Movies_Review_Sentiment_Analysis in NLP

Import Required Libraries

In [1]:
```
import nltk
import csv
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
nltk.download("punkt")
nltk.download('wordnet')
import warnings
warnings.filterwarnings("ignore")
#stopword
nltk.download('stopwords')
# NN
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,Embedding, Flatten,Sim
# preprocessing
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing import sequence

```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Ajit\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Ajit\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Ajit\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

load movies review data

In [2]:
```
df=pd.read_csv("IMDB Dataset.csv")
```

In [3]:
```
1 df.head()
```

Out[3]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Perform EDA

In [4]:
```
1 df.shape
```

Out[4]:
```
(50000, 2)
```

In [5]:
```
1 df.isnull().sum()
```

Out[5]:
```
review       0
sentiment    0
dtype: int64
```

In [6]:
```
1 df.describe()
```

Out[6]:

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

In [7]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

Check the unique value¶

In [8]:
```
1 df['sentiment'].unique()
```

Out[8]:
```
array(['positive', 'negative'], dtype=object)
```

Count the sentiment

In [9]:
```
1 df['sentiment'].value_counts()
```

Out[9]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

In [10]:
```
1 df =df.iloc[:10000]
```

In [11]:
```
1 df['review'][1]
```

Out[11]:
```
'A wonderful little production. <br /><br />The filming technique is very una
ssuming- very old-time-BBC fashion and gives a comforting, and sometimes disc
omforting, sense of realism to the entire piece. <br /><br />The actors are e
xtremely well chosen- Michael Sheen not only "has got all the polari" but he
has all the voices down pat too! You can truly see the seamless editing guide
d by the references to Williams\' diary entries, not only is it well worth th
e watching but it is a terrificly written and performed piece. A masterful pr
oduction about one of the great master\'s of comedy and his life. <br /><br /
>The realism really comes home with the little things: the fantasy of the gua
rd which, rather than use the traditional \'dream\' techniques remains solid
then disappears. It plays on our knowledge and our senses, particularly with
the scenes concerning Orton and Halliwell and the sets (particularly of their
flat with Halliwell\'s murals decorating every surface) are terribly well don
e.'
```

In [12]:
```
1 df.duplicated().sum()
```

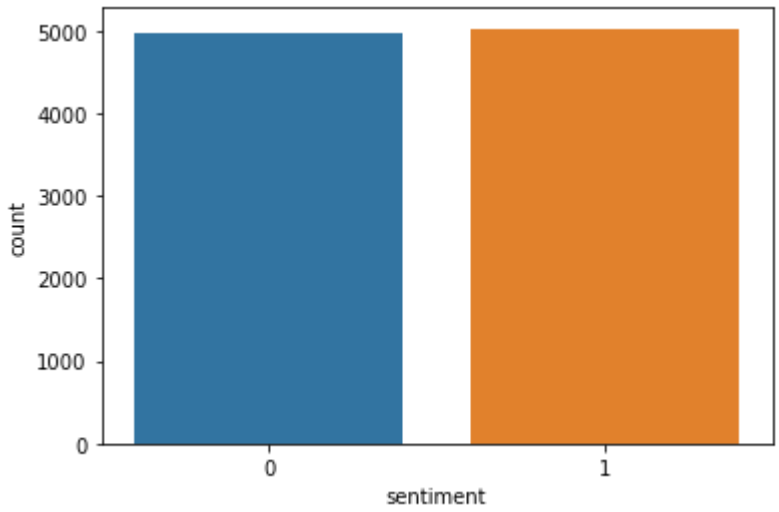
Out[12]:
```
17
```

Visualize the count of sentiment

In [60]:
```
1 import seaborn as sns
```

In [61]:
```
1 sns.countplot(df['sentiment'])
```

Out[61]: <AxesSubplot:xlabel='sentiment', ylabel='count'>



Apply LabelEncoding to make target feature into numerical¶

In [16]:
```
1 label=LabelEncoder()
2 df['sentiment']=label.fit_transform(df['sentiment'])
```

In [17]:
```
1 df.head()
```

Out[17]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1

Divide data into independent and dependent¶

In [18]:
```
1 x=df['review']
2 y=df['sentiment']
```

In [20]:
```
1 xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state
```

Tokenization

In [21]:
```
tok=Tokenizer()
tok.fit_on_texts(xtrain)
```

vocubulary

In [22]:
```
tok.word_index
voc=tok.word_index
```

In [23]:
```
print(voc)
```

```
{'the': 1, 'a': 2, 'and': 3, 'of': 4, 'to': 5, 'is': 6, 'br': 7, 'in': 8,
'it': 9, 'i': 10, 'this': 11, 'that': 12, 'was': 13, 'as': 14, 'with': 15,
'movie': 16, 'for': 17, 'but': 18, 'film': 19, 'on': 20, 'you': 21, 'not':
22, 'are': 23, 'his': 24, 'have': 25, 'be': 26, 'one': 27, 'he': 28, 'al
l': 29, 'at': 30, 'by': 31, 'an': 32, 'they': 33, 'so': 34, 'who': 35, 'fr
om': 36, 'like': 37, 'or': 38, 'just': 39, 'her': 40, 'about': 41, "it's":
42, 'out': 43, 'if': 44, 'has': 45, 'there': 46, 'what': 47, 'some': 48,
'good': 49, 'when': 50, 'more': 51, 'very': 52, 'up': 53, 'no': 54, 'my':
55, 'even': 56, 'she': 57, 'time': 58, 'would': 59, 'which': 60, 'story':
61, 'really': 62, 'only': 63, 'see': 64, 'their': 65, 'had': 66, 'well': 6
7, 'can': 68, 'me': 69, 'were': 70, 'much': 71, 'than': 72, 'been': 73, 'g
et': 74, 'because': 75, 'we': 76, 'great': 77, 'bad': 78, 'do': 79, 'wil
l': 80, 'first': 81, 'other': 82, 'into': 83, 'people': 84, 'also': 85, 'm
ost': 86, 'how': 87, 'him': 88, 'made': 89, "don't": 90, 'its': 91, 'mak
e': 92, 'way': 93, 'them': 94, 'then': 95, 'too': 96, 'movies': 97, 'any':
98, 'after': 99, 'could': 100, 'think': 101, 'characters': 102, 'films': 1
03, 'watch': 104, 'little': 105, 'never': 106, 'character': 107, 'being':
108, 'seen': 109, 'many': 110, 'two': 111, 'love': 112, 'did': 113, 'actin
g': 114, 'best': 115, 'life': 116, 'know': 117, 'plot': 118, 'show': 119,
```

In [24]:
```
xtrain.shape
```

Out[24]: (7000,)

In [25]:
```
xtest.shape
```

Out[25]: (3000,)

In [26]:
```
ytrain.shape
```

Out[26]: (7000,)

In [27]:
```
voclen=len(voc)
```

sequence

In [28]:
```
trainseq=tok.texts_to_sequences(xtrain)
```

In [29]:
```
print(trainseq)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [30]:
```
len(trainseq)
```

Out[30]: 7000

doc length

In [32]:
```
doclen=[]
for i in trainseq:
    doclen.append(len(i))
print(doclen)
```

```
[658, 858, 178, 261, 146, 215, 349, 140, 116, 626, 84, 103, 788, 536, 147,
146, 191, 635, 108, 492, 112, 80, 119, 149, 142, 197, 828, 123, 233, 127,
51, 108, 58, 49, 174, 89, 103, 282, 322, 58, 236, 142, 129, 226, 393, 150,
93, 151, 107, 142, 860, 212, 187, 139, 153, 232, 95, 236, 103, 113, 328, 7
01, 163, 617, 134, 232, 211, 140, 492, 323, 162, 96, 184, 215, 135, 153, 2
38, 264, 139, 163, 128, 368, 170, 18, 241, 171, 143, 871, 123, 158, 58, 19
4, 111, 91, 315, 142, 73, 311, 641, 60, 988, 267, 727, 137, 80, 322, 178,
310, 587, 49, 422, 584, 196, 55, 199, 304, 260, 296, 223, 372, 112, 107, 1
36, 189, 118, 179, 93, 160, 153, 390, 248, 169, 345, 128, 138, 377, 90, 17
9, 267, 88, 283, 255, 131, 77, 316, 63, 110, 127, 276, 126, 127, 361, 254,
252, 116, 153, 166, 147, 457, 120, 404, 243, 650, 28, 665, 157, 174, 383,
251, 544, 310, 262, 115, 82, 634, 503, 35, 113, 498, 123, 124, 279, 124, 1
04, 164, 135, 170, 118, 70, 76, 167, 50, 450, 98, 137, 401, 469, 96, 150,
281, 367, 121, 110, 143, 291, 844, 108, 105, 479, 61, 217, 217, 137, 123,
35, 111, 160, 173, 147, 162, 96, 226, 364, 160, 201, 166, 763, 837, 103, 3
46, 137, 352, 373, 298, 206, 104, 434, 164, 239, 169, 354, 170, 133, 53, 1
81, 181, 122, 80, 486, 138, 138, 234, 774, 345, 62, 136, 156, 614, 651, 19
4, 126, 164, 312, 196, 132, 243, 173, 126, 133, 143, 51, 432, 147, 58, 21
4, 149, 56, 177, 278, 551, 137, 185, 200, 122, 162, 648, 310, 675, 286, 16
```

In [33]:
```
max(doclen)
```

Out[33]: 1850

In [34]:
```
np.quantile(doclen,1)
```

Out[34]: 1850

In [35]:
```
np.quantile(doclen,0.99)
```

Out[35]: 893.0100000000002

In [36]:
```
max_length=np.quantile(doclen,1)
```

Padding.

In [37]:
```
trainmatrix = sequence.pad_sequences(trainseq,maxlen = max_length)
trainmatrix
```

Out[37]:
```
array([[   0,    0,    0, ...,    7,  701,  151],
       [   0,    0,    0, ...,   41,   11,  119],
       [   0,    0,    0, ...,   39,  104,    9],
       ...,
       [   0,    0,    0, ...,    3,  269, 6845],
       [   0,    0,    0, ..., 3647,  464,  151],
       [   0,    0,    0, ...,   67,  223, 7112]])
```

In [38]:
```
testseq = tok.texts_to_sequences(xtest)
```

padding on test data

In [39]:
```
testmatrix = sequence.pad_sequences(testseq,maxlen = max_length)
testmatrix
```

Out[39]:
```
array([[    0,     0,     0, ...,  1059,   313,   233],
       [    0,     0,     0, ...,    80,  2114,   173],
       [    0,     0,     0, ...,  1405,   643,    21],
       ...,
       [    0,     0,     0, ...,   256,     9,  1631],
       [    0,     0,     0, ...,  6723,     3, 20869],
       [    0,     0,     0, ...,   188,  8399,  4999]])
```

In [40]:
```
testmatrix.shape
```

Out[40]: (3000, 1850)

In [106]:
```
trainmatrix.shape
```

Out[106]:
```
(7000, 1850)
```

NN (Flatten)

In [42]:
```
model = Sequential()
model.add(Embedding(input_dim=voclen+1,
           output_dim=100,
           input_length=max_length,
           mask_zero=True))
model.add(Flatten())
model.add(Dense(16,activation="relu"))
model.add(Dense(8,activation="relu"))
model.add(Dense(1,activation="sigmoid"))
```

In [43]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 1850, 100)         5085800

 flatten (Flatten)           (None, 185000)            0

 dense (Dense)               (None, 16)                2960016

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 8,045,961
Trainable params: 8,045,961
Non-trainable params: 0
_________________________________________________________________
```

In [44]:
```
model.compile(optimizer = "adam", loss = "binary_crossentropy",metrics ="
model.fit(trainmatrix,ytrain,epochs=10,batch_size=32)
```

```
Epoch 1/10
219/219 [==============================] - 51s 216ms/step - loss: 0.6945 - ac
curacy: 0.5553
Epoch 2/10
219/219 [==============================] - 49s 224ms/step - loss: 0.3414 - ac
curacy: 0.8683
Epoch 3/10
219/219 [==============================] - 47s 213ms/step - loss: 0.0718 - ac
curacy: 0.9809
Epoch 4/10
219/219 [==============================] - 49s 222ms/step - loss: 0.0152 - ac
curacy: 0.9989
Epoch 5/10
219/219 [==============================] - 48s 221ms/step - loss: 0.0059 - ac
curacy: 0.9997
Epoch 6/10
219/219 [==============================] - 49s 223ms/step - loss: 0.0036 - ac
curacy: 0.9997
Epoch 7/10
219/219 [==============================] - 46s 209ms/step - loss: 0.0031 - ac
curacy: 0.9997
Epoch 8/10
219/219 [==============================] - 46s 208ms/step - loss: 0.0021 - ac
curacy: 0.9997
Epoch 9/10
219/219 [==============================] - 47s 214ms/step - loss: 0.0017 - ac
curacy: 0.9997
Epoch 10/10
219/219 [==============================] - 45s 204ms/step - loss: 0.0014 - ac
curacy: 0.9997
```

Out[44]:
```
<keras.callbacks.History at 0x18632a73970>
```

Prediction

In [45]:
```
ypred = model.predict(testmatrix)
ypred
```

```
94/94 [==============================] - 2s 20ms/step
```

Out[45]:
```
array([[5.2143552e-04],
       [3.4554247e-02],
       [9.9954540e-01],
       ...,
       [9.9069917e-01],
       [9.9797583e-01],
       [9.9522579e-01]], dtype=float32)
```

In [46]:
```
ypred = np.where(ypred>=0.5,1,0)
ypred
```

Out[46]:
```
array([[0],
       [0],
       [1],
       ...,
       [1],
       [1],
       [1]])
```

In [47]:
```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.88      0.88      1478
           1       0.88      0.89      0.89      1522

    accuracy                           0.88      3000
   macro avg       0.88      0.88      0.88      3000
weighted avg       0.88      0.88      0.88      3000
```

SimpleRNN

In [48]:
```
model=Sequential()
model.add(Embedding(input_dim=voclen+1, #input will be all the tokens + 1
                    output_dim=100,
                    input_length=max_length,
                    mask_zero=True)) #mask zero will skip all the zeros
model.add(SimpleRNN(32))
model.add(Dense(16,activation="relu"))
model.add(Dense(8,activation="relu"))
model.add(Dense(1,activation="sigmoid"))
```

In [49]:
```
1 model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 1850, 100)         5085800

 simple_rnn (SimpleRNN)      (None, 32)                4256

 dense_3 (Dense)             (None, 16)                528

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 1)                 9

=================================================================
Total params: 5,090,729
Trainable params: 5,090,729
Non-trainable params: 0
_________________________________________________________________
```

In [50]:
```
model.compile(optimizer="adam",loss="binary_crossentropy",metrics="accura
model.fit(trainmatrix,ytrain,epochs=20,batch_size=32)
```

```
Epoch 1/20
219/219 [==============================] - 297s 1s/step - loss: 0.6616 - accu
racy: 0.5946
Epoch 2/20
219/219 [==============================] - 300s 1s/step - loss: 0.3870 - accu
racy: 0.8319
Epoch 3/20
219/219 [==============================] - 302s 1s/step - loss: 0.1151 - accu
racy: 0.9589
Epoch 4/20
219/219 [==============================] - 292s 1s/step - loss: 0.0187 - accu
racy: 0.9964
Epoch 5/20
219/219 [==============================] - 292s 1s/step - loss: 0.0036 - accu
racy: 0.9994
Epoch 6/20
219/219 [==============================] - 287s 1s/step - loss: 5.9743e-04 -
accuracy: 1.0000
Epoch 7/20
219/219 [==============================] - 288s 1s/step - loss: 3.2388e-04 -
accuracy: 1.0000
Epoch 8/20
219/219 [==============================] - 289s 1s/step - loss: 2.1022e-04 -
accuracy: 1.0000
Epoch 9/20
219/219 [==============================] - 288s 1s/step - loss: 1.4781e-04 -
accuracy: 1.0000
Epoch 10/20
219/219 [==============================] - 288s 1s/step - loss: 1.0940e-04 -
accuracy: 1.0000
Epoch 11/20
219/219 [==============================] - 288s 1s/step - loss: 8.3586e-05 -
accuracy: 1.0000
Epoch 12/20
219/219 [==============================] - 287s 1s/step - loss: 6.5112e-05 -
accuracy: 1.0000
Epoch 13/20
219/219 [==============================] - 287s 1s/step - loss: 5.1855e-05 -
accuracy: 1.0000
Epoch 14/20
219/219 [==============================] - 288s 1s/step - loss: 4.1835e-05 -
accuracy: 1.0000
Epoch 15/20
219/219 [==============================] - 293s 1s/step - loss: 3.4103e-05 -
accuracy: 1.0000
Epoch 16/20
219/219 [==============================] - 255s 1s/step - loss: 2.8105e-05 -
accuracy: 1.0000
Epoch 17/20
219/219 [==============================] - 249s 1s/step - loss: 2.3315e-05 -
accuracy: 1.0000
Epoch 18/20
219/219 [==============================] - 246s 1s/step - loss: 1.9505e-05 -
accuracy: 1.0000
Epoch 19/20
219/219 [==============================] - 279s 1s/step - loss: 1.6364e-05 -
accuracy: 1.0000
```

```
Epoch 20/20
219/219 [==============================] - 270s 1s/step - loss: 1.3828e-05 -
accuracy: 1.0000
```

Out[50]:
```
<keras.callbacks.History at 0x18634d6d1c0>
```

Prediction

In [51]:
```
ypred=model.predict(testmatrix)
ypred
```

```
94/94 [==============================] - 18s 182ms/step
```

Out[51]:
```
array([[1.5789201e-05],
       [9.7502309e-01],
       [9.9963069e-01],
       ...,
       [9.9999404e-01],
       [9.9999863e-01],
       [9.9995518e-01]], dtype=float32)
```

In [52]:
```
ypred=np.where(ypred>=0.5,1,0)
ypred
```

Out[52]:
```
array([[0],
       [1],
       [1],
       ...,
       [1],
       [1],
       [1]])
```

In [53]:
```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.77      0.79      1478
           1       0.79      0.82      0.80      1522

    accuracy                           0.80      3000
   macro avg       0.80      0.80      0.80      3000
weighted avg       0.80      0.80      0.80      3000
```

Bidirectional RNN

In [54]:
```
model1=Sequential()
model1.add(Embedding(input_dim=voclen+1, #input will be all the tokens + 1
                     output_dim=100,
                     input_length=max_length,
                     mask_zero=True)) #mask zero will skip all the zeros
model1.add(Bidirectional(SimpleRNN(32)))
model1.add(Dense(16,activation="relu"))
model1.add(Dense(8,activation="relu"))
model1.add(Dense(1,activation="sigmoid"))
```

In [109]:
```
model1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 1850, 100)         5085800

 bidirectional (Bidirectiona  (None, 64)               8512
 l)

 dense_6 (Dense)             (None, 16)                1040

 dense_7 (Dense)             (None, 8)                 136

 dense_8 (Dense)             (None, 1)                 9

=================================================================
Total params: 5,095,497
Trainable params: 5,095,497
Non-trainable params: 0
_________________________________________________________________
```

In [55]:
```
model1.compile(optimizer="adam",loss="binary_crossentropy",metrics="accura
model1.fit(trainmatrix,ytrain,epochs=5,batch_size=64)
```

```
Epoch 1/5
110/110 [==============================] - 323s 3s/step - loss: 0.6848 - accu
racy: 0.5473
Epoch 2/5
110/110 [==============================] - 324s 3s/step - loss: 0.4886 - accu
racy: 0.7904
Epoch 3/5
110/110 [==============================] - 281s 3s/step - loss: 0.1065 - accu
racy: 0.9653
Epoch 4/5
110/110 [==============================] - 278s 3s/step - loss: 0.0098 - accu
racy: 0.9989
Epoch 5/5
110/110 [==============================] - 270s 2s/step - loss: 0.0011 - accu
racy: 1.0000
```

Out[55]:
```
<keras.callbacks.History at 0x1863bffee50>
```

In [56]:
```
ypred=model1.predict(testmatrix)
ypred
```

```
94/94 [==============================] - 21s 214ms/step
```

Out[56]:
```
array([[1.1229620e-04],
       [9.9213314e-01],
       [9.9947816e-01],
       ...,
       [9.8659348e-01],
       [9.6974248e-01],
       [9.9304515e-01]], dtype=float32)
```

In [57]:
```
ypred=np.where(ypred>=0.5,1,0)
ypred
```

Out[57]:
```
array([[0],
       [1],
       [1],
       ...,
       [1],
       [1],
       [1]])
```

In [58]:
```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.81      1478
           1       0.83      0.78      0.81      1522

    accuracy                           0.81      3000
   macro avg       0.81      0.81      0.81      3000
weighted avg       0.81      0.81      0.81      3000
```

LSTM

In [113]:
```
model4=Sequential()
model4.add(Embedding(input_dim=voclen+1, #input will be all the tokens + 1
                     output_dim=100,
                     input_length=max_length,
                     mask_zero=True)) #mask zero will skip all the zeros
model4.add(LSTM(32))
model4.add(Dense(16,activation="relu"))
model4.add(Dropout(0.2))
model4.add(Dense(8,activation="relu"))
model4.add(Dense(1,activation="sigmoid"))
```

In [114]:
```
1 model4.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 1850, 100)         5085800

 lstm_2 (LSTM)               (None, 32)                17024

 dense_18 (Dense)            (None, 16)                528

 dropout_3 (Dropout)         (None, 16)                0

 dense_19 (Dense)            (None, 8)                 136

 dense_20 (Dense)            (None, 1)                 9

=================================================================
Total params: 5,103,497
Trainable params: 5,103,497
Non-trainable params: 0
_________________________________________________________________
```

In [63]:
```
1 model4.compile(optimizer="adam",loss="binary_crossentropy",metrics="accura
2 model4.fit(trainmatrix,ytrain,epochs=5,batch_size=64)
```

```
Epoch 1/5
110/110 [==============================] - 250s 2s/step - loss: 0.6273 - accu
racy: 0.6620
Epoch 2/5
110/110 [==============================] - 237s 2s/step - loss: 0.4294 - accu
racy: 0.8363
Epoch 3/5
110/110 [==============================] - 238s 2s/step - loss: 0.1814 - accu
racy: 0.9401
Epoch 4/5
110/110 [==============================] - 239s 2s/step - loss: 0.0643 - accu
racy: 0.9807
Epoch 5/5
110/110 [==============================] - 238s 2s/step - loss: 0.0266 - accu
racy: 0.9939
```

Out[63]:
```
<keras.callbacks.History at 0x1864319f070>
```

In [64]:
```
ypred=model4.predict(testmatrix)
ypred
```

```
94/94 [==============================] - 40s 389ms/step
```

Out[64]:
```
array([[2.7912389e-04],
       [2.6113407e-03],
       [9.9967343e-01],
       ...,
       [9.9993855e-01],
       [9.9907529e-01],
       [9.9978328e-01]], dtype=float32)
```

In [65]:
```
ypred=np.where(ypred>=0.5,1,0)
ypred
```

Out[65]:
```
array([[0],
       [0],
       [1],
       ...,
       [1],
       [1],
       [1]])
```

In [66]:
```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.85      0.84      1478
           1       0.85      0.84      0.84      1522

    accuracy                           0.84      3000
   macro avg       0.84      0.84      0.84      3000
weighted avg       0.84      0.84      0.84      3000
```

GRU

In [67]:
```
model6=Sequential()
model6.add(Embedding(input_dim=voclen+1, #input will be all the tokens + 1
                     output_dim=100,
                     input_length=max_length,
                     mask_zero=True)) #mask zero will skip all the zeros
model6.add(GRU(32))
model6.add(Dense(16,activation="relu"))
model6.add(Dropout(0.2))
model6.add(Dense(8,activation="relu"))
model6.add(Dense(1,activation="sigmoid"))
```

In [68]:
```
model6.compile(optimizer="adam",loss="binary_crossentropy",metrics="accura
model6.fit(trainmatrix,ytrain,epochs=5,batch_size=32)
```

```
Epoch 1/5
219/219 [==============================] - 337s 1s/step - loss: 0.5841 - accu
racy: 0.6733
Epoch 2/5
219/219 [==============================] - 316s 1s/step - loss: 0.2756 - accu
racy: 0.8934
Epoch 3/5
219/219 [==============================] - 316s 1s/step - loss: 0.1268 - accu
racy: 0.9593
Epoch 4/5
219/219 [==============================] - 317s 1s/step - loss: 0.0349 - accu
racy: 0.9904
Epoch 5/5
219/219 [==============================] - 319s 1s/step - loss: 0.0149 - accu
racy: 0.9964
```

Out[68]:
```
<keras.callbacks.History at 0x1864b994b80>
```

In [69]:
```
ypred=model6.predict(testmatrix)
ypred
```

```
94/94 [==============================] - 34s 329ms/step
```

Out[69]:
```
array([[2.2199813e-05],
       [5.6832740e-03],
       [9.9913043e-01],
       ...,
       [9.9995571e-01],
       [9.9405134e-01],
       [9.9990505e-01]], dtype=float32)
```

In [70]:
```
ypred=np.where(ypred>=0.5,1,0)
ypred
```

Out[70]:
```
array([[0],
       [0],
       [1],
       ...,
       [1],
       [1],
       [1]])
```

In [71]:
```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.77      0.82      1478
           1       0.80      0.89      0.84      1522

    accuracy                           0.83      3000
   macro avg       0.83      0.83      0.83      3000
weighted avg       0.83      0.83      0.83      3000
```

In [82]:
```
from nltk.stem import PorterStemmer
import re
```

Remove all special and numeric character from data and also remove stopwords and apply stemming

In [83]:
```
ps = PorterStemmer()
corpus = []

for i in range(len(x)):
    print(i)
    review = re.sub("[^a-zA-Z]"," ",x[i])
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if word not in set(stopwor
    review = " ".join(review)
    corpus.append(review)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

In [84]:
```
1 corpus
```

Out[84]:
```
['one review mention watch oz episod hook right exactli happen br br first
thing struck oz brutal unflinch scene violenc set right word go trust show
faint heart timid show pull punch regard drug sex violenc hardcor classic
use word br br call oz nicknam given oswald maximum secur state penitentar
i focus mainli emerald citi experiment section prison cell glass front fac
e inward privaci high agenda em citi home mani aryan muslim gangsta latino
christian italian irish scuffl death stare dodgi deal shadi agreement neve
r far away br br would say main appeal show due fact goe show dare forget
pretti pictur paint mainstream audienc forget charm forget romanc oz mess
around first episod ever saw struck nasti surreal say readi watch develop
tast oz got accustom high level graphic violenc violenc injustic crook gua
rd sold nickel inmat kill order get away well manner middl class inmat tur
n prison bitch due lack street skill prison experi watch oz may becom comf
ort uncomfort view that get touch darker side',
 'wonder littl product br br film techniqu unassum old time bbc fashion gi
ve comfort sometim discomfort sens realism entir piec br br actor extrem w
ell chosen michael sheen got polari voic pat truli see seamless edit guid
refer william diari entri well worth watch terrificli written perform piec
master product one great master comedi life br br realism realli come home
```

Apply TfidfVectorizer to make text data into vectors

In [85]:
```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 cv=TfidfVectorizer(max_features=5000)
3 x=cv.fit_transform(corpus).toarray()
```

In [86]:
```
1 x.shape
```

Out[86]:
```
(10000, 5000)
```

Split data into train and test

In [87]:
```
1 xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state
```

In [88]:
```
1 xtrain.shape,xtest.shape,ytrain.shape,ytest.shape
```

Out[88]:
```
((8000, 5000), (2000, 5000), (8000,), (2000,))
```

In [100]:
```
1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn.metrics import accuracy_score, classification_report, confus
3 import pickle
```

Define Naive-bayes model

In [99]:
```
mnb=MultinomialNB()
mnb.fit(xtrain,ytrain)
```

Out[99]:
```
MultinomialNB()
```

Test model using test data

In [91]:
```
pred=mnb.predict(xtest)
```

Check Accuracy_score,confusion_matrix and classification_report

In [95]:
```
print(accuracy_score(ytest,pred))
print(confusion_matrix(ytest,pred))
print(classification_report(ytest,pred))
```

```
0.85
[[828 174]
 [126 872]]
              precision    recall  f1-score   support

           0       0.87      0.83      0.85      1002
           1       0.83      0.87      0.85       998

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000
```

Difference between Actual and Predicted data

In [96]:
```
pd.DataFrame(np.c_[ytest,pred],columns=["Actual","Predict"])
```

Out[96]:

	Actual	Predict
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
1995	1	0
1996	1	1
1997	1	1
1998	1	1
1999	1	1

2000 rows × 2 columns

Save my trained naive-bayes model and TfidfVectorizer

In [101]:
```
pickle.dump(cv,open("count-Vectorizer.pkl","wb"))
pickle.dump(mnb,open("Movies_Review_Classification.pkl","wb"))   #1: pos,
```

Load my naive-bayes model and TfidfVectorizer

In [102]:
```
save_cv=pickle.load(open('count-Vectorizer.pkl','rb'))
model=pickle.load(open('movies_Review_classification.pkl','rb'))
```

define my function to test model

In [103]:
```
def test_model(sentence):
    sen=save_cv.transform([sentence]).toarray()
    res=model.predict(sen)[0]
    if res==1:
        return 'Positive review'
    else:
        return 'Negative review'
```

Test first positive review and check that what does model predict and it predicted correct

In [104]:
```
sen='This is the wonderful movie of my life'
res=test_model(sen)
print(res)
```

```
Positive review
```

Test second negative review and check that what does model predict and it predicted correct

In [105]:
```
sen='This is the worst movie i have ever seen in my life'
res=test_model(sen)
print(res)
```

```
Negative review
```

In [ ]:
```

```