In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\MSI\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
```

VADER's SentimentIntensityAnalyzer() tokes in a string and returns a dictionary of scores in each of four categories:

- Negative
- Neutral
- Positive Compound (computed by normalizing the scores above)

In [3]:

```
1  a = 'This was a good movie'
2  b = 'The was the best, most awesome movie EVER MADE!!!'
3  c = 'This was the worst film to ever disgrace the screen.'
4
5  print(sid.polarity_scores(a))
6  print(sid.polarity_scores(b))
7  print(sid.polarity_scores(c))
```

```
{'neg': 0.0, 'neu': 0.508, 'pos': 0.492, 'compound': 0.4404}
{'neg': 0.0, 'neu': 0.425, 'pos': 0.575, 'compound': 0.8877}
{'neg': 0.477, 'neu': 0.523, 'pos': 0.0, 'compound': -0.8074}
```

Use VADER to analyze Amazon Reviews

In [4]:

```
data = pd.read_csv('IMDB Dataset.csv')
data.head()
```

Out[4]:

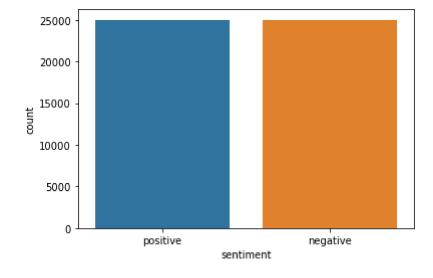
	review	sentiment
0	One of the other reviewers has mentioned that	positive
1	A wonderful little production. The	positive
2	I thought this was a wonderful way to spend ti	positive
3	Basically there's a family where a little boy	negative
4	Petter Mattei's "Love in the Time of Money" is	positive

In [5]:

```
1 sns.countplot(x='sentiment', data=data)
```

Out[5]:

<AxesSubplot:xlabel='sentiment', ylabel='count'>



In [6]:

```
data['scores'] = data['review'].apply(lambda review: sid.polarity_scores(review))
```

In [7]:

```
1 data.head()
```

Out[7]:

	review	sentiment	scores
0	One of the other reviewers has mentioned that	positive	{'neg': 0.203, 'neu': 0.748, 'pos': 0.048, 'co
1	A wonderful little production. The	positive	{'neg': 0.053, 'neu': 0.776, 'pos': 0.172, 'co
2	I thought this was a wonderful way to spend ti	positive	{'neg': 0.094, 'neu': 0.714, 'pos': 0.192, 'co
3	Basically there's a family where a little boy	negative	{'neg': 0.138, 'neu': 0.797, 'pos': 0.065, 'co
4	Petter Mattei's "Love in the Time of Money" is	positive	{'neg': 0.052, 'neu': 0.801, 'pos': 0.147, 'co

In [8]:

```
data['compound'] = data['scores'].apply(lambda score_dict: score_dict['compound'])
data['comp_score'] = data['compound'].apply(lambda c: 'positive' if c >= 0 else 'negation')
```

In [9]:

```
1 data.head()
```

Out[9]:

	review	sentiment	scores	compound	comp_score
0	One of the other reviewers has mentioned that	positive	{'neg': 0.203, 'neu': 0.748, 'pos': 0.048, 'co	-0.9951	negative
1	A wonderful little production. /> br />The	positive	{'neg': 0.053, 'neu': 0.776, 'pos': 0.172, 'co	0.9641	positive
2	I thought this was a wonderful way to spend ti	positive	{'neg': 0.094, 'neu': 0.714, 'pos': 0.192, 'co	0.9605	positive
3	Basically there's a family where a little boy	negative	{'neg': 0.138, 'neu': 0.797, 'pos': 0.065, 'co	-0.9213	negative
4	Petter Mattei's "Love in the Time of Money" is	positive	{'neg': 0.052, 'neu': 0.801, 'pos': 0.147, 'co	0.9744	positive

In [10]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
accuracy_score(data['sentiment'], data['comp_score'])
```

Out[10]:

0.69556

7611 7612

```
In [11]:
  1 | confusion_matrix(data['sentiment'], data['comp_score'])
Out[11]:
array([[13364, 11636],
        [ 3586, 21414]], dtype=int64)
In [12]:
  1 pd.DataFrame(classification_report(data['sentiment'], data['comp_score'], output_dict=1
Out[12]:
              negative
                            positive accuracy
                                                          weighted avg
                                                macro avg
              0.788437
                           0.647927
                                      0.69556
                                                  0.718182
                                                               0.718182
 precision
              0.534560
                           0.856560
                                      0.69556
                                                  0.695560
                                                               0.695560
    recall
  f1-score
              0.637139
                           0.737778
                                      0.69556
                                                  0.687459
                                                               0.687459
  support 25000.000000 25000.000000
                                      0.69556
                                              50000.000000
                                                          50000.000000
In [13]:
    data = pd.read_csv('train.csv')
In [14]:
    print(data.target)
0
1
2
3
7608
7609
7610
```

Classic Machine Learning Models

Name: target, Length: 7613, dtype: int64

In [15]:

```
import spacy
   import pandas as pd
 2
   import numpy as np
 5
   from sklearn.model selection import train test split
   from sklearn.feature_extraction.text import TfidfVectorizer
 7
   from sklearn.linear_model import LogisticRegression
 8
9
   from sklearn.svm import LinearSVC
   from sklearn.naive bayes import MultinomialNB
10
11
12
   from sklearn.metrics import accuracy score, classification report
13
14
   def train_model(model_name, model, X_train, X_test, y_train, y_test):
       print(f'BEGIN. {model_name.upper()}.....')
15
16
       model.fit(X train, y train)
       y pred = model.predict(X test)
17
       y train pred = model.predict(X train)
18
       print(f'TESTING DATA----> {model_name.upper()}: \t\t{accuracy_score(y_test, y_pred)}
19
       print(f'TRAINING DATA---> {model_name.upper()}: \t\t{accuracy_score(y_train, y_train)}
20
21
       print(classification_report(y_test, y_pred))
22
       print(f'END. {model name.upper()}')
23
       print('========')
24
       return y_pred
25
   data = pd.read_csv('train.csv')
26
```

In [16]:

```
print('========Splitting the data========')
   X = data.text
   y = data.target
   print(f'Data shape: {data.shape}')
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
 7
    print(f'X_Train shape: {X_train.shape}, y_train shape: {y_train.shape}')
    print(f'X_Test shape: {X_test.shape}, y_test shape: {y_test.shape}')
 8
 9
10
   print('\n========Message Preprocessing========')
11
   vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_df=0.9, min_df=2, stop_words='engl
    X_train_vect = vectorizer.fit_transform(X_train.astype('U'))
13
   X_test_vect = vectorizer.transform(X_test)
14
15
    print('Training and testing data shape after pre-processing:')
16
    print(f'X_Train shape: {X_train_vect.shape}, y_train shape: {y_train.shape}')
    print(f'X Test shape: {X test vect.shape}, y test shape: {y test.shape}')
17
18
    print('\n========Model Building=========')
19
20
   lr_model = LogisticRegression()
    lr_y_pred = train_model('Logistic Regression', lr_model, X_train_vect, X_test_vect, y_1
22
23
    svm model = LinearSVC()
24
    svm_y_pred = train_model('Support Vector Machine', svm_model, X_train_vect, X_test_vect
25
26
   nb_model = MultinomialNB()
27
   nb_y_pred = train_model('Naive Bayes', nb_model, X_train_vect, X_test_vect, y_train, y_
28
========Splitting the data========
Data shape: (7613, 5)
X_Train shape: (5329,), y_train shape: (5329,)
X_Test shape: (2284,), y_test shape: (2284,)
=======Message Preprocessing=======
Training and testing data shape after pre-processing:
X_Train shape: (5329, 8557), y_train shape: (5329,)
X_Test shape: (2284, 8557), y_test shape: (2284,)
=======Model Building========
BEGIN. LOGISTIC REGRESSION.....
TESTING DATA----> LOGISTIC REGRESSION:
                                               80.12%
TRAINING DATA---> LOGISTIC REGRESSION:
                                               88.40%
             precision
                       recall f1-score support
          0
                  0.80
                            0.88
                                      0.84
                                                1318
                  0.81
                            0.69
                                      0.75
                                                 966
                                      0.80
                                                2284
   accuracy
                            0.79
                                      0.79
  macro avg
                  0.80
                                                2284
                  0.80
                            0.80
                                      0.80
                                                2284
weighted avg
END. LOGISTIC REGRESSION
BEGIN. SUPPORT VECTOR MACHINE.....
TESTING DATA----> SUPPORT VECTOR MACHINE:
                                                       77.76%
TRAINING DATA---> SUPPORT VECTOR MACHINE:
                                                       95.91%
             precision
                          recall f1-score
                                            support
```

		vadei_senti	mont_analysis	oup
0.80	0.82	0.81	1318	
0.75	0.72	0.73	966	
		0.78	2284	
0.77	0.77	0.77	2284	
0.78	0.78	0.78	2284	
VECTOR MACH	HINE			
		=======		:
> NAIVE	BAYES:	87.1	5%	
precision	recall	f1-score	support	
0.78	0.91	0.84	1318	
		0.74	966	
		0.80	2284	
0.82	0.79	0.79	2284	
0.81	0.80	0.80	2284	
YES				
	=======	========		:
	0.75 0.77 0.78 VECTOR MACH ======== BAYES > NAIVE > NAIVE precision 0.78 0.85 0.82 0.81	0.75 0.72 0.77 0.78 VECTOR MACHINE BAYES > NAIVE BAYES: > NAIVE BAYES: precision recall 0.78 0.91 0.85 0.66 0.82 0.79 0.81 0.80	0.80 0.82 0.81 0.75 0.72 0.73 0.78 0.77 0.77 0.78 0.78 0.78 VECTOR MACHINE BAYES > NAIVE BAYES: 80.47 > NAIVE BAYES: 87.19 precision recall f1-score 0.78 0.91 0.84 0.85 0.66 0.74 0.80 0.82 0.79 0.79 0.81 0.80 0.80	0.75 0.72 0.73 966 0.78 2284 0.77 0.77 0.77 2284 0.78 0.78 0.78 2284 VECTOR MACHINE BAYES > NAIVE BAYES: 80.47%> NAIVE BAYES: 87.15% precision recall f1-score support 0.78 0.91 0.84 1318 0.85 0.66 0.74 966 0.80 2284 0.82 0.79 0.79 2284 0.81 0.80 0.80 2284

Recurrent Neural Networks

In [17]:

```
import pandas as pd
   import numpy as np
 2
 4
   from sklearn.model_selection import train_test_split
 5
   from sklearn.feature_extraction.text import TfidfVectorizer
 6
 7
   from sklearn.linear_model import LogisticRegression
8
   from sklearn.svm import LinearSVC
9
   from sklearn.naive_bayes import MultinomialNB
10
11
   from sklearn.metrics import accuracy_score, classification_report
12
13
   def train_model(model_name, model, X_train, X_test, y_train, y_test):
       print(f'BEGIN. {model_name.upper()}.....')
14
       model.fit(X_train, y_train)
15
16
       y pred = model.predict(X test)
       y train pred = model.predict(X train)
17
18
       print(f'TESTING DATA----> {model_name.upper()}: \t\t{accuracy_score(y_test, y_pred)}
       print(f'TRAINING DATA---> {model_name.upper()}: \t\t{accuracy_score(y_train, y_train)}
19
       print(classification_report(y_test, y_pred))
20
21
       print(f'END. {model_name.upper()}')
22
       print('=======')
23
       return y pred
24
25
   data = pd.read_csv('train.csv')
26
27
   print('========Splitting the data========')
28
   X = data.text
29
   y = data.target
30
   print(f'Data shape: {data.shape}')
31
32
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
33
   print(f'X_Train shape: {X_train.shape}, y_train shape: {y_train.shape}')
34
   print(f'X_Test shape: {X_test.shape}, y_test shape: {y_test.shape}')
35
   print('\n========Message Preprocessing========')
36
   vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_df=0.9, min_df=2, stop_words='engl
37
38
   X_train_vect = vectorizer.fit_transform(X_train)
39
   X_test_vect = vectorizer.transform(X_test)
40
41
   print('Training and testing data shape after pre-processing:')
42
   print(f'X_Train shape: {X_train_vect.shape}, y_train shape: {y_train.shape}')
43
   print(f'X_Test shape: {X_test_vect.shape}, y_test shape: {y_test.shape}')
44
   print('\n========Model Building========')
45
   lr_model = LogisticRegression()
46
   lr_y_pred = train_model('Logistic Regression', lr_model, X_train_vect, X_test_vect, y_1
47
48
   svm_model = LinearSVC()
49
50
   svm y pred = train model('Support Vector Machine', svm model, X train vect, X test vect
51
52 | nb model = MultinomialNB()
53
   nb_y_pred = train_model('Naive Bayes', nb_model, X_train_vect, X_test_vect, y_train, y
```

========Message Preprocessing======== Training and testing data shape after pre-processing: X_Train shape: (5329, 8557), y_train shape: (5329,) X_Test shape: (2284, 8557), y_test shape: (2284,)					
=========	-Model Buildin	g=====	=======	=	
	IC REGRESSION	_			
	> LOGISTIC		ION:	80.12%	
TRAINING DATA	A> LOGISTIC	REGRESS	ION:	88.40%	
	precision	recall	f1-score	support	
2	0.00	0.00	0.04	4240	
0	0.80	0.88	0.84	1318	
1	0.81	0.69	0.75	966	
accuracy			0.80	2284	
macro avg	0.80	0.79	0.30	2284	
weighted avg	0.80	0.80	0.80	2284	
weighted dvg	0.00	0.00	0.00	2204	
END. LOGISTIC	REGRESSION				
=========	=========	======	=======	=======	
	RT VECTOR MACH				
	·> SUPPORT '				77.76%
TRAINING DATA	A> SUPPORT '				95.91%
	precision	recall	f1-score	support	
2	0.00	0.00	0.01	4240	
0 1	0.80	0.82	0.81	1318	
1	0.75	0.72	0.73	966	
accuracy			0.78	2284	
macro avg	0.77	0.77	0.73	2284	
weighted avg	0.78	0.78	0.77	2284	
weighted avg	0.78	0.76	0.76	2204	
END. SUPPORT	VECTOR MACHIN	E			
BEGIN. NAIVE BAYES					
TESTING DATA> NAIVE BAYES: 80.47%					
TRAINING DATA	A> NAIVE BA	YES:	87.15	5%	
	precision	recall	f1-score	support	
0	0.78	0.01	0.84	1210	
0	0.78 0.85	0.91	0.84 0.74	1318	
1	0.85	0.66	0.74	966	
accuracy			0.80	2284	
macro avg	0.82	0.79	0.79	2284	
weighted avg		0.80	0.80	2284	
	0.01	0.00	0.00	220-1	
END. NAIVE BAYES					

Recurrent Neural Networks

In [18]:

```
import pandas as pd
 2
   import numpy as np
   import matplotlib.pyplot as plt
   from sklearn.model_selection import train_test_split
 5
   from tensorflow.keras.preprocessing.text import Tokenizer
6
7
   from tensorflow.keras.preprocessing.sequence import pad_sequences
   from tensorflow.keras.layers import Dense, Input, GlobalAveragePooling1D, Dropout, Spa
   from tensorflow.keras.layers import LSTM, Embedding
9
   from tensorflow.keras.models import Model
10
   from tensorflow.keras.optimizers import Adam
11
12
13
   from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
14
15
16
   def plot loss evaluation(r):
       plt.figure(figsize=(12, 8))
17
18
       plt.subplot(2, 2, 1)
19
       plt.plot(r.history['loss'], label='loss')
20
21
       plt.plot(r.history['val_loss'], label='val_loss')
22
       plt.legend()
23
       plt.subplot(2, 2, 2)
24
       plt.plot(r.history['accuracy'], label='accuracy')
25
       plt.plot(r.history['val_accuracy'], label='val_acc')
26
27
       plt.legend()
28
29
       plt.title('Training and Loss fuction evolution')
30
   def evaluate(model, X_train, X_test, y_train, y_test):
31
32
       y_pred_train = np.round(model.predict(X_train))
33
       y_pred_test = np.round(model.predict(X_test))
34
35
       print("=======Training Data=======")
36
       print(confusion_matrix(y_train, y_pred_train))
       print(classification_report(y_train, y_pred_train))
37
38
       print(f"Accuracy score: {accuracy_score(y_train, y_pred_train) * 100:.2f}%")
39
       print("========Testing Data=======")
40
41
       print(confusion_matrix(y_test, y_pred_test))
42
       print(classification_report(y_test, y_pred_test))
43
       print(f"Accuracy score: {accuracy_score(y_test, y_pred_test) * 100:.2f}%")
44
45
   data = pd.read_csv("train.csv")
46
   print('========Splitting the data========')
47
48
   X = data.text
49
   y = data.target
50
   print(f'Data shape: {data.shape}')
51
52
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
53
   print(f'X_Train shape: {X_train.shape}, y_train shape: {y_train.shape}')
   print(f'X_Test shape: {X_test.shape}, y_test shape: {y_test.shape}')
54
55
56
   print('========Convert Sentences to Sequences=========')
57
   MAX_VOCAB_SIZE = 20000
58
   tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE, char_level=False)
59
   tokenizer.fit_on_texts(X_train)
```

```
sequences_train = tokenizer.texts_to_sequences(X_train)
60
61
    sequences_test = tokenizer.texts_to_sequences(X_test)
62
    # pad sequence do that we get a NxT matrix
63
64
    data train = pad sequences(sequences train)
    data_test = pad_sequences(sequences_test, maxlen=data_train.shape[1])
65
    print(f"Found {len(tokenizer.word_index)} unique tokens.")
    print(f"Training Data shape: {data_train.shape}")
67
    print(f"Testing Data shape: {data_test.shape}")
68
69
    print('=======Create The Model========')
70
   # We get to choose embedding dimensionality
71
72 D = 100
73
    # Hidden state dimentionality
74
   M = 64
75
   V = len(tokenizer.word index)
76
    T = data_train.shape[1]
77
78
   # model.add(embedding)
   # model.add(SpatialDropout1D(0.2))
79
    # model.add(LSTM(64, dropout=0.2, recurrent dropout=0.2))
80
81
    # model.add(Dense(1, activation='sigmoid'))
82
83
    i = Input(shape=(T,))
84
    x = Embedding(V + 1, D)(i)
85
    x = SpatialDropout1D(0.2)(x)
   x = LSTM(M, return sequences=True, activation='relu')(x)
87
    x = GlobalAveragePooling1D()(x)
88
    \# x = Dropout(0.2)(x)
    x = Dense(1, activation='sigmoid')(x)
89
90
    model = Model(i, x)
91
92
    optimizer = Adam(learning_rate=1e-5)
93
    # Compile and fit
94
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
95
    print('Training model....')
    r = model.fit(data_train, y_train, epochs=50,
96
                 validation_data=(data_test, y_test),
97
98
                 batch_size=16)
99
100
    evaluate(model, data_train, data_test, y_train, y_test)
101
102
    plot_loss_evaluation(r)
103
```

```
accuracy: 0.5701 - val_loss: 0.6896 - val_accuracy: 0.5771
Epoch 3/50
334/334 [============= ] - 10s 31ms/step - loss: 0.6899 -
accuracy: 0.5596 - val loss: 0.6869 - val accuracy: 0.5771
In [19]:
 1 data.text.str.len()
Out[19]:
        69
0
1
        38
2
       133
3
        65
        88
7608
        83
7609
       125
        65
7610
       137
7611
7612
        94
Name: text, Length: 7613, dtype: int64
```

Convolutional Neural Networks

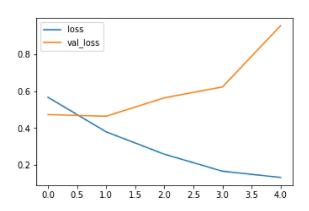
In [20]:

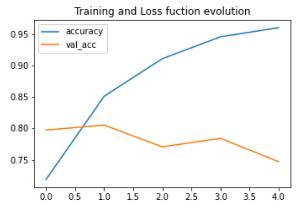
```
1
   import pandas as pd
 2
   import numpy as np
   import matplotlib.pyplot as plt
   from sklearn.model_selection import train_test_split
 5
   from tensorflow.keras.preprocessing.text import Tokenizer
6
7
   from tensorflow.keras.preprocessing.sequence import pad_sequences
   from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D, MaxPooling1D
9
   from tensorflow.keras.layers import Conv1D, Embedding, Dropout
10
   from tensorflow.keras.models import Model
11
12
   from sklearn.metrics import confusion matrix, classification report, accuracy score
13
14
   def plot_loss_evaluation(r):
15
16
       plt.figure(figsize=(12, 8))
17
18
       plt.subplot(2, 2, 1)
       plt.plot(r.history['loss'], label='loss')
19
       plt.plot(r.history['val_loss'], label='val_loss')
20
21
       plt.legend()
22
23
       plt.subplot(2, 2, 2)
       plt.plot(r.history['accuracy'], label='accuracy')
24
       plt.plot(r.history['val_accuracy'], label='val_acc')
25
26
       plt.legend()
27
28
       plt.title('Training and Loss fuction evolution')
29
   def evaluate(model, X_train, X_test, y_train, y_test):
30
31
       y_pred_train = np.round(model.predict(X_train))
32
       y_pred_test = np.round(model.predict(X_test))
33
34
       print("========Training Data========")
       print(confusion_matrix(y_train, y_pred_train))
35
36
       print(classification_report(y_train, y_pred_train))
       print(f"Accuracy score: {accuracy_score(y_train, y_pred_train) * 100:.2f}%")
37
38
39
       print("=======Testing Data=======")
       print(confusion_matrix(y_test, y_pred_test))
40
41
       print(classification_report(y_test, y_pred_test))
42
       print(f"Accuracy score: {accuracy_score(y_test, y_pred_test) * 100:.2f}%")
43
44
   data = pd.read_csv("train.csv")
45
   print('========Splitting the data========')
46
47
   X = data.text
48
   y = data.target
49
   print(f'Data shape: {data.shape}')
50
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
51
52
   print(f'X_Train shape: {X_train.shape}, y_train shape: {y_train.shape}')
53
   print(f'X_Test shape: {X_test.shape}, y_test shape: {y_test.shape}')
54
55
   print('========Convert Sentences to Sequences=========')
56
   MAX_VOCAB_SIZE = 20000
57
   tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
58
   tokenizer.fit_on_texts(X_train)
59
   sequences_train = tokenizer.texts_to_sequences(X_train)
```

```
sequences_test = tokenizer.texts_to_sequences(X_test)
60
61
    # pad sequence do that we get a NxT matrix
62
    data_train = pad_sequences(sequences_train)
63
    data test = pad sequences(sequences test, maxlen=data train.shape[1])
    print(f"Found {len(tokenizer.word_index)} unique tokens.")
65
    print(f"Training Data shape: {data_train.shape}")
    print(f"Testing Data shape: {data_test.shape}")
67
68
69
    print('======Create The Model==========')
70
    # We get to choose embedding dimensionality
    D = 100
71
72
73
    V = len(tokenizer.word index)
74
    T = data train.shape[1]
75
76
    i = Input(shape=(T,))
77
    x = Embedding(V + 1, D)(i)
78
79
    x = Conv1D(32, 2, activation='relu')(x)
80
    x = MaxPooling1D()(x)
81
    x = Dropout(0.1)(x)
82
83
    x = Conv1D(64, 2, activation='relu')(x)
84
    x = MaxPooling1D()(x)
85
    x = Dropout(0.2)(x)
86
87
    x = Conv1D(128, 2, activation='relu')(x)
88
    x = MaxPooling1D()(x)
    x = Dropout(0.3)(x)
89
90
    x = Conv1D(264, 2, activation='relu')(x)
91
92
    x = GlobalMaxPooling1D()(x)
93
94
    x = Dropout(0.5)(x)
95
96
    x = Dense(1, activation='sigmoid')(x)
97
98
    model = Model(i, x)
99
100
    # Compile and fit
101
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
102
    print('Training model....')
103
    r = model.fit(data_train, y_train, epochs=5,
104
                 validation data=(data test, y test),
                 batch_size=1)
105
106
    107
    evaluate(model, data_train, data_test, y_train, y_test)
108
109
    plot_loss_evaluation(r)
```

```
Training model.....
Epoch 1/5
5329/5329 [=============== ] - 134s 25ms/step - loss: 0.6245 -
accuracy: 0.6507 - val_loss: 0.4736 - val_accuracy: 0.7973
5329/5329 [================ ] - 133s 25ms/step - loss: 0.3623 -
accuracy: 0.8628 - val_loss: 0.4646 - val_accuracy: 0.8052
Epoch 3/5
5329/5329 [=============== ] - 124s 23ms/step - loss: 0.2426 -
accuracy: 0.9193 - val_loss: 0.5639 - val_accuracy: 0.7706
Epoch 4/5
accuracy: 0.9457 - val_loss: 0.6227 - val_accuracy: 0.7842
Epoch 5/5
5329/5329 [=============== ] - 127s 24ms/step - loss: 0.1178 -
accuracy: 0.9615 - val loss: 0.9541 - val accuracy: 0.7469
=========Model Evaluation==========
========Training Data=========
[[2972
       52]
   55 2250]]
            precision
                       recall f1-score
                                        support
                         0.98
                                  0.98
                0.98
         0
                                           3024
         1
                0.98
                         0.98
                                  0.98
                                           2305
                                  0.98
                                           5329
   accuracy
                0.98
                         0.98
                                  0.98
                                           5329
  macro avg
                         0.98
                                  0.98
weighted avg
                0.98
                                           5329
Accuracy score: 97.99%
========Testing Data========
[[981 337]
[241 725]]
            precision
                       recall f1-score
                                         support
                         0.74
                                  0.77
         0
                0.80
                                           1318
                0.68
                         0.75
         1
                                  0.71
                                            966
                                  0.75
                                           2284
   accuracy
                0.74
                         0.75
                                  0.74
                                           2284
  macro avg
weighted avg
                0.75
                         0.75
                                  0.75
                                           2284
```

Accuracy score: 74.69%





NLTK Sentiment VADER

```
In [21]:
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
data['scores'] = data['text'].apply(lambda text: sid.polarity_scores(text))
data['compound'] = data['scores'].apply(lambda score_dict: score_dict['compound'])
data['comp_score'] = data['compound'].apply(lambda c: 0 if c >= 0 else 1)
```

```
In [22]:
```

```
1 accuracy_score(data['target'], data['comp_score'])
```

Out[22]:

0.5724418757388677

Making submission

In [23]:

```
# /kaggle/input/nlp-getting-started/test.csv
# /kaggle/input/nlp-getting-started/sample_submission.csv
test = pd.read_csv('test.csv')

print('==========Convert Sentences to Sequences=======')
sequences_test = tokenizer.texts_to_sequences(test.text)

# pad sequence do that we get a NxT matrix
data_test = pad_sequences(sequences_test, maxlen=data_train.shape[1])
print(f"Found {len(tokenizer.word_index)} unique tokens.")
print(f"Testing Data shape: {data_test.shape}")
```

In [24]:

```
sample_sub=pd.read_csv('sample_submission.csv')
y_pre = model.predict(data_test)
y_pre = np.round(y_pre).astype(int).reshape(3263)
sub = pd.DataFrame({'id':sample_sub['id'].values.tolist(), 'target':y_pre})
sub.to_csv('submission.csv', index=False)
```

```
In [25]:
```

1 sub.head()

Out[25]:

	id	target
0	0	0
1	2	1
2	3	1
3	9	0
4	11	1

In []:

1