Natural Language Processing - CSE556

ASSIGNMENT 1

SIDDHANT AGARWAL (2020247) and SHANTANU DIXIT (2020118)

SECTION 1:
Libraries used:
Pandas for loading data
Re for Regular Expressions

Part A:

a. Assumption 1: sentences are a set of continuous tokens ending with the punctuations [.], [!] or [?].
   Assumption 2: in tokenisation, we tokenise only the words and do not consider the punctuations as separate tokens.
   Wrote regex expressions for matching sentences and tokens as below and then used re.findall() to match get all the sentences and tokens in the text data
   regex_sentence = r'\b[^!\?\.]+[!\?\.]?'
   regex_token = r'\b\S+\b'
   <u>Observations</u>:
   For Class Label 0 :
      Average Sentences = 1.8455
      Average Tokens = 13.3965
   For Class Label 1 :
      Average Sentences = 1.9588981198076083
      Average Tokens = 12.584171403585483

b. Used the following regex expressions for matching tokens starting with consonants and vowels respectively.
   regex_consonant =
   r'\b[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]\S*\b'
   regex_vowel = r'\b[aeiouAEIOU]\S*\b'
   Used re.findall() function for matching from data text.
   <u>Observations</u>:
   For Class Label 0 :
      Words starting with consonants = 20057
      Words starting with vowels = 6963
   For Class Label 1 :
      Words starting with consonants = 21710

Words starting with vowels = 7169

c. Used regex_token = r'\b\S+\b' expression to get a token.
   Before lowercasing, we observe:
   For class 0 :
   Total number of tokens = 26793
   Unique number of tokens = 6682
   For class 1 :
   Total number of tokens = 28780
   Unique number of tokens = 8163
   Then we use re.sub() method to lowerall all the text alphabet by alphabet.
   After lowercasing we observe:
   For class 0 :
   Total number of tokens = 26793
   Unique number of tokens = 5909
   For class 1 :
   Total number of tokens = 28780
   Unique number of tokens = 7164

d. We use the regex expression regex = r'@\w{4,15}\b' to find out all usernames. An assumption taken from the twitter official guidelines for twitter usernames is that they can only be between 4 and 15 characters long (both inclusive) and may use only alphanumeric characters. (represented by \w in regex).
   <u>Observations</u>:
   For class label 0 :
   Number of Usernames = 802
   For class label 1 :
   Number of Usernames = 1301
   A list of all these usernames is given in the .ipynb file.
   We have also mentioned another initial regex expression that failed to capture usernames following a [.] or enclosed in brackets. We believe this is a good edge-case that we have captured successfully in our final regex expression.
e. Assumption: URLs can be of two form in the data: the first is those with start with http:// and the second are those which do not start with http:// but have .com in the middle such as digg.com
   regex =
   r'http://[a-zA-z0-9_\-\.]+\.[a-zA-z0-9_\-\.~:/\?#\[\]@!\$&\'*\+,

;=]+|[a-zA-z0-9_\-\.]+\.com[a-zA-z0-9_\-\.~:/\?#\[\]@!\$&\'*\+,;=]*'

This regex captures both the cases using an | (OR) operator.

Observations:

For class 0 :

Number of URLs = 60

For class 1 :

Number of URLs = 142

A list of all these URLs is present in the .ipynb file

f. We used re.search() function with the DATE_TIME column of the data in this task. Using if/else conditions we checked for all the seven days of the week.

Observations:

For Class 0 Number of tweets per day:
Mon: 391
Tue: 154
Wed: 127
Thu: 171
Fri: 473
Sat: 119
Sun: 565

For Class 1 Number of tweets per day:
Mon: 481
Tue: 132
Wed: 172
Thu: 50
Fri: 391
Sat: 298
Sun: 763

Part B:

a. We first define the class_label and given_word which can be edited as per the requirements of the tester.
We use the regex, regex_word = r'\b{word}\b'.format(word = given_word)

and regex_sentence =
r'\b[^!\?\.]+\b{word}\b[^!\?\.]*[!\?\.]?|\b{word}\b[^!\?\.]*[!\?\.]?'.format(word=given_word)
To match all occurrences of our given word and the sentences containing the given word. We use re.findall() function for these matchings.
We store the sentences obtained from this in a list.
An example is given in the .ipynb file.

b. We use the sentences obtained in the previous part and use the regex, regex_start = r'^\b{word}\b[^!\?\.]*[!\?\.]?'.format(word = given_word)
with the re.findall() function to find all matches of sentences where the sentence starts with the given word.

c. We use the sentences obtained in the previous part and use the regex, regex_end =
r'\b[^!\?\.]*\b{word}\b[!\?\.]|\b[^!\?\.]*\b{word}\b$'.format(word = given_word)
with the re.findall() function to find all matches of sentences where the sentence ends with the given word.

SECTION 2:

Libraries used :
Pandas for data loading
NLTK for preprocessing tasks
string for punctuation removals
Re for regular expressions

1. Tokenization -
   used word_tokenize to separate words from sentences
   Function used: tokenize(txt)
2. Spelling Correction - used Py SpellChecker by Peter Norvig
   a. Involves removal of punctuations and whitespace characters both of which are implemented by regex
      Functions used:
         1.punctuation_removal(txt)
         2.ltos(s)

    3.formsentence(txt)
    4.remove_extraspaces(txt)
    5.SpellParent(s)

   b. After performing (a), pass the text through .unknown() functionality of SpellChecker()
   c. A list containing probable misspelled words is obtained, store them in a dictionary/lists as per choice

   d. Iterate through the tweet and rectify the misspelled words

3. Lemmatization: used lemmatization to change a particular word to its root word
   a. Use the functionality WordNetLemmatizer provided by NLTK

   Function used:
   → lemmatize(txt)
4. Punctuation Removal: removed all the punctuations present in string.punctuation in one pass

   Function used:
   → punctuation_removal(txt)
5. Stopword removal: imported all the stopwords as per NLTK, regex for removal of those.

   Function used:
   → stopword_removal(txt)
6. Extra white spaces removal: used re.sub() functionality for removal of extra whitespaces.

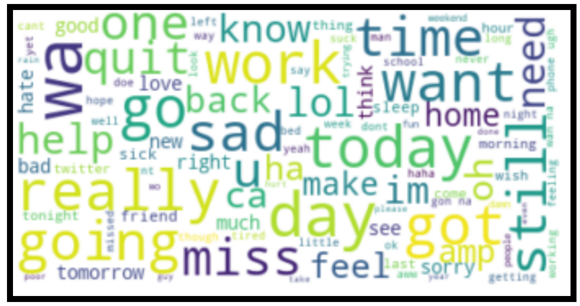
   Function used:
   → remove_extraspaces(txt)
7. HTML and URL tags removal: used re.sub() for html and url removal.

   Function used:
   → removeURL_TAG(txt)

SECTION 3:

Libraries used:

Word cloud for analyzing the frequently occurring words in both the classes



Positive word cloud

Negative word cloud

a. Generated word cloud after tokenizing and lowercasing the tweets for both classes
b. OBSERVATIONS:
   1.
   For the positive tweets,the most frequently occuring words in the preprocessed text include "love","good","new","lol","thank","hope","great" all which are commonly used in positive tweets.
   2.
   For the negative tweets,the most frequently occuring words in the preprocessed text include "sad","miss","sorry","quit", which are used often in negative tweets. Words such as "really" and "still" also find context in not so positively stated sentences and hence are prominent in negative cloud.
   3.
   There are a few outliers as well, such as "quit" but context of the sentence changes as they are used .For example: "I won’t quit ever" and "I am such a loser,I quit" have two completely different meanings but both use the word "quit".

SECTION 4:
VADER sentiment analysis
Libraries used: VADER for sentiment intensity analyzer
Metric for classification: compound score
Obtained raw text prediction and preprocessed text prediction

User written accuracy function which matches the prediction while going through the predictions in one pass.
Best possible accuracy obtained was corresponding to compound_score>=0
Raw text accuracy obtained = 68.62%
Preprocessed text accuracy obtained = 66.57%

CONTRIBUTIONS:
Siddhant Agarwal 2020247 - PART 1 (20 MARKS)
Shantanu Dixit 2020118 - PART 2, 3, 4 (30 MARKS)

- constant discussion and inputs for all parts by both team members.