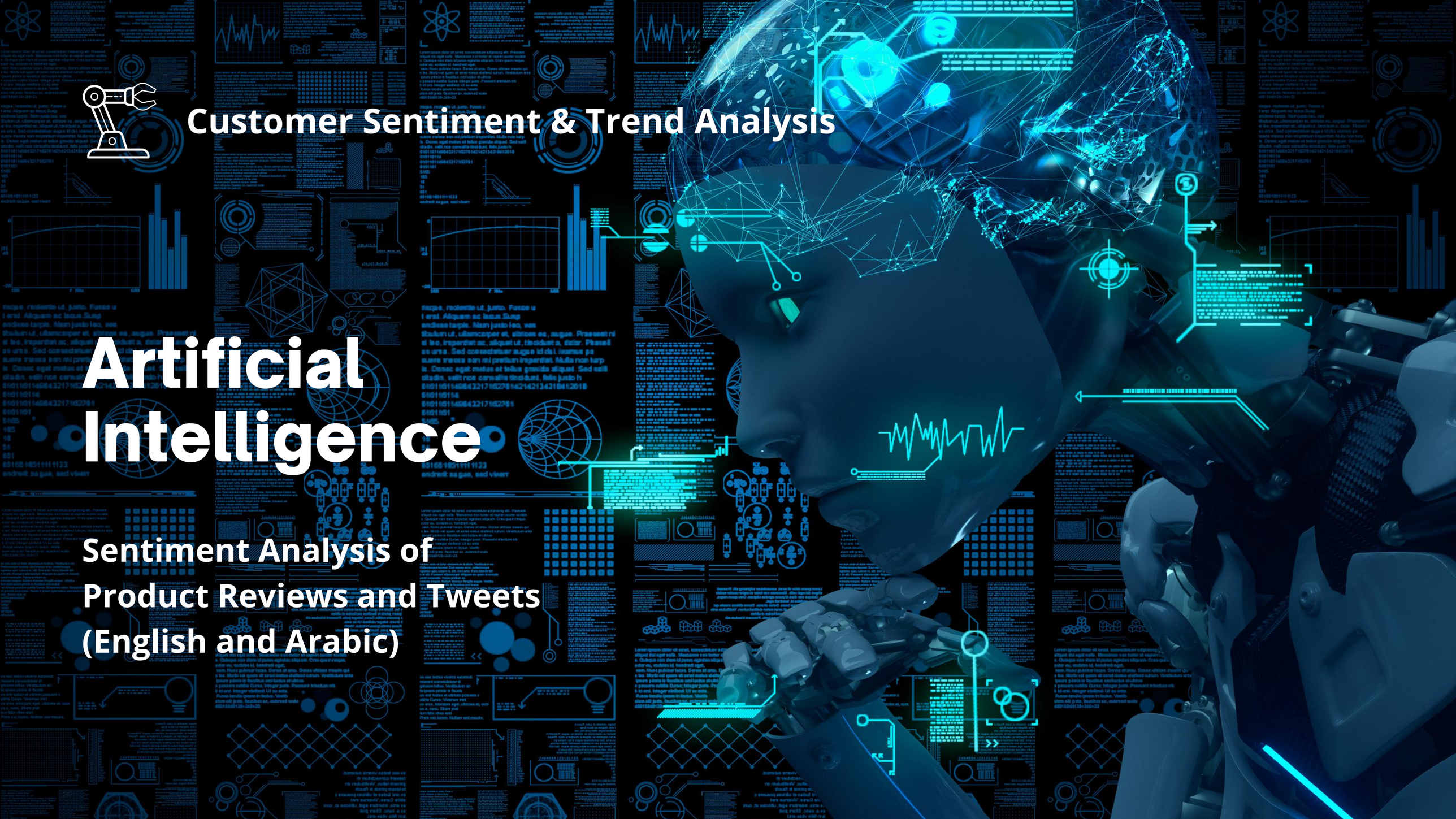
Customer Sentiment & Trend Analysis

Artificial Intelligence

Sentiment Analysis of Product Reviews and Tweets (English and Arabic)



Team members

- Ahmed Abdelhamed Hussein
- Ahmed Awad Ata
- Ahmed Mohamed Emad
- Ahmed Samy Farahat Abdelfatah
- Amr Maher Abdallah Mohamed
- Mohamed Wael Mohamed Khalifa

الفريق الأول	الفريق الثاني	الفريق الثالث
فرحات + عوض	عمرو + عماد	وائل + عبد الحميد
Sentiment140	Amazon Reviews	IMDB Movie Reviews

الفريق	الأعضاء	المهام
فريق MLOps	Farahat - Wael - Emad -	- استخدام MLflow لتتبع وإدارة نماذج التعلم الآلي. - إعداد التقارير الخاصة بإدارة النماذج. - ضمان تكامل النماذج مع البنية التحتية الحالية.
فريق GANs	Amr - Abdelhamid - Awad -	- بناء وتنفيذ نموذج Generative Adversarial Network (GAN). - توليد بيانات اصطناعية لتحليل تعليقات العملاء. - اختبار وتحسين أداء النموذج.
فريق العرض النهائي والتقارير	Farahat - Amr - Wael - Abdelhamid - Awad - Emad -	- إعداد التقرير النهائي للمشروع. - تحضير العرض التقديمي للعرض النهائي. - جمع المعلومات وتنسيق الشرائح. - مراجعة وتحرير المحتوى لضمان الجودة.

TABLE OF CONTENTS

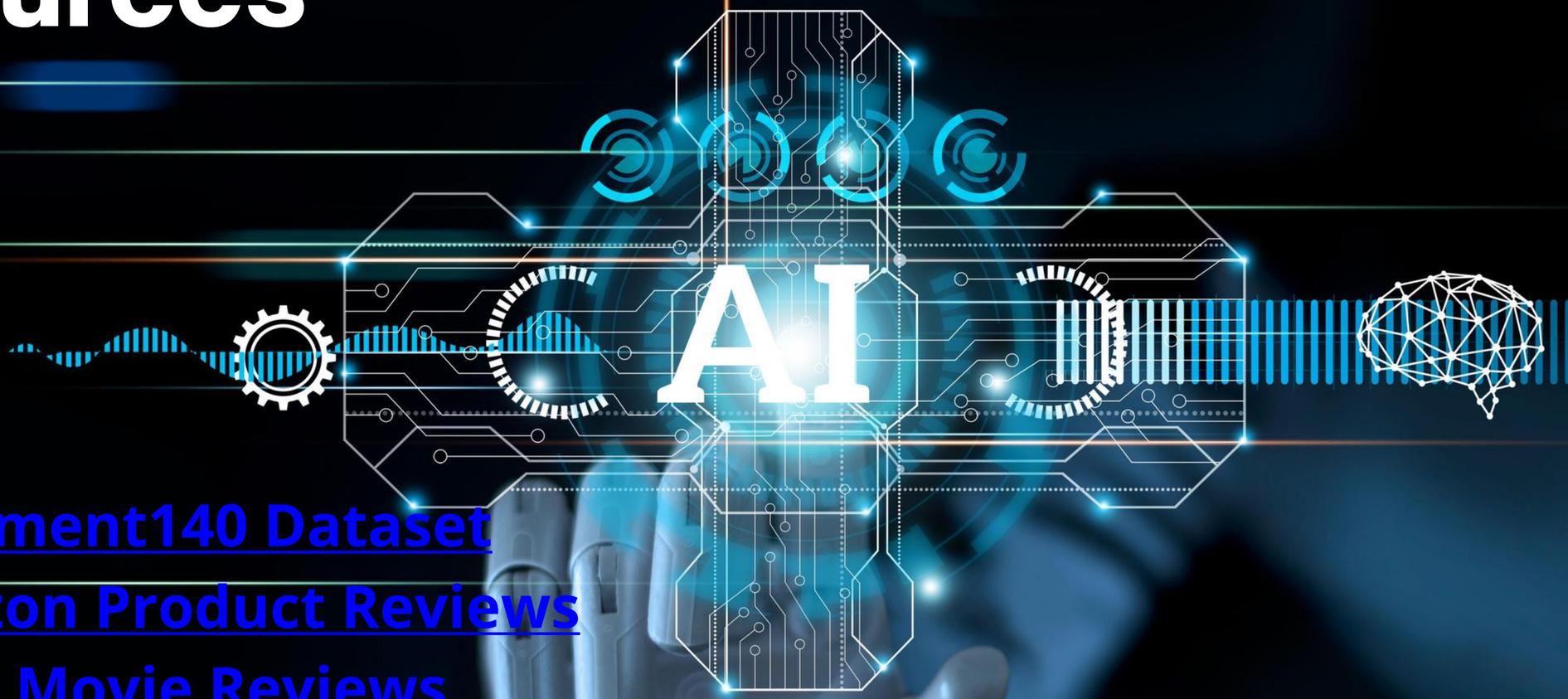
Project Overview

Objective: Analyze customer sentiment and uncover trends using datasets from product reviews and tweets in both English and Arabic.

Techniques: Apply NLP and machine learning techniques for sentiment classification (positive, negative, neutral).

Goal: Improve prediction accuracy using traditional machine learning and deep learning (e.g., transformers, LSTMs).

Data Sources



- Dataset 1: Sentiment140 Dataset
- Dataset 2: Amazon Product Reviews
- Dataset 3: IMDB Movie Reviews
- Dataset 4: Arabic Sentiment Analysis

METHODOLOGY

01

Data Preprocessing: Cleaning, tokenization, and padding to handle both English and Arabic text.

02

Feature Engineering: Using word embeddings (GloVe for English, custom embeddings for Arabic) to capture word semantics.

03

Model Training: Applying ML algorithms and deep learning models such as LSTMs, transformers, and RNNs.

Models & Techniques

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis tempor porttitor velit nec accumsan. Integer vehicula augue at purus maximus placerat.

In id nisl nec odio varius sodales. In pellentesque massa eu tempor congue. In in vestibulum nibh.

OUR PROJECTS

01 Sentiment Analysis of Twitter Data

02 Sentiment Analysis of IMDB Movie Reviews

03 Advanced Tracking and Prediction System

04 Sentiment Analysis of Arabic Tweets

05 Sentiment Analysis of Amazon reviews

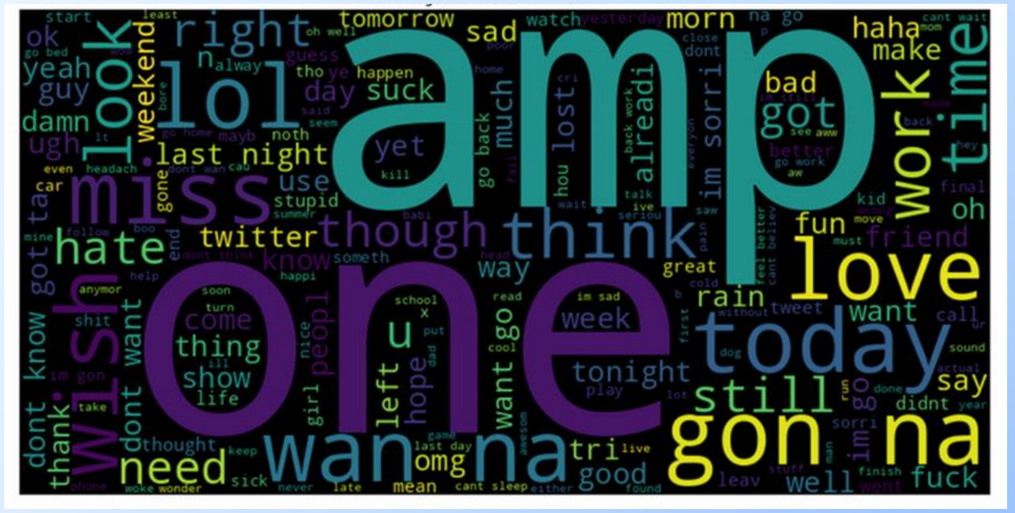
Sentiment Analysis of

Twitter Data

This project explores sentiment analysis using a dataset of 1.6 million tweets, categorized into positive and negative sentiments. The key steps include:
Data Preprocessing: Tweets were cleaned by removing URLs, mentions, numbers, and punctuation. Stopwords were also removed to improve model performance.

Positive Tweets Word Cloud

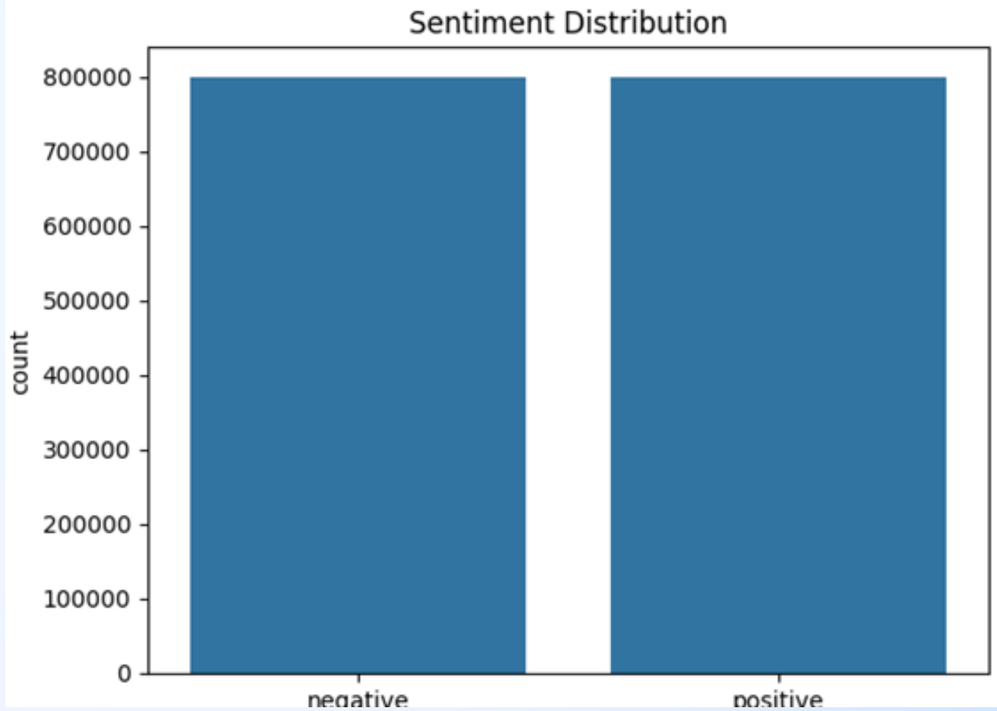
Negative Tweets Word Cloud



```
1          upset cant update facebook texting might cry r...
2          dived many times ball managed save rest go bounds
3                           whole body feels itchy like fire
4                                   behaving im mad cant see
                                ...
1599995                          woke school best feeling ever
1599996         thewdbcom cool hear old walt interviews â¦ «
1599997                        ready mojo makeover ask details
1599998    happy th birthday boo alll time tupac amaru sh...
1599999                                   happy charitytuesday
Name: cleaned_text, Length: 1600000, dtype: object
```

```python
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

df['cleaned_text'] = df['cleaned_text'].apply(lambda text: ' '.join([stemmer.stem(word) for word in text.split()]))

df['cleaned_text'] = df['cleaned_text'].apply(lambda text: ' '.join([lemmatizer.lemmatize(word) for word in text.split()]))
```

```python
sns.countplot(x='sentiment', data=df)
plt.title('Sentiment Distribution')
plt.show()
```



Text Tokenization and Padding: The text data was tokenized and transformed into sequences, which were padded to ensure uniform input length for the model.

[10]:
```
def get_most_common_words(texts, num=20):
    words = ' '.join(texts).split()
    counter = Counter(words)
    return counter.most_common(num)

print("Most common words in positive tweets:")
print(get_most_common_words(df[df['sentiment'] == 'positive']['cleaned_text']))

print("\nMost common words in negative tweets:")
print(get_most_common_words(df[df['sentiment'] == 'negative']['cleaned_text']))
```

```
Most common words in positive tweets:
[('im', 4590), ('love', 3893), ('good', 3859), ('go', 3698), ('day', 3485), ('than
k', 3159), ('get', 3072), ('like', 2509), ('u', 2322), ('lol', 2126), ('time', 210
7), ('got', 2019), ('work', 1844), ('today', 1818), ('know', 1758), ('see', 1747),
('one', 1744), ('watch', 1655), ('new', 1630), ('great', 1566)]

Most common words in negative tweets:
[('im', 6574), ('go', 4883), ('get', 3821), ('work', 3624), ('day', 3104), ('miss',
2978), ('cant', 2787), ('dont', 2748), ('like', 2578), ('got', 2446), ('want', 244
4), ('today', 2332), ('feel', 2257), ('back', 2000), ('realli', 1944), ('time', 189
7), ('u', 1822), ('one', 1796), ('sad', 1781), ('still', 1771)]
```

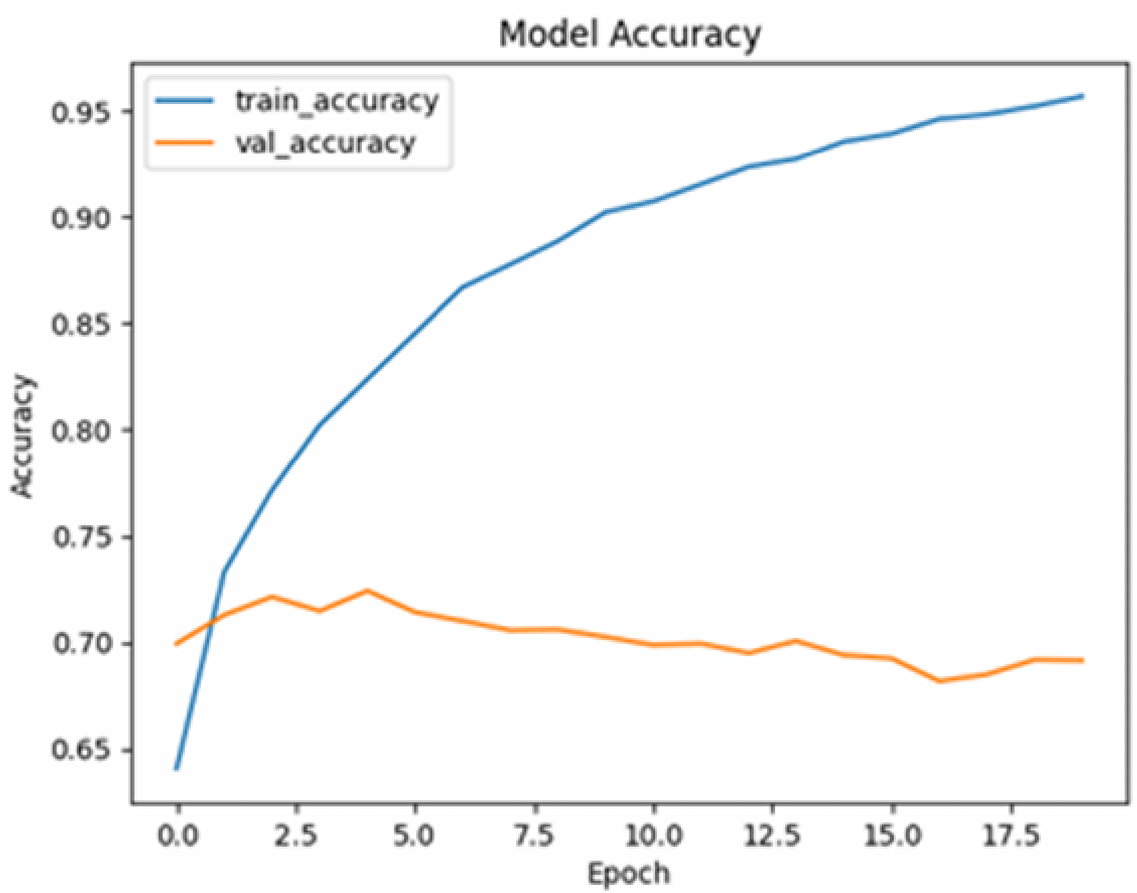
Feature Engineering: Both stemming and lemmatization techniques were applied to reduce words to their root form. Additionally, emoticons were replaced with corresponding textual representations.

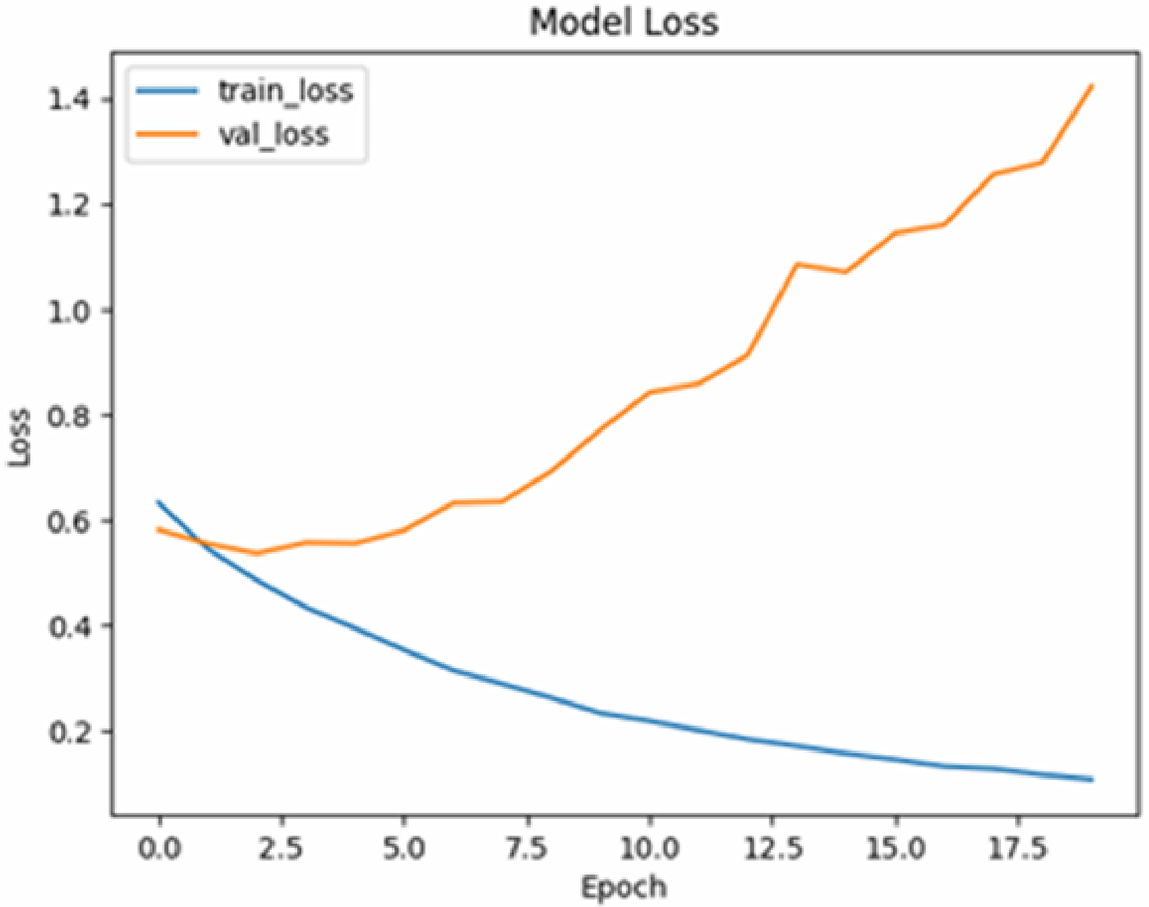
```
le = LabelEncoder()
y = le.fit_transform(y)  # Encode positive: 1, negative: 0

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
max_words = 10000  # Maximum number of words in the vocabulary
max_len = 100  # Maximum sequence length

tokenizer = Tokenizer(num_words=max_words, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Padding sequences to ensure uniform input size
X_train_padded = pad_sequences(X_train_seq, maxlen=max_len, padding='post', truncat
X_test_padded = pad_sequences(X_test_seq, maxlen=max_len, padding='post', truncatin
```

Model Training: A Bidirectional LSTM model was used for sentiment classification. The model leveraged GloVe pre-trained word embeddings for enhanced accuracy in understanding the semantic meaning of words.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 100, 100)          1000000

 bidirectional (Bidirectiona  (None, 256)               234496
 l)

 dense_1 (Dense)              (None, 1)                 257

=================================================================
Total params: 1,234,753
Trainable params: 1,234,753
Non-trainable params: 0
```


Model Accuracy
train_accuracy
val_accuracy
Accuracy
Epoch
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Model Loss
train_loss
val_loss
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Sentiment Analysis of *IMDB Movie Reviews*

Project Description:

This project focuses on sentiment analysis of the IMDB movie reviews dataset, which is a binary classification task. The objective is to classify reviews as either positive or negative using deep learning models. The notebook leverages multiple recurrent neural network (RNN) architectures to build, train, and evaluate the performance of these models.

Key Steps and Features

Data Preprocessing:
Dataset: The dataset consists of IMDB movie reviews labeled as either positive or negative.

Text Cleaning and Tokenization:
The reviews are first cleaned to remove unnecessary characters, such as punctuation and special symbols.
Text is tokenized using a tokenizer, converting the words into sequences of integers.
The sequences are then padded to ensure a uniform length across the dataset. This step is crucial since RNNs require fixed-length inputs.

Feature Engineering:
The text data is transformed into a format suitable for neural network models through embeddings. This allows the model to capture semantic relationships between words.

Search

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

Untitled (10).ipynb

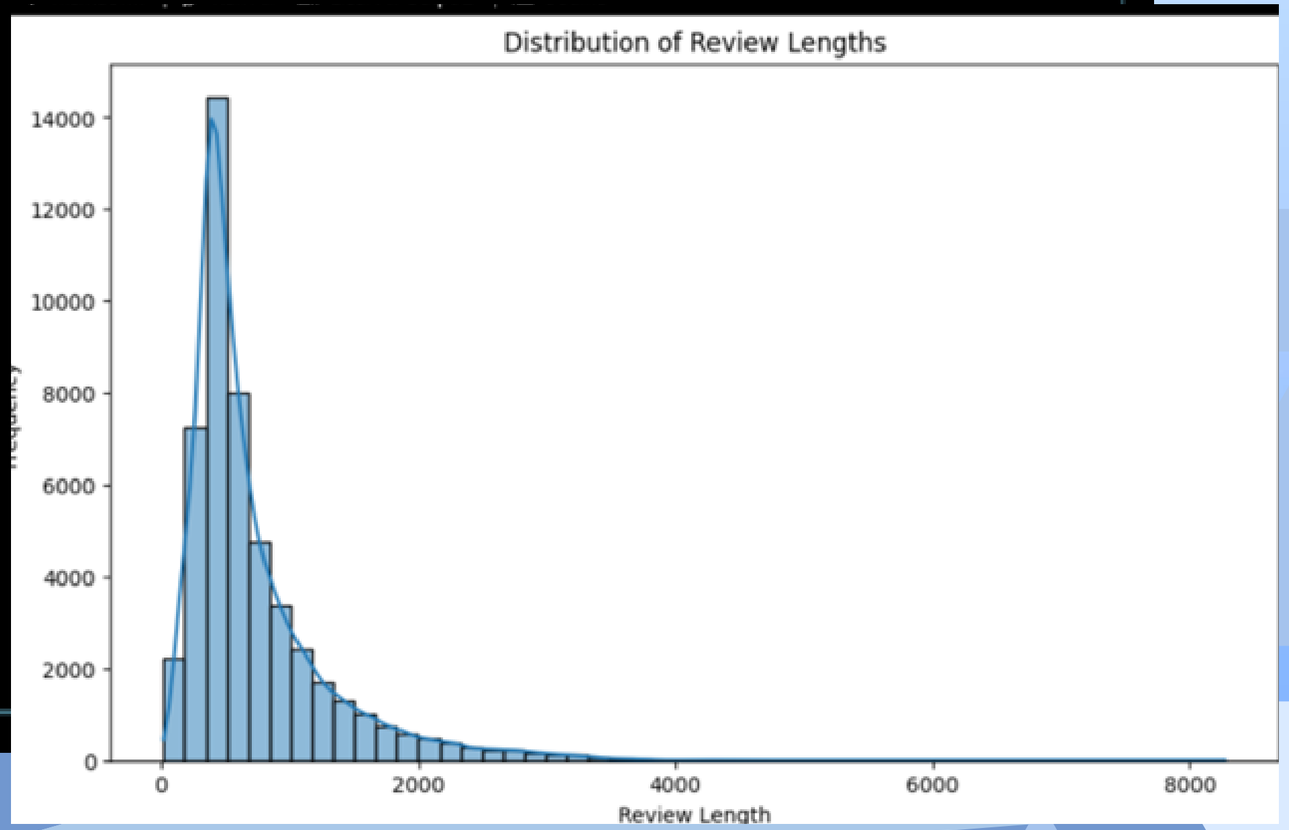
D: > Downloads > Downloads > Untitled (10).ipynb > import nltk

+ Code + Markdown | Run All Clear All Outputs | Outline ... Select Kernel

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()
def preprocess_text(sentence):

    sentence = re.sub(r"https?://\S+|www\.\S+", " ", sentence)


    sentence = re.sub(r"<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});", " ", sentence)

    sentence = re.sub(r"[^\w\s]", " ", sentence)

    sentence = re.sub(r"\w*\d\w*", " ", sentence)

    sentence = re.sub(r"[0-9]+", " ", sentence)

    sentence = re.sub(r"\s+", " ", sentence).strip()

    sentence = sentence.lower()

    tokens = []
    for token in sentence.split():
        if token not in stop_words:
            lemmatized_token = lemmatizer.lemmatize(token)
            stemmed_token = stemmer.stem(lemmatized_token)
            tokens.append(stemmed_token)

    return " ".join(tokens)
```

[12]

Restricted Mode ⊗ 7 ⚠ 56 0

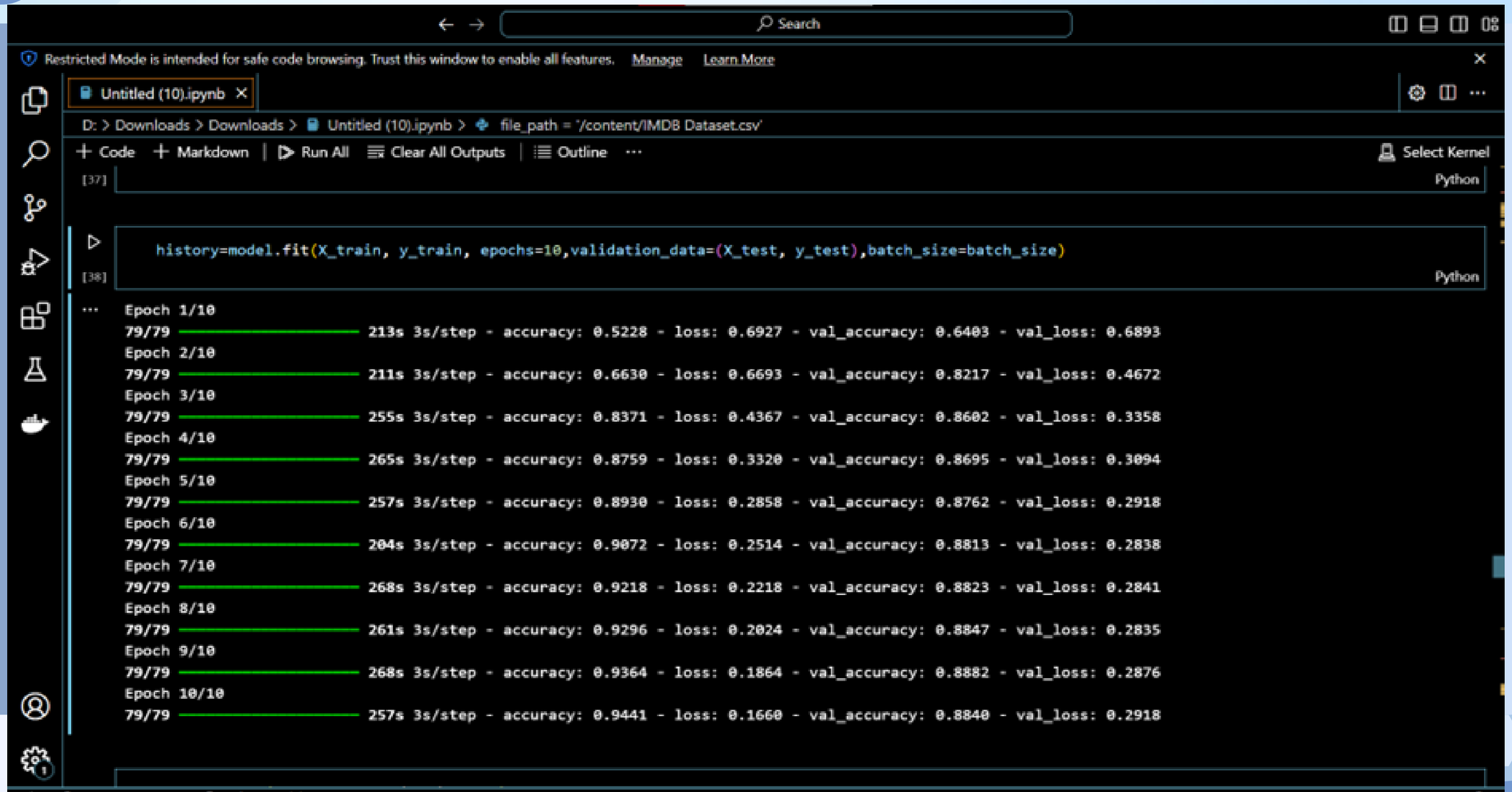


LSTM Model:

The LSTM model is designed to capture long-term dependencies in the text, which is essential for understanding the context in movie reviews.

Model Architectures: The notebook explores three types of RNN-based models to classify the reviews:

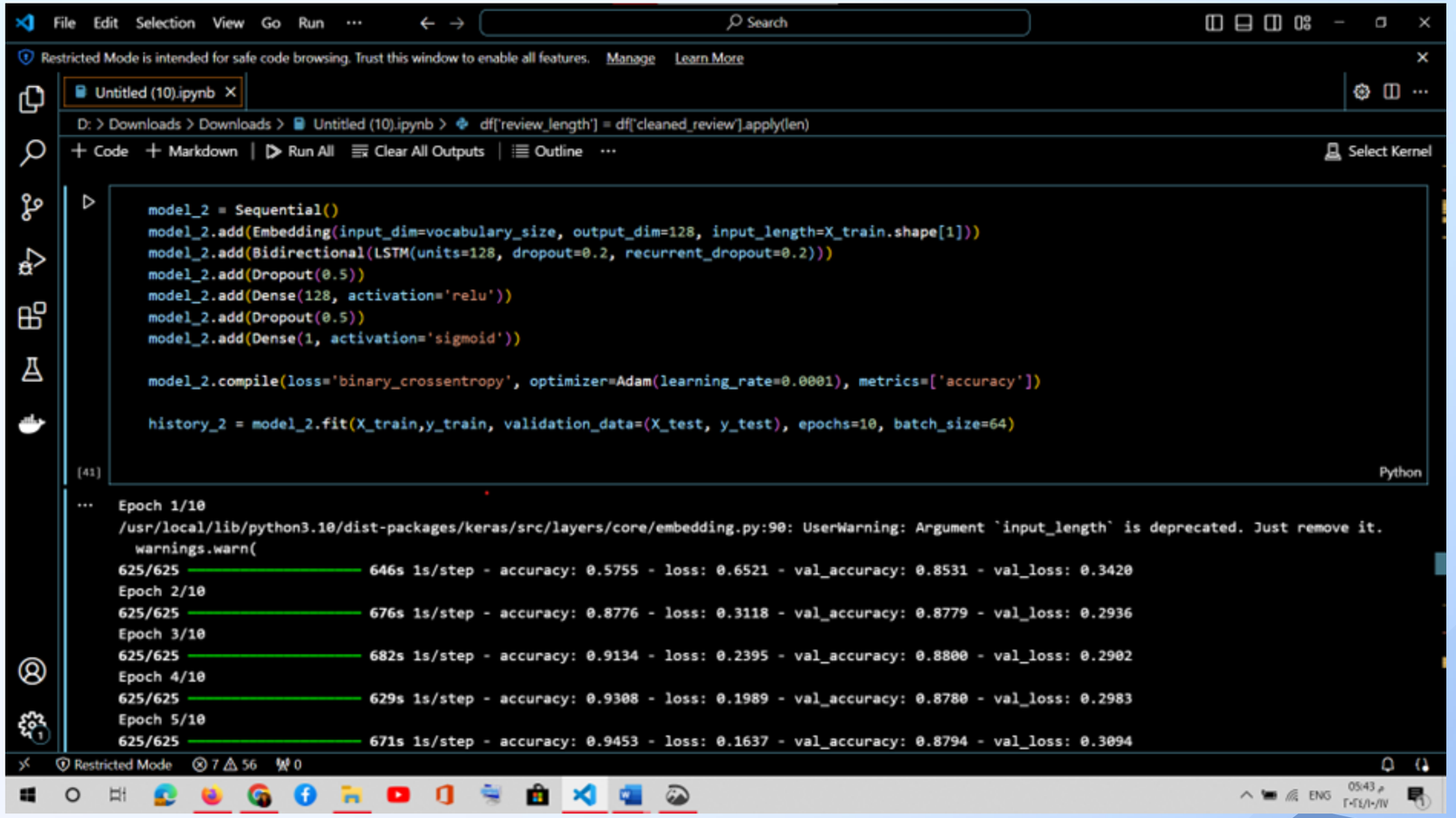
The model architecture includes an embedding layer, an LSTM layer, dropout layers to prevent overfitting, and a dense layer with a sigmoid activation function for binary classification.



Bidirectional LSTM Model:

This model enhances the LSTM by reading the sequence in both directions (forward and backward), improving the ability to understand context.

Similar to the LSTM model but with additional power due to bidirectional processing.



GRU Model

The Gated Recurrent Unit (GRU) model is another variant of RNN, offering a simpler structure compared to LSTMs, while still capturing temporal dependencies in the data. GRU models are often faster to train and require fewer computational resources.

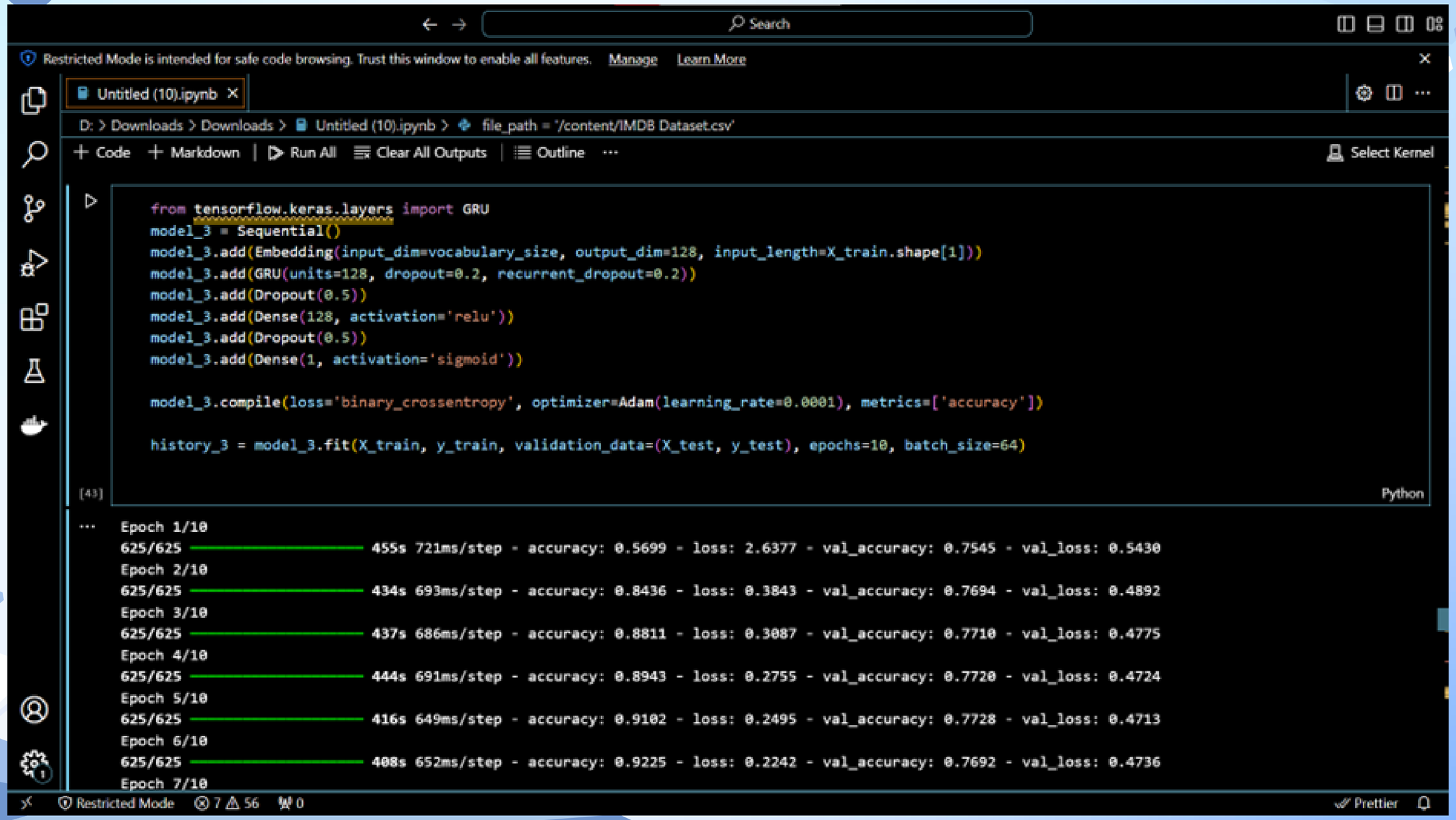
Training and Evaluation:

Training: Each model is trained on the preprocessed IMDB dataset using the Adam optimizer with binary cross-entropy loss. The models are trained for 10 epochs.

Evaluation Metrics: The performance is tracked using accuracy and loss. The training and validation accuracy are plotted to assess model performance.

Batch Size: The models are trained with a batch size of 64, balancing between computational efficiency and model convergence.

GRU Model

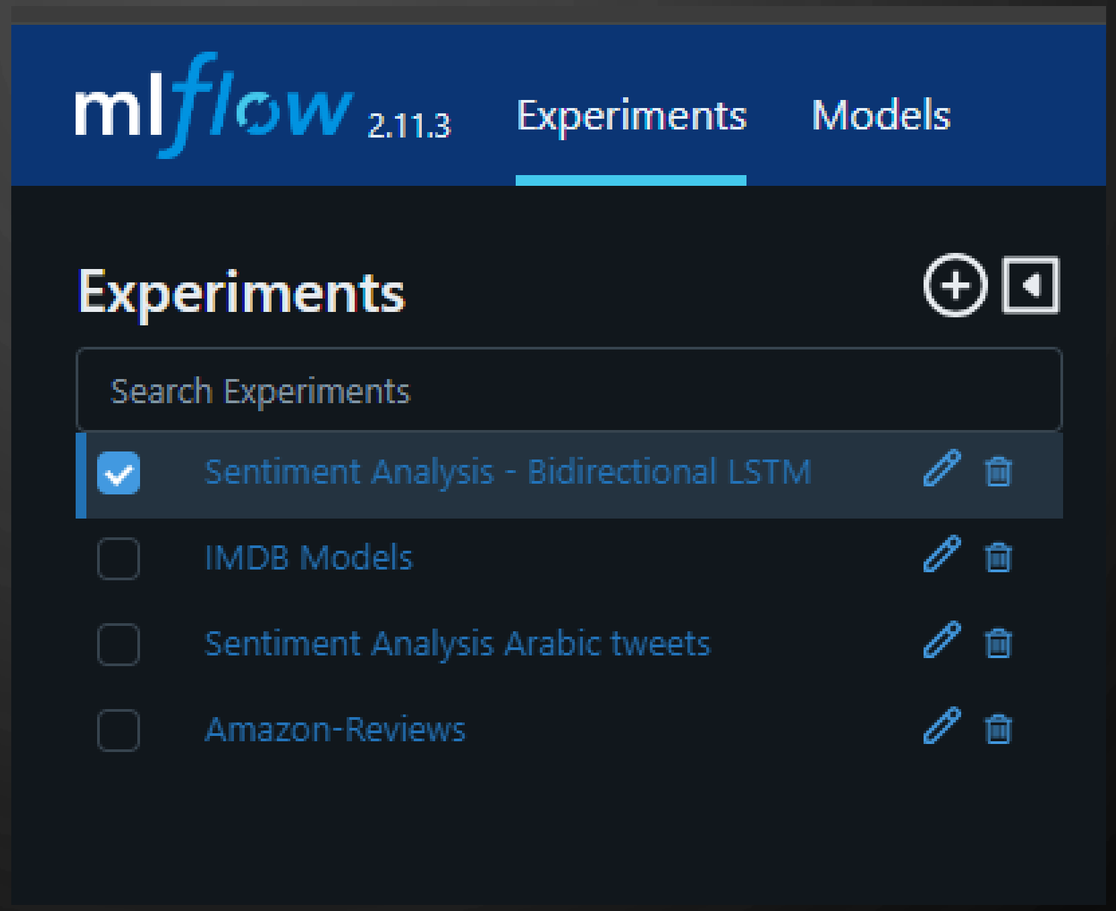

Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
Untitled (10).ipynb
D: > Downloads > Downloads > Untitled (10).ipynb > file_path = '/content/IMDB Dataset.csv'
+ Code + Markdown Run All Clear All Outputs Outline ··· Select Kernel
from tensorflow.keras.layers import GRU
model_3 = Sequential()
model_3.add(Embedding(input_dim=vocabulary_size, output_dim=128, input_length=X_train.shape[1]))
model_3.add(GRU(units=128, dropout=0.2, recurrent_dropout=0.2))
model_3.add(Dropout(0.5))
model_3.add(Dense(128, activation='relu'))
model_3.add(Dropout(0.5))
model_3.add(Dense(1, activation='sigmoid'))
model_3.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001), metrics=['accuracy'])
history_3 = model_3.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=64)
[43]
Python
Epoch 1/10
625/625 455s 721ms/step - accuracy: 0.5699 - loss: 2.6377 - val_accuracy: 0.7545 - val_loss: 0.5430
Epoch 2/10
625/625 434s 693ms/step - accuracy: 0.8436 - loss: 0.3843 - val_accuracy: 0.7694 - val_loss: 0.4892
Epoch 3/10
625/625 437s 686ms/step - accuracy: 0.8811 - loss: 0.3087 - val_accuracy: 0.7710 - val_loss: 0.4775
Epoch 4/10
625/625 444s 691ms/step - accuracy: 0.8943 - loss: 0.2755 - val_accuracy: 0.7720 - val_loss: 0.4724
Epoch 5/10
625/625 416s 649ms/step - accuracy: 0.9102 - loss: 0.2495 - val_accuracy: 0.7728 - val_loss: 0.4713
Epoch 6/10
625/625 408s 652ms/step - accuracy: 0.9225 - loss: 0.2242 - val_accuracy: 0.7692 - val_loss: 0.4736
Epoch 7/10
Restricted Mode 7 56 0
Prettier

MLFLOW SENTIMENT ANALYSIS PROJECT

COMPREHENSIVE MODEL TRAINING AND EVALUATION

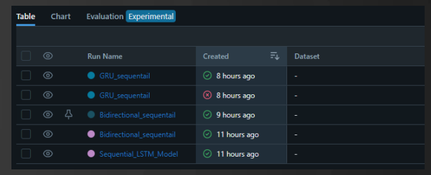
DATASETS USED

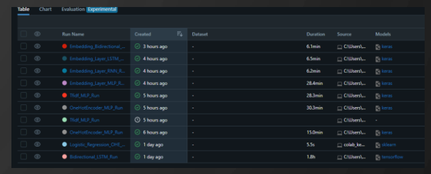
- We utilized well-known datasets for sentiment analysis:
- • Sentiment140 Dataset: 1.6M tweets labeled with sentiment.
- • Amazon Product Reviews: Amazon product reviews with sentiment labels.
- • IMDB Movie Reviews: 50k movie reviews labeled as positive or negative.
- • Arabic Sentiment Analysis: Arabic tweets with sentiment categories.

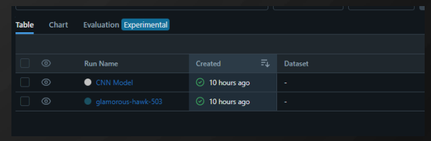


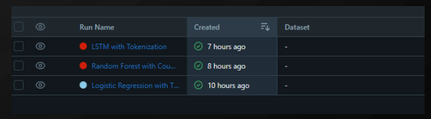
MODELS TRAINED

- Multiple models were trained and evaluated:
- • Logistic Regression
- • Support Vector Machine (SVM)
- • Random Forest
- • Bidirectional LSTM
- • Pre-trained Transformer (BERT)



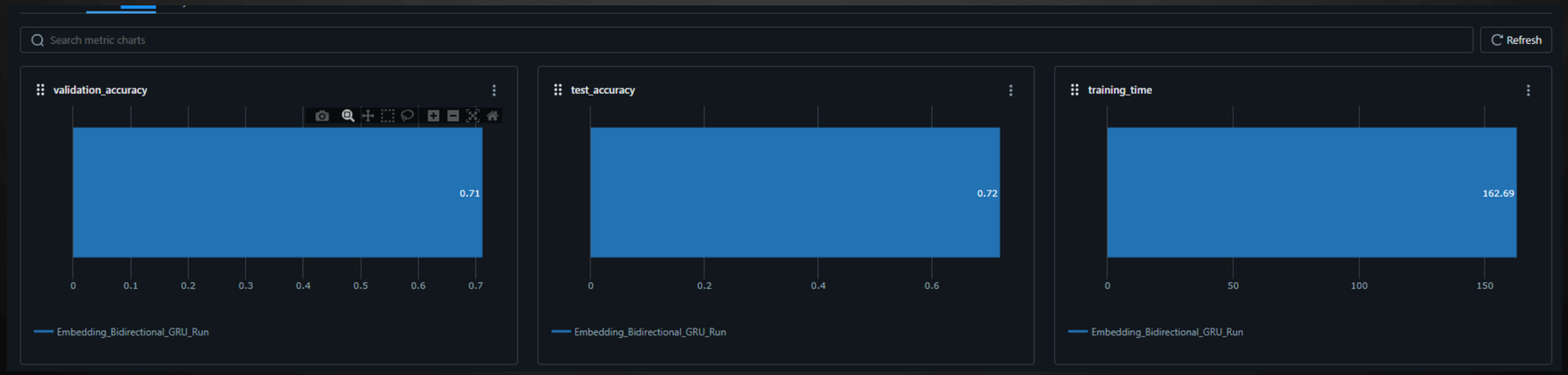






MLFLOW FOR MODEL TRACKING

- MLFlow was used for tracking the performance of different models:
- • Track metrics such as accuracy, precision, recall, and F1-score.
- • Compare model versions and hyperparameter tuning results.



LINK TO MLFLOW EXPERIMENT

- Access the full experiment details here:
- MLFlow Link: https://dagshub.com/asamy4194/DEPI_Final_Project.mlflow/#/experiments/0?viewStateShareKey=d537fdc0c959fa49a2e0b67727b54010fcd4ffd194d195a2598ac3fdf5c6e596

MLflow Integration

- MLflow for Collaboration: Linking team notebooks and tracking experiments to ensure all models and results are centrally managed.
- Experiment Tracking: Tracking model parameters, accuracy, and results for different datasets.

SENTIMENT ANALYSIS OF ARABIC TWEETS

USING DEEP LEARNING TO CLASSIFY SENTIMENTS

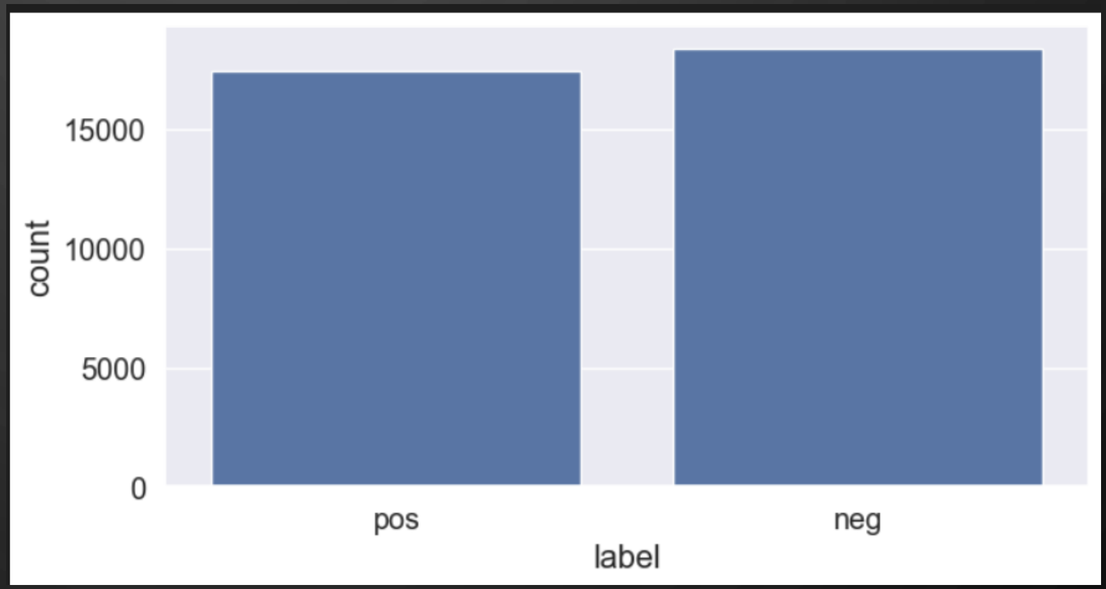
OVERVIEW OF THE PROJECT

Objective: Build a model to classify Arabic tweets into positive and negative sentiments.

Scope: Utilize deep learning and natural language processing (NLP) for sentiment analysis.

DATA COLLECTIO N

- Datasets: Merged two datasets of positive and negative Arabic tweets.
- Data Cleaning: Removed duplicates, handled missing values.



PREPROCESSING STEPS

Stop Words Removal: Custom and existing stop words were used.

Emoji/Text Handling: Converted emojis, cleaned URLs, mentions, and punctuation.

Text Normalization: ISRI Stemmer and Arabic Light Stemmer were used.

FEATURE ENGINEERIN G

- Tokenization & Padding: Keras tokenizer applied to standardize input length.
- Vocabulary Size Limitation: Most frequent words were selected to reduce complexity.

```
print(X[:5])

print('#' * 50)

print(y[:5])
```

```
0    مناسب وز هلال حب سحب على يفو  رتوي تابع سحب س...
1    اتحاد نصر اتحسب سي بالطواق انب وقف مع ماح حاجت...
2    مفروض كو حساب موبايل ويتر يافيس لي عند يس مينز...
3                  ول اسفل باك حال BACK      …
4                          جي عوض معليش مكروف عرب رم هه
Name: text, dtype: object
##################################################
0    pos
1    pos
2    neg
3    neg
4    neg
Name: label, dtype: object
```

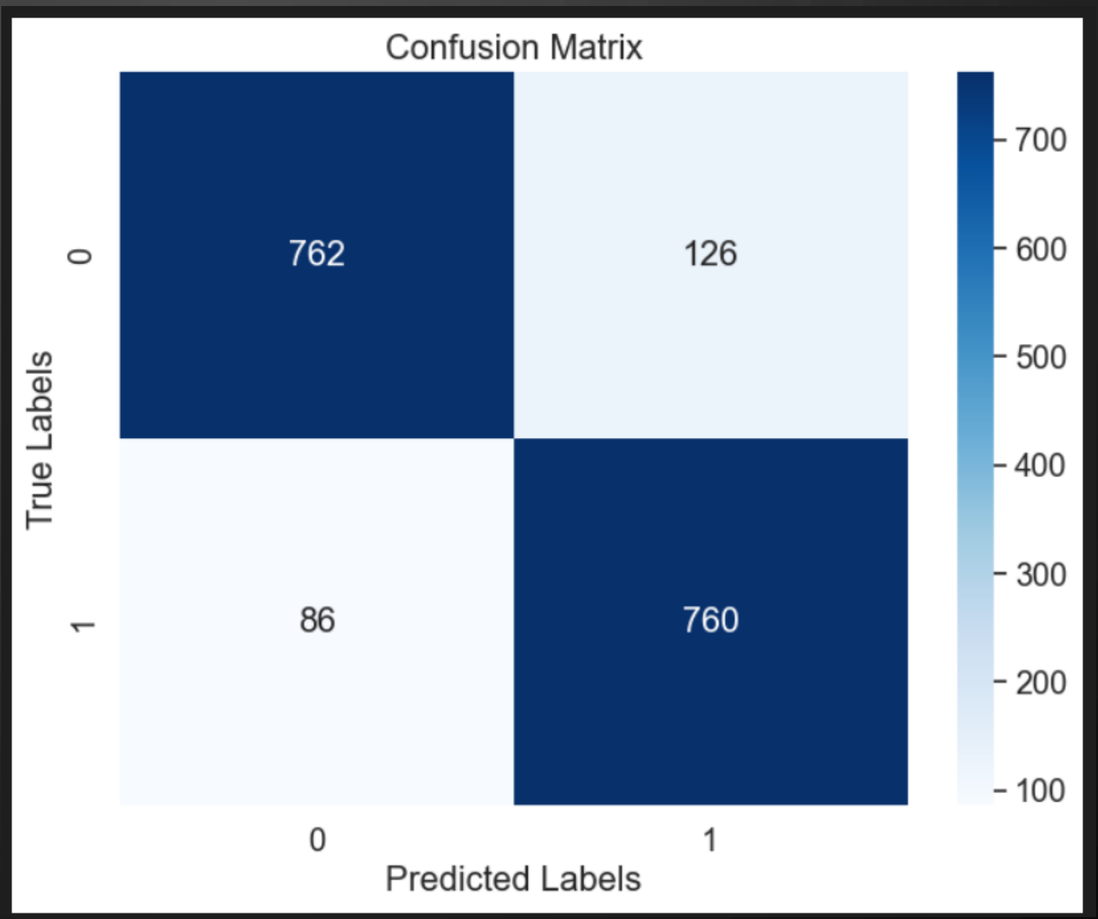
MODEL BUILDING

- Model: Bidirectional LSTM to capture sequential dependencies.
- Training & Validation: Trained on 30,000 tweets, validated with a test set.
- Overfitting Prevention: Applied dropout and batch normalization.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 64)          1754048

 bidirectional (Bidirectiona  (None, None, 128)        66048
 l)

 dropout (Dropout)           (None, None, 128)         0

 batch_normalization (BatchN  (None, None, 128)        512
 ormalization)

 bidirectional_1 (Bidirectio  (None, 64)               41216
 nal)

 dropout_1 (Dropout)         (None, 64)                0

 batch_normalization_1 (Batc  (None, 64)               256
 hNormalization)

 dense (Dense)               (None, 32)                2080

 dropout_2 (Dropout)         (None, 32)                0
...
Total params: 1,864,226
Trainable params: 1,863,842
Non-trainable params: 384
```

RESULTS AND FUTURE WORK

- Accuracy: 96.2% on training, 87.7% on validation.
- Confusion Matrix & Accuracy Graph: Used to visualize performance.
- Future Work: Expand dataset, fine-tune hyperparameters, and incorporate more advanced techniques.



Sentiment Analysis of

Amazon reviews

Taking sample from train data and test data:

In [3]:
```
train.columns = ["polarity","title", "text"]
train = train.sample(100000,random_state = 99)
train.head()
```

Out[3]:

	polarity	title	text
2136295	2	Making a suit of human skin is hard work. Real...	Yes, go ahead and laugh. But some of the other...
2620775	2	If Only Amazon allowed my 6 *s.	Wow,This movie is a classic. Family movie by a...
1588872	1	I THOUGHT APPLE WAS USER FRIENDLY	I THOUGHT APPLE WAS USER FRIENDLY???????? I GU...
3125694	2	Very Helpful	I bought this book after joining Curves. It he...
1668956	1	Academically Repugnant	The first ten minutes of the film passed with ...

In [4]:
```
test.columns = ["polarity","title", "text"]
test = test.sample(50000,random_state = 99)
test.head()
```

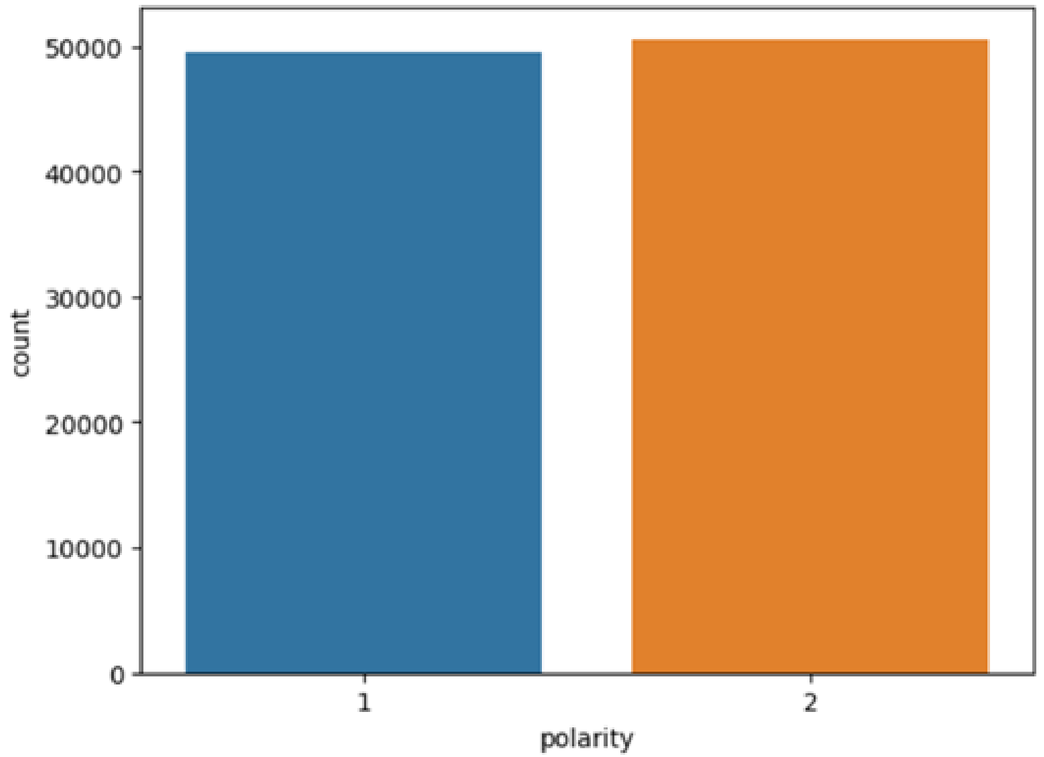
Out[4]:

	polarity	title	text
110227	1	A Useless Tool for Novel Writing	I met the author of this book/software at a wr...
118016	1	A load of historical BS	I bought this book several years ago and it wa...
180926	1	Don't do it	I have had two of these units and both worked ...
121412	1	Cute, but...	Really liked the Mobiblue 1500 cube at first -...
128565	1	Totally disappointed	This trilogy was recommended to me and I was s...

Count plot between negative (2) and positive (1) reviews in train and test data:

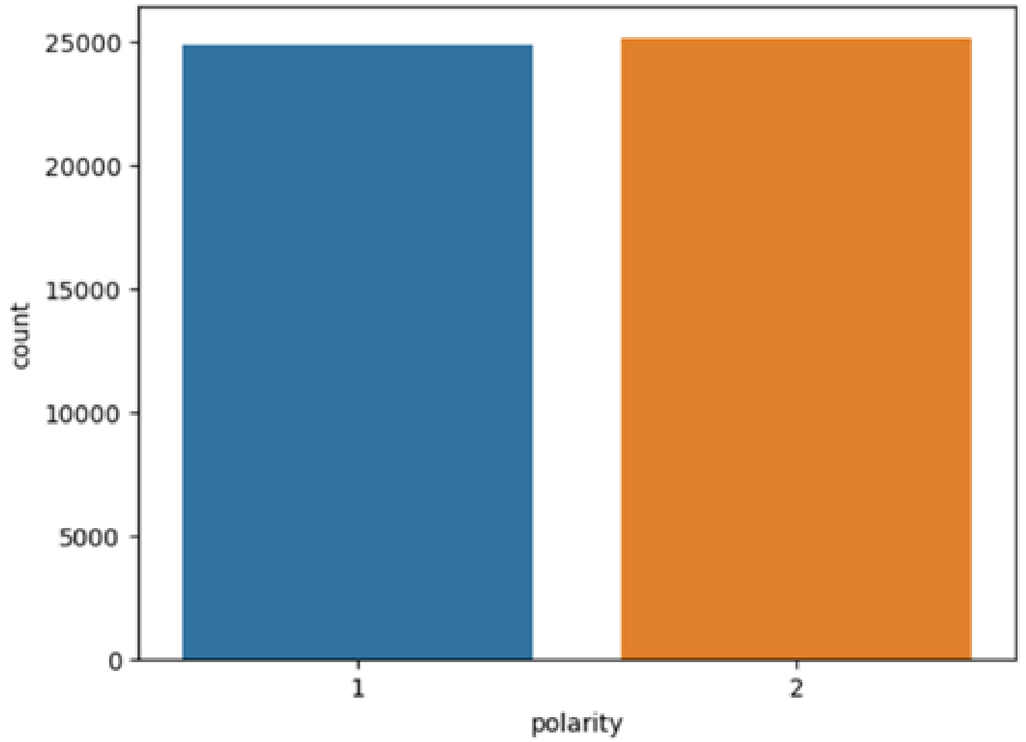
In [9]:
```
sns.countplot(x ='polarity', data = train)
```

Out[9]:
```
<Axes: xlabel='polarity', ylabel='count'>
```



In [10]:
```
sns.countplot(x ='polarity', data = test)
```

Out[10]:
```
<Axes: xlabel='polarity', ylabel='count'>
```



Text preprocessing

```
pr=PorterStemmer()
def clean(text):
    text=text.lower()
    without_punctuation ="".join(i for i in text if i not in string.punctuation)
    Without_punctuation= ''.join(c for c in without_punctuation if not ud.category(c).startswith('P'))
    nums = re.compile(r'\d+')
    clean_text = nums.sub(r'', Without_punctuation)
    clean_text =  ' '.join([word for word in  clean_text.split() if word not in stop_words])
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"
                               u"\U0001F300-\U0001F5FF"
                               u"\U0001F680-\U0001F6FF"
                               u"\U0001F1E0-\U0001F1FF"
                               u"\U00002500-\U00002BEF"
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
                               u"\u200d"
                               u"\u23cf"
                               u"\u23e9"
                               u"\u231a"
                               u"\ufe0f"
                               u"\u3030"
                               "]+", flags=re.UNICODE)
    clean = emoji_pattern.sub(r'', clean_text)
    words = word_tokenize(clean)
    stemmed_words=[pr.stem(word) for word in words]
    return TreebankWordDetokenizer().detokenize(stemmed_words)
```

Splitting data and applying vectorization using TFIDF:

In [18]:
```
x_train = train["text"]
y_train = train["polarity"]
x_test = test["text"]
y_test = test["polarity"]
```

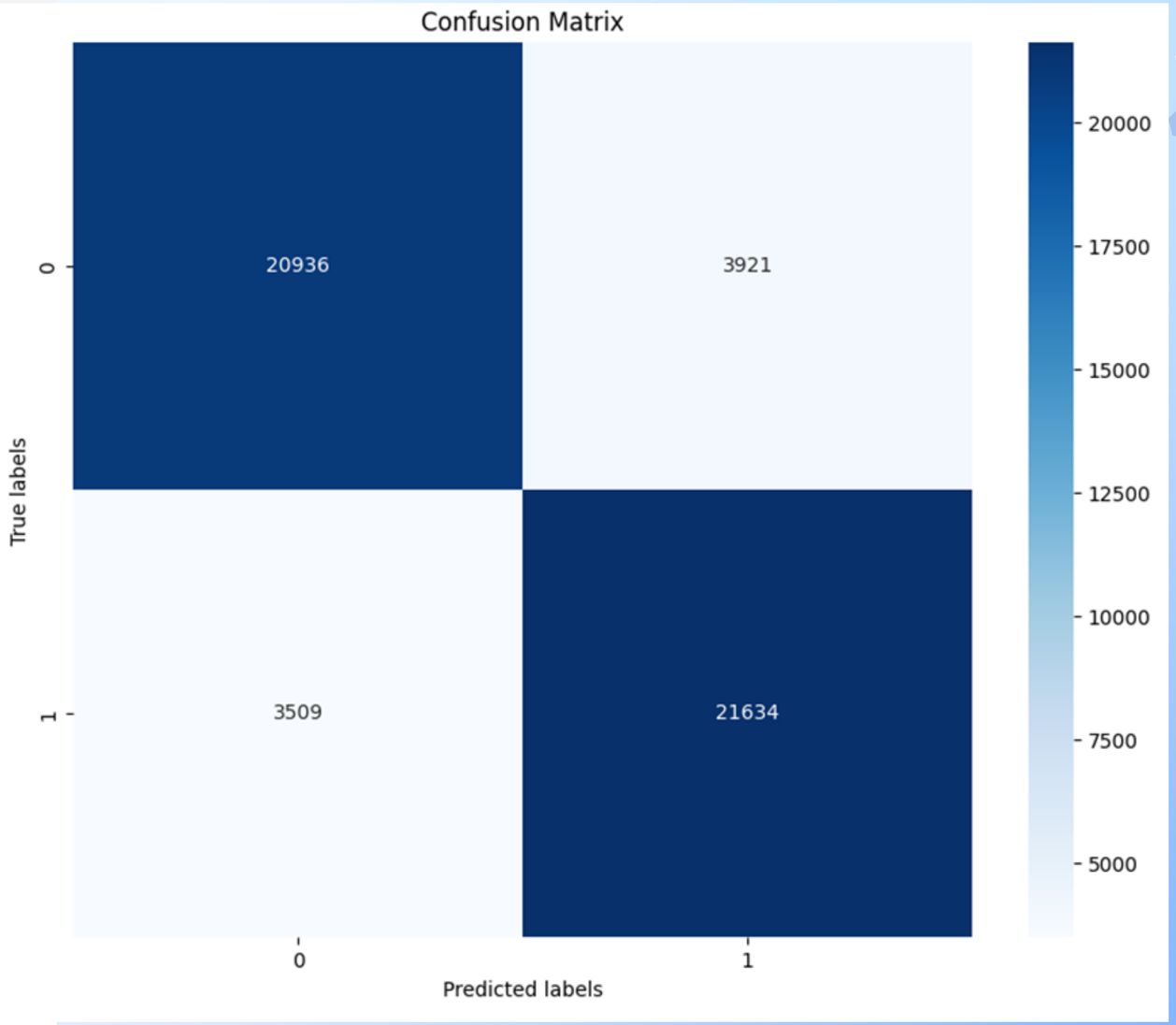
In [19]:
```
vectorizer = TfidfVectorizer().fit(x_train)
x_train_tfidf = vectorizer.transform(x_train)
x_test_tfidf = vectorizer.transform(x_test)
```

Machine learning section:
1- logistic model with TFIDF

In [20]:
```
model = LogisticRegression()
model.fit(x_train_tfidf,y_train)
y_train_predict = model.predict(x_train_tfidf)
y_test_predict = model.predict(x_test_tfidf)
```

In [21]:
```
train_mse = mean_squared_error(y_train, y_train_predict)
train_r2 = r2_score(y_train, y_train_predict)
train_accurcy=accuracy_score(y_train, y_train_predict)

test_mse = mean_squared_error(y_test, y_test_predict)
test_r2 = r2_score(y_test, y_test_predict)
test_accurcy=accuracy_score(y_test, y_test_predict)

print(f"Training MSE: {train_mse}")
print(f"Training R2 Score: {train_r2}")
print(f"training accurcy: {train_accurcy}")
print(f"Testing MSE: {test_mse}")
print(f"Testing R2 Score: {test_r2}")
print(f"testing accurcy: {test_accurcy}")
```

```
Training MSE: 0.11279
Training R2 Score: 0.5487854030337671
training accurcy: 0.88721
Testing MSE: 0.1486
Testing R2 Score: 0.4055805515467178
testing accurcy: 0.8514
```

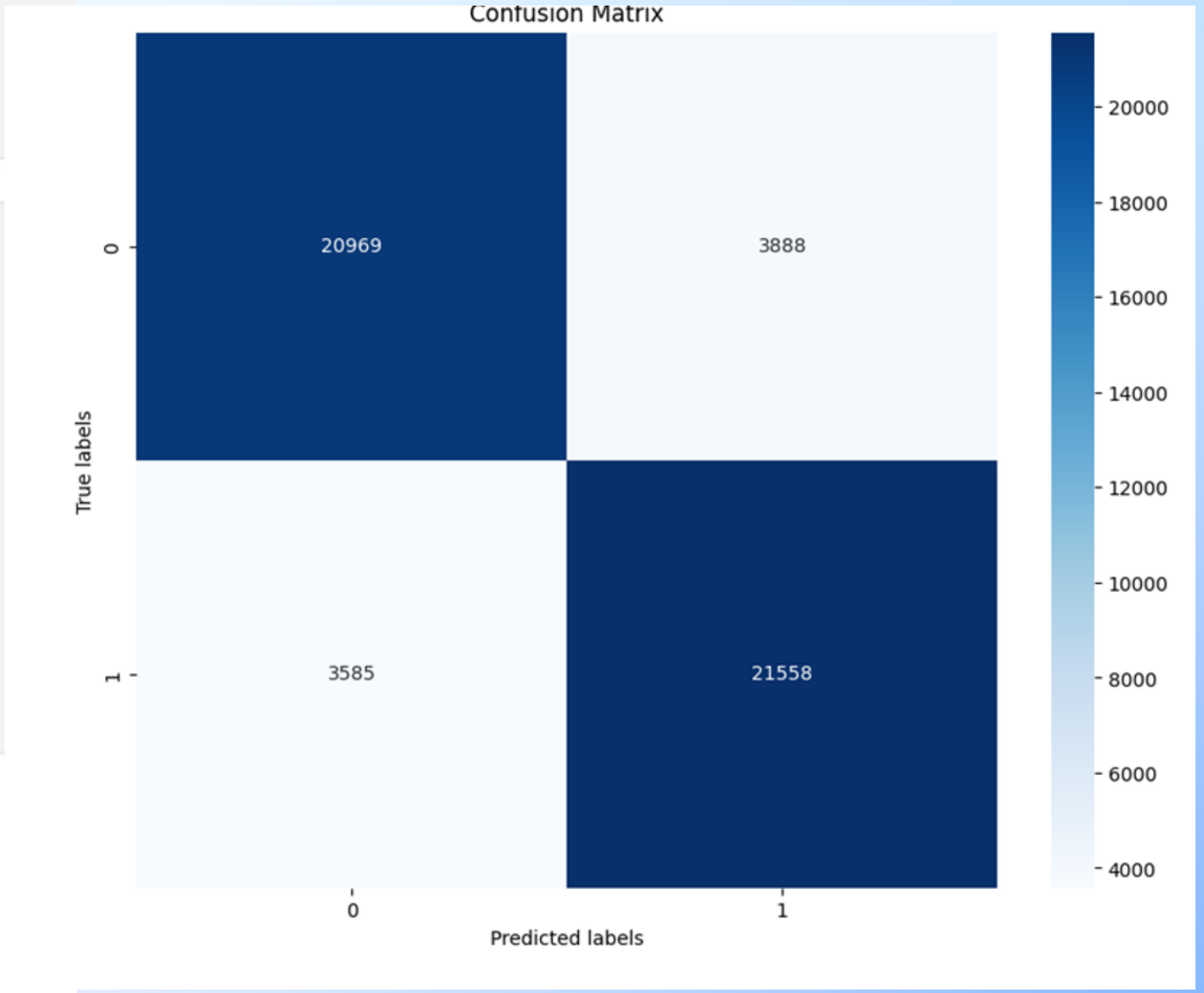


Machine learning section:
Logistic with N-Gram

```
model = LogisticRegression()
model.fit(X_train_ngrams, y_train)
y_train_predictlo = model.predict(X_train_ngrams)
y_test_predictlo = model.predict(X_test_ngrams)
```

```
train_mse1 = mean_squared_error(y_train, y_train_predictlo)
train_r21 = r2_score(y_train, y_train_predictlo)
train_accurcy=accuracy_score(y_train, y_train_predictlo)

test_mse1 = mean_squared_error(y_test, y_test_predictlo)
test_r21 = r2_score(y_test, y_test_predictlo)
test_accurcy=accuracy_score(y_test, y_test_predictlo)

print(f"Training MSE: {train_mse1}")
print(f"Training R2 Score: {train_r21}")
print(f"training accurcy: {train_accurcy}")
print(f"Testing MSE: {test_mse1}")
print(f"Testing R2 Score: {test_r21}")
print(f"testing accurcy: {test_accurcy}")
```

```
Training MSE: 0.00033
Training R2 Score: 0.9986798402606716
training accurcy: 0.99967
Testing MSE: 0.1306
Testing R2 Score: 0.4775829073485959
testing accurcy: 0.8694
```



Deep learning model

In [33]:

```
max_words = 2000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train['text'])

X_train = tokenizer.texts_to_sequences(train['text'])
X_train = pad_sequences(X_train, maxlen=max_len)
X_test = tokenizer.texts_to_sequences(test['text'])
X_test = pad_sequences(X_test, maxlen=max_len)

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=False)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

In [34]:

```
Lr_schedule = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1)
optimizer = tf.keras.optimizers.Adam(learning_rate=.0001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=64, callbacks=[Lr_schedule])
```

Deep learning model

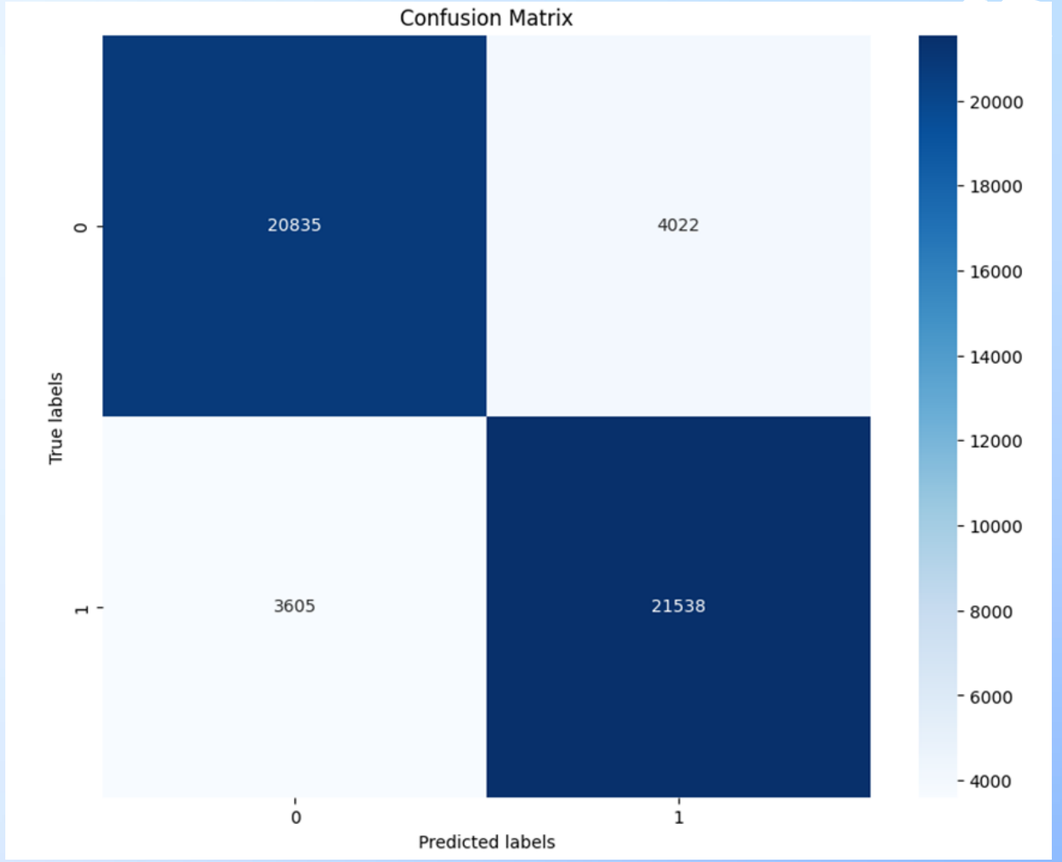
```
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8394 - loss: 0.3657 - val_accuracy: 0.8377 - val_loss: 0.3616 - learn
ing_rate: 1.0000e-04
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8472 - loss: 0.3476 - val_accuracy: 0.8295 - val_loss: 0.3807 - learn
ing_rate: 1.0000e-04
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8497 - loss: 0.3393 - val_accuracy: 0.8376 - val_loss: 0.3690 - learn
ing_rate: 1.0000e-04
Epoch 5/10
1561/1563 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.8587 - loss: 0.3266
Epoch 5: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-06.
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8587 - loss: 0.3266 - val_accuracy: 0.8400 - val_loss: 0.3616 - learn
ing_rate: 1.0000e-04
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 32ms/step - accuracy: 0.8663 - loss: 0.3099 - val_accuracy: 0.8475 - val_loss: 0.3510 - learn
ing_rate: 1.0000e-05
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8651 - loss: 0.3129 - val_accuracy: 0.8479 - val_loss: 0.3516 - learn
ing_rate: 1.0000e-05
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8661 - loss: 0.3075 - val_accuracy: 0.8481 - val_loss: 0.3544 - learn
ing_rate: 1.0000e-05
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.8675 - loss: 0.3061
Epoch 9: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-07.
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8675 - loss: 0.3061 - val_accuracy: 0.8480 - val_loss: 0.3537 - learn
ing_rate: 1.0000e-05
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.8683 - loss: 0.3038 - val_accuracy: 0.8475 - val_loss: 0.3538 - learn
ing_rate: 1.0000e-06
```

Model summary

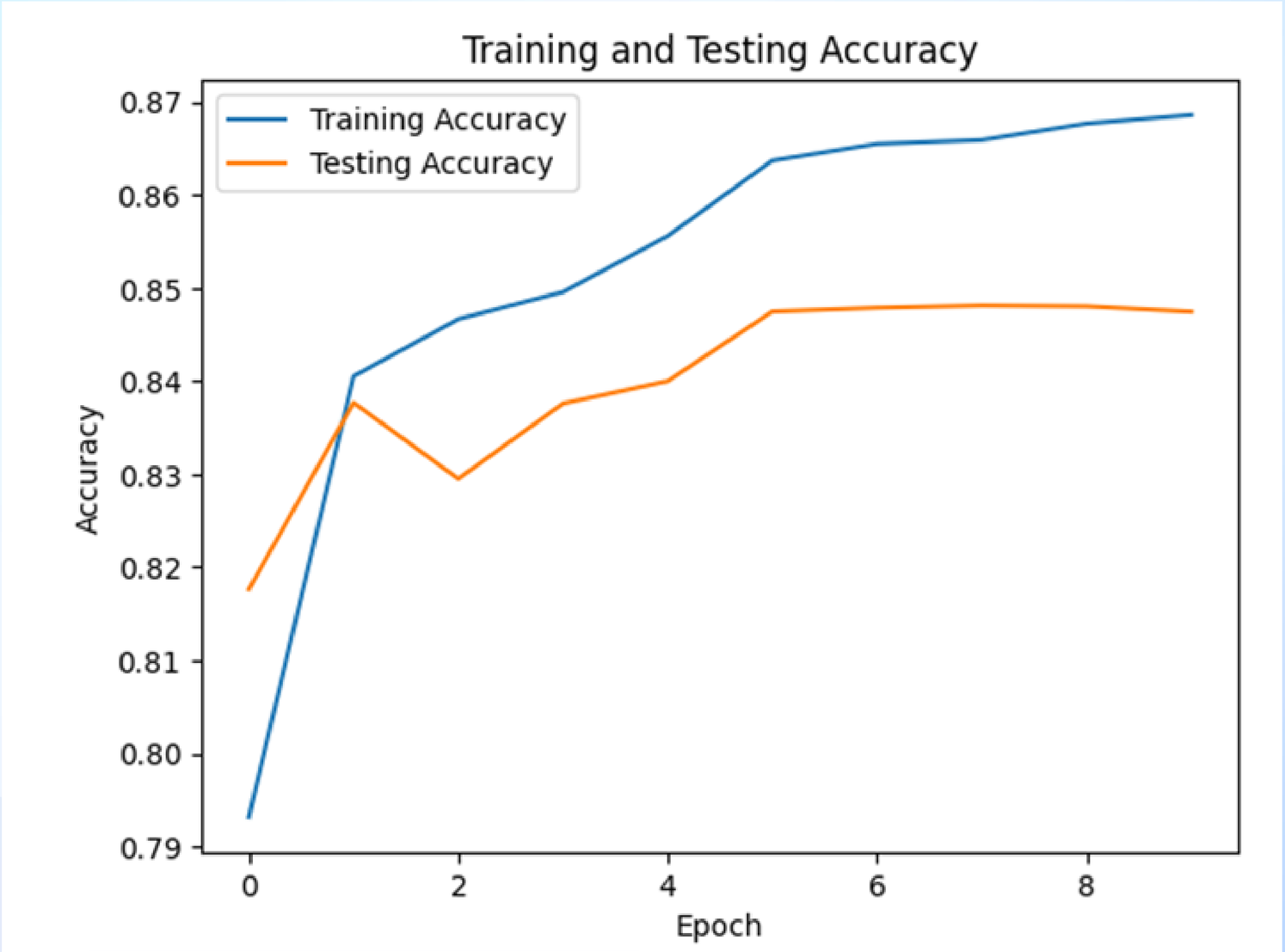
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 128)	256,000
bidirectional (Bidirectional)	(None, 100, 128)	98,816
bidirectional_1 (Bidirectional)	(None, 128)	98,816
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 32)	1,056
batch_normalization (BatchNormalization)	(None, 32)	128
dense_3 (Dense)	(None, 1)	33



The compression between training and testing accuracy:



Thank You