Fake News Detection analysis- LSTM Classification

Import Modules

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import re
import nltk
import warnings
%matplotlib inline

warnings.filterwarnings('ignore')
```

Loading the Dataset

```
Start coding or generate with AI.
```

```
import pandas as pd
import numpy as np

# Try reading the CSV file with a different encoding, like 'latin-1'
# If 'latin-1' doesn't work, try other encodings like 'ISO-8859-1', 'cp1252', etc.

#Try using the 'python' engine
# Changed 'error_bad_lines' to 'on_bad_lines' and set it to 'skip' to skip bad lines
df = pd.read_csv('/content/train.csv', encoding='latin-1', engine='python', on_bad_lines='skip')
# If 'latin-1' doesn't work, try other encodings like 'ISO-8859-1', 'cp1252', etc.

df.head()
```

	id	title	author	text	label
0	0	House Dem Aide: We Didnâ□□t Even See Comeyâ□□s...	Darrell Lucus	House Dem Aide: We Didnâ□□t Even See Comeyâ□□s...	1.0
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0.0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1.0
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1.0

```
df['title'][0]
```

```
df['text'][0]
```

```
'House Dem Aide: We Didnâ\x80\x99t Even See Comeyâ\x80\x99s Letter Until Jason Chaffetz Tweeted It By Darrell Lucus on October 30,
2016 Subscribe Jason Chaffetz on the stump in American Fork, Utah ( image courtesy Michael Jolley, available under a Creative Commo
ns-BY license) \nWith apologies to Keith Olbermann, there is no doubt who the Worst Person in The World is this weekâ\x80\x93FBI Di
rector James Comey. But according to a House Democratic aide, it looks like we also know who the second-worst person is as well. It
turns out that when Comey sent his now-infamous letter announcing that the FBI was looking into emails that may be related to Hilla
ry Clintonâ\x80\x99s email server, the ranking Democrats on the relevant committees didnâ\x80\x99t hear about it from Comey. They f
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14234 entries, 0 to 14233
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      14234 non-null  object
 1   title   13851 non-null  object
 2   author  12875 non-null  object
 3   text    14188 non-null  object
 4   label   14218 non-null  float64
dtypes: float64(1), object(4)
memory usage: 556.1+ KB
```

Data Proprocessing

```
# drop unnecessary columns
df = df.drop(columns=['id', 'title', 'author'], axis=1)
```

```
# drop null values
df = df.dropna(axis=0)
```

```
len(df)
```

```
14187
```

```
# remove special characters and punctuations
```

```
df['clean_news'] = df['text'].str.lower()
df['clean_news']
```

```
                                               clean_news
    0    house dem aide: we didnâ□□t even see comeyâ□□s...
    1           ever get the feeling your life circles the rou...
    2           why the truth might get you fired october 29, ...
    3            videos 15 civilians killed in single us airstr...
    4        print \nan iranian woman has been sentenced to...
  ...                                                       ...
14229        by dean baker, the co-founder of cepr. origina...
14230      america has a new pharmaceutical villain. her ...
14231     non-mainstream poll shows trump poised to win ...
14232   6 reasons why youâ□□re not happy 6 reasons w...
14233      in less than a day, the american populace will...

14187 rows × 1 columns
```

```
df['clean_news'] = df['clean_news'].str.replace('[^A-Za-z0-9\s]', '')
df['clean_news'] = df['clean_news'].str.replace('\n', '')
df['clean_news'] = df['clean_news'].str.replace('\s+', ' ')
df['clean_news']
```

```
                                               clean_news
    0    house dem aide: we didnâ□□t even see comeyâ□□s...
    1           ever get the feeling your life circles the rou...
    2           why the truth might get you fired october 29, ...
    3            videos 15 civilians killed in single us airstr...
    4       print an iranian woman has been sentenced to s...
  ...                                                       ...
14229        by dean baker, the co-founder of cepr. origina...
14230      america has a new pharmaceutical villain. her ...
14231     non-mainstream poll shows trump poised to win ...
14232   6 reasons why youâ□□re not happy 6 reasons w...
14233      in less than a day, the american populace will...

14187 rows × 1 columns
```

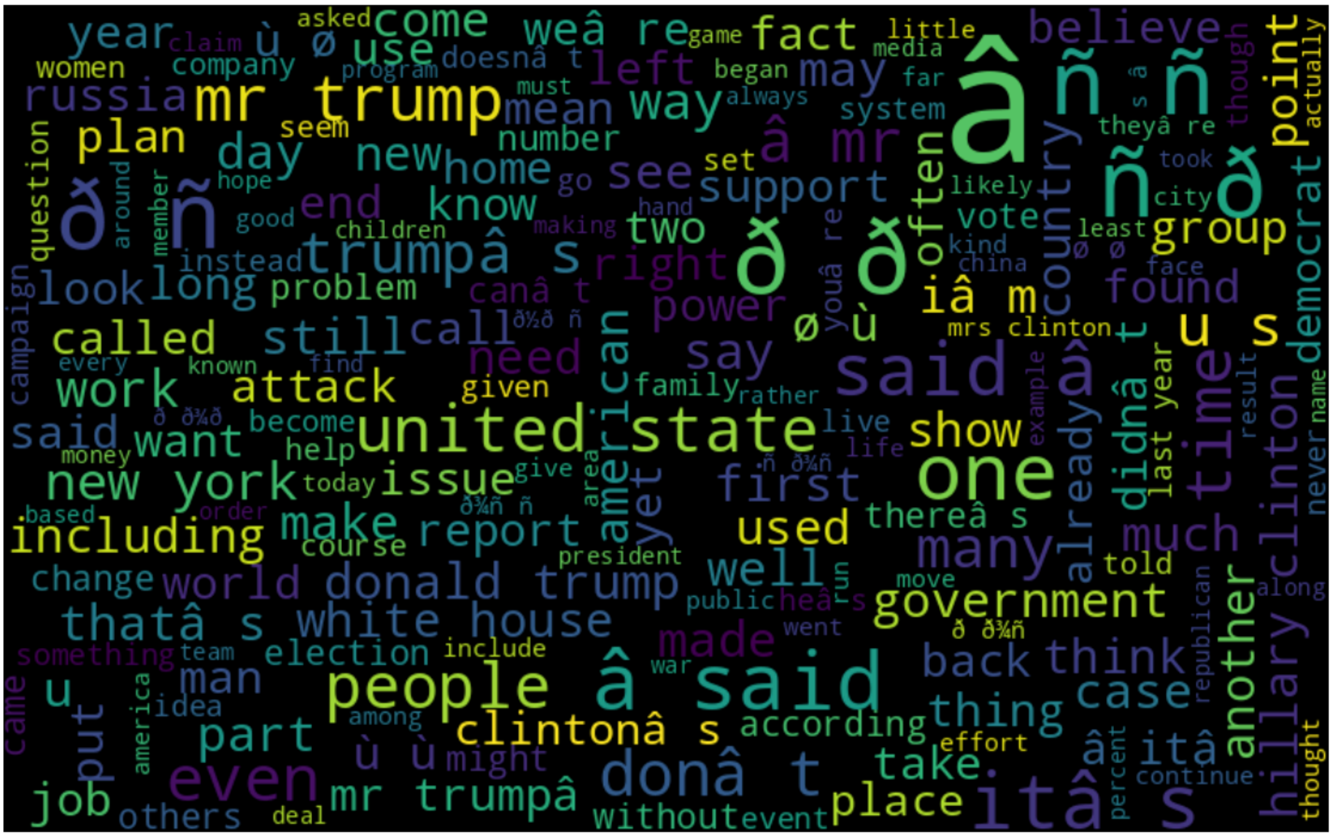
```
import nltk

# Download the 'stopwords' dataset
nltk.download('stopwords')

# Now, you can proceed with your code
from nltk.corpus import stopwords
stop = stopwords.words('english')
df['clean_news'] = df['clean_news'].apply(lambda x: " ".join([word for word in x.split() if word not in stop]))
df.head()
```

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

	text	label	clean_news
0	House Dem Aide: We Didnâ□□t Even See Comeyâ□□s...	1.0	house dem aide: didnâ□□t even see comeyâ□□s le...
1	Ever get the feeling your life circles the rou...	0.0	ever get feeling life circles roundabout rathe...
2	Why the Truth Might Get You Fired October 29, ...	1.0	truth might get fired october 29, 2016 tension...
3	Videos 15 Civilians Killed In Single US Airstr...	1.0	videos 15 civilians killed single us airstrike...
4	Print \nAn Iranian woman has been sentenced to...	1.0	print iranian woman sentenced six years prison...

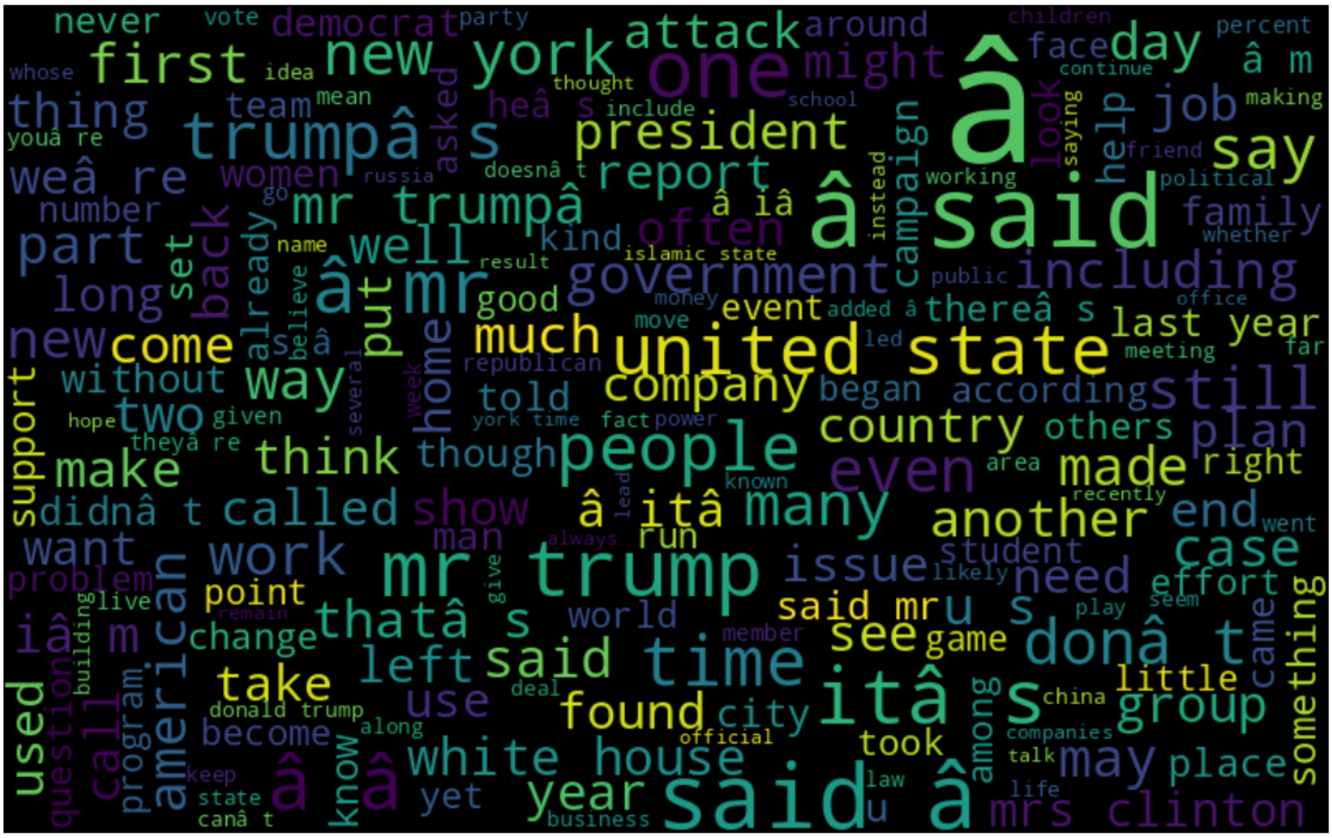
Exploratory Data Analysis

```
# visualize the frequent words
all_words = " ".join([sentence for sentence in df['clean_news']])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

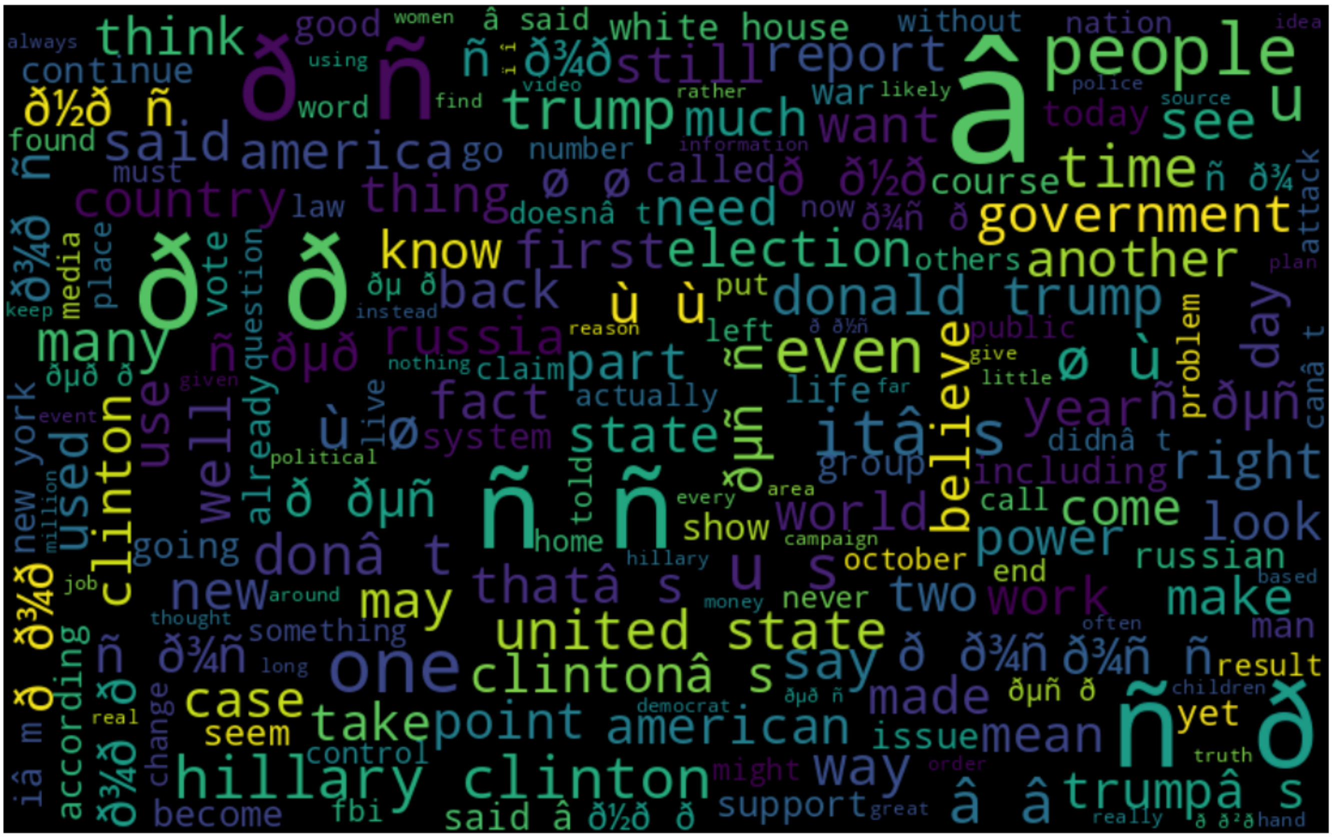


```
# visualize the frequent words for genuine news
all_words = " ".join([sentence for sentence in df['clean_news'][df['label']==0]])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)
```

```
# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
# visualize the frequent words for fake news
all_words = " ".join([sentence for sentence in df['clean_news'][df['label']==1]])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Create Word Embeddings

```
!pip install tensorflow
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.6
# tokenize text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['clean_news'])
word_index = tokenizer.word_index
vocab_size = len(word_index)
vocab_size
```

```
193392
```

```
# padding data
sequences = tokenizer.texts_to_sequences(df['clean_news'])
padded_seq = pad_sequences(sequences, maxlen=500, padding='post', truncating='post')
```

```
# create embedding index
embedding_index = {}
with open('/content/glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs
```

```
--2025-04-06 06:29:20--  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]
--2025-04-06 06:29:21--  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]
--2025-04-06 06:29:21--  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.1’

glove.6B.zip.1      100%[===================>] 822.24M  5.04MB/s    in 2m 49s

2025-04-06 06:32:11 (4.87 MB/s) - ‘glove.6B.zip.1’ saved [862182613/862182613]

Archive:  glove.6B.zip
replace glove.6B.50d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace glove.6B.100d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace glove.6B.200d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace glove.6B.300d.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
```

```
# Create the embedding matrix
embedding_matrix = np.zeros((vocab_size + 1, 100))
for word, i in word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        # Check if the embedding vector has the expected dimension
        if embedding_vector.shape == (100,):
            embedding_matrix[i] = embedding_vector
        else:
            # Handle cases where the dimension is not 100
            print(f"Warning: Embedding vector for '{word}' has unexpected shape: {embedding_vector.shape}")
            # You can choose to skip or pad these vectors to fit the expected shape
```

```
Warning: Embedding vector for 'regale' has unexpected shape: (24,)
```

```
embedding_matrix[1]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Input Split

```
padded_seq[1]
```

```
array([   256,     33,   1667,     99,   4889,  26007,    455,   2463,
         2106,    449,    569,   1780,   5773,     34,     13,    842,
          195,    400,   1976,  18014,    860,  21317,   3142,   2621,
          678,    795,    276,   2181,   3429,  10461,    142,     44,
         2312,  74657,   3296,   3877,   3809,   3429,    857,    179,
         3595,     34,  10265,  61079,   3367,     21,    234,     23,
         3555,  21317,   9631,    603,     12,    528,   2543,   2552,
         8201,    478,    778,   1645,    640,    157,     17,  26008,
          964,  27004,    365,     10,    190,   3822,     59,  11884,
          936,   2742,     19,    893,   5137,    178,     53,   5436,
         2312,   4858,     46,    279,    857,    222,     15,      4,
         2234,     15,      1,   2312,  10265,     66,   8328,  12464,
       101075,  27005,   3842, 101076, 101077, 101078,   5415,  47131,
        23305,   8202,  19221,  74658,   4595,      4,    175,    275,
        20738,   4595,    177,    385,    937,   1661,    458,   1406,
           23,    408,     41,   2096,     43,     36,    425,     68,
         8951,   3481,  18015,    192,   1559,    798,  47132,   2476,
         7656,   1755,   2559,    165,     13,    501,  10462,  11885,
        16954,   9235,   6901,    109,  74659,  74660,  10654,  14949,
        74661,    337,      9,    129,    654,    544,    405,    107,
        52836,   2536,  21318,  28150,      1,  10265,   7702,   3823,
         8201,    503,      6,    877,    183,    603,  13659,    468,
           58,    130,    565,    192,      2,  74662,   1135,    497,
         1139,   1135,     15,     44,    386,    202,    497,   1135,
         7703,    505,     58,     32,     58,   1667,  28151,   1839,
          576,     19,   2806,  20739,      1,  28150,  61080,   1046,
        12465,    313,     59,    572,   1046,     19,    634,      8,
        20739,    728,  17622,  34479,   1756,   1953,   3674,   2276,
          272,     13,    113,  20739,   1143,    316,   8102,  13283,
         1239,  18767,     57,  13660,   8005,     25,      8,      4,
            5,  21319,   3367,     78,   3127,     78,    307,    971,
         2138,  20740,     41,      4,   1750,    269,    490,   3088,
        13660,     13,    617, 101079,    515,   7760,  61081,      1,
        47133,      1,  24107,   1728,      1,    488,   1233,    618,
          490,      1,    714,   1581,     25,      8,   2477,   1456,
          618,   2083,    217,  16009,    151,    222,     48,     67,
            8,    109,  14083,   1728,    240,    307,    130,   1030,
         1048,    143,    211,    703,      1,   1057,   1792,  19696,
            1,     10,     18,      8,   5071,   3772,     34,     13,
        10807,    970,     91,   3018,   1614,    203,    178,  12466,
         1495,  21317,    528,    107,     12,    190,   2575,     13,
         9799,     81,    159,    660,   1773,   2829,    286,  32551,
        42893,   4404,     22,  21317,  13478,   5138,    135,     44,
           49,   3162,   9235,   2249,     91,    353,   2598,    499,
         1049,    378,   1432,   1325,    499,   1049,    262,    963,
         1993,    378,      1,    188,    195,    400,   1976,   4452,
          400,     50,     47,   7898,    524,   1556,     19,    378,
         3824,   1729,    641,    792,   1086,      1,    183,    986,
           35,   3385,    679,     12,    400,   6409,   7899, 101080,
        39477,    170,  47134,   7514,   1456,      4,  11886,    662,
          363,  18768,    462,   1750,     41,     62,     10,   1895,
        52837,  11887,   3809,     62,  20206,   1052,  13871,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(padded_seq, df['label'], test_size=0.20, random_state=42, stratify=df['label'])
```

Model Training

```
model2.build(input_shape=(32, 100))
model2.summary()
```

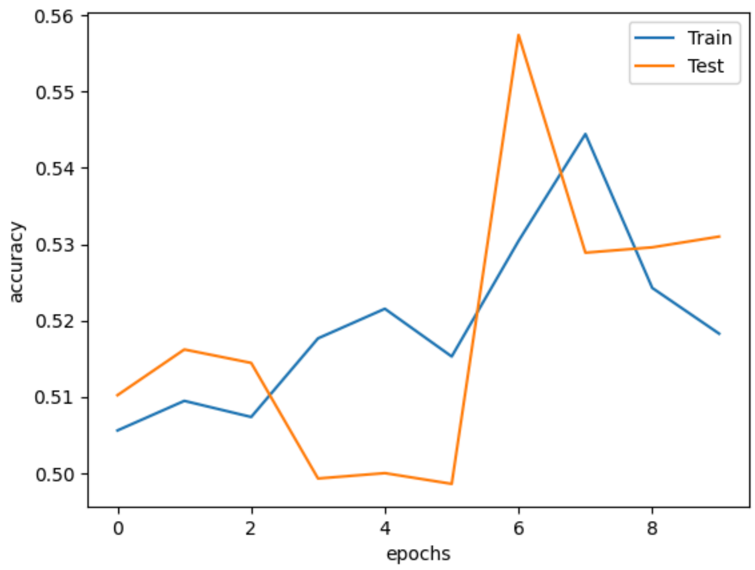
```
Model: "sequential_1"
```

```
# train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=256, validation_data=(x_test, y_test))
```

```
 embedding (Embedding)          │ (32, 100, 100)          │       1,000,100 │
 dropout (Dropout)              │ (32, 100, 100)          │               0 │
 lstm (LSTM)                    │ (32, 128)               │         117,248 │
 dropout_1 (Dropout)            │ (32, 128)               │               0 │
 dense_2 (Dense)                │ (32, 256)               │          33,024 │
 dense_3 (Dense)                │ (32, 1)                 │             257 │
 Total params: 1,150,629 (4.39 MB)
```

```
Epoch 1/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 156s 3s/step - accuracy: 0.5002 - loss: 0.6964 - val_accuracy: 0.5102 - val_loss: 0.6925
Epoch 2/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 191s 3s/step - accuracy: 0.5142 - loss: 0.6919 - val_accuracy: 0.5162 - val_loss: 0.6921
Epoch 3/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 150s 3s/step - accuracy: 0.5147 - loss: 0.6915 - val_accuracy: 0.5144 - val_loss: 0.6912
Epoch 4/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 200s 3s/step - accuracy: 0.5179 - loss: 0.6901 - val_accuracy: 0.4993 - val_loss: 0.6870
Epoch 5/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 201s 3s/step - accuracy: 0.5323 - loss: 0.7222 - val_accuracy: 0.5000 - val_loss: 0.6903
Epoch 6/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 197s 3s/step - accuracy: 0.5120 - loss: 0.6886 - val_accuracy: 0.4986 - val_loss: 0.6884
Epoch 7/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 149s 3s/step - accuracy: 0.5135 - loss: 0.6849 - val_accuracy: 0.5574 - val_loss: 0.6770
Epoch 8/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 152s 3s/step - accuracy: 0.5687 - loss: 0.6733 - val_accuracy: 0.5289 - val_loss: 0.6879
Epoch 9/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 195s 3s/step - accuracy: 0.5224 - loss: 0.6840 - val_accuracy: 0.5296 - val_loss: 0.6868
Epoch 10/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 202s 3s/step - accuracy: 0.5152 - loss: 0.6845 - val_accuracy: 0.5310 - val_loss: 0.6853
```

```
# visualize the results
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['Train', 'Test'])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['Train', 'Test'])
plt.show()
```


accuracy
epochs
Train
Test