AIM: SENTIMENT ANALYSIS & TEXT CLASSIFICATION

THEORY:

SENTIMENT ANALYSIS:

Sentiment analysis (or opinion mining) is a natural language processing (NLP) technique used to determine whether data is positive, negative or neutral. Sentiment analysis is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback, and understand customer needs.

TEXT CLASSFICATION:

Text classification also known as text tagging or text categorization is the process of categorizing text into organized groups. By using Natural Language Processing (NLP), text classifiers can automatically analyze text and then assign a set of pre-defined tags or categories based on its content.

A) SENTIMENT ANALYSIS USING VADERSENTIMENT LIB: [ENGLISH]

SOURCE CODE:

```
!pip install vaderSentiment
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
     |████████████████████████████████| 125 kB 5.2 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from vaderSentiment
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requ
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from reque
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
sid_obj = SentimentIntensityAnalyzer()

# polarity_scores method of SentimentIntensityAnalyzer
# object gives a sentiment dictionary.
# which contains pos, neg, neu, and compound scores.
def senntiment_sentences(sentences):
  sentiment_dict = sid_obj.polarity_scores(sentence)

  print("Overall sentiment dictionary is : ", sentiment_dict)
  print("sentence was rated as ", sentiment_dict['neg']*100, "% Negative")
  print("sentence was rated as ", sentiment_dict['neu']*100, "% Neutral")
  print("sentence was rated as ", sentiment_dict['pos']*100, "% Positive")
  print("Sentence Overall Rated As", end = " ")

   # decide sentiment as positive, negative and neutral
  if sentiment_dict['compound'] >= 0.05 :
```

```
    print("Positive")

  elif sentiment_dict['compound'] <= - 0.05 :
    print("Negative")

  else:
    print("Neutral")


print("\n1st statement :")
sentence = "Geeks For Geeks is the best portal for the computer science engineering student
s."
senntiment_sentences(sentence)

print("\n2nd Statement :")
sentence = "study is going on as usual"
senntiment_sentences(sentence)

print("\n3rd Statement :")
sentence = "I am very sad today."
senntiment_sentences(sentence)
```

OUTPUT:

```
1st statement :
Overall sentiment dictionary is :  {'neg': 0.165, 'neu': 0.588, 'pos': 0.247, 'compound': 0.5267}
sentence was rated as  16.5 % Negative
sentence was rated as  58.8 % Neutral
sentence was rated as  24.7 % Positive
Sentence Overall Rated As Positive

2nd Statement :
Overall sentiment dictionary is :  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
sentence was rated as  0.0 % Negative
sentence was rated as  100.0 % Neutral
sentence was rated as  0.0 % Positive
Sentence Overall Rated As Neutral

3rd Statement :
Overall sentiment dictionary is :  {'neg': 0.459, 'neu': 0.541, 'pos': 0.0, 'compound': -0.5256}
sentence was rated as  45.9 % Negative
sentence was rated as  54.1 % Neutral
sentence was rated as  0.0 % Positive
Sentence Overall Rated As Negative
```

B) SENTIMENT ANALYSIS ON US AIRELINE REVIEWS:

SOURCE CODE:

```
import pandas as pd
import matplotlib.pyplot as plt

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

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense, Dropout, SpatialDropout1D
from tensorflow.keras.layers import Embedding

df = pd.read_csv("./Tweets.csv")

df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin	NaN	0
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino	NaN	0
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn	NaN	0
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino	NaN	0
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino	NaN	0

```
df.columns
```

```
Index(['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence',
       'negativereason', 'negativereason_confidence', 'airline',
       'airline_sentiment_gold', 'name', 'negativereason_gold',
       'retweet_count', 'text', 'tweet_coord', 'tweet_created',
       'tweet_location', 'user_timezone'],
      dtype='object')
```

```
tweet_df = df[['text','airline_sentiment']]
print(tweet_df.shape)
tweet_df.head(5)
```

```
(4493, 2)
```

	text	airline_sentiment
0	@VirginAmerica What @dhepburn said.	neutral
1	@VirginAmerica plus you've added commercials t...	positive
2	@VirginAmerica I didn't today... Must mean I n...	neutral
3	@VirginAmerica it's really aggressive to blast...	negative
4	@VirginAmerica and it's a really big bad thing...	negative

```
tweet_df = tweet_df[tweet_df['airline_sentiment'] != 'neutral']
print(tweet_df.shape)
tweet_df.head(5)
```

```
(3579, 2)
```

	text	airline_sentiment
1	@VirginAmerica plus you've added commercials t...	positive
3	@VirginAmerica it's really aggressive to blast...	negative
4	@VirginAmerica and it's a really big bad thing...	negative
5	@VirginAmerica seriously would pay $30 a fligh...	negative
6	@VirginAmerica yes, nearly every time I fly VX...	positive

```
tweet_df["airline_sentiment"].value_counts()
```

```
negative    2906
positive     673
Name: airline_sentiment, dtype: int64
```

```
sentiment_label = tweet_df.airline_sentiment.factorize()
sentiment_label
```

```
(array([0, 1, 1, ..., 0, 1, 1]),
 Index(['positive', 'negative'], dtype='object'))
```

```
tweet = tweet_df.text.values
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(tweet)
vocab_size = len(tokenizer.word_index) + 1
encoded_docs = tokenizer.texts_to_sequences(tweet)
padded_sequence = pad_sequences(encoded_docs, maxlen=200)
print(tokenizer.word_index)
```

{'united': 1, 'to': 2, 'the': 3, 'i': 4, 'a': 5, 'you': 6, 'and': 7, 'flight': 8, 'for': 9, 'my': 10, 'on': 11, 'is': 12, 'in': 13, 'of': 14, 'your': 15, 'it': 16, 'me': 17, 'not': 18, 'was':

```
print(tweet[0])
print(encoded_docs[0])
```

```
@VirginAmerica plus you've added commercials to the experience... tacky.
[26, 325, 413, 1047, 2058, 2, 3, 156, 2970]
```

```
print(padded_sequence[0])
```

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0   26  325  413 1047 2058
    2    3  156 2970]
```

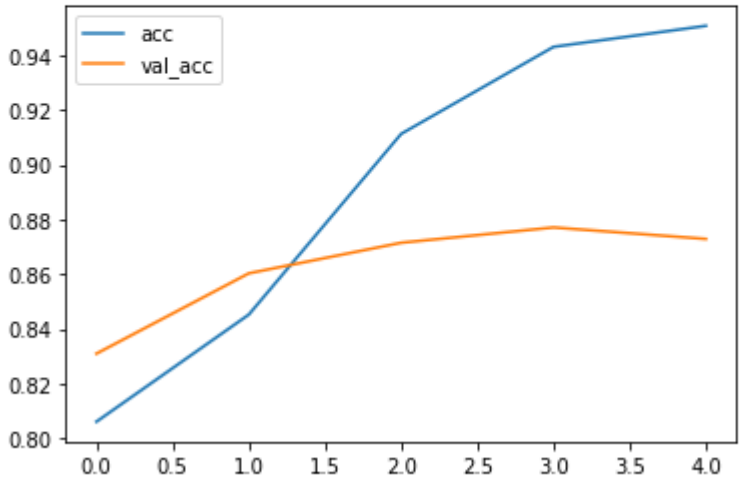
```
embedding_vector_length = 32
model = Sequential()
model.add(Embedding(vocab_size, embedding_vector_length, input_length=200) )
model.add(SpatialDropout1D(0.25))
model.add(LSTM(50, dropout=0.5, recurrent_dropout=0.5))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 32)           215520

 spatial_dropout1d (SpatialD  (None, 200, 32)          0
 ropout1D)

 lstm (LSTM)                 (None, 50)                16600

 dropout (Dropout)           (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 232,171
Trainable params: 232,171
Non-trainable params: 0
_________________________________________________________________
None
```

```
history = model.fit(padded_sequence,sentiment_label[0],validation_split=0.2, epochs=5, batch_size=32)
```

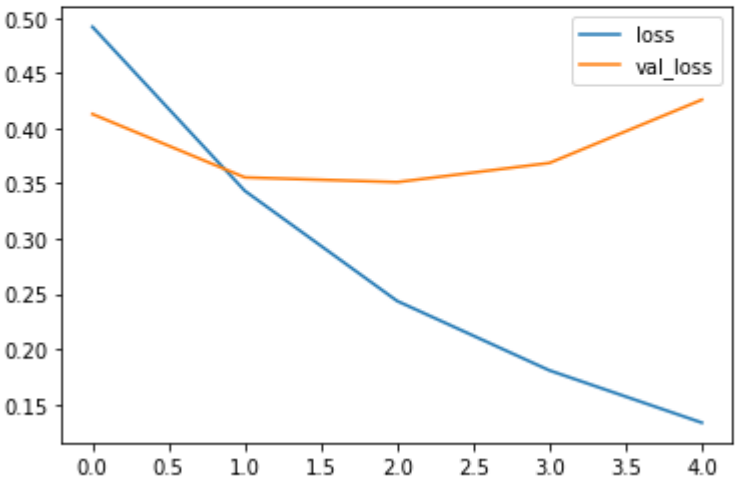
```
Epoch 1/5
90/90 [==============================] - 32s 315ms/step - loss: 0.4918 - accuracy: 0.8061 - val_loss: 0.4128 - val_accuracy: 0.8310
Epoch 2/5
90/90 [==============================] - 24s 263ms/step - loss: 0.3434 - accuracy: 0.8453 - val_loss: 0.3555 - val_accuracy: 0.8603
Epoch 3/5
90/90 [==============================] - 23s 254ms/step - loss: 0.2437 - accuracy: 0.9113 - val_loss: 0.3512 - val_accuracy: 0.8715
Epoch 4/5
90/90 [==============================] - 23s 257ms/step - loss: 0.1808 - accuracy: 0.9431 - val_loss: 0.3686 - val_accuracy: 0.8771
Epoch 5/5
90/90 [==============================] - 23s 256ms/step - loss: 0.1335 - accuracy: 0.9508 - val_loss: 0.4258 - val_accuracy: 0.8729
```

```
plt.plot(history.history['accuracy'], label='acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.legend()
plt.show()
plt.savefig("Accuracy plot.jpg")
```




```
<Figure size 432x288 with 0 Axes>
```

```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend()
plt.show()
plt.savefig("Loss plot.jpg")
```




```
<Figure size 432x288 with 0 Axes>
```

```
def predict_sentiment(text):
    tw = tokenizer.texts_to_sequences([text])
    tw = pad_sequences(tw,maxlen=200)
    prediction = int(model.predict(tw).round().item())
    print("Predicted label: ", sentiment_label[1][prediction])
```

```
test_sentence1 = "I enjoyed my journey on this flight."
predict_sentiment(test_sentence1)

test_sentence2 = "This is the worst flight experience of my life!"
predict_sentiment(test_sentence2)
```

```
Predicted label:  positive
Predicted label:  negative
```

C) SENTIMENT ANALYIS OF HINDI LANGUAGE:

SOURCE CODE:

```
!pip install deep-translator vaderSentiment
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: deep-translator in /usr/local/lib/python3.7/dist-packages (1.8.3)
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
     |████████████████████████████████| 125 kB 4.3 MB/s
Requirement already satisfied: beautifulsoup4<5.0.0,>=4.9.1 in /usr/local/lib/python3.7/dist-packages (from deep-translator) (4.11.1)
Requirement already satisfied: requests<3.0.0,>=2.23.0 in /usr/local/lib/python3.7/dist-packages (from deep-translator) (2.23.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.7/dist-packages (from beautifulsoup4<5.0.0,>=4.9.1->deep-translator) (2.3.2.post1)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.23.0->deep-translator) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.23.0->deep-translator) (2022.6.15)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.23.0->deep-translator) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.23.0->deep-translator) (1.24.3)
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
# codecs provides access to the internal Python codec registry
import codecs

# This is to translate the text from Hindi to English
from deep_translator import GoogleTranslator

# This is to analyse the sentiment of text
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Read the hindi text into 'sentences'
sentences = ['गोवा की यात्रा बहुत अच्छी रही।',
        'समुद्र तट बहुत गर्म थे।',
        'मुझे समुद्र तट पर खेलने में बहुत मजा आया।',
        'मेरी बेटी बहुत गुस्से में थी।']

# with codecs.open('SampleHindiText.txt', encoding='utf-8') as f:
#   sentences = f.readlines()
```

```
for sentence in sentences:
 translated_text = GoogleTranslator(source='auto', target='en').translate(sentence)
 #print(translated_text)
 analyzer = SentimentIntensityAnalyzer()
 sentiment_dict = analyzer.polarity_scores(translated_text)

 print("\nTranslated Sentence=",translated_text, "\nDictionary=",sentiment_dict)
 if sentiment_dict['compound'] >= 0.05 :
    print("It is a Positive Sentence")

 elif sentiment_dict['compound'] <= - 0.05 :
    print("It is a Negative Sentence")
 else :
  print("It is a Neutral Sentence")
```

OUTPUT:

```
Translated Sentence= The trip to Goa was great.
Dictionary= {'neg': 0.0, 'neu': 0.549, 'pos': 0.451, 'compound': 0.6249}
It is a Positive Sentence

Translated Sentence= The beaches were very hot.
Dictionary= {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
It is a Neutral Sentence

Translated Sentence= I really enjoyed playing on the beach.
Dictionary= {'neg': 0.0, 'neu': 0.469, 'pos': 0.531, 'compound': 0.688}
It is a Positive Sentence

Translated Sentence= My daughter was very angry.
Dictionary= {'neg': 0.473, 'neu': 0.527, 'pos': 0.0, 'compound': -0.5563}
It is a Negative Sentence
```

D) TEXT CLASSFICATION IN NLP:

SOURCE CODE:

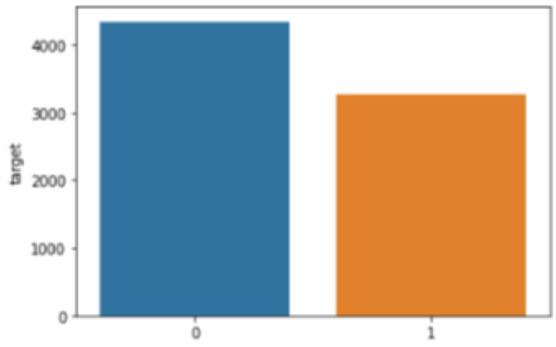
```
import pandas as pd
import numpy as np
#for text pre-processing
import re, string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('omw-1.4')
#for model-building
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, f1_score, accuracy_score, confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
# bag of words
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
#for word embedding
import gensim
from gensim.models import Word2Vec
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Unzipping corpora/omw-1.4.zip.
```

```
[2] df_train= pd.read_csv('train.csv')
    df_test=pd.read_csv('test.csv')
```

```
[3] import seaborn as sns
    x=df_train['target'].value_counts()
    print(x)
    sns.barplot(x.index,x)
```

```
0    4342
1    3271
Name: target, dtype: int64
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following va
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f1e13c88bd0>
```



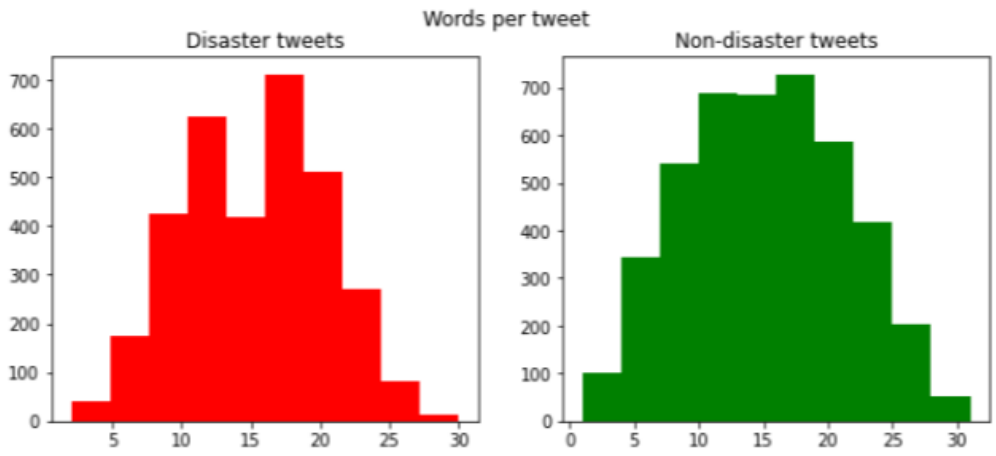

```
[4] df_train.isna().sum()
```

```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

```
[5] # WORD-COUNT
    df_train['word_count'] = df_train['text'].apply(lambda x: len(str(x).split()))
    print(df_train[df_train['target']==1]['word_count'].mean()) #Disaster tweets
    print(df_train[df_train['target']==0]['word_count'].mean()) #Non-Disaster tweets
```

```
15.167532864567411
14.704744357438969
```

```
import matplotlib.pyplot as plt
# PLOTTING WORD-COUNT
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,4))
train_words=df_train[df_train['target']==1]['word_count']
ax1.hist(train_words,color='red')
ax1.set_title('Disaster tweets')
train_words=df_train[df_train['target']==0]['word_count']
ax2.hist(train_words,color='green')
ax2.set_title('Non-disaster tweets')
fig.suptitle('Words per tweet')
plt.show()
```




```
# CHARACTER-COUNT
df_train['char_count'] = df_train['text'].apply(lambda x: len(str(x)))
print(df_train[df_train['target']==1]['char_count'].mean()) #Disaster tweets
print(df_train[df_train['target']==0]['char_count'].mean()) #Non-Disaster tweets
```

```
108.11342097217977
95.70681713496084
```


```
# CHARACTER-COUNT
df_train['char_count'] = df_train['text'].apply(lambda x: len(str(x)))
print(df_train[df_train['target']==1]['char_count'].mean()) #Disaster tweets
print(df_train[df_train['target']==0]['char_count'].mean()) #Non-Disaster tweets
```

```
108.11342097217977
95.70681713496084
```

```
#convert to lowercase, strip and remove punctuations
def preprocess(text):
    text = text.lower()
    text=text.strip()
    text=re.compile('<.*?>').sub('', text)
    text = re.compile('[%s]' % re.escape(string.punctuation)).sub(' ', text)
    text = re.sub('\s+', ' ', text)
    text = re.sub(r'\[[0-9]*\]',' ',text)
    text=re.sub(r'[^\w\s]', '', str(text).lower().strip())
    text = re.sub(r'\d',' ',text)
    text = re.sub(r'\s+',' ',text)
    return text


# STOPWORD REMOVAL
def stopword(string):
    a= [i for i in string.split() if i not in stopwords.words('english')]
    return ' '.join(a)
#LEMMATIZATION
# Initialize the lemmatizer
wl = WordNetLemmatizer()

# This is a helper function to map NTLK position tags
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
# Tokenize the sentence
def lemmatizer(string):
    word_pos_tags = nltk.pos_tag(word_tokenize(string)) # Get position tags
    a=[wl.lemmatize(tag[0], get_wordnet_pos(tag[1])) for idx, tag in enumerate(word_pos_tags)] # Map the position tag and lemmatize the word/token
    return " ".join(a)
```

```
def finalpreprocess(string):
    return lemmatizer(stopword(preprocess(string)))
df_train['clean_text'] = df_train['text'].apply(lambda x: finalpreprocess(x))
df_train.head()
```

	id	keyword	location	text	target	word_count	clean_text
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	13	deed reason earthquake may allah forgive u
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	7	forest fire near la ronge sask canada
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	22	resident ask shelter place notify officer evac...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	8	people receive wildfire evacuation order calif...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	16	get sent photo ruby alaska smoke wildfires pou...

```
[10] #SPLITTING THE TRAINING DATASET INTO TRAIN AND TEST
     X_train, X_test, y_train, y_test = train_test_split(df_train["clean_text"],df_train["target"],test_size=0.2,shuffle=True)
     #Word2Vec
     # Word2Vec runs on tokenized sentences
     X_train_tok= [nltk.word_tokenize(i) for i in X_train]
     X_test_tok= [nltk.word_tokenize(i) for i in X_test]
```

```
[10] #SPLITTING THE TRAINING DATASET INTO TRAIN AND TEST
     X_train, X_test, y_train, y_test = train_test_split(df_train["clean_text"],df_train["target"],test_size=0.2,shuffle=True)
     #Word2Vec
     # Word2Vec runs on tokenized sentences
     X_train_tok= [nltk.word_tokenize(i) for i in X_train]
     X_test_tok= [nltk.word_tokenize(i) for i in X_test]
```

```
[28] # create Word2vec model
     #here words_f should be a list containing words from each document. say 1st row of the list is words from the 1st document/sentence
     #length of words_f is number of documents/sentences in your dataset
     df_train['clean_text_tok']=[nltk.word_tokenize(i) for i in df_train['clean_text']] #convert preprocessed sentence to tokenized sentence
     model = Word2Vec(df_train['clean_text_tok'],min_count=1)  #min_count=1 means word should be present at least across all documents,
     #if min_count=2 means if the word is present less than 2 times across all the documents then we shouldn't consider it


     w2v = dict(zip(model.wv.index2word, model.wv.syn0))  #combination of word and its vector

     #for converting sentence to vectors/numbers from word vectors result by Word2Vec
     class MeanEmbeddingVectorizer(object):
         def __init__(self, word2vec):
             self.word2vec = word2vec
             # if a text is empty we should return a vector of zeros
             # with the same dimensionality as all the other vectors
             self.dim = len(next(iter(word2vec.values())))

         def fit(self, X, y):
             return self

         def transform(self, X):
             return np.array([
                 np.mean([self.word2vec[w] for w in words if w in self.word2vec]
                         or [np.zeros(self.dim)], axis=0)
                 for words in X
             ])
```

```
#SPLITTING THE TRAINING DATASET INTO TRAINING AND VALIDATION

# Input: "reviewText", "rating" and "time"
# Target: "log_votes"
X_train, X_val, y_train, y_val = train_test_split(df_train["clean_text"],
                                                  df_train["target"],
                                                  test_size=0.2,
                                                  shuffle=True)
X_train_tok= [nltk.word_tokenize(i) for i in X_train]  #for word2vec
X_val_tok= [nltk.word_tokenize(i) for i in X_val]      #for word2vec

#TF-IDF
# Convert x_train to vector since model can only run on numbers and not words- Fit and transform
tfidf_vectorizer = TfidfVectorizer(use_idf=True)
X_train_vectors_tfidf = tfidf_vectorizer.fit_transform(X_train) #tfidf runs on non-tokenized sentences unlike word2vec
# Only transform x_test (not fit and transform)
X_val_vectors_tfidf = tfidf_vectorizer.transform(X_val) #Don't fit() your TfidfVectorizer to your test data: it will
#change the word-indexes & weights to match test data. Rather, fit on the training data, then use the same train-data-
#fit model on the test data, to reflect the fact you're analyzing the test data only based on what was learned without
#it, and the have compatible


#Word2vec
# Fit and transform
modelw = MeanEmbeddingVectorizer(w2v)
X_train_vectors_w2v = modelw.transform(X_train_tok)
X_val_vectors_w2v = modelw.transform(X_val_tok)
```

```
#FITTING THE CLASSIFICATION MODEL using Logistic Regression(tf-idf)

lr_tfidf=LogisticRegression(solver = 'liblinear', C=10, penalty = 'l2')
lr_tfidf.fit(X_train_vectors_tfidf, y_train)  #model

#Predict y value for test dataset
y_predict = lr_tfidf.predict(X_val_vectors_tfidf)
y_prob = lr_tfidf.predict_proba(X_val_vectors_tfidf)[:,1]


print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       825
           1       0.77      0.74      0.75       698

    accuracy                           0.78      1523
   macro avg       0.78      0.78      0.78      1523
weighted avg       0.78      0.78      0.78      1523

Confusion Matrix: [[669 156]
 [181 517]]
AUC: 0.8509933142311366
```

```
#FITTING THE CLASSIFICATION MODEL using Logistic Regression (W2v)
lr_w2v=LogisticRegression(solver = 'liblinear', C=10, penalty = 'l2')
lr_w2v.fit(X_train_vectors_w2v, y_train)  #model

#Predict y value for test dataset
y_predict = lr_w2v.predict(X_val_vectors_w2v)
y_prob = lr_w2v.predict_proba(X_val_vectors_w2v)[:,1]


print(classification_report(y_val,y_predict))
print('Confusion Matrix:',confusion_matrix(y_val, y_predict))

fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)
print('AUC:', roc_auc)
```

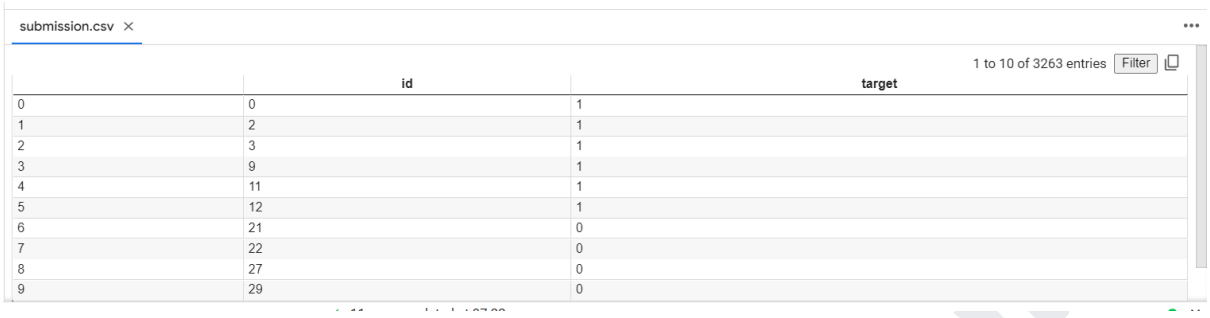
```
              precision    recall  f1-score   support

           0       0.58      0.78      0.67       825
           1       0.57      0.34      0.43       698

    accuracy                           0.58      1523
   macro avg       0.58      0.56      0.55      1523
weighted avg       0.58      0.58      0.56      1523

Confusion Matrix: [[645 180]
 [458 240]]
AUC: 0.6400329947034819
```

```
#Testing it on new dataset with the best model
df_test=pd.read_csv('test.csv')  #reading the data
df_test['clean_text'] = df_test['text'].apply(lambda x: finalpreprocess(x)) #preprocess the data
X_test=df_test['clean_text']
X_vector=tfidf_vectorizer.transform(X_test) #converting X_test to vector
y_predict = lr_tfidf.predict(X_vector)      #use the trained model on X_vector
y_prob = lr_tfidf.predict_proba(X_vector)[:,1]
df_test['predict_prob']= y_prob
df_test['target']= y_predict
print(df_test.head())
final=df_test[['id','target']].reset_index(drop=True)
final.to_csv('submission.csv')
```

```
   id keyword location                                          text  \
0   0     NaN      NaN                 Just happened a terrible car crash
1   2     NaN      NaN  Heard about #earthquake is different cities, s...
2   3     NaN      NaN  there is a forest fire at spot pond, geese are...
3   9     NaN      NaN           Apocalypse lighting. #Spokane #wildfires
4  11     NaN      NaN      Typhoon Soudelor kills 28 in China and Taiwan

                                  clean_text  predict_prob  target
0                      happen terrible car crash      0.920693       1
1  heard earthquake different city stay safe ever...      0.766494       1
2  forest fire spot pond geese flee across street...      0.820214       1
3                apocalypse light spokane wildfire      0.625511       1
4               typhoon soudelor kill china taiwan      0.991087       1
```

OUTPUT:

1- REAL DISATER, 0- NO REAL DISATER [TWEETS]



submission.csv ×

1 to 10 of 3263 entries Filter

	id	target
0	0	1
1	2	1
2	3	1
3	9	1
4	11	1
5	12	1
6	21	0
7	22	0
8	27	0
9	29	0

CONCLUSION:

From this tutorial, I have learned & implemented the sentiment analysis of text and text classification in python.