NLP
REPORT
ASSIGNMENT 1

NAVIDHA JAIN: B-TECH 2020223
MANAV SAINI: B-TECH 2020518

SECTION I: REGULAR EXPRESSION

A) We use the pandas library to read the csv file.

a) i) **Methodology**:Simply using the split function of regex to break the tweets into sentences according to our assumptions. Regex used follows the assumptions mentioned below.

Assumptions:

- Sentence ends with a delimiter, ie., ? or ! or .
- String1.string2 is one sentence.
- String… or string.. Is one sentence
- String.\s Is one sentence
- Decimals do not come under different sentences
- string1!string2! are 2 different sentences, string1 and string2

Helper Functions: self-made function for an individual tweet which also eliminates the empty strings formed.

ii) **Methodology:**First we break up the tweets according to white spaces, then for each string obtained we tokenize it and add it to the list of our tokens. Regex used follows the assumptions mentioned below.

Assumptions:

- A word is a token.
- A special character is a token.
- A permutation of digits, special characters and alphabets are different tokens respectively.
- Decimals are broken into 3 tokens, full stop and 2 numbers
- A number is a single token

Helper Function: self-made function for an individual tweet which also eliminates the empty strings formed and makes sure the tokens are not repeated..

b) **Methodology:**We make different functions to count the number of words starting with consonants and vowels respectively. We split the individual tweet according to white spaces and check whether each word starts with a consonant or a vowel and adds it to our respective list.

Assumptions: NA

Helper Functions: self-made function for an individual tweet which maintains the list of consonants and vowels respectively.

c) **Methodology:** We use the already obtained total tokens from part (a). We make use of the Match Object class here which is basically the matched part of the regex from the string provided. We maintain 2 dictionaries to ensure the uniqueness, one for before lowercasing and one for after lowercasing.

Assumptions: all assumptions from part (a) are applicable.

Helper Functions: self-made function to change the case and python's lower() to change the case.

Pre-processing Steps: Breaking the text into tokens.

d) **Methodology:** We use a function to count the number of usernames in a single tweet and add it to the final count of usernames. Regex used follows the assumptions mentioned below.

Assumptions: a username is a string of alphanumeric characters and underscores starting with '@'.

Helper Functions: self-made function for an individual tweet.

e) **Methodology:** We use a function to count the number of urls in a single tweet and add it to the final count of urls. Regex used follows the assumptions mentioned below.

Assumptions: a url is a string of alphanumeric characters, with '/'s and '.'s in between starting with 'http://'.

Helper Functions: self-made function for an individual tweet.

f) **Methodology:** We combine all the entries in the DATE_TIME column to a single string and then using the regex findall function we maintain the frequency of the tweets each day of the week.

Assumptions: NA.

Helper functions: strip(): to remove the leading and trailing white space from the string and process it.

B)

a) i) **Methodology:** We input the word from the user and using a loop count the number of occurrences using the regex functions using the finditer function of the re library. Regex used follows the assumptions mentioned below.

Assumptions: The search is case sensitive.
Word is exactly the sequence of characters entered.
All previous assumptions are valid.

Helper Functions: self-made function for an individual tweet.

ii) **Methodology:** We input the word from the user and using a loop count the number of sentences containing that word using the regex functions. Regex used follows the assumptions mentioned below.

Assumptions: The search is case sensitive.
Word is exactly the sequence of characters entered.
All previous assumptions are valid.

Helper Functions: NA

Pre-processing Steps: Breaking the text into sentences.

b) **Methodology:** We input the word from the user and using a loop count the number of sentences that start with that word using the regex functions. Regex used follows the assumptions mentioned below.

Assumptions: The search is case sensitive.
Word is exactly the sequence of characters entered.
All previous assumptions are valid.

Helper Functions: NA

Pre-processing Steps: Breaking the text into sentences.

c) **Methodology:** We input the word from the user and using a loop count the number of sentences that end with that word using the regex functions. Regex used follows the assumptions mentioned below.

Assumptions: The search is case sensitive.
Word is exactly the sequence of characters entered.
All previous assumptions are valid.

Helper Functions: NA

Pre-processing Steps: Breaking the text into sentences.

SECTION II: TEXT PREPROCESSING

We use pandas library to read the csv file and the nltk library to carry out the preprocessing steps.

a) **Methodology:** We have used word_tokenize function from the nltk.tokenize library to tokenize the sentences into words.

Assumptions: It splits tokens based on white space and punctuation.

Helper Functions: NA

b) **Methodology:** To correct the spellings we have used the fixfragment function from jamspell library.
Assumptions: NA

Helper Functions: NA

c) **Methodology:** We have used WordNetLemmatizer from nltk.stem library to stem the words. We have used PorterStemmer from nltk.stem library to lemmatize the words. We use these function on each word present in the list that we got from part a.

Assumptions: NA

Helper Functions: NA

d) **Methodology:** To remove the punctuations we have use the sub function present in the library re. The regex that is declared inside the function is [^\w\s] which means that all the characters that are not an alphabet with a space should be substituted with a space in the string passed.

   Assumptions: Punctuations are the characters which are not alphabets succeeding with a space

   Helper Functions: NA

e) **Methodology:** To remove the stopwords we first build a regex statement that contains all the stopwords with a '|' character between them using the compile function from the re library. We have imported all the stopwords present in the english language from the nltk.corpus library. Then we have used the sub function present in the re library to substitute all the stopwords with an empty string.

   Assumptions: The stopwords imported from the nltk library are the only stopwords

   Helper Functions: NA

f) **Methodology:** To remove the extra spaces we have used the sub function from the re library to replace all the spaces with extra white space with a single space. The regex declared in the function is \s\s+ which means the all the spaces succeeding with on or more spaces should be replaced with a single space.

   Assumptions: NA

   Helper Functions: NA

g) **Methodology:** to remove the URL and the HTML tags we have used the sub function present in the re library. The regex declared in the function is (\<\w+>)|(\bwww.\w+.\w+/?\w*)|(\bhttp://\w+.\w+/\w+) which means that the possible urls and the html tags should be replaced with empty string in the string passed to the function.

   Assumptions: NA

   Helper Functions: NA

SECTION III: VISUALIZATION

a) **Methodology:** We have used the WordCloud function from the wordcloud library to create the wordcloud and have used the matplotlib to plot it.

   Assumptions: NA

   Helper Functions: NA

b) Methodology: We have created separate dataframes for each class and then have created wordclouds for each class.

   Assumptions: NA

   Helper Functions: NA

   Observations: The wordcloud of positive class contains words like love, thanks, good, work, eat, think, go etc.
   The wordcloud of negative class contains words like go, time, sad, feel, sorry, kill, poor,etc.
   Some words like go, work are common in both the word clouds because such words can be used in positive as well as negative sentences.

SECTION IV: RULE-BASED SENTIMENT ANALYSIS

a) Methodology: To get the sentiment value of each text we used VADER which is an inbuilt package. For pre processed and raw data we passed all the sentences to the function and stored the sentiment value in the dataframes.

   Assumptions: NA

   Helper Functions: NA

b) Methodology: We compared the sentiment value that was already given to us with the values we got from part a and calculated accuracy with the help of formula no.of correct classifications/total classifications

   Assumptions: NA

   Helper Functions: NA