Module 9 Assignment 1: Natural Language Processing with Disaster Tweets

Dataset Overview and Model Training

The dataset for this project consists of 7,613 tweets in the training set and 3,263 tweets in the test set. The training set includes five columns: id, text, target, location, and keyword. The test set excludes the target column. Notably, there are 222 unique keywords and 3,342 unique locations in the training set, compared to 222 unique keywords and 1,603 unique locations in the test set. The memory usage for the training set is 0.29 MB, while the test set consumes 0.10 MB.

Model Training Environment

The models were trained on Google Colab, utilizing both CPU and GPU resources. Specifically, the GPU used was a Tesla T4 with 14.6 GB of memory, which significantly accelerated the training process.

Model Performance

Three models were trained using GLoVe embeddings with different dimensions (100D, 200D, and 300D), all implemented using LSTM layers. Here are the key results:

- **GLoVe-LSTM 100D**:
  - **Training Progress**: Achieved a training accuracy of 85.48% and a validation accuracy of 81.78% by the 10th epoch.
  - **Performance**: The model produced a confusion matrix indicating strong precision (0.79 for class 0, 0.85 for class 1) and recall, with an overall accuracy of 81%.
- **GLoVe-LSTM 200D**:
  - **Training Progress**: Reached a training accuracy of 87.37% and a validation accuracy of 80.93% by the 10th epoch.
  - **Performance**: The model showed similar performance to the 100D model, with an overall accuracy of 81% but slightly different recall and precision values, indicating some variation in the model's classification strength.
- **GLoVe-LSTM 300D**:

- **Training Progress**: Achieved a training accuracy of 89.57% and a validation accuracy of 79.94% by the 10th epoch.
- **Performance**: This model maintained a high level of precision but showed slightly lower recall for class 1 compared to the other models, resulting in an overall accuracy of 80%.

Conclusion

The three models trained using GLoVe embeddings of varying dimensions (100D, 200D, and 300D) all demonstrated strong performance in classifying disaster-related tweets. Each model showed improvement over the training epochs, with the following key observations:

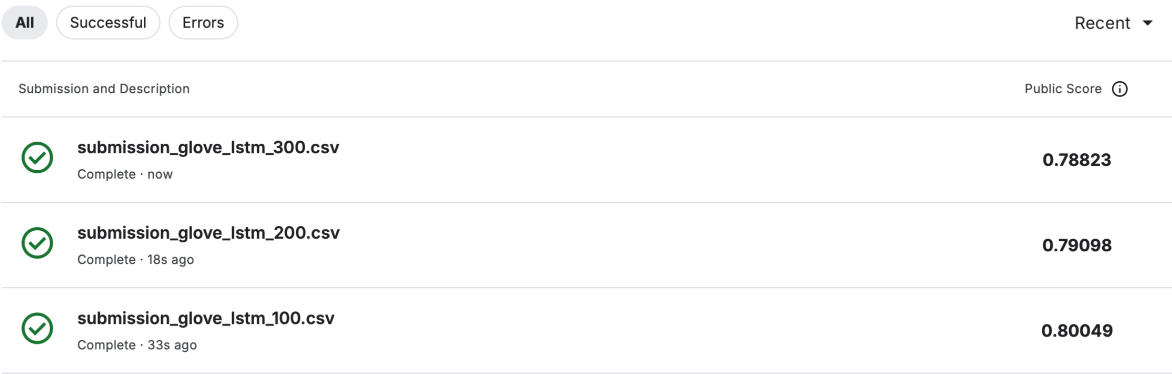
1. **Performance Consistency**:
   - The GLoVe-LSTM models with 100D and 200D embeddings both achieved similar overall accuracy (~81%) and demonstrated balanced precision and recall. This indicates that these models are reliable in correctly identifying disaster-related tweets and minimizing false positives and negatives.
2. **Impact of Embedding Dimensions**:
   - Interestingly, increasing the dimensionality of the GLoVe embeddings to 300D did not result in a significant performance gain. In fact, the 300D model exhibited slightly lower validation accuracy (79.94%) and higher validation loss in later epochs compared to the 100D and 200D models. This suggests that the added complexity from higher-dimensional embeddings might have led to overfitting or diminished returns in this particular task.
3. **Model Stability**:
   - All three models showed consistent learning behavior, with validation loss generally decreasing across epochs, particularly in the early stages of training. The 100D and 200D models reached a stable state sooner than the 300D model, indicating that these embeddings might be more suitable for this dataset in terms of achieving optimal performance without unnecessary complexity.
4. **Confusion Matrix Insights**:
   - The confusion matrices across all models revealed a strong performance in predicting non-disaster tweets (class 0), with precision and recall both hovering around 0.80 to 0.85. However, disaster tweets (class 1) were slightly

harder to classify, with recall dropping to the mid-0.70s. This suggests that while the models are effective, there is room for improvement, particularly in enhancing recall for disaster-related tweets to reduce the risk of missing critical information.

Kaggle

Username: sachinsharma03

Submission Screenshot:



All | Successful | Errors | Recent

Submission and Description	Public Score
submission_glove_lstm_300.csv Complete · now	0.78823
submission_glove_lstm_200.csv Complete · 18s ago	0.79098
submission_glove_lstm_100.csv Complete · 33s ago	0.80049

Appendix

Attached python code

Module 9 Assignment 1: Natural Language Processing with Disaster Tweets

The goal of this project is to develop a model that Predict which Tweets are about real disasters and which ones are not

Sachin Sharma

MSDS-422

08/14/2024

Management/Research Question

In layman's terms, what is the management/research question of interest, and why would anyone care?

Requirements

1. Conduct your analysis using a cross-validation design.
2. Conduct EDA.
3. Build at least three RNN models based on hyperparameter tuning.
4. Evaluate goodness of fit metrics.
5. Once you have your best-performing models, classify the test data and submit it to Kaggle. Provide your Kaggle.com user name and screen snapshots of your scores.
6. Discuss your model's performance.

Import Libraries

```
import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import STOPWORDS
import string
import warnings
warnings.filterwarnings("ignore")
from collections import  Counter
from sklearn.feature_extraction.text import CountVectorizer

import torch
from tensorflow.python.client import device_lib
from tensorflow.config import list_physical_devices
```

```
from torch import cuda
import random

import nltk
# nltk.download('stopwords') one time activity
# nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re

import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.initializers import Constant
from tensorflow.keras.layers import (LSTM,
                          Embedding,
                          BatchNormalization,
                          Dense,
                          TimeDistributed,
                          Dropout,
                          Bidirectional,
                          Flatten,
                          GlobalMaxPool1D,
                          Input)
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import ModelCheckpoint,
ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Model
from tensorflow.keras import Input

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    accuracy_score,
    confusion_matrix
)
from sklearn.model_selection import train_test_split

import transformers
from tqdm.notebook import tqdm
from transformers import BertTokenizer
from transformers import TFBertModel
from sklearn.model_selection import KFold

!wget http://nlp.stanford.edu/data/glove.6B.zip
```

```
--2024-08-13 05:09:34--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80...
connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-08-13 05:09:35--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|
171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
[following]
--2024-08-13 05:09:35--
https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)...
171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|
171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.15MB/s    in
3m 21s

2024-08-13 05:12:57 (4.09 MB/s) - ‘glove.6B.zip’ saved
[862182613/862182613]
```

```
!unzip glove*.zip
```

```
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
ls
```

```
glove.6B.100d.txt  glove.6B.300d.txt  glove.6B.zip
sample_data/
glove.6B.200d.txt  glove.6B.50d.txt   nlp-getting-started/
```

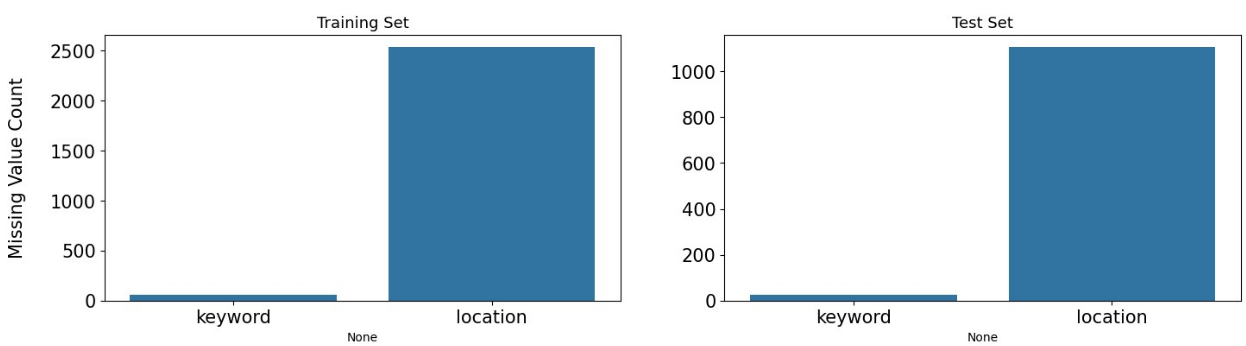
EDA

```
df_train = pd.read_csv('nlp-getting-started/train.csv')
df_test = pd.read_csv('nlp-getting-started/test.csv')

print('Training Set Shape = {}'.format(df_train.shape))
print('Training Set Memory Usage = {:.2f}
MB'.format(df_train.memory_usage().sum() / 1024**2))
```

```
print('Test Set Shape = {}'.format(df_test.shape))
print('Test Set Memory Usage = {:.2f}
MB'.format(df_test.memory_usage().sum() / 1024**2))
train_df = df_train.copy()
test_df = df_test.copy()
```

```
Training Set Shape = (7613, 5)
Training Set Memory Usage = 0.29 MB
Test Set Shape = (3263, 4)
Test Set Memory Usage = 0.10 MB
```

```
missing_cols = ['keyword', 'location']

fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)

sns.barplot(x=df_train[missing_cols].isnull().sum().index,
y=df_train[missing_cols].isnull().sum().values, ax=axes[0])
sns.barplot(x=df_test[missing_cols].isnull().sum().index,
y=df_test[missing_cols].isnull().sum().values, ax=axes[1])

axes[0].set_ylabel('Missing Value Count', size=15, labelpad=20)
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title('Training Set', fontsize=13)
axes[1].set_title('Test Set', fontsize=13)

plt.show()

for df in [df_train, df_test]:
    for col in ['keyword', 'location']:
        df[col] = df[col].fillna(f'no_{col}')
```



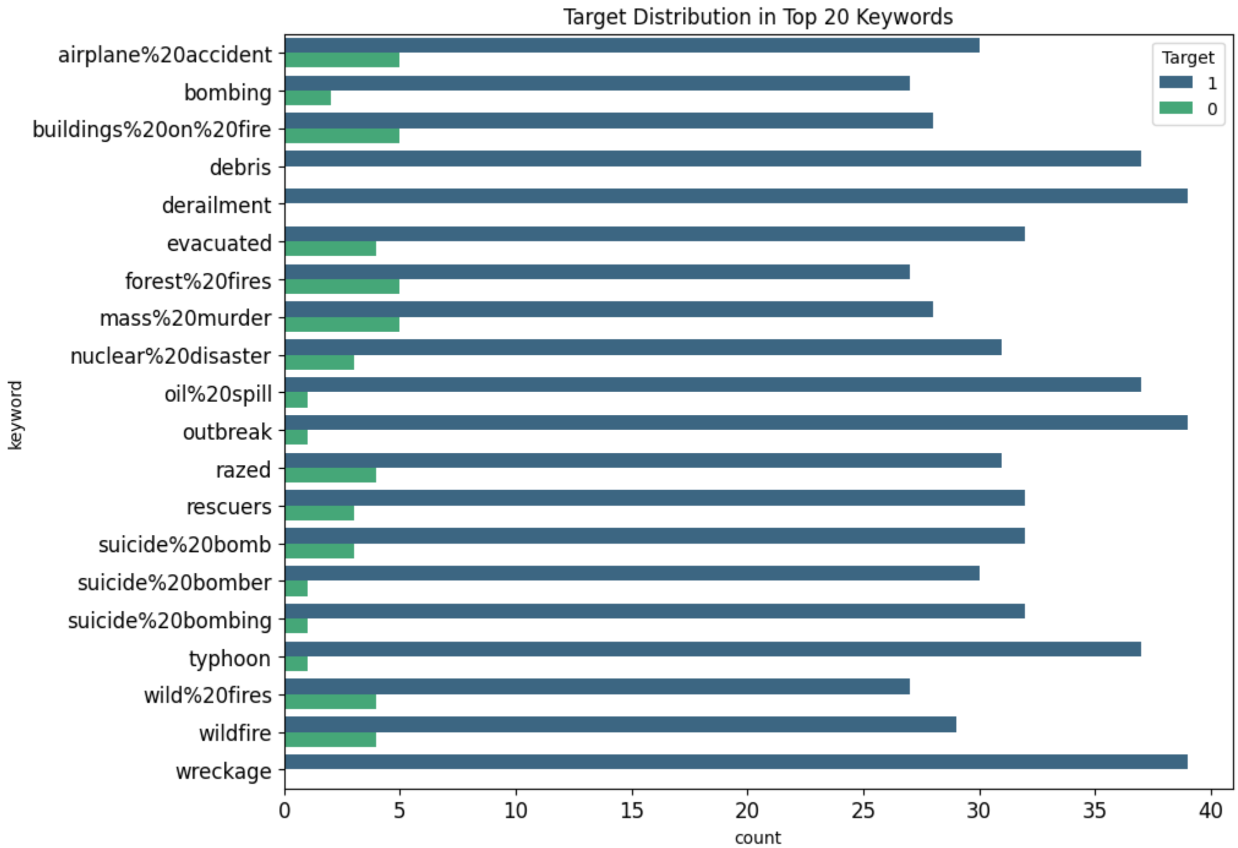
Missing Values

Both training and test set have same ratio of missing values in keyword and location

```
print(f'Number of unique values in keyword =
{df_train["keyword"].nunique()} (Training) -
{df_test["keyword"].nunique()} (Test)')
print(f'Number of unique values in location =
{df_train["location"].nunique()} (Training) -
{df_test["location"].nunique()} (Test)')
```

```
Number of unique values in keyword = 222 (Training) - 222 (Test)
Number of unique values in location = 3342 (Training) - 1603 (Test)
```

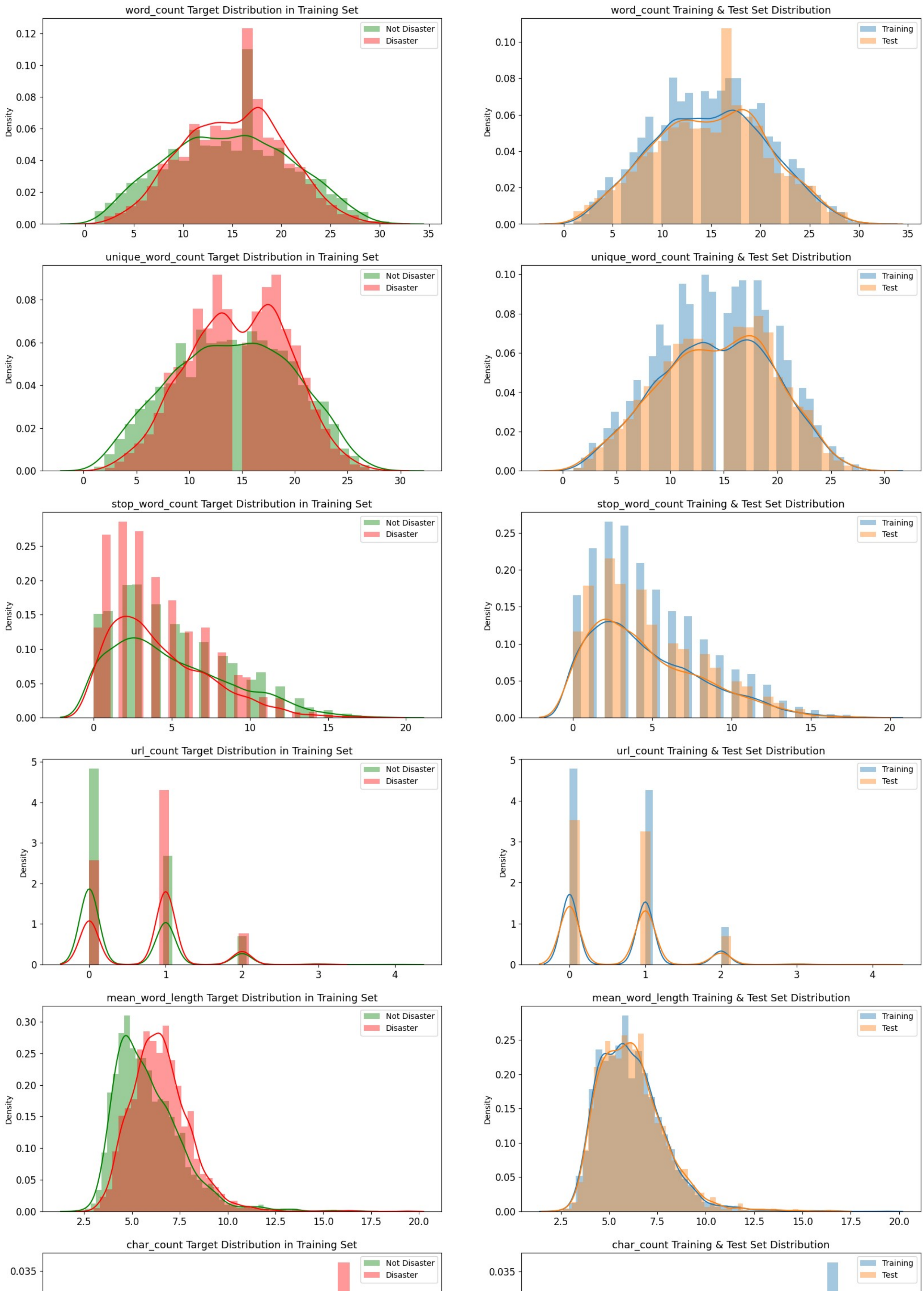
```
# Temporary conversion of target to string for plotting
df_train['target_str'] = df_train['target'].astype(str)

# Calculate target mean
df_train['target_mean'] = df_train.groupby('keyword')
['target'].transform('mean')

# Sort by target mean and select top 20 keywords
top_keywords = df_train[['keyword',
'target_mean']].drop_duplicates().sort_values(by='target_mean',
ascending=False).head(20)

# Filter the DataFrame to include only top keywords
df_top_keywords =
df_train[df_train['keyword'].isin(top_keywords['keyword'])]

# Plot
plt.figure(figsize=(10, 8), dpi=100)
sns.countplot(y=df_top_keywords['keyword'],
hue=df_top_keywords['target_str'], palette="viridis")

plt.tick_params(axis='x', labelsize=12)
plt.tick_params(axis='y', labelsize=12)
plt.legend(loc=1, title="Target")
plt.title('Target Distribution in Top 20 Keywords')

plt.show()

# # Clean up
df_train.drop(columns=['target_mean'], inplace=True)
```



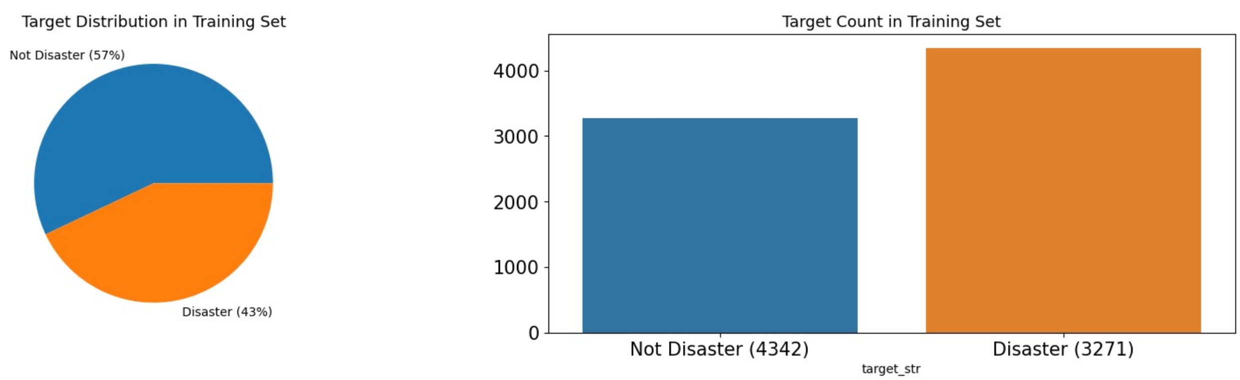

```python
# word_count
df_train['word_count'] = df_train['text'].apply(lambda x:
len(str(x).split()))
df_test['word_count'] = df_test['text'].apply(lambda x:
len(str(x).split()))

# unique_word_count
df_train['unique_word_count'] = df_train['text'].apply(lambda x:
len(set(str(x).split())))
df_test['unique_word_count'] = df_test['text'].apply(lambda x:
len(set(str(x).split())))

# stop_word_count
df_train['stop_word_count'] = df_train['text'].apply(lambda x: len([w
for w in str(x).lower().split() if w in STOPWORDS]))
df_test['stop_word_count'] = df_test['text'].apply(lambda x: len([w
for w in str(x).lower().split() if w in STOPWORDS]))

# url_count
df_train['url_count'] = df_train['text'].apply(lambda x: len([w for w
in str(x).lower().split() if 'http' in w or 'https' in w]))
df_test['url_count'] = df_test['text'].apply(lambda x: len([w for w in
str(x).lower().split() if 'http' in w or 'https' in w]))
```

```
# mean_word_length
df_train['mean_word_length'] = df_train['text'].apply(lambda x:
np.mean([len(w) for w in str(x).split()]))
df_test['mean_word_length'] = df_test['text'].apply(lambda x:
np.mean([len(w) for w in str(x).split()]))

# char_count
df_train['char_count'] = df_train['text'].apply(lambda x: len(str(x)))
df_test['char_count'] = df_test['text'].apply(lambda x: len(str(x)))

# punctuation_count
df_train['punctuation_count'] = df_train['text'].apply(lambda x:
len([c for c in str(x) if c in string.punctuation]))
df_test['punctuation_count'] = df_test['text'].apply(lambda x: len([c
for c in str(x) if c in string.punctuation]))

# hashtag_count
df_train['hashtag_count'] = df_train['text'].apply(lambda x: len([c
for c in str(x) if c == '#']))
df_test['hashtag_count'] = df_test['text'].apply(lambda x: len([c for
c in str(x) if c == '#']))

# mention_count
df_train['mention_count'] = df_train['text'].apply(lambda x: len([c
for c in str(x) if c == '@']))
df_test['mention_count'] = df_test['text'].apply(lambda x: len([c for
c in str(x) if c == '@']))

METAFEATURES = ['word_count', 'unique_word_count', 'stop_word_count',
'url_count', 'mean_word_length',
                'char_count', 'punctuation_count', 'hashtag_count',
'mention_count']
DISASTER_TWEETS = df_train['target'] == 1

fig, axes = plt.subplots(ncols=2, nrows=len(METAFEATURES),
figsize=(20, 50), dpi=100)

for i, feature in enumerate(METAFEATURES):
    sns.distplot(df_train.loc[~DISASTER_TWEETS][feature], label='Not
Disaster', ax=axes[i][0], color='green')
    sns.distplot(df_train.loc[DISASTER_TWEETS][feature],
label='Disaster', ax=axes[i][0], color='red')

    sns.distplot(df_train[feature], label='Training', ax=axes[i][1])
    sns.distplot(df_test[feature], label='Test', ax=axes[i][1])

    for j in range(2):
        axes[i][j].set_xlabel('')
        axes[i][j].tick_params(axis='x', labelsize=12)
        axes[i][j].tick_params(axis='y', labelsize=12)
```

```
        axes[i][j].legend()

    axes[i][0].set_title(f'{feature} Target Distribution in Training 
Set', fontsize=13)
    axes[i][1].set_title(f'{feature} Training & Test Set 
Distribution', fontsize=13)

plt.show()
```

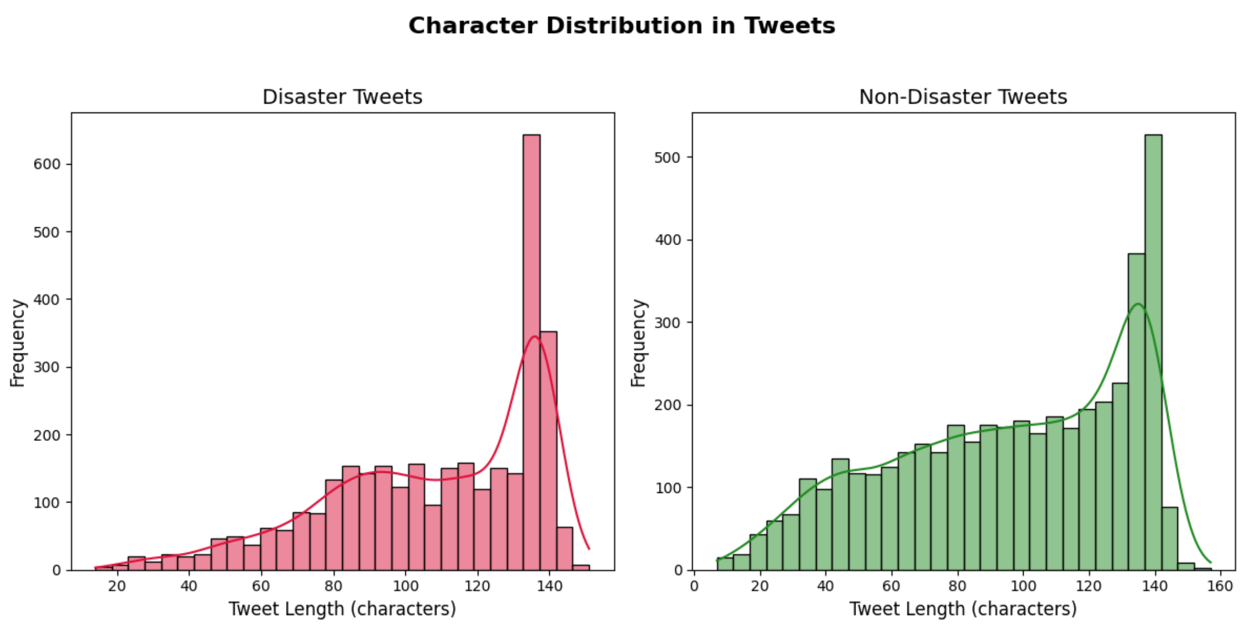

word_count Target Distribution in Training Set
word_count Training & Test Set Distribution
unique_word_count Target Distribution in Training Set
unique_word_count Training & Test Set Distribution
stop_word_count Target Distribution in Training Set
stop_word_count Training & Test Set Distribution
url_count Target Distribution in Training Set
url_count Training & Test Set Distribution
mean_word_length Target Distribution in Training Set
mean_word_length Training & Test Set Distribution
char_count Target Distribution in Training Set
char_count Training & Test Set Distribution
Not Disaster
Disaster
Training
Test
Density

```
fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)
plt.tight_layout()

df_train.groupby('target_str').count()['id'].plot(kind='pie',
ax=axes[0], labels=['Not Disaster (57%)', 'Disaster (43%)'])
sns.countplot(x=df_train['target_str'], hue=df_train['target_str'],
ax=axes[1])

axes[0].set_ylabel('')
axes[1].set_ylabel('')
axes[1].set_xticklabels(['Not Disaster (4342)', 'Disaster (3271)'])
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title('Target Distribution in Training Set', fontsize=13)
axes[1].set_title('Target Count in Training Set', fontsize=13)

plt.show()
```



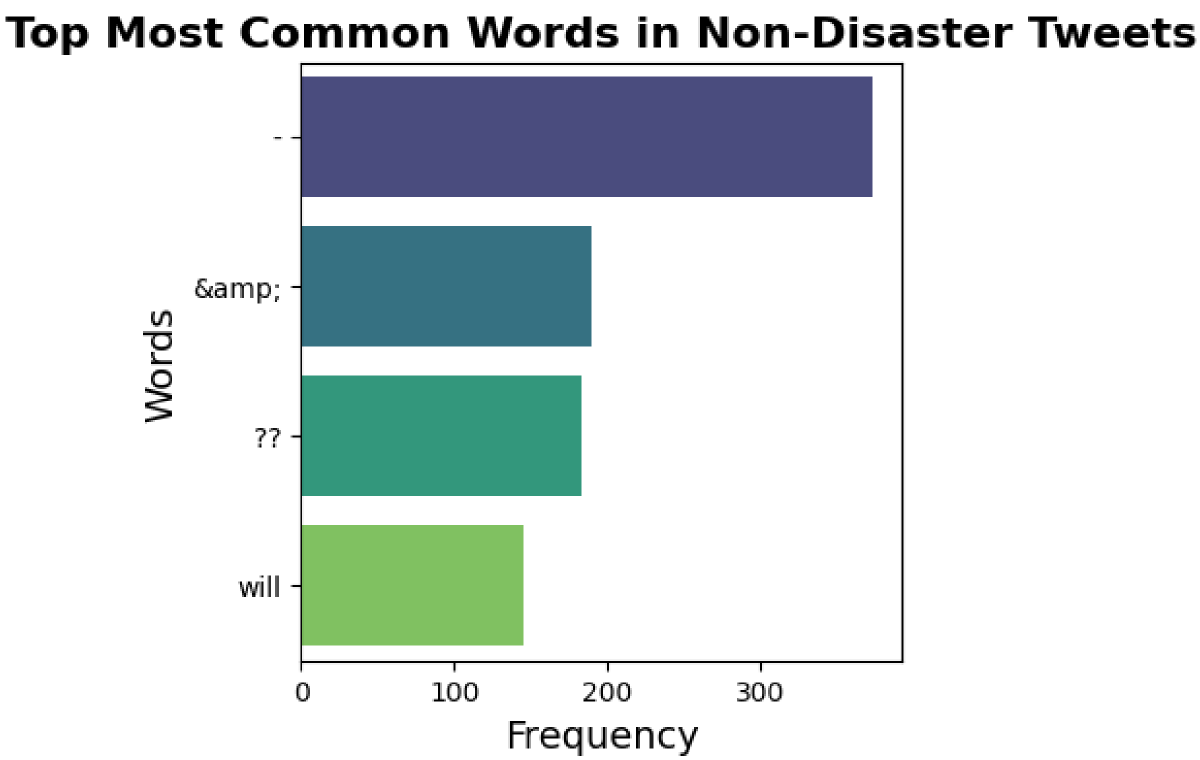

```
# Create a figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot the length of disaster tweets
tweet_len_disaster = df_train[df_train['target'] == 1]
['text'].str.len()
sns.histplot(tweet_len_disaster, bins=30, kde=True, color='crimson',
ax=ax1)
ax1.set_title('Disaster Tweets', fontsize=14)
ax1.set_xlabel('Tweet Length (characters)', fontsize=12)
ax1.set_ylabel('Frequency', fontsize=12)

# Plot the length of non-disaster tweets
tweet_len_non_disaster = df_train[df_train['target'] == 0]
['text'].str.len()
sns.histplot(tweet_len_non_disaster, bins=30, kde=True,
```

```
color='forestgreen', ax=ax2)
ax2.set_title('Non-Disaster Tweets', fontsize=14)
ax2.set_xlabel('Tweet Length (characters)', fontsize=12)
ax2.set_ylabel('Frequency', fontsize=12)

# Set a common title for the figure
fig.suptitle('Character Distribution in Tweets', fontsize=16,
fontweight='bold')

# Adjust layout for better spacing
plt.tight_layout(rect=[0, 0, 1, 0.95])

# Show the plot
plt.show()
```



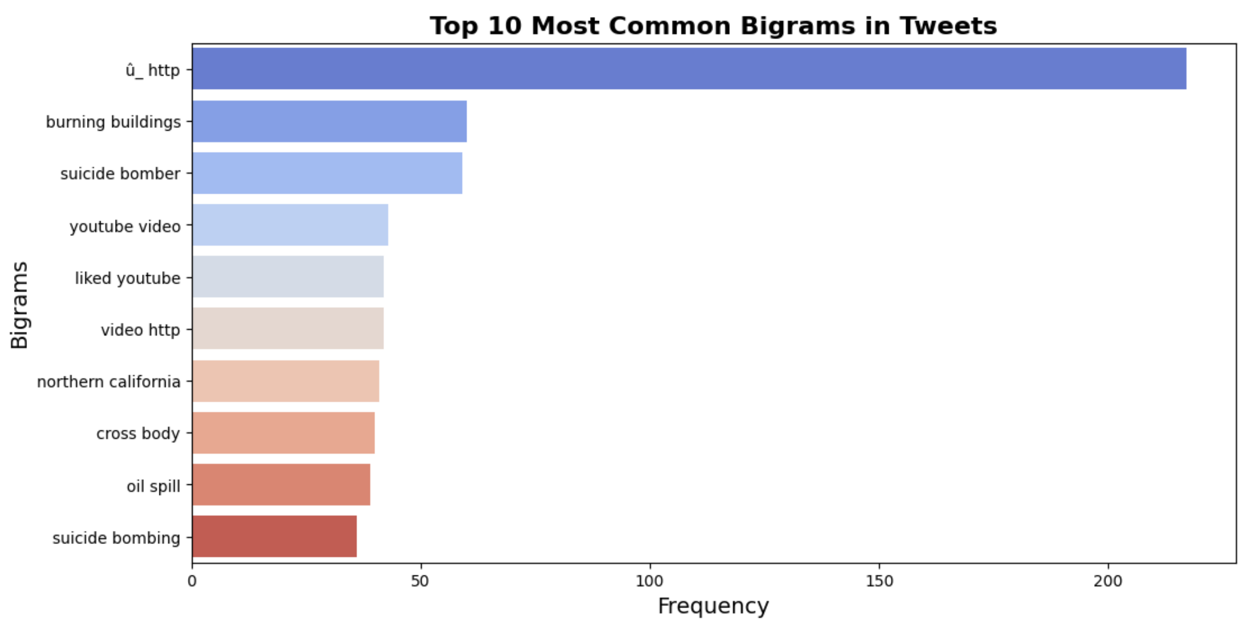

```
# Use STOPWORDS from the wordcloud library
stop = STOPWORDS

def create_corpus(target):
    corpus = []
    for x in df_train[df_train['target_str'] == target]
['text'].str.split():
        for i in x:
            corpus.append(i)
    return corpus

# Create corpus for non-disaster tweets

corpus = create_corpus('0')  # Using string '0' because of earlier
conversion
```

```
counter = Counter(corpus)
most = counter.most_common()

# Prepare data for plotting
x = []
y = []
for word, count in most[:50]:  # Top 50 words
    if word.lower() not in stop:
        x.append(word)
        y.append(count)

# Plot the data
plt.figure(figsize=(4, 4))
sns.barplot(x=y, y=x, palette="viridis", ci=None)
plt.title('Top Most Common Words in Non-Disaster Tweets', fontsize=16,
fontweight='bold')
plt.xlabel('Frequency', fontsize=14)
plt.ylabel('Words', fontsize=14)
plt.show()
```




```
def get_top_tweet_bigrams(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2),
stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
```

```python
    words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:n]

# Extract the top 10 bigrams
top_tweet_bigrams = get_top_tweet_bigrams(df_train['text'], n=10)

# Separate bigrams and their frequencies
x, y = zip(*top_tweet_bigrams)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x=list(y), y=list(x), palette="coolwarm")
plt.title('Top 10 Most Common Bigrams in Tweets', fontsize=16,
fontweight='bold')
plt.xlabel('Frequency', fontsize=14)
plt.ylabel('Bigrams', fontsize=14)
plt.show()

# Clean up
df_train.drop(columns=['target_str'], inplace=True)
```



Model Building

Data Prep

```python
list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```python
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 17806966287802526317
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14626652160
locality {
  bus_id: 1
  links {
  }
}
incarnation: 9881401946061084862
physical_device_desc: "device: 0, name: Tesla T4, pci bus id:
0000:00:04.0, compute capability: 7.5"
xla_global_id: 416903419
]
```

```python
train_df = train_df.dropna(how="any", axis=1)
```

```python
def remove_url(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'', text)


def remove_emoji(text):
    emoji_pattern = re.compile(
        '['
        u'\U0001F600-\U0001F64F'  # emoticons
        u'\U0001F300-\U0001F5FF'  # symbols & pictographs
        u'\U0001F680-\U0001F6FF'  # transport & map symbols
        u'\U0001F1E0-\U0001F1FF'  # flags (iOS)
        u'\U00002702-\U000027B0'
        u'\U000024C2-\U0001F251'
        ']+',
        flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)


def remove_html(text):
    html = re.compile(r'<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]
{1,6});')
    return re.sub(html, '', text)

# https://www.kaggle.com/tanulsingh077
```

```
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove
links,remove punctuation
    and remove words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub(
        'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-
fA-F][0-9a-fA-F]))+',
        '',
        text
    )
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)

    text = remove_url(text)
    text = remove_emoji(text)
    text = remove_html(text)

    return text

def show_metrics(pred_tag, y_test):
    print("F1-score: ", f1_score(pred_tag, y_test))
    print("Precision: ", precision_score(pred_tag, y_test))
    print("Recall: ", recall_score(pred_tag, y_test))
    print("Acuracy: ", accuracy_score(pred_tag, y_test))
    print("-"*50)
    print(classification_report(pred_tag, y_test))

def embed(corpus):
    return word_tokenizer.texts_to_sequences(corpus)

def plot_learning_curves(history, arr):
    fig, ax = plt.subplots(1, 2, figsize=(20, 5))
    for idx in range(2):
        ax[idx].plot(history.history[arr[idx][0]])
        ax[idx].plot(history.history[arr[idx][1]])
        ax[idx].legend([arr[idx][0], arr[idx][1]],fontsize=18)
        ax[idx].set_xlabel('A ',fontsize=16)
        ax[idx].set_ylabel('B',fontsize=16)
        ax[idx].set_title(arr[idx][0] + ' X ' + arr[idx]
[1],fontsize=16)

def preprocess_data(text):
    # Clean puntuation, urls, and so on
    text = clean_text(text)
    # Remove stopwords and Stemm all the words in the sentence
```

```
    text = ' '.join(stemmer.stem(word) for word in text.split(' ') if
word not in stop_words)

    return text

stop_words = stopwords.words('english')
more_stopwords = ['u', 'im', 'c']
stop_words = stop_words + more_stopwords

stemmer = nltk.SnowballStemmer("english")

test_df['text_clean'] = test_df['text'].apply(preprocess_data)

train_df['text_clean'] = train_df['text'].apply(preprocess_data)
train_df.head()
```


```
{"summary":"{\n  \"name\": \"train_df\",\n  \"rows\": 7613,\n
\"fields\": [\n    {\n      \"column\": \"id\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 3137,\n
\"min\": 1,\n        \"max\": 10873,\n        \"num_unique_values\":
7613,\n        \"samples\": [\n          3796,\n          3185,\n
7769\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"text\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7503,\n        \"samples\": [\n
\"Three Homes Demolished in Unrecognized Arab Village - International
Middle East Media Center http://t.co/ik8m4Yi9T4\",\n          \"Reid
Lake fire prompts campground evacuation order
http://t.co/jBODKM6rBU\",\n          \"FAAN orders evacuation of
abandoned aircraft at MMA http://t.co/dEvYbnVXGQ via @todayng\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"target\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"text_clean\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 6909,\n        \"samples\": [\n
\"transport panel show video pileup  snowstorm wy crash crash crash \\
u0089\\u00fb\\u00efmake stop\\u0089\\u00fb\\u009d amssumm\",\n
\"bomb appropri  seen famili jamaican love shout bullet \"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"train_df"}
```


```
train_tweets = train_df['text_clean'].values
test_tweets = test_df['text_clean'].values
train_target = train_df['target'].values
```

```python
# Calculate the length of our vocabulary
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(train_tweets)

vocab_length = len(word_tokenizer.word_index) + 1
vocab_length

longest_train = max(train_tweets, key=lambda sentence:
len(word_tokenize(sentence)))
length_long_sentence = len(word_tokenize(longest_train))

train_padded_sentences = pad_sequences(
    embed(train_tweets),
    length_long_sentence,
    padding='post'
)

test_padded_sentences = pad_sequences(
    embed(test_tweets),
    length_long_sentence,
    padding='post'
)

train_padded_sentences
```

```
array([[3635,  467,  201, ...,    0,    0,    0],
       [ 137,    2,  106, ...,    0,    0,    0],
       [1338,  502, 1806, ...,    0,    0,    0],
       ...,
       [ 448, 1328,    0, ...,    0,    0,    0],
       [  28,  162, 2636, ...,    0,    0,    0],
       [ 171,   31,  413, ...,    0,    0,    0]], dtype=int32)
```

```python
#KFOLD
# Parameters
k = 5  # Number of folds
kf = KFold(n_splits=k, shuffle=True, random_state=42)
```

Model 1: GLoVe-LSTM 100D

```python
embeddings_dictionary = dict()
embedding_dim = 100

# Load GloVe 100D embeddings
with open('glove.6B.100d.txt', encoding="utf8") as fp:
    for line in fp.readlines():
        records = line.split()
        word = records[0]
        vector_dimensions = np.asarray(records[1:], dtype='float32')
        embeddings_dictionary [word] = vector_dimensions
```

```python
embedding_matrix = np.zeros((vocab_length, embedding_dim))

for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

def glove_lstm_100():
    model = Sequential()

    model.add(Embedding(
        input_dim=embedding_matrix.shape[0],
        output_dim=embedding_matrix.shape[1],
        weights = [embedding_matrix],
        input_length=length_long_sentence
    ))

    model.add(Bidirectional(LSTM(
        length_long_sentence,
        return_sequences = True,
        recurrent_dropout=0.2
    )))

    model.add(GlobalMaxPool1D())
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation = 'sigmoid'))
    model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

    return model

model = glove_lstm_100()

model_glove_lstm_filename = 'model_glove_lstm_100.keras'
# Lists to store performance metrics for each fold
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

# K-Fold Cross Validation
for train_index, val_index in kf.split(train_padded_sentences):
    X_train_fold, X_val_fold = train_padded_sentences[train_index],
train_padded_sentences[val_index]
    y_train_fold, y_val_fold = train_target[train_index],
train_target[val_index]
```

```
    model_glove_lstm = glove_lstm_100()  # Instantiate a new model for each fold

    checkpoint = ModelCheckpoint(
        model_glove_lstm_filename,
        monitor='val_loss',
        verbose=1,
        save_best_only=True
    )

    reduce_lr = ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        verbose=1,
        patience=5,
        min_lr=0.001
    )

    history_glove_lstm = model_glove_lstm.fit(
        X_train_fold,
        y_train_fold,
        epochs=10,
        batch_size=32,
        validation_data=(X_val_fold, y_val_fold),
        verbose=1,
        callbacks=[reduce_lr, checkpoint]
    )

    # Load the best model from this fold
    model_final = load_model(model_glove_lstm_filename)
    y_pred = (model_final.predict(X_val_fold) > 0.5).astype("int32")

    # Calculate metrics
    accuracy_scores.append(accuracy_score(y_val_fold, y_pred))
    precision_scores.append(precision_score(y_val_fold, y_pred))
    recall_scores.append(recall_score(y_val_fold, y_pred))
    f1_scores.append(f1_score(y_val_fold, y_pred))

    print(f"Fold completed with Accuracy: {accuracy_scores[-1]}, Precision: {precision_scores[-1]}, Recall: {recall_scores[-1]}, F1-Score: {f1_scores[-1]}")
    print('-'*50)

# Compute average performance across all folds
print(f"Average Accuracy: {np.mean(accuracy_scores)}")
print(f"Average Precision: {np.mean(precision_scores)}")
print(f"Average Recall: {np.mean(recall_scores)}")
print(f"Average F1-Score: {np.mean(f1_scores)}")
```

```
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.5678 - loss:
0.8934
Epoch 1: val_loss improved from inf to 0.65101, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 26s 119ms/step - accuracy: 0.5679 - loss:
0.8929 - val_accuracy: 0.6678 - val_loss: 0.6510 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.6453 - loss:
0.6565
Epoch 2: val_loss improved from 0.65101 to 0.53583, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 101ms/step - accuracy: 0.6454 - loss:
0.6563 - val_accuracy: 0.7787 - val_loss: 0.5358 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.7169 - loss:
0.5701
Epoch 3: val_loss improved from 0.53583 to 0.48140, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 104ms/step - accuracy: 0.7169 - loss:
0.5700 - val_accuracy: 0.7912 - val_loss: 0.4814 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.7446 - loss:
0.5293
Epoch 4: val_loss improved from 0.48140 to 0.45209, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 102ms/step - accuracy: 0.7447 - loss:
0.5292 - val_accuracy: 0.7984 - val_loss: 0.4521 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.7928 - loss:
0.4791
Epoch 5: val_loss did not improve from 0.45209
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 109ms/step - accuracy: 0.7928 - loss:
0.4791 - val_accuracy: 0.8017 - val_loss: 0.4575 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.8164 - loss:
0.4430
Epoch 6: val_loss improved from 0.45209 to 0.44816, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 115ms/step - accuracy: 0.8164 - loss:
0.4430 - val_accuracy: 0.8043 - val_loss: 0.4482 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.8263 - loss:
0.4201
```

```
Epoch 7: val_loss improved from 0.44816 to 0.44778, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 100ms/step - accuracy: 0.8263 - loss:
0.4200 - val_accuracy: 0.8043 - val_loss: 0.4478 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.8464 - loss:
0.3859
Epoch 8: val_loss did not improve from 0.44778
191/191 ━━━━━━━━━━━━━━━━━━━━ 19s 99ms/step - accuracy: 0.8464 - loss:
0.3859 - val_accuracy: 0.8050 - val_loss: 0.4763 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 0.8628 - loss:
0.3489
Epoch 9: val_loss did not improve from 0.44778
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 103ms/step - accuracy: 0.8628 - loss:
0.3489 - val_accuracy: 0.8050 - val_loss: 0.5048 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.8815 - loss:
0.3154
Epoch 10: val_loss did not improve from 0.44778
191/191 ━━━━━━━━━━━━━━━━━━━━ 23s 114ms/step - accuracy: 0.8815 - loss:
0.3154 - val_accuracy: 0.8037 - val_loss: 0.5581 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step
Fold completed with Accuracy: 0.804333552199606, Precision:
0.8292682926829268, Recall: 0.6810477657935285, F1-Score:
0.7478849407783419
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.5366 - loss:
0.8480
Epoch 1: val_loss improved from inf to 0.65226, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 26s 106ms/step - accuracy: 0.5367 - loss:
0.8476 - val_accuracy: 0.7072 - val_loss: 0.6523 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.6599 - loss:
0.6418
Epoch 2: val_loss improved from 0.65226 to 0.55154, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 113ms/step - accuracy: 0.6600 - loss:
0.6417 - val_accuracy: 0.7728 - val_loss: 0.5515 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.7383 - loss:
0.5439
```

```
Epoch 3: val_loss improved from 0.55154 to 0.47434, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 116ms/step - accuracy: 0.7384 - loss:
0.5439 - val_accuracy: 0.7886 - val_loss: 0.4743 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.7878 - loss:
0.4917
Epoch 4: val_loss improved from 0.47434 to 0.46572, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 98ms/step - accuracy: 0.7878 - loss:
0.4917 - val_accuracy: 0.7997 - val_loss: 0.4657 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.8129 - loss:
0.4522
Epoch 5: val_loss did not improve from 0.46572
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 104ms/step - accuracy: 0.8129 - loss:
0.4522 - val_accuracy: 0.8024 - val_loss: 0.4669 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.8365 - loss:
0.4291
Epoch 6: val_loss did not improve from 0.46572
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 103ms/step - accuracy: 0.8365 - loss:
0.4291 - val_accuracy: 0.8011 - val_loss: 0.4662 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.8471 - loss:
0.3784
Epoch 7: val_loss did not improve from 0.46572
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 112ms/step - accuracy: 0.8471 - loss:
0.3784 - val_accuracy: 0.8011 - val_loss: 0.4955 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8708 - loss:
0.3551
Epoch 8: val_loss did not improve from 0.46572
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 116ms/step - accuracy: 0.8708 - loss:
0.3552 - val_accuracy: 0.7965 - val_loss: 0.4866 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.8716 - loss:
0.3204
Epoch 9: val_loss did not improve from 0.46572
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 116ms/step - accuracy: 0.8716 - loss:
0.3205 - val_accuracy: 0.7899 - val_loss: 0.5182 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - accuracy: 0.8941 - loss:
```

```
0.2957
Epoch 10: val_loss did not improve from 0.46572
191/191 ━━━━━━━━━━━━━━━━━━━━ 19s 101ms/step - accuracy: 0.8941 - loss:
0.2957 - val_accuracy: 0.7919 - val_loss: 0.5919 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step
Fold completed with Accuracy: 0.7997373604727511, Precision:
0.8185185185185185, Recall: 0.6810477657935285, F1-Score:
0.7434819175777966
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.5338 - loss:
0.8643
Epoch 1: val_loss improved from inf to 0.65143, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 28s 121ms/step - accuracy: 0.5340 - loss:
0.8638 - val_accuracy: 0.7209 - val_loss: 0.6514 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.6542 - loss:
0.6431
Epoch 2: val_loss improved from 0.65143 to 0.53743, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 118ms/step - accuracy: 0.6543 - loss:
0.6430 - val_accuracy: 0.7978 - val_loss: 0.5374 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.7337 - loss:
0.5583
Epoch 3: val_loss improved from 0.53743 to 0.45378, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 112ms/step - accuracy: 0.7337 - loss:
0.5583 - val_accuracy: 0.8004 - val_loss: 0.4538 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - accuracy: 0.7645 - loss:
0.5135
Epoch 4: val_loss improved from 0.45378 to 0.42862, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 102ms/step - accuracy: 0.7645 - loss:
0.5134 - val_accuracy: 0.8109 - val_loss: 0.4286 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.8056 - loss:
0.4598
Epoch 5: val_loss improved from 0.42862 to 0.42059, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 105ms/step - accuracy: 0.8055 - loss:
0.4599 - val_accuracy: 0.8122 - val_loss: 0.4206 - learning_rate:
0.0010
```

```
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.8197 - loss:
0.4406
Epoch 6: val_loss did not improve from 0.42059
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 116ms/step - accuracy: 0.8197 - loss:
0.4406 - val_accuracy: 0.8142 - val_loss: 0.4304 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.8434 - loss:
0.3849
Epoch 7: val_loss did not improve from 0.42059
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 115ms/step - accuracy: 0.8434 - loss:
0.3849 - val_accuracy: 0.8214 - val_loss: 0.4251 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.8480 - loss:
0.3774
Epoch 8: val_loss did not improve from 0.42059
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 99ms/step - accuracy: 0.8480 - loss:
0.3774 - val_accuracy: 0.8089 - val_loss: 0.5023 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.8751 - loss:
0.3350
Epoch 9: val_loss did not improve from 0.42059
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 99ms/step - accuracy: 0.8751 - loss:
0.3351 - val_accuracy: 0.8122 - val_loss: 0.5088 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.8917 - loss:
0.2978
Epoch 10: val_loss did not improve from 0.42059
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 111ms/step - accuracy: 0.8916 - loss:
0.2979 - val_accuracy: 0.8129 - val_loss: 0.5159 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step
Fold completed with Accuracy: 0.8122127380170716, Precision:
0.8378378378378378, Recall: 0.7034795763993948, F1-Score:
0.7648026315789473
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.5360 - loss:
0.9072
Epoch 1: val_loss improved from inf to 0.63874, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 25s 101ms/step - accuracy: 0.5363 - loss:
0.9065 - val_accuracy: 0.6820 - val_loss: 0.6387 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step - accuracy: 0.6792 - loss:
```

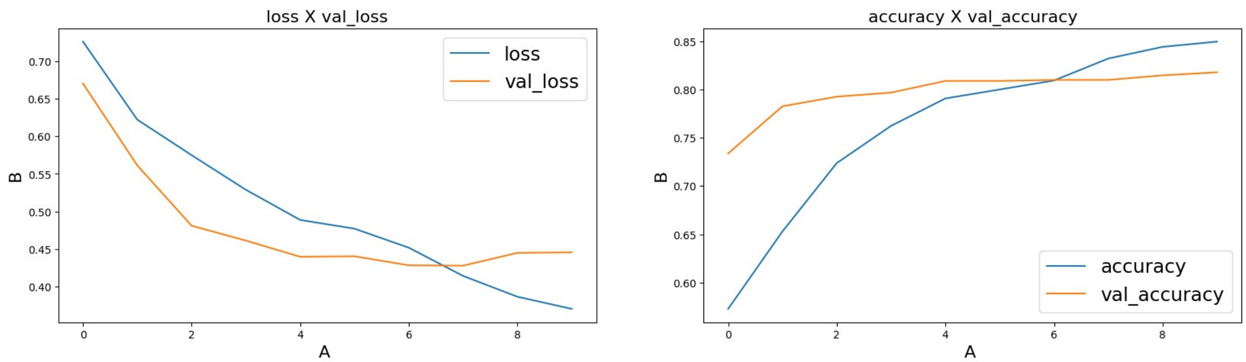
```
0.6274
Epoch 2: val_loss improved from 0.63874 to 0.52479, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 107ms/step - accuracy: 0.6793 - loss:
0.6273 - val_accuracy: 0.7700 - val_loss: 0.5248 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.7636 - loss:
0.5267
Epoch 3: val_loss improved from 0.52479 to 0.46695, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 116ms/step - accuracy: 0.7636 - loss:
0.5267 - val_accuracy: 0.7943 - val_loss: 0.4670 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.8033 - loss:
0.4684
Epoch 4: val_loss improved from 0.46695 to 0.46532, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 121ms/step - accuracy: 0.8032 - loss:
0.4684 - val_accuracy: 0.7845 - val_loss: 0.4653 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.8226 - loss:
0.4457
Epoch 5: val_loss improved from 0.46532 to 0.45173, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 103ms/step - accuracy: 0.8226 - loss:
0.4457 - val_accuracy: 0.8016 - val_loss: 0.4517 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.8416 - loss:
0.3959
Epoch 6: val_loss did not improve from 0.45173
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 100ms/step - accuracy: 0.8416 - loss:
0.3959 - val_accuracy: 0.7792 - val_loss: 0.4626 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step - accuracy: 0.8512 - loss:
0.3825
Epoch 7: val_loss did not improve from 0.45173
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 107ms/step - accuracy: 0.8512 - loss:
0.3825 - val_accuracy: 0.8055 - val_loss: 0.4766 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8653 - loss:
0.3320
Epoch 8: val_loss did not improve from 0.45173
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 117ms/step - accuracy: 0.8654 - loss:
0.3320 - val_accuracy: 0.7904 - val_loss: 0.5294 - learning_rate:
```

```
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step - accuracy: 0.8799 - loss:
0.3248
Epoch 9: val_loss did not improve from 0.45173
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 117ms/step - accuracy: 0.8798 - loss:
0.3249 - val_accuracy: 0.7884 - val_loss: 0.5682 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.8911 - loss:
0.2877
Epoch 10: val_loss did not improve from 0.45173
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 104ms/step - accuracy: 0.8911 - loss:
0.2878 - val_accuracy: 0.7937 - val_loss: 0.6013 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step
Fold completed with Accuracy: 0.8015768725361366, Precision:
0.8531746031746031, Recall: 0.6534954407294833, F1-Score:
0.7401032702237522
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.5376 - loss:
0.8238
Epoch 1: val_loss improved from inf to 0.65344, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 28s 124ms/step - accuracy: 0.5378 - loss:
0.8234 - val_accuracy: 0.7254 - val_loss: 0.6534 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.6437 - loss:
0.6355
Epoch 2: val_loss improved from 0.65344 to 0.54112, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 110ms/step - accuracy: 0.6438 - loss:
0.6353 - val_accuracy: 0.7523 - val_loss: 0.5411 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.7221 - loss:
0.5659
Epoch 3: val_loss improved from 0.54112 to 0.50064, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 100ms/step - accuracy: 0.7222 - loss:
0.5658 - val_accuracy: 0.7694 - val_loss: 0.5006 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.7801 - loss:
0.4945
Epoch 4: val_loss improved from 0.50064 to 0.48059, saving model to
model_glove_lstm_100.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 103ms/step - accuracy: 0.7801 - loss:
```

```
0.4944 - val_accuracy: 0.7871 - val_loss: 0.4806 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.8123 - loss:
0.4645
Epoch 5: val_loss did not improve from 0.48059
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 113ms/step - accuracy: 0.8123 - loss:
0.4645 - val_accuracy: 0.7943 - val_loss: 0.4914 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8405 - loss:
0.4071
Epoch 6: val_loss did not improve from 0.48059
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 116ms/step - accuracy: 0.8405 - loss:
0.4071 - val_accuracy: 0.7989 - val_loss: 0.4813 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.8559 - loss:
0.3764
Epoch 7: val_loss did not improve from 0.48059
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 105ms/step - accuracy: 0.8558 - loss:
0.3765 - val_accuracy: 0.8009 - val_loss: 0.5185 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.8623 - loss:
0.3503
Epoch 8: val_loss did not improve from 0.48059
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 108ms/step - accuracy: 0.8622 - loss:
0.3504 - val_accuracy: 0.8029 - val_loss: 0.5083 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step - accuracy: 0.8817 - loss:
0.3008
Epoch 9: val_loss did not improve from 0.48059
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 109ms/step - accuracy: 0.8817 - loss:
0.3009 - val_accuracy: 0.7963 - val_loss: 0.5628 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.8932 - loss:
0.3052
Epoch 10: val_loss did not improve from 0.48059
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 116ms/step - accuracy: 0.8932 - loss:
0.3052 - val_accuracy: 0.7865 - val_loss: 0.6243 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step
Fold completed with Accuracy: 0.7871222076215506, Precision: 0.8125,
Recall: 0.6559633027522935, F1-Score: 0.7258883248730964
--------------------------------------------------
Average Accuracy: 0.8009965461694231
Average Precision: 0.8302598504427774
```

```
Average Recall: 0.6750067702936458
Average F1-Score: 0.7444322170063868
```

```
plot_learning_curves(history_glove_lstm, [['loss', 'val_loss'],
['accuracy', 'val_accuracy']])
```




```
model_glove_lstm_final = load_model(model_glove_lstm_filename)
y_pred_glove_lstm = (model_glove_lstm_final.predict(X_val) >
0.5).astype("int32")

cm = confusion_matrix(y_val,y_pred_glove_lstm)
print('confusion matrix:\n',cm)
print(classification_report(y_val, y_pred_glove_lstm))
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step
confusion matrix:
 [[967 104]
 [258 575]]
              precision    recall  f1-score   support

           0       0.79      0.90      0.84      1071
           1       0.85      0.69      0.76       833

    accuracy                           0.81      1904
   macro avg       0.82      0.80      0.80      1904
weighted avg       0.81      0.81      0.81      1904
```

Submission

```
y_glove_lstm = (model_glove_lstm_final.predict(test_padded_sentences)
> 0.5).astype("int32")
y_labels = []
for i in range (0, len(y_glove_lstm)):
    y_labels.append(y_glove_lstm[i][0])

# create submission file
submission_glove_lstm = pd.DataFrame({"id": (test_df['id']),"target":
```

```
y_labels})
submission_glove_lstm.to_csv('submission_glove_lstm_100.csv',
index=False)
```

```
102/102 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step
```

Model 2: GLoVe-LSTM 200D

```
embeddings_dictionary = dict()
embedding_dim = 200

# Load GloVe 200D embeddings
with open('glove.6B.200d.txt', encoding="utf8") as fp:
    for line in fp.readlines():
        records = line.split()
        word = records[0]
        vector_dimensions = np.asarray(records[1:], dtype='float32')
        embeddings_dictionary [word] = vector_dimensions

embedding_matrix = np.zeros((vocab_length, embedding_dim))

for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

def glove_lstm_200():
    model = Sequential()

    model.add(Embedding(
        input_dim=embedding_matrix.shape[0],
        output_dim=embedding_matrix.shape[1],
        weights = [embedding_matrix],
        input_length=length_long_sentence
    ))

    model.add(Bidirectional(LSTM(
        length_long_sentence,
        return_sequences = True,
        recurrent_dropout=0.2
    )))

    model.add(GlobalMaxPool1D())
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation = 'sigmoid'))
    model.compile(optimizer='rmsprop', loss='binary_crossentropy',
```

```
metrics=['accuracy'])

    return model

model = glove_lstm_200()

model_glove_lstm_filename = 'model_glove_lstm_200.keras'
# Lists to store performance metrics for each fold
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

# K-Fold Cross Validation
for train_index, val_index in kf.split(train_padded_sentences):
    X_train_fold, X_val_fold = train_padded_sentences[train_index],
train_padded_sentences[val_index]
    y_train_fold, y_val_fold = train_target[train_index],
train_target[val_index]

    model_glove_lstm = glove_lstm_200()  # Instantiate a new model for
each fold

    checkpoint = ModelCheckpoint(
        model_glove_lstm_filename,
        monitor='val_loss',
        verbose=1,
        save_best_only=True
    )

    reduce_lr = ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        verbose=1,
        patience=5,
        min_lr=0.001
    )

    history_glove_lstm = model_glove_lstm.fit(
        X_train_fold,
        y_train_fold,
        epochs=10,
        batch_size=32,
        validation_data=(X_val_fold, y_val_fold),
        verbose=1,
        callbacks=[reduce_lr, checkpoint]
    )

    # Load the best model from this fold
    model_final = load_model(model_glove_lstm_filename)
```

```
    y_pred = (model_final.predict(X_val_fold) > 0.5).astype("int32")

    # Calculate metrics
    accuracy_scores.append(accuracy_score(y_val_fold, y_pred))
    precision_scores.append(precision_score(y_val_fold, y_pred))
    recall_scores.append(recall_score(y_val_fold, y_pred))
    f1_scores.append(f1_score(y_val_fold, y_pred))

    print(f"Fold completed with Accuracy: {accuracy_scores[-1]},
Precision: {precision_scores[-1]}, Recall: {recall_scores[-1]}, F1-
Score: {f1_scores[-1]}")
    print('-'*50)

# Compute average performance across all folds
print(f"Average Accuracy: {np.mean(accuracy_scores)}")
print(f"Average Precision: {np.mean(precision_scores)}")
print(f"Average Recall: {np.mean(recall_scores)}")
print(f"Average F1-Score: {np.mean(f1_scores)}")
```

```
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.5617 - loss:
0.8135
Epoch 1: val_loss improved from inf to 0.65057, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 27s 101ms/step - accuracy: 0.5618 - loss:
0.8132 - val_accuracy: 0.7472 - val_loss: 0.6506 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - accuracy: 0.6253 - loss:
0.6594
Epoch 2: val_loss improved from 0.65057 to 0.56502, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 19s 99ms/step - accuracy: 0.6255 - loss:
0.6592 - val_accuracy: 0.7800 - val_loss: 0.5650 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.7114 - loss:
0.5910
Epoch 3: val_loss improved from 0.56502 to 0.49223, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 112ms/step - accuracy: 0.7115 - loss:
0.5909 - val_accuracy: 0.7932 - val_loss: 0.4922 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.7727 - loss:
0.5143
Epoch 4: val_loss improved from 0.49223 to 0.46736, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 116ms/step - accuracy: 0.7727 - loss:
0.5143 - val_accuracy: 0.8024 - val_loss: 0.4674 - learning_rate:
```

```
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.7950 - loss:
0.4762
Epoch 5: val_loss improved from 0.46736 to 0.45678, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 115ms/step - accuracy: 0.7950 - loss:
0.4762 - val_accuracy: 0.8024 - val_loss: 0.4568 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.8056 - loss:
0.4635
Epoch 6: val_loss improved from 0.45678 to 0.45060, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 19s 101ms/step - accuracy: 0.8056 - loss:
0.4635 - val_accuracy: 0.8083 - val_loss: 0.4506 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.8293 - loss:
0.4244
Epoch 7: val_loss did not improve from 0.45060
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 97ms/step - accuracy: 0.8293 - loss:
0.4244 - val_accuracy: 0.8030 - val_loss: 0.4579 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.8393 - loss:
0.3946
Epoch 8: val_loss did not improve from 0.45060
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 100ms/step - accuracy: 0.8393 - loss:
0.3947 - val_accuracy: 0.8089 - val_loss: 0.4682 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.8525 - loss:
0.3707
Epoch 9: val_loss did not improve from 0.45060
191/191 ━━━━━━━━━━━━━━━━━━━━ 23s 113ms/step - accuracy: 0.8526 - loss:
0.3706 - val_accuracy: 0.7991 - val_loss: 0.5002 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step - accuracy: 0.8626 - loss:
0.3421
Epoch 10: val_loss did not improve from 0.45060
191/191 ━━━━━━━━━━━━━━━━━━━━ 23s 118ms/step - accuracy: 0.8626 - loss:
0.3421 - val_accuracy: 0.8076 - val_loss: 0.5051 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step
Fold completed with Accuracy: 0.8082731451083388, Precision:
0.8227848101265823, Recall: 0.7010785824345146, F1-Score:
0.7570715474209652
```

```
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.5354 - loss:
0.8688
Epoch 1: val_loss improved from inf to 0.65472, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 26s 117ms/step - accuracy: 0.5355 - loss:
0.8684 - val_accuracy: 0.7085 - val_loss: 0.6547 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.6468 - loss:
0.6443
Epoch 2: val_loss improved from 0.65472 to 0.54468, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 110ms/step - accuracy: 0.6469 - loss:
0.6442 - val_accuracy: 0.7761 - val_loss: 0.5447 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.7385 - loss:
0.5530
Epoch 3: val_loss improved from 0.54468 to 0.48289, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 100ms/step - accuracy: 0.7385 - loss:
0.5530 - val_accuracy: 0.7905 - val_loss: 0.4829 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7692 - loss:
0.5179
Epoch 4: val_loss improved from 0.48289 to 0.45932, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 101ms/step - accuracy: 0.7692 - loss:
0.5178 - val_accuracy: 0.8024 - val_loss: 0.4593 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step - accuracy: 0.8005 - loss:
0.4791
Epoch 5: val_loss improved from 0.45932 to 0.44649, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 107ms/step - accuracy: 0.8005 - loss:
0.4791 - val_accuracy: 0.8063 - val_loss: 0.4465 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.8216 - loss:
0.4452
Epoch 6: val_loss did not improve from 0.44649
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 114ms/step - accuracy: 0.8216 - loss:
0.4452 - val_accuracy: 0.8024 - val_loss: 0.4497 - learning_rate:
0.0010
Epoch 7/10
```

```
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.8358 - loss:
0.4058
Epoch 7: val_loss did not improve from 0.44649
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 116ms/step - accuracy: 0.8358 - loss:
0.4058 - val_accuracy: 0.8083 - val_loss: 0.4534 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.8526 - loss:
0.3947
Epoch 8: val_loss did not improve from 0.44649
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 98ms/step - accuracy: 0.8526 - loss:
0.3947 - val_accuracy: 0.8089 - val_loss: 0.4925 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.8702 - loss:
0.3464
Epoch 9: val_loss did not improve from 0.44649
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 98ms/step - accuracy: 0.8702 - loss:
0.3464 - val_accuracy: 0.7971 - val_loss: 0.4743 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step - accuracy: 0.8723 - loss:
0.3414
Epoch 10: val_loss did not improve from 0.44649
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 106ms/step - accuracy: 0.8723 - loss:
0.3414 - val_accuracy: 0.7846 - val_loss: 0.5093 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step
Fold completed with Accuracy: 0.8063033486539725, Precision:
0.8568548387096774, Recall: 0.6548536209553159, F1-Score:
0.7423580786026202
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.5567 - loss:
0.8346
Epoch 1: val_loss improved from inf to 0.66980, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 26s 103ms/step - accuracy: 0.5567 - loss:
0.8343 - val_accuracy: 0.6697 - val_loss: 0.6698 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.6505 - loss:
0.6465
Epoch 2: val_loss improved from 0.66980 to 0.55014, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 111ms/step - accuracy: 0.6506 - loss:
0.6464 - val_accuracy: 0.7814 - val_loss: 0.5501 - learning_rate:
0.0010
Epoch 3/10
```

```
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.7110 - loss:
0.5735
Epoch 3: val_loss improved from 0.55014 to 0.48042, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 114ms/step - accuracy: 0.7111 - loss:
0.5735 - val_accuracy: 0.7886 - val_loss: 0.4804 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.7643 - loss:
0.5315
Epoch 4: val_loss improved from 0.48042 to 0.44056, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 103ms/step - accuracy: 0.7643 - loss:
0.5314 - val_accuracy: 0.8089 - val_loss: 0.4406 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.7885 - loss:
0.4806
Epoch 5: val_loss improved from 0.44056 to 0.42585, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 19s 97ms/step - accuracy: 0.7885 - loss:
0.4806 - val_accuracy: 0.8234 - val_loss: 0.4258 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.8116 - loss:
0.4436
Epoch 6: val_loss did not improve from 0.42585
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 96ms/step - accuracy: 0.8116 - loss:
0.4437 - val_accuracy: 0.8083 - val_loss: 0.4456 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.8249 - loss:
0.4241
Epoch 7: val_loss did not improve from 0.42585
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 104ms/step - accuracy: 0.8249 - loss:
0.4242 - val_accuracy: 0.8043 - val_loss: 0.4523 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.8403 - loss:
0.3926
Epoch 8: val_loss did not improve from 0.42585
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 115ms/step - accuracy: 0.8403 - loss:
0.3926 - val_accuracy: 0.8221 - val_loss: 0.4376 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.8634 - loss:
0.3587
Epoch 9: val_loss did not improve from 0.42585
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 119ms/step - accuracy: 0.8633 - loss:
```

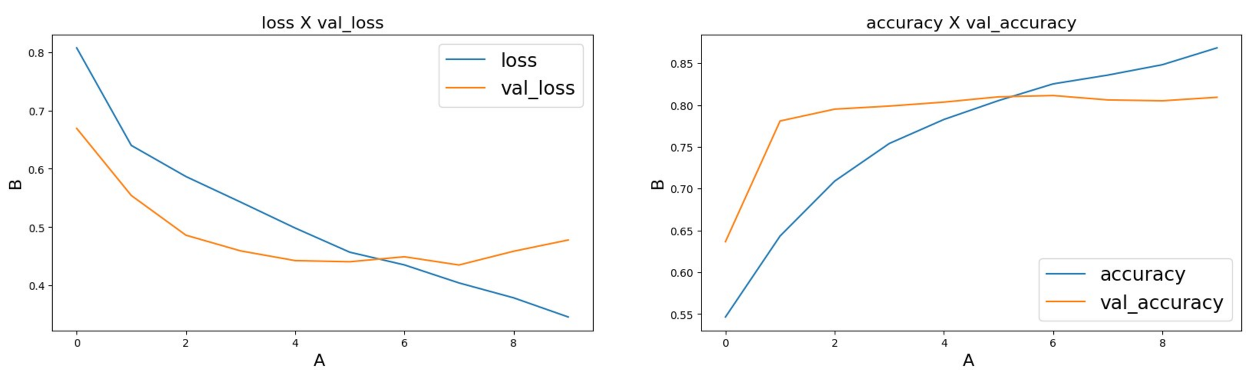
```
0.3587 - val_accuracy: 0.8188 - val_loss: 0.4586 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.8773 - loss:
0.3331
Epoch 10: val_loss did not improve from 0.42585
191/191 ━━━━━━━━━━━━━━━━━━━━ 36s 95ms/step - accuracy: 0.8773 - loss:
0.3331 - val_accuracy: 0.8148 - val_loss: 0.4812 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step
Fold completed with Accuracy: 0.8233749179251477, Precision:
0.8512544802867383, Recall: 0.7186081694402421, F1-Score:
0.7793273174733388
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.5488 - loss:
0.8294
Epoch 1: val_loss improved from inf to 0.64139, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 28s 121ms/step - accuracy: 0.5490 - loss:
0.8291 - val_accuracy: 0.7148 - val_loss: 0.6414 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.6592 - loss:
0.6420
Epoch 2: val_loss improved from 0.64139 to 0.54350, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 37s 102ms/step - accuracy: 0.6593 - loss:
0.6419 - val_accuracy: 0.7714 - val_loss: 0.5435 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.7433 - loss:
0.5541
Epoch 3: val_loss improved from 0.54350 to 0.48315, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 103ms/step - accuracy: 0.7433 - loss:
0.5541 - val_accuracy: 0.7930 - val_loss: 0.4831 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.7899 - loss:
0.5077
Epoch 4: val_loss improved from 0.48315 to 0.46800, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 23s 116ms/step - accuracy: 0.7899 - loss:
0.5077 - val_accuracy: 0.7950 - val_loss: 0.4680 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.7925 - loss:
0.4833
```

```
Epoch 5: val_loss improved from 0.46800 to 0.45654, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 115ms/step - accuracy: 0.7926 - loss:
0.4833 - val_accuracy: 0.7950 - val_loss: 0.4565 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.8154 - loss:
0.4470
Epoch 6: val_loss did not improve from 0.45654
191/191 ━━━━━━━━━━━━━━━━━━━━ 37s 94ms/step - accuracy: 0.8155 - loss:
0.4470 - val_accuracy: 0.8009 - val_loss: 0.4727 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.8255 - loss:
0.4140
Epoch 7: val_loss did not improve from 0.45654
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 101ms/step - accuracy: 0.8256 - loss:
0.4140 - val_accuracy: 0.8068 - val_loss: 0.4600 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.8387 - loss:
0.3954
Epoch 8: val_loss did not improve from 0.45654
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 102ms/step - accuracy: 0.8388 - loss:
0.3953 - val_accuracy: 0.8042 - val_loss: 0.4723 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.8602 - loss:
0.3541
Epoch 9: val_loss did not improve from 0.45654
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 115ms/step - accuracy: 0.8602 - loss:
0.3541 - val_accuracy: 0.7957 - val_loss: 0.4742 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.8753 - loss:
0.3299
Epoch 10: val_loss did not improve from 0.45654
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 116ms/step - accuracy: 0.8753 - loss:
0.3299 - val_accuracy: 0.7970 - val_loss: 0.5221 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step
Fold completed with Accuracy: 0.7950065703022339, Precision:
0.8914027149321267, Recall: 0.5987841945288754, F1-Score:
0.7163636363636364
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.5697 - loss:
0.7393
Epoch 1: val_loss improved from inf to 0.64512, saving model to
```

```
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.5698 - loss:
0.7391 - val_accuracy: 0.7254 - val_loss: 0.6451 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.6725 - loss:
0.6032
Epoch 2: val_loss improved from 0.64512 to 0.54147, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 116ms/step - accuracy: 0.6726 - loss:
0.6032 - val_accuracy: 0.7484 - val_loss: 0.5415 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.7509 - loss:
0.5359
Epoch 3: val_loss improved from 0.54147 to 0.49669, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 102ms/step - accuracy: 0.7509 - loss:
0.5359 - val_accuracy: 0.7608 - val_loss: 0.4967 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7926 - loss:
0.4967
Epoch 4: val_loss did not improve from 0.49669
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 107ms/step - accuracy: 0.7926 - loss:
0.4966 - val_accuracy: 0.7700 - val_loss: 0.4970 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.8144 - loss:
0.4504
Epoch 5: val_loss did not improve from 0.49669
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 110ms/step - accuracy: 0.8144 - loss:
0.4503 - val_accuracy: 0.7838 - val_loss: 0.4993 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 0.8258 - loss:
0.4204
Epoch 6: val_loss improved from 0.49669 to 0.49620, saving model to
model_glove_lstm_200.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 23s 120ms/step - accuracy: 0.8258 - loss:
0.4204 - val_accuracy: 0.7943 - val_loss: 0.4962 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.8510 - loss:
0.3904
Epoch 7: val_loss did not improve from 0.49620
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 122ms/step - accuracy: 0.8510 - loss:
0.3905 - val_accuracy: 0.7878 - val_loss: 0.5091 - learning_rate:
0.0010
```

```
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.8572 - loss:
0.3778
Epoch 8: val_loss did not improve from 0.49620
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 109ms/step - accuracy: 0.8572 - loss:
0.3778 - val_accuracy: 0.7970 - val_loss: 0.5317 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.8736 - loss:
0.3482
Epoch 9: val_loss did not improve from 0.49620
191/191 ━━━━━━━━━━━━━━━━━━━━ 40s 103ms/step - accuracy: 0.8736 - loss:
0.3482 - val_accuracy: 0.7970 - val_loss: 0.5640 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.8764 - loss:
0.3390
Epoch 10: val_loss did not improve from 0.49620
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 109ms/step - accuracy: 0.8764 - loss:
0.3389 - val_accuracy: 0.7812 - val_loss: 0.6148 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step
Fold completed with Accuracy: 0.7943495400788436, Precision:
0.8310679611650486, Recall: 0.654434250764526, F1-Score:
0.7322497861420016
--------------------------------------------------
Average Accuracy: 0.8054615044137072
Average Precision: 0.8506729610440347
Average Recall: 0.6655517636246948
Average F1-Score: 0.7454740732005124
```

```
plot_learning_curves(history_glove_lstm, [['loss', 'val_loss'],
['accuracy', 'val_accuracy']])
```




```
model_glove_lstm_final = load_model(model_glove_lstm_filename)
y_pred_glove_lstm = (model_glove_lstm_final.predict(X_val) >
0.5).astype("int32")
```

```
cm = confusion_matrix(y_val,y_pred_glove_lstm)
print('confusion matrix:\n',cm)
print(classification_report(y_val, y_pred_glove_lstm))
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step
confusion matrix:
 [[991  80]
 [289 544]]
              precision    recall  f1-score   support

           0       0.77      0.93      0.84      1071
           1       0.87      0.65      0.75       833

    accuracy                           0.81      1904
   macro avg       0.82      0.79      0.79      1904
weighted avg       0.82      0.81      0.80      1904
```

Submission

```
y_glove_lstm = (model_glove_lstm_final.predict(test_padded_sentences)
> 0.5).astype("int32")
y_labels = []
for i in range (0, len(y_glove_lstm)):
    y_labels.append(y_glove_lstm[i][0])

# create submission file
submission_glove_lstm = pd.DataFrame({"id": (test_df['id']),"target":
y_labels})
submission_glove_lstm.to_csv('submission_glove_lstm_200.csv',
index=False)
```

```
102/102 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step
```

Model 3: GLoVe-LSTM 300D

```
embeddings_dictionary = dict()
embedding_dim = 300

# Load GloVe 300D embeddings
with open('glove.6B.300d.txt', encoding="utf8") as fp:
    for line in fp.readlines():
        records = line.split()
        word = records[0]
        vector_dimensions = np.asarray(records[1:], dtype='float32')
        embeddings_dictionary [word] = vector_dimensions


embedding_matrix = np.zeros((vocab_length, embedding_dim))
```

```
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

def glove_lstm_300():
    model = Sequential()

    model.add(Embedding(
        input_dim=embedding_matrix.shape[0],
        output_dim=embedding_matrix.shape[1],
        weights = [embedding_matrix],
        input_length=length_long_sentence
    ))

    model.add(Bidirectional(LSTM(
        length_long_sentence,
        return_sequences = True,
        recurrent_dropout=0.2
    )))

    model.add(GlobalMaxPool1D())
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation = 'sigmoid'))
    model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

    return model

model = glove_lstm_300()

model_glove_lstm_filename = 'model_glove_lstm_300.keras'
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

# K-Fold Cross Validation
for train_index, val_index in kf.split(train_padded_sentences):
    X_train_fold, X_val_fold = train_padded_sentences[train_index],
train_padded_sentences[val_index]
    y_train_fold, y_val_fold = train_target[train_index],
train_target[val_index]

    model_glove_lstm = glove_lstm_300()  # Instantiate a new model for
each fold
```

```python
    checkpoint = ModelCheckpoint(
        model_glove_lstm_filename,
        monitor='val_loss',
        verbose=1,
        save_best_only=True
    )

    reduce_lr = ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        verbose=1,
        patience=5,
        min_lr=0.001
    )

    history_glove_lstm = model_glove_lstm.fit(
        X_train_fold,
        y_train_fold,
        epochs=10,
        batch_size=32,
        validation_data=(X_val_fold, y_val_fold),
        verbose=1,
        callbacks=[reduce_lr, checkpoint]
    )

    # Load the best model from this fold
    model_final = load_model(model_glove_lstm_filename)
    y_pred = (model_final.predict(X_val_fold) > 0.5).astype("int32")

    # Calculate metrics
    accuracy_scores.append(accuracy_score(y_val_fold, y_pred))
    precision_scores.append(precision_score(y_val_fold, y_pred))
    recall_scores.append(recall_score(y_val_fold, y_pred))
    f1_scores.append(f1_score(y_val_fold, y_pred))

    print(f"Fold completed with Accuracy: {accuracy_scores[-1]}, Precision: {precision_scores[-1]}, Recall: {recall_scores[-1]}, F1-Score: {f1_scores[-1]}")
    print('-'*50)

# Compute average performance across all folds
print(f"Average Accuracy: {np.mean(accuracy_scores)}")
print(f"Average Precision: {np.mean(precision_scores)}")
print(f"Average Recall: {np.mean(recall_scores)}")
print(f"Average F1-Score: {np.mean(f1_scores)}")
```

```
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.5362 - loss: 0.9592
```

```
Epoch 1: val_loss improved from inf to 0.64454, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 28s 122ms/step - accuracy: 0.5363 - loss:
0.9586 - val_accuracy: 0.7242 - val_loss: 0.6445 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.6540 - loss:
0.6429
Epoch 2: val_loss improved from 0.64454 to 0.53397, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 112ms/step - accuracy: 0.6541 - loss:
0.6427 - val_accuracy: 0.7748 - val_loss: 0.5340 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7359 - loss:
0.5549
Epoch 3: val_loss improved from 0.53397 to 0.45968, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 103ms/step - accuracy: 0.7360 - loss:
0.5549 - val_accuracy: 0.8004 - val_loss: 0.4597 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step - accuracy: 0.7852 - loss:
0.4925
Epoch 4: val_loss improved from 0.45968 to 0.44347, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 110ms/step - accuracy: 0.7852 - loss:
0.4924 - val_accuracy: 0.7958 - val_loss: 0.4435 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step - accuracy: 0.8013 - loss:
0.4720
Epoch 5: val_loss improved from 0.44347 to 0.44215, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 118ms/step - accuracy: 0.8013 - loss:
0.4720 - val_accuracy: 0.7997 - val_loss: 0.4422 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.8201 - loss:
0.4322
Epoch 6: val_loss did not improve from 0.44215
191/191 ━━━━━━━━━━━━━━━━━━━━ 40s 114ms/step - accuracy: 0.8201 - loss:
0.4322 - val_accuracy: 0.8056 - val_loss: 0.4442 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.8456 - loss:
0.3840
Epoch 7: val_loss did not improve from 0.44215
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 103ms/step - accuracy: 0.8456 - loss:
```

```
0.3841 - val_accuracy: 0.8011 - val_loss: 0.4541 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step - accuracy: 0.8569 - loss:
0.3767
Epoch 8: val_loss did not improve from 0.44215
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 104ms/step - accuracy: 0.8569 - loss:
0.3767 - val_accuracy: 0.8043 - val_loss: 0.4664 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.8592 - loss:
0.3551
Epoch 9: val_loss did not improve from 0.44215
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 112ms/step - accuracy: 0.8592 - loss:
0.3550 - val_accuracy: 0.7991 - val_loss: 0.5036 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.8760 - loss:
0.3219
Epoch 10: val_loss did not improve from 0.44215
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 115ms/step - accuracy: 0.8760 - loss:
0.3218 - val_accuracy: 0.7997 - val_loss: 0.5735 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step
Fold completed with Accuracy: 0.7997373604727511, Precision:
0.8659574468085106, Recall: 0.6271186440677966, F1-Score:
0.7274352100089365
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.5353 - loss:
0.8516
Epoch 1: val_loss improved from inf to 0.65912, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.5354 - loss:
0.8512 - val_accuracy: 0.7525 - val_loss: 0.6591 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.6432 - loss:
0.6447
Epoch 2: val_loss improved from 0.65912 to 0.55770, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 116ms/step - accuracy: 0.6434 - loss:
0.6446 - val_accuracy: 0.7853 - val_loss: 0.5577 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.7122 - loss:
0.5690
Epoch 3: val_loss improved from 0.55770 to 0.46311, saving model to
model_glove_lstm_300.keras
```

```
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 100ms/step - accuracy: 0.7123 - loss:
0.5689 - val_accuracy: 0.7892 - val_loss: 0.4631 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.7626 - loss:
0.5209
Epoch 4: val_loss improved from 0.46311 to 0.45512, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 103ms/step - accuracy: 0.7627 - loss:
0.5208 - val_accuracy: 0.8024 - val_loss: 0.4551 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.7984 - loss:
0.4657
Epoch 5: val_loss improved from 0.45512 to 0.45205, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 114ms/step - accuracy: 0.7984 - loss:
0.4657 - val_accuracy: 0.8043 - val_loss: 0.4521 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8313 - loss:
0.4255
Epoch 6: val_loss did not improve from 0.45205
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 116ms/step - accuracy: 0.8313 - loss:
0.4255 - val_accuracy: 0.8102 - val_loss: 0.4554 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.8459 - loss:
0.3777
Epoch 7: val_loss did not improve from 0.45205
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 102ms/step - accuracy: 0.8459 - loss:
0.3777 - val_accuracy: 0.8135 - val_loss: 0.4603 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.8577 - loss:
0.3646
Epoch 8: val_loss did not improve from 0.45205
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 105ms/step - accuracy: 0.8577 - loss:
0.3646 - val_accuracy: 0.8122 - val_loss: 0.4697 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.8738 - loss:
0.3276
Epoch 9: val_loss did not improve from 0.45205
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 105ms/step - accuracy: 0.8738 - loss:
0.3277 - val_accuracy: 0.8135 - val_loss: 0.4911 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.8852 - loss:
```

```
0.3056
Epoch 10: val_loss did not improve from 0.45205
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 116ms/step - accuracy: 0.8852 - loss:
0.3056 - val_accuracy: 0.8109 - val_loss: 0.5317 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step
Fold completed with Accuracy: 0.804333552199606, Precision:
0.8268156424581006, Recall: 0.6841294298921418, F1-Score:
0.7487352445193929
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.5315 - loss:
0.8598
Epoch 1: val_loss improved from inf to 0.64592, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 29s 119ms/step - accuracy: 0.5316 - loss:
0.8594 - val_accuracy: 0.7177 - val_loss: 0.6459 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step - accuracy: 0.6401 - loss:
0.6585
Epoch 2: val_loss improved from 0.64592 to 0.55128, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 120ms/step - accuracy: 0.6403 - loss:
0.6584 - val_accuracy: 0.7794 - val_loss: 0.5513 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.7262 - loss:
0.5611
Epoch 3: val_loss improved from 0.55128 to 0.45843, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 38s 107ms/step - accuracy: 0.7262 - loss:
0.5611 - val_accuracy: 0.8011 - val_loss: 0.4584 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.7790 - loss:
0.5042
Epoch 4: val_loss improved from 0.45843 to 0.43301, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 111ms/step - accuracy: 0.7790 - loss:
0.5042 - val_accuracy: 0.8129 - val_loss: 0.4330 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step - accuracy: 0.7948 - loss:
0.4583
Epoch 5: val_loss improved from 0.43301 to 0.42854, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 41s 110ms/step - accuracy: 0.7948 - loss:
0.4583 - val_accuracy: 0.8063 - val_loss: 0.4285 - learning_rate:
```

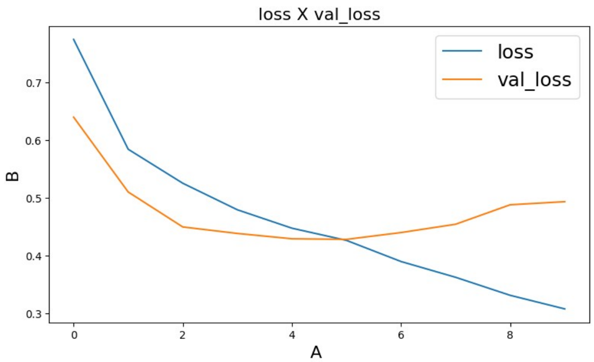
```
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8309 - loss:
0.4128
Epoch 6: val_loss did not improve from 0.42854
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 116ms/step - accuracy: 0.8309 - loss:
0.4128 - val_accuracy: 0.8037 - val_loss: 0.4388 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.8493 - loss:
0.3915
Epoch 7: val_loss did not improve from 0.42854
191/191 ━━━━━━━━━━━━━━━━━━━━ 39s 107ms/step - accuracy: 0.8493 - loss:
0.3915 - val_accuracy: 0.8122 - val_loss: 0.4682 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.8679 - loss:
0.3554
Epoch 8: val_loss did not improve from 0.42854
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 106ms/step - accuracy: 0.8679 - loss:
0.3555 - val_accuracy: 0.7978 - val_loss: 0.4987 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.8810 - loss:
0.3189
Epoch 9: val_loss did not improve from 0.42854
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 101ms/step - accuracy: 0.8810 - loss:
0.3189 - val_accuracy: 0.8030 - val_loss: 0.5566 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8960 - loss:
0.2814
Epoch 10: val_loss did not improve from 0.42854
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 116ms/step - accuracy: 0.8960 - loss:
0.2815 - val_accuracy: 0.8102 - val_loss: 0.5514 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step
Fold completed with Accuracy: 0.8063033486539725, Precision:
0.823321554770318, Recall: 0.7049924357034796, F1-Score:
0.7595762021189895
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.5307 - loss:
0.8092
Epoch 1: val_loss improved from inf to 0.67941, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 27s 110ms/step - accuracy: 0.5308 - loss:
0.8089 - val_accuracy: 0.6117 - val_loss: 0.6794 - learning_rate:
0.0010
```

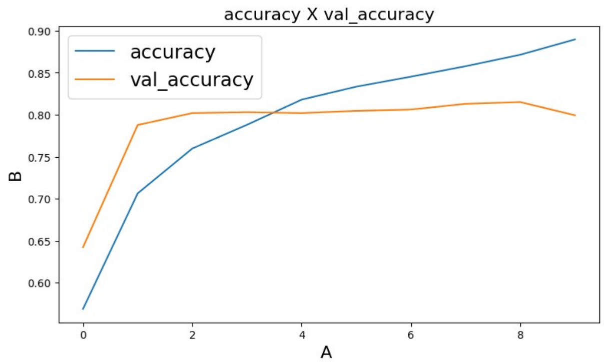
```
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.6256 - loss:
0.6572
Epoch 2: val_loss improved from 0.67941 to 0.54711, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 111ms/step - accuracy: 0.6257 - loss:
0.6571 - val_accuracy: 0.7694 - val_loss: 0.5471 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.7226 - loss:
0.5638
Epoch 3: val_loss improved from 0.54711 to 0.48613, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 115ms/step - accuracy: 0.7227 - loss:
0.5637 - val_accuracy: 0.7871 - val_loss: 0.4861 - learning_rate:
0.0010
Epoch 4/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.7648 - loss:
0.5121
Epoch 4: val_loss improved from 0.48613 to 0.47290, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 117ms/step - accuracy: 0.7649 - loss:
0.5120 - val_accuracy: 0.7884 - val_loss: 0.4729 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.8093 - loss:
0.4615
Epoch 5: val_loss improved from 0.47290 to 0.46079, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 104ms/step - accuracy: 0.8093 - loss:
0.4615 - val_accuracy: 0.7989 - val_loss: 0.4608 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step - accuracy: 0.8460 - loss:
0.3942
Epoch 6: val_loss did not improve from 0.46079
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 100ms/step - accuracy: 0.8460 - loss:
0.3943 - val_accuracy: 0.7930 - val_loss: 0.4640 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.8398 - loss:
0.3901
Epoch 7: val_loss improved from 0.46079 to 0.45924, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 101ms/step - accuracy: 0.8398 - loss:
0.3901 - val_accuracy: 0.7989 - val_loss: 0.4592 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step - accuracy: 0.8763 - loss:
```

```
0.3434
Epoch 8: val_loss did not improve from 0.45924
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 106ms/step - accuracy: 0.8763 - loss:
0.3434 - val_accuracy: 0.7878 - val_loss: 0.5252 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step - accuracy: 0.8857 - loss:
0.3150
Epoch 9: val_loss did not improve from 0.45924
191/191 ━━━━━━━━━━━━━━━━━━━━ 23s 119ms/step - accuracy: 0.8857 - loss:
0.3151 - val_accuracy: 0.7845 - val_loss: 0.5248 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.8932 - loss:
0.2904
Epoch 10: val_loss did not improve from 0.45924
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 122ms/step - accuracy: 0.8932 - loss:
0.2904 - val_accuracy: 0.7943 - val_loss: 0.5479 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step
Fold completed with Accuracy: 0.7989487516425755, Precision:
0.8271375464684015, Recall: 0.6762917933130699, F1-Score:
0.7441471571906354
--------------------------------------------------
Epoch 1/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step - accuracy: 0.5491 - loss:
0.8968
Epoch 1: val_loss improved from inf to 0.64961, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 27s 121ms/step - accuracy: 0.5493 - loss:
0.8962 - val_accuracy: 0.7024 - val_loss: 0.6496 - learning_rate:
0.0010
Epoch 2/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.6695 - loss:
0.6196
Epoch 2: val_loss improved from 0.64961 to 0.53830, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 117ms/step - accuracy: 0.6696 - loss:
0.6195 - val_accuracy: 0.7497 - val_loss: 0.5383 - learning_rate:
0.0010
Epoch 3/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.7328 - loss:
0.5559
Epoch 3: val_loss improved from 0.53830 to 0.49361, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 37s 96ms/step - accuracy: 0.7328 - loss:
0.5559 - val_accuracy: 0.7727 - val_loss: 0.4936 - learning_rate:
0.0010
Epoch 4/10
```

```
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.7850 - loss:
0.5042
Epoch 4: val_loss did not improve from 0.49361
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 97ms/step - accuracy: 0.7850 - loss:
0.5042 - val_accuracy: 0.7733 - val_loss: 0.5082 - learning_rate:
0.0010
Epoch 5/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.8198 - loss:
0.4418
Epoch 5: val_loss improved from 0.49361 to 0.47706, saving model to
model_glove_lstm_300.keras
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 112ms/step - accuracy: 0.8198 - loss:
0.4418 - val_accuracy: 0.7898 - val_loss: 0.4771 - learning_rate:
0.0010
Epoch 6/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8204 - loss:
0.4172
Epoch 6: val_loss did not improve from 0.47706
191/191 ━━━━━━━━━━━━━━━━━━━━ 42s 116ms/step - accuracy: 0.8205 - loss:
0.4172 - val_accuracy: 0.7904 - val_loss: 0.4822 - learning_rate:
0.0010
Epoch 7/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.8446 - loss:
0.3878
Epoch 7: val_loss did not improve from 0.47706
191/191 ━━━━━━━━━━━━━━━━━━━━ 22s 113ms/step - accuracy: 0.8446 - loss:
0.3878 - val_accuracy: 0.7858 - val_loss: 0.5440 - learning_rate:
0.0010
Epoch 8/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.8697 - loss:
0.3379
Epoch 8: val_loss did not improve from 0.47706
191/191 ━━━━━━━━━━━━━━━━━━━━ 20s 102ms/step - accuracy: 0.8697 - loss:
0.3380 - val_accuracy: 0.7957 - val_loss: 0.5288 - learning_rate:
0.0010
Epoch 9/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.8822 - loss:
0.3088
Epoch 9: val_loss did not improve from 0.47706
191/191 ━━━━━━━━━━━━━━━━━━━━ 19s 97ms/step - accuracy: 0.8822 - loss:
0.3088 - val_accuracy: 0.7898 - val_loss: 0.6474 - learning_rate:
0.0010
Epoch 10/10
191/191 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.8872 - loss:
0.2978
Epoch 10: val_loss did not improve from 0.47706
191/191 ━━━━━━━━━━━━━━━━━━━━ 21s 110ms/step - accuracy: 0.8872 - loss:
0.2978 - val_accuracy: 0.7957 - val_loss: 0.6314 - learning_rate:
0.0010
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step
```

```
Fold completed with Accuracy: 0.7897503285151117, Precision: 0.834,
Recall: 0.6376146788990825, F1-Score: 0.7227036395147313
--------------------------------------------------
Average Accuracy: 0.7998146682968034
Average Precision: 0.8354464381010661
Average Recall: 0.666029396375114
Average F1-Score: 0.7405194906705372

plot_learning_curves(history_glove_lstm, [['loss', 'val_loss'],
['accuracy', 'val_accuracy']])
```






```
model_glove_lstm_final = load_model(model_glove_lstm_filename)
y_pred_glove_lstm = (model_glove_lstm_final.predict(X_val) >
0.5).astype("int32")
cm = confusion_matrix(y_val,y_pred_glove_lstm)
print('confusion matrix:\n',cm)
print(classification_report(y_val, y_pred_glove_lstm))

60/60 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step
confusion matrix:
 [[1001   70]
 [ 302  531]]
              precision    recall  f1-score   support

           0       0.77      0.93      0.84      1071
           1       0.88      0.64      0.74       833

    accuracy                           0.80      1904
   macro avg       0.83      0.79      0.79      1904
weighted avg       0.82      0.80      0.80      1904
```

Submission

```
y_glove_lstm = (model_glove_lstm_final.predict(test_padded_sentences)
> 0.5).astype("int32")
y_labels = []
for i in range (0, len(y_glove_lstm)):
```

```
    y_labels.append(y_glove_lstm[i][0])

# create submission file
submission_glove_lstm = pd.DataFrame({"id": (test_df['id']),"target":
y_labels})
submission_glove_lstm.to_csv('submission_glove_lstm_300.csv',
index=False)
```

```
102/102 ━━━━━━━━━━━━━━━━━━━━ 4s 36ms/step
```

The management or research question for this project is: "**How can we accurately determine if a tweet is about a real disaster or not?**"

Why is this important?

In today's world, where information flows rapidly through social media platforms like Twitter, it's crucial for disaster relief organizations, news agencies, and emergency services to quickly and accurately identify genuine disaster-related tweets. These tweets can provide real-time information during emergencies, helping to mobilize resources, provide timely assistance, and potentially save lives.

However, not all tweets that mention disasters are actually about real events. Some may use disaster-related language metaphorically or humorously, which can confuse automated systems. By developing a model that can reliably distinguish between real disaster tweets and those that aren't, we can ensure that critical information reaches the right people at the right time, making response efforts more effective and efficient.