

Sentiment Analysis of Tweets using Long short-term memory (LSTM)

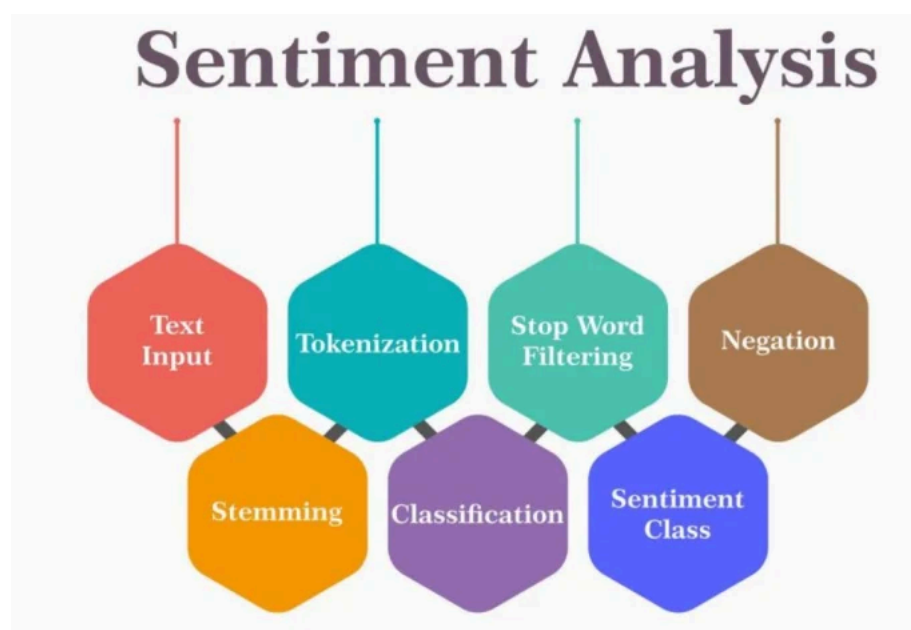
Introduction

```
1 Start coding or generate with AI.  
--NORMAL--
```

Text Classification is a process involved in Sentiment Analysis. It is classification of peoples opinion or expressions into different sentiments. Sentiments include *Positive*, *Neutral*, and *Negative*, *Review Ratings* and *Happy*, *Sad*. Sentiment Analysis can be done on different consumer centered industries to analyse people's opinion on a particular product or subject.

Sentiment Analysis

Sentiment Classification is a perfect problem in NLP for getting started in it. You can really learn a lot of concepts and techniques to master through doing project. Kaggle is a great place to learn and contribute your own ideas and creations. I learnt lot of things from other, now it's my turn to make document my project.



In this notebook, we will Classfy text from a dataset of milions tweets into positive or negative , by using Long short-term memory (LSTM).

Table of Content

- 1- IMPORTING LIBRARIES
- 2- LOADING DATASET
- 3- Dataset Preprocessing
- 4- Text Preprocessing
- 5- Build the Neural Network
- 6- Evaluate the model
- 7- Make Predicting Labels

✓ 1.Importing Libraries

We shall start by importing all the necessary libraries.

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4 from tensorflow.keras.preprocessing.text import Tokenizer
5 from tensorflow.keras.preprocessing.sequence import pad_sequences
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Embedding, LSTM, Dense
8 from sklearn.preprocessing import LabelEncoder
9 import matplotlib.pyplot as plt
10
11
12 import nltk
13 nltk.download('stopwords')
14 from nltk.corpus import stopwords
15 from nltk.stem import SnowballStemmer
16
17
```

```
[nltk_data] Downloading package stopwords to
[nltk_data] /Users/ghadeer/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

✓ 2.Load the dataset

In this notebook, I am using Sentiment-140 from Kaggle. It contains a labels data of 1.6 Million Tweets and I find it a good amount of data to train our model.

```
1 df = pd.read_csv('data1.csv',encoding = 'latin',header=None)
2 df
```

	0	1	2	3	4	5
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all...
...
1599995	4	2193601966	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	AmandaMarie1028	Just woke up. Having no school is the best fee...
1599996	4	2193601969	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	TheWDBboards	TheWDB.com - Very cool to hear old Walt interv...
1599997	4	2193601991	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	bpbabe	Are you ready for your MoJo Makeover? Ask me f...
1599998	4	2193602064	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	tinydiamondz	Happy 38th Birthday to my boo of alll time!!! ...
1599999	4	2193602129	Tue Jun 16 08:40:50 PDT 2009	NO_QUERY	RyanTrevMorris	happy #charitytuesday @theNSPCC @SparksCharity...

1600000 rows x 6 columns

You can see the columns are without any proper names. Lets rename them for our reference

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
#   Column  Non-Null Count  Dtype
---  -
0     0      1600000 non-null  int64
1     1      1600000 non-null  int64
2     2      1600000 non-null  object
3     3      1600000 non-null  object
4     4      1600000 non-null  object
5     5      1600000 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

```
1 df.isnull().sum()
```

```

0      0
1      0
2      0
3      0
4      0
5      0
dtype: int64

```

3. Dataset Preprocessing

```

1 #rename the Coulmns for our dataset
2 df.columns = ['sentiment', 'id', 'date', 'query', 'user_id', 'text']
3 df.head()

```

```

0      0  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY  _TheSpecialOne_  @switchfoot http://twitpic.com/2y1zI - Awww, t...
1      0  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY  scotthamilton  is upset that he can't update his Facebook by ...
2      0  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY  mattycus  @Kenichan I dived many times for the ball. Man...
3      0  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY  ElleCTF  my whole body feels itchy and like its on fire
4      0  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY  Karoli  @nationwideclass no, it's not behaving at all....

```

```
1 df = df.drop(['id', 'date', 'query', 'user_id'], axis=1)
```

Creating 'sentiment' Column:

This is an important preprocessing phase, we are deciding the outcome column (sentiment of review) based on the overall score. If the score is greater than 3, we take that as positive and if the value is less than 3 it is negative. If it is equal to 3, we take that as neutral sentiment.

```
1 df['sentiment'].value_counts()
```

```

0      800000
4      800000
Name: sentiment, dtype: int64

```

```

1 #this function is to maps the label 0 to "Negative" and the label 4 to "Positive"
2 lab_to_sentiment = {0:"Negative", 4:"Positive"}
3 def label_decoder(label):
4     return lab_to_sentiment[label]
5 df.sentiment = df.sentiment.apply(lambda x: label_decoder(x))
6 df.head()

```

```

0      Negative  @switchfoot http://twitpic.com/2y1zI - Awww, t...
1      Negative  is upset that he can't update his Facebook by ...
2      Negative  @Kenichan I dived many times for the ball. Man...
3      Negative  my whole body feels itchy and like its on fire
4      Negative  @nationwideclass no, it's not behaving at all....

```

```
1 df.iloc[1]
```

```

sentiment      Negative
text      is upset that he can't update his Facebook by ...
Name: 1, dtype: object

```

```
1 df.sentiment.value_counts()
```

```

Negative      800000
Positive      800000
Name: sentiment, dtype: int64

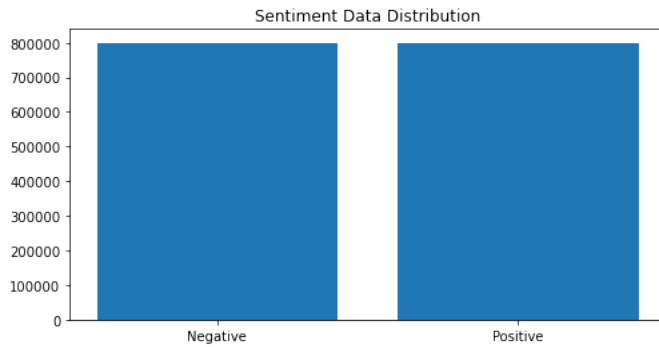
```

```

1 #Here to see the two words balance
2 val_count = df.sentiment.value_counts()
3
4 plt.figure(figsize=(8,4))
5 plt.bar(val_count.index, val_count.values)
6 plt.title("Sentiment Data Distribution")

```

Text(0.5, 1.0, 'Sentiment Data Distribution')



```
1 # creates random indexes to choose from dataframeimport random
2 random_idx_list = [random.randint(1,len(df.text)) for i in range(10)]
3 df.loc[random_idx_list,:].head(10)
```

	sentiment	text
1466075	Positive	forgot to watch Hatching Pete in Disney Channe...
1307538	Positive	had a great day! Relaxing facial, dinner @ ca...
512612	Negative	wants to go on ad adventure why won't anyone ...
977340	Positive	http://twitpic.com/5ezx7 - we are super ASIAN ...
250981	Negative	Longest shift of my life... I wanna go home
574815	Negative	@PhilMasteller Maybe I won't upgrade tonight. ...
947619	Positive	Is thinking a lot lately. College, ASL, Summer...
923829	Positive	rise and shine sunshines
1253838	Positive	@PennyAsh I was upset too at the time, but it'...
1385589	Positive	@rpisharody came home for the weekend..

Looks like we have a nasty data in text. Because in general we use lot of punctuations and other words without any contextual meaning. It have no value as feature to the model we are training. So we need to get rid of them.

4. Text Preprocessing

Tweet texts often consists of other user mentions, hyperlink texts, emoticons and punctuations. In order to use them for learning using a Language Model. We cannot permit those texts for training a model. So we have to clean the text data using various preprocessing and cleansing methods. Let's continue

1. Stemming/ Lemmatization

For grammatical reasons, documents are going to use different forms of a word, such as *write*, *writing* and *writes*. Additionally, there are families of derivationally related words with similar meanings. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Stemming usually refers to a process that chops off the ends of words in the hope of achieving goal correctly most of the time and often includes the removal of derivational affixes.

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to

Stemming	Lemmatization
adjustable → adjust	was → (to) be
formality → formaliti	better → good
formaliti → formal	meeting → meeting
airliner → airlin	

remove inflectional endings only and to return the base and dictionary form of a word

2. Hyperlinks and Mentions

Twitter is a social media platform where people can tag and mentions other people's ID and share videos and blogs from internet. So the tweets often contain lots of Hyperlinks and twitter mentions.

- Twitter User Mentions - Eg. @arunk7, @andrewng
- Hyperlinks - Eg. <https://keras.io>, <https://tensorflow.org>

3. Stopwords

Stopwords are commonly used words in English which have no contextual meaning in an sentence. So therefore we remove them before

```
> stopwords("english")
[1] "i"      "me"      "my"      "myself"  "we"
[6] "our"    "ours"    "ourselves" "you"     "your"
[11] "yours"  "yourself" "yourselves" "he"      "him"
[16] "his"    "himself" "she"      "her"     "hers"
[21] "herself" "it"      "its"      "itself"  "they"
[26] "them"   "their"   "theirs"   "themselves" "what"
[31] "which"  "who"     "whom"    "this"    "that"
[36] "these"  "those"   "am"      "is"      "are"
[41] "was"    "were"    "be"      "been"    "being"
[46] "have"   "has"     "had"     "having"  "do"
[51] "does"   "did"     "doing"   "would"   "should"
[56] "could"  "ought"   "i'm"     "you're"  "he's"
[61] "she's"  "it's"    "we're"   "they're" "i've"
[66] "you've" "we've"   "they've" "i'd"     "you'd"
[71] "he'd"   "she'd"   "we'd"    "they'd"  "i'll"
[76] "you'll" "he'll"   "she'll"  "we'll"   "they'll"
[81] "isn't"  "aren't"  "wasn't"  "weren't" "hasn't"
[86] "haven't" "hadn't"  "doesn't" "don't"   "didn't"
[91] "won't"  "wouldn't" "shan't"  "shouldn't" "can't"
[96] "cannot" "couldn't" "mustn't" "let's"   "that's"
[101] "who's"  "what's"  "here's"  "there's" "when's"
[106] "where's" "why's"   "how's"   "a"      "an"
```

classification. Some stopwords are...

That looks like a tedious process, isn't?. Don't worry there is always some library in Python to do almost any work. The world is great!!!

NLTK is a python library which got functions to perform text processing task for NLP.

```
1 stop_words = stopwords.words('english')
2 stemmer = SnowballStemmer('english')
3
4 text_cleaning_re = "@\S+|https?:\S+|http?:\S|^[A-Za-z0-9]+"

1 def preprocess(text, stem=False):
2     text = re.sub(text_cleaning_re, ' ', str(text).lower()).strip()
3     tokens = []
4     for token in text.split():
5         if token not in stop_words:
6             if stem:
7                 tokens.append(stemmer.stem(token))
8             else:
9                 tokens.append(token)
10    return " ".join(tokens)

1 df.text = df.text.apply(lambda x: preprocess(x))
```

It is clean and tidy now. Now let's see some word cloud visualizations of it.**

✓ Positive Words

```
1 from wordcloud import WordCloud
2
3 plt.figure(figsize = (20,20))
4 wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.sentiment == 'Positive'].text))
5 plt.imshow(wc , interpolation = 'bilinear')
6
```

[illegible]

```
1 plt.figure(figsize = (20,20))
2 wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.sentiment == 'Negative'].text))
3 plt.imshow(wc , interpolation = 'bilinear')
```

[illegible]

```
1 # 1. Label Encoding
2 le = LabelEncoder()
```



```

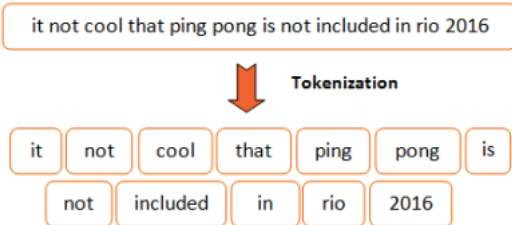
3 df['sentiment'] = le.fit_transform(df['sentiment'])
4

1 # 2. Split the dataset into training and testing sets
2 X = df['text']
3 y = df['sentiment']
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

✓ Tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens*, perhaps at the same time throwing away certain characters, such as punctuation. The process is called **Tokenization**.



`tokenizer` create tokens for every word in the data corpus and map them to a index using dictionary.

`word_index` contains the index for each word

`vocab_size` represents the total number of word in the data corpus

```

1 # Tokenize the text data
2 max_words = 10000
3 tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
4 tokenizer.fit_on_texts(X_train)
5

```

Now we got a `tokenizer` object, which can be used to covert any word into a Key in dictionary (number).

Since we are going to build a sequence model. We should feed in a sequence of numbers to it. And also we should ensure there is no variance in input shapes of sequences. It all should be of same lenght. But texts in tweets have different count of words in it. To avoid this, we seek a little help from `pad_sequence` to do our job. It will make all the sequence in one constant length `MAX_SEQUENCE_LENGTH`.

```

1 #sequence the text data
2 X_train_seq = tokenizer.texts_to_sequences(X_train)
3 X_test_seq = tokenizer.texts_to_sequences(X_test)
4

```

✓ Word Emddeding

In Language Model, words are represented in a way to intend more meaning and for learning the patterns and contextual meaning behind it.

Word Embedding is one of the popular representation of document vocabulary.It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

Basically, it's a feature vector representation of words which are used for other natural language processing applications.

```

1 #Word Emddeding
2 max_sequence_length = 50
3 X_train_padded = pad_sequences(X_train_seq, maxlen=max_sequence_length, padding='post', truncating='post')
4 X_test_padded = pad_sequences(X_test_seq, maxlen=max_sequence_length, padding='post', truncating='post')
5

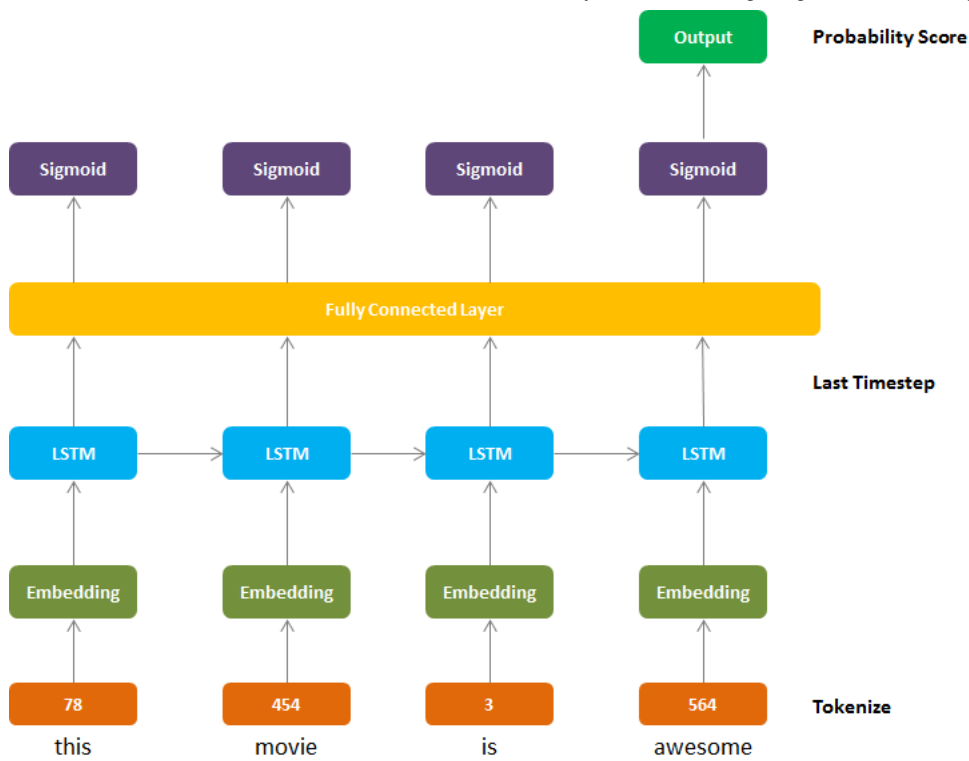
```

✓ 5.Model Training - LSTM

We are clear to build our Deep Learning model. While developing a DL model, we should keep in mind of key things like Model Architecture, Hyperparameter Tuning and Performance of the model.

As you can see in the word cloud, the some words are predominantly feature in both Positive and Negative tweets. This could be a problem if we are using a Machine Learning model like Naive Bayes, SVD, etc.. That's why we use **Sequence Models**.

Sequence Model



Recurrent Neural Networks can handle a sequence of data and learn a pattern of input sequence to give either sequence or scalar value as output. In our case, the Neural Network outputs a scalar value prediction.

For model architecture, we use

- 1) **Embedding Layer** - Generates Embedding Vector for each input sequence.
- 2) **Conv1D Layer** - Its using to convolve data into smaller feature vectors.
- 3) **LSTM** - Long Short Term Memory, its a variant of RNN which has memory state cell to learn the context of words which are at further along the text to carry contextual meaning rather than just neighbouring words as in case of RNN.
- 4) **Dense** - Fully Connected Layers for classification

```
1 # Build an LSTM model
2 model = Sequential()
3 model.add(Embedding(max_words, 32, input_length=max_sequence_length))
4 model.add(LSTM(64))
5 model.add(Dense(1, activation='sigmoid'))

1 #Compiling our Recurrent neural network
2
3 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
4 model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 50, 32)	320000
lstm_2 (LSTM)	(None, 64)	24832
dense_2 (Dense)	(None, 1)	65
Total params: 344897 (1.32 MB)		
Trainable params: 344897 (1.32 MB)		
Non-trainable params: 0 (0.00 Byte)		

```
1 history = model.fit(X_train_padded, y_train, validation_data=(X_test_padded, y_test), epochs=5, batch_size=32)
```

```
Epoch 1/5
40000/40000 [=====] - 825s 21ms/step - loss: 0.5462 - accuracy: 0.6943 - val_loss: 0.4602 - val
Epoch 2/5
40000/40000 [=====] - 902s 23ms/step - loss: 0.4548 - accuracy: 0.7840 - val_loss: 0.4531 - val
Epoch 3/5
40000/40000 [=====] - 900s 22ms/step - loss: 0.4418 - accuracy: 0.7915 - val_loss: 0.4536 - val
Epoch 4/5
40000/40000 [=====] - 520s 13ms/step - loss: 0.4324 - accuracy: 0.7971 - val_loss: 0.4512 - val
Epoch 5/5
```

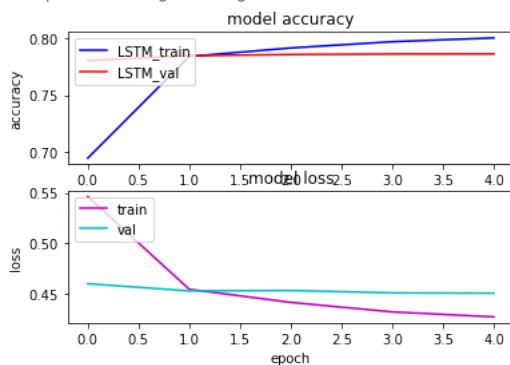

40000/40000 [=====] - 488s 12ms/step - loss: 0.4275 - accuracy: 0.8004 - val_loss: 0.4509 - val

8. Plot the Accuracy and Loss Graph

Here is the Learning Curve of loss and accuracy of the model on each epoch.

```
1 s, (at, al) = plt.subplots(2,1)
2 at.plot(history.history['accuracy'], c= 'b')
3 at.plot(history.history['val_accuracy'], c='r')
4 at.set_title('model accuracy')
5 at.set_ylabel('accuracy')
6 at.set_xlabel('epoch')
7 at.legend(['LSTM_train', 'LSTM_val'], loc='upper left')
8
9 al.plot(history.history['loss'], c='m')
10 al.plot(history.history['val_loss'], c='c')
11 al.set_title('model loss')
12 al.set_ylabel('loss')
13 al.set_xlabel('epoch')
14 al.legend(['train', 'val'], loc = 'upper left')
```

 <matplotlib.legend.Legend at 0x7fac61cc40a0>




```
1 #Save the model parameters
2 model.save("Sentiment.keras")
```

 /opt/anaconda3/lib/python3.9/site-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as saving_api.save_model(

8. Model Evaluation

Now that we have trained the model, we can evaluate its performance.


```
1 # Final evaluation of the model
2 loss, accuracy = model.evaluate(X_test_padded, y_test)
3 print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

 10000/10000 [=====] - 45s 5ms/step - loss: 0.4509 - accuracy: 0.7862
Test Loss: 0.4509, Test Accuracy: 0.7862

Here is the total accuracy is 78% which considering good

9. Make Predicting Labels

```
1 scores = model.predict(X_test_padded, verbose=1, batch_size=10000)
```

 32/32 [=====] - 17s 527ms/step

```
1
2 def decode_sentiment(score):
3     return "Positive" if score > 0.5 else "Negative"
4
5 for i in range(len(scores)):
6     sentiment = decode_sentiment(scores[i])
7     print("Sentence:", X_test.iloc[i])
8     print("Sentiment Score:", scores[i])
9     print("Predicted Sentiment:", sentiment)
```

```
10      print()
```