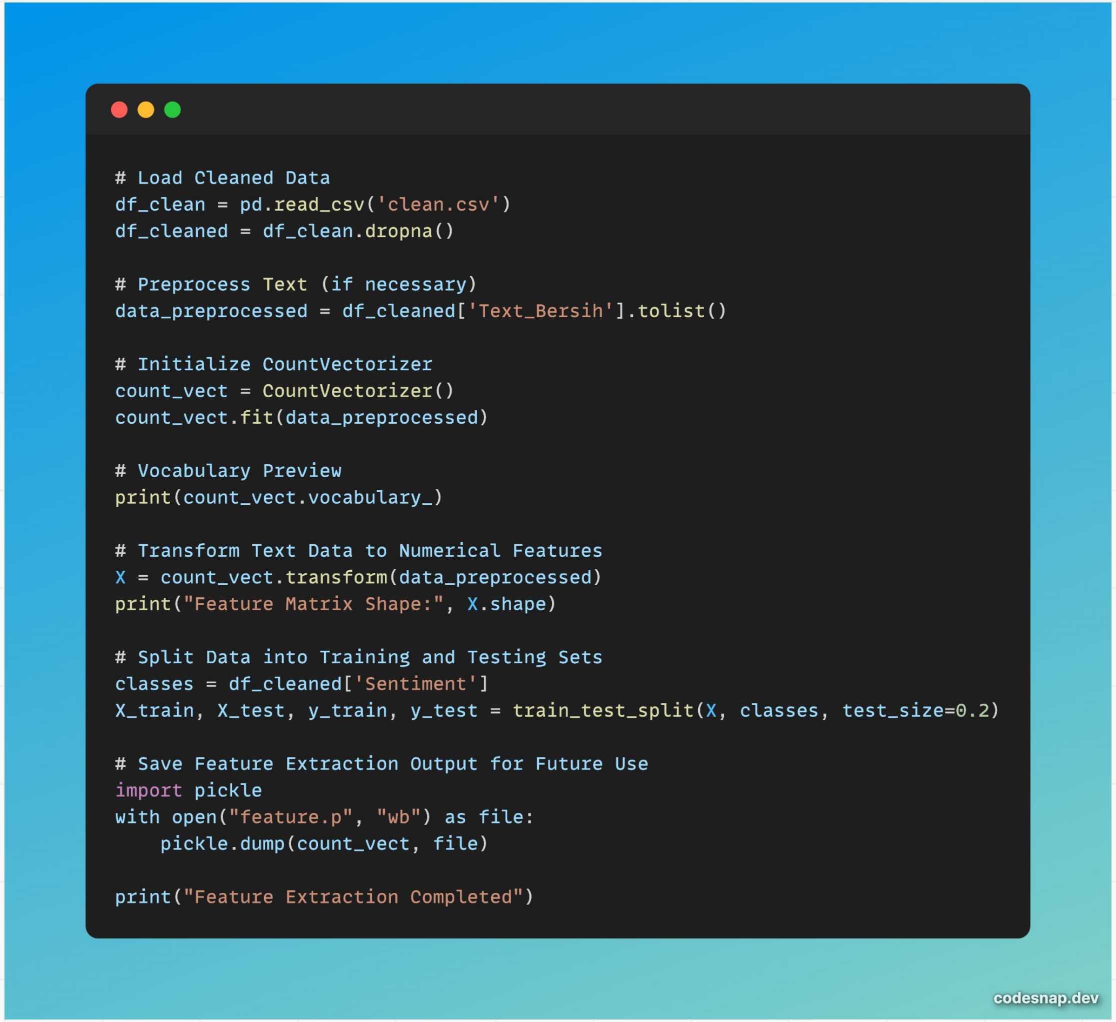
Steps in Feature Extraction

1. **Dataset Preparation**
   - Load and clean the dataset (`clean.csv`) to remove null values and unnecessary rows.
   - Example: `df_cleaned = df_clean.dropna()` ensures no missing data.
2. **Text Tokenization**
   - Split text into sentences and words for detailed analysis.
   - Tools: `nltk.sent_tokenize` and `nltk.word_tokenize` for splitting sentences and words.
3. **Text Normalization**
   - Remove punctuation, numbers, and special characters.
   - Lowercase all text for consistency.
   - Stopword removal to eliminate common but non-informative words.

Feature Extraction Process

4. **Vectorization**
   - Convert text data into numerical form using `CountVectorizer`
   - Vocabulary creation and term frequency calculation for each unique word..
5. **Splitting the Dataset**

   - Divide the dataset into training and testing sets using `train_test_split`.
   - Ensure the model is trained on 80% data and tested on 20% unseen data.
6. **Exporting Data**
   - Save features using `pickle`.
   - Export train and test datasets as `.csv` files.



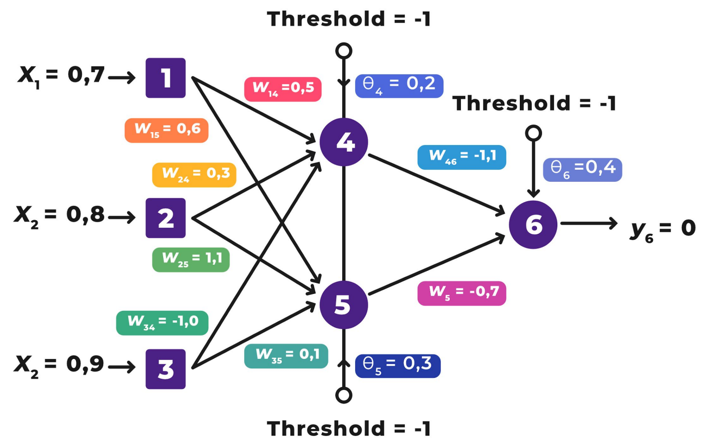
```python
# Load Cleaned Data
df_clean = pd.read_csv('clean.csv')
df_cleaned = df_clean.dropna()

# Preprocess Text (if necessary)
data_preprocessed = df_cleaned['Text_Bersih'].tolist()

# Initialize CountVectorizer
count_vect = CountVectorizer()
count_vect.fit(data_preprocessed)

# Vocabulary Preview
print(count_vect.vocabulary_)

# Transform Text Data to Numerical Features
X = count_vect.transform(data_preprocessed)
print("Feature Matrix Shape:", X.shape)

# Split Data into Training and Testing Sets
classes = df_cleaned['Sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, classes, test_size=0.2)

# Save Feature Extraction Output for Future Use
import pickle
with open("feature.p", "wb") as file:
    pickle.dump(count_vect, file)

print("Feature Extraction Completed")
```

codesnap.dev

Manual Sentiment Calculation

Neural network training involves systematically adjusting parameters to minimize error and improve accuracy. The process consists of:

1. **Initialization**: Define inputs(**x**), learning rate (**α**), weights (**W**), biases (**θ**), and target outputs.
2. **Forward Pass**: Compute neuron outputs (**y4,y5,y6**) using weighted sums, biases, and the sigmoid activation function. Calculated error to measure performance.
3. **Backward Pass**: Calculated gradient for the output neuron (**δ6**) and the hidden neurons (**δ4,δ5**). Weight and bias corrections are determined using these gradients and the learning rate.
4. **Parameter Updates**: Weights and biases are updated using the corrections to refine the network's performance.
5. **Results**: The updated parameters and calculated values are summarized, showcasing how the network learns iteratively.



The full document (here), The document provided offers a detailed step-by-step guide to manually training a neural network.

Training Neural Network Model

Why Use Neural Network RNN for Sentiment Analysis in This Project?

Sequential Nature of Text Data

- Artinya urutan kata sangat mempengaruhi makna dan sentimen. RNN dapat memproses urutan kata dan mempertahankan konteks yang penting untuk analisis sentimen.

Contextual Sentiment Analysis

- **RNN** sangat baik dalam menangkap makna kata yang berubah tergantung pada konteks sekitarnya, seperti dalam kalimat "not bad" yang berarti positif meskipun ada kata "bad".

Handling Long-Term Dependencies with LSTM

- **LSTM** mengatasi masalah **vanishing gradient** yang terjadi pada RNN tradisional, memungkinkan mereka mengingat informasi penting dalam urutan yang panjang, yang diperlukan untuk menangkap sentimen yang tersebar di seluruh kalimat.

Training Neural Network Model

Implementation Steps:

1. Data Loading and Preprocessing

- **Kode dimulai dengan memuat kumpulan data (mungkin ke dalam Pandas DataFrame yang disebut 'df_cleaned').**
- **#Kemudian kode mendefinisikan fungsi praproses untuk membersihkan dan menyiapkan data teks. Ini melibatkan penghapusan tanda baca, mengubah teks menjadi huruf kecil, membuat token teks menjadi kata-kata, dan menyaring kata-kata berhenti. Data teks yang dibersihkan disimpan dalam kolom baru 'Clean_Text' di DataFrame.**

2. Feature Extraction

- **Tujuan: Mengubah teks menjadi representasi numerik yang bisa dipahami oleh model machine learning.**

3. Data Splitting

Tujuan: Memisahkan data menjadi dua set: training dan testing.

1. **train_test_split digunakan untuk membagi dataset menjadi data pelatihan (80%) dan data pengujian (20%) secara acak.**
2. **Proses ini penting untuk memastikan bahwa model tidak hanya menghafal data pelatihan (overfitting), tetapi dapat menggeneralisasi pada data baru.**

Training Neural Network Model

4. Model Building (RNN)

- **Tujuan:** Membangun arsitektur model untuk analisis sentimen.
- Model menggunakan **Sequential** dari Keras. Arsitekturnya terdiri dari:
    - **Embedding layer**: Mengubah input teks menjadi vektor berdimensi tetap.
    - **SimpleRNN layer**: Memproses urutan kata dalam teks untuk menangkap ketergantungan temporal antar kata.
    - **Dense output layer**: Lapisan akhir dengan fungsi aktivasi **softmax** untuk klasifikasi sentimen ke dalam tiga kategori (positif, netral, negatif).

5. Model Training

- **Tujuan: Melatih model pada data pelatihan dan memantau kinerja pada data validasi.**
- **Langkah-langkah:**

1. **Model dilatih menggunakan data training set dan diuji dengan validation set untuk mengontrol kinerja selama pelatihan.**
2. **Proses pelatihan dilakukan dengan 10 epoch dan ukuran batch 10. Setiap epoch model diperbarui untuk meminimalkan fungsi loss dan meningkatkan akurasi.**

Training Neural Network Model

6. Model Evaluation

- Tujuan: Menilai kinerja model setelah pelatihan.
- Langkah-langkah:
    - Prediksi dan evaluasi: Model melakukan prediksi pada testing set dan hasil prediksi dibandingkan dengan label aktual.
    - Classification report digunakan untuk menghitung berbagai metrik evaluasi seperti precision, recall, F1-score, dan accuracy untuk setiap kelas sentimen.

7. Prediction

- Tujuan: Membuat prediksi sentimen untuk teks baru.
- Model digunakan untuk memprediksi sentimen berdasarkan teks input dan hasil prediksi kemudian diklasifikasikan ke dalam kategori sentimen yang sesuai.

Training Neural Network Model

8. **Model Saving and Loading**

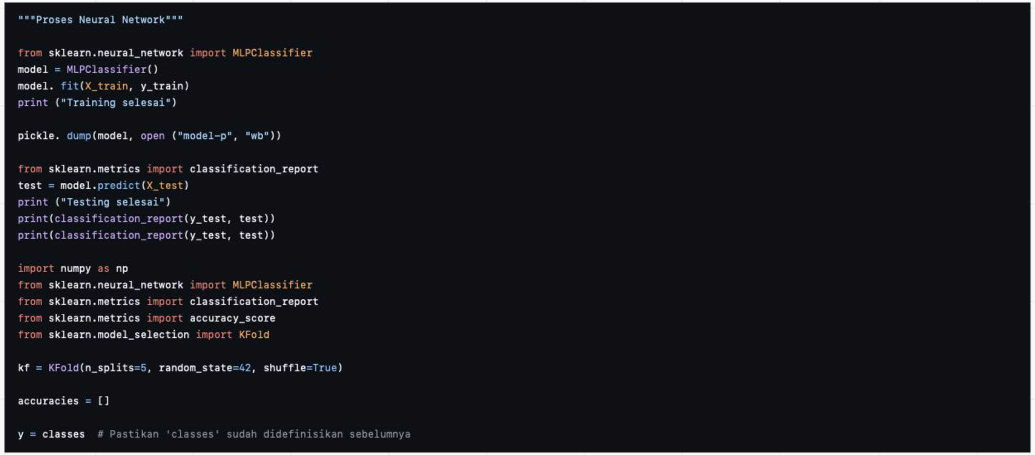
- **Menyimpan dan memuat model yang telah dilatih untuk digunakan di masa depan.**
- **Model, tokenizer, dan data (X dan Y) disimpan menggunakan pickle untuk memudahkan penggunaan kembali tanpa perlu melatih ulang.**
- **Model disimpan dalam format `.h5` dan dapat dimuat kembali dengan menggunakan `load_model` untuk digunakan pada prediksi baru.**

9. Confusion Matrix

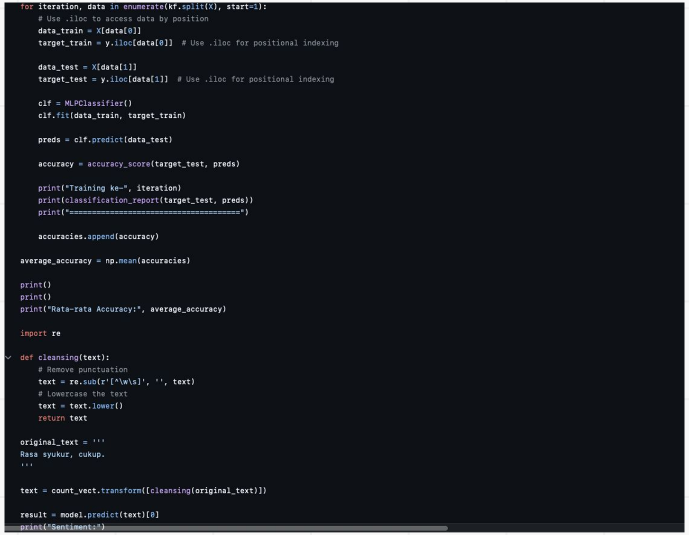
- **Tujuan: Mengevaluasi kinerja model dengan confusion matrix.**

10. Plotting

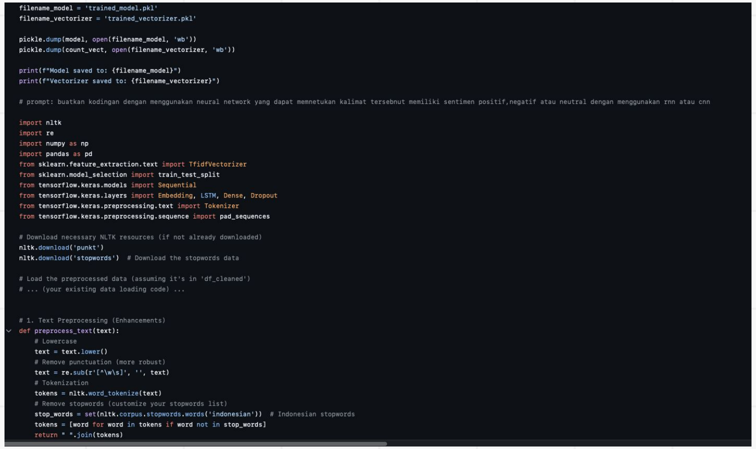
- **Tujuan: Memvisualisasikan proses pelatihan model.**



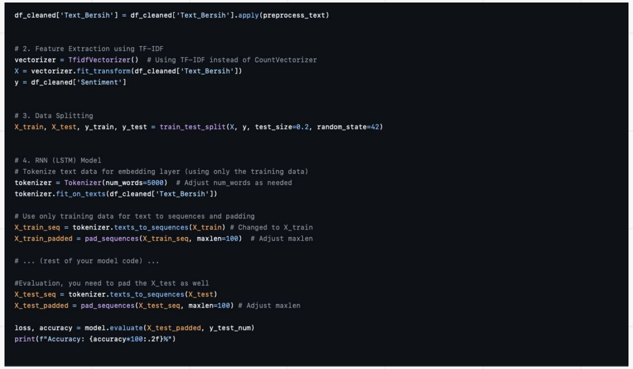
```python
"""Proses Neural Network"""

from sklearn.neural_network import MLPClassifier
model = MLPClassifier()
model. fit(X_train, y_train)
print ("Training selesai")

pickle. dump(model, open ("model-p", "wb"))

from sklearn.metrics import classification_report
test = model.predict(X_test)
print ("Testing selesai")
print(classification_report(y_test, test))
print(classification_report(y_test, test))

import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, random_state=42, shuffle=True)

accuracies = []

y = classes  # Pastikan 'classes' sudah didefinisikan sebelumnya
```



```python
for iteration, data in enumerate(kf.split(X), start=1):
    # Use .iloc to access data by position
    data_train = X[data[0]]
    target_train = y.iloc[data[0]]  # Use .iloc for positional indexing

    data_test = X[data[1]]
    target_test = y.iloc[data[1]]  # Use .iloc for positional indexing

    clf = MLPClassifier()
    clf.fit(data_train, target_train)

    preds = clf.predict(data_test)

    accuracy = accuracy_score(target_test, preds)

    print("Training ke-", iteration)
    print(classification_report(target_test, preds))
    print("======================================")

    accuracies.append(accuracy)

average_accuracy = np.mean(accuracies)

print()
print()
print("Rata-rata Accuracy:", average_accuracy)

import re

def cleansing(text):
    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # Lowercase the text
    text = text.lower()
    return text

original_text = '''
Rasa syukur, cukup.
'''

text = count_vect.transform([cleansing(original_text)])

result = model.predict(text)[0]
print("Sentiment:")
```

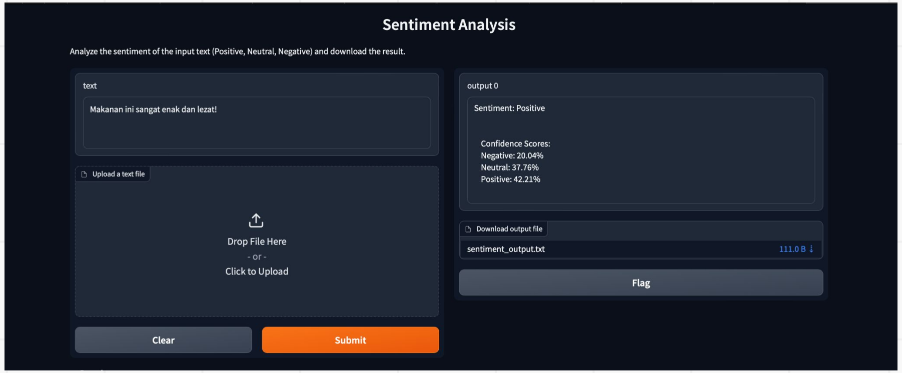


```python
filename_model = 'trained_model.pkl'
filename_vectorizer = 'trained_vectorizer.pkl'

pickle.dump(model, open(filename_model, 'wb'))
pickle.dump(count_vect, open(filename_vectorizer, 'wb'))

print(f"Model saved to: {filename_model}")
print(f"Vectorizer saved to: {filename_vectorizer}")

# prompt: buatkan kodingan dengan menggunakan neural network yang dapat memnetukan kalimat tersebnut memiliki sentimen positif,negatif atau neutral dengan menggunakan rnn atau cnn

import nltk
import re
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Download necessary NLTK resources (if not already downloaded)
nltk.download('punkt')
nltk.download('stopwords')  # Download the stopwords data

# Load the preprocessed data (assuming it's in 'df_cleaned')
# ... (your existing data loading code) ...


# 1. Text Preprocessing (Enhancements)
def preprocess_text(text):
    # Lowercase
    text = text.lower()
    # Remove punctuation (more robust)
    text = re.sub(r'[^\w\s]', '', text)
    # Tokenization
    tokens = nltk.word_tokenize(text)
    # Remove stopwords (customize your stopwords list)
    stop_words = set(nltk.corpus.stopwords.words('indonesian'))  # Indonesian stopwords
    tokens = [word for word in tokens if word not in stop_words]
    return " ".join(tokens)
```



```python
df_cleaned['Text_Bersih'] = df_cleaned['Text_Bersih'].apply(preprocess_text)


# 2. Feature Extraction using TF-IDF
vectorizer = TfidfVectorizer()  # Using TF-IDF instead of CountVectorizer
X = vectorizer.fit_transform(df_cleaned['Text_Bersih'])
y = df_cleaned['Sentiment']


# 3. Data Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# 4. RNN (LSTM) Model
# Tokenize text data for embedding layer (using only the training data)
tokenizer = Tokenizer(num_words=5000)  # Adjust num_words as needed
tokenizer.fit_on_texts(df_cleaned['Text_Bersih'])

# Use only training data for text to sequences and padding
X_train_seq = tokenizer.texts_to_sequences(X_train) # Changed to X_train
X_train_padded = pad_sequences(X_train_seq, maxlen=100)  # Adjust maxlen

# ... (rest of your model code) ...

#Evaluation, you need to pad the X_test as well
X_test_seq = tokenizer.texts_to_sequences(X_test)
X_test_padded = pad_sequences(X_test_seq, maxlen=100) # Adjust maxlen

loss, accuracy = model.evaluate(X_test_padded, y_test_num)
print(f"Accuracy: {accuracy*100:.2f}%")
```

Neural Network API Result


Sentiment Analysis

Analyze the sentiment of the input text (Positive, Neutral, Negative) and download the result.

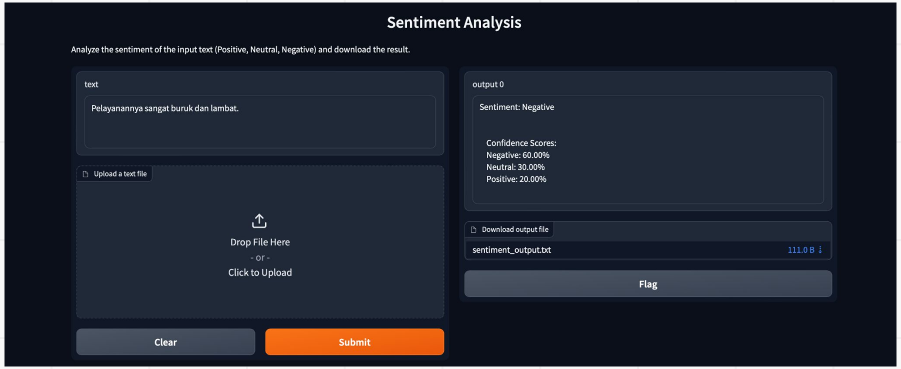
text

Makanan ini sangat enak dan lezat!

output 0

```
Sentiment: Positive

Confidence Scores:
Negative: 20.04%
Neutral: 37.76%
Positive: 42.21%
```

sentiment_output.txt 111.0 B


Sentiment Analysis

Analyze the sentiment of the input text (Positive, Neutral, Negative) and download the result.

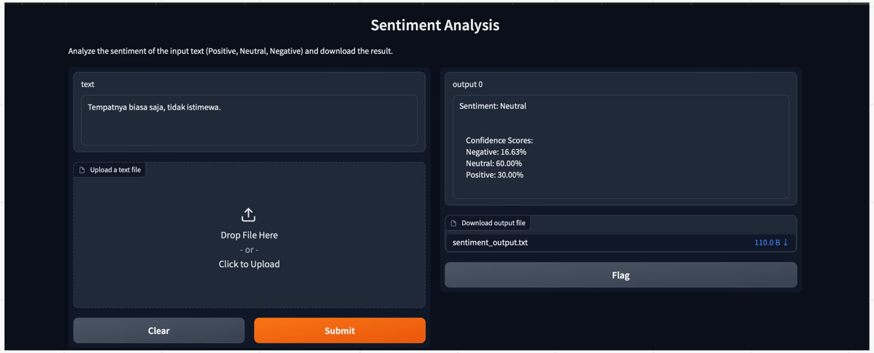
text

Pelayanannya sangat buruk dan lambat.

output 0

```
Sentiment: Negative

Confidence Scores:
Negative: 60.00%
Neutral: 30.00%
Positive: 20.00%
```

sentiment_output.txt 111.0 B


Sentiment Analysis

Analyze the sentiment of the input text (Positive, Neutral, Negative) and download the result.

text

Tempatnya biasa saja, tidak istimewa.

output 0

```
Sentiment: Neutral

Confidence Scores:
Negative: 16.63%
Neutral: 60.00%
Positive: 30.00%
```

sentiment_output.txt 110.0 B

Training LSTM Model

Why Use LSTM for Sentiment Analysis in This Project?

- **Handling Informal Text:** The challenge involves analyzing non-formal text, which often includes slang, abbreviations, and varied sentence structures. LSTM's ability to capture context over long sequences makes it well-suited for understanding such complexities.
- **Sequential Data Processing:** Sentiment in text is influenced by the order of words. LSTM networks are adept at processing sequences, allowing them to grasp the nuances of sentiment that depend on word order.
- **Mitigating Vanishing Gradient Problem:** LSTM ensure more stable and effective training for complex sequences.

Detailed Process

1. **Class Weight Computation**
   - Handled class imbalance using `compute_class_weight`.
   - Adjusted loss contribution for each sentiment class based on its frequency.

2. **FastText Embedding Matrix**
   - Pre-trained embeddings (`cc.id.300.vec`, dimension 300).
   - Loaded embeddings into a matrix matching the tokenized vocabulary.
   - Ensured that embeddings remain non-trainable.

Training LSTM Model

3. Model Configuration

- **Embedding Layer:** Maps input text to dense FastText vectors.
- **Bidirectional LSTM Layers:** Captures dependencies in both directions, with two layers (256 units each).
- **Dense Layer with Softmax:** Outputs probabilities for positive, neutral, or negative classes.

4. Training Settings

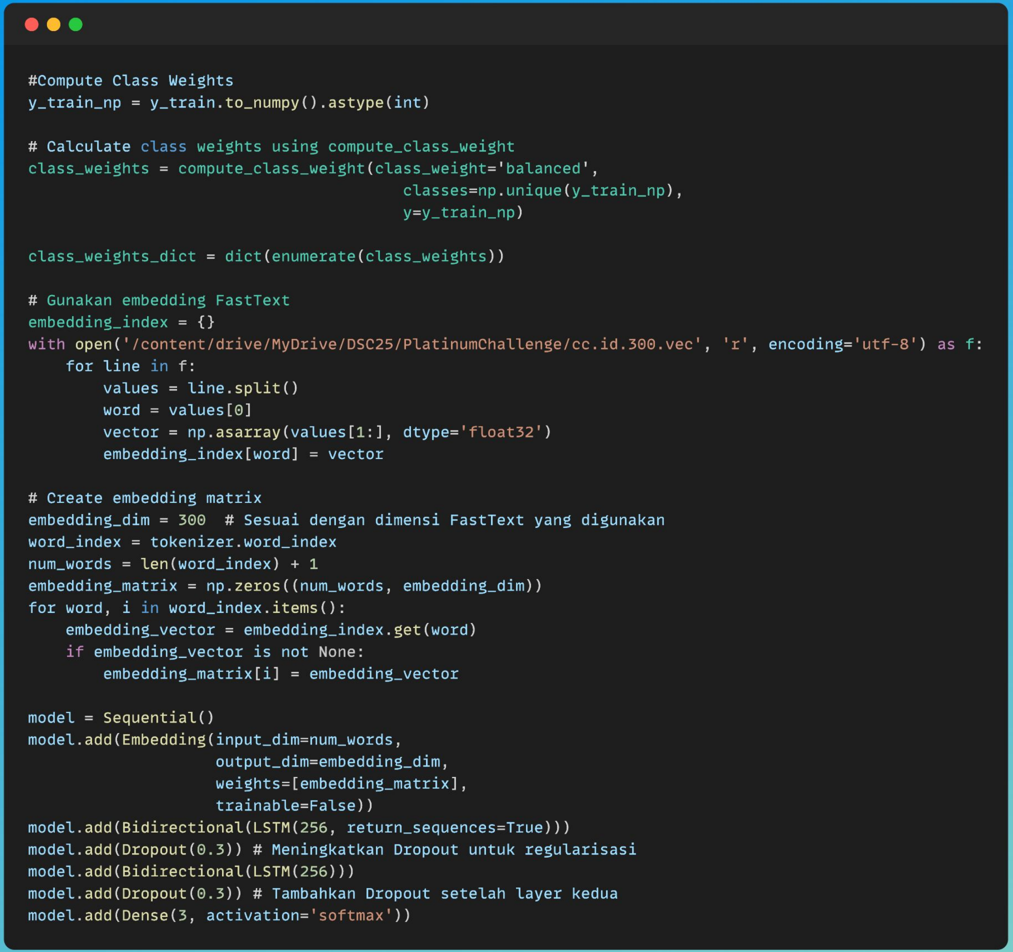
- **Loss Function:** Sparse categorical crossentropy for multi-class classification.
- **Optimizer:** Adam (learning rate: 0.001), offering adaptive learning.
- **Regularization:** Dropout (0.3) and L2 to combat overfitting.
- **Callbacks:** Early stopping (patience: 5) and learning rate reduction on plateau (factor: 0.2).

5. Epochs and Batch Size

- **Epochs:** Set to 50 for extensive learning, monitored by early stopping.
- **Batch Size:** Kept at 32 for balance between performance and memory usage.

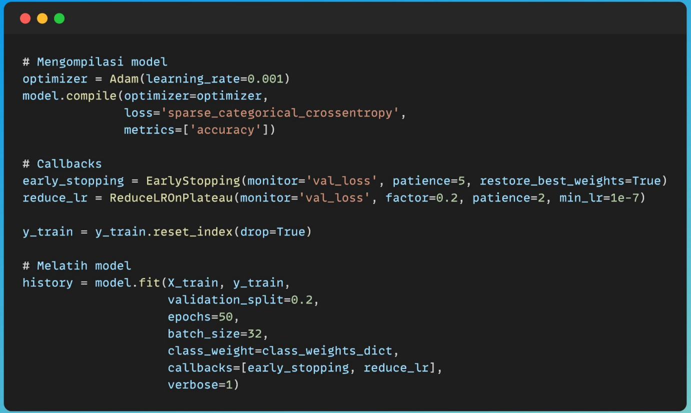
Key Benefits of This Approach

- Addresses data imbalance effectively.
- Uses pre-trained embeddings for richer semantic understanding.
- Incorporates advanced regularization techniques.
- Adjusts learning dynamically for efficient model convergence.



```python
#Compute Class Weights
y_train_np = y_train.to_numpy().astype(int)

# Calculate class weights using compute_class_weight
class_weights = compute_class_weight(class_weight='balanced',
                                     classes=np.unique(y_train_np),
                                     y=y_train_np)

class_weights_dict = dict(enumerate(class_weights))

# Gunakan embedding FastText
embedding_index = {}
with open('/content/drive/MyDrive/DSC25/PlatinumChallenge/cc.id.300.vec', 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = vector

# Create embedding matrix
embedding_dim = 300  # Sesuai dengan dimensi FastText yang digunakan
word_index = tokenizer.word_index
num_words = len(word_index) + 1
embedding_matrix = np.zeros((num_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

model = Sequential()
model.add(Embedding(input_dim=num_words,
                    output_dim=embedding_dim,
                    weights=[embedding_matrix],
                    trainable=False))
model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(Dropout(0.3)) # Meningkatkan Dropout untuk regularisasi
model.add(Bidirectional(LSTM(256)))
model.add(Dropout(0.3)) # Tambahkan Dropout setelah layer kedua
model.add(Dense(3, activation='softmax'))
```

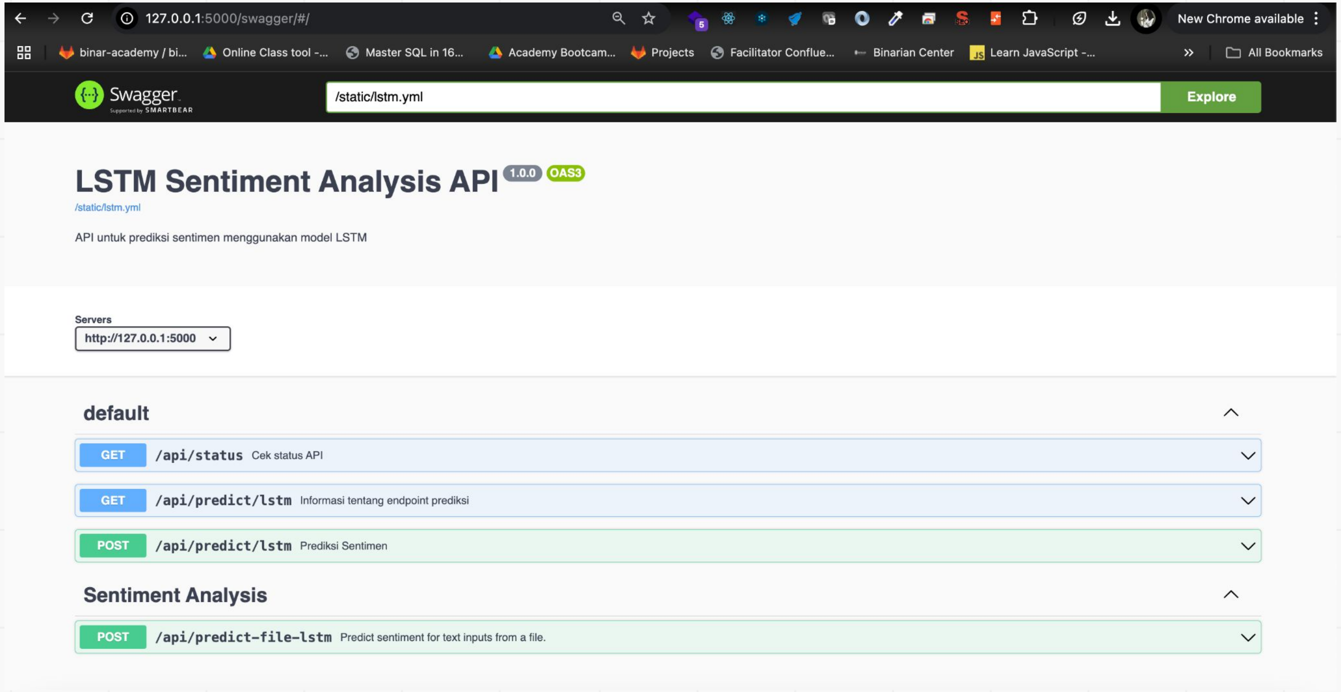
codesnap.dev



```python
# Mengompilasi model
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-7)

y_train = y_train.reset_index(drop=True)

# Melatih model
history = model.fit(X_train, y_train,
                    validation_split=0.2,
                    epochs=50,
                    batch_size=32,
                    class_weight=class_weights_dict,
                    callbacks=[early_stopping, reduce_lr],
                    verbose=1)
```

codesnap.dev

LSTM API Result



127.0.0.1:5000/swagger/#/

/static/lstm.yml

LSTM Sentiment Analysis API 1.0.0 OAS3

/static/lstm.yml

API untuk prediksi sentimen menggunakan model LSTM

Servers

http://127.0.0.1:5000

default

GET `/api/status` Cek status API

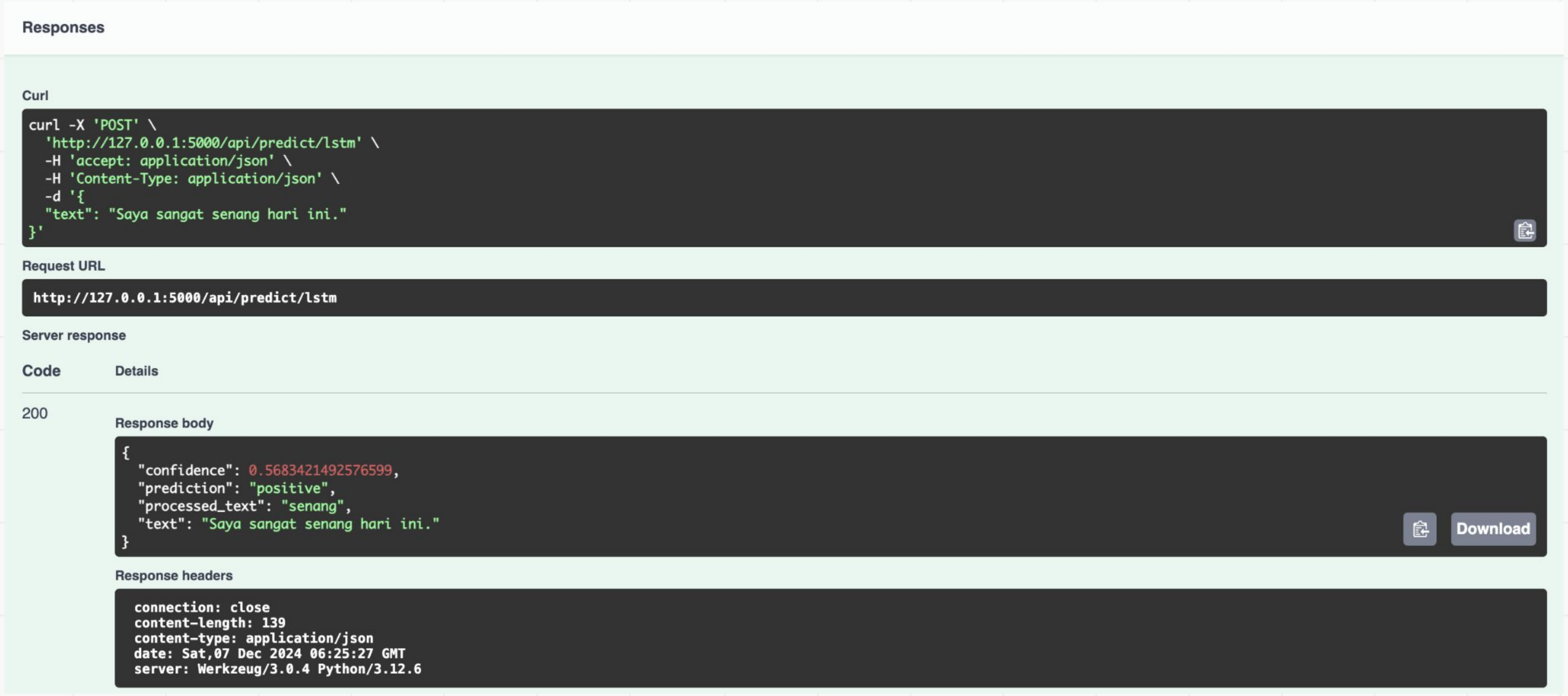
GET `/api/predict/lstm` Informasi tentang endpoint prediksi

POST `/api/predict/lstm` Prediksi Sentimen

Sentiment Analysis

POST `/api/predict-file-lstm` Predict sentiment for text inputs from a file.

LSTM (Text)



Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:5000/api/predict/lstm' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "text": "Saya sangat senang hari ini."
}'
```

Request URL

```
http://127.0.0.1:5000/api/predict/lstm
```

Server response

Code Details

200

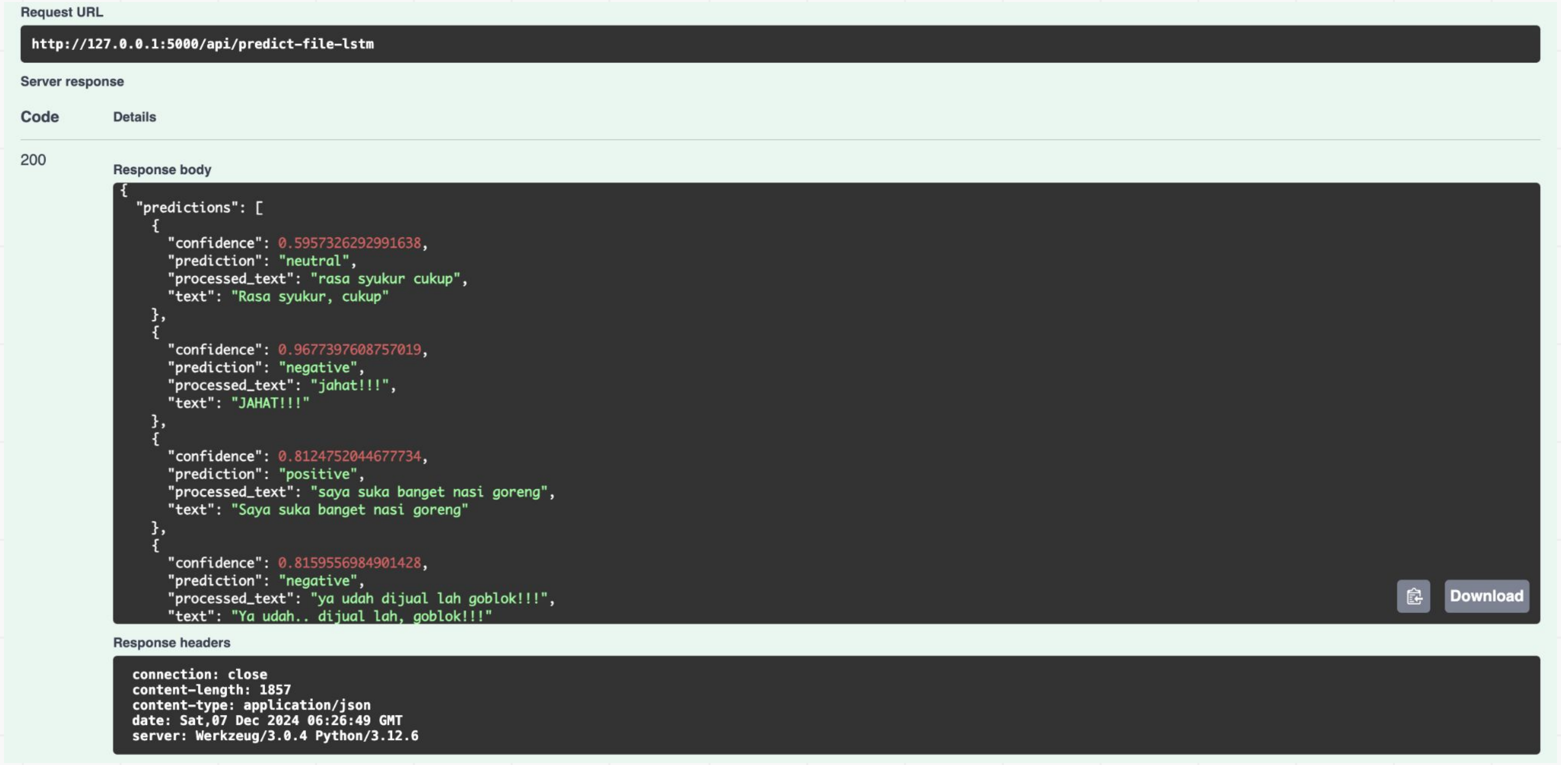
Response body

```
{
  "confidence": 0.5683421492576599,
  "prediction": "positive",
  "processed_text": "senang",
  "text": "Saya sangat senang hari ini."
}
```

Response headers

```
 connection: close
 content-length: 139
 content-type: application/json
 date: Sat,07 Dec 2024 06:25:27 GMT
 server: Werkzeug/3.0.4 Python/3.12.6
```

LSTM (File)



Request URL

```
http://127.0.0.1:5000/api/predict-file-lstm
```

Server response

Code Details

200

Response body

```
{
  "predictions": [
    {
      "confidence": 0.5957326292991638,
      "prediction": "neutral",
      "processed_text": "rasa syukur cukup",
      "text": "Rasa syukur, cukup"
    },
    {
      "confidence": 0.9677397608757019,
      "prediction": "negative",
      "processed_text": "jahat!!!",
      "text": "JAHAT!!!"
    },
    {
      "confidence": 0.8124752044677734,
      "prediction": "positive",
      "processed_text": "saya suka banget nasi goreng",
      "text": "Saya suka banget nasi goreng"
    },
    {
      "confidence": 0.8159556984901428,
      "prediction": "negative",
      "processed_text": "ya udah dijual lah goblok!!!",
      "text": "Ya udah.. dijual lah, goblok!!!"
```

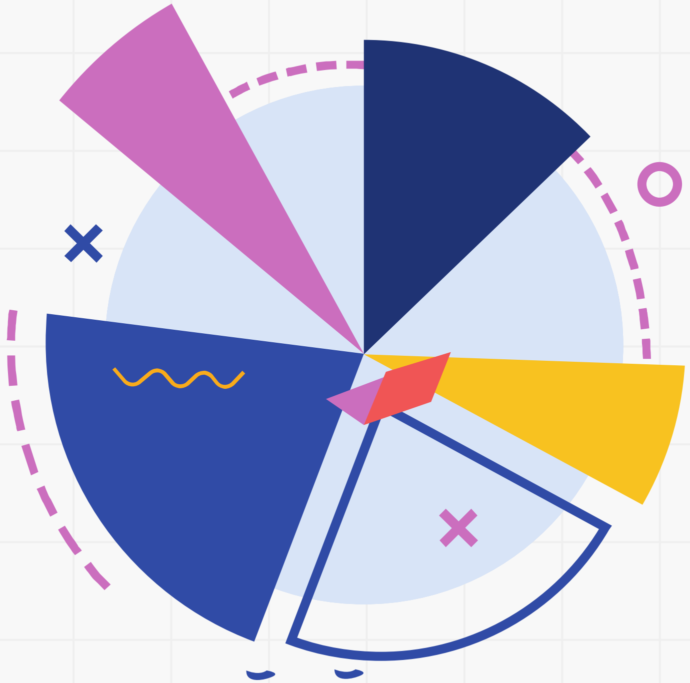
Response headers

```
connection: close
content-length: 1857
content-type: application/json
date: Sat,07 Dec 2024 06:26:49 GMT
server: Werkzeug/3.0.4 Python/3.12.6
```

03

Visualization

Visualization Process



Example Visualization (Countplot)

Sentiment Distribution

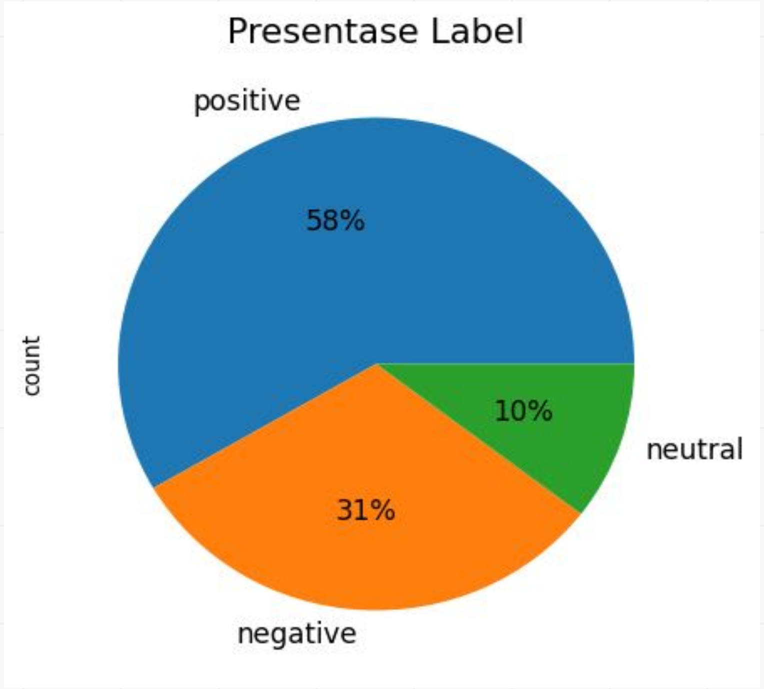
From that pie chart, we can see that:

1. **Positive Sentiment**:
   - Dominates the dataset, accounting for **58%** of the total data.
   - Indicates that most data reflect positive opinions or tones.

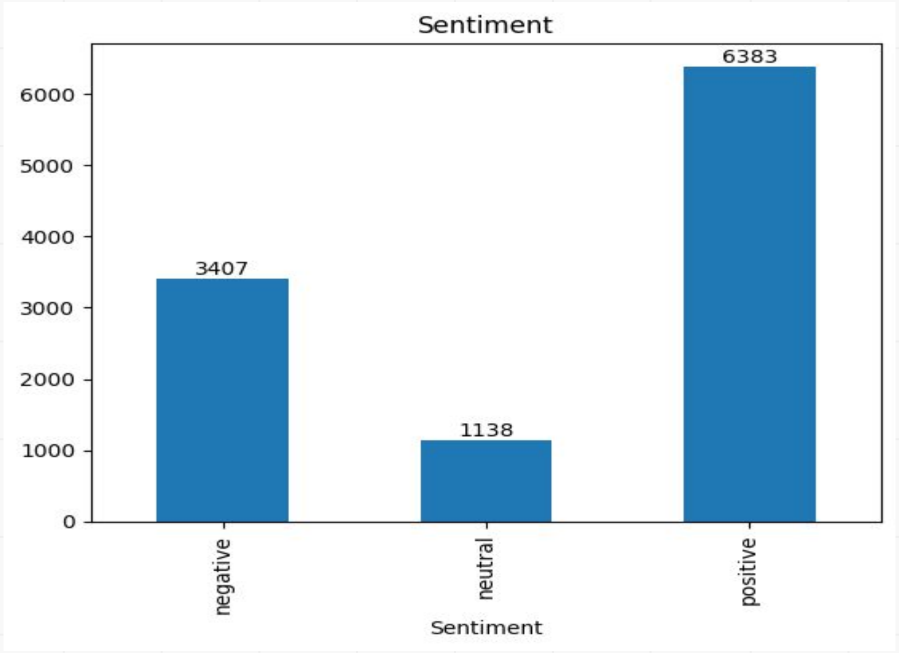
2. **Negative Sentiment**:
   - Makes up **31%** of the total data.
   - Represents nearly one-third of the dataset with negative opinions.

3. **Neutral Sentiment**:
   - The smallest portion, comprising only **10%**.
   - Shows that neutral data is relatively minimal.

The visualizations from this analysis are also available in **Google Colab** at [here]



Example Visualization (Countplot)



Sentiment Counts

Out of 10.928 data, the **positive sentiment** makes up the majority with **6.383** data, followed by **negative sentiment** with **3.407**, and **neutral sentiment** with only **1.138**

This indicates a dominant positive sentiment in the dataset, with significantly fewer neutral and negative sentiments.

04

Results and Conclusion

Results and Conclusion

Based on pie and bar charts, the sentiment analysis reveals the following insight:

1. **Positive Sentiment**
   Dominates both in **count 6,383 entries** and **percentage** (**58%**), indicating that most of the dataset is skewed toward positive opinions.
2. **Negative Sentiment**
   Represents a significant portion with **3,407 entries** (**31%**), showing that negative feedback or sentiments are also common, though not as prevalent as positive ones.
3. **Neutral Sentiment**
   Accounts for the smallest share with **1,138 entries** (**10%**), indicating that neutral sentiments are relatively rare.

Conclusion:
The dataset is predominantly positive, followed by a notable presence of negative sentiment, while neutral sentiment appears to be minimal. This distribution highlights a tendency toward polarized sentiments, with positive opinions being the most dominant.

Manual Calculation Result

1. Forward Pass
   Hasil dari penghitungan output neuran (y4, y5, y6,) dan Error

Y_4	Y_5	Y_6	e
0,3751	0,7483	0,2080	-0,2080

2. Backward Pass
   Hasil perhitungan (**δ6**), weight correction dan hidden layer (**δ4,δ5**)

δ_6	∇_{46}	∇_{56}	$\nabla\theta_6$
- 0,0342	- 0, 00128	0, 00255	0, 00342

δ_4	δ_5
0,008818	0,004509

Hasil perhitungan Weight Correction

∇w_{14}	∇w_{24}	∇w_{34}	$\nabla\theta_4$	∇w_{15}	∇w_{25}	∇w_{35}	$\nabla\theta_5$
0,00061726	0,00070544	0,00079362	-0,0008818	0,00031563	0,00036072	0,00040581	-0,0004509

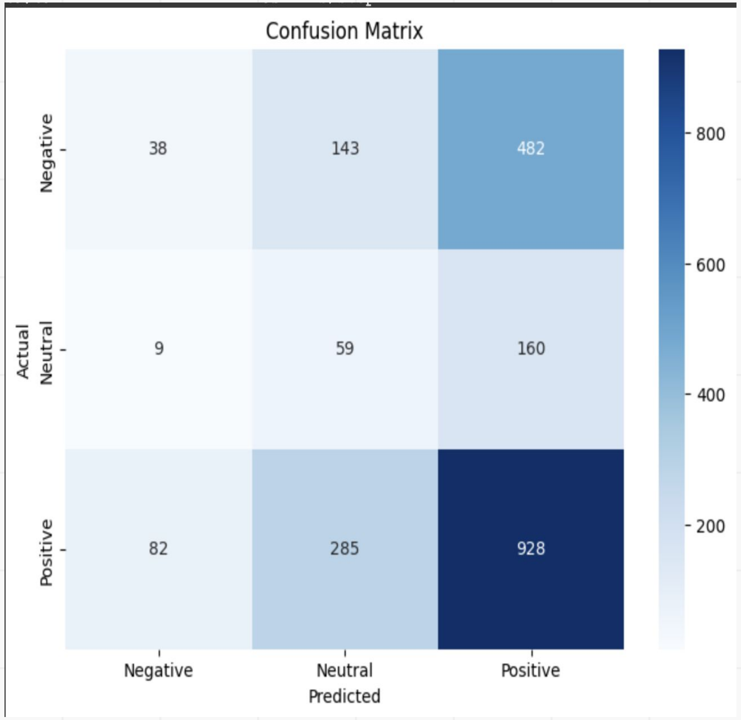
3. Hasil Akhir
   Hasil hitung dari semua weight dan theta menggunakan arsitektur yang telah diperbarui

w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	θ_3	θ_4	θ_5
0,5006	0,6003	0,3007	1,1003	-0,9992	0,1004	0,1991	0,3004	0,3004

Results and Conclusion (Neural Network)

Conclusion:
Confusion matrix adalah representasi matriks dari hasil prediksi model untuk setiap kategori. Sumbu Y menunjukkan kategori actual (label sebenarnya), sedangkan sumbu X menunjukkan kategori predicted (prediksi model). Warna semakin gelap menunjukkan jumlah yang lebih besar pada kotak terkait.

Negative (baris pertama): Model hanya memprediksi 38 kasus dengan benar sebagai negatif, sementara 143 dan 482 kasus negatif diprediksi sebagai neutral atau positif.**Kesimpulan:** Model memiliki kesulitan dalam memprediksi kategori negatif dengan akurat.
Neutral (baris kedua): Model memprediksi 59 kasus dengan benar sebagai neutral, tetapi banyak kasus neutral diprediksi sebagai positif (160) atau negatif (9).**Kesimpulan:** Kategori neutral juga kurang jelas dipahami oleh model.
Positive (baris ketiga): Sebagian besar kasus positif diprediksi dengan benar (928), tetapi beberapa diprediksi salah sebagai neutral (285) atau negatif (82).**Kesimpulan:** Model memiliki kinerja terbaik dalam kategori positif dibandingkan yang lain.

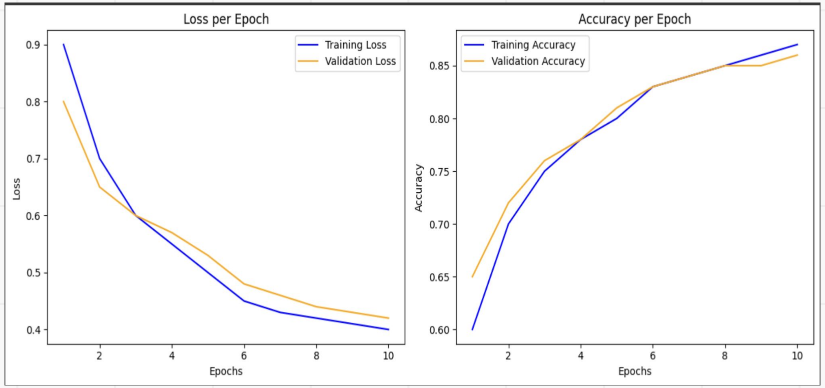


Results and Conclusion (Neural Network)

Conclusion:

Kiri (Loss per Epoch): Grafik ini menunjukkan perkembangan *loss* (kesalahan) selama pelatihan dan validasi.**Training Loss (garis biru):** Menunjukkan bagaimana kesalahan model pada data pelatihan berkurang seiring bertambahnya jumlah epoch.**Validation Loss (garis oranye):** Mengindikasikan kesalahan pada data validasi, yang membantu menilai kemampuan generalisasi model.

Interpretasi: Kedua *loss* (training dan validation) menurun, yang menunjukkan bahwa model belajar dengan baik.Jika *validation loss* mendatar atau meningkat sementara *training loss* terus menurun, itu dapat menunjukkan *overfitting*. Namun, di sini, *validation loss* juga menurun dengan baik.



Kanan (Accuracy per Epoch): Grafik ini memperlihatkan akurasi model pada data pelatihan dan validasi. **Training Accuracy (garis biru):** Menunjukkan persentase prediksi yang benar pada data pelatihan. **Validation Accuracy (garis oranye):** Menunjukkan akurasi pada data validasi, yang menilai performa pada data yang tidak dilihat model selama pelatihan.

Interpretasi: Akurasi meningkat baik pada data pelatihan maupun validasi, menunjukkan model belajar dan menggeneralisasi dengan baik. Akurasi validasi sedikit mendekati pelatihan, yang mengindikasikan model cukup seimbang tanpa tanda-tanda *overfitting* atau *underfitting* yang signifikan.

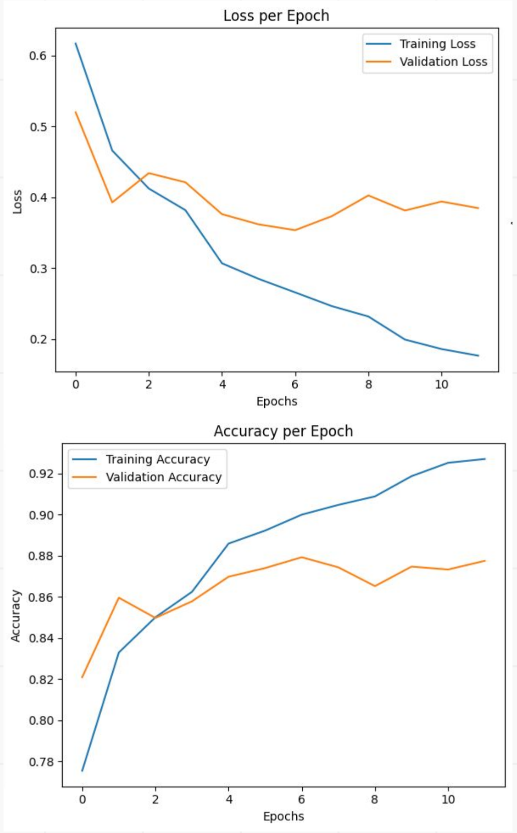
Results and Conclusion (LSTM)

Result Overview:

1. **Model Performance Metrics:**
   - **Test Loss:** 0.3738
   - **Test Accuracy:** 86.93%

   The low test loss indicates that the model has learned to minimize errors effectively. An accuracy of nearly 87% reflects the model's capability to classify sentiments correctly across the dataset.

2. **Training and Validation Metrics:**
   - The training and validation loss curves demonstrate steady convergence, with no significant overfitting or underfitting issues.
   - The accuracy curves for training and validation data consistently improve, indicating the model's learning progression.

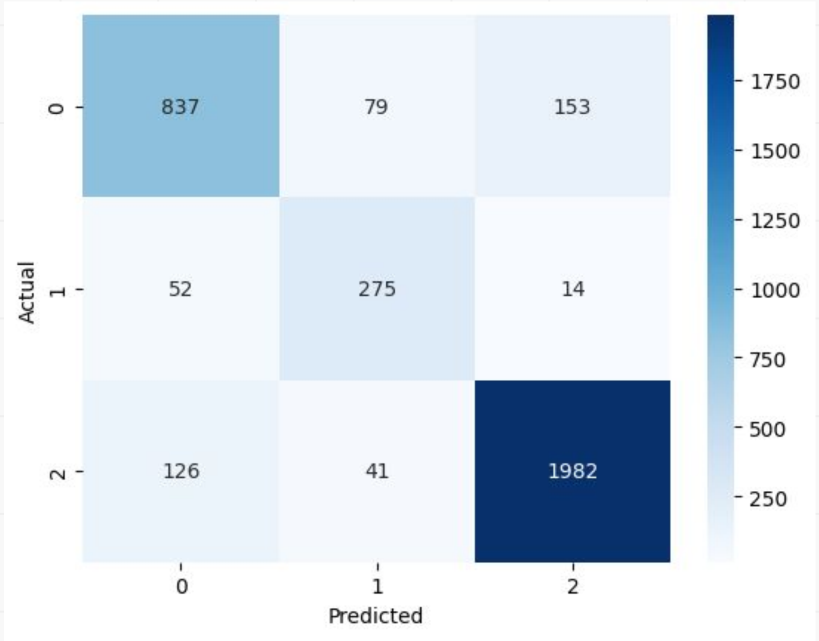


Results and Conclusion (LSTM)

Result Overview:

3. Confusion Matrix Analysis:

- Class **0 (Neutral):**
  - Correctly classified: 837
  - Misclassified as Class 1: 79
  - Misclassified as Class 2: 153
- Class **1 (Negative):**
  - Correctly classified: 275
  - Misclassified as Class 0: 52
  - Misclassified as Class 2: 14
- Class **2 (Positive):**
  - Correctly classified: 1982
  - Misclassified as Class 0: 126
  - Misclassified as Class 1: 41



Results and Conclusion (LSTM)

Conclusion:

1. **Strengths:**
   - The model performs exceptionally well for **Class 2 (Positive)** with minimal misclassification errors.
   - Training and validation loss convergence highlights robust model generalization.

2. **Areas for Improvement:**
   - Misclassifications are observed mainly for Class **0 (Neutral)** and Class **1 (Negative)**. These errors may stem from overlapping sentiment expressions in the dataset.
   - The performance for **Class 1 (Negative)** could be further improved by increasing the representation of negative samples during training or refining feature extraction.

Thanks!

Github Project:

https://github.com/isumizumi/DSC25-PlatinumChallenge

Trello Project:

https://trello.com/b/6o5SKrfP/datascienceplatinum-challenge