FIT5196 Assignment 1

Task 1

Student name: Ayush Sharma

Due date: 13 Sept 2020

Environment: Python 3.6.0 and Anaconda 4.3.0 (64-bit)

Introduction

Steps followed:

1.) Import xlsx data into dataframe and Remove empty rows/columns and Remove non-english tweets using langid Combine all tweets for a sheet.

2.) Tokenize combined tweet using RegexpTokenizer

3.)Generate top 100 bigrams for each sheet using BigramCollocationFinder and Write top 100 bigrams, count for each date to bigram.txt

4.) Remove stop words using stopword file, Remove words appearing in more than 60 sheets and less than 5 sheets, Remove words with length less than 3

5.) Stemming all the tokens

6.) Get top 100 unigrams for each sheet and count and Write top 100 unigrams, count for each date to unigram.txt

7.) Generate top 200 bigrams on complete text from the original tokens

8.) Get vocab by taking unique words from all stemmed unigrams Add 200 bigrams to the vocab appending them with underscore Write vocab to vocab.txt

9.) Create sparse matrix by with vocab ids of words in for each date and their count and Write this to countVec.txt

Libraries Used:

- re: This Library is used to tokenize the
- langid: “langid” package, only keeps the tweets that are in English language
- pandas as pd: Load, explore, work on the dataframe that is loaded
- nltk: NLTK is a Natural language text processing liabrary used to perform text analysis
- nltk.tokenize , RegexpTokenizer: This library tokenise the regular expression passed into It.
- from nltk.collocations import * used to create the bigrams
- from nltk.stem import PorterStemmer: This library is used form stemming
- from itertools import chain: Helps in combing various Lists
- from nltk import bigrams: Generate bigram

- from nltk import FreqDist: gives frequencey of the bigram or unigram

1.) Importing Libraries

In [1]:

```
import re
import langid
import pandas as pd
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk import bigrams
from nltk.util import ngrams
from nltk.collocations import *
from nltk.stem import PorterStemmer
from itertools import chain
from nltk import bigrams
from nltk import FreqDist
import itertools
from nltk.collocations import *
from collections import Counter
```

Step 2: Reading and Analysing data

The data contains of 80 sheets of excel that have thousands of tweet text and date along with id

- Read the data
- parsing through the excel files and sheets and removing unwanted NA's and reshaping it
- Applying langid on each text if not english the text will not be appended tp the dictionary
- finally creating a dictionary that has date as key and text as a list of values

In [2]:

```
# Reading the Excel data
data=pd.ExcelFile('30823293.xlsx')
# creating blank dict for sheets
tweetDict={}
# Iterating through each sheet and re shaping it into a better and more usefull format
for i in range(len(data.sheet_names)):
    # parsing through each sheet
    df=data.parse(i)
    # removing na values
    df=df.dropna(0, how = 'all')
    df=df.dropna(1, how = 'all')
    # Seting columns names
    df.columns = ["Text","Id","Date"]
    # resetting indexes
    df.index = range(len(df.index))
    df=df.drop([0])
    df.index = range(len(df.index))
    # iterating over the Text column to perform tasks like checking if the type is str
    # Dropping all those values that are not required
    for j in df['Text']:
        if type(j)!=str:
            nonstrindex=df[df['Text']==j].index
            df.drop(nonstrindex,inplace=True)
            df.reset_index(drop = True)
    # Using the langid checking each of the Text to remove all non english tweet
        if langid.classify(str(j))[0]!='en':
            textindex=df[df['Text']==j].index
            df.drop(textindex,inplace=True)
            df.reset_index(drop = True)
    # Creating a Dictionary with date as key and tweets texts as value
    tweetDict[data.sheet_names[i]] = '\n'.join(df.Text.values).lower()
```

Step 3: Tokenizing

- Tokenization refers to creating each words in the text as token by applying the regex `[a-zA-Z]+(?:[-'][a-zA-Z]+)?`
- Tokeinzing the data of text and applying regex to create the token
- applying tokenization for to the dictionary that has all the text

In [3]:

```
# Creating tokens
tokenizer= RegexpTokenizer("[a-zA-Z]+(?:[-'][a-zA-Z]+)?")
unigaram_token={}
# Applying token to the dictionary data
for i,j in tweetDict.items():
    unigaram_token[i]=tokenizer.tokenize(j)


```

Step 4 Generating 100 Bigram:

- Using the Counter function to calculate the top 100 bigrams by using the package nltk.bigrams that finds each bigram for the text
- Then structuring the bigrams that we have in a way that includes the count of the bigram and formating according to output file
- Finally having a structured formate of a dictionary with key as date and bigrams as a list of values with bigrams within a tuple and a tuple that has the count of each bigram.
- Writing a text file that have the

In [4]:

```
# Finding the top 100 bigrams words and counting their occurences
bigramdict={}
for i,j in unigaram_token.items():
    bigramword=list(nltk.bigrams(j))
    bigramcount=Counter(bigramword).most_common(100)
    bigramdict[i]=bigramcount
```

In [5]:

```
# Formating the bigrams extracted in the required way to write the file
final_bi_file_str=str()
# iteratin within the dict to create the text
for i,j in bigramdict.items():
    final_bi_file_str=final_bi_file_str+str(i)+':'+str(j)+'\n'
```

In [11]:

```
# creating and writing the 100bi.txt
bigramtxtfile = open("30823293_100bi.txt", "w",encoding='UTF-8')
bigramtxtfile.write(str(final_bi_file_str))

bigramtxtfile.close()
```

Step 5 Removing stopwords

- Stop words are the words least significant words that are appearing frequently like a, an , the , is etc
- Stop words are removed using the text file provided by the faculty that will help us remove such words.
- we will ieterate over our token dictionary by refering to each key which in our case is the date and look for each element within the value and selecting the element of the value to check if it is a stopword or not if it is a stop word then it will be removed from the data.
- after that we have calculated the document frequency of the word by using `FreqDist(list(chain.from_iterable)` to find if a word is in more that 60 document or if less than 5 documents after finding them these words are removed from the dictionary
- now appending the new dictionary after handling all the removals

In [12]:

```
# opening the stopword file provided to us by faculty
stopword=[]
with open('stopwords_en.txt') as f:
    stopword=f.read().splitlines()
```

In [13]:

```
# filtering and removing the stopwords
tokenised_dict={}
for i,j in unigaram_token.items():
    tokenised_dict[i]= [x for x in j if x not in stopword]
# Filtering the words with len less that 3 characters within the dictionary
for i,j in tokenised_dict.items():
    tokenised_dict[i]= [x for x in j if len(x)>3]
```

In [14]:

```
# romoving less favaroute word i.e words with < 5 count in the document frequency
#and words with >60 in frequency and creating new dict
list_final={}
for i,j in tokenised_dict.items():
    list_final = FreqDist(list(chain.from_iterable([set(x) for x in tokenised_dict.valu

list_remove=[]
for i in list_final.keys():
    if list_final.get(i)>60 or list_final.get(i)<5:
        list_remove.append(i)

for i in list_remove:
    list_final.pop(i)
```

In [15]:

```
# appending the above results to new dict
tokenised_dict2 = {}
for i in tokenised_dict:
    lst=tokenised_dict.get(i)
    refined_list=[]
    for each in lst:
        if(each  in list_final):
            refined_list.append(each)
    tokenised_dict2[i]=refined_list
```

Step 6 Stemming

- We have used PortterStemmer for stemming because we were asked to use it.
- (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English.
- The Porter stemming algorithm removes the commoner morphological and inflexional endings from words in English. like consider, considering, speed, speeding , sleep, sleeping
- remove the words including suffix like `ing` or `e` etc thus such words will be converted to similar words after stemming.

In [16]:

```
# Stemming the tokens using porterstemmer and passing it into a dictionary
tokenised_dict3 = {}
ps = PorterStemmer()
for i in tokenised_dict2:
    lst=tokenised_dict2.get(i)
    refined_list=[]
    for each in lst:
        refined_list.append(ps.stem(each))
    tokenised_dict3[i]=refined_list
```

Step 6 Generating top 100 Unigrams:

- In this step we have all the umograms after removing unwanted characters and words and steming the tokens
- now we will count the freqency of all the token for each date
- Finally we will collect and keep the top 100 unigrams that we need to keep
- In the final step we will format the top 100 unigram as per the required output and write a text file for it.

In [17]:

```
# Extracted the freq of each remaining tokens after stemming and removing stopwords
#and selecting the top 100 unigrams and storing it
unigram_dictCount={}
for i,j in tokenised_dict3.items():
    unigram_dictCount[i]=FreqDist((j)).most_common(100)

```

In [18]:

```
# Formating the unigrams according to the required output to be stored
final_uni_file_str=str()
for i,j in unigram_dictCount.items():
    final_uni_file_str=final_uni_file_str+str(i)+':'+str(j)+'\n'
```

In [19]:

```
# writting the 100uni.txt files
unigramtxtfile = open("30823293_100uni.txt", "w",encoding='UTF-8')
unigramtxtfile.write(str(final_uni_file_str))
unigramtxtfile.close()
```

Step 7 Generating top 200 Bigrams:

- with the help of `nltk.collocations.BigramAssocMeasures()` and `nltk.collocations.BigramCollocationFinder.from_words` we will apply this to get the top 200 bigrams using pmi method

In [20]:

```
# Generating top 200 bigrams from all tokens within data
allbigram=[]
bigram_measure=nltk.collocations.BigramAssocMeasures()

#for i,j in unigaram_token.items():
words_bi = list(chain.from_iterable(unigaram_token.values()))
finder=nltk.collocations.BigramCollocationFinder.from_words(words_bi)
allbigram.append(finder.nbest(bigram_measure.pmi,200))

```

Step 7 Creating Vocab of Words

- now we have 200 bigrams and unigrams in we will format them to keep them in the list and create a formated list
- we will iterate on the final list and sort it and then put it in a string that will make it formated as per the requirement
- finally we will write the vocab.txt

In [21]:

```
# creating a list of bigrams with each bigram appened to each other as a word for vocab
bigramList=[]
for i in allbigram:
    for j in i:
        if j not in bigramList:
            bigramList.append(j[0]+'_'+j[1])

```

In [22]:

```
#Creating a list of unigrams
vocabList=[]
for i,j in tokenised_dict3.items():
    for x in j:
        vocabList.append(x)


```

In [24]:

```
Vocab = []
finalVocab = []
finalVocab=vocabList+bigramList
Vocab=set(finalVocab)
Vocab_list = list(Vocab)
Vocab_list = sorted(Vocab_list)
```

In [25]:

```
string_vocab = ''
for i in range(len(Vocab_list)):
        string_vocab = string_vocab + str(Vocab_list[i]) + ':' + str(i) + '\n'


# writting the vocab.txt files
vocabtxtfile = open("30823293_vocab.txt", "w",encoding='UTF-8')
vocabtxtfile.write(str(string_vocab))
vocabtxtfile.close()
```

In [26]:

```
# writting the vocab.txt files
vocabtxtfile = open("30823293_vocab.txt", "w",encoding='UTF-8')
vocabtxtfile.write(str(string_vocab))
vocabtxtfile.close()
```

Conclusion:

In this task I have created tokens for each date. Then I have carried out steps which includes removal of stopwords, context independent words, context dependent words amd words with length less than 3. After this I have carried out stemming for each token. Hence I have created unigrams and vocab files. Also I have generated bigrams which occur the most frequently using collocations and pmi measure techniques. Hence the following files have been generated.

Hence after this task I have gained knowledge on how to work with nltk library and I have understood the concepts of unigrams and bigrams

In [ ]:

```

```