Text Analytics Pipeline

TABLE OF CONTENTS

Imports

In [1]:
```python
# #installing libraries
# %pip install fasttext
# %pip install wget
# !wget https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.1
```

In [3]:
```python
# for data handling
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
```

In [4]:
```python
# for nlp
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.tag import pos_tag
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')

# for language detection
import fasttext
import wget

# for feature extraction and expanding contractions
import gensim.downloader as gensim_api
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import BaseEstimator, TransformerMixin

# for oversampling
from imblearn.over_sampling import SMOTE

# for topic modelling
from sklearn.decomposition import NMF
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Gayathri Girish
[nltk_data]     Nair\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package stopwords to C:\Users\Gayathri Girish
[nltk_data]     Nair\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\Gayathri Girish
[nltk_data]     Nair\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to C:\Users\Gayathri Girish
[nltk_data]     Nair\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to C:\Users\Gayathri Girish
[nltk_data]     Nair\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [5]:
```python
# for ml
from sklearn.pipeline import Pipeline # to organize code into steps in
from sklearn.model_selection import train_test_split # for creating tra
from keras.models import Sequential # for neural networks
from keras.layers import Dense, Dropout # for neural networks
```

In [6]:
```python
# for model evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import label_binarize
from sklearn.metrics import silhouette_samples, silhouette_score
```

In [7]:
```python
# for visualization
import matplotlib.pyplot as plt
plt.style.use("dark_background")
import seaborn as sns
from wordcloud import WordCloud
WORD_CLOUD = WordCloud()
import pyLDAvis
import pyLDAvis.sklearn
from sklearn.decomposition import LatentDirichletAllocation as lda
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV as GSCV
```

In [8]:
```python
# to interact with file system
import os
import glob
```

In [9]:
```python
# for computational aid
import math
```

In [10]:
```python
# utility functions (!pip install ipynb)
from ipynb.fs.full.Utility_Functions import TextStyle
from ipynb.fs.full.Utility_Functions import print_list
from ipynb.fs.full.Utility_Functions import print_dict
```

In [11]:
```python
# classifiers
from sklearn.naive_bayes import BernoulliNB,GaussianNB,MultinomialNB, C
from sklearn.preprocessing import label_binarize
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
```

1. Load Data

In [12]:
```python
TWEETS_DF = pd.read_csv("data/tweets_labelled.csv")
EMOJI_DF = pd.read_csv("data/emoji_sentiment_data.csv")
GLOVE_TWITTER_MODEL_200 = gensim_api.load('glove-twitter-200')
display(TWEETS_DF.head(3))
```

	Tweet Body	TextBlob Sentiment Score	TextBlob Sentiment Label	Vader Sentiment Score	Vader Sentiment Label	Emoji Sentiment Score	Emoji Sentiment Label	Avera Sentime Sco
0	Its #Expo2020 Day	0.00000	0	0.0000	0	-2.0	-2	0.00
1	We celebrated the National Day of the Slovak R...	0.61875	1	0.9531	1	-2.0	-2	0.95
2	Vertebral Deformity Measurements on MRI, CT, a...	0.00000	0	0.0000	0	-2.0	-2	0.00

2. Data cleaning

Cleaning of tweets as performed here involves first converting tweets to lowercase, stripping them of leading/trailing whitespaces, encoding them to "utf-8" and then decoding them back before using regular expressions to match and remove undesirable parts of the text.

Various strategies of cleaning text by taking advantage of regular expressions. Popular strategies researched prior to chosen strategy include ones where all patterns to be eliminated are searched for using regular expressions such as @(\w+)|(https?)://(www)?.?(\w+).(\w+)/?(\w+)?|[0-9]|(<)|(>)|(&)|[!"$%&'()*+,-./:;<=>?[]^_`{|}~]+|\n. (Chong, 2021)

Regex search can also be performed to identify tweets with only english language content which can be achieved using regular expressions like r'^(?:[a-zA-Z]|\P{L})+$' (The Forgotten Warrior, 2020).

Here however, a simpler strategy is followed where the regex expressions try to match and remove 'all patterns other than the desired ones' instead of trying to match 'all possible undesirable patterns'. This strategy is effective with easier to comprehend regex expressions that account for all cases other than desired patterns and also filters out arabic characters in tweets since all other characters other than the desirable english alphabets are matched to be discarded.

<u>Breakdown of regular expressions used.</u>

All of the patterns mentioned below are to be mathched and substituted with "" to remove them.

- `&amp` -> special case for html embeddings; to be substituted for with "and"
- `[&@</>#]\w*` -> @username, #hashtag, html embeddings (<)|(>)|(&)
- `[^a-z {emojis_as_string}]` -> Anything that's not an english alphabet, whitespace

```
                                 or emoji (from the 'emoji_sentiment_
    ta.csv' dataset).
                                 This means all numbers, special charec
    ters and escape
                                 sequences like \n, \t etc, will matche
    d.
```

- `(https\w*)|(www\w*)` -> Any strings with "https" and "www" in it. After matches with above 2

```
                            patterns have been removed, mathing with
    his pattern will match all
                            whats left of URLs.
```

In [12]:
```python
# regular expressions to match and remove by substituting with ''
emojis_string = "".join([emoji for emoji in EMOJI_DF["Emoji"]])
PATTERNS_TO_REMOVE = [
    r'&amp', #special case for html embeddings to be substituted with "
    r'[&@</>#]\w*', # @username, #hashtag, html embeddings
    r'[^a-z {}]'.format(emojis_string), # not an english alphabet, spac
    r'(https\w*)|(www\w*)' # URL
]

def clean(text):
    ''' Function that returns a given piece of text after removing
        matches in the text with given regex patterns. Before searching
        for given regex patterns in the given text, it is converted to
        lowercase, stripped of leading/training spaces and encoded to u
        then decoded back for better readability. While looping through
        regex list, if the pattern encountered is "&amp" it is substitu
        for with "and", in all other cases, the regex is substituted fo
        @param text: The text to clean.
    '''
    text = text.lower().encode("utf-8", "ignore").decode().strip()
    for pattern in PATTERNS_TO_REMOVE:
        if pattern=="&amp": text = re.sub(pattern, "and", text)
        else: text = re.sub(pattern, "", text)
    return text
```

3. Expanding Contractions

(Lukei, 2019), (Radimrehurek, 2021)

- There is really no strict format of writing to which Twitter users are requested to adhere to. In addition, due to each tweet having a maximum length of 240 characters, users resort to using contractions to shorten their sentences to fit within that specified limit. Contractions represent a combination words, shortened by omitting some letters and adding an apostrophe instead. This shortens those combination of words into something much more compact while retaining the same meaning. E.g. "she'd" is "she had" or "she would" depending on the sentence the contraction is in as the context matters when expanding the contractions. Computers are not aware of these contracted forms of words are and so presenting tweets full of contractions would not help the machine learning model. It would increase the dimensionality of the feature space as a computer would not be able to make the connection that "it'll" is the same as "it will".

- The initial approach to tackle this issue was creating a dictionary with a list of all the possible contractions available. However, it would require a sophisticated solution to in order to understand the tense/context of the sentence and expand the contractions correctly. As we saw above, "she'd" can be expanded to either "she had" or "she would" and the expanded contraction depends on the tense of the sentence the contraction existed in.

- A python package called "pycontraction" was the solution to this. The package recognizes the contractions in the text and finds its corresponding expanded form. If there are multiple expanded forms of the contraction, as previously shown, then all possible texts are produced. E.g. if "she'd bought it already" is the sentence given, the output hypotheses will be "she had bought it already" and "she would bought it already".

- These possible outputs will be further examined using to a grammar checker and the Word Mover's Distance is calculated for each sentence.

- Briefly, what Word Mover's Distance does is its calculate the distances between the different sentences using word embeddings by matching the relevant words in the sentences together. It does this to find the similarity between the sentences. In our case, we need it is used to showcase how similar the potential sentence is to the original text that is being expanded.

- In the case where the contraction has one possible form of expanding then the sentence that is grammatically sound is chosen. If there are multiple ways of expansion, then the sentence with the highest Word Mover's Distance score is selected, i.e. the sentence is the most similar to the original text is chosen. Between "she had already bought it" is the grammatically sound sentence and is the sentence with the highest WMD score, because this is what the package gives as a final output.

- "she would have bought it already" is the grammatically sound sentence when the contraction is in the following sentence "she'd have bought it already", where the presence of "have" changes the grammar.

In [13]:

```python
from pycontractions import Contractions
CONTRACTIONS_EXPANDER = Contractions(kv_model = GLOVE_TWITTER_MODEL_200
def pycontract(text):
    ''' Function that returns the expanded contractions within text.
        @param text: Text that includes contractions to be performed up
    '''
    if not type(text) == list: text = [text]
    expanded = list(CONTRACTIONS_EXPANDER.expand_texts(text, precise =
    return (" ".join(expanded))

print(pycontract(["she'd bought it already"]))
print(pycontract(["she'd have bought it already"]))
```

```
she had bought it already
she would have bought it already
```

Another package that is available is called "contractions", however, the accuracy produced by the results of "pycontractions" were the reasons, it was ultimately chosen, as Lukei (2019) has noted that "pycontractions" performed with 89% accuracy than the "contractions" package was 10% shy off.

OBSERVATION: As shown in the screenshot below, the package managed to successfully expand the contractions given to it. However it did expand the "r", the shortened version of "are". "pycontractions" faced the same issues with this type of shortened language, also known as slang. These slangs, especially when a word is shortened why beyond what its original form looked like, e.g. cld which stands for could, were not recognized. Other informal contractions were not expanded using pycontractions, e.g. whatcha doing remained after expanding as whatcha doing. This posed as an issue, as the language used on social media often included these words typed out in these forms. Having them not expand to their original form meant that the semantics of sentences would change drastically when a computer is to interpret it.

In [14]:

```python
print(pycontract(["They're going to the zoo"]))
print(pycontract(["They'll come"]))
print(pycontract(["How r u? I've heard you ain't been well."]))
```

```
they are going to the zoo
they will come
How r you? I have heard you have not been well.
```

Thus, a dictionary with most common contractions were compiled from sources online to be used to find and replace contractions.

In [13]:
```python
# dictionary terms obtained from the following sources:
# 1. https://www.webopedia.com/reference/twitter-dictionary-guide/
# 2. https://stackoverflow.com/questions/19790188/expanding-english-lar

CONTRACTIONS = {
    "aint": "is not",
    "arent": "are not",
    "cant": "cannot",
    "cantve": "cannot have",
    "cause": "because",
    "couldve": "could have",
    "couldnt": "could not",
    "couldntve": "could not have",
    "didnt": "did not",
    "doesnt": "does not",
    "dont": "do not",
    "hadnt": "had not",
    "hadntve": "had not have",
    "hasnt": "has not",
    "havent": "have not",
    "hed": "he had",
    "hedve": "he would have",
    "hell": "he will",
    "hellve": "he will have",
    "hes": "he has / he is",
    "howd": "how did",
    "howdy": "how do you",
    "howll": "how will",
    "hows": " how is",
    "id": "I would",
    "idve": "I would have",
    "ill": "I will",
    "illve": "I will have",
    "im": "I am",
    "ive": "I have",
    "isnt": "is not",
    "itd": "it would",
    "itdve": "it would have",
    "itll": "it will",
    "itllve": "it will have",
    "its": "it is",
    "lets": "let us",
    "maam": "madam",
    "maynt": "may not",
    "mightve": "might have",
    "mightnt": "might not",
    "mightntve": "might not have",
    "mustve": "must have",
    "mustnt": "must not",
    "mustntve": "must not have",
    "neednt": "need not",
    "needntve": "need not have",
    "oclock": "of the clock",
    "oughtnt": "ought not",
    "oughtntve": "ought not have",
    "shant": "shall not",
    "shant": "shall not",
    "shantve": "shall not have",
    "shed": "she would",
    "shedve": "she would have",
    "shell": "she will",
    "shellve": "she will have",
    "shes": "she is",
    "shouldve": "should have",
    "shouldnt": "should not",
    "shouldntve": "should not have",
    "sove": "so have",
    "sos": "so is",
    "thatd": "that would",
    "thatdve": "that would have",
    "thats": "that is",
    "thered": "there would",
    "theredve": "there would have",
    "theres": "there is",
    "theyd": "they would",
    "theydve": "they would have",
    "theyll": "they will",
    "theyllve": "they will have",
    "theyre": "they are",
    "theyve": "they have",
    "tove": "to have",
    "wasnt": "was not",
    "wed": "we would",
    "wedve": "we would have",
    "well": "we will",
    "wellve": "we will have",
    "were": "we are",
    "weve": "we have",
    "werent": "were not",
    "whatll": "what will",
    "whatllve": "what will have",
    "whatre": "what are",
    "whats": "what is",
    "whatve": "what have",
    "whens": "when is",
    "whenve": "when have",
    "whered": "where did",
    "wheres": "where is",
    "whereve": "where have",
    "wholl": "who will",
    "whollve": "who will have",
    "whos": "who is",
    "whove": "who have",
    "whys": "why is",
    "whyve": "why have",
    "willve": "will have",
    "wont": "will not",
    "wontve": "will not have",
    "wouldve": "would have",
    "wouldnt": "would not",
    "wouldntve": "would not have",
    "yall": "you all",
    "yalld": "you all would",
    "yalldve": "you all would have",
    "yallre": "you all are",
    "yallve": "you all have",
    "youd": "you would",
    "youdve": "you would have",
    "youll": "you will",
    "youllve": "you will have",
    "youre": "you are",
    "youve": "you have",
    "abt": "about",
    "ab": "about",
    "bc": "because",
    "cos": "because",
    "coz": "because",
    "b4": "before",
    "bfn": "bye for now",
    "bgd": "background",
    "br": "best regards",
    "btw": "by the way",
    "chk": "check",
    "clk": "click",
    "cld": "could",
    "cre8": "create",
    "da": "the",
    "dm": "direct message",
    "em": "email",
    "eml": "email",
    "ema": "email address",
    "f2f": "face to face",
    "ftl": "for the loss",
    "ftw": "for the win",
    "ic": "I see",
    "idk": "i dont know",
    "kk": "okay",
    "k": "okay",
    "c": "see",
    "rt": "retweet",
    "mrt": "modified retweet",
    "nts": "note to self",
    "prt": "please retweet",
    "smh": "shaking my head",
    "sp": "sponsored",
    "tbh": "to be honest",
    "tftf": "thanks for the following",
    "tmb": "tweet me back",
    "u": "you",
    "woz": "was",
    "wtv": "whatever",
    "yolo": "you only live once",
    "yoyo": "you are on your own"
}

def expand_contractions(text): # DO AFTER CLEANING
    ''' Function that returns the given text with contractions expanded
        @param text: Text that includes contractions to be performed up
    '''
    for key, val in CONTRACTIONS.items():
        re.sub(pattern=f"\\b{key}\\b", repl=val, string=text)
    return text
```

4. Tokenization

Tokenization in Natural Language Processing (NLP) refers to the process of splitting the raw input text into words or terms called “tokens”. These “tokens” could be emojis, special characters, words, numbers, etc., and are used in the lemmatization and stemming process.

By analyzing this list of “tokens” generated from the tokenization process, we can interpret the context of the input text. (Chakravarthy, 2020)

An example of tokenization in NLP is given below: Consider the string “Twitter Sentiment Analysis”. After tokenization, we get `[‘Twitter’, ‘Sentiment’, ‘Analysis’]`. Tokenization is an essential step to be performed during the pre-processing stage, as the “tokens” generated are vectors that represent the input text. These vectors can then be used by a Machine Learning/NLP pipeline to produce desired results. (neptune.ai, 2021)

Libraries used in Tokenization

Natural Language Toolkit (NLTK) and spaCy are the most popular Python libraries used in various NLP tasks such as tokenization, stemming, lemmatization, etc.

The Natural Language Toolkit (NLTK) is an NLP library developed by researchers at Microsoft (Chakravarthy, 2020). There are multiple uses of this library, out of which we have used the module `tokenize`, which offers the function `word_tokenize()`. With the help of this function, we have split each of the multiple tweet bodies into list of tokens.

Reasons for choosing NLTK over spaCy

(Igor Bobriakov, 2018), (Zubair Hossian, 2022)

- The user has more flexibility with NLTK; hence, it is more suitable for students and academic researchers. This is because one can try out new strategies by combining various functions or algorithms for a task. However, spaCy is regarded to be rigid since it automatically chooses an algorithm for implementing a task.
- NLTK offers numerous functions and modules for multiple NLP tasks.
- As it is the most used and one of the oldest NLP Python libraries, NLTK has more documentation available.

The procedure followed for Tokenization

- The function `tokenize()` takes in a tweet and passes it to NLTK’s function, `word_tokenize()`.
- The `word_tokenize()` function is used to perform tokenization, and it generates a list of tokens for the tweet. The function then returns this list.

In [14]:
```python
def tokenize(text):
    ''' Function that tokenizes a piece of text and removes
        stop words from it before returning the list of tokens
        obtained.
        @param text: The text to tokenize.
        @returns: Given text as list of tokens.
    '''
    tokens = word_tokenize(text) # alternatively: text.split() # (Diego
    return tokens
```

5. Stop Word Removal

Words such as “the”, “me”, “is”, “because”, etc. are called **stop words**, as they do not add much information to a text/sentence. While pre-processing a raw text, such words can be removed without changing the meaning of the text. (Teja, 2020)

Removing stop words from the raw input text is a crucial step in text pre-processing so that the NLP pipeline can focus on only the important information in a text. (Khanna, 2021)

Should stop words always be removed?

Since we are analyzing the sentiment of multiple tweets, removing certain stop words can completely change the sentiment of the text. An example of this problem is as shown below:

Before removal of stop words	After removal of stop words
The painting is really beautiful **(Positive)**	painting really beautiful **(Positive)**
The movie was boring **(Negative)**	movie boring **(Negative)**
The food was not tasty **(Negative)**	food tasty **(Positive)**
I didn't like the show **(Negative)**	like show **(Positive)**

From the above example, we can see statements such as “The food was not tasty” was clearly negative. However, after the removal of the stop words, the statement became positive (“food tasty”). This can negatively affect the results of the sentiment analysis problem.

The procedure followed for the removal of Stop Words

- As mentioned previously, removing certain stop words such as “didn’t”, “not”, etc. can affect the results while analyzing the sentiments of the tweets.
- To fix this issue, a set called `STOP_WORD_EXCEMPTIONS` was initialized. To this set, we have added terms or words which play a vital role in deciding whether the sentiment is positive or negative.
- NLTK’s stop words list was used to get the list of popular stop words. Since the program only deals with English tweets, we set the language to English for the `stopwords.words()` . The stop words were then stored in a set to get the unique values.
- To exclude the set of stop words present in `STOP_WORD_EXCEMPTIONS` , we subtract this set from NLTK’s set of stop words. The result is stored in a set `STOP_WORDS` .
- We have defined a function, `remove_stopwords()` , to remove the stop words present in a tweet body.
- The function iterates through each of the words present in a tweet body and checks if the current word is not present in our new set of stop words, i.e., `STOP_WORDS` .
- If the word is not present in `STOP_WORDS` , then append the word to a list and return the list. This list will contain all the words from the tweet body, except the stop words.

By performing the above steps, we removed all the stop words present in each tweet body. As mentioned previously, the code does not remove certain stop words that have an essential role in the sentiment analysis problem.

In [15]:
```python
STOP_WORD_EXCEMPTIONS = set([ #(Teja, 2020)
    "isnt", "didnt", "dont", "couldnt", "wont",
    "wouldnt", "not", "doesnt", "shouldnt", "wasnt"
])
STOP_WORDS = set(stopwords.words('english')) - STOP_WORD_EXCEMPTIONS #(

def remove_stopwords(word_list): # (M.D. Pietro, 2017), (GeeksforGeeks,
    ''' Function that removes stopwords from a given list of words
        and returns this new possibly smaller list.
        @param text: The list of words from which to remove stopwords f
    '''
    return [word for word in word_list if not word in STOP_WORDS]
```

(Bengfort, Bilbro and Ojeda, 2018, p. 72)

In this coursework, we deal primarily with text and to keep only the most important words for sentiment analysis it benefits to normalise the text. Text normalisation allows for the feature space to reduce and thus allowing the data to be more informative. (Bengfort, Bilbro and Ojeda, 2018, p. 72) This helps improve the generalization of machine learning algorithms.

6. Stemming

6.1. What Is Stemming?

(Bernardo, 2021), (Elia, 2020)

- Stemming is the process of removing the affixes/inflections that modify the meaning of a word to bring it to its stem. Usually this happens through a series of sequential steps each based on a fixed rule to trim the word to its stem.
- It is also a text normalization technique that enables the standardization of words that are derivationally related. E.g. a word has multiple forms like a noun, adjective, verb, adverb, etc, so "shovels" "shoveled" and "shoveling" although they are different tenses, once they are stemmed we'll end up with "shovel" which is the common "stem" for these different forms. It also helps to standardize words that share the same suffix. E.g. removing "s" or "es" (indicate plurality) from words to bring word to their singular form. (Bernardo, 2021).
- In morphology, a stem is a form of the word before the addition of any affixes.
- Thus, stemming might produce stems that are invalid words on their own, i.e. lexicographically incorrect/will not show up in a dictionary. However, that does not mean that the process of stemming is not helpful, because the idea is to have the variations of the same word map to the same stem. (Elia, 2020)
- E.g. common removed suffixes are "s" or "es" that indicate plurality, so "cats" becomes "cats" and "eats" becomes "eat". "oranges" becomes "orang" just because the word ends in "es". This is an example of when the word stem produced is not a valid word. Although "orang" is an invalid word, what is important is that both words, in our case "orange" and "oranges" map to the same stem, "orang". (Elia, 2020)
- For example, we may have a suffix rule that, based on a list of known suffixes, cuts them off. In the English language, we have suffixes like “-ed” and “-ing” which may be useful to cut off in order to map the words “cook,” “cooking,” and “cooked” all to the same stem of “cook.”

6.2. Types Of Stemmers

There are different types of stemmers or stemming algorithms and the ones that will be discussed are the ones that are most common and supported by NLTK library. They are: Porter Stemmer, Snowball stemming language and Lancaster Stemmer.

6.2.1 Porter Stemmer

(Bernardo, 2021), (Jabeen, 2018)

- The Porter Stemmer is limited to English words. It uses a set of five phases, where each phase has its own set of rules to allow the identification and the removal of inflections to bring words to their stems. The stemmer is applied sequentially. The five phases are as follows:
  - Phase 1: is where the plural form and the past participle is removed. E.g. plural common suffixes are: "s", "es" and "ies". The stemmer will look for these common endings in words and once they are identified, they are removed to bring words to their stems.
  - Phase 2 to 5: based of the different combinations of preceding consonants and vowels that a word is made up of, the stemmer will remove the word endings. E.g. "activate" becomes "activ" / "demonstrates" becomes "demonstr"
- No lookup tables of the stems of each word is kept, this stemmer merely strips suffixes based on the rules defined for it. This is one of its biggest issues as it sometimes produces non-semantic words as stems but it is also one of its biggest benefits as this method allows the stemmer to be very quick and efficient. Having a look up table would increase the time it would require to complete the process of stemming. (Jabeen, 2018)

In [16]:
```python
STEMMER = nltk.stem.porter.PorterStemmer()
def stem(word_list): # (M.D. Pietro, 2017)
    ''' Function that returns a given list of words after stemming.
        @param word_list: List of words to perform stemming upon.
    '''
    global STEMMER
    return [STEMMER.stem(word) for word in word_list]
```

In [17]:
```python
# All the examples listed above with actionable code
print("The stem of shoveled, shoveling, shovels is ", stem(["shoveled",
print("cats after stemming becomes ", STEMMER.stem("cats"))
print("eats after stemming becomes ", STEMMER.stem("eats"))
print("The stem of oranges and orange is ", stem(["oranges", "orange"])
print("The stem of cook, cooking and cooked is ", stem(["cook", "cookin
print("active after stemming becomes ", STEMMER.stem("active"))
print("demonstrates after stemming becomes ", STEMMER.stem("demonstrate
```

```
The stem of shoveled, shoveling, shovels is  ['shovel', 'shovel', 'shove
']
cats after stemming becomes  cat
eats after stemming becomes  eat
The stem of oranges and orange is  ['orang', 'orang']
The stem of cook, cooking and cooked is  ['cook', 'cook', 'cook']
active after stemming becomes  activ
demonstrates after stemming becomes  demonstr
```

6.2.2 Snowball

(Bernardo, 2021), (Snowball Stem, n.d.)

- Snowball is a language for processing small strings to create stemming algorithms. In NLTK, this language is used to build the Snowball Stemmer. It is based on the original Porter Stemmer and is thought to be an upgraded version of it. Due to these optimizations, the Snowball Stemmer is speedier than the Porter Stemmer.
- It follows the same procedure of the Porter Stemmer with the difference being that this stemmer also works with other languages. As we are only dealing with English text within the tweets, the multi-lingual capabilities of this stemmer were not necessary.
- There were difference between the Porter Stemmer and Snowball Stemmer. While the Porter Stemmer rules prevent overstemming from happening, they still occur. E.g. the words "generically" and "generous" are both returned with stem "gener" when using the Porter Stemmer. Whereas using the Snowball Stemmer, "generically" is returned as "generic" and "generously" is returned as "generous". The Snowball's method retains both words' distinctive meaning whereas Porter's stemmer overstems and reduces two different words to the same stem.
- Snowball stemmer is better on normalizing some adverbs. In the case with adverbs, if the word "warmly" is to undergo stemming, in the Snowball stemmer it would be "warm" whereas in Porter's Stemmer it would be "warmli".

In [18]:
```
print("generically and generous after using Porter stemmer become: ", S
SNOWBALL_STEMMER = nltk.stem.SnowballStemmer(language="english")
print("generically and generous after using Snowball stemmer become: ",
print("readily after using Porter stemmer becomes: ", STEMMER.stem("war
print("readily after using Snowball stemmer becomes: ", SNOWBALL_STEMME
```

```
generically and generous after using Porter stemmer become:  gener gener
generically and generous after using Snowball stemmer become:  generic gen
erous
readily after using Porter stemmer becomes:  warmli
readily after using Snowball stemmer becomes:  warm
```

6.2.3 Lancaster Stemmer

(Jabeen, 2018)

- Is the most aggressive stemmer from the two stemmers discussed previously. It is an iterative algorithm based on a set of externally rules saved. The rules are indexed by the last letter of a suffix and each rule indicates either deleting or replacing the ending of the word. On every iteration, the algorithm tries to find a rule based on the last letter of a word, if a rule is found and deleting nor replacing the ending is specified the algorithm terminates. If a word begins with a vowel and there are two constants left (e.g. ate) or if a word starts with a constant and 3 letters are left, then the algorithm terminates.
- It is aggressive as it overstems the words to produce the shortest stem possible. Using the example of "generous" and "generic", the Lancaster stemmer, will produce the following stems for the two respective words: "gen" and "gen". The meaning of both words has been erased in the stemming procedure.
- The Lancaster stemmer may be used to count the occurrence of a certain stem e.g. "giv" from "giving, given, gives" etc.
- One of the benefits the Lancaster Stemmer offers in the option to create a new set of rules.

In [19]:
```
LANCASTER_STEMMER = nltk.stem.LancasterStemmer()
print("generically and generous after using Lancaster stemmer become: "
print("giving, given and gives after using Lancaster stemmer become: ",
```

```
generically and generous after using Lancaster stemmer become:  gen gen
giving, given and gives after using Lancaster stemmer become:  giv giv giv
```

6.3. Overstemming Vs Understemming?

(Srinidhi, 2020)

- Overstemming happens when a word is trimmed more than is necessary. This results in stems that lose their meanings and enables different words to incorrectly share the same stem when these words should each have different stems. E.g. "university" and "universe" are two separate words. In the case of overstemming, they both get reduced to share the share stem "univers" thus implying that those different words represent the same thing.
- Understemming, is the opposite idea to overstemming. The words that share the same meaning after being stemmed do not end up sharing the same stem, but have different stems. E.g. "data" and "datum", data is the plural form of datum but they mean the same thing. Some stemming algorithms produce the following as the stem" "dat" and "datu" respectively. They should be reduced to the same stem being "dat".
- Therefore, it is very important to choose a stemming algorithm carefully.

After reviewing the three stemmers, the aggressive nature of the Lancaster Stemmer does not fit with the purpose of text analysis where having a resultant meaningful word is key. Between the Porter Stemmer and the Snowball Stemmer, the Snowball Stemmer would be the likely choice to go for. However, as stemming is not the text normalization technique that consistently provided lexicographical roots/valid words this will hinder the process of analyzing the sentiment of words. Thus, lemmatization will be the technique of choice to proceed with to normalize the data gathered.

7. Lemmatization

7.1. What Is Lemmatization?

(Jabeen, 2018), (Bengfort, Bilbro and Ojeda, 2018, p. 72)

- Lemmatization is the process of reducing words to their base forms/roots. It does this by taking into consideration a language's full vocabulary, where it looks up the words in the dictionary/database to return their morphological roots called lemmas. Thus, a lemma will always be a semantically sound word as it is listed in the dictionary.
- It also takes into account a word's part of speech. This is because the part of speech showcases what a word means in sentences as well as its grammatical purpose. A word can be a noun, an adjective, an adverb, a verb, etc. This is because the context in which the lemmatization will happen in is given. If the context is not given, no actual root form will be given to the words within the text. (Jabeen, 2018) This will be discussed in details in the following section.
- This will change how the word looks and what will be removed from the word when

lemmatization occurs. This process takes uses NLTK's part-of-speech tagger. E.g. "is" and "are" become "be"

- However, this very fact, makes it slower than stemming. The look-up aspect and tagging the token with parts-of-speech to help in identifies the lemma of word is time consuming however it is effective and informative as it retains the original meaning of the word. (Bengfort, Bilbro and Ojeda, 2018, p. 72)
- `print(lemmatizer.lemmatize("are"))` are does not change because the default for the lemmatizer is noun.

In [20]:
```
# without declaring the POS tag
print("'is' and 'are' without the POS tag are lemmatized as: ", nltk.st
print("'is' and 'are' with the POS tag are lemmatized as: ", nltk.stem.
```

```
'is' and 'are' without the POS tag are lemmatized as:  is are
'is' and 'are' with the POS tag are lemmatized as:  be be
```

7.2. POS Tagging To Improve Lemmatizing/Get Proper Lemma

In [23]:
```
def tag_token(tag):
    ''' Function that returns a parts of speech tag that the lemmatizer
        @param tag: parts of speech tag.
    '''
    # Since the pos param only takes the following:
    #   n (noun), v (verb), r (adverb), a (adjectives), s (satellite ad
    if tag.startswith("N"): return wordnet.NOUN
    elif tag.startswith("V"): return wordnet.VERB
    elif tag.startswith("R"): return wordnet.ADV
    elif tag.startswith("J"): return wordnet.ADJ
    return wordnet.NOUN # the default tag is noun
```

In [24]:
```
POS_TAG = nltk.tag.pos_tag
def get_pos(tokens):
    ''' Function that returns a given list of words and their respectiv
        @param word_list: List of words to perform POS tagging upon.
    '''
    global POS_TAG
    return [POS_TAG([word])[0] for word in tokens]
```

In [25]:
```
LEMMATIZER = nltk.stem.wordnet.WordNetLemmatizer()

def lemmatize(word_list):
    ''' Function that returns a given list of words after lemmatizing
        each word in that list.
        @param word_list: List of words to lemmatize.
    '''
    global LEMMATIZER
    tagged_entry = get_pos(word_list)
    return [LEMMATIZER.lemmatize(word, tag_token(tag)) for (word, tag)
```

(Gupta, 2020), (Patel, 2020), (Princeton University, n.d.), (W3Schools, n.d.)

- There exists eight parts of speech in the English language, the lemmatizer recognizes 4 of them. These 4 are: nouns, verbs, adverbs, adjectives. (Patel, 2020)
- Initially, the lemmatization process was done without part-of-speech tagging any of the words within the tweets. This meant that the lemmatizer, by default, assumed that all

the tokenized text passed to it were nouns, when in reality they were not. This was an ineffective approach to lemmatization to take as a large amount of words were not normalized and thus they existed and expanded the feature space of our data set.

- NLTK provides 12 tag classes.
- To find the POS tag of a word, the word has to have been cleaned and tokenized. The `word_tokenize()` function can be used to tokenize the words. Then function `pos_tag()` is used to return a list of tuples with each word and its respective POS tag. E.g. `test = word_tokenize("morning world!"); nltk.pos_tag(text)`

In [21]:
```
test_1 = word_tokenize("morning world!")
nltk.pos_tag(test_1)
```

Out[21]:
```
[('morning', 'NN'), ('world', 'NN'), ('!', '.')]
```

- Following that same idea, every tweet was POS tagged. However, due to the tweets having undergone cleaning, there were many words in between that have gone missing as they were for example written as a hashtag. E.g. writing a word as `#Expo` instead of `Expo` . This meant that depending on the available POS tagger to dissect the context of the sentence and tag every word accordingly was going to provide inaccurate results since sentences were missing many contextual clues. E.g. as shown below.

In [22]:
```
test_2 = ['celebrated', 'national', 'day', 'slovak', 'republic', 'expo'
nltk.pos_tag(test_2)
```

Out[22]:
```
[('celebrated', 'JJ'),
 ('national', 'JJ'),
 ('day', 'NN'),
 ('slovak', 'VBZ'),
 ('republic', 'JJ'),
 ('expo', 'NN'),
 ('dubai', 'NN'),
 ('honoured', 'VBD'),
 ('welcome', 'JJ'),
 ('excellency', 'NN'),
 ('zuzana', 'NNP'),
 ('caputova', 'NN'),
 ('president', 'NN'),
 ('slovak', 'VBZ'),
 ('republic', 'JJ'),
 ('attended', 'VBD'),
 ('ceremony', 'NN'),
 ('met', 'VBD'),
 ('excellency', 'NN'),
 ('sarah', 'NN'),
 ('al', 'NN'),
 ('amiri', 'NN')]
```

- To tackle this issue, the function `get_pos()` was built that will take each token and return the part of speech associated with the token. If the token was "jogging" then the function would return verb. `pos_tag(["jogging"])` . As the POS tagger returns the tagged text in the format of `('VRB', 'jogging')` , the only relevant information is the POS tag so the function returns this.
- Since the lemmatizer only recognizes a small subset of all the existing POS tag set, it meant that the POS tags that will be passed to the lemmatizer have to be ensured to be

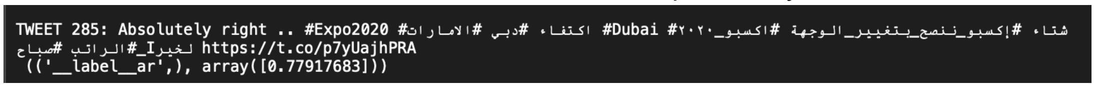
ones it can recognize. Thus function `tag_token()` was created. It would identify if the tag passed to it, starts with a letter that represented one of the categories recognized; if so, those categories were returned. If not, then the POS tag was returned to be a noun as that is the default tag of a lemmatizer.

- Thus, the tweets would be lemmatized word by word, and for each word the POS tag was found and converted to a tag that lemmatizer recognizes and can work with.
- Now, although the lemmatizer recognizes satellite adjective, the universal part-of-speech tag set does not include it as part of the set. In essence, an adjective can be categorized into "head" and "satellite" synsets. (Princeton University, n.d.) Thus, in the practical uses, we can view it as a semantic label.

8. Language Detection/English Filtering

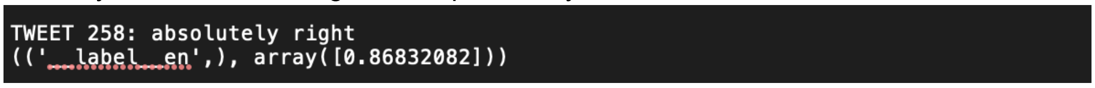
- Although, when collecting tweets to build our dataset, we have specifically set filtering options to only return English tweets as discussed in Part A's documentation. That was not a sufficient measure as some of the tweets gathered include foreign words written in English. E.g. `Assalamu alaikum, maza aa gaya, etc.` These foreign words would affect the word embeddings to be in the later stage.
- JCharisTech (2021) has presented a set of possible packages that perform detect the language. The idea is if the language detector can detect the foreign words in the tweets and categorize the tweet as non-English, we will end up with a data set clear comprising of full English tweets.
- The majority of these packages were tested with tweets that either fully compromised of English words and ones that contained non-English words, i.e. words that are not in the English dictionary. The accuracy levels produced by each package's analysis of the given input were compared and the package with the highest accuracy levels and thus best language detection was selected. The tests were run on Google Colab as it did not require installing the packages locally. This was a prevalent issue, where packages could not be installed due to requiring additional tools to be installed to function.
- "fasttext" was the package selected due to its speed and accuracy. It had managed to detect correctly and to a high accuracy when the sentences were non-English. E.g. It detected Italian and Spanish tweets that existed in the dataset. However, in the case of the tweets that contained foreign terms, they largely consisted of English words therefore they were viewed as English text.
- The `predict()` method of "fasttext" returned the highest probable language (represent in the ISO 639 code) for a given input and the numeric probability of the sentence belonging to that language. `(('__label__en',), array([0.47550538]))`.
- Using this output, the model was tweaked to return output given a specific estimated probability. The hypothesis was, if the estimated probability is set to high values e.g. 60 or 70, the model will return back the tweets in which it is 60% to 70% sure they are English. This will be a high enough threshold to eliminate the presence of tweets with foreign terms. This experiment was conducted on the original form of tweets (i.e. tweets that did not undergo cleaning), tweets that have been cleaned but not tokenized and lemmatized tweets (i.e. tweets that were pre-processed and undergone lemmatization). In addition, the model was set to return only English labelled tweets with the given estimated probability.
- The results on the set of tweets that were in their original form were not good. Due to the tweets not having been cleaned, there remained many unnecessary things such as hashtags. For example, the following tweet contained many Arabic text in the form of

hashtags. Although there are English words within the tweet, due to the majority being Arabic, the model identified the tweet with Arabic with probability of 77%.



```
TWEET 285: Absolutely right .. #Expo2020 #الامارات #دبي #اكتفاء #Dubai #اكسبو_٢٠٢٠ #إكسبو_ننصح_بتغيير_الوجهة #شتاء
#صباح #الراتب_#I_لخير https://t.co/p7yUajhPRA
 (('__label__ar',), array([0.77917683]))
```

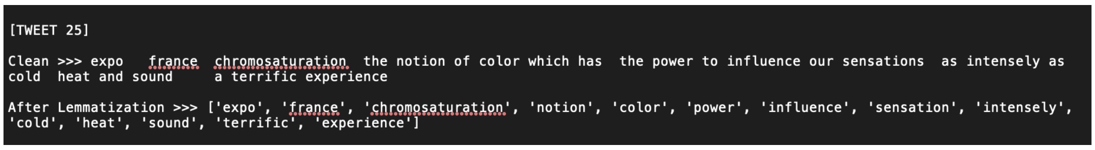
The tweet after being cleaned consists only of English words and thus the model correctly identifies it as English with probability 86%.



```
TWEET 258: absolutely right
(('__label__en',), array([0.86832082]))
```

Therefore, due to such results, using the original form of the tweets was ruled out from usage.

- The results on the set of tweets that were cleaned performed were the best out of the other options. Initially, the estimated probability was set to high values e.g. 70% as that was the range at which the tweets that contained the foreign words were detected at. Thus, the threshold has to be that high in order to allow the majority of English tweets to remain. However, this came with a bigger issue and that was many perfectly English tweets were removed as well. Around 1000 tweets were eliminated in this process and this made our data set significantly smaller. Thus, the estimated probability was lowered to 45% as that is where the balance was created between the tweets between the number of tweets being removed (around 300 tweets were removed) and the number of tweets being kept.
- The results on the set of lemmatized tweets mimicked the results on the cleaned tweets but with a generally lower thresholds of estimated probabilities across all tweets. E.g. The same tweet had different estimated probabilities given its language. The cleaned tweet scored more than the lemmatized tweet. This is because the stop words are removed in the lemmatized tweet whereas the cleaned tweet has them. We can see how the grammatical structure differs in the lemmatized tweets mainly due to the stop words removal.



```
[TWEET 25]

Clean >>> expo   france  chromosaturation  the notion of color which has  the power to influence our sensations  as intensely as
cold  heat and sound     a terrific experience

After Lemmatization >>> ['expo', 'france', 'chromosaturation', 'notion', 'color', 'power', 'influence', 'sensation', 'intensely',
'cold', 'heat', 'sound', 'terrific', 'experience']
```

This affect's the models' confidence levels of its predictions because the training data used to build the model came from Wikipedia, Tatoeba and SETimes (Fasttext, n.d.). This was one of two nodels provided by fasttext for language detection.

The data from the aforementioned sources are generally upheld to a high standard of writing and is required to follow the grammatical rules of the English language. This standard of writing upheld in the training data is unlike what the language used on social media is.

This explains why the lemmatized tweets scored much lower in general.

Due to this general dip in confidence of prediction, filtering the tweets using higher values like 70% meant that many valid English tweets were being removed in process. Thus, the tweets were filtered using lower values like 20% or 25%. What was interesting to note is that, when the input was the lemmatized tweets, the ones being filtered out were similar to the tweets being filtered when the input came from cleaned tweets with predictions set at 70%.

The general issue with this approach was that even at 25%, due to the lack of context within each tweet, perfectly English sentences were labelled with low probability, so the better option of achieving the same affect was using the language detection with cleaned tweets as input.

In [26]:
```
FASTTEXT_LANG_MODEL = fasttext.load_model('lid.176.ftz')
def english_filter(text):
    ''' This function receives a piece of text as input. If it is engli
        probability of > 45%, it is returned. Else, an empty string is
        @param text: The list that the language check will be performed
    '''
    lang_pred = FASTTEXT_LANG_MODEL.predict(text=text, k=1)
    if lang_pred[0][0] == '__label__en' or lang_pred[1][0] > 0.45: retu
    else: return ""
```

9. Preprocessing Pipeline

In [27]:
```
def preprocess_text(text, analysis_method="lemmatization"): # (M.D. Pie
    ''' Function that performs preprocessing steps on text and retruns
        @param text: The piece of text to preprocess.
    '''
    if analysis_method not in ["lemmatization", "stemming"]: # check if
        raise Exception("preprocess_text: invalid analysis method")
    text = clean(text) # cleaning text
    text = expand_contractions(text) # expand contractions
    text = english_filter(text) # filter to keep only mostly english tw
    token_list = tokenize(text) # tokenization
    token_list = remove_stopwords(token_list) # stop word removal
    if analysis_method == "lemmatization": token_list = lemmatize(token
    else: token_list = stem(token_list) # stemming performed
    return " ".join(token_list) # return processed text as string
```

In [28]:
```
class PreprocessText(BaseEstimator, TransformerMixin): # (H. Chandra, 2
    ''' Custom Transformer to use for text preprocessing which can be u
        with scikit-learn Pipeline.
    '''

    def __init__(self, text_feature_name, mode='lemmatization'):
        ''' Constructor '''
        if not mode in ["lemmatization", "stemming"]: raise Exception("
        self.text_feature_name = text_feature_name
        self.mode = mode

    def fit(self, X, y=None):
        ''' Fit function does nothing here. '''
        return self

    def transform(self, X, y=None):
        ''' Returns a dataframe with preprocessed text as its only colu
            to be used as input to subsequent estimators/transformers i
            a Pipeline.
        '''
        X_ = pd.DataFrame({
            self.text_feature_name: X[self.text_feature_name].apply(
                lambda text : preprocess_text(text=text, analysis_metho
            )
        })
        return X_

    def set_mode(self, mode):
        ''' Allows mode change after input sanity check. '''
        if not mode in ["lemmatization", "stemming"]: raise Exception("
        self.mode = mode
```

In [29]:
```
TEXT_PREPROCESSING_TRANSFORMER = PreprocessText(text_feature_name="Twee
```

In [30]:
```
TEXT_PREPROCESSING_TRANSFORMER.set_mode("lemmatization")
TWEETS_DF["Tweet Body Processed (Lemmatization)"] = TEXT_PREPROCESSING_
TEXT_PREPROCESSING_TRANSFORMER.set_mode("stemming")
TWEETS_DF["Tweet Body Processed (Stemming)"] = TEXT_PREPROCESSING_TRANS
display(TWEETS_DF.head(3))
```

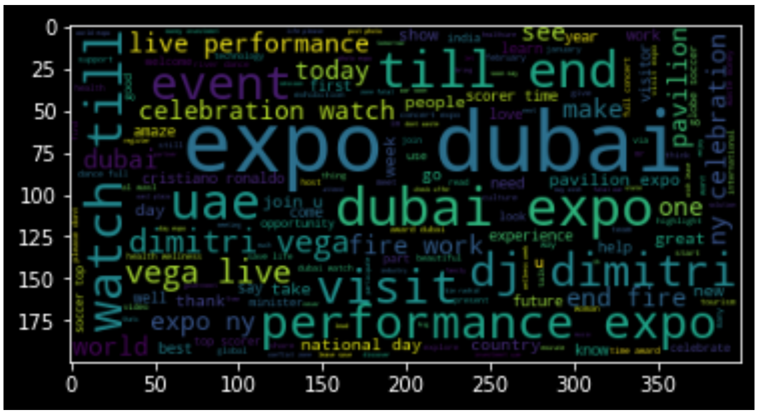
	Tweet Body	TextBlob Sentiment Score	TextBlob Sentiment Label	Vader Sentiment Score	Vader Sentiment Label	Emoji Sentiment Score	Emoji Sentiment Label	Avera Sentime Sco
0	Its #Expo2020 Day	0.00000	0	0.0000	0	-2.0	-2	0.00
1	We celebrated the National Day of the Slovak R...	0.61875	1	0.9531	1	-2.0	-2	0.95
2	Vertebral Deformity Measurements on MRI, CT, a...	0.00000	0	0.0000	0	-2.0	-2	0.00

In [31]:
```
# # UNCOMMENT TO CREATE A FILE WITH ALL PREPROCESSING STAGE OUTPUT COMP
# # save original v/s cleaned v/s tokenized v/s lemmatized v/s stemmed
# # to demonstrate and compare text cleaning, tokenization, stop word r
# tweets_original = TWEETS_DF["Tweet Body"].to_list()
# tweets_clean = [clean(tweet) for tweet in tweets_original]
# tweets_no_contractions = [expand_contractions(tweet) for tweet in twe
# tweets_tokenized = [tokenize(tweet) for tweet in tweets_no_contractio
# tweets_no_stopwords = [remove_stopwords(tweet) for tweet in tweets_to
# tweets_after_stemming = [stem(tweet) for tweet in tweets_no_stopwords
# tweets_after_lemmatization = [lemmatize(tweet) for tweet in tweets_no
# with open("./preprocessing_tweets.txt", "w", encoding="utf-8") as f:
#     "[TWEET {}]\n".format(i)
#     + "Original >>> {}\n".format(tweets_original[i])
#     + "No Contractions >>> {}\n".format(tweets_no_contractions[i])
#     + "Clean >>> {}\n".format(tweets_clean[i])
#     + "Tokenized >>> {}\n".format(tweets_tokenized[i])
#     + "No Stopwords >>> {}\n".format(tweets_no_stopwords[i])
#     + "After Stemming >>> {}\n".format(tweets_after_stemming[i])
#     + "After Lemmatization >>> {}\n".format(tweets_after_lemmatizatio
#     + "\n\n\n"
#     for i in range(len(tweets_original))
# ]))
```

OBSERVATION: A quick way to check if the preprocessing has happened correctly and also to quickly get an overview of the content of the tweets is to generate a word cloud as shown below.

In [32]:
```
plt.imshow(WORD_CLOUD.generate(" ".join(TWEETS_DF["Tweet Body Processed
```

Out[32]:
```
<matplotlib.image.AxesImage at 0x16598a9fb20>
```

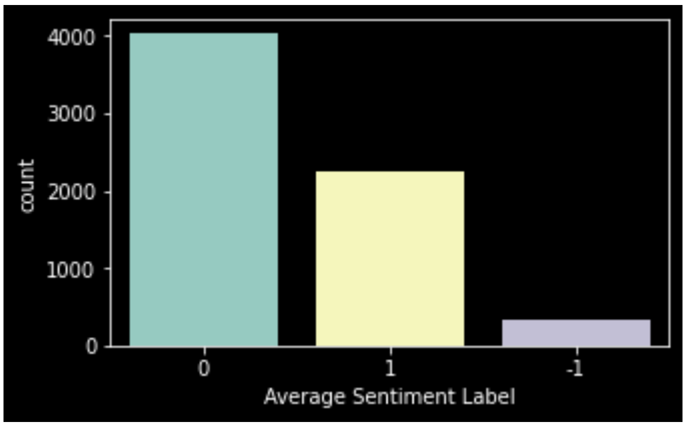


OBSERVATION: Expo 2020 held in Dubai, UAE is all about showcasing/promoting /celebrating innovation in art, science and business. Global companies and organizations gather here and set up pavilions displaying their exceptional work or culture/heritage /advancements/ideas from their respective countries. Acclaimed artists from all over the world visit the expo and showcase their talent. Over 65 million people from around the globe have visited the expo thus far. It can be seen from the word cloud that it does indeed capture the essence of Expo 2020. This shows that the preprocessing has succeeded in preserving meaningful words while removing unnecessary terms.

10. Data Splitting

In [33]:
```python
plt.figure(figsize=(5,3))
np.bincount(TWEETS_DF["Average Sentiment Label"].replace(-1, 2))
sns.countplot(
    x=TWEETS_DF["Average Sentiment Label"],
    order=TWEETS_DF["Average Sentiment Label"].value_counts().index
)
```

Out[33]:
```
<AxesSubplot:xlabel='Average Sentiment Label', ylabel='count'>
```



OBSERVATION: The frequency distribution of data associated with each label is highly uneven. Tweets with negative sentiment are very few compared to others. Thus, the oversampling technique, Synthetic Minority Oversampling Technique (SMOTE) shall be used to address the uneven distribution issue.

SMOTE (Synthetic Minority Oversampling Technique) is a popular approach used for tackling the imbalanced dataset problem. The algorithm generates “synthetic” samples by over sampling the minority class. (Kim, 2018)

While creating the “synthetic” samples, a data is chosen for copying. SMOTE looks k-nearest neighbors of the chosen data. Between the chosen sample and its k-nearest neighbors, SMOTE creates random values in the feature space. (Kim, 2018)

SMOTE is more ideal to use than the regular oversampling method that is performed using `resample()` method (offered by `sklearn.utils`) as it generates using the nearest neighbors of the chosen data and doesn't just make copies of the minority class.

In [34]:
```python
SMOTE_MODEL = SMOTE(random_state=30)
```

In [35]:
```python
DF = pd.DataFrame({ # get only relevent columns
    "text": TWEETS_DF["Tweet Body Processed (Lemmatization)"],
    "sentiment": TWEETS_DF["Average Sentiment Label"]
})
DF = DF.drop_duplicates(subset=['text']) # remove duplicates
DF = DF[DF["text"]!=""] # remove empty string rows
print("n_samples before filtering empty strings = {} and after filterin
print("check if any null still exists = {}".format(DF["text"].isnull().
DF = DF.reset_index(drop=True)
display(DF.head(3))
```

```
n_samples before filtering empty strings = 6613 and after filtering = 55
check if any null still exists = False
```

	text	sentiment
0	day	0
1	celebrate national day slovak republic expo du...	1
2	vertebral deformity measurement mri ct radiogr...	0

In [36]:
```python
X_train_text, X_test_text, y_train, y_test = train_test_split(
    DF["text"],
    DF["sentiment"],
    test_size=0.3,
    random_state=23,
    shuffle=True,
    stratify=DF["sentiment"]
)
```

11. Feature Extraction

Feature extraction within context of Natural Language Processing (NLP) refers to converting variable length text into fixed length numbers so that it can be provided as input to ML models that are trying to perform classification or regression tasks. The following are 2 broad popular approaches to Feature Extraction for NLP. (Eiki, 2019)

11.1. Using Bag Of Words (BOW) Model

(Great Learning Team, 2020), (J. Brownlee, 09 Oct 2017), (Unfold Data Science, Apr 24 2020)

The aim of a BOG representation is to convert documents in a corpus to numeric vectors. The BOG model assumes that every word is independent of others and does not capture order/grammar of the words in text.

Generating a BOW model involves determining unique words (vocabulary) in a given corpus against which words in documents in the corpus may be compared and assigned numerical scores. Thus, documents in a corpus can be converted to vectors of numerical scores corresponding to each term in vocabulary.

Often a BOW model is represented using a table containing a column for each word in the identified vocabulary and a row for each document in the corpus as shown below.

BOW	**word 1**	**word 2**	**...**	**word n**
doc 1	score	score	...	score
doc 2	score	score	...	score
...	...	...	...	...
doc n	score	score	...	score

The complexity of a BOW model depends on the scoring function it uses. A very simple scoring function would be where for every word in vocabulary, if it is present in a document, then a score of 1 is assigned while 0 is assigned otherwise. Such an approach is called binary scoring because the resulting vectors would contain binary values (0/1) only.

As vocabulary increases with increasingly large corpora, it gets more and more difficult and computationally expensive (especially in terms of storage space) to model. This is where text preprocessing techniques like stop word removal, punctuation removal, stemming and lemmatization help in reducing the size of identified vocabulary by eliminating unwanted/largely useless words.

But, even after text is cleaned, a binary representation would probably not be the best choice when it comes to complex classification tasks or tasks involving large corpora since then, the sparse matrices generated as a result of using binary scoring, would still require an unnecessarily large amount of memory to store a small amount of knowledge about the given text.

Thus, there is a need to represent vocabulary more efficiently for which the n-gram approach may be explored. Here, words in the vocabulary are grouped into groups of "n" sequentially occurring words in the corpus (a "word"/"token" here is a.k.a. a "gram") which can lead to a reduction in the no. of columns that the matrix/table representing the corpus would have. The term n-gram is a general one. When "n" equals 1, the representation is called a unigram model, when "n" equals 2, it is called a bigram model, when "n" equals 3, it is called a trigram model and so on. N-gram models also have the added advantage that they capture some though little order of words as well. So, using a bigram BOG model would help capture a more meaning than using a simpler unigram BOG. BOG models would still however not be able to capture context of words due to the unrealistic assumptions that it makes.

Once vocabulary has been suitably represented, the bag of words model still requires a scoring function using which numeric values can be assigned to documents based on their words corresponding to vocabulary. Apart from the binary approach mentioned previously, there are other simple ways of scoring documents. One method is to simply count the no. of times a particular term (column) in the vocabulary has occurred in a document and then setting this count as the value. Another popular simple approach is to consider the frequency of occurrence of words in a document instead of the count.

Simple scoring functions can work for many situations but for large corpora and models trying to solve more complex problems, slightly more complex scoring functions may be required to extract more useful features. If a primary concern is the large size of vocabulary due to the corpus being very large, and thereby difficult to efficiently represent as a model of vectors, then a possible solution would to use hashing to score documents. A hash function would map words in the document to a fixed length number. This solves the issue of variable length documents and eliminates the need to have a column for every word/group of words in the vocabulary. However, hashing comes with the challenges of finding suitable hash functions that results in the most useful representation while ensuring minimum collisions (a collision in this context refers to 2 different documents being assigned the same hash value).

Another popular way of scoring documents is called TF-IDF. This method assigns high values to frequently occurring words (Term Frequency (TF)) just like in the simple scoring approach based on word frequency. But frequency alone may not be the best tool to judge the importance of a word. For example, a word like "the" can be very common in text but does not convey much knowledge about the meaning of the text. To overcome this issue, TF-IDF, also considers Inverse Document Frequency which is the rarity of occurrence of a word amongst all documents. TF-IDF assigns positive scores for words with higher frequency in a document but suppresses the score if those same words are very common across all documents. Thus, TF-IDF succeeds in highlighting distinct and likely most useful/important words in each document.

OBSERVATION: If we are to use a BOW based text representation to extract features for sentiment analysis from tweets here, then the best option would be to represent vocabulary using n-gram representation and then to use TF-IDF to assign numeric values to each document (tweet) in the corpus (dataset of all tweets). For sentiment analysis, capturing the meaning of text is very important. BOW models are notorious for not being able to capture the meaning/context of words. This is why n-gram notation and TF-IDF scoring is to be used here to maximize capturing of the little if any word structure that can be captured using BOW models.

OBSERVATION: N = 2 is a good choice for n-gram vectorization of text w.r.t. sentiment analysis since bigrams are not as computationally expensive to compute with lots of text data as n-grams with n > 2 while at the same time, they are able to capture negation of negative terms which unigrams are unable to capture. For example, the term "bad" carries a negative sentiment but the term pair "not bad" carries a positive sentiment but if unigram vectorization was used instead of bigram vectorization, then the term pair "not bad" would get split into "not" and "bad" thereby incorrectly indicating negative sentiment. (K. Thiel, 2016)

In [37]:
```python
print("the smallest no. of words seen in a document = {}".format(
    min([len(doc.split()) for doc in X_train_text])
))
```

```
the smallest no. of words seen in a document = 1
```

OBSERVATION: In the given preprocessed corpus, there exists documents with only 1 word in them, these words are, however not very likely to be distinct words and thus, they shall be ignored during vectorization of documents. An alternate approach would be to consider both unigrams and bigrams but this would lead to an increase in no. of features significantly the negative impact of which is likely to outweigh any positive effects that may come with including unigrams. Thus, the decision here is to perform a strictly bigram based vectorization of documents.

BOW with Bigram vectorization using TF-IDF Scores

Sujanmul(P. Sujanmulk, 2021)

In [38]:
```
# train and test data before vectorization
print("Train and test data before vectorization.")
print("X_train_text (shape = {})".format(X_train_text.shape))
print("y_train (shape = {})".format(y_train.shape))
print("X_test_text (shape = {})".format(X_test_text.shape))
print("y_test (shape = {})".format(y_test.shape))
```

```
Train and test data before vectorization.
X_train_text (shape = (3892,))
y_train (shape = (3892,))
X_test_text (shape = (1668,))
y_test (shape = (1668,))
```

In [39]:
```
tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,2))
X_train_vec = tfidf_vectorizer.fit_transform(X_train_text)
print(len(X_train_text))
print(f"Text Vectors: {repr(X_train_vec)}")
print(f"\nno. of unique words = {len(tfidf_vectorizer.vocabulary_)}")
print(f"\nvocabulary = {list(tfidf_vectorizer.vocabulary_.items())[:10]
```

```
3892
Text Vectors: <3892x29822 sparse matrix of type '<class 'numpy.float64'>'
        with 39149 stored elements in Compressed Sparse Row format>

no. of unique words = 29822

vocabulary = [('second day', 22824), ('day make', 5604), ('make official',
15824), ('official visit', 18303), ('visit able', 27958), ('able learn', 3
1), ('learn country', 14852), ('country structure', 5026), ('structure use
', 24873), ('use attract', 27544)] ...
```

In [40]:
```
# storing TF-IDF matrix as a dataframe
TFIDF_TRAIN_DF = pd.DataFrame(
    data = X_train_vec.toarray(),
    columns = tfidf_vectorizer.get_feature_names_out(),
)
display(TFIDF_TRAIN_DF.head(3))
print("TFIDF matrix shape = {}".format(TFIDF_TRAIN_DF.shape))
```

	aaai le	aafaq islamic	aap ibef	abcc participate	abdallah thanks	abduct abdul	abduct intelligence	abduction force	abdul hafeez	abdul hamid	.
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	.
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	.
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	.

3 rows × 29822 columns

```
TFIDF matrix shape = (3892, 29822)
```

OBSERVATION: Though bigram vectorization was performed, it can be seen that the TF-IDF matrix is still sparse. One of the BOW method's less desireable features can thus be observed here

In [41]:
```python
# features with highest and lowest TF-IDF scores.
feature_names = tfidf_vectorizer.get_feature_names_out()

max_value = X_train_vec.max(axis=0).toarray().ravel()
sorted_by_tfidf_high = max_value.argsort()
highest_tfidf_features = feature_names[sorted_by_tfidf_high]
print("features with highest TF-IDF scores = {} ... (length = {})".form
    highest_tfidf_features[0:10],
    len(highest_tfidf_features)
))

min_value = X_train_vec.min(axis=0).toarray().ravel()
sorted_by_tfidf_low = min_value.argsort()
lowest_tfidf_features = feature_names[sorted_by_tfidf_low]
print("features with lowest TF-IDF scores = {} ... (length = {})".forma
    lowest_tfidf_features[0:10],
    len(lowest_tfidf_features)
))
```

```
features with highest TF-IDF scores = ['please father' 'treatment please
'condition bad' 'money treatment'
 'kidney patient' 'bad sir' 'father kidney' 'patient dont'
 'please condition' 'since february'] ... (length = 29822)
features with lowest TF-IDF scores = ['aaai le' 'pistachio pcsavailable' '
piss week' 'piss prove' 'piss expo'
 'piss cost' 'piss brother' 'pisa report' 'pirate coast' 'pippa malmgren']
... (length = 29822)
```

In [42]:
```python
doc_i = 100
min_max_score = {"min": ["", math.inf], "max": ["", -math.inf]}
for key, val in dict(TFIDF_TRAIN_DF.iloc[doc_i, :]).items():
    if val > min_max_score["max"][1]:
        min_max_score["max"][0] = key
        min_max_score["max"][1] = val
    if val < min_max_score["min"][1] and val > 0:
        min_max_score["min"][0] = key
        min_max_score["min"][1] = val
print("word with highest TF-IDF score in document {} = {} (score = {})"
    doc_i, min_max_score["max"][0], round(min_max_score["max"][1], 2)
))
print("word with lowest TF-IDF score in document {} = {} (score = {})".
    doc_i, min_max_score["min"][0], round(min_max_score["min"][1], 2)
))
print('\ndocument {} = "{}"'.format(doc_i, X_train_text.iloc[doc_i]))
```

```
word with highest TF-IDF score in document 100 = ceo mtcc (score = 0.25)
word with lowest TF-IDF score in document 100 = expo dubai (score = 0.08)

document 100 = "jan pm uae pm myt join siti syaliza mustapha ceo mtcc expo
dubai seminar forest management chain custody certification entail"
```

OBSERVATION: In the above printed example, it can be seen that the bigram "expo dubai" has been rightly assigned a low score since all the documents are related to the expo held in dubai. So, those words are expected to appear in many documents in the corpus. The words "ceo mtcc" however, likely rarely appears in other documents (high IDF) in which it appears more than once (high TF) and hence is rightly given a higher TF-IDF score. Thus, the effectiveness of calculating TF-IDF scores to help identify distinct and important terms is thereby evident.

OBSERVATION: To reduce no. of features so as to use only the most distinguishable yet important features of each document, 80% of the no. of total features worth of bigrams that were found to have lowest TF-IDF scores shall be removed.

In [43]:
```
features_to_drop = lowest_tfidf_features[:int(len(lowest_tfidf_features
print("features to drop = {}".format(features_to_drop))
```

```
features to drop = ['aaai le' 'pistachio pcsavailable' 'piss week' ... '
lp identify'
 'help help' 'help give']
```

In [44]:
```
print("before feature reduction =", TFIDF_TRAIN_DF.shape)
X_train = TFIDF_TRAIN_DF.drop(features_to_drop, axis=1)
print("after feature reduction =", X_train.shape)
```

```
before feature reduction = (3892, 29822)
after feature reduction = (3892, 5965)
```

In [45]:
```
X_test_vec = tfidf_vectorizer.transform(X_test_text)
TFIDF_TEST_DF = pd.DataFrame (
    data = X_test_vec.toarray(),
    columns = tfidf_vectorizer.get_feature_names_out(),
)
print((TFIDF_TEST_DF.columns == TFIDF_TRAIN_DF.columns).all())

X_test = TFIDF_TEST_DF.drop(features_to_drop, axis=1)
print(X_test.shape)
```

```
True
(1668, 5965)
```

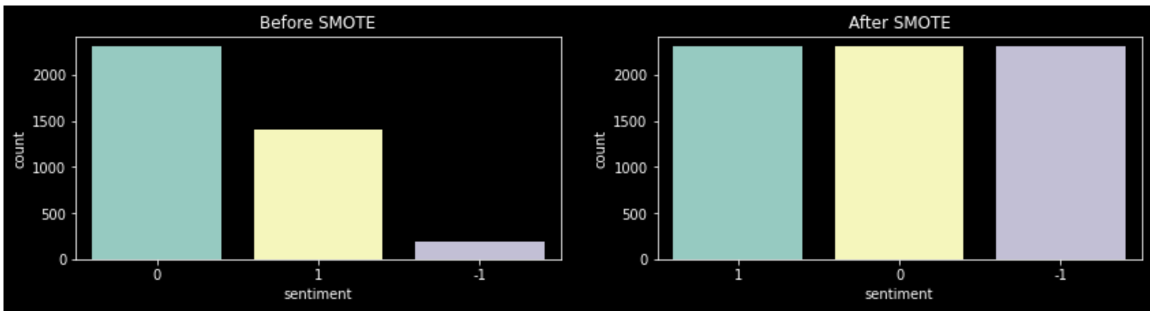
OBSERVATION: In the above example, the elimination of 80% of lowest scoring features that were undesirable was done based on TF-IDF scores after they were calculated for all features. If the corpus was much larger, then, this may not have been the best approach as then, the no. of features would be much bigger and calculating all their values only to discard a portion of them later is not efficient. Instead, the `TfidfVectorizer` has parameters like `min_df` and `max_features` which when set, limits the no. of features output. The `min_df` when set to a number say "n" ensures that only features that occur in at least "n" documents are part of the output. The `max_features` parameter allows one to set the no. of output features desired. Once this number is set to say 'm', the 'm' no. of features with highest TF-IDF scores are returned. Such parameters amongst others, make it convenient to leverage TF-IDF scores for feature reduction.

In [46]:
```
def bow_embed(docs, vectorizer, drop_pc=None, features_to_drop=None):
    ''' Function that converts given documents
        into vectors using a bag of words model.
        @param docs: Documents to embed as an iterable.
        @param vectorizer: Vectorizer already fitted on desired data.
        @param drop_pc: The % of features to drop based on TF-IDF score
        @return docs_df: Embeddings as a dataframe.
        @return features_to_drop: Feature that were dropped
    '''
    # get TD-IDF scores
    docs_vec = vectorizer.transform(docs)
    docs_df = pd.DataFrame(data=docs_vec.toarray(), columns=vectorizer.

    # drop lowest scoring features
    if type(features_to_drop) == type(None):
        feature_names = vectorizer.get_feature_names_out()
        min_value = X_train_vec.min(axis=0).toarray().ravel()
        lowest_features = feature_names[min_value.argsort()]
        features_to_drop = lowest_features[:int(len(lowest_features)*dr

    return docs_df.drop(features_to_drop, axis=1), features_to_drop
```

In [47]:
```
tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,2))
tfidf_vectorizer.fit(X_train_text)

X_train_bow, features_to_drop = bow_embed(docs=X_train_text, vectorizer
X_test_bow, _ = bow_embed(docs=X_test_text, vectorizer=tfidf_vectorizer
y_train_bow = y_train
y_test_bow = y_test
```

In [48]:
```
# resampling
plt.figure(figsize=(14,3))
plt.subplot(121)
plt.title("Before SMOTE")
sns.countplot(x=y_train_bow, order=y_train_bow.value_counts().index)
plt.subplot(122)
X_train_bow, y_train_bow = SMOTE_MODEL.fit_resample(X_train_bow, y_trai
plt.title("After SMOTE")
sns.countplot(x=y_train_bow, order=y_train_bow.value_counts().index)
```

Out[48]:
```
<AxesSubplot:title={'center':'After SMOTE'}, xlabel='sentiment', ylabel=
ount'>
```



In [49]:
```
# convert to numpy array
X_train_bow = X_train_bow.to_numpy()
y_train_bow = y_train_bow.to_numpy()
X_test_bow = X_test_bow.to_numpy()
y_test_bow = y_test_bow.to_numpy()
```

In [50]:
```python
# train and test data before vectorization
print("Train and test data before vectorization.")
print("X_train_text (shape = {})".format(X_train_text.shape))
print("y_train (shape = {})".format(y_train.shape))
print("X_test_text (shape = {})".format(X_test_text.shape))
print("y_test (shape = {})".format(y_test.shape))

# train and test data after smote and vectorization
print("\nTrain and test data after vectorization.")
print("X_train_bow (shape = {})".format(X_train_bow.shape))
print("y_train_bow (shape = {})".format(y_train_bow.shape))
print("X_test_bow (shape = {})".format(X_test_bow.shape))
print("y_test_bow (shape = {})".format(y_test_bow.shape))
```

```
Train and test data before vectorization.
X_train_text (shape = (3892,))
y_train (shape = (3892,))
X_test_text (shape = (1668,))
y_test (shape = (1668,))

Train and test data after vectorization.
X_train_bow (shape = (6906, 5965))
y_train_bow (shape = (6906,))
X_test_bow (shape = (1668, 5965))
y_test_bow (shape = (1668,))
```

OBSERVATION: The train and test sets (`X_train_bow` , `y_train_bow` , `X_test_bow` , `y_test_bow`) where document vectors were derived through a BOW approach is now available to be used with classifiers.

11.2. Word Embedding Method

(J. Brownlee, 11 Oct 2017), (Unfold Data Science, 05 May 2020)

Just like the bag of words model, word embedding also converts a documents into vectors of numeric values. But unlike bag of words method which does not consider the interdependencies between words, word embedding tries to capture relationships between words to better capture the context/meaning of words in a given piece of text. This is the main difference between the bag of word approach and the word embedding approach of text vectorization.

In the bag of word approach, the features (columns) are simply vocabulary or variations thereof and the scoring function is a fixed predefined method but word embedding follows an entirely different strategy. It learns the vectors associated with each word using machine learning models like neural networks. Due to the use of ML tools to learn word vectors, word embedding is able to learn complex relationships between words which the bag of words approach is unable to do curtsy of the relatively simpler and less flexible scoring functions that it uses to arrive at the vectors associated with each document.

The 2 broad types of word embeddings are frequency based and prediction based word embedding. Two of the most popular existing implementations of word embedding are Word2Vec and GloVe.

2.1. Word2Vec (codebasics, 2021)

Word2Vec performs prediction based word embedding. At its core, Word2Vec algorithm makes use of a neural network with one hidden layer that has fewer units than the input and output layer. Updating the parameters of this neural network (NN) is what leads to it learning vectors associated with each word as per its context learned from input data which is then used to make predictions that solve problems like a sentence completion task.

To train a Word2Vec model to capture correct word vectors, one usually starts with a dummy problem that the network is to try and solve. The training samples for this problem is generated using documents in the corpus. For example, if the dummy task is sentence completion by predicting a word given 2 preceding words, then, for this task, the training samples can be generated by representing vocabulary from a corpus as trigrams where the input data would be the first 2 words in each trigram and the ground truth that the neural network is to try and output would be the 3rd word of the trigram.

Once training data has been obtained from a corpus, the model is trained using this data to solve the particular selected problem. It is by trying to solve tasks like the example task mentioned before, that the neural network captures word embeddings. A word embedding here, refers to the vectors associated with a word that captures its meaning based on context in which it was seen in the input data.

Word2Vec models may be trained for predicting a target word given some context as input in which case it is said to use a Continuous Bag Of Words (CBOW) approach or they may be trained to predict context words given a word as input in which case the approach is called a Step Gram approach.

2.2. GloVe (Normalized Nerd, 2020) (Sciforce, 2018)

GloVe is short for Global Vectors. It performs frequency/count based word embedding using a co-occurrence matrix. A co-occurrence matrix like in the example one given below, stores no. of times a word (word in a column) occurs after another word (word in a row). The rows and columns are all the words of vocabulary identified from the corpus such that matrix value [i,j] is the no. of times the word at column j comes after the word at row i in the corpus.

co-occurrence	**word 1**	**word 2**	**...**	**word j**
word 1	count	count	...	count
word 2	count	count	...	count
...	...	...	...	...
word i	count	count	...	count

Unlike Word2Vec which uses a feed forward neural network and learns the embedding of a word based on the context of that word gathered from short inputs thereby following the prediction based approach of word embedding, GloVe, follows count/frequency based embedding which involves calculating the probability of a word occurring within context of others from values in a co-occurrence matrix of vocabulary from the entire corpus (not one sentence/sequence of words at a time) where the dimensionality of the co-occurrence matrix is reduced to make computation feasible. GloVe models make use of log-bilinear functions.

OBSERVATION: Since word embedding methods capture meaning and context of words in a given piece of text, using a word embedding approach as opposed to a using a BOG model would lead to better results with classifiers w.r.t. sentiment analysis of tweets. So, here a pre-trained Glove model trained on 2 billion tweets shall be used to generate the vectors associated with each document.

Glove Text Embedding

In [51]:
```
print("Train and test data before vectorization.")
print("X_train_text (shape = {})".format(X_train_text.shape))
print("y_train (shape = {})".format(y_train.shape))
print("X_test_text (shape = {})".format(X_test_text.shape))
print("y_test (shape = {})".format(y_test.shape))
```

```
Train and test data before vectorization.
X_train_text (shape = (3892,))
y_train (shape = (3892,))
X_test_text (shape = (1668,))
y_test (shape = (1668,))
```

In [52]:
```
print("In this glove model, words like the word '{}' is represented as
    "expo", len(GLOVE_TWITTER_MODEL_200["expo"])
))
```

```
In this glove model, words like the word 'expo' is represented as a vect
of size 200.
```

OBSERVATION: It can be observed here that unlike in the BOW approach (35232 features), the no. of features in the vector resulting from a word embedding approach is much fewer and compact (200 features). Since vectors resulting from word embedding are learnt using an ML model, though having fewer features, those features are likely more meaningful and capture the meaning of the each word better.

In [53]:
```
print("words most similar to the word {} = {}".format("expo", GLOVE_TWI
```

```
words most similar to the word expo = [('festival', 0.669904351234436),
workshop', 0.5815601348876953), ('event', 0.5714000463485718), ('showcase
', 0.5705898404121399), ('center', 0.5439842343330383), ('exhibition', 0.5
394160747528076), ('conference', 0.5378116965293884), ('fest', 0.531081140
0413513), ('forum', 0.531076192855835), ('convention', 0.529899835586547
9)]
```

OBSERVATION: Here, it can be observed that words returned as similar to the word "expo" by the glove word embedding model are indeed associated with happenings associated with an expo. This further highlights the advantage of using a word embedding model for word vectorization for sentiment analysis. The vectors so obtained being more meaningful yet compact will aid in avoiding the "curse of dimensionality" w.r.t. training ML models using these vectors as input.

OBSERVATION: It can be seen that the GloVe model produces 1 vector per word in the

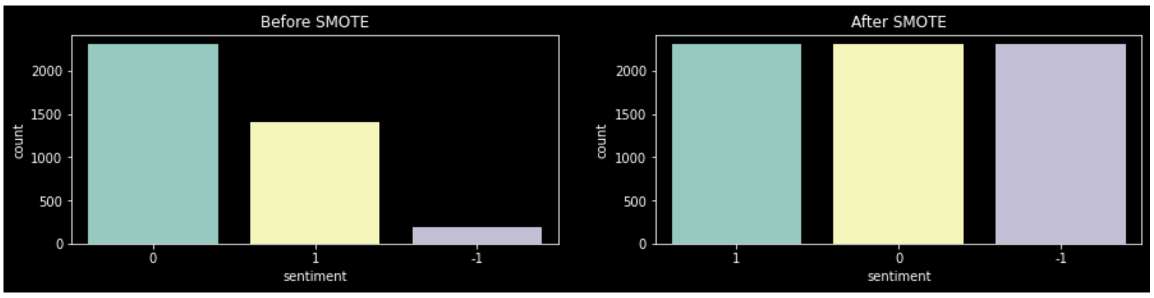
document. So, the following cells shall calculate the sum of vectors of all words in a
document to obtain a vector that represents each document

In [54]:
```python
def glove_embed(docs):
    ''' Function that converts given documents
        into vectors using a GloVe model.
        @param docs: Documents to embed as an iterable.
        @return docs: Embedded documents as np array.
    '''
    docs_embedded = []
    for doc_words in [tokenize(doc) for doc in docs]:
        doc_vector = np.array([0.0]*200)
        for word in doc_words:
            try: word_vec = GLOVE_TWITTER_MODEL_200[word]
            except: word_vec = np.array([0.0]*200)
            doc_vector += word_vec
        docs_embedded.append(doc_vector)
    return np.array(docs_embedded)
```

In [55]:
```python
X_train_we = glove_embed(X_train_text)
X_test_we = glove_embed(X_test_text)
y_train_we = y_train
y_test_we = y_test
```

In [56]:
```python
# resampling
plt.figure(figsize=(15,3))
plt.subplot(121)
plt.title("Before SMOTE")
sns.countplot(x=y_train_we, order=y_train_we.value_counts().index)
plt.subplot(122)
X_train_we, y_train_we = SMOTE_MODEL.fit_resample(X_train_we, y_train_w
plt.title("After SMOTE")
sns.countplot(x=y_train_we, order=y_train_we.value_counts().index)
```

Out[56]:
```
<AxesSubplot:title={'center':'After SMOTE'}, xlabel='sentiment', ylabel=
ount'>
```



In [57]:
```python
# convert to numpy array
X_train_we = np.array(X_train_we)
y_train_we = np.array(y_train_we)
X_test_we = np.array(X_test_we)
y_test_we = np.array(y_test_we)
```

In [58]:
```python
# train and test data before vectorization
print("Train and test data before vectorization.")
print("X_train_text (shape = {})".format(X_train_text.shape))
print("y_train (shape = {})".format(y_train.shape))
print("X_test_text (shape = {})".format(X_test_text.shape))
print("y_test (shape = {})".format(y_test.shape))

# train and test data after smote and vectorization
print("\nTrain and test data after vectorization.")
print("X_train_we (shape = {})".format(X_train_we.shape))
print("y_train_we (shape = {})".format(y_train_we.shape))
print("X_test_we (shape = {})".format(X_test_we.shape))
print("y_test_we (shape = {})".format(y_test_we.shape))
```

```
Train and test data before vectorization.
X_train_text (shape = (3892,))
y_train (shape = (3892,))
X_test_text (shape = (1668,))
y_test (shape = (1668,))

Train and test data after vectorization.
X_train_we (shape = (6906, 200))
y_train_we (shape = (6906,))
X_test_we (shape = (1668, 200))
y_test_we (shape = (1668,))
```

OBSERVATION: The train and test sets (`X_train_we` , `y_train_we` , `X_test_we` , `y_test_we`) derived through the GloVe word embedding approach is now available to be used with classifiers.

12. Classifier Evaluation

A confusion matrix (CM) is a great way to see the performance of a classifier at a glance. The aim is for the diagonals (True positives/negatives of each class) to be high while other values remain low.

In addition to the CM, an ROC curve that plots the true positive rate against the false positive rate shall be calculated for predictions of each class. Area under the ROC curve can be calculated and the trade-off between sensitivity and specificity/between true and false positives for different classes can be shown using a gain chart.

In [59]:
```python
def calculate_performance_measures(cm_df):
    ''' Calculates and returns
        * TP, TN, FP, FN
        * Precision, Recall
        * F-Measure
        for every class in a confusion matrix given as a dataFrame.
        @param cm_df = Confusion matrix as a dataFrame.
        @return metrics: Dictionary with all calculated metrics.
    '''
    metrics = {}
    attributes = list(cm_df.columns)

    for attribute in attributes:
        tp = cm_df.loc[[attribute], [attribute]].values[0][0]
        fp = cm_df.loc[:, [attribute]].drop([attribute],axis=0).values.
        fn = cm_df.loc[[attribute], :].drop([attribute],axis=1).values.
        tn = sum([cm_df.loc[[label], [label]].values[0][0] for label in
        precision = tp/(tp+fn)
        recall = tp/(tp+fp)
        f = 2*((recall*precision)/(recall+precision))
        tp_rate = tp/(tp+fn)
        fp_rate = fp/(fp+tn)
        metrics[attribute] = {
            "TP": tp,
            "TN": tn,
            "FP": fp,
            "FN": fn,
            "Precision": precision,
            "Recall": recall,
            "F": f,
            "TP Rate": tp_rate,
            "FP Rate": fp_rate
        }

    return metrics
```

In [60]:
```python
def get_performance_measures(classifier_name, df_name, truth, predictio
    ''' Returns all performance measures of current model on given data
        all calculated metrics.
        @param classifier_name: A name for the classifier.
        @param df_name: A name for the dataframe.
        @param truth: Ground truth.
        @param prediction: Predicted values.
        @param labels: The labels corresponding to classes.
                       (for example: [-1, 0, 1] corresponds to ["negati
        @param prediction_prob: Probability of predictions
        @returns performance_measures: Dictionary with calculated
                                        * Accuracy
                                        * Confusion Matrix
                                        * Class Specific Performance
                                        * ROC
                                       values.
    '''

    print(TextStyle.GREEN(TextStyle.BOLD("Performance of {} on {}.".for

    performance_measures = {}
    truth = np.array(truth)

    accuracy = accuracy_score(truth, prediction)
    performance_measures["Accuracy"] = accuracy

    # confusion matrix
    cm = pd.DataFrame(confusion_matrix(truth, prediction), index=labels
    performance_measures["Confusion Matrix"] = cm.values.tolist()

    # other performance measures for each attribute
    performance_measures["Class Specific Performance"] = calculate_perf

    plt.figure(figsize=(15,5))
    plt.subplot(121)
    sns.heatmap(cm, cmap="viridis", annot=True)  # plot confusion matri
    plt.title("Confusion Matrix")

    FPRateROC = dict()
    TPRateROC = dict()
    t = label_binarize(truth,classes=labels)
    AUC = dict()

    for i in range(3):
        FPRateROC[i], TPRateROC[i], _ = roc_curve(t[:,i], prediction_pr
        AUC[i] = auc(FPRateROC[i], TPRateROC[i])

    plt.subplot(122)
    plt.title("ROC AUC Curves")
    plt.plot(FPRateROC[0], TPRateROC[0], color='gray', label='positives
    plt.plot(FPRateROC[1], TPRateROC[1], color='pink',label='negatives
    plt.plot(FPRateROC[2], TPRateROC[2], color='purple', label='neutral
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=12)
    plt.ylabel('True Positive Rate (Recall)', fontsize=12)
    plt.legend(loc="lower right")

    return performance_measures
```

13. Experimenting With Classifiers

13.1. Logistic Regression Classifier

(Brownlee, 2021)

Logistic Regression is a classification model that is used to estimate the probability and it does so by using the sigmoid function (1/1+e^-z) which returns values within the range of 0 and 1. It works really well with linearly separable data. The challenge stems from Logistic Regression doing well as a binary classification problem, whereas, we have a multi-class classification problem. However, scikit-learn (2021b) has set by default the "multi_class" parameter to "auto" so that when a binary problem is fit to the model "ovr" will be select and when a multi-class problem is fit to the model "multinomial" will be automatically selected.

Further modifications to the model will be done by tweaking the hyper parameters of the using Grid Search.

OBSERVATION: While instantiating our Logistic Regression Model, we can use a parameter called "class_weight" which represents the weights associated with the classes. Since our data set contains imbalanced data, the "class_weight" can be set to "balanced" which would attribute a larger weight to the classes with low frequency and smaller weight to the classes with high frequency. (scikit-learn, 2021b)

However, as SMOTE has been introduced, this implementation was no longer necessary as it was one form to deal with the imbalanced in the data.

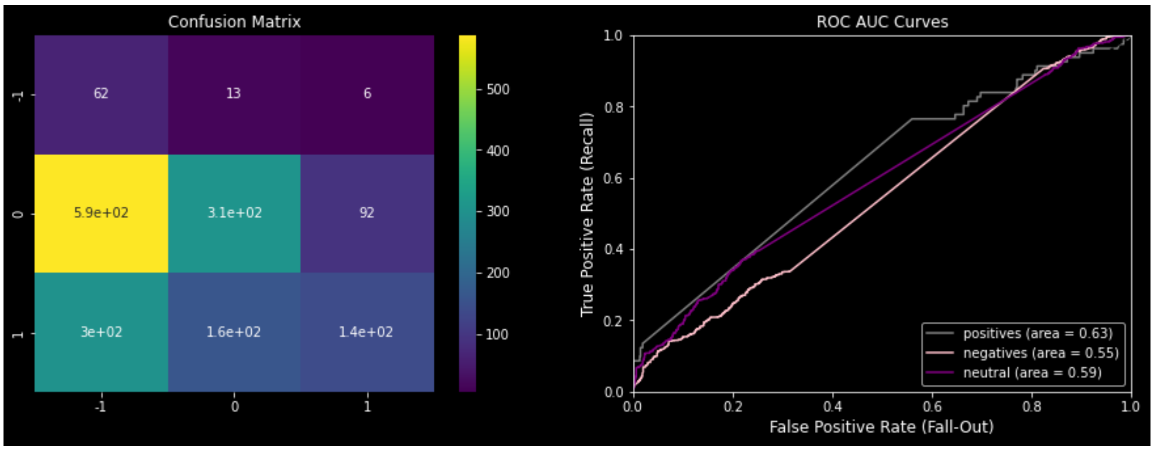
Working with BOW Embeddings

In [61]:
```
logreg = LogisticRegression()
logreg.fit(X_train_bow, y_train_bow)
predictions = logreg.predict(X_test_bow)
prediction_prob = logreg.predict_proba(X_test_bow)
```

```
In [62]: print('Logistic Regression Train accuracy %s' % logreg.score(X_train_bo
         print('Logistic Regression Test accuracy %s' % accuracy_score(predictio
         get_performance_measures("Logistic Regression", "tweets", y_test_bow, p
```

```
Logistic Regression Train accuracy 0.8004633651896901
Logistic Regression Test accuracy 0.3039568345323741
Performance of Logistic Regression on tweets.
```

```
Out[62]: {'Accuracy': 0.3039568345323741,
          'Confusion Matrix': [[62, 13, 6], [587, 307, 92], [302, 161, 138]],
          'Class Specific Performance': {-1: {'TP': 62,
            'TN': 445,
            'FP': 889,
            'FN': 19,
            'Precision': 0.7654320987654321,
            'Recall': 0.06519453207150368,
            'F': 0.12015503875968991,
            'TP Rate': 0.7654320987654321,
            'FP Rate': 0.6664167916041979},
           0: {'TP': 307,
            'TN': 200,
            'FP': 174,
            'FN': 679,
            'Precision': 0.31135902636916835,
            'Recall': 0.6382536382536382,
            'F': 0.4185412406271302,
            'TP Rate': 0.31135902636916835,
            'FP Rate': 0.46524064171123},
           1: {'TP': 138,
            'TN': 369,
            'FP': 98,
            'FN': 463,
            'Precision': 0.22961730449251247,
            'Recall': 0.5847457627118644,
            'F': 0.3297491039426523,
            'TP Rate': 0.22961730449251247,
            'FP Rate': 0.20985010706638116}}}
```



OBSERVATION: The model overfits as the training data's accuracy is much higher than the test data's accuracy. Although, having used SMOTE to oversample and balance the data it overfits. The data is not linearly separable.

Working with GloVe Embeddings

```
In [63]:
logreg.fit(X_train_we,y_train_we)
predictions = logreg.predict(X_test_we)
prediction_prob = logreg.predict_proba(X_test_we)
```

```
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linea
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

OBSERVATION: The model does not converge when using the GloVe Embedding. The maximum number of iteration are set by default to 100 (scikit-learn, 2021b), so we can introduce a parameter to increase the number of iterations.

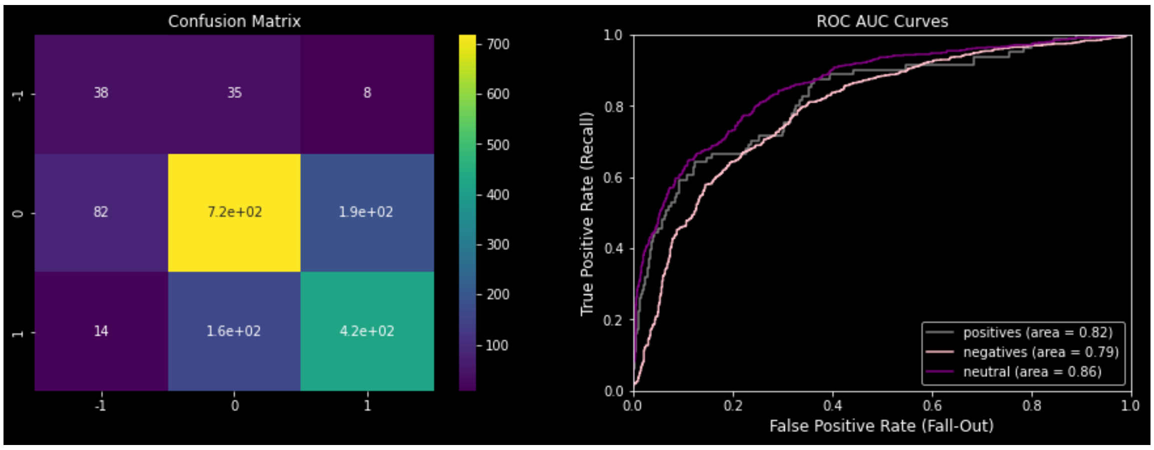
```
In [64]:
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_we,y_train_we)
predictions = logreg.predict(X_test_we)
prediction_prob = logreg.predict_proba(X_test_we)
```

```
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linea
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [65]:
```python
print('Logistic Regression Train accuracy %s' % logreg.score(X_train_we
print('Logistic Regression Test accuracy %s' % accuracy_score(predictio
print(confusion_matrix(y_test_we, predictions))
get_performance_measures("Logisitic Regression", "tweets", y_test_we, p
```

```
Logistic Regression Train accuracy 0.8718505647263249
Logistic Regression Test accuracy 0.7074340527577938
[[ 38  35   8]
 [ 82 718 186]
 [ 14 163 424]]
Performance of Logisitic Regression on tweets.
```

Out[65]:
```
{'Accuracy': 0.7074340527577938,
 'Confusion Matrix': [[38, 35, 8], [82, 718, 186], [14, 163, 424]],
 'Class Specific Performance': {-1: {'TP': 38,
   'TN': 1142,
   'FP': 96,
   'FN': 43,
   'Precision': 0.4691358024691358,
   'Recall': 0.2835820895522388,
   'F': 0.3534883720930232,
   'TP Rate': 0.4691358024691358,
   'FP Rate': 0.07754442649434572},
  0: {'TP': 718,
   'TN': 462,
   'FP': 198,
   'FN': 268,
   'Precision': 0.7281947261663286,
   'Recall': 0.7838427947598253,
   'F': 0.7549947423764458,
   'TP Rate': 0.7281947261663286,
   'FP Rate': 0.3},
  1: {'TP': 424,
   'TN': 756,
   'FP': 194,
   'FN': 177,
   'Precision': 0.7054908485856906,
   'Recall': 0.686084142394822,
   'F': 0.6956521739130433,
   'TP Rate': 0.7054908485856906,
   'FP Rate': 0.20421052631578948}}}
```



OBSERVATION: The model performs relatively well using the GloVe embeddings. This stark difference in accuracy between the model that performed poorly using BOW embeddings and the model that used GloVe embeddings can be attributed to the way both embeddings are at their core.

(Shah, 2021), (Raschka, 2016), (scikit-learn, 2021c)

Now, we can try to improve the performance of the model by using Grid Search which would allow us to find the best hyperparameters given the data. It does that by trying out all the potential combinations of the hyperparameters given to it to a scoring metric (e.g. accuracy) that we want to optimise the model on. Cross validation is also used to optimise the model. (Shah, 2021)

This will be very much needed to improve the performance on the BOW. Conducting an extensive search for many parameters can be very time consuming due to the nature of testing thus the main hyperparameters that will be changed are "C" which represents the regularization and the "solver" which is the algorithm used.

Since the model overfits on BOW embeddings, the experimentation with regularization is improve the model's generalisation, i.e. improve the model's performance on the unseen data, thus preventing it from overfitting. Regularization is the penalty that gets added to the model. Since the goal is to prevent overfitting, the higher the value of regularization is, the higher the penalty is thus we are inducing bias as our model suffers from high variance. (Raschka, 2016)

OBSERVATION: After testing all models available for multi-classification i.e. [lbfgs,newton-cg, sag, saga], the types of solver the models generally gravitated towards were `lbfgs` and `saga` and so these two solvers were used with different `C` values to minimise the total running time of Grid Search.

In [66]:
```python
# a dictionary of hyperparameter values
param_dic = {
    'solver' : ['lbfgs','saga'],
    'C' : [1000, 100, 0.1, 1e2, 1e-5],
    'max_iter' : [1000]
}
```

In [67]:
```python
optim_logreg = GSCV(estimator = LogisticRegression(),
                    param_grid = param_dic,
                    verbose = 1, # gives the details of the process
                    #cv = 2 #default is 5 folds cross-val so decreased to
                  )
```

```
In [68]:  optim_bow = optim_logreg.fit(X_train_bow,y_train_bow)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linea
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_sag.py:352: ConvergenceWarning: The max_iter was reached which mean
s the coef_ did not converge
  warnings.warn(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_sag.py:352: ConvergenceWarning: The max_iter was reached which mean
s the coef_ did not converge
  warnings.warn(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_sag.py:352: ConvergenceWarning: The max_iter was reached which mean
s the coef_ did not converge
  warnings.warn(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_sag.py:352: ConvergenceWarning: The max_iter was reached which mean
s the coef_ did not converge
  warnings.warn(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_sag.py:352: ConvergenceWarning: The max_iter was reached which mean
s the coef_ did not converge
  warnings.warn(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [69]:

```python
print("BOW",optim_bow.best_params_)
```

```
BOW {'C': 1000, 'max_iter': 1000, 'solver': 'lbfgs'}
```

In [70]:
```
optim_glove = optim_logreg.fit(X_train_we,y_train_we)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linea
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\linear_
model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

In [71]:
```
print("Glove",optim_logreg.best_params_)
```

```
Glove {'C': 100, 'max_iter': 1000, 'solver': 'lbfgs'}
```

After Applying the Optimal Parameters

In [72]:
```
logreg = LogisticRegression(max_iter=10000, C=1000)
logreg.fit(X_train_bow,y_train_bow)
predictions = logreg.predict(X_test_bow)
prediction_prob = logreg.predict_proba(X_test_bow)
```

In [73]:
```
print('Logistic Regression Train accuracy on BOW %s' % logreg.score(X_t
print('Logistic Regression Test accuracy on BOW %s' % accuracy_score(pr
#get_performance_measures("Logisitic Regression", "tweets", y_test_bow,
```

```
Logistic Regression Train accuracy on BOW 0.8395598030697944
Logistic Regression Test accuracy on BOW 0.30335731414868106
```

In [74]:
```
logreg = LogisticRegression(max_iter=10000, C=100, solver="lbfgs")
logreg.fit(X_train_we,y_train_we)
predictions = logreg.predict(X_test_we)
prediction_prob = logreg.predict_proba(X_test_we)
```

In [75]:
```
print('Logistic Regression Train accuracy on Glove %s' % logreg.score(X
print('Logistic Regression Test accuracy on Glove %s' % accuracy_score(
#get_performance_measures("Logistic Regression", "tweets", y_test_we, p
```

```
Logistic Regression Train accuracy on Glove 0.8722849695916595
Logistic Regression Test accuracy on Glove 0.7080335731414868
```

OBSERVATION: Both BOW and GloVe embeddings have their optimal parameters as `C=100`, `solver=lbfgs` and `C=1000`, `solver=lbfgs` respectively. However, after running the logistic regression models with these updated hyperparameters, the results have not improved.

For the Glove embeddings, using the updated parameters increased the training data's accuracy by ~1%. The results have improved slightly by ~3% for training data accuracy on the BOW model and 0.004% decrease on the test data accuracy thus indicating that the issue of overfitting stems from elsewhere and trying to improve the hyperparameters of the models did not help in improving the generalization on the data.

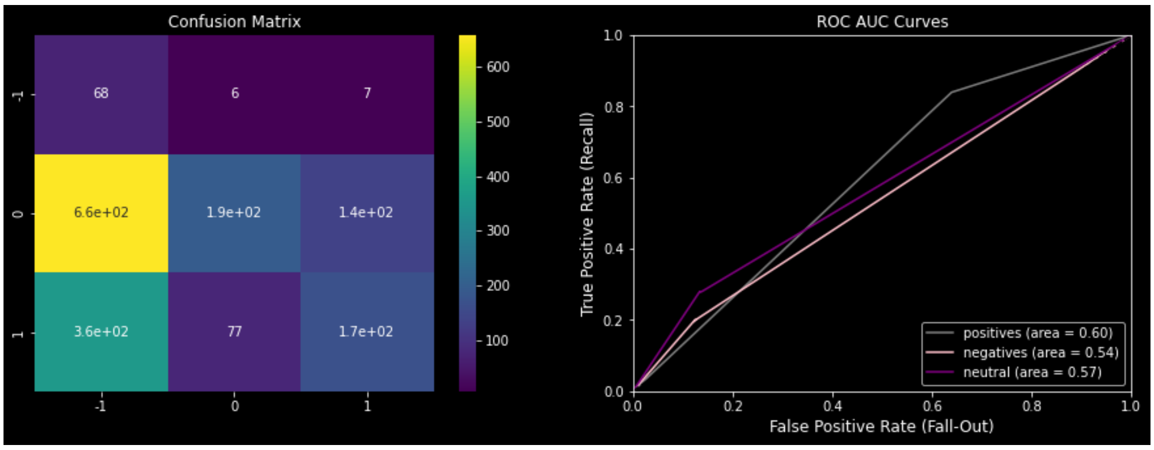
These results could possibly, for example, have to do with the limited number of training data given to the models which explains why the BOW model was overfitting.

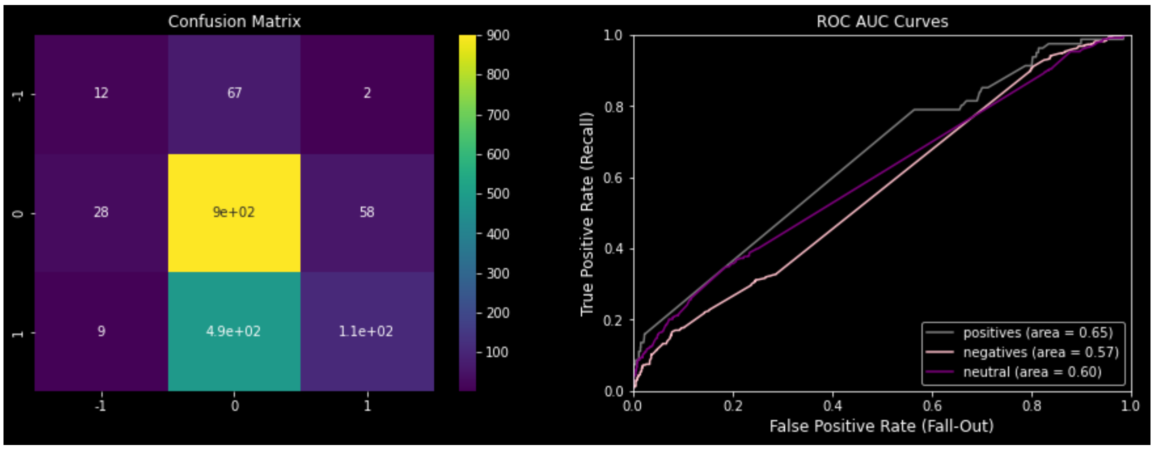
13.2. Naive Bayes Classifier

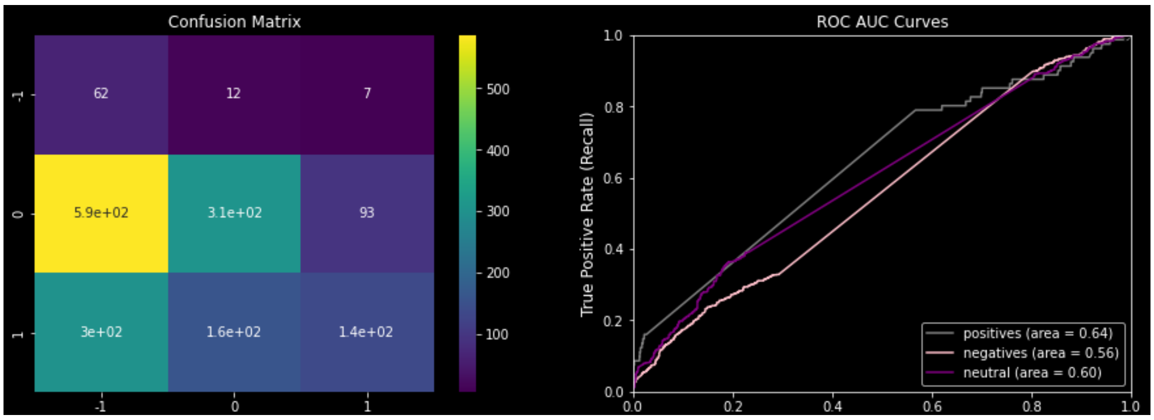
Naive bayes is a supervised machine learning algorithm that works based on bayes theorem. It presupposes that the presence of one feature in a class has nothing to do with the presence of any other feature. Although this theorem apparently provides "overly simplified assumptions", this was tested for the corpus due to its good performance in some situations (sometimes even performing better than more sophisticated models) and speed especially in large datasets. (scikit-learn, 2021a) (Analytics Vidhya, 2017)

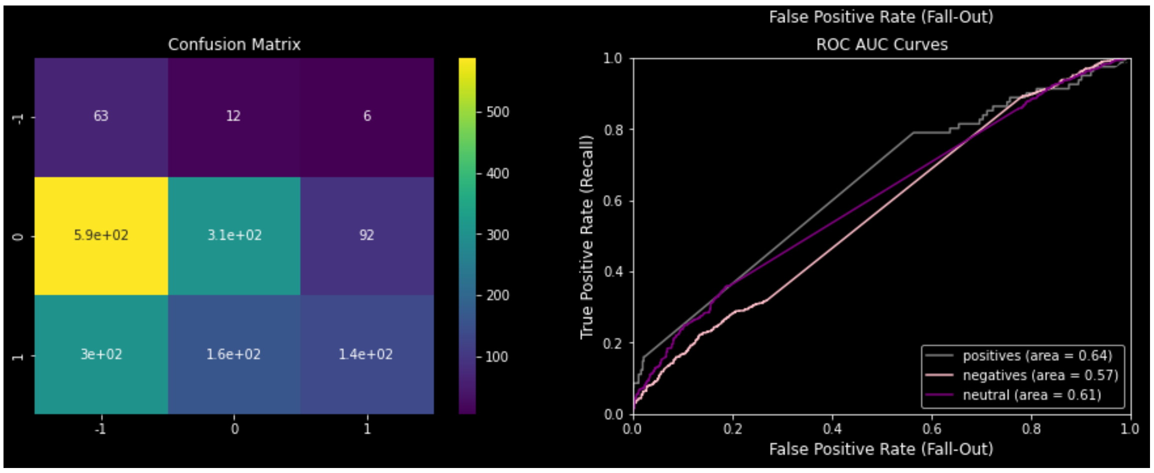
```
In [76]: models = [GaussianNB(), BernoulliNB(), MultinomialNB(), ComplementNB()]

         print("Testing on BOW embeddings")
         for nb_model in models:
             #print(nb_model)
             nb_model.fit(X_train_bow, y_train_bow)
             predictions = nb_model.predict(X_test_bow)
             prediction_prob = nb_model.predict_proba(X_test_bow)
             get_performance_measures(nb_model, "tweets", y_test_bow, prediction
             print(nb_model.score(X_test_bow, y_test_bow))
```

```
Testing on BOW embeddings
Performance of GaussianNB() on tweets.
0.2565947242206235
Performance of BernoulliNB() on tweets.
0.6103117505995204
Performance of MultinomialNB() on tweets.
0.30395683453237 41
Performance of ComplementNB() on tweets.
0.30335731414868106
```





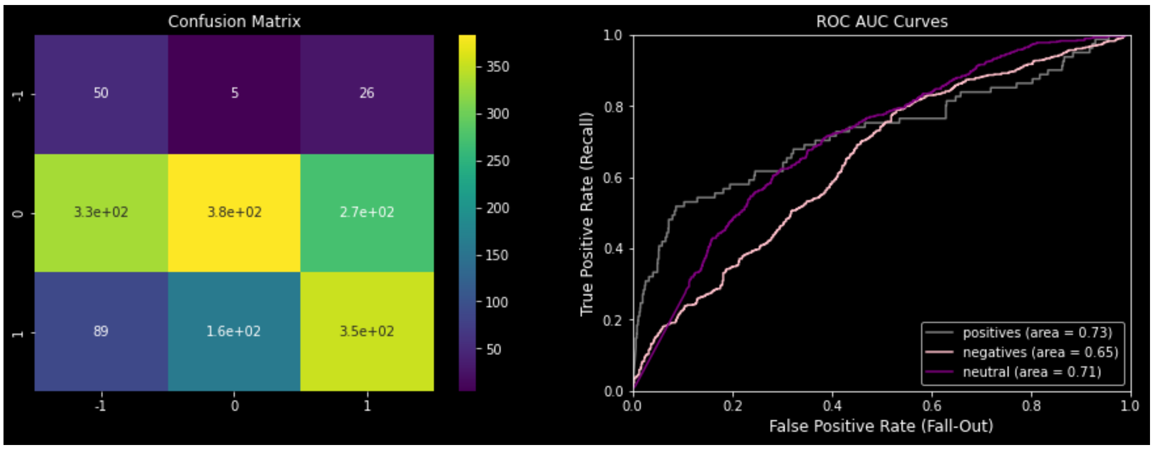


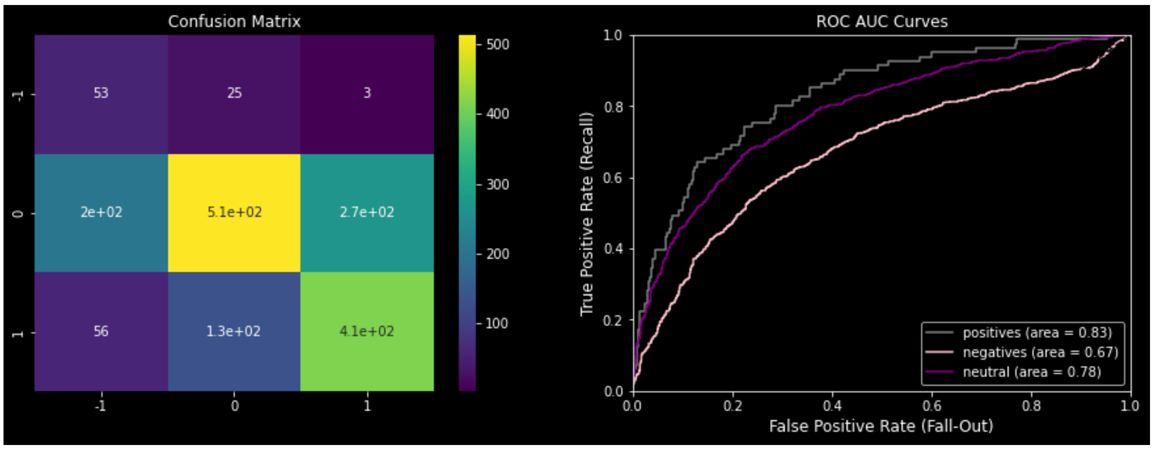


In [77]:

```
print("\nTesting on GloVe embeddings")
for nb_model in models[:2]:
    nb_model.fit(X_train_we, y_train_we)
    predictions = nb_model.predict(X_test_we)
    prediction_prob=nb_model.predict_proba(X_test_we)
    get_performance_measures(nb_model, "tweets", y_test_we, predictions
    print(nb_model.score(X_test_we, y_test_we))
```

```
Testing on GloVe embeddings
Performance of GaussianNB() on tweets.
0.4712230215827338
Performance of BernoulliNB() on tweets.
0.5875299760191847
```





OBSERVATION: For the model trained using the TFIDF vectors, the naive bayes classifier gave the highest accuracy when the it was trained using the glove word embeddings, the highest accuracy was achieved using BernouilliNB.

OBSERVATION: Curiously, the Naive Bayes Classifier performs better with BOW embeddings as opposed to GloVe embeddings. This may be due to the fact that both Naive Bayes Classifier and BOW embedding are probability based.

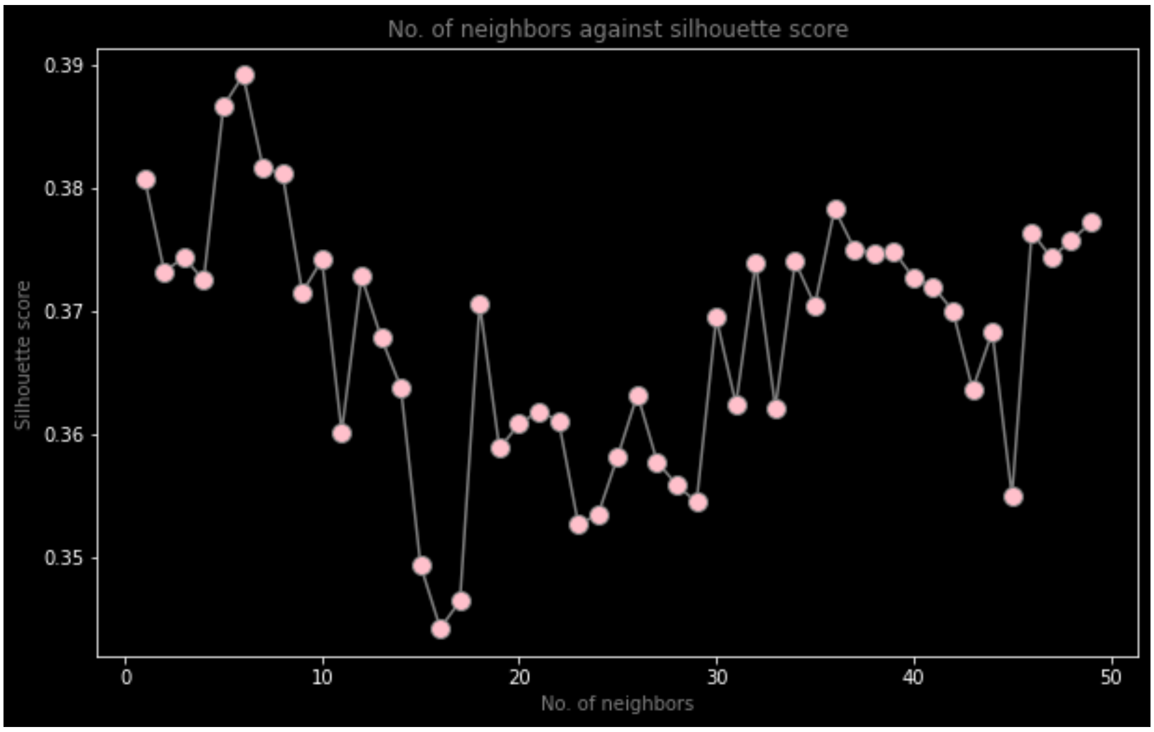
13.3. K-Nearest Neighbors Classifier

The K-nearest neighbor classifier is an unsupervised machine learning algorithm which is fairly easy to implement and has reasonably good predicting power. It can take in multiple parameters out of which the most crucial/relevent would be the number of neighbors (k). This algorithm works on the assumption that similar examples will group together. The "clusters" or neighborhoods are made according to the defined k. (Analytics Vidhya, 2018) (Harrison, 2018)

Working with BOW Embeddings

In [78]:
```
silhouette_scores=[]
for i in range(1,50):
    model = KNeighborsClassifier(i)
    model.fit(X_train_bow,y_train_bow)
    predictions = model.predict(X_test_bow)
    silhouette_scores.append(silhouette_score(X_test_bow, predictions))

plt.figure(figsize=(10,6))
plt.plot(range(1,50),silhouette_scores,color='gray', marker='o',markerf
plt.title('No. of neighbors against silhouette score',color="gray")
plt.xlabel('No. of neighbors',color="gray")
plt.ylabel('Silhouette score',color="gray")
```

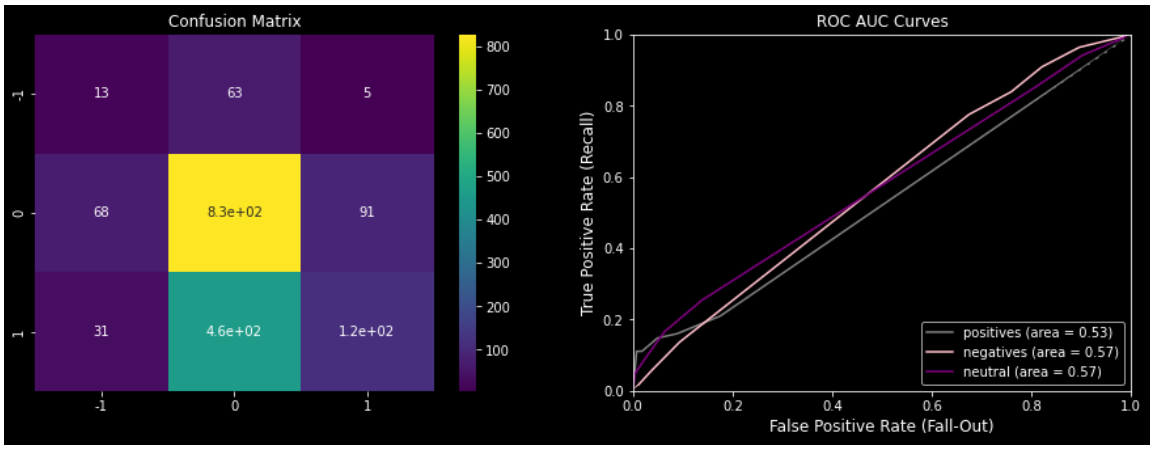
Out[78]:
```
Text(0, 0.5, 'Silhouette score')
```



```
In [79]: model =KNeighborsClassifier(n_neighbors=6)
         model.fit(X_train_bow,y_train_bow)
         predictions=model.predict(X_test_bow)

         print(f"accuracy score = {accuracy_score(predictions,y_test_bow)}")
         prediction_prob=model.predict_proba(X_test_bow)
         get_performance_measures("K nearest neighbor", "tweets", y_test_bow, pr
```

```
accuracy score = 0.5725419664268585
Performance of K nearest neighbor on tweets.
```

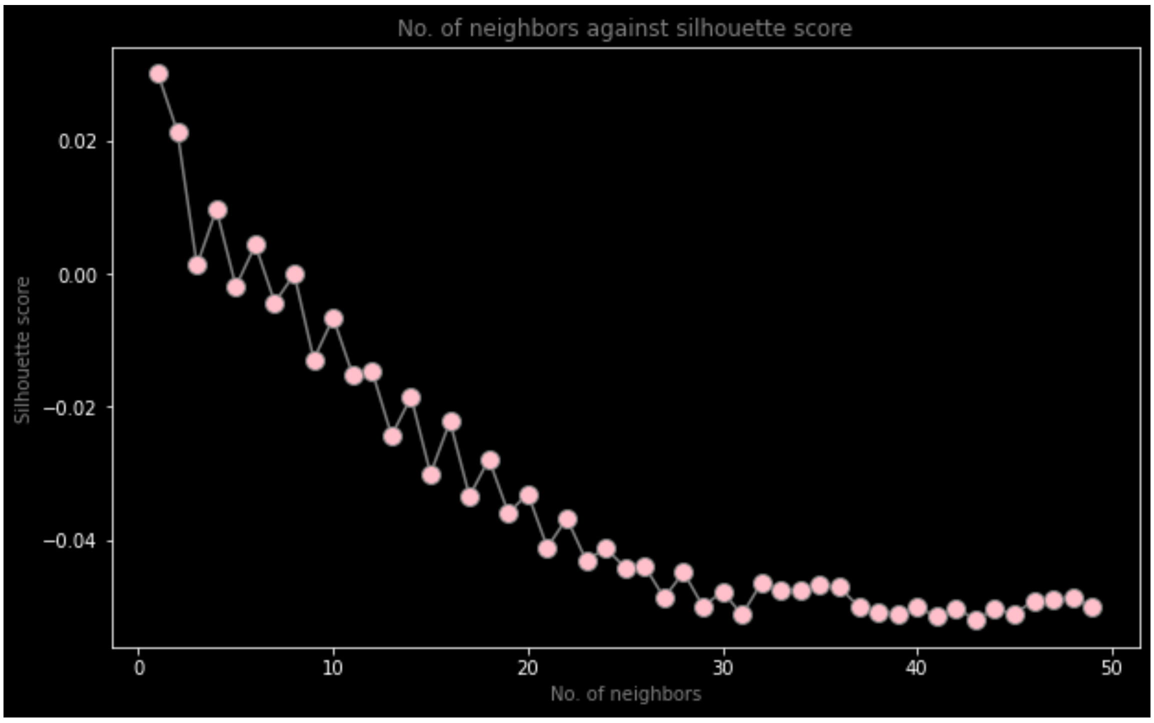
```
Out[79]: {'Accuracy': 0.5725419664268585,
          'Confusion Matrix': [[13, 63, 5], [68, 827, 91], [31, 455, 115]],
          'Class Specific Performance': {-1: {'TP': 13,
            'TN': 942,
            'FP': 99,
            'FN': 68,
            'Precision': 0.16049382716049382,
            'Recall': 0.11607142857142858,
            'F': 0.13471502590673576,
            'TP Rate': 0.16049382716049382,
            'FP Rate': 0.09510086455331412},
           0: {'TP': 827,
            'TN': 128,
            'FP': 518,
            'FN': 159,
            'Precision': 0.8387423935091278,
            'Recall': 0.6148698884758365,
            'F': 0.7095667095667096,
            'TP Rate': 0.8387423935091278,
            'FP Rate': 0.8018575851393189},
           1: {'TP': 115,
            'TN': 840,
            'FP': 96,
            'FN': 486,
            'Precision': 0.19134775374376040,
            'Recall': 0.5450236966824644,
            'F': 0.2832512315270936,
            'TP Rate': 0.1913477537437604,
            'FP Rate': 0.10256410256410256}}}
```



Working with GloVe Embeddings

In [80]:
```
silhouette_scores=[]
for i in range(1,50):
    model = KNeighborsClassifier(i)
    model.fit(X_train_we, y_train_we)
    predictions = model.predict(pd.DataFrame(X_test_we))
    silhouette_scores.append(silhouette_score(X_test_we, predictions))

plt.figure(figsize=(10,6))
plt.plot(range(1,50),silhouette_scores,color='gray', marker='o',  marke
plt.title('No. of neighbors against silhouette score',color="gray")
plt.xlabel('No. of neighbors',color="gray")
plt.ylabel('Silhouette score',color="gray")
```

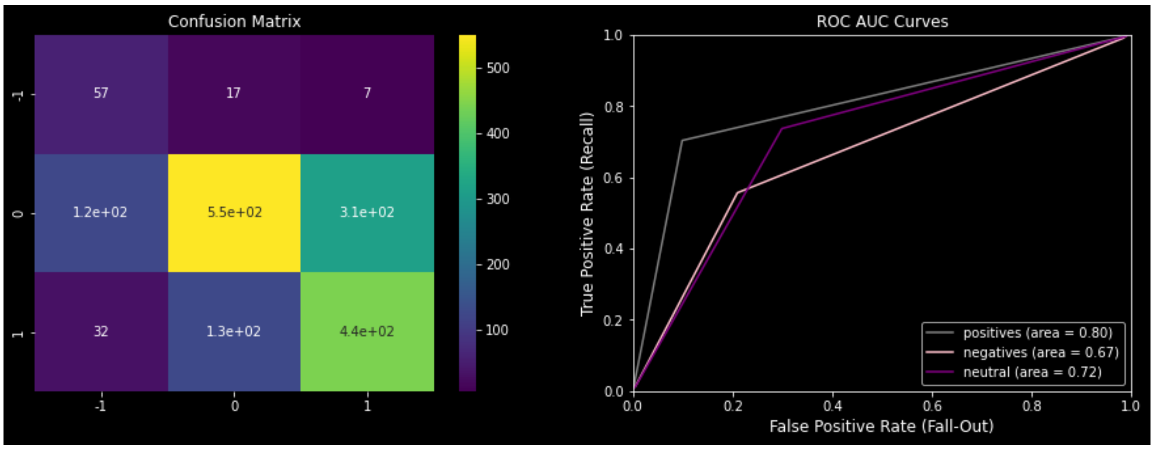
Out[80]:
```
Text(0, 0.5, 'Silhouette score')
```



```
In [81]:
model = KNeighborsClassifier(n_neighbors=1)
model.fit(X_train_we, y_train_we)
predictions=model.predict(X_test_we)

print(f"accuracy score = {accuracy_score(predictions,y_test_we)}")
prediction_prob=model.predict_proba(X_test_we)
get_performance_measures("K nearest neighbor", "tweets", y_test_we, pre
```

```
accuracy score = 0.6288968824940048
Performance of K nearest neighbor on tweets.
```

```
Out[81]: {'Accuracy': 0.6288968824940048,
 'Confusion Matrix': [[57, 17, 7], [125, 549, 312], [32, 126, 443]],
 'Class Specific Performance': {-1: {'TP': 57,
   'TN': 992,
   'FP': 157,
   'FN': 24,
   'Precision': 0.7037037037037037,
   'Recall': 0.26635514018691586,
   'F': 0.38644067796610165,
   'TP Rate': 0.7037037037037037,
   'FP Rate': 0.13664055700609226},
  0: {'TP': 549,
   'TN': 500,
   'FP': 143,
   'FN': 437,
   'Precision': 0.5567951318458418,
   'Recall': 0.7933526011560693,
   'F': 0.6543504171632897,
   'TP Rate': 0.5567951318458418,
   'FP Rate': 0.2223950233281493},
  1: {'TP': 443,
   'TN': 606,
   'FP': 319,
   'FN': 158,
   'Precision': 0.7371048252911814,
   'Recall': 0.5813648293963255,
   'F': 0.6500366837857667,
   'TP Rate': 0.7371048252911814,
   'FP Rate': 0.34486486486486484}}}
```



Checking if GridSearch can find parameters that may give better results.

Grid-search is used to identify the model's optimal hyperparameters that are used produce the most correct predictions by conducting an exhaustive search. The grid search tests all possible combinations of the parameters provided to it to retain the best possible combination for the data being fitted. It is a method that helps preserve effort resources and time.(Joseph, 2018) (scikit-learn, 2021) (scikit-learn, 2012) (Malik, 2020)

In [82]:
```
optim_para = {'p': (1, 2), 'weights': ('uniform', 'distance')}
optim_knn_bow = GSCV(KNeighborsClassifier(n_neighbors=6), param_grid=op
optim_knn_glove = GSCV(KNeighborsClassifier(n_neighbors=1), param_grid=
optim_knn_glove.fit(X_train_bow,y_train_bow)
optim_knn_bow.fit(X_train_bow,y_train_bow)
print("Glove",optim_knn_glove.best_params_)
print("BOW",optim_knn_bow.best_params_)
```

```
Glove {'p': 1, 'weights': 'uniform'}
BOW {'p': 1, 'weights': 'distance'}
```

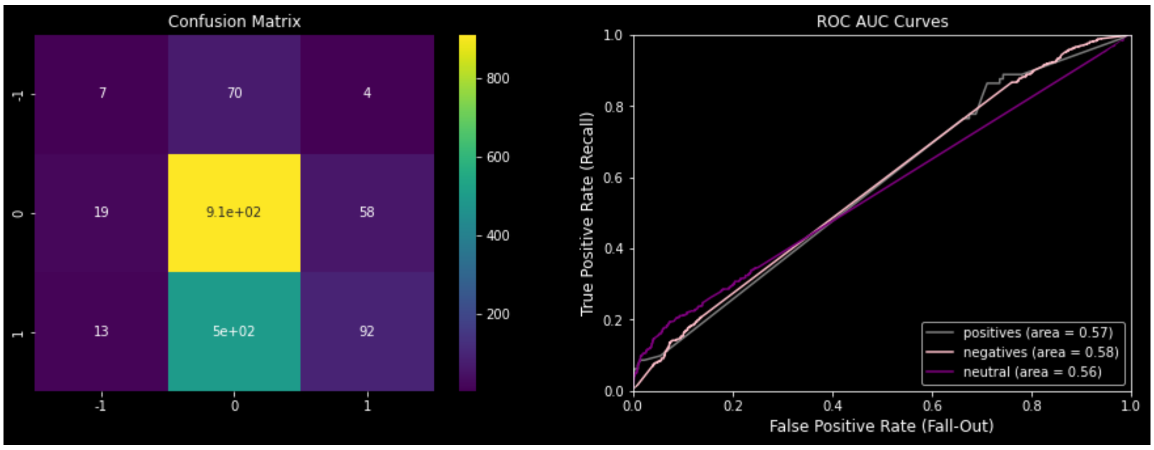
In [83]:
```
X_test_bow.shape
```

Out[83]:
```
(1668, 5965)
```

In [84]:
```
model_bow = KNeighborsClassifier(n_neighbors=6, p=1, weights='distance'
model_bow.fit(X_train_bow, y_train_bow)
predictions = model_bow.predict(X_test_bow)
print(f"accuracy score = {accuracy_score(predictions, y_test_bow)}")
prediction_prob = model_bow.predict_proba(X_test_bow)
get_performance_measures("K nearest neighbor", "tweets", y_test_bow, pr
```

```
accuracy score = 0.60431654676259
Performance of K nearest neighbor on tweets.
```

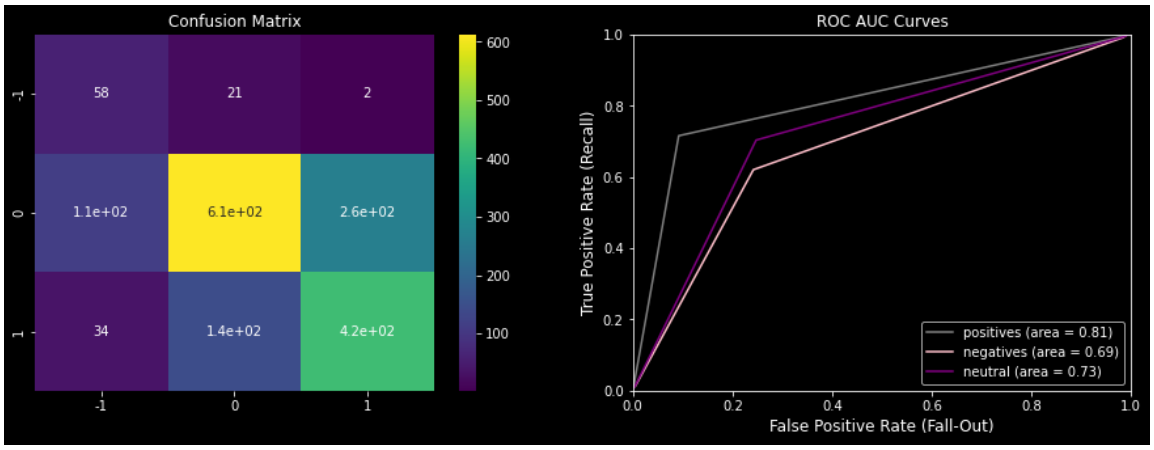
Out[84]:
```
{'Accuracy': 0.60431654676259,
 'Confusion Matrix': [[7, 70, 4], [19, 909, 58], [13, 496, 92]],
 'Class Specific Performance': {-1: {'TP': 7,
   'TN': 1001,
   'FP': 32,
   'FN': 74,
   'Precision': 0.08641975308641975,
   'Recall': 0.1794871794871795,
   'F': 0.11666666666666667,
   'TP Rate': 0.08641975308641975,
   'FP Rate': 0.030977734753146177},
  0: {'TP': 909,
   'TN': 99,
   'FP': 566,
   'FN': 77,
   'Precision': 0.9219066937119675,
   'Recall': 0.616271186440678,
   'F': 0.7387240958959772,
   'TP Rate': 0.9219066937119675,
   'FP Rate': 0.8511278195488722},
  1: {'TP': 92,
   'TN': 916,
   'FP': 62,
   'FN': 509,
   'Precision': 0.15307820299500832,
   'Recall': 0.5974025974025974,
   'F': 0.24370860927152319,
   'TP Rate': 0.15307820299500832,
   'FP Rate': 0.06339468302658487}}}
```



```
In [85]: model_glove = KNeighborsClassifier(n_neighbors=1, p=1, weights='uniform
         model_glove.fit(X_train_we, y_train_we)
         predictions=model_glove.predict(X_test_we)
         print(f"accuracy score = {accuracy_score(predictions,y_test_we)}")
         prediction_prob=model_glove.predict_proba(X_test_we)
         get_performance_measures("K nearest neighbor", "tweets", y_test_we, pre
```

```
accuracy score = 0.6552757793764988
Performance of K nearest neighbor on tweets.
```

```
Out[85]: {'Accuracy': 0.6552757793764988,
          'Confusion Matrix': [[58, 21, 2], [112, 612, 262], [34, 144, 423]],
          'Class Specific Performance': {-1: {'TP': 58,
            'TN': 1035,
            'FP': 146,
            'FN': 23,
            'Precision': 0.7160493827160493,
            'Recall': 0.28431372549019607,
            'F': 0.4070175438596491,
            'TP Rate': 0.7160493827160493,
            'FP Rate': 0.12362404741744284},
           0: {'TP': 612,
            'TN': 481,
            'FP': 165,
            'FN': 374,
            'Precision': 0.6206896551724138,
            'Recall': 0.7876447876447876,
            'F': 0.6942711287577992,
            'TP Rate': 0.6206896551724138,
            'FP Rate': 0.25541795665634676},
           1: {'TP': 423,
            'TN': 670,
            'FP': 264,
            'FN': 178,
            'Precision': 0.7038269550748752,
            'Recall': 0.6157205240174672,
            'F': 0.656832298136646,
            'TP Rate': 0.7038269550748752,
            'FP Rate': 0.2826552462526767}}}
```



13.4. Neural Network Classifier

Neural networks are models created by layering neurons/perceptrons such that there is an input layer, an output layer and 1 or more hidden layers. Each perceptron is by default a linear regressor. Whats special about a neural network is that it can be made to be non linear by simply using a non-linear function as activation function of each neuron. As the no. of layers in the neural network increases, the no. of weights and biases of the network increases and it becomes capable of learning more complex tasks. A neural network with lots of hidden layers is called a deep neural network. (IBM Cloud Education, 2017)

NLP presents a uniquely complex problem space for ML due to the inherent complexity of human language and the fact that words/phrases have different meanings under varying context. Thus, deep neural networks are often employed w.r.t. NLP tasks.

Thus here, a neural network with 9 hidden layers shall be trained to try and predict the sentiment (positive, negative, neutral) from input text. Stochastic Gradient Descent shall be the chosen optimization algorithm here.

Testing on GloVe Embeddings

In [86]:
```
def make_model_we():
    ''' Creates and returns NN model. '''
    nn_model = Sequential()

    nn_model.add(Dense(units=200, activation='tanh')) # input layer

    nn_model.add(Dense(units=200, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=200, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=200, activation='tanh')) # hidden layer
    nn_model.add(Dropout(rate=0.5))

    nn_model.add(Dense(units=100, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=100, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=100, activation='tanh')) # hidden layer
    nn_model.add(Dropout(rate=0.5))

    nn_model.add(Dense(units=50, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=50, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=50, activation='tanh')) # hidden layer
    nn_model.add(Dropout(rate=0.5))

    nn_model.add(Dense(units=3, activation='softmax')) # output layer
    nn_model.compile(optimizer='sgd', loss='sparse_categorical_crossent
    return nn_model
```

In [87]:
```
X_train = X_train_we
y_train = np.where(y_train_we==-1, 2, y_train_we)
X_test = X_test_we
y_test = np.where(y_test_we==-1, 2, y_test_we)
```

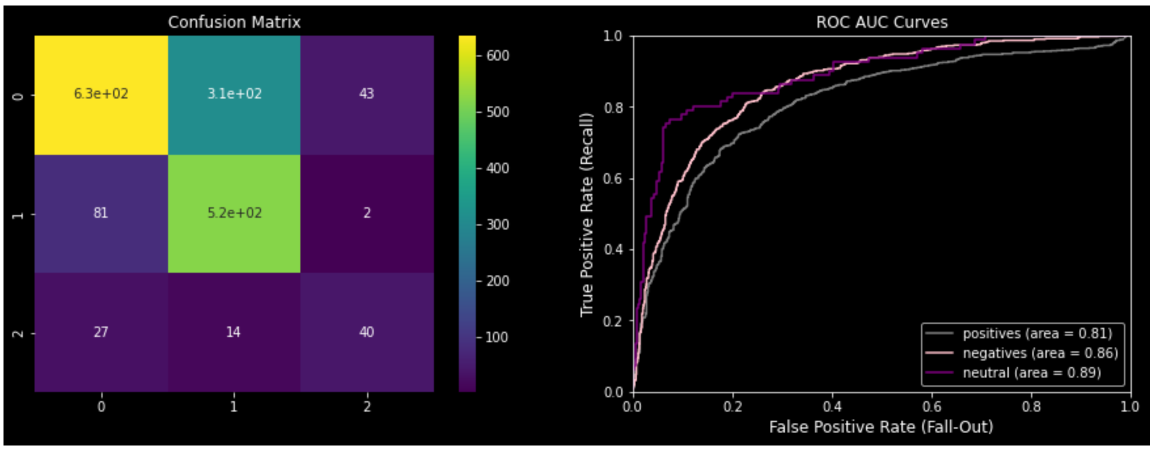
In [88]:
```
nn_model = make_model_we()
nn_model.fit(x=X_train, y=y_train, epochs=20)
pred_prob = nn_model.predict(X_test)
pred = np.array([np.argmax(p) for p in pred_prob])
print("\ntrain results")
nn_model.evaluate(X_train, y_train, verbose=1)
print("test results")
test_loss, test_acc = nn_model.evaluate(X_test,  y_test, verbose=1)
```

```
Epoch 1/20
216/216 [==============================] - 2s 3ms/step - loss: 0.9507 - ac
curacy: 0.5308
Epoch 2/20
216/216 [==============================] - 1s 3ms/step - loss: 0.6871 - ac
curacy: 0.6690
Epoch 3/20
216/216 [==============================] - 1s 3ms/step - loss: 0.6034 - ac
curacy: 0.7326
Epoch 4/20
216/216 [==============================] - 1s 4ms/step - loss: 0.5314 - ac
curacy: 0.7777
Epoch 5/20
216/216 [==============================] - 1s 3ms/step - loss: 0.5207 - ac
curacy: 0.7841
Epoch 6/20
216/216 [==============================] - 1s 3ms/step - loss: 0.4743 - ac
curacy: 0.8094
Epoch 7/20
216/216 [==============================] - 1s 3ms/step - loss: 0.4752 - ac
curacy: 0.8123
Epoch 8/20
216/216 [==============================] - 1s 5ms/step - loss: 0.4378 - ac
curacy: 0.8270
Epoch 9/20
216/216 [==============================] - 1s 4ms/step - loss: 0.4074 - ac
curacy: 0.8469
Epoch 10/20
216/216 [==============================] - 1s 6ms/step - loss: 0.4191 - ac
curacy: 0.8367
Epoch 11/20
216/216 [==============================] - 1s 4ms/step - loss: 0.3849 - ac
curacy: 0.8538
Epoch 12/20
216/216 [==============================] - 1s 3ms/step - loss: 0.3598 - ac
curacy: 0.8635
Epoch 13/20
216/216 [==============================] - 1s 3ms/step - loss: 0.3751 - ac
curacy: 0.8642
Epoch 14/20
216/216 [==============================] - 1s 3ms/step - loss: 0.3537 - ac
curacy: 0.8665
Epoch 15/20
216/216 [==============================] - 1s 3ms/step - loss: 0.3454 - ac
curacy: 0.8704
Epoch 16/20
216/216 [==============================] - 1s 3ms/step - loss: 0.3443 - ac
curacy: 0.8746
Epoch 17/20
216/216 [==============================] - 1s 3ms/step - loss: 0.3424 - ac
```

```
curacy: 0.8749
Epoch 18/20
216/216 [==============================] - 1s 3ms/step - loss: 0.3247 - ac
curacy: 0.8820
Epoch 19/20
216/216 [==============================] - 1s 3ms/step - loss: 0.2926 - ac
curacy: 0.8955
Epoch 20/20
216/216 [==============================] - 1s 4ms/step - loss: 0.3057 - ac
curacy: 0.8907

train results
216/216 [==============================] - 1s 4ms/step - loss: 0.2551 - ac
curacy: 0.8995
test results
53/53 [==============================] - 0s 3ms/step - loss: 0.8042 - accu
racy: 0.7146
```

In [89]:
```python
performance_metrics = get_performance_measures(
    classifier_name = "neural network",
    df_name = "glove embedded tweets",
    truth = y_test,
    prediction = pred,
    prediction_prob = pred_prob,
    labels = [0, 1, 2]
)
print_dict(performance_metrics)
```

```
Performance of neural network on glove embedded tweets.
Accuracy: 0.7146282973621103
Confusion Matrix: [[634, 309, 43], [81, 518, 2], [27, 14, 40]]
Class Specific Performance:
        0:
                TP: 634
                TN: 558
                FP: 108
                FN: 352
                Precision: 0.6430020283975659
                Recall: 0.8544474393530997
                F: 0.7337962962962962
                TP Rate: 0.6430020283975659
                FP Rate: 0.16216216216216217
        1:
                TP: 518
                TN: 674
                FP: 323
                FN: 83
                Precision: 0.8618968386023295
                Recall: 0.6159334126040428
                F: 0.7184466019417476
                TP Rate: 0.8618968386023295
                FP Rate: 0.3239719157472417
        2:
                TP: 40
                TN: 1152
                FP: 45
                FN: 41
                Precision: 0.49382716049382713
                Recall: 0.47058823529411764
                F: 0.4819277108433735
                TP Rate: 0.49382716049382713
                FP Rate: 0.03759398496240601
```



OBSERVATION: Despite adding dropout layers, the model is overfitting.

Following are 4 main reasons for overfitting in general. (Simplilearn, 2021)

1. Training data is not cleaned/has noise.
2. Model has high variance.
3. Insufficient amount of training data.
4. Model is too complex.

W.r.t. the given situation, the training data was cleaned and lemmatized. Text data in general, especially social media data is noisy. The amount of data used to train here are only a few thousand samples even after oversampling because of the limits set by tweepy api adhering to which one must extract tweets (can fetch limited no. of tweets in set time intervals, can no longer fetch tweets from specific intervals of time etc.).

Thus, the model's high variance here, is attributed to the lack in amount of training data and the inherent complexity of NLP tasks such as sentiment analysis.

Nevertheless, it can be observed that the model still achieves considerably good accuracy that is comparable to results of NLP experiments conducted by others like (S. Li, 2018).

Verify Using Cross Validation

In K-fold cross validation, data is split into k portions and the model is trained and tested k times with different different portions of the data as train and test data each time. This is often done when there is limited or largely varying data to check if the model performs consistently on different data distributions. Usually 10 fold cross validation is performed. However, here, 3 fold-cross validation shall be performed to reduce computation time.

In [90]:
```
df = pd.DataFrame({
    "tweet": DF["text"],
    "sentiment": DF["sentiment"].replace(-1, 2)
})
display(df.head(5))
```

	tweet	sentiment
0	day	0
1	celebrate national day slovak republic expo du...	1
2	vertebral deformity measurement mri ct radiogr...	0
3	🚨 important statement 🇪 yemeni arm force come ...	0
4	hopeless imagine pathetic woman deliver absolu...	2

In [91]:
```
X = np.array(glove_embed(df["tweet"]))
y = df["sentiment"]
X.shape, y.shape
```

Out[91]:
```
((5560, 200), (5560,))
```

In [92]:
```
from sklearn.model_selection import StratifiedKFold
kf = StratifiedKFold(n_splits=3, random_state=23, shuffle=True)
```

In [93]:

```python
train_accuracy = []
test_accuracy = []
for train_slice, test_slice in kf.split(X, y):
    X_train = X[train_slice]
    X_test = X[test_slice]
    y_train = y[train_slice]
    y_test = y[test_slice]
    X_train, y_train = SMOTE_MODEL.fit_resample(X_train, y_train)
    nn_model = make_model_we()
    nn_model.fit(x=X_train, y=y_train, epochs=20)
    train_accuracy.append(nn_model.evaluate(X_train, y_train)[1])
    test_accuracy.append(nn_model.evaluate(X_test, y_test)[1])
print("\ntrain scores =", train_accuracy)
print("test scores =", test_accuracy)
print("average train accuracy =", sum(train_accuracy)/len(train_accurac
print("average test accuracy =", sum(test_accuracy)/len(test_accuracy))
```

```
Epoch 1/20
206/206 [==============================] - 1s 5ms/step - loss: 0.8880 - ac
curacy: 0.5674
Epoch 2/20
206/206 [==============================] - 1s 3ms/step - loss: 0.6583 - ac
curacy: 0.7102
Epoch 3/20
206/206 [==============================] - 1s 4ms/step - loss: 0.5650 - ac
curacy: 0.7608
Epoch 4/20
206/206 [==============================] - 1s 4ms/step - loss: 0.5312 - ac
curacy: 0.7877
Epoch 5/20
206/206 [==============================] - 1s 4ms/step - loss: 0.4921 - ac
curacy: 0.8005
Epoch 6/20
206/206 [==============================] - 1s 4ms/step - loss: 0.4625 - ac
curacy: 0.8169
Epoch 7/20
```

OBSERVATION: Since similar train and test accuracies were obtained for train and test sets across 3-fold cross validation, it can be confirmed that the model performs consistently as seen before on different proportions of train and test data.

Testing on BOW Embeddings

In [94]:

```python
X_train = X_train_bow
y_train = np.where(y_train_bow==-1, 2, y_train_bow)
X_test = X_test_bow
y_test = np.where(y_test_bow==-1, 2, y_test_bow)
print(X_train.shape, y_train.shape)
```

```
(6906, 5965) (6906,)
```

In [95]:
```python
def make_model_bow():
    ''' Creates and returns NN model. '''
    nn_model = Sequential()

    nn_model.add(Dense(units=5964, activation='tanh')) # input layer

    nn_model.add(Dense(units=200, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=200, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=200, activation='tanh')) # hidden layer
    nn_model.add(Dropout(rate=0.5))

    nn_model.add(Dense(units=100, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=100, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=100, activation='tanh')) # hidden layer
    nn_model.add(Dropout(rate=0.5))

    nn_model.add(Dense(units=50, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=50, activation='tanh')) # hidden layer
    nn_model.add(Dense(units=50, activation='tanh')) # hidden layer
    nn_model.add(Dropout(rate=0.5))

    nn_model.add(Dense(units=3, activation='softmax')) # output layer
    nn_model.compile(optimizer='sgd', loss='sparse_categorical_crossent
    return nn_model
```

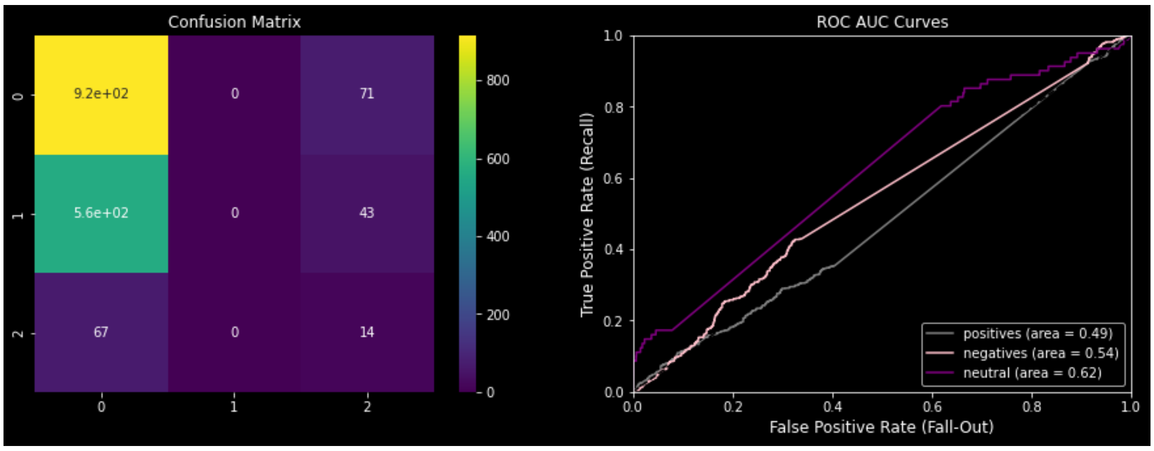
In [96]:
```
nn_model = make_model_bow()
nn_model.fit(x=X_train, y=y_train, epochs=10)
pred_prob = nn_model.predict(X_test)
pred = np.array([np.argmax(p) for p in pred_prob])
print("\ntrain results")
nn_model.evaluate(X_train, y_train, verbose=1)
print("test results")
test_loss, test_acc = nn_model.evaluate(X_test,  y_test, verbose=1)
```

```
Epoch 1/10
216/216 [==============================] - 2s 5ms/step - loss: 1.1048 - ac
curacy: 0.3338
Epoch 2/10
216/216 [==============================] - 1s 6ms/step - loss: 1.1047 - ac
curacy: 0.3346
Epoch 3/10
216/216 [==============================] - 2s 8ms/step - loss: 1.1026 - ac
curacy: 0.3478
Epoch 4/10
216/216 [==============================] - 2s 8ms/step - loss: 1.1030 - ac
curacy: 0.3319
Epoch 5/10
216/216 [==============================] - 2s 7ms/step - loss: 1.1005 - ac
curacy: 0.3501
Epoch 6/10
216/216 [==============================] - 2s 7ms/step - loss: 1.0953 - ac
curacy: 0.3562
Epoch 7/10
216/216 [==============================] - 2s 9ms/step - loss: 1.0924 - ac
curacy: 0.3636
Epoch 8/10
216/216 [==============================] - 1s 6ms/step - loss: 1.0880 - ac
curacy: 0.3752
Epoch 9/10
216/216 [==============================] - 1s 6ms/step - loss: 1.0795 - ac
curacy: 0.3814
Epoch 10/10
216/216 [==============================] - 1s 6ms/step - loss: 1.0624 - ac
curacy: 0.4159

train results
216/216 [==============================] - 1s 3ms/step - loss: 1.0279 - ac
curacy: 0.5072
test results
53/53 [==============================] - 0s 4ms/step - loss: 1.0458 - accu
racy: 0.5570
```

In [97]:
```
get_performance_measures(
    classifier_name = "neural network",
    df_name = "glove embedded tweets",
    truth = y_test,
    prediction = pred,
    prediction_prob = pred_prob,
    labels = [0, 1, 2]
)
```

```
Performance of neural network on glove embedded tweets.

C:\Users\GAYATH~1\AppData\Local\Temp/ipykernel_13116/2687044907.py:19: R
timeWarning: invalid value encountered in longlong_scalars
  recall = tp/(tp+fp)
```

Out[97]:
```
{'Accuracy': 0.5569544364508393,
 'Confusion Matrix': [[915, 0, 71], [558, 0, 43], [67, 0, 14]],
 'Class Specific Performance': {0: {'TP': 915,
   'TN': 14,
   'FP': 625,
   'FN': 71,
   'Precision': 0.9279918864097363,
   'Recall': 0.5941558441558441,
   'F': 0.7244655581947743,
   'TP Rate': 0.9279918864097363,
   'FP Rate': 0.9780907668231612},
  1: {'TP': 0,
   'TN': 929,
   'FP': 0,
   'FN': 601,
   'Precision': 0.0,
   'Recall': nan,
   'F': nan,
   'TP Rate': 0.0,
   'FP Rate': 0.0},
  2: {'TP': 14,
   'TN': 915,
   'FP': 114,
   'FN': 67,
   'Precision': 0.1728395061728395,
   'Recall': 0.109375,
   'F': 0.1339712918660287,
   'TP Rate': 0.1728395061728395,
   'FP Rate': 0.11078717201166181}}}
```



OBSERVATION: It can be seen that while the NN that worked with document embeddings produced using the GloVe model gave accuracy > 75%, with bag of words embeddings (top 20% bi-grams as per TF-IDF scoring), resulted in very poor accuracy of just over 35%. This however, was expected because BOW embeddings do not capture any meaning/context of words which is crucial to sentiment analysis while the GloVe embeddings do capture some meaning. Hence, the results of comparing sentiment classification accuracy achieved using GloVe Vs BOW document embeddings have produced in expected results.

OBSERVATION: The following paragraphs provide a quick overview of all insights thus achived through experimentation with ANNs for tweet sentiment analysis.

Different experiments were conducted before settling on the existing model architecture.

- The model began with just 3 hidden layers. This model underfitted tremendously.
- Then, layers were increased and trials were conducted with different no. of neurons and layers. It was found that very large no. of layers ike 20 or so, led to low bias but high variance leading to higher degree of overfitting. Larger models also required a larger amount of computing time. So, layers were reduced. The current model with 9 hidden layers also show some degree of overfitting despite adding counter measures such as Dropout layers. This is estimated to be because of 2 main reasons.
    - A simple ANN model such as this one has insufficient "Capacity" to fully grasp the increasingly complex problem space of a challenging NLP task such as that of "Sentiment Analysis". (sdiabr, 2018)
    - The amount of data used for training was too little. Especially for the negative class due to highly varied data distribution in the tweets which had to be accounted for using SMOTE.

- It was also observed that varying no. of neurons in a layer did'nt make as much difference as did increasing the no. of layers. This is likely because while increasing neurons can make the model more non-linear, it is increasing layers that present it with better opportunity to fine tune learning as more layers contribute to substantially more no. of tunable parameters. (Quora, 2019)
- The accuracy achieved my the current model is fairly acceptable at over 70% for such a relatively simple model. This accuracy while not ideal is still, comparable to that of other models that try to perform NLP tasks. (P. Borele and D.A. Borikar, 2016)

LSTM

RNN (Recurrent Neural Network) is a supervised deep learning algorithm that uses information from previous neurons as input to the next neuron. A major drawback of RNNs is the vanishing gradient problem.

To fix this issue, academic researchers now opt to use a variant of RNN, i.e., LSTMs

LSTMs (Long short-term memory) have a memory cell that efficiently carries information from a particular neuron to the upcoming neurons. This makes LSTMs better than RNNs as they can remember quite a lot of information from the previous neurons. The data from the current neuron and the previous neuron’s hidden layer result are fed as input to the LSTM. Alongside different layers, various activation functions such as ReLU, sigmoid, etc. are used for computing the output. (Karra, 2021)

OBSERVATION: LSTMs only take in real numbers as input (Abishek PSS, 2021). To map each word to a real number, keras offers a function `Tokenizer()` , that tokenizes each sentence and maps each word of the sentence to a real numbers. Since we opted to use NLTK's `word_tokenize()` function to tokenize the tweet bodies, hence the LSTM failed to work in our case. NLTK's `word_tokenize()` and keras' `Tokenizer()` work differently. The `word_tokenize()` function just tokenizes the sentences and unlike `Tokenizer()` , it doesn't map each unique word to a real number. As the LSTM failed to work after multiple tries, we therefore decided to use an ANN (Artificial Neural Network).

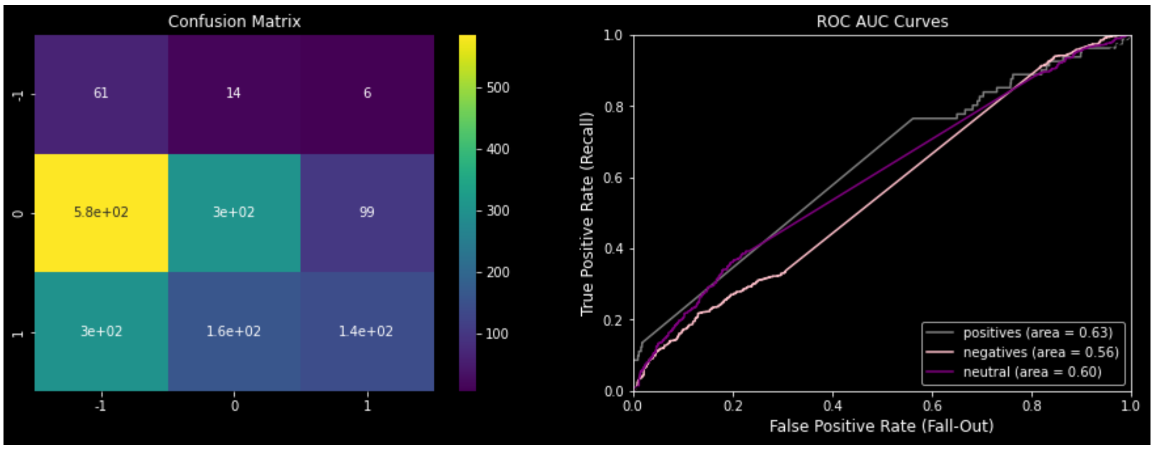
13.5. Linear Support Vector Machine Classifier

SVM or support vector machines are a collection of machine learning methods for data classification and more. On one hand, SVMs are extremely capable in situations that involve spaces with high dimensionality. It is also highly memory efficient. On the other hand, the probability estimates are calculated using an expensive n-fold cross validation unlike other classifiers. SVC, LinearSVC and NuSVC are the classes that take care of the classification.

```
In [98]: svm_classifier_bow = svm.LinearSVC()
         svm_classifier_bow.fit(X_train_bow,y_train_bow)
         prediction_svm_bow = svm_classifier_bow.predict(X_test_bow)
         pred_prob = np.array(svm_classifier_bow.decision_function(X_test_bow))
         print(accuracy_score(prediction_svm_bow ,y_test_bow))
         get_performance_measures(svm_classifier_bow, "tweets", y_test_bow, pred
```

```
0.302757793764988
Performance of LinearSVC() on tweets.
```

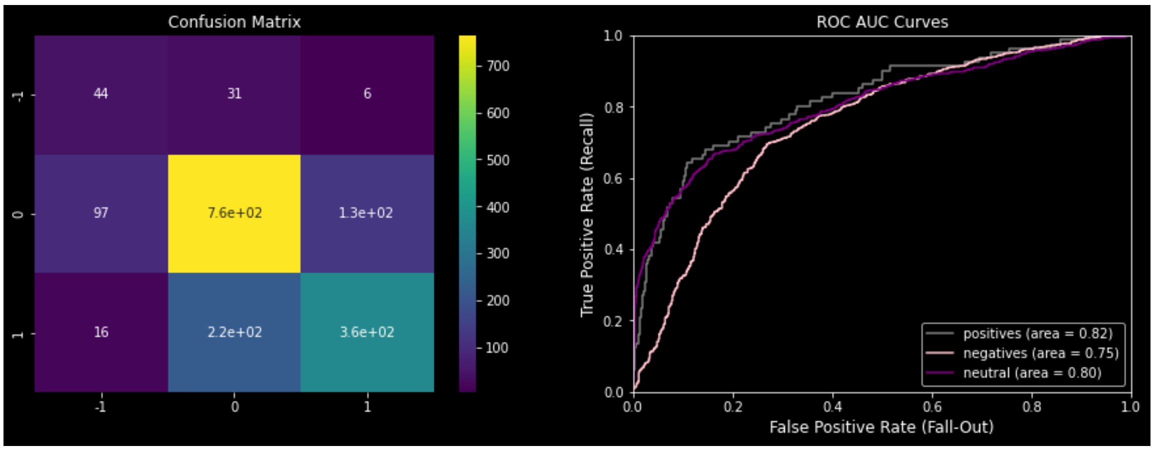
```
Out[98]: {'Accuracy': 0.302757793764988,
          'Confusion Matrix': [[61, 14, 6], [584, 303, 99], [300, 160, 141]],
          'Class Specific Performance': {-1: {'TP': 61,
            'TN': 444,
            'FP': 884,
            'FN': 20,
            'Precision': 0.7530864197530864,
            'Recall': 0.06455026455026455,
            'F': 0.1189083820662768,
            'TP Rate': 0.7530864197530864,
            'FP Rate': 0.6656626506024096},
           0: {'TP': 303,
            'TN': 202,
            'FP': 174,
            'FN': 683,
            'Precision': 0.30730223123732253,
            'Recall': 0.6352201257861635,
            'F': 0.41421736158578265,
            'TP Rate': 0.30730223123732253,
            'FP Rate': 0.4627659574468085},
           1: {'TP': 141,
            'TN': 364,
            'FP': 105,
            'FN': 460,
            'Precision': 0.23460898502495842,
            'Recall': 0.573170731707317,
            'F': 0.33293978748524206,
            'TP Rate': 0.23460898502495842,
            'FP Rate': 0.22388059701492538}}}
```



In [99]:
```
svm_classifier_we = svm.LinearSVC()
svm_classifier_we.fit(X_train_we, y_train_we)
prediction_svm_we = svm_classifier_we.predict(pd.DataFrame(X_test_we))
pred_prob = np.array(svm_classifier_we.decision_function(X_test_we))
print(accuracy_score(prediction_svm_we ,y_test_we))
get_performance_measures(svm_classifier_we, "tweets", y_test_we, predic
```

```
0.7008393285371702
Performance of LinearSVC() on tweets.

C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\svm\_
se.py:1206: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
  warnings.warn(
```

Out[99]:
```
{'Accuracy': 0.7008393285371702,
 'Confusion Matrix': [[44, 31, 6], [97, 763, 126], [16, 223, 362]],
 'Class Specific Performance': {-1: {'TP': 44,
   'TN': 1125,
   'FP': 113,
   'FN': 37,
   'Precision': 0.5432098765432098,
   'Recall': 0.2802547770700637,
   'F': 0.3697478991596639,
   'TP Rate': 0.5432098765432098,
   'FP Rate': 0.0912762520193861},
  0: {'TP': 763,
   'TN': 406,
   'FP': 254,
   'FN': 223,
   'Precision': 0.7738336713995944,
   'Recall': 0.7502458210422812,
   'F': 0.7618572141787319,
   'TP Rate': 0.7738336713995944,
   'FP Rate': 0.38484848484848483},
  1: {'TP': 362,
   'TN': 807,
   'FP': 132,
   'FN': 239,
   'Precision': 0.6023294509151415,
   'Recall': 0.7327935222672065,
   'F': 0.6611872146118721,
   'TP Rate': 0.6023294509151415,
   'FP Rate': 0.14057507987220447}}}
```



OBSERVATION: Initially used SVC but LinearSVC is a faster implementation of SVC in the case of a linear kernel (which we were following). We were able to achieve almost the same results (1% difference) using the LinearSVC with only 1/60th of the initial time.

OBSERVATION: SVC allows to define whether or not the dataset is balanced using a parameter "class_weight". Using this would mean not having to run SMOTE on the initially imbalanced data. Testing with this modification, we were able to achieve an accuracy of ~72%.

14. Topic Modelling

(R. Chawla, 2017)

Topic modelling is an unsupervised machine learning technique that is used to discover common themes/topics that appear in a collection of documents. All models used to do topic modelling receive the document term matrix as input and must be provided the no. of topics to identify from the set of documents.

The 2 main kinds of models that can be used to perform topic modelling are as listed below.

In [100]:
```
def tfidf(docs, discard_pc):
    ''' Function that creates a document term matrix (ngram = 2) with T
        scores from given documents and discards terms with low TF-IDF
        @param docs: Documents as an iterable.
        @param discard_pc: The percentage of terms to discard.
        @return vectorizer: Vectorizer used to determine TF-IDF scores.
        @return doc_term_df: Document term matrix after feature reducti
    '''
    vectorizer = TfidfVectorizer(ngram_range=(2,2))
    matrix = vectorizer.fit_transform(docs)
    features = vectorizer.get_feature_names_out()
    doc_term_df = pd.DataFrame(data = matrix.toarray(), columns=feature

    # discard terms with lowest TF-IDF scores (feature reduction)
    min_value = matrix.min(axis=0).toarray().ravel()
    sorted_ascending = min_value.argsort()
    lowest_features = features[sorted_ascending]
    print("features with lowest TF-IDF scores = {} ... (length = {})".f
        lowest_features[0:10], len(lowest_features)
    ))
    features_to_drop = lowest_features[:int(len(lowest_features)*discar
    print(f"features to drop = {features_to_drop}")
    doc_term_df = doc_term_df.drop(features_to_drop, axis=1)

    return vectorizer, doc_term_df, matrix
```

14.1. Probabilistic Model

(R. Kulshrestha, 2019)(Analytics Vidhya, 2016)(Wang, 2020)

- Latent Dirichlet Analysis (LDA) is an example of such a model that calculates the probability of words appearing in each topic (`P(word w|topic t)`) and that of topics appearing in each document with the corresponding word in it (`P(topic t|document d)`) to assign topics to words and determine topics in documents.

- It also involves matrix factorization using which it converts a document term matrix to improved lower dimension matrices by making use of sampling.
- LDA assumes that each hidden topic follows a Dirichlet distribution over the vocabulary.
- LDA begins with random probability values which are iteratively updated (`P(word w with topic t) = p(topic t | document d) * p(word w | topic t)`) as words in documents are assigned topics based on probabilities calculated in each iteration, until the algorithm converges.
- A Reverse Engineering approach is thus adopted to try and come up with topics that comprise the documents.

pyLDAvis is an open source library that allows visualization of topics that LDA has attempted to categorize the corpus into. (Sharma, 2021)

In this section, both term frequency (tf) and term frequency–inverse document frequency (tfidf) were used to visualize the topics. For the tf portion, countVectorizer() had been utilized and TfidfVectorizer() has been used for the tfidf portion. Since the model produced using TfidfVectorizer() yielded more relevent results, the results shown are only regarding the same whereas observations have been made regarding the model using just the term frequency.

LDA is a model that tries to predict probability distributions for topics in the corpus and also the words within the topic. Therefore, a TFIDF score is not necessary for an LDA visualization but that being said it can be used to improve existing results. (W, 2017)

Grid search was used to identify the fittest hyperparameters such as the number of topics and the learning decay.

With Term Frequency Inverse Document Frequency Matrix

In [101]:
```python
tfidf_vect, tfidf_df, tfidf_mtx = tfidf(docs=DF["text"], discard_pc=0.2

lda_tfidf = lda(n_components=5, learning_decay= 0.9)
lda_tfidf.fit(tfidf_mtx.toarray())

vis=pyLDAvis.sklearn.prepare(lda_tfidf, tfidf_mtx, tfidf_vect)
pyLDAvis.display(vis)
```

```
features with lowest TF-IDF scores = ['aa gayakhumariyaan' 'place create
'place day' 'place development'
 'place discover' 'place dubai' 'place due' 'place empowerment'
 'place couple' 'place exactly'] ... (length = 41010)
features to drop = ['aa gayakhumariyaan' 'place create' 'place day' ... 'n
ot achieve'
 'not activate' 'not afraid']

C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\utils
eprecation.py:87: FutureWarning: Function get_feature_names is deprecated;
get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please
use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\pyLDAvis\_prepa
re.py:246: FutureWarning: In a future version of pandas all arguments of D
ataFrame.drop except for the argument 'labels' will be keyword-only.
  default_term_info = default_term_info.sort_values(
```

Out[101]:

In [102]:
```python
# lda_tfidf = GSCV(lda(), optim_parameters)
# lda_tfidf.fit(X_train_vec)
```

In [103]:
```python
output = lda_tfidf.transform(tfidf_mtx)
topics = ["Topic " + str(i) for i in range(lda_tfidf.n_components)]
tweets = [tweet for tweet in DF["text"]]
dfidftf = pd.DataFrame(np.round(output, 2), columns=topics, index=tweet
final_topic = np.argmax(dfidftf.values, axis=1)
dfidftf['final topic'] = final_topic
dfidftf=dfidftf.drop(columns=["Topic 0","Topic 1","Topic 2","Topic 3","
```

In [104]:
```python
# topic 0
dfidftf.loc[dfidftf["final topic"] == 0]
```
Out[104]:

	final topic
day	0
dj dimitri vega live performance expo ny celebration watch till end fire work	0
beautifully say ♥a great place learn austria use technology create future opportunity austria pavilion expo dubai	0
right friend want donate put account number poor kind hbl bank account number pkhabb	0
awadh seghayer al ketbi director general dha monaco didier gamerdinger minister health social affair signing agreement promote health	0
...	...
yes lmao	0
young 🇳 chef restaurant michael laurajane esther share like work	0
finally	0
watch ronaldo visit live manchester united star footballer visit sits qanda al wasl plaza give special message fansfull video	0
yes question u gon na get hell man city u play suck never see highlight come u u waste money im city fan btw	0

1318 rows × 1 columns

In [105]:
```python
# topic 1
dfidftf.loc[dfidftf["final topic"] == 1]
```
Out[105]:

	final topic
vertebral deformity measurement mri ct radiography use	1
kenya pavilion expo dubai create platform tourist diaspora come reminisce wonderful time kenya american friend live kenya year come sung kenya national anthem	1
magnificent ♥♥♥♥♥♥ uae popular tourist destination middle east must visit progressive 🙏 need people like you 🙏 india love uaeindia stand uae	1
expo dubai thanks unsung hero vital frontline health worker	1
get experience mobility district creates connection drive world forward break divide physical digital worldstwende dubaiexpo tembea na sisi 👉 📞	1
...	...
hm ambassador highlight uk offer uk reception proud company join u physically virtually	1
congratulation mega event record visit januaryread	1
sami yusuf true lover live expo dubai via month today 😌	1
healthcare become one india large sector term revenue employmentvisit healthcare wellness week th january know	1
yemeni military spokesperson yehya saree expo u might lose urge change destination	1

983 rows × 1 columns

```
In [106]:
# topic 2
dfidftf.loc[dfidftf["final topic"] == 2]
```

Out[106]:

	final topic
celebrate national day slovak republic expo dubai honour welcome excellency zuzana caputova president slovak republic attend ceremony met excellency sarah al amiri	2
chairman abu dhabi executive office praise effort pavilion team showcase culture	2
war yemen become world worsthumanitarian crisis accord un million yemeni child age five currently suffer acute malnutrition expect suffer lifethreatening severe malnutrition come month	2
please advise say true indian pavilion	2
one trample constitution	2
...	...
football goat ronaldo weekend dubai go wild	2
take crowntune january vote best local business watch livestream brand botswana bitc facebook pagesvisit u	2
khyber pakhtunkhwa kpk rapidly grow province pakistan wednesday announce launch country first digital city pakistan digital city accelerate digital transformation promote information technology country	2
im go post expo pic aesthetic bias thread	2
new role client service technicianapply	2

1029 rows × 1 columns

```
In [107]:
# topic 3
dfidftf.loc[dfidftf["final topic"] == 3]
```

Out[107]:

	final topic
yemen start operation pirate coast khalifa burj threaten	3
fcking joke destructive campaign south african must end	3
accelerate towards grand finale expo dubai day away end forever marvel	3
soooo funny	3
cristiano ronaldo accepts globe soccer top scorer time award dubai expo 🏆	3
...	...
philippine pavilion expo dubai highlight culinary experience trade potential middle east	3
partner enterprise estonia host event yesterday insightful discussion around solution egovernance believe citizen key stakeholder	3
expo dubai celebrates australian national day	3
way built city way different way built need ensure building sustainable habiba al marashijoin u virtually	3
global expert gather expo dubai discus future travel urban landscape part singaporeimagine global conversation series	3

1188 rows × 1 columns

In [108]:
```python
# topic 4
dfidftf.loc[dfidftf["final topic"] == 4]
```

Out[108]:

	final topic
🚨 important statement 🇪 yemeni arm force come hour announce detail largescale military operation 🇪	4
hopeless imagine pathetic woman deliver absolutely nothing portfolio president country thankfully anc go collapse battle honest corruptsouth africa survive without	4
new role medical representativeapply	4
sheikh hamdan visit dp world pavilion crown prince praise dubai company enhance uaes global competitiveness	4
thjanuary student across uae select participate expo world majlis dubai nityaansh parekh j represent dp sharjah speaker topic forum school tomorrow	4
...	...
number company withdraw fair serious threat revenge attack late attack	4
criminal pretend minister	4
government sign memorandum understanding mous foreign investment company attracts billion investment dubai international expo january	4
start cancel event precaution expect attack	4
organize national council culture art literature workshop child topic recycle present etidal al miraj	4

1042 rows × 1 columns

OBSERVATION: Topic inferences

All of the topics creates using TF-IDF scores all have the n-grams "expo dubai", "national day", etc so that all of the tweets in that topic are related to the expo itself.

`Topic 1` : Topic one contains ads/offers. It also contains a lot of tweets that talk about national days which might be the reason for a lot of tweets that talk about the events or highlights of specific pavilions.

`Topic 2` : Topic two seemed to mainly talk about immediate healthcare and events that involve gatherings.

`Topic 3` : Topic three contains mainly negative sentiments. A lot of the negative sentiment in this corpus are almost entirely unrelated to the expo.

`Topic 4` : Topic four seems to focus mainly on celebrations or more specifically milestones (eg: awards, hitting a million visits etc). There is heavy discussion on Cristiano Ronaldo's award ceremony in the corpus overall which might be what influenced this. There is also a lot of discussion on money and investment. It is noteworthy that a lot of the tweets about war fall into this topic becasue of the discussion of money in these tweets. Following up on that, the tweets about war could also be the reason as to why military is mentioned a lot in the same topic.

`Topic 5` : A heavy theme in topic 5 is creativity, inspiration and innovation. There are a lot fo tweets about the "food future summit" which might be the cause.

14.2. Linear-Algebraic Model

Non Negative Matrix Factorization (NNMF) is an example of a linear algebraic model. Here, the document matrix model is factorized into 2 matrices (document topic matrix and topic term matrix).

- The aim of NNMF is compute $\boxed{Minimize_{U,V}||D - UV^T||_F^2}$ $[1]$.
- Here, D is the original document term matrix, U is the term topic matrix and V^T is the topic document matrix such that:
  - U and V are mutually orthogonal.
  - All the weights in the matrices U and V are greater than 0.
- By computing $[1]$, one is essentially trying to reduce the distance between the document term matrix (D) and its factorization (UV^T) so that the product of the 2 factors into which the original document term matrix was decomposed into, resembles it.
- NNMF is thus also a means of dimensionality reduction.

NNMF is different from LDA in that it calculates how much each document talks about each topic while LDA determines the different topics that may be contained each document. It has been shown that NMMF whilst also being faster to compute, in some cases where the length of the text is small like when working with social media text data, works better than LDA. (A. Klos, 2020)

OBSERVATION: The following cells shall use an NMF model to extract topics from positive and negative tweets to discover aspects or happenings at Expo 2020 that the public really liked as well as ones they did'nt. This is also a great opportunity to compare performance of the NMMF model as compared to the LDA model on this dataset w.r.t. topic modelling.

In [109]:
```python
def get_topic_terms(topic_term_df, n_terms):
    ''' Function returns top n_words no. of words in each topic.
        @param topic_term_df: The topic term matrix as a pandas datafra
        @param n_terms: The no. of top words to return.
        @return topic_words: A DataFrame with top words per topic.
    '''
    topic_words = {}
    for i in range(topic_term_df.shape[0]):
        topic_words[f"topic{i+1}"] = list(topic_term_df.loc[i,:].sort_v
    return pd.DataFrame(topic_words)
```

In [110]:
```python
def get_doc_topics(doc_topic_df, docs_df):
    ''' Function returns top n_words no. of words in each topic.
        @param doc_topic_df: The document topic matrix as a pandas data
        @param docs_df: The original documents as a matrix.
        @return docs_df: The given docs_df with all the topics appearin
                         most commonly occuring topic per document bein
                         least occuring one being the rightmost column.
    '''
    doc_topics = []
    for i in range(doc_topic_df.shape[0]):
        doc_topics.append(list(doc_topic_df.loc[i,:].sort_values(ascend
    doc_topics_df = pd.DataFrame(doc_topics)
    return pd.concat([docs_df, doc_topics_df], axis=1)
```

In [111]:
```python
def nmmf(docs, n_topics, n_terms):
    ''' Function that perform non negative matrix factorization.
        @param docs: Documents as an iterable.
        @param n_topics: The no. of topics to identify.
        @param n_terms: The no. of top words to return.
        @return topic_term_df: Pandas dataframe with terms per topic.
        @return doc_topic_df: Pandas dataframe with topics per document
    '''
    vectorizer, doc_term_df, tfidf_mat = tfidf(docs=docs, discard_pc=0.
    vocab = list(doc_term_df.columns)

    nmf_model = NMF(n_components=n_topics, random_state=23) # NMMF mode
    nmf_U = nmf_model.fit_transform(doc_term_df.values) # document topi
    nmf_V = nmf_model.components_                       # topic term ma

    topic_term_df = pd.DataFrame(data=nmf_V, columns=vocab)
    doc_topic_df = pd.DataFrame(data=nmf_U, columns=[f"topic{i+1}" for

    topic_terms = pd.DataFrame(get_topic_terms(topic_term_df, n_terms))
    doc_topics = get_doc_topics(doc_topic_df, docs)

    return topic_terms, doc_topics
```

In [112]:
```python
def print_topic_tweets(doc_topics_df, topic, n_tweets):
    ''' Prints tweets of a topic.
        @param doc_topics_df: Document topic matrix as received after N
        @param topic: The topic.
        @param n_tweets: No. of tweets to print.
    '''
    print(TextStyle.PURPLE("some tweets with " + str(topic) + " as its
    for tweet in doc_topics_df[doc_topics_df[0]==topic][:n_tweets]["twe
        print(">", tweet)
```

14.2.1. What was Great at Expo 2020?

In [113]:
```
DF_POS = pd.DataFrame({
    "tweet": TWEETS_DF[TWEETS_DF["Average Sentiment Label"]==1]["Tweet
}).reset_index(drop=True)
print(TextStyle.GREEN("tweets with positive sentiment"))
display(DF_POS.head(3))
```

```
tweets with positive sentiment
```

	tweet
0	celebrate national day slovak republic expo du...
1	beautifully say ❤a great place learn austria u...
2	chairman abu dhabi executive office praise eff...

In [114]:
```
pos_topic_terms_nmmf, pos_doc_topics_nmmf = nmmf(docs=DF_POS["tweet"],
```

```
features with lowest TF-IDF scores = ['aa missile' 'personality belief'
ersonality artistic' 'personal hero'
 'personal expo' 'personal experience' 'person treatment'
 'personality intellectual' 'person something' 'person look'] ... (length
= 19165)
features to drop = ['aa missile' 'personality belief' 'personality artisti
c' ... 'new book'
 'new blog' 'new electronic']

C:\Users\Gayathri Girish Nair\miniconda3\lib\site-packages\sklearn\decom
sition\_nmf.py:289: FutureWarning: The 'init' value, when 'init=None' and
n_components is less than n_samples and n_features, will be changed from '
nndsvd' to 'nndsvda' in 1.1 (renaming of 0.26).
  warnings.warn(
In [115]:  display(pos_topic_terms_nmmf)
```

	topic1	topic2	topic3	topic4
0	top scorer	innovation smart	innovation workshop	expo dubai
1	soccer top	fintech innovation	sap house	national day
2	scorer time	city expo	ism innovation	day celebrate
3	time award	singapore fintech	celebrate sap	celebrate expo
4	globe soccer	smart city	workshop day	day expo
5	award dubai	dubai via	great guest	celebrates national
6	cristiano ronaldo	expo dubai	ariba go	dubai join
7	dubai expo	sami yusuf	house celebrate	armenia national
8	accepts globe	dubai read	dubai sap	khumariyaan expo
9	cristiano ronaldoreceives	yusuf wine	sap ariba	amphitheater love
10	ronaldoreceives globe	wine love	go live	millennium amphitheater
11	grab globe	time spotlight	day great	dubai millennium
12	expo cristiano	dubai middle	guest etisalat	egyptian pavilion
13	expo read	spotlight country	etisalat expo	beautiful egyptian
14	friday afternoon	dubai fintech	expo dubai	global matter
15	expo friday	africa mea	house dubai	highlight expo
16	expo trophy	fintech time	join sap	discus global
17	accepted globe	follow look	february gtm	today business
18	award cristiano	fair expo	dubai heart	business highlight
19	total goal	east africa	heart february	join discus
20	score total	firstever world	unique onsite	future vision
21	goal career	country take	workshop access	celebrate national
22	another award	world fair	gtm register	cambodia celebrates
23	scorer award	take part	access cuttingedge	israel national
24	cristiano ronaldohaspicked	middle east	onsite innovation	enjoy variety
25	accepts soccer	via dont	usecases join	join enjoy
26	expo goat	dont forget	cuttingedge usecases	variety great
27	expo marveltvupdatesread	forget keep	guest expo	activity show
28	give autograph	keep watch	way work	weekend activity
29	selfie fan	true lover	business way	great weekend
30	fan give	yusuf true	dubai envision	dubai celebrates
31	take selfie	visit expo	transform business	happy national
32	award expo	fantastic tour	best join	slovakia celebrates
33	new post	star live	sap transform	dubai celebrate
34	bet via	much siemens	work best	armenian national

	topic1	topic2	topic3	topic4
35	expo bet	iiot heart	improve decisionmaking	day dubai
36	award holiday	smarter safer	become datadriven	celebrate israel
37	smile receives	app make	datadriven simplify	visit expo
38	expo click	dubai infrastructure	move join	dubai january
39	ronaldohaspicked glob	city app	simple move	dubai via
40	holiday dubai	tour mindsphere	sap revolutionize	rwanda celebrates
41	the ronaldo	dubai smarter	data improve	israel celebrates
42	expo the	greener iiot	four simple	welcome excellency
43	expo ronaldo	stand pioneer	decisionmaking become	health wellness
44	career far	infrastructure stand	february register	el salvador
45	trophy goat	safer greener	simplify four	rwanda national
46	win globe	siemens fantastic	dubai february	expo via
47	click link	mindsphere smart	beautiful egyptian	world expo
48	interacts fan	heart expo	egyptian pavilion	dubai today

```
In [116]:  print_topic_tweets(doc_topics_df=pos_doc_topics_nmmf, topic="topic1", n
```

```
some tweets with topic1 as its topic
> lol uhm sure cool let say
> pleased proud see dr ujala nayyar speak pakistan work drive minute
> cristiano ronaldo accepts globe soccer top scorer time award dubai expo
🏆
> attraction key gain visitor enjoyment win field
> call innovator bpo practitioner express interest participate part team u
g e innovation month e dubai expo deadline submission expression interest
th february
> congratulation
> thank sofia bring izel life remember feel inspire virtual crunchyroll ex
po hop mesoamerican story told go show many rich unique story community of
fer 💕
> aoa respect hh small wish brother huge fan wan na meet hh kindly meet us
🇵🇰
> spotify soundcloud promotioni natural sound could spotify music advancem
ent make viral container post promotion lift webbased medium platform shar
e url natural client whats topnotch music advancement site
> cristiano ronaldo smile receives globe soccer top scorer time award holi
day dubai fan turn force catch glimpse man united starvia
```

```
In [117]:   1  print_topic_tweets(doc_topics_df=pos_doc_topics_nmmf, topic="topic2", n
```

```
some tweets with topic2 as its topic
> singapore fintech innovation smart city expo dubai via
> yusuf wine love live dubai via
> honour present associate partner year vong sdg championship modern guruk
ul envisions school move educate career prepare child life
> believe tournament also meant fun part vong sdg championship multitalent
ed yearolds talent show
> incredible masterpiece ♥sami yusuf wine love live dubai via
> pavilion honorable mention award competition best small category country
take top honor deadline loom enter today
> singapore fintech innovation smart city expo dubai via
> singapore fintech innovation smart city expo dubai
> phenomenal captivate exceptionally breathtaking concert sami yusuf beyon
d star live dubai via
> singapore fintech innovation smart city expo dubai
```

```
In [118]:   1  print_topic_tweets(doc_topics_df=pos_doc_topics_nmmf, topic="topic3", n
```

```
some tweets with topic3 as its topic
> discover second day great day investment promotion entrepreneurship duba
i director
> day great guest etisalat expo dubai sap house celebrate sap ariba go liv
e part ism innovation workshop day
> special day rwanda delegation innovator make trip celebrate national her
o day great occasion learn fintech ict innovation
> participate unique onsite innovation workshop access cuttingedge usecase
s join u sap house dubai heart february gtm register
> participate unique onsite innovation workshop access cuttingedge usecase
s join u sap house dubai heart february gtm register
> sap transform business way work best join u sap house dubai envision
> promotes spread choose ad promotes cause people india humanity thru atta
ch surrogate ad
> sap transform business way work best join u sap house dubai envision
> day great guest etisalat expo dubai sap house celebrate sap ariba go liv
e part ism innovation workshop day
> participate unique onsite innovation workshop access cuttingedge usecase
s join u sap house dubai heart february gtm register
```

```
In [119]:  print_topic_tweets(doc_topics_df=pos_doc_topics_nmmf, topic="topic4", n
```

```
some tweets with topic4 as its topic
> celebrate national day slovak republic expo dubai honour welcome excelle
ncy zuzana caputova president slovak republic attend ceremony met excellen
cy sarah al amiri
> beautifully say ❤a great place learn austria use technology create futu
re opportunity austria pavilion expo dubai
> chairman abu dhabi executive office praise effort pavilion team showcase
culture
> please advise say true indian pavilion
> right friend want donate put account number poor kind hbl bank account n
umber pkhabb
> kenya pavilion expo dubai create platform tourist diaspora come reminisc
e wonderful time kenya american friend live kenya year come sung kenya nat
ional anthem
> awadh seghayer al ketbi director general dha monaco didier gamerdinger m
inister health social affair signing agreement promote health
> magnificent ❤❤❤❤❤❤ uae popular tourist destination middle east must
visit progressive 🙏 need people like you🙏india love uaeindia stand uae
> expo dubai thanks unsung hero vital frontline health worker
> sheikh hamdan visit dp world pavilion crown prince praise dubai company
enhance uaes global competitiveness
```

OBSERVATION Topic Inference

4 Topics were identified and 50 words representing each topic were printed. The number 4 was chosen after trying different no. od topics. This is one of the challenges of topic modelling as it cannot be predetermined, what the right no. of topics would be. Having 4 topics leads to easier manual investigation and topics that are human comprehensible.

Upon investigating the terms under each topic and some documents with highest scores under each topic, the following conclusions were made.

`Topic1` : This topic seems to be related to winning and is heavily influenced by the event where the football/soccer legend Christiano Ronaldo was awarded Globe Soccer's Top Scorer of All Time Award at Dubai Expo 2020 on a friday. There is emphasis on celebrity visits to expo. It is also likely that tweets with promotional content like ones that promise readers prizes upon clicking links etc, or advertisements fall under this topic.

`Topic2` : This topic seems to be about exhibition of innovative architecture/infrastructure as well as performances and showcasing of talent. People really seem to have enjoyed concerts ("For True Lovers", "Beyond The Stars" "The Wine Of Love"), by famous British singer, songwriter, multi-instrumentalist and composer, Sami Yusuf which took place at the expo.

`Topic3` : This topic seems to be related to innovation, technology, business and their future. This is highlighted by the fact that the company "SAP" which is the Innovative Enterprise Software Partner of Expo 2020 Dubai shows up often in terms related to this topic in addition to terms like "business", "data driven", "cutting edge" and so on.

`Topic4` : This topic hints at being about wellness/health, the spirit of celebration and a display of pride and national sentiment.

14.2.2. What was Not so Great at Expo 2020?

In [120]:
```python
DF_NEG = pd.DataFrame({
    "tweet": TWEETS_DF[TWEETS_DF["Average Sentiment Label"]==-1]["Tweet
}).reset_index(drop=True)
print(TextStyle.RED("tweets with negative sentiment"))
display(DF_NEG.head(3))
```

```
tweets with negative sentiment
```

	tweet
0	hopeless imagine pathetic woman deliver absolu...
1	war yemen become world worsthumanitarian crisi...
2	much pay tweet crap

In [121]:
```python
neg_topic_terms_nmmf, neg_doc_topics_nmmf = nmmf(docs=DF_NEG["tweet"],
```

```
features with lowest TF-IDF scores = ['abduction force' 'president count
' 'presidential venue'
 'presidential visit' 'presidentso think' 'pressure aggressor'
 'pretend minister' 'prey different' 'primate fight' 'present disaster']
... (length = 2866)
features to drop = ['abduction force' 'president country' 'presidential ve
nue'
 'presidential visit' 'presidentso think' 'pressure aggressor'
 'pretend minister' 'prey different' 'primate fight' 'present disaster'
 'prime minister' 'princess bully' 'prisoner year' 'probably bullshit'
 'probably cry' 'product anc' 'prof cunningham' 'project people'
 'promote nonsense' 'primitive primate' 'propaganda stop'
 'preexist already' 'precaution expect' 'post nonsensical' 'post pic'
 'post publish' 'postpone certain' 'postpone dec' 'postpone due'
 'postpone event' 'postpone new' 'predator prey' 'postpone visit'
 'postponement event' 'postpones dubai' 'potato heal'
 'powerful commemorate' 'ppl wore' 'pr exercise' 'practice today'
 'pray gulf' 'postpone year' 'proper sa' 'properly lose' 'proposes change'
 'radebe ibrahim' 'raise flag' 'raise fund' 'rampant racial'
```

```
In [122]:  display(neg_topic_terms_nmmf)
```

	topic1	topic2	topic3	topic4
0	zone fatal	war yemen	kill innocent	not safe
1	soon may	dubai longer	innocent civilian	yemeni army
2	conflict zone	threat retaliation	uae blood	expo dubai
3	war soon	cancel trip	civilian uae	sure loss
4	wish leave	trip due	affair supply	warning serious
5	investment uae	imminent threat	foreign affair	dubai sure
6	dont waste	due imminent	uae minister	serious everyone
7	fatal war	emirate war	drone kill	everyone stay
8	leave save	due emirate	blood hand	away expo
9	life please	longer safe	supply drone	drone yemeni
10	waste money	dubai expo	terrorist group	come warning
11	anymore conflict	isnt brought	tplf kill	army come
12	not anymore	threaten strike	group earth	force drone
13	bla bla	houthis threaten	civilian whole	stay away
14	die palestine	strike dubai	earth equivalence	uae not
15	palestine please	yemen isnt	whole world	threat target
16	leave die	expo war	equivalence tplf	army threat
17	world conflict	brought endread	carry water	dubai start
18	city world	brought end	bottle coffee	cancellation yemeni
19	not safest	announce washington	fighter carry	start see
20	bla conflict	yemen siege	weapon lady	help matter
21	yemeni citizen	know next	lady tdf	uae whats
22	not rebles	yemen military	killedget lose	serious hardship
23	not not	uae saudia	coffee cup	essence visit
24	citizen conflict	end uae	im sure	passport still
25	youve waste	expodubai save	look innocent	hardship passport
26	sorry youve	visit end	civilian killedget	undergo serious
27	covid kill	important postpone	tdf fighter	whats essence
28	im sorry	visitor expo	wicked take	government nigerian
29	kill soz	target yemen	take photo	nigerian uae
30	sponsor covid	force expodubai	switch weapon	visit expo
31	throw im	dubai must	sure switch	nigerian embassy
32	im city	arabia coalition	shoot im	nigerian government
33	hell man	coalition war	cup look	dubai nigerian
34	fan btw	life important	water bottle	still prideful
35	get hell	next target	come attack	embassy uae

	topic1	topic2	topic3	topic4
36	come waste	siege not	careful dont	nigerian undergo
37	suck never	not stop	dont come	not help
38	city fan	siege announce	attack war	dubai expo
39	city play	unless announce	war uae	jubilee stage
40	gon na	aggressive war	uae yemen	stage expo
41	highlight come	short aggressive	visitor due	unforeseen circumstance
42	never see	aggression declaration	inform visitor	inform visitor
43	yes question	declaration war	unforeseen circumstance	visitor due
44	show conversion	stop unless	jubilee stage	january jubilee
45	fluff pr	nutshell aggression	due unforeseen	concert originally
46	show showboating	washington well	concert originally	due unforeseen
47	showboating elevate	washington short	january jubilee	war yemen

```
In [123]:  print_topic_tweets(doc_topics_df=neg_doc_topics_nmmf, topic="topic1", n
```

```
some tweets with topic1 as its topic
> hopeless imagine pathetic woman deliver absolutely nothing portfolio pre
sident country thankfully anc go collapse battle honest corruptsouth afric
a survive without
> mother nature deeply depressed global collective action option available
humanity survive devastate effect climate change world confront today
> conflict zone fatal war soon may wish leave save life please dont waste
money investment uae
> fuck go post pic grealish pl trophie first year fuck stupid
> due threat possible imminent yemeni attack expo decide postpone certain
activity another date determine later
> waste data post nonsensical find another way appease madam overinflated
ego shes useless
> paso bloquear esta cuenta publicitaria como protesta la censura de twitt
er invito mi seguidores hacer lo mismo im go block ad account protest cens
orship twitter invite follower
> wtf lie barbie busy anc squabble
> conflict zone fatal war soon may wish leave save life please dont waste
money investment uae
> conflict zone fatal war soon may wish leave save life please dont waste
money investment uae
```

```
In [124]:  print_topic_tweets(doc_topics_df=neg_doc_topics_nmmf, topic="topic2", n
```

```
some tweets with topic2 as its topic
> war yemen become world worsthumanitarian crisis accord un million yemeni
child age five currently suffer acute malnutrition expect suffer lifethrea
tening severe malnutrition come month
> much pay tweet crap
> hey yesterday flight jeddah abu dhabi say website im eligible expo ticke
t everytime fill info say something go wrong try
> visitor expo dubai must know next target yemen military missile force ex
podubai save life important postpone visit end uae saudia arabia coalition
war yemen
> get copy deposition researcher lady complain see modi everywhere indian
pavilion
> lie pure evil lock month care home die alone without family save
> there medium blackout uae ia prosecute user share tbese video obviously
unsafe uae yemen threaten bomb target burj khalifa drone
> pray gulf safetygod punish yemeni terrorist houthis want attack innocent
people living area affect country economyi heard threaten target dubai exp
o hope nothing bad happens someone need stop terrorist
> israel isaac herzog speaks dubai expo first presidential visit uae hour
gulf state say intercept ballistic missile fire houthis
> yemen threaten bomb dubai
```

```
In [125]:  print_topic_tweets(doc_topics_df=neg_doc_topics_nmmf, topic="topic3", n
```

```
some tweets with topic3 as its topic
> people really wicked take photo shoot im sure switch weapon lady tdf fig
hter carry water bottle coffee cup look innocent civilian killedget lose
> stupid prick
> jfc people pathetic
> terrorism allow international law
> bad half class amongst
> still metro train
> yall bunch crook dishonest criminal complicit brand
> nothing love el salvador
> yes everybody know youre afraid yemeni strike idiotic show
> husband liar ive expo time visit indian pavillion twice nothing like hus
band say teach truthful not u family else imagine child like parent grow
```

```
In [126]:  print_topic_tweets(doc_topics_df=neg_doc_topics_nmmf, topic="topic4", n
```

```
some tweets with topic4 as its topic
> someone pathetically stupid
> siege kill civilian destroy civilian facility abdulsalam write post publ
ish twitter pagehe add security stability everyone none
> kenya end dubai travel ban opening door vaccinate tourist
> steal role
> hate mielie rib give pap vleis chakalaka
> uae not safe attack
> luxembourg pavilion present disaster rapidresponse kit expo dubai
> sterling shit last year cry grealish dont best player world fuck urself
> first expo one first booth set cancel remember walk empty hall wonder he
ll happen world
> regret inform visitor due unforeseen circumstance cairokee concert origi
nally schedule january jubilee stage expo dubai postpone new date announce
soon across social medium channel
```

OBSERVATION: As far as the negative topics go, there is not much human observable segregation between the topics. This may be due to there being much fewer negative tweets compared to positive or neutral ones. The topics all point at "aggression", "war" etc and are not directly connected to the Expo. Topics 3 and 4 are the only ones that seem to be a little related to the Expo since there is some mention of events that were delayed which may have caused inconvenience.

OBSERVATION: Overall, it can be inferred that Expo 2020 held in Dubai has been a fun, engaging, inspiring or enlightening experience for the majority of people who visited/participated.

15. Word Clouds

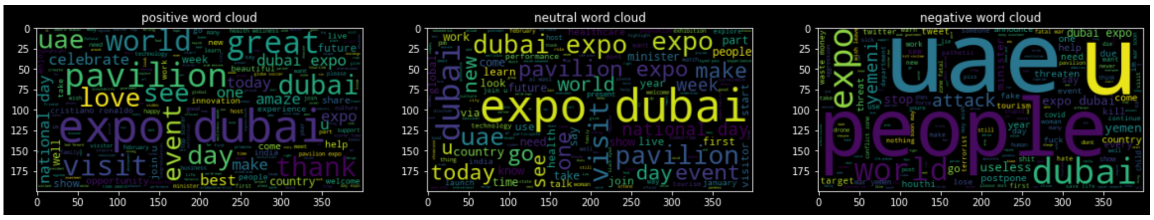
Another way to see what the general themes of conversation were w.r.t. the positive, neutral and negative tweets separately is to generate 3 separate world clouds using only positive, neutral or negative tweets respectively.

Word clouds are visualizations of a corpus or a selected group in a corpus that will help to better understand the content of it. As the volume of unstructured data in the form of text grows at an unprecedented rate, particularly in the sphere of social media, there is a growing need to evaluate the huge amounts of text generated by these systems. (Super User, 2020).

This section intends to display the most used words per sentiment. The way that this is done is by making one massive sentence per sentiment (combine all the sentences of that sentiment) and then generate it using WordCloud.

In [127]:
```
plt.figure(figsize=(20,5))

plt.subplot(131)
plt.title("positive word cloud")
plt.imshow(WORD_CLOUD.generate(" ".join(DF[DF["sentiment"]==1]["text"])

plt.subplot(132)
plt.title("neutral word cloud")
plt.imshow(WORD_CLOUD.generate(" ".join(DF[DF["sentiment"]==0]["text"])

plt.subplot(133)
plt.title("negative word cloud")
plt.imshow(WORD_CLOUD.generate(" ".join(DF[DF["sentiment"]==-1]["text"]
```

Out[127]:
```
<matplotlib.image.AxesImage at 0x168332af3a0>
```



OBSERVATION: The "u" in the word clouds are a result of lemmatization mapping the word "us" to "u". This is because NLTK WordNet Lemmatizer has in its mappings, "us" mapped to "u" (pgmank, 2019). Other than this, the words w.r.t. positive, negative and neutral sentiment tweets are content appropriate.

- It can be noted that positive adjectives like "love", "great", "amaze", "thank" all appear with aspects of the Expo 2020 like "pavilion", "event", "expo" indicating that the Expo2020 was indeed a positive experience.
- Words like "go", "visit", "expo", "dubai", that are not adjectives indicating experiences in general rightly appear most in the neutral word cloud.
- Since most of the negative tweets were about other events like the attack on UAE that occurred around the time of the tweet collection, words like "people", "attack", "uae" appear most in the negative world cloud once again indicating that there was not much negative about the Expo itself. The only negative experience w.r.t. expo as observed before from topic modelling is that some event(s) seem to have been postponed on account of which the word "postpone" has appeared in the negative word cloud.

16. Conclusion

The data collection (Part A) was done using the Twitter API and python’s Tweepy library. The data structure set(), was used for storing the tweets to ensure unique tweets were collected. Additionally, all retweets extracted by the API were filtered out. This was done as retweets increase the possibility of having duplicate tweets in the dataset and using a set does not fix the problem of the presence of retweets. In addition to the keyword “Expo”, several other keywords such as “Golden Jubilee”, “Pavilion”, “Forsan” were used for extracting the tweets, to ensure a wide and diverse collection of tweets were extracted. After the collection of tweets was complete, the tweets were saved in a pandas DataFrame, which was then converted to a CSV file. The CSV file generated serves as the primary dataset for this coursework.

To label the data (Part B) textBlob, Vader, and a dedicated separate function for calculating a score for tweets that contain emojis was tested. After further analysis, it was concluded that an average of the scores provided by Vader and the defined emoji tweet calculator (in case there are emojis in the tweet) gave the best results.

Data Labelling (Part B) was then followed by part C which began with text cleaning.

It was observed that the raw tweets contained lots of special characters like # and @ and others due to hashtags and mentions that are often part of tweets. Content like URLs also do not contribute to the sentiment of the text and were removed. The decision was made to retain EMOJIs in the text as they represent sentiment. After cleaning, the text still contained contractions from standard english as well as from type language (u => you, r => are, etc) which were expanded to full words.

The cleaning of text was followed by tokenization, stemming and lemmatization. However, it was found that lemmatization produced words meaningful words. Since the plan was to embed the documents using a GloVe embedding model which works best with existing meaningful words, the decision was made to opt for lemmatization as opposed to stemming.

The final preprocessing step involves identifying and removing tweets that are not mostly english. All the preprocessing steps were encapsulated in a class that extends the "BaseEstimator", "TransformerMixin" class by scikit-learn so that it can be used with scikit-learn Pipeline.

The next step was to split the data into Train Test sets and it was observed that the data was very unevenly distributed. Unevenly distributed data can lead to ML algorithms being unable to learn to classify some classes due to it not being enough examples of that class. To avoid this issue, the oversampling technique, SMOTE was used.

The next step involved extracting features/document embedding. 2 Different approaches (Bag of Words) and (GloVe Word Embedding) were explored. The overall observation was as expected. All the classifiers performed better with GloVe embeddings which capture more context than BOW embeddings.

Before experiments could be conducted with classifiers, a evaluation function had to be written that would calculate all the performance metrics w.r.t. each classifier and display the ROC-AUC curve. Performance metrics calculated based on which classifier performance was judges were TP, TN, FP, FN, Precision, Recall, F-Measure, TP-Rate, FP-Rate, AUC of ROC curve.

The data (both TFIDF and Glove versions) was trained on 5 classifiers to check for the best results. While using the Naive Bayes Classifier, for both the verisons, best results were observed in Bernoulli Naive Bayes. While the Glove embeddings achieved an accuracy of ~58%, the BOW embeddings were able to top that by providing an accuracy of ~61%.

As for the K nearest neighbor classifier, the optimal k was selected by plotting the silhouette score against the number of neighbors(k) and the k was chosen accordingly. After using the most optimal k and a few other optimal parameters found through grid search, the BOW embeddings achieved an accuracy of ~60%, the Glove embeddings were able to top that by providing an accuracy of ~66%.

As for the Logistic Regression classifier, the BOW embeddings achieved a low accuracy of ~30%, whereas, the Glove embeddings outperformed it by scoring ~80%. Using Grid search to find the best parameters caused a very minimal increase in training accuracy of ~3% in the case of the model fitted with BOW data, but did not increase the generalizations of the models on test data.

As for the SVM classifier, LinearSVC() was used as opposed to the complete SVC model as LinearSVC is a more lightwight and faster version for linear kernels. The BOW embeddings achieved an accuracy of ~30%, the Glove embeddings were able to top that by providing an accuracy of ~70%.

With the Neural Network as well, test accuracy with GloVe model was much higher at ~72% while with BOW model, it was only around ~35% as expected since GloVe embeddings capture more context/meaning of words than BOW embeddings. Little overfitting was observed despite Dropout. It was concluded after several experiments with the architecture of the ANN that this was simply due to the fact that sentiment analysis is a very challenging task for ML models.

For the visualization, word clouds, pyLDAvis and NMF were used. The prominent words through the positive, negative and neutral sentiments were visualized using the WordCloud library. In the NMF portion, the main positive and negative tweets were used to model topics per sentiment. Although some of the negative topics formed were irrelevant to the corpus, some useful topics were generated from the positive ones. As for the LDA portion, both TFIDF() and CountVectorizer() were tested to model the topics. The former produced topics that were more specific and relevant to the corpus. It was observed that NMF seemed to produce topics that were slightly better; more segregated.

The key observation made after detailed analysis of Tweets related to Expo 2020 Dubai categorized by sentiment, was that for majority of its participants and visitors, Expo was a fun/inspiring positive experience.

Analysis of sentiment associated with social media discussions about events like performed as part of this project can provide insights about how the event is received by the general public and what can be done to improve the experience. For example, it was observed that the only thing that was negative w.r.t. the Expo was inconvenience caused due to postponing of some events on short notice. This learning could be leveraged by the organizers of Expo to strive for further improvement of event management.

Overall, this project was an enlightening one with all group members being able to gain real world data analysis and visualization skills to application of all taught theoretical material.

Detailed explanations of concepts explored and justification of decisions made are provided under the respective sections of this notebook. Observations made have also been highlighted throughout this notebook.

References

A

- Alex Klos. (2020) *Topic modeling: LDA vs. NMF for newbies*. Link: https://alexklos.ca/blog/natural-language-processing-lda-vs-nmf-for-newbies/ (https://alexklos.ca/blog/natural-language-processing-lda-vs-nmf-for-newbies/). [Last Accessed: 25/02/2022]

- Abishek PSS. (2021) *Sentiment Analysis using LSTM - MLearning.ai*, Medium. Link: https://medium.com/mlearning-ai/sentiment-analysis-using-lstm-21767a130857 (https://medium.com/mlearning-ai/sentiment-analysis-using-lstm-21767a130857) [Last Accessed 08 Mar 2022]
- Analytics Vidhya. (2016). Beginners Guide to Topic Modeling in Python and Feature Selection. [online] Available at: https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/ (https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/) [Last Accessed 25 Feb. 2022].
- Analytics Vidhya. (2017). Learn Naive Bayes Algorithm | Naive Bayes Classifier Examples. [online] Available at: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/ (https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/) [Accessed 7 Mar. 2022].
- Analytics Vidhya. (2018). K Nearest Neighbor | KNN Algorithm | KNN in Python & R. [online] Available at: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/ (https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/) [Accessed 7 Mar. 2022].

B

- Band, A. (2020). How to find the optimal value of K in KNN? - Towards Data Science. [online] Medium. Available at: https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb (https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb) [Last Accessed 14 Feb. 2022].
- Bengfort, B., Bilbro, R. and Ojeda, T. (2018). *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning.* Sebastopol, Ca: O'reilly Media, Inc.
- Bernardo, I. (2021). *Stemming Text with NLTK - Towards Data Science.* Medium. Link: https://towardsdatascience.com/stemming-corpus-with-nltk-7a6a6d02d3e5 (https://towardsdatascience.com/stemming-corpus-with-nltk-7a6a6d02d3e5) [Last Accessed 08 Feb 2022]
- Brownlee, J. (2021). *Multinomial Logistic Regression with Python.* Machine Learning Mastery. Link: https://machinelearningmastery.com/multinomial-logistic-regression-with-python/ (https://machinelearningmastery.com/multinomial-logistic-regression-with-python/) [Last Accessed 01 Mar 2022]

C

- codebasics. (2021) *What is Word2vec? A Simple Explanation* | *Deep Learning Tutorial 41 (Tensorflow, Keras & Python)*, YouTube Video. Link: https://www.youtube.com/watch?v=hQwFeIupNP0 (https://www.youtube.com/watch?v=hQwFeIupNP0). [Last Accessed 08 Feb 2022]
- Coderzcolumn.com. (2020). Scikit-Learn - Naive Bayes by Sunny Solanki. [online] Available at: https://coderzcolumn.com/tutorials/machine-learning/scikit-learn-sklearn-naive-bayes (https://coderzcolumn.com/tutorials/machine-learning/scikit-learn-sklearn-naive-bayes) [Accessed 7 Mar. 2022].
- Chakravarthy, S. (2020) *Tokenization for Natural Language Processing*, Towards Data Science. Link: https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4 (https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4) [Last Accessed 08 Feb 2022]
- Chen, Y. (2018). How to generate an LDA Topic Model for Text Analysis. [online]

Medium. Available at: https://yanlinc.medium.com/how-to-build-a-lda-topic-model-using-from-text-601cdcbfd3a6 (https://yanlinc.medium.com/how-to-build-a-lda-topic-model-using-from-text-601cdcbfd3a6) [Last Accessed 1 Mar. 2022].

- Coderzcolumn.com. (2020). Scikit-Learn - Naive Bayes by Sunny Solanki. [online] Available at: https://coderzcolumn.com/tutorials/machine-learning/scikit-learn-sklearn-naive-bayes (https://coderzcolumn.com/tutorials/machine-learning/scikit-learn-sklearn-naive-bayes) [Accessed 6 Mar. 2022].

D

- Diego Lopez Yse (2021) *Text Normalization for Natural Language Processing (NLP)*, Medium. Link: https://towardsdatascience.com/text-normalization-for-natural-language-processing-nlp-70a314bfa646 (https://towardsdatascience.com/text-normalization-for-natural-language-processing-nlp-70a314bfa646) [Last Accessed 06 Feb 2022]

E

- Eiki. (2019) *Feature Extraction in Natural Language Processing with Python*, Medium.com. Link: https://medium.com/@eiki1212/feature-extraction-in-natural-language-processing-with-python-59c7cdcaf064 (https://medium.com/@eiki1212/feature-extraction-in-natural-language-processing-with-python-59c7cdcaf064). [Last Accessed 07 Feb 2022]
- Elia, F. (2020). *Stemming vs Lemmatization* | *Baeldung on Computer Science.* Baeldung on Computer Science. Link: https://www.baeldung.com/cs/stemming-vs-lemmatization (https://www.baeldung.com/cs/stemming-vs-lemmatization) [Last Accessed 08 Feb 2022]

F

- Fasttext. (n.d.). *Language Identification* Link: https://fasttext.cc/docs/en/language-identification.html (https://fasttext.cc/docs/en/language-identification.html) [Last Accessed 20 Feb 2022]

G

- GeeksforGeeks. (2017) *Removing stop words with NLTK in Python*, GeeksforGeeks. Link: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/ (https://www.geeksforgeeks.org/removing-stop-words-nltk-python/) [Last Accessed 06 Feb 2022]
- Great Learning Team. (2020) *An Introduction to Bag of Words (BoW)* | *What is Bag of Words?*, mygreatlearning.com. Link: https://www.mygreatlearning.com/blog/bag-of-words/ (https://www.mygreatlearning.com/blog/bag-of-words/). [Last Accessed 07 Feb 2022]
- Gupta, M. (2020). *Using WordNetLemmatizer.lemmatize() with pos_tags Throws KeyError.* [online] Stack Overflow. Link: https://stackoverflow.com/questions/61982023/using-wordnetlemmatizer-lemmatize-with-pos-tags-throws-keyerror (https://stackoverflow.com/questions/61982023/using-wordnetlemmatizer-lemmatize-with-pos-tags-throws-keyerror) [Last Accessed 08 Feb 2022]

H

- Harrison, O. (2018). Machine Learning Basics with the K-Nearest Neighbors Algorithm. [online] Medium. Available at: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761 (https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761) [Accessed 7 Mar. 2022].
- Himanshu Chandra. (2020) *Pipelines & Custom Transformers in scikit-learn: The step-by-step guide (with Python code)*, towards data science. Link: https://towardsdatascience.com/pipelines-custom-transformers-in-scikit-learn-the-step-by-step-guide-with-python-code-4a7d9b068156 (https://towardsdatascience.com/pipelines-custom-transformers-in-scikit-learn-the-step-by-step-guide-with-python-code-4a7d9b068156). [Last Accessed: 17/02/2022]

I

- IBM Cloud Education. (2017) *Neural Networks*, ibm.com. Link: https://www.ibm.com/cloud/learn/neural-networks (https://www.ibm.com/cloud/learn/neural-networks). [Last Accessed 07 March 2022]
- Igor Bobriakov. (2018) *Comparison of Top 6 Python NLP Libraries*, Medium. Link: https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb (https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb) [Last Accessed 08 Feb 2022]

J

- Jabeen, H. (2018). *Stemming and Lemmatization in Python.* DataCamp Community. Link: https://www.datacamp.com/community/tutorials/stemming-lemmatization-python (https://www.datacamp.com/community/tutorials/stemming-lemmatization-python) [Last Accessed 07 Feb 2022]
- Jason Brownlee. (09 Oct 2017) *A Gentle Introduction to the Bag-of-Words Model*, machinelearningmastery.com. Link: https://machinelearningmastery.com/gentle-introduction-bag-words-model/ (https://machinelearningmastery.com/gentle-introduction-bag-words-model/). [Last Accessed 07 Feb 2022]
- Jason Brownlee. (11 Oct 2017) *What Are Word Embeddings for Text?*, machinelearningmastery.com. Link: https://machinelearningmastery.com/what-are-word-embeddings/ (https://machinelearningmastery.com/what-are-word-embeddings/). [Last Accessed 07 Feb 2022]
- JCharisTech (2021). *7 NLP Tools For Language Detection In Python.* YouTube Video. Link: https://www.youtube.com/watch?v=JJdJePbmCyw (https://www.youtube.com/watch?v=JJdJePbmCyw) [Last Accessed 20 Feb 2022]
- Joseph, R. (2018). Grid Search for model tuning - Towards Data Science. [online] Medium. Available at: https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e (https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e) [Accessed 7 Mar. 2022].

K

- Karra, J. (2021) *Sentiment Analysis using LSTM - Jagath Karra*, Medium. Link: https://jagathprasad0.medium.com/sentiment-analysis-using-lstm-b3efee46c956 (https://jagathprasad0.medium.com/sentiment-analysis-using-lstm-b3efee46c956) [Last Accessed 08 Mar 2022]
- KD Nuggets (2018). *Text Wrangling & Pre-processing: A Practitioner's Guide to NLP.* KDnuggets. Link: https://www.kdnuggets.com/2018/08/practitioners-guide-processing-understanding-text-2.html (https://www.kdnuggets.com/2018/08/practitioners-guide-processing-understanding-text-2.html) [Last Accessed 07 Feb 2022]
- Khanna, C. (2021) *Text preprocessing: Stop words removal*, Towards Data Science. Link: https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a (https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a) [Last Accessed 09 Feb 2022]
- Kilian Thiel. (2016) *Sentiment Analysis with N-grams*, Knime. Link: https://www.knime.com/blog/sentiment-analysis-with-n-grams (https://www.knime.com/blog/sentiment-analysis-with-n-grams). [Last Accessed: 09/02/2022]
- Kim, R. (2018) *Yet Another Twitter Sentiment Analysis Part 1 — tackling class imbalance*, Medium. Link: https://towardsdatascience.com/yet-another-twitter-sentiment-analysis-part-1-tackling-class-imbalance-4d7a7f717d44 (https://towardsdatascience.com/yet-another-twitter-sentiment-analysis-part-1-tackling-class-imbalance-4d7a7f717d44) [Last Accessed 08 Mar 2022]

L

- Lukei (2019). *Dealing With Contractions in NLP,* Medium. Link: https://medium.com/@lukei_3514/dealing-with-contractions-in-nlp-d6174300876b (https://medium.com/@lukei_3514/dealing-with-contractions-in-nlp-d6174300876b) [Last Accessed 03 Mar 2022]

M

- Mauro Di Pietro. (2017) *Text Analysis & Feature Engineering with NLP*, towards data science. Link: https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d (https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d). [Last Accessed 06 Feb 2022]
- Menzli, A. (2021) *Tokenization in NLP: Types, Challenges, Examples, Tools*, neptune.ai. Link: https://neptune.ai/blog/tokenization-in-nlp (https://neptune.ai/blog/tokenization-in-nlp) [Last Accessed 08 Feb 2022]
- Malik, F. (2020). What Is Grid Search? - FinTechExplained - Medium. [online] Medium. Available at: https://medium.com/fintechexplained/what-is-grid-search-c01fe886ef0a (https://medium.com/fintechexplained/what-is-grid-search-c01fe886ef0a) [Accessed 7 Mar. 2022].

N

- Normalized Nerd. (2020) *Introduction to NLP | GloVe Model Explained*, YouTube Video. Link: https://www.youtube.com/watch?v=Fn_U2OG1uqI (https://www.youtube.com/watch?v=Fn_U2OG1uqI). [Last Accessed 08 Feb 2022]
- NLTK (2019). *5. Categorizing and Tagging Words. NLTK.* Link: https://www.nltk.org/book/ch05.html (https://www.nltk.org/book/ch05.html) [Last Accessed 09 Feb 2022]

P

- Patel, S. (2020). *NLP: POS (Part of speech) Tagging & Chunking.* Medium. Link: https://suneelpatel18.medium.com/nlp-pos-part-of-speech-tagging-chunking-f72178cc7385 (https://suneelpatel18.medium.com/nlp-pos-part-of-speech-tagging-chunking-f72178cc7385) [Accessed 09 Feb 2022]
- pgmank (2019). NLTK WordNetLemmatizer processes “US” as “u.” [online] Stack Overflow. Available at: https://stackoverflow.com/questions/54784287/nltk-wordnetlemmatizer-processes-us-as-u (https://stackoverflow.com/questions/54784287/nltk-wordnetlemmatizer-processes-us-as-u) [Accessed 7 Mar. 2022].
- Pranali Borele, Dilipkumar A. Borikar. (2016) *An Approach to Sentiment Analysis using Artificial Neural Network with Comparative Analysis of Different Techniques*, TABLE1. Link: https://www.iosrjournals.org/iosr-jce/papers/Vol18-issue2/Version-5/K1802056469.pdf (https://www.iosrjournals.org/iosr-jce/papers/Vol18-issue2/Version-5/K1802056469.pdf). [Last Accessed 14 Mar 2022]
- Praveen Sujanmulk. (2021) *Sentiment Analysis on Amazon Reviews using TF-IDF Approach, Analytics Vidhya*. Link: https://www.youtube.com/watch?v=Fn_U2OG1uqI (https://www.youtube.com/watch?v=Fn_U2OG1uqI). [Last Accessed 10 Feb 2022]
- Princeton University (n.d.). *wngloss(7WN)* | *WordNet.* Princeton. Link: https://wordnet.princeton.edu/documentation/wngloss7wn (https://wordnet.princeton.edu/documentation/wngloss7wn) [Accessed 04 Mar. 2022]
- Python (2020). DataSklr. [online] DataSklr. Available at: https://www.datasklr.com/select-classification-methods/k-nearest-neighbors (https://www.datasklr.com/select-classification-methods/k-nearest-neighbors) [Accessed 7 Mar. 2022].

Q

- Quora. (2019) *What is a difference between adding one more layer and increasing neurons in one layer?. Link: https://www.quora.com/What-is-a-difference-between-adding-one-more-layer-and-increasing-neurons-in-one-layer (https://www.quora.com/What-is-a-difference-between-adding-one-more-layer-and-increasing-neurons-in-one-layer). [Last Accessed 14 Mar 2022]*

R

- Radimrehurek. (2021). *Gensim: Topic Modelling for Humans.* Link: https://radimrehurek.com/gensim/auto_examples/tutorials/run_wmd.html (https://radimrehurek.com/gensim/auto_examples/tutorials/run_wmd.html) [Last Accessed: 03 Mar 2022]
- Raschka, S. (2016). *Regularization in Logistic Regression: Better Fit and Better Generalization?.* KDnuggets. Link: https://www.kdnuggets.com/2016/06/regularization-logistic-regression.html (https://www.kdnuggets.com/2016/06/regularization-logistic-regression.html) [Last Accessed: 03 Mar 2022]
- Ravish Chawla. (2017) *Topic Modeling with LDA and NMF on the ABC News Headlines dataset*, medium. Link: https://medium.com/ml2vec/topic-modeling-is-an-unsupervised-learning-approach-to-clustering-documents-to-discover-topics-fdfbf30e27df (https://medium.com/ml2vec/topic-modeling-is-an-unsupervised-learning-approach-to-clustering-documents-to-discover-topics-fdfbf30e27df). [Last Accessed: 25/02/2022]
- Ria Kulshrestha. (2019) *A Beginner’s Guide to Latent Dirichlet Allocation(LDA)*, towards

data science. Link: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2 (https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2). [Last Accessed: 25/02/2022]

S

- Sciforce. (2018) *Word Vectors in Natural Language Processing: Global Vectors (GloVe)*, KDnuggets. Link: https://www.kdnuggets.com/2018/08/word-vectors-nlp-glove.html (https://www.kdnuggets.com/2018/08/word-vectors-nlp-glove.html). [Last Accessed 08 Feb 2022]
- scikit-learn. (2021a) *Receiver Operating Characteristic (ROC)*. [online] Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#plot-roc-curves-for-the-multilabel-problem (https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#plot-roc-curves-for-the-multilabel-problem) [Last Accessed 17 Feb. 2022].
- scikit-learn. (2021b). *sklearn.linear_model.LogisticRegression.* [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) [Accessed 25 Feb 2022].
- scikit-learn. (2021c). *sklearn.model_selection.GridSearchCV.* [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) [Accessed 28 Feb 2022].

- scikit-learn. (2021). 1.4. Support Vector Machines. [online] Available at: https://scikit-learn.org/stable/modules/svm.html#:~:text=Support%20vector%20machines%20(SVMs)%20are,than%20the%20r (https://scikit-learn.org/stable/modules/svm.html#:~:text=Support%20vector%20machines%20(SVMs)%20are,than%20the%20r [Accessed 7 Mar. 2022].
- scikit-learn. (2021). sklearn.model_selection.GridSearchCV. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) [Accessed 7 Mar. 2022].
- scikit-learn. (2012). 3.2. Tuning the hyper-parameters of an estimator. [online] Available at: https://scikit-learn.org/stable/modules/grid_search.html#exhaustive-grid-search (https://scikit-learn.org/stable/modules/grid_search.html#exhaustive-grid-search) [Accessed 7 Mar. 2022].
- scikit-learn. (2021). 1.9. Naive Bayes. [online] Available at: https://scikit-learn.org/stable/modules/naive_bayes.html (https://scikit-learn.org/stable/modules/naive_bayes.html) [Accessed 7 Mar. 2022].
- scikit-learn. (2021). Receiver Operating Characteristic (ROC). [online] Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#plot-roc-curves-for-the-multilabel-problem (https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#plot-roc-curves-for-the-multilabel-problem) [Last Accessed 17 Feb. 2022].
- Selvaraj, N. (2020). A Beginner's Guide to Sentiment Analysis with Python. [online] Medium. Available at: https://towardsdatascience.com/a-beginners-guide-to-sentiment-analysis-in-python-95e354ea84f6 (https://towardsdatascience.com/a-beginners-guide-to-sentiment-analysis-in-python-95e354ea84f6) [Last Accessed 23 Feb. 2022].

- sdiabr. (2018) *What is effect of increasing number of hidden layers in a Feed Forward NN? [duplicate]*, Stack Overflow. Link: https://stats.stackexchange.com/questions/338255/what-is-effect-of-increasing-number-of-hidden-layers-in-a-feed-forward-nn (https://stats.stackexchange.com/questions/338255/what-is-effect-of-increasing-number-of-hidden-layers-in-a-feed-forward-nn). [Last Accessed 14 Mar 2022].
- Shah, R. (2021). *Tune Hyperparameters with GridSearchCV.* Analytics Vidhya. Link: https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/ (https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/) [Last Accessed 28 Feb 2022]
- Sharma, H. (2021). Topic Model Visualization using pyLDAvis - Towards Data Science. [online] Medium. Available at: https://towardsdatascience.com/topic-model-visualization-using-pyldavis-fecd7c18fbf6 (https://towardsdatascience.com/topic-model-visualization-using-pyldavis-fecd7c18fbf6) [Last Accessed 25 Feb. 2022].
- Simplilearn. (2021) *Overfitting And Underfitting Machine Learning | Machine Learning Tutorial For Beginners |Simplilearn*, YouTube Video, Link: https://www.simplilearn.com/tutorials/machine-learning-tutorial/overfitting-and-underfitting?source=sl_frs_nav_user_clicks_on_previous_tutorial (https://www.simplilearn.com/tutorials/machine-learning-tutorial/overfitting-and-underfitting?source=sl_frs_nav_user_clicks_on_previous_tutorial). [Last Accessed 3 Mar 2022]
- Snowball Stem (n.d.). *Snowball*. Snowballstem. Link: https://snowballstem.org/ (https://snowballstem.org/) [Last Accessed 07 Feb 2022]
- Susan Li. (2018) *Multi-Class Text Classification Model Comparison and Selection*, towardsdatascience.com, Link: https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568 (https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568). [Last Accessed 5 Mar 2022]
- Super User (2020). Visualizing Text Analysis Results with Word Clouds. [online] Dundas.com. Available at: https://www.dundas.com/resources/blogs/dundas-bi-features-functionality/visualizing-text-analysis-results-with-word-clouds (https://www.dundas.com/resources/blogs/dundas-bi-features-functionality/visualizing-text-analysis-results-with-word-clouds) [Accessed 5 Mar. 2022].

- Srinidhi, S. (2020). *Stemming of words in Natural Language Processing, what is it?* Medium. Link: https://towardsdatascience.com/stemming-of-words-in-natural-language-processing-what-is-it-41a33e8996e2 (https://towardsdatascience.com/stemming-of-words-in-natural-language-processing-what-is-it-41a33e8996e2) [Last Accessed 07

T

- Tableau. (2022). What Is Data Visualization? Definition, Examples, And Learning Resources. [online] Available at: https://www.tableau.com/learn/articles/data-visualization#:~:text=Data%20visualization%20is%20the%20graphical,outliers%2C%20and%20patterns%20in%20data (https://www.tableau.com/learn/articles/data-visualization#:~:text=Data%20visualization%20is%20the%20graphical,outliers%2C%20and%20patterns%20in%20data). [Accessed 5 Mar. 2022].
- Teja, S. (2020) *Stop Words in NLP*, Medium. Link: https://medium.com/@saitejaponugoti/stop-words-in-nlp-5b248dadad47 (https://medium.com/@saitejaponugoti/stop-words-in-nlp-5b248dadad47) [Last Accessed 06 Feb 2022]

U

- Unfold Data Science. (24 Apr 2020) *Feature Extraction techniques from text - BOW and TF IDF|What is TF-IDF and bag of words in NLP*, YouTube Video. Link: https://www.youtube.com/watch?v=76jmgV_ZPUs&list=PLmPJQXJiMoUUSqSV7jcqGiiypGmQ_ogtb&index=6 (https://www.youtube.com/watch?v=76jmgV_ZPUs&list=PLmPJQXJiMoUUSqSV7jcqGiiypGmQ_ogtb&index=6). [Last Accessed 07 Feb 2022]
- Unfold Data Science. (05 May 2020) *Understanding Word Embedding|What is Word Embedding|Word Embedding in Natural language processing*, YouTube Video. Link: https://www.youtube.com/watch?v=jN08NAD_-Ws&list=PLmPJQXJiMoUUSqSV7jcqGiiypGmQ_ogtb&index=9 (https://www.youtube.com/watch?v=jN08NAD_-Ws&list=PLmPJQXJiMoUUSqSV7jcqGiiypGmQ_ogtb&index=9). [Last Accessed 08 Feb 2022]

W

- w3resource. (2020) *NLTK corpus: Omit some given stop words from the stopwords list*, w3resource. Link: https://www.w3resource.com/python-exercises/nltk/nltk-corpus-exercise-5.php (https://www.w3resource.com/python-exercises/nltk/nltk-corpus-exercise-5.php) [Last Accessed 06 Feb 2022]
- W3Schools (n.d.). *Python String startswith() Method.* W3schools. Link: https://www.w3schools.com/python/ref_string_startswith.asp (https://www.w3schools.com/python/ref_string_startswith.asp) [Last Accessed 09 Feb 2022]
- Wang, J. (2020). Sentiment Analysis & Topic Modeling for Hotel Reviews. [online] Medium. Available at: https://medium.com/swlh/sentiment-analysis-topic-modeling-for-hotel-reviews-6b83653f5b08 (https://medium.com/swlh/sentiment-analysis-topic-modeling-for-hotel-reviews-6b83653f5b08) [Last Accessed 25 Feb. 2022].
- W, L. (2017). Necessary to apply TF-IDF to new documents in gensim LDA model? [online] Stack Overflow. Available at: https://stackoverflow.com/questions/44781047/necessary-to-apply-tf-idf-to-new-documents-in-gensim-lda-model/44789327#44789327 (https://stackoverflow.com/questions/44781047/necessary-to-apply-tf-idf-to-new-documents-in-gensim-lda-model/44789327#44789327) [Last Accessed 25 Feb. 2022].

Z

- Zubair Hossian (2022) *Implementation of Basic NLP Techniques with spaCy*, Towards Data Science. Link: https://towardsdatascience.com/hands-on-implementation-of-basic-nlp-techniques-nltk-or-spacy-687099e02816 (https://towardsdatascience.com/hands-on-implementation-of-basic-nlp-techniques-nltk-or-spacy-687099e02816) [Last Accessed 08 Feb 2022]