**Import Requirements**

In [1]:

**import** re

**import** nltk

**import** string

**import** numpy **as** np

**import** pandas **as** pd

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**from** nltk.corpus **import** stopwords

**from** nltk.stem **import** SnowballStemmer, WordNetLemmatizer

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

**from** sklearn.preprocessing **import** LabelEncoder

**from** sklearn.model\_selection **import** train\_test\_split

**from** tensorflow.keras.utils **import** to\_categorical

**from** tensorflow.keras.preprocessing.text **import** Tokenizer

**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences

**from** tensorflow.keras.optimizers **import** Adam

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.callbacks **import** EarlyStopping

**from** tensorflow.keras.layers **import** Dense, LSTM, Embedding, Bidirectional

*#nltk.download("stopwords")*

stop\_words **=** set(stopwords**.**words("english"))

lemmatizer**=** WordNetLemmatizer()

*# Modelling*

**from** sklearn.model\_selection **import** train\_test\_split,KFold, GridSearchCV

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** accuracy\_score,confusion\_matrix, classification\_report

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.metrics **import** f1\_score

*#Lime*

**from** lime **import** lime\_text

**from** lime.lime\_text **import** LimeTextExplainer

**from** lime.lime\_text **import** IndexedString,IndexedCharacters

**from** lime.lime\_base **import** LimeBase

**from** lime.lime\_text **import** explanation

sns**.**set(font\_scale**=**1.3)

nltk**.**download('omw-1.4')

*# Logging*

**import** logging

logging**.**basicConfig(level**=**logging**.**INFO)

[nltk\_data] Error loading omw-1.4: <urlopen error [Errno 11001]

[nltk\_data] getaddrinfo failed>

**Read Data**

In [2]:

df **=** pd**.**read\_csv('./data/tweet\_emotions.csv', delimiter**=**',')

df**.**head()

Out[2]:

|  | **tweet\_id** | **sentiment** | **content** |
| --- | --- | --- | --- |
| **0** | 1956967341 | empty | @tiffanylue i know i was listenin to bad habi... |
| **1** | 1956967666 | sadness | Layin n bed with a headache ughhhh...waitin o... |
| **2** | 1956967696 | sadness | Funeral ceremony...gloomy friday... |
| **3** | 1956967789 | enthusiasm | wants to hang out with friends SOON! |
| **4** | 1956968416 | neutral | @dannycastillo We want to trade with someone w... |

**Exploratory Data Analysis (EDA)**

**Counting number of tweets in Data**

In [3]:

print('No. of Tweets: ', df**.**shape[0])

No. of Tweets: 40000

In [4]:

print()

print("-- Number of null values in the columns --")

print(df**.**isnull()**.**sum())

print()

print('-- Sentiment-Count --')

print()

print(df**.**sentiment**.**value\_counts())

-- Number of null values in the columns --

tweet\_id 0

sentiment 0

content 0

dtype: int64

-- Sentiment-Count --

neutral 8638

worry 8459

happiness 5209

sadness 5165

love 3842

surprise 2187

fun 1776

relief 1526

hate 1323

empty 827

enthusiasm 759

boredom 179

anger 110

Name: sentiment, dtype: int64

In [5]:

all\_classes **=** df**.**sentiment**.**unique()**.**tolist()

print(all\_classes)

['empty', 'sadness', 'enthusiasm', 'neutral', 'worry', 'surprise', 'love', 'fun', 'hate', 'happiness', 'boredom', 'relief', 'anger']

**Distribution of Sentiments and Checks for Data Imbalance**

In [6]:

col **=** 'sentiment'

fig, (ax1, ax2) **=** plt**.**subplots(nrows**=**1, ncols**=**2, figsize**=**(12,8))

explode **=** list((np**.**array(list(df[col]**.**dropna()**.**value\_counts()))**/**sum(list(df[col]**.**dropna()**.**value\_counts())))[::**-**1])[:10]

labels **=** list(df[col]**.**dropna()**.**unique())[:10]

sizes **=** df[col]**.**value\_counts()[:10]

*#ax.pie(sizes, explode=explode, colors=bo, startangle=60, labels=labels,autopct='%1.0f%%', pctdistance=0.9)*

ax2**.**pie(sizes, explode**=**explode, startangle**=**60, labels**=**labels,autopct**=**'%1.0f%%', pctdistance**=**0.9)

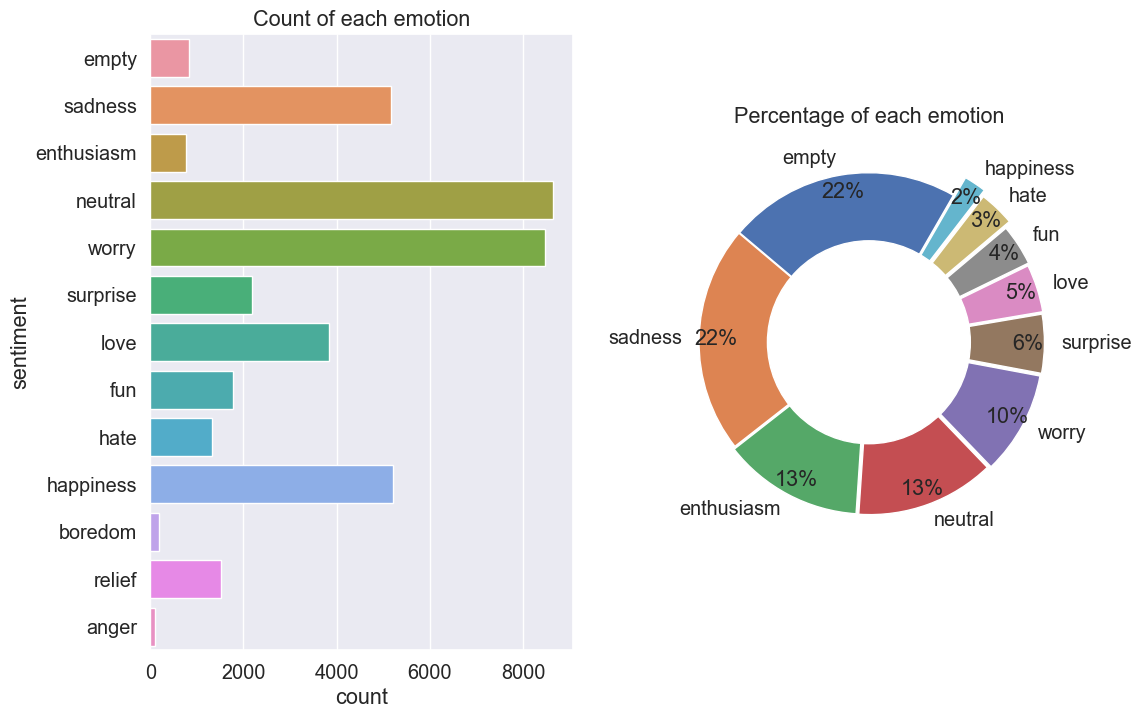
ax2**.**add\_artist(plt**.**Circle((0,0),0.6,fc**=**'white'))

sns**.**countplot(y **=**col, data **=** df, ax**=**ax1)

ax1**.**set\_title("Count of each emotion")

ax2**.**set\_title("Percentage of each emotion")

plt**.**show()



We notice that there's data imbalance, as some classes are very large (neutral, worry, happiness), while others are very small (anger, boredom, empty, etc). We proceed to applying data balancing technique.

**Data Balancing by Upsampling**

upsample the other classes to match the largest class in dataframe (neural)

In [7]:

**from** sklearn.utils **import** resample

maxx **=** 3

df\_majority **=** df[df**.**sentiment**==**all\_classes[maxx]]

**for** cl **in** range(13):

df\_minority **=** df[df**.**sentiment**==**all\_classes[cl]]

df\_minority\_upsampled **=** resample(df\_minority, replace**=True**, n\_samples**=**len(df\_majority), random\_state**=**123)

**if** cl **==** 0:

df\_upsampled **=** pd**.**concat([df\_minority\_upsampled, df\_majority])

**if** cl**>**0 **and** cl**!=**maxx:

df\_upsampled **=** pd**.**concat([df\_minority\_upsampled, df\_upsampled])

df\_upsampled['sentiment']**.**value\_counts()

Out[7]:

anger 8638

relief 8638

boredom 8638

happiness 8638

hate 8638

fun 8638

love 8638

surprise 8638

worry 8638

enthusiasm 8638

sadness 8638

empty 8638

neutral 8638

Name: sentiment, dtype: int64

In [8]:

df\_upsampled**.**shape

Out[8]:

(112294, 3)

**View Balanced Distribution**

In [9]:

col **=** 'sentiment'

fig, (ax1, ax2) **=** plt**.**subplots(nrows**=**1, ncols**=**2, figsize**=**(12,8))

explode **=** list((np**.**array(list(df\_upsampled[col]**.**dropna()**.**value\_counts()))**/**sum(list(df\_upsampled[col]**.**dropna()**.**value\_counts())))[::**-**1])[:10]

labels **=** list(df\_upsampled[col]**.**dropna()**.**unique())[:10]

sizes **=** df\_upsampled[col]**.**value\_counts()[:10]

*#ax.pie(sizes, explode=explode, colors=bo, startangle=60, labels=labels,autopct='%1.0f%%', pctdistance=0.9)*

ax2**.**pie(sizes, explode**=**explode, startangle**=**60, labels**=**labels,autopct**=**'%1.0f%%', pctdistance**=**0.9)

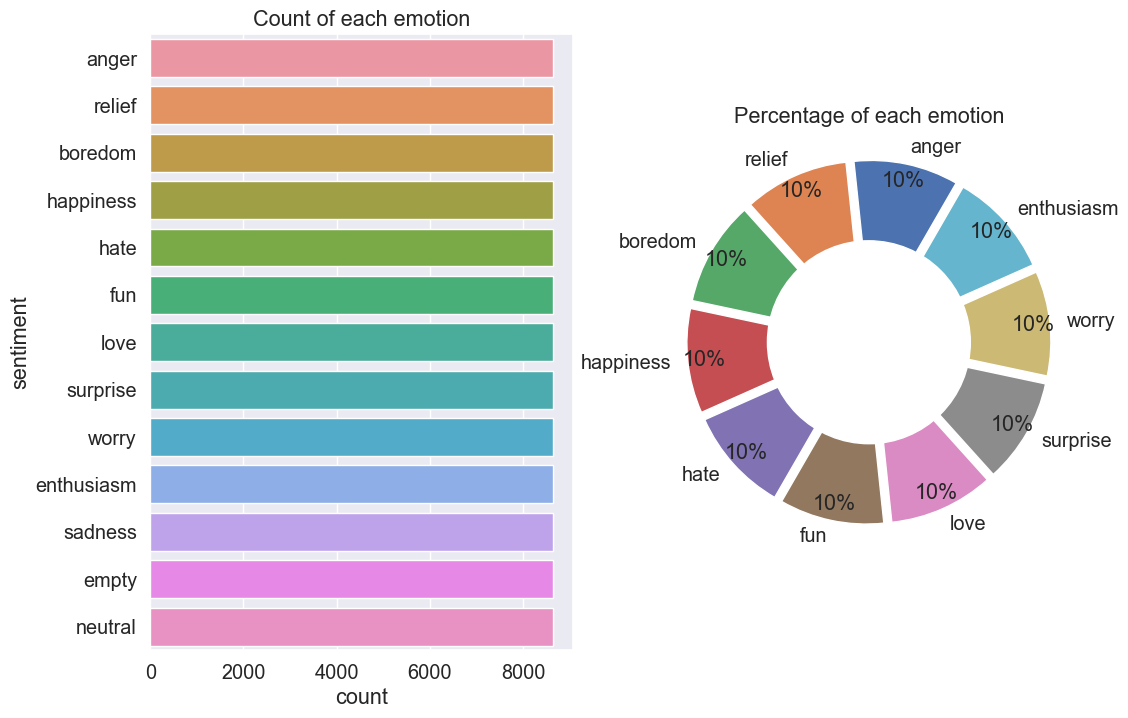
ax2**.**add\_artist(plt**.**Circle((0,0),0.6,fc**=**'white'))

sns**.**countplot(y **=**col, data **=** df\_upsampled, ax**=**ax1)

ax1**.**set\_title("Count of each emotion")

ax2**.**set\_title("Percentage of each emotion")

plt**.**show()



**Distribution of character length and token length overall**

In [10]:

df\_upsampled\_copy **=** df\_upsampled**.**copy()

In [11]:

df\_upsampled\_copy['char\_length'] **=** df\_upsampled\_copy['content']**.**apply(**lambda** x : len(x))

df\_upsampled\_copy['token\_length'] **=** df\_upsampled\_copy['content']**.**apply(**lambda** x : len(x**.**split(" ")))

In [12]:

fig, (ax1, ax2) **=** plt**.**subplots(nrows**=**1, ncols**=**2, figsize**=**(12,6))

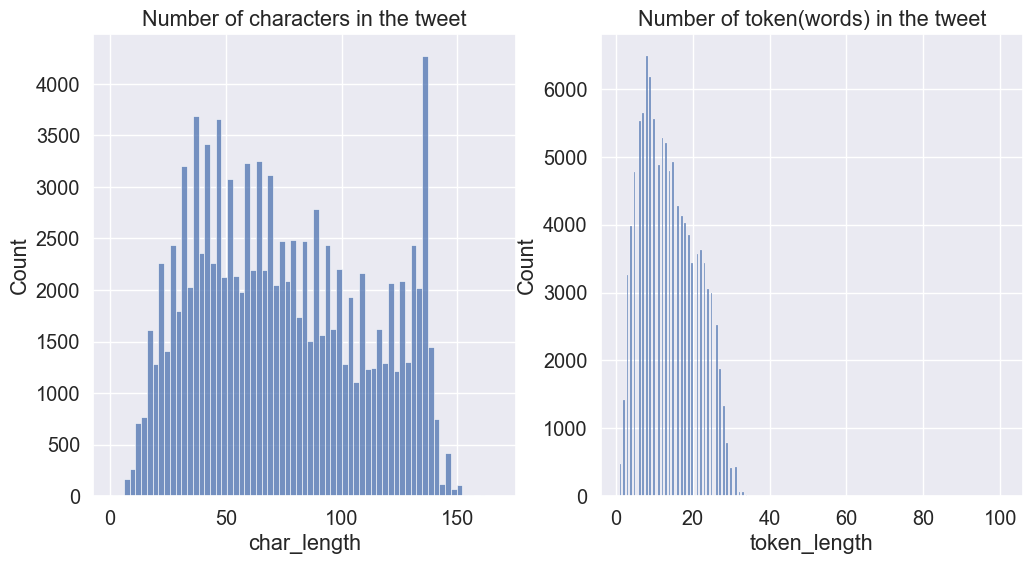
sns**.**histplot(df\_upsampled\_copy['char\_length'], ax**=**ax1)

sns**.**histplot(df\_upsampled\_copy['token\_length'], ax**=**ax2)

ax1**.**set\_title('Number of characters in the tweet')

ax2**.**set\_title('Number of token(words) in the tweet')

plt**.**show()



**Distribution of character length sentiment-wise [Top 6 sentiments]**

In [13]:

fig, ax **=** plt**.**subplots(figsize**=**(16,8))

**for** sentiment **in** df\_upsampled\_copy['sentiment']**.**value\_counts()**.**sort\_values()[**-**6:]**.**index**.**tolist():

*#print(sentiment)*

sns**.**kdeplot(df\_upsampled\_copy[df\_upsampled\_copy['sentiment']**==**sentiment]['char\_length'],ax**=**ax, label**=**sentiment)

ax**.**legend()

ax**.**set\_title("Distribution of character length sentiment-wise [Top 6 sentiments]")

plt**.**show()



**Distribution of token length sentiment-wise [Top 6 sentiments]**

In [14]:

fig, ax **=** plt**.**subplots(figsize**=**(8,6))

**for** sentiment **in** df\_upsampled\_copy['sentiment']**.**value\_counts()**.**sort\_values()[**-**6:]**.**index**.**tolist():

*#print(sentiment)*

sns**.**kdeplot(df\_upsampled\_copy[df\_upsampled\_copy['sentiment']**==**sentiment]['token\_length'],ax**=**ax, label**=**sentiment)

ax**.**legend()

ax**.**set\_title("Distribution of token length sentiment-wise [Top 6 sentiments]")

plt**.**show()



**Average Character and Token Length for each emotion class**

In [15]:

avg\_df **=** df\_upsampled\_copy**.**groupby('sentiment')**.**agg({'char\_length':'mean', 'token\_length':'mean'})

In [16]:

fig, (ax1, ax2) **=** plt**.**subplots(nrows**=**1, ncols**=**2, figsize**=**(20,10))

ax1**.**bar(avg\_df**.**index, avg\_df['char\_length'])

ax2**.**bar(avg\_df**.**index, avg\_df['token\_length'], color**=**'green')

ax1**.**set\_title('Avg number of characters')

ax2**.**set\_title('Avg number of token(words)')

ax1**.**set\_xticklabels(avg\_df**.**index, rotation **=** 45)

ax2**.**set\_xticklabels(avg\_df**.**index, rotation **=** 45)

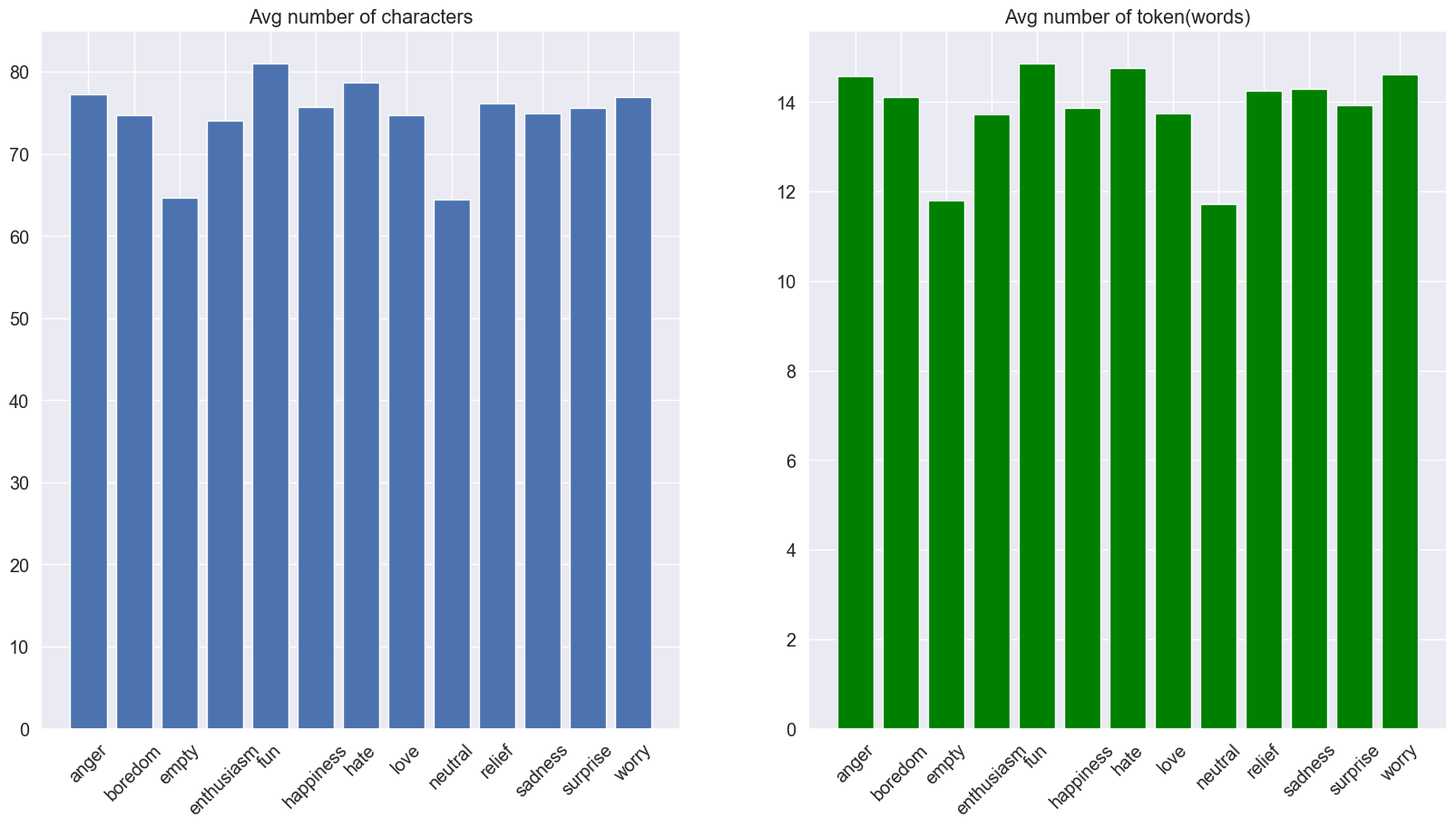
plt**.**show()

C:\Users\hp\AppData\Local\Temp\ipykernel\_6928\2512640700.py:6: UserWarning: FixedFormatter should only be used together with FixedLocator

ax1.set\_xticklabels(avg\_df.index, rotation = 45)

C:\Users\hp\AppData\Local\Temp\ipykernel\_6928\2512640700.py:7: UserWarning: FixedFormatter should only be used together with FixedLocator

ax2.set\_xticklabels(avg\_df.index, rotation = 45)



**Data Preprocessing**

In [17]:

**from** sklearn.preprocessing **import** LabelEncoder

**from** sklearn.model\_selection **import** train\_test\_split

**import** nltk

*#from num2words import num2words*

**import** inflect

**import** contractions

**from** bs4 **import** BeautifulSoup

**import** re**,** string**,** unicodedata

**from** nltk **import** word\_tokenize, sent\_tokenize

**from** nltk.corpus **import** stopwords

nltk**.**download('punkt')

nltk**.**download('wordnet')

**from** nltk.stem **import** LancasterStemmer, WordNetLemmatizer

[nltk\_data] Error loading punkt: <urlopen error [Errno 11001]

[nltk\_data] getaddrinfo failed>

[nltk\_data] Error loading wordnet: <urlopen error [Errno 11001]

[nltk\_data] getaddrinfo failed>

**Text normalization function**

In [18]:

**def** text\_preprocessing\_platform(df, text\_col, remove\_stopwords**=True**):

**def** denoise\_text(text):

*# Strip html if any. For ex. removing <html>, <p> tags*

soup **=** BeautifulSoup(text, "html.parser")

text **=** soup**.**get\_text()

*# Replace contractions in the text. For ex. didn't -> did not*

text **=** contractions**.**fix(text)

**return** text

**def** remove\_non\_ascii(words):

"""Remove non-ASCII characters from list of tokenized words"""

new\_words **=** []

**for** word **in** words:

new\_word **=** unicodedata**.**normalize('NFKD', word)**.**encode('ascii', 'ignore')**.**decode('utf-8', 'ignore')

new\_words**.**append(new\_word)

**return** new\_words

**def** to\_lowercase(words):

"""Convert all characters to lowercase from list of tokenized words"""

new\_words **=** []

**for** word **in** words:

new\_word **=** word**.**lower()

new\_words**.**append(new\_word)

**return** new\_words

**def** remove\_punctuation(words):

"""Remove punctuation from list of tokenized words"""

new\_words **=** []

**for** word **in** words:

new\_word **=** re**.**sub(r'[^\w\s]', '', word)

**if** new\_word **!=** '':

new\_words**.**append(new\_word)

**return** new\_words

**def** replace\_numbers(words):

"""Replace all interger occurrences in list of tokenized words with textual representation"""

p **=** inflect**.**engine()

new\_words **=** []

**for** word **in** words:

**if** word**.**isdigit():

new\_word **=** p**.**number\_to\_words(word)

new\_words**.**append(new\_word)

**else**:

new\_words**.**append(word)

**return** new\_words

**def** remove\_stopwords(words):

"""Remove stop words from list of tokenized words"""

new\_words **=** []

**for** word **in** words:

**if** word **not** **in** stopwords**.**words('english'):

new\_words**.**append(word)

**return** new\_words

**def** stem\_words(words):

"""Stem words in list of tokenized words"""

stemmer **=** LancasterStemmer()

stems **=** []

**for** word **in** words:

stem **=** stemmer**.**stem(word)

stems**.**append(stem)

**return** stems

**def** lemmatize\_verbs(words):

"""Lemmatize verbs in list of tokenized words"""

lemmatizer **=** WordNetLemmatizer()

lemmas **=** []

**for** word **in** words:

lemma **=** lemmatizer**.**lemmatize(word, pos**=**'v')

lemmas**.**append(lemma)

**return** lemmas

*### A wrap-up function for normalization*

**def** normalize\_text(words, remove\_stopwords):

words **=** remove\_non\_ascii(words)

words **=** to\_lowercase(words)

words **=** remove\_punctuation(words)

words **=** replace\_numbers(words)

**if** remove\_stopwords:

words **=** remove\_stopwords(words)

*#words = stem\_words(words)*

words **=** lemmatize\_verbs(words)

**return** words

*# Tokenize tweet into words*

**def** tokenize(text):

**return** nltk**.**word\_tokenize(text)

*# A overall wrap-up function*

**def** text\_prepare(text):

text **=** denoise\_text(text)

text **=** ' '**.**join([x **for** x **in** normalize\_text(tokenize(text), remove\_stopwords)])

**return** text

*# run every-step*

df[text\_col] **=** [text\_prepare(x) **for** x **in** df[text\_col]]

*# return processed df*

**return** df

In [19]:

print("Before Text Preprocessing")

display(df\_upsampled**.**head()[['content']])

processed\_df **=** text\_preprocessing\_platform(df\_upsampled, 'content', remove\_stopwords**=False**)

print("After Text Preprocessing")

display(processed\_df**.**head()[['content']])

Before Text Preprocessing

|  | **content** |
| --- | --- |
| **39222** | @johncmayer you are one of my favorite musicia... |
| **15506** | Had a shower. it's 5:55 PM. Triple 5's! Crap, ... |
| **24880** | wakey wakey lemon shakeyyyy! haha, goin' 2 sc... |
| **32244** | @Buddy021193 i hear you.. it pisses me off haha |
| **33361** | @roberto121 that's some serious shit steve. wh... |

c:\Users\hp\miniconda3\envs\nlp\lib\site-packages\bs4\\_\_init\_\_.py:435: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to open this file and pass the filehandle into Beautiful Soup.

warnings.warn(

After Text Preprocessing

|  | **content** |
| --- | --- |
| **39222** | johncmayer one favorite musiciansartists ever ... |
| **15506** | shower five hundred and fifty-five pm triple f... |
| **24880** | wakey wakey lemon shakeyyyy haha go two school... |
| **32244** | buddy021193 hear piss haha |
| **33361** | roberto121 serious shit steve send picture cal... |

**Train-test Split**

**Encode sentiment**

**apply one-hot encoding to the target variable, "sentiment"**

In [20]:

*# Label encoding target column*

le **=** LabelEncoder()

df\_upsampled['sentiment'] **=** le**.**fit\_transform(df\_upsampled['sentiment'])

*## df for training and prediction:*

df **=** df\_upsampled

In [21]:

preprocess **=** **True**

text **=** 'content'

target **=** 'sentiment'

MAX\_SEQUENCE\_LENGTH **=** 60

In [22]:

X **=** df[text]

y **=** df[target]

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.2)

In [23]:

y**.**tail()

Out[23]:

39990 8

39991 8

39992 8

39993 8

39995 8

Name: sentiment, dtype: int32

In [24]:

X**.**head()

Out[24]:

39222 johncmayer one favorite musiciansartists ever ...

15506 shower five hundred and fifty-five pm triple f...

24880 wakey wakey lemon shakeyyyy haha go two school...

32244 buddy021193 hear piss haha

33361 roberto121 serious shit steve send picture cal...

Name: content, dtype: object

**Embedding**

**Selected embedding:**

* **GloVe**: Global Vector for Word Representation

In [25]:

**def** loadData\_Tokenizer(X\_train, X\_test, MAX\_NB\_WORDS**=**75000, MAX\_SEQUENCE\_LENGTH**=**60):

np**.**random**.**seed(7)

text **=** np**.**concatenate((X\_train, X\_test), axis**=**0)

text **=** np**.**array(text)

tokenizer **=** Tokenizer(num\_words**=**MAX\_NB\_WORDS)

tokenizer**.**fit\_on\_texts(text)

sequences **=** tokenizer**.**texts\_to\_sequences(text)

word\_index **=** tokenizer**.**word\_index

text **=** pad\_sequences(sequences, maxlen**=**MAX\_SEQUENCE\_LENGTH)

print('Found %s unique tokens.' **%** len(word\_index))

indices **=** np**.**arange(text**.**shape[0])

*# np.random.shuffle(indices)*

text **=** text[indices]

print(text**.**shape)

X\_train **=** text[0:len(X\_train), ]

X\_test **=** text[len(X\_train):, ]

embeddings\_index **=** {}

f **=** open("./glove.42B.300d/glove.42B.300d.txt", encoding**=**'utf-8')

**for** line **in** f:

**try**:

values **=** line**.**split()

word **=** values[0]

**try**:

coefs **=** np**.**asarray(values[1:], dtype**=**'float32')

**except**:

**pass**

embeddings\_index[word] **=** coefs

**except** UnicodeDecodeError:

**pass**

f**.**close()

print('Total %s word vectors.' **%** len(embeddings\_index))

**return** (X\_train, X\_test, word\_index,embeddings\_index, tokenizer)

**Model Evaluation Functions**

In [26]:

**from** sklearn.metrics **import** matthews\_corrcoef, confusion\_matrix

**from** sklearn **import** metrics

**from** sklearn.utils **import** shuffle

In [27]:

**def** get\_eval\_report(labels, preds):

mcc **=** matthews\_corrcoef(labels, preds)

tn, fp, fn, tp **=** confusion\_matrix(labels, preds)**.**ravel()

precision **=** (tp)**/**(tp**+**fp)

recall **=** (tp)**/**(tp**+**fn)

f1 **=** (2**\***(precision**\***recall))**/**(precision**+**recall)

**return** {

"mcc": mcc,

"tp": tp,

"tn": tn,

"fp": fp,

"fn": fn,

"precision" : precision,

"recall" : recall,

"F1" : f1,

"accuracy": (tp**+**tn)**/**(tp**+**tn**+**fp**+**fn)

}

**def** compute\_metrics(labels, preds):

**assert** len(preds) **==** len(labels)

**return** get\_eval\_report(labels, preds)

**def** plot\_graphs(history, string):

plt**.**plot(history**.**history[string])

plt**.**plot(history**.**history['val\_'**+**string], '')

plt**.**xlabel("Epochs")

plt**.**ylabel(string)

plt**.**legend([string, 'val\_'**+**string])

plt**.**show()

**def** class\_balance(df, target):

cls **=** df[target]**.**value\_counts()

cls**.**plot(kind**=**'bar')

plt**.**show()

**Confusion Matrix**

In [28]:

**from** sklearn.metrics **import** confusion\_matrix

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**def** plot\_confusion\_matrix(confusion\_mtx, model\_name, classes):

plt**.**figure(figsize**=**(10, 10))

sns**.**heatmap(confusion\_mtx, annot**=True**, fmt**=**'d', cmap**=**'Blues', xticklabels**=**classes, yticklabels**=**classes)

plt**.**title(f'Confusion Matrix - {model\_name}')

plt**.**xlabel('Predicted')

plt**.**ylabel('Actual')

plt**.**show()

**Model Creation & Training |*Bidirectional LSTM***

In [29]:

**import** numpy **as** np

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Embedding, LSTM, Dropout, Dense, Bidirectional

**def** Build\_Model\_RNN\_Text(word\_index, embeddings\_index, nclasses,

MAX\_SEQUENCE\_LENGTH**=**60, EMBEDDING\_DIM**=**300, dropout**=**0.5, lstm\_node**=**32,

optimizer**=**'adam'):

"""Builds a Bidirectional LSTM-based neural network for text classification using word embeddings."""

*# Model building*

model **=** Sequential()

*# Create an embedding matrix using pre-trained word embeddings*

embedding\_matrix **=** np**.**zeros((len(word\_index) **+** 1, EMBEDDING\_DIM))

**for** word, i **in** word\_index**.**items():

embedding\_vector **=** embeddings\_index**.**get(word)

**if** embedding\_vector **is** **not** **None**:

embedding\_matrix[i] **=** embedding\_vector

*# Add the Embedding layer*

model**.**add(Embedding(len(word\_index) **+** 1,

EMBEDDING\_DIM,

weights**=**[embedding\_matrix],

input\_length**=**MAX\_SEQUENCE\_LENGTH,

trainable**=True**))

*# Add multiple Bidirectional LSTM layers with dropout*

**for** i **in** range(2):

model**.**add(Bidirectional(LSTM(lstm\_node, return\_sequences**=True**, recurrent\_dropout**=**0.5)))

model**.**add(Dropout(dropout))

*# Add the final Bidirectional LSTM layer with dropout*

model**.**add(Bidirectional(LSTM(lstm\_node, recurrent\_dropout**=**0.5)))

model**.**add(Dropout(dropout))

*# Add a Dense hidden layer with ReLU activation*

model**.**add(Dense(256, activation**=**'relu'))

*# Add the output Dense layer with softmax activation for multi-class classification*

model**.**add(Dense(nclasses, activation**=**'softmax'))

*# Compile the model*

model**.**compile(loss**=**'sparse\_categorical\_crossentropy',

optimizer**=**optimizer,

metrics**=**['accuracy'])

**return** model

In [30]:

print("Generating Glove Embeddings...")

X\_train\_Glove,X\_test\_Glove, word\_index,embeddings\_index, tokenizer **=** loadData\_Tokenizer(X\_train,X\_test, MAX\_SEQUENCE\_LENGTH**=**MAX\_SEQUENCE\_LENGTH)

Generating Glove Embeddings...

Found 40818 unique tokens.

(112294, 60)

Total 1917495 word vectors.

In [31]:

print(X\_train\_Glove**.**shape, X\_test\_Glove**.**shape)

(89835, 60) (22459, 60)

**Model Training**

In [32]:

**import** warnings

**import** tensorflow **as** tf

nclasses **=** len(y\_train**.**unique())

**with** warnings**.**catch\_warnings():

print("Building Model ...")

model\_RNN **=** Build\_Model\_RNN\_Text(word\_index,embeddings\_index, nclasses)

model\_RNN**.**summary()

tf**.**keras**.**utils**.**plot\_model(model\_RNN, show\_shapes **=** **True**)

print("\n Starting Training ... \n")

RNN\_history **=** model\_RNN**.**fit(X\_train\_Glove, y\_train,

validation\_data**=**(X\_test\_Glove, y\_test),

epochs**=**5,

batch\_size**=**128,

verbose**=**1)

warnings**.**simplefilter("ignore")

Building Model ...

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 60, 300) 12245700

bidirectional (Bidirectiona (None, 60, 64) 85248

l)

dropout (Dropout) (None, 60, 64) 0

bidirectional\_1 (Bidirectio (None, 60, 64) 24832

nal)

dropout\_1 (Dropout) (None, 60, 64) 0

bidirectional\_2 (Bidirectio (None, 64) 24832

nal)

dropout\_2 (Dropout) (None, 64) 0

dense (Dense) (None, 256) 16640

dense\_1 (Dense) (None, 13) 3341

=================================================================

Total params: 12,400,593

Trainable params: 12,400,593

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot\_model to work.

Starting Training ...

Epoch 1/5

702/702 [==============================] - 759s 1s/step - loss: 1.8604 - accuracy: 0.3634 - val\_loss: 1.1988 - val\_accuracy: 0.6007

Epoch 2/5

702/702 [==============================] - 669s 953ms/step - loss: 0.9717 - accuracy: 0.6803 - val\_loss: 0.8174 - val\_accuracy: 0.7385

Epoch 3/5

702/702 [==============================] - 687s 978ms/step - loss: 0.6374 - accuracy: 0.7952 - val\_loss: 0.7117 - val\_accuracy: 0.7881

Epoch 4/5

702/702 [==============================] - 850s 1s/step - loss: 0.4652 - accuracy: 0.8591 - val\_loss: 0.6401 - val\_accuracy: 0.8139

Epoch 5/5

702/702 [==============================] - 1247s 2s/step - loss: 0.3612 - accuracy: 0.8920 - val\_loss: 0.6641 - val\_accuracy: 0.8236

In [33]:

print("\n Plotting results ... \n")

RNN\_history **=** RNN\_history**.**history

fig, axs **=** plt**.**subplots(1,2, figsize**=**(16,8))

axs[0]**.**set\_title('Loss\_Curve')

ep **=** range(len(RNN\_history['loss']))

axs[0]**.**plot(ep, RNN\_history['loss'],'o--r',label **=** 'Training\_loss')

axs[0]**.**plot(ep, RNN\_history['val\_loss'],'o--b',label **=** 'Val\_loss')

axs[0]**.**set\_xlabel('epoch')

axs[0]**.**set\_ylabel('Loss')

axs[0]**.**legend()

axs[1]**.**set\_title('Acc\_Curve')

ep **=** range(len(RNN\_history['loss']))

axs[1]**.**plot(ep, RNN\_history['accuracy'],'o--r',label **=** 'Training\_acc')

axs[1]**.**plot(ep, RNN\_history['val\_accuracy'],'o--b',label **=** 'Val\_acc')

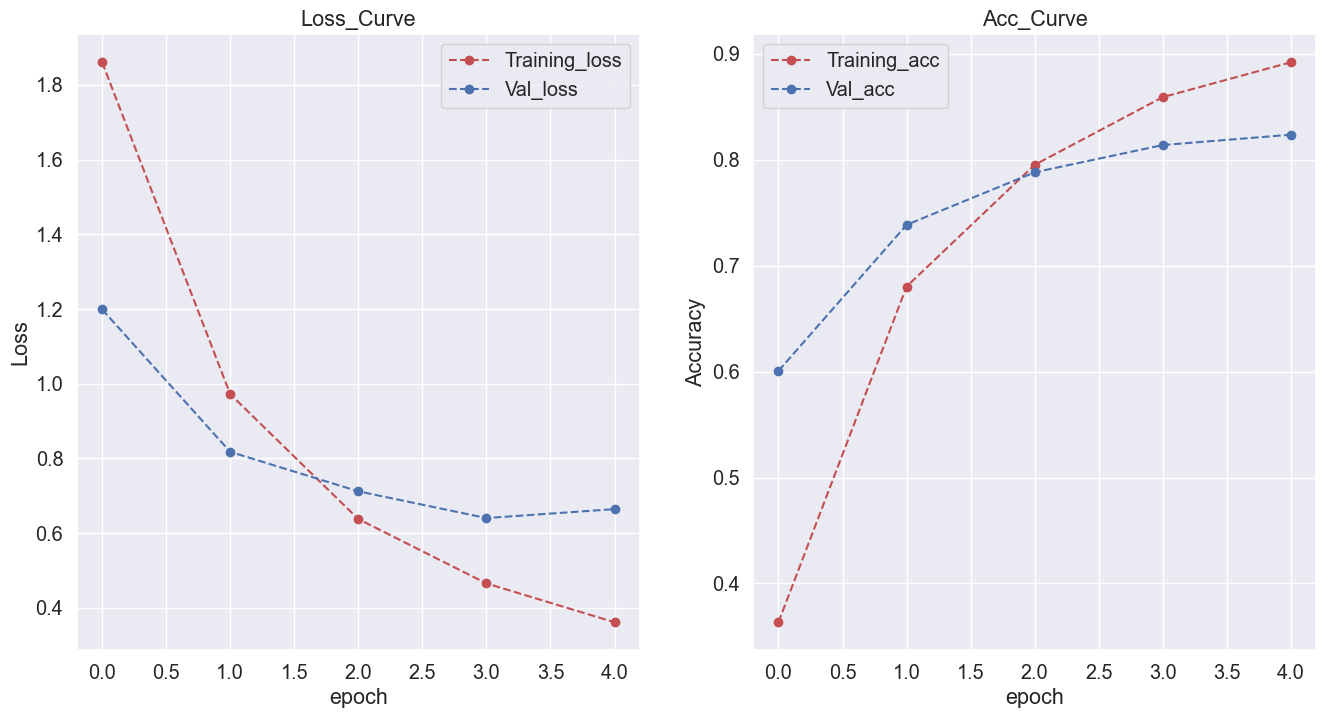
axs[1]**.**set\_xlabel('epoch')

axs[1]**.**set\_ylabel('Accuracy')

axs[1]**.**legend()

plt**.**show()

Plotting results ...



In [34]:

*# Evaluating Model*

predicted\_probabilities **=** model\_RNN**.**predict(X\_test\_Glove)

predicted\_classes **=** np**.**argmax(predicted\_probabilities, axis**=**1)

print(metrics**.**classification\_report(y\_test, predicted\_classes))

702/702 [==============================] - 206s 278ms/step

precision recall f1-score support

0 0.99 1.00 1.00 1678

1 0.99 0.99 0.99 1743

2 0.92 0.96 0.94 1796

3 0.93 0.98 0.96 1752

4 0.92 0.92 0.92 1709

5 0.72 0.69 0.70 1697

6 0.92 0.97 0.94 1680

7 0.79 0.80 0.80 1744

8 0.48 0.34 0.39 1682

9 0.88 0.92 0.90 1749

10 0.68 0.68 0.68 1747

11 0.76 0.88 0.82 1747

12 0.60 0.56 0.58 1735

accuracy 0.82 22459

macro avg 0.81 0.82 0.82 22459

weighted avg 0.81 0.82 0.82 22459

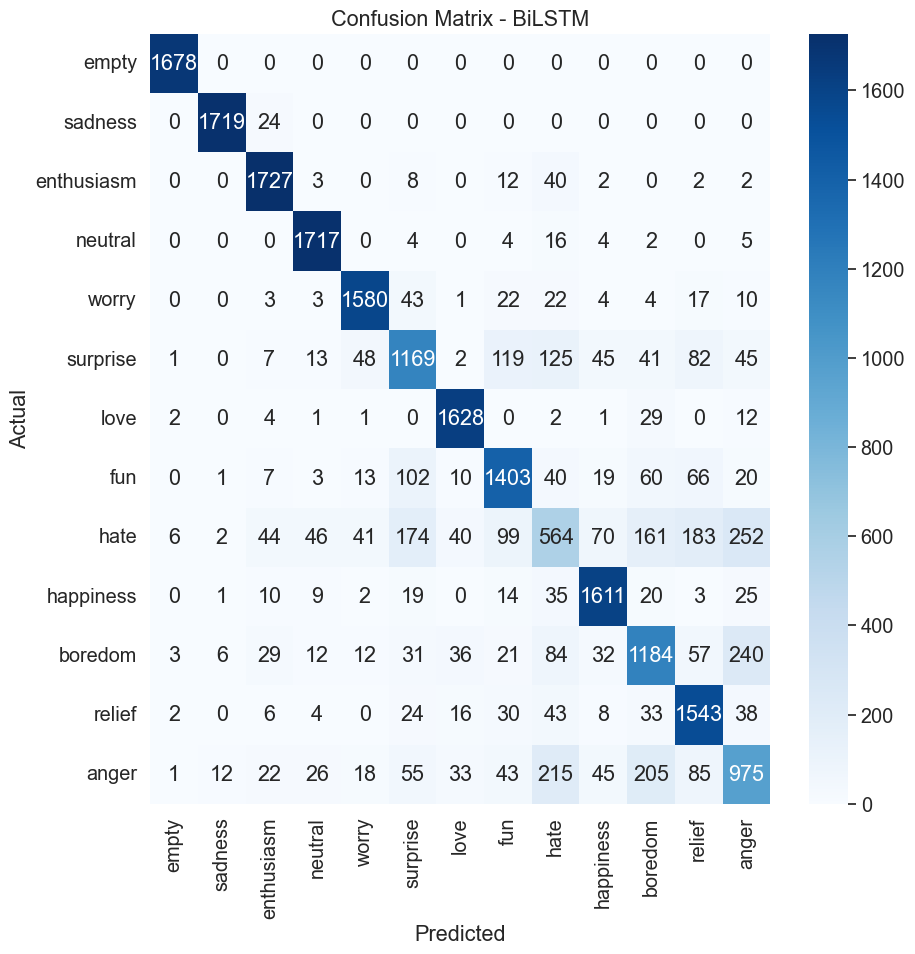
**Plot Confusion Matrix for BiLSTM**

In [35]:

confusion\_matrix\_bilstm **=** confusion\_matrix(y\_test, predicted\_classes)

In [36]:

plot\_confusion\_matrix(confusion\_matrix\_bilstm, "BiLSTM", all\_classes)



**Model Creation & Training |*Transformer***

In [37]:

**import** numpy **as** np

**import** tensorflow **as** tf

**from** tensorflow.keras.models **import** Model

**from** tensorflow.keras.layers **import** Input, Embedding, Dense, GlobalAveragePooling1D

**from** tensorflow.keras.optimizers **import** Adam

**def** build\_transformer\_model(word\_index, glove\_embeddings, nclasses,

MAX\_SEQUENCE\_LENGTH**=**60, EMBEDDING\_DIM**=**300, num\_transformer\_blocks**=**1,

intermediate\_dim**=**256, learning\_rate**=**1e-4):

"""Builds a simplified Transformer-based neural network for text classification using pre-trained GloVe word embeddings."""

*# Input layer*

input\_layer **=** Input(shape**=**(MAX\_SEQUENCE\_LENGTH,), dtype**=**'int32')

*# Embedding layer with pre-trained GloVe embeddings*

num\_words **=** len(word\_index) **+** 1

embedding\_matrix **=** np**.**zeros((num\_words, EMBEDDING\_DIM))

**for** word, i **in** word\_index**.**items():

embedding\_vector **=** glove\_embeddings**.**get(word)

**if** embedding\_vector **is** **not** **None**:

embedding\_matrix[i] **=** embedding\_vector

embedding\_layer **=** Embedding(num\_words, EMBEDDING\_DIM, weights**=**[embedding\_matrix],

input\_length**=**MAX\_SEQUENCE\_LENGTH, trainable**=True**)(input\_layer)

*# Transformer blocks*

transformer\_output **=** embedding\_layer

**for** \_ **in** range(num\_transformer\_blocks):

transformer\_output **=** Dense(intermediate\_dim, activation**=**'relu')(transformer\_output)

*# Global Average Pooling*

pooled\_output **=** GlobalAveragePooling1D()(transformer\_output)

*# Dense layers for classification*

dense\_layer **=** Dense(128, activation**=**'relu')(pooled\_output)

output\_layer **=** Dense(nclasses, activation**=**'softmax')(dense\_layer)

*# Build the model*

model **=** Model(inputs**=**input\_layer, outputs**=**output\_layer)

*# Compile the model*

model**.**compile(loss**=**'sparse\_categorical\_crossentropy',

optimizer**=**Adam(learning\_rate),

metrics**=**['accuracy'])

**return** model

In [ ]:

In [38]:

**import** warnings

**import** tensorflow **as** tf

nclasses **=** len(y\_train**.**unique())

**with** warnings**.**catch\_warnings():

print("Building Model ...")

model\_transformer **=** build\_transformer\_model(word\_index,embeddings\_index, nclasses)

model\_transformer**.**summary()

tf**.**keras**.**utils**.**plot\_model(model\_transformer, show\_shapes **=** **True**)

print("\n Starting Training ... \n")

transformer\_history **=** model\_transformer**.**fit(X\_train\_Glove, y\_train,

validation\_data**=**(X\_test\_Glove, y\_test),

epochs**=**5,

batch\_size**=**128,

verbose**=**1)

warnings**.**simplefilter("ignore")

Building Model ...

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_1 (InputLayer) [(None, 60)] 0

embedding\_1 (Embedding) (None, 60, 300) 12245700

dense\_2 (Dense) (None, 60, 256) 77056

global\_average\_pooling1d (G (None, 256) 0

lobalAveragePooling1D)

dense\_3 (Dense) (None, 128) 32896

dense\_4 (Dense) (None, 13) 1677

=================================================================

Total params: 12,357,329

Trainable params: 12,357,329

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot\_model to work.

Starting Training ...

Epoch 1/5

702/702 [==============================] - 203s 284ms/step - loss: 2.4952 - accuracy: 0.1790 - val\_loss: 2.3457 - val\_accuracy: 0.2356

Epoch 2/5

702/702 [==============================] - 155s 220ms/step - loss: 2.1830 - accuracy: 0.2808 - val\_loss: 2.0595 - val\_accuracy: 0.3149

Epoch 3/5

702/702 [==============================] - 192s 274ms/step - loss: 1.9363 - accuracy: 0.3603 - val\_loss: 1.8723 - val\_accuracy: 0.3806

Epoch 4/5

702/702 [==============================] - 227s 324ms/step - loss: 1.7741 - accuracy: 0.4191 - val\_loss: 1.7486 - val\_accuracy: 0.4208

Epoch 5/5

702/702 [==============================] - 130s 186ms/step - loss: 1.6514 - accuracy: 0.4653 - val\_loss: 1.6473 - val\_accuracy: 0.4579

In [39]:

print("\n Plotting results ... \n")

transformer\_history **=** transformer\_history**.**history

fig, axs **=** plt**.**subplots(1,2, figsize**=**(16,8))

axs[0]**.**set\_title('Loss\_Curve')

ep **=** range(len(transformer\_history['loss']))

axs[0]**.**plot(ep, transformer\_history['loss'],'o--r',label **=** 'Training\_loss')

axs[0]**.**plot(ep, transformer\_history['val\_loss'],'o--b',label **=** 'Val\_loss')

axs[0]**.**set\_xlabel('epoch')

axs[0]**.**set\_ylabel('Loss')

axs[0]**.**legend()

axs[1]**.**set\_title('Acc\_Curve')

ep **=** range(len(transformer\_history['loss']))

axs[1]**.**plot(ep, transformer\_history['accuracy'],'o--r',label **=** 'Training\_acc')

axs[1]**.**plot(ep, transformer\_history['val\_accuracy'],'o--b',label **=** 'Val\_acc')

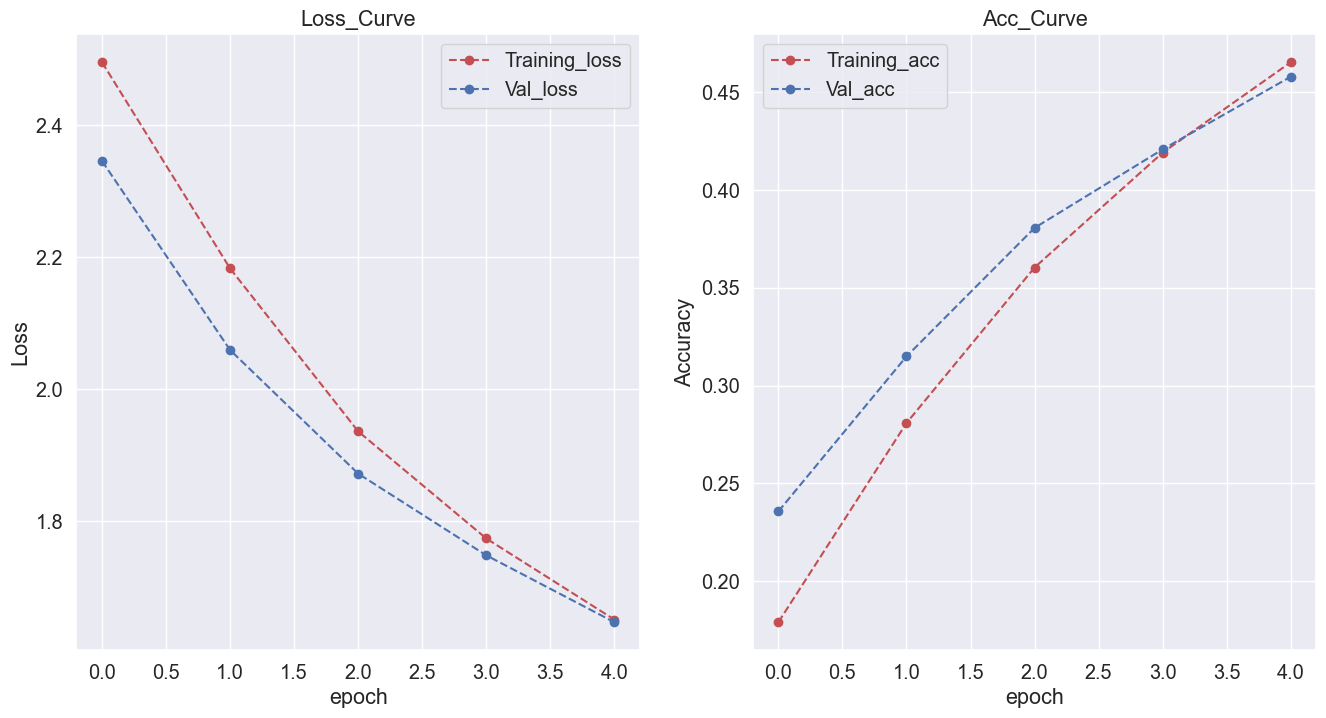
axs[1]**.**set\_xlabel('epoch')

axs[1]**.**set\_ylabel('Accuracy')

axs[1]**.**legend()

plt**.**show()

Plotting results ...



In [40]:

*# Evaluating Model*

predicted\_probabilities **=** model\_transformer**.**predict(X\_test\_Glove)

predicted\_classes **=** np**.**argmax(predicted\_probabilities, axis**=**1)

print(metrics**.**classification\_report(y\_test, predicted\_classes))

702/702 [==============================] - 7s 10ms/step

precision recall f1-score support

0 0.96 0.99 0.97 1678

1 0.88 0.95 0.91 1743

2 0.36 0.48 0.41 1796

3 0.36 0.54 0.43 1752

4 0.38 0.39 0.39 1709

5 0.29 0.24 0.26 1697

6 0.60 0.72 0.66 1680

7 0.46 0.49 0.47 1744

8 0.20 0.20 0.20 1682

9 0.39 0.39 0.39 1749

10 0.32 0.28 0.30 1747

11 0.31 0.18 0.23 1747

12 0.26 0.10 0.15 1735

accuracy 0.46 22459

macro avg 0.44 0.46 0.44 22459

weighted avg 0.44 0.46 0.44 22459

In [41]:

confusion\_matrix\_transformer **=** confusion\_matrix(y\_test, predicted\_classes)

In [42]:

plot\_confusion\_matrix(confusion\_matrix\_transformer, "Transformer", all\_classes)

