FIT5196 Assessment 1

Student Name: Rahul Agarwal

Student ID: 30745012

Data: 13/09/2020

Version: 2.0

Environment: Python 3.8 and Anaconda 6.1.1 (64-bit)

Libraries used:

- os (For reading the data from the system)
- langid (for check the language of the text)
- nltk 3.2.2 (Natural Language Toolkit, included in Anaconda Python 3.6)
- nltk.collocations (for finding bigrams, included in Anaconda Python 3.6)
- nltk.tokenize (for tokenization, included in Anaconda Python 3.6)
- nltk.corpus (for stop words, not included in Anaconda, `nltk.download('stopwords')` provided)

1. Introduction

This assignment comprises the execution of an Excel file regarding COVID-19 tweets. The excel file contains the 80+ days of COVID-19 related tweets. The excel file contains the 80+ sheets and each sheet has 2000 tweets.

Each excel sheet contains information regarding tweets i.e `id`, `text`, and `created_at` attributes.

The task requires the following steps:

1. Extracting the data from the excel file.
2. check whether the tweet text is English or not and filtered out the non-English tweets.
3. Tokenize the data according to `[a-zA-Z]+[?[:!\-]*[a-zA-Z]+]?`
4. Generate the 100 most common bigrams and store the frequency of the bigrams into a text file.
5. Generate the 100 most common unigrams and store the frequency of the unigram into a text file.
6. Create a stored and indexed vocabulary list contain 200 most common bigrams and all the stemmed unigrams.
7. Create a count vector-matrix based on the vocabulary list created in the previous step and store the frequency of each vector according to date into a text file.

2. Import libraries

In [1]:
```
import pandas as pd
import langid
import nltk
from nltk.tokenize import RegexpTokenizer
import re
import sys
```

3. Reading the data

As a first step, all the excel files are loaded into Dataframe and then remove the unnecessary rows and columns. Then the text is extracted from the data frame and saved into a dictionary.

In [2]:
```
# Reading the excel file and removing unwanted rows
def format_data(data):
    '''
    Return text column from the excel file

    Remove NaN and duplicate columns
    '''
    data.dropna(axis=0, how='all',inplace=True)
    data.dropna(axis=1, how='all', inplace=True)
    data.reset_index(drop=True,inplace=True)
    data.drop(0,axis=0,inplace=True)
    data.columns = ['text','id','created_at']
    data.reset_index(drop=True,inplace=True)
    data.drop_duplicates(inplace=True)
    return data['text']

# Reading the excel file
# df = pd.ExcelFile('part2/part2/sample.xlsx')
# df = pd.ExcelFile('30745012.xlsx')
sheets = df.sheet_names
data_dict = {sheet:format_data(df.parse(sheet)) for sheet in sheets}
```

4. Filtering out non-English tweets

Langid package is used to remove the non-English tweets and store the data into a dictionary

In [3]:
```
# Filter out the non-English text
data_dict = dict(map(lambda x: (x[0],[word for word in x[1] if langid.classify(str(word).encode('utf-16','surrogatepass').decode('utf-16'))[0] =='en']),data_dict.items()))

data_dict_str = dict(map( lambda x:(x[0], ' '.join(map(str, x[1]))), data_dict.items() ))
```

5.Tokenizing the data

The data is tokenized using `RegexTokenizer` and store the data into a dictionary where the key is the date and tokenized text as value

In [4]:
```
tokenizer = RegexpTokenizer("[a-zA-Z]+[?[:!\-]*[a-zA-Z]+]?")

def tokenised_text(key,text):
    '''
    return the key and token list

    key =  dates
    text = twitter text in string format
    '''
    tokens = tokenizer.tokenize(str(text).replace("\n"," ").lower())
    return (key, tokens)


tokenised_data = dict(tokenised_text(key, value) for key,value in data_dict_str.items())
```

6.Generating the 100 most common bigrams

The tokenized dictionary is used to generate 100 most common bigrams and store the frequency of bigrams into a text file.

In [5]:
```
from nltk.util import ngrams
from nltk.probability import *

def generate_bigram(token_list):
    '''
    return the 100 most common bigrams
    input: token_list for each date
    '''
    return FreqDist(ngrams(token_list, n=2)).most_common(100)

bigrams = dict(map( lambda x : (x[0], generate_bigram(x[1])), tokenised_data.items() ))
```

In [6]:
```
def write_file(file_name, data):
    '''
    write the data into the given file_name

    input:  file Name and data

    '''
    with open(file_name, 'w') as f:
        for key,value  in data.items():
            f.write("{}:{}".format(key,value))
            f.write('\n')

write_file('30745012_100bi.txt', bigrams)
```

7. Generating the unigrams

7.1. Removing stop word and stemming

In order to generate 100 most common unigram, we have to remove the stop word and stemmed the data using porter stemmer.

Then `FreqDist` is used to calculate the frequency of each token and store the result into a text file.

In [7]:
```
# Read the stop word file
file = open('part2/stopwords_en.txt','r')
content_independent_stop_words  = file.read().split('\n')
file.close()
# content_independent_stop_words = content_independent_stop_words
#creating the porter stemmer object
ps = nltk.PorterStemmer()


def remove_stop_word(key,value):
    '''
    return the key value pair after removing the stop and applying stemming

    '''
    tokens =  list(filter(lambda x: len(x)>2, value))
    tokens =  list(filter(lambda x: x not in content_independent_stop_words, tokens))
    return (key,tokens)

def apply_stemming(key, value):
    '''
    return the key value pair after applying stemming

    '''
    stemmed_list = list(map(lambda x:ps.stem(x),value))
    return (key,stemmed_list)

text_without_stop_word = dict(remove_stop_word(key,value) for key,value in tokenised_data.items())

stemmed_text = dict(apply_stemming(key,value) for key,value in text_without_stop_word.items())
```

In [8]:
```
def generate_unigram(token_list):
    '''
    return: 100 most common unigrams

    input: list of token for each date
    '''
    return FreqDist(token_list).most_common(100)


unigrams = dict(map( lambda x:(x[0], generate_unigram(x[1])), stemmed_text.items() ))
write_file('30745012_uni.txt', unigrams)
```

8. Creating the vocabulary

To create Vocabulary:

- **Generate 200 most common bigram using `nltk.collocations.BigramAssocMeasures` function through PMI measure**
- **Generate unigram after removal of dependent and independent stopword and then stemmed the tokenized data**
- **Concatenate the unigram and bigram and store the sorted result into the text file.**

In [9]:
```
from itertools import chain
from nltk.tokenize import MWETokenizer

# generating the unique data wise text in the dictionary
unique_data_wise_text = dict((key,set(value)) for key,value in text_without_stop_word.items())

# combining the dictionary value and store into list
word_list = list(chain.from_iterable(unique_data_wise_text.values()))
# calculating the the frequency of each token
word_freq = FreqDist(word_list)
# filtering the token whose document frequency is less than 5 and greater than 60
word_freq = dict(filter(lambda x : x[1]>=5 and x[1]<=60, word_freq.items()))
# applying the porter stemmer
words = list(map(lambda x : ps.stem(x), word_freq.keys()))


# generating the bigrams according to pmi measure
bigram_measures = nltk.collocations.BigramAssocMeasures()
bi_token = list(chain.from_iterable(tokenised_data.values()))
finder = nltk.collocations.BigramCollocationFinder.from_words(bi_token)
pmi_bigram_200 = finder.nbest(bigram_measures.pmi, 200)

mwe_tokenizer = MWETokenizer(pmi_bigram_200)

def mwe_generator(token_list):

    mwe_token =  mwe_tokenizer.tokenize(token_list)
    bigram_tokens =  list(filter(lambda x: '_' in x, mwe_token))
    return bigram_tokens

mwe_tokens = list(map(lambda x: mwe_generator(x), tokenised_data.values()))
mwe_tokens = list(set(chain.from_iterable(mwe_tokens)))

# # adding unigram and bigram
vocab = list(set(words)) + mwe_tokens

vocab.sort()
with open('30745012_vocab.txt','w') as f:
    for index,word in enumerate(vocab):
        f.write("{}:{}".format(word,index))
        f.write('\n')
```

9. Creating a count vector

This task consists of extracting the tweets and calculating its corresponding sparse count vector. The count vector is a collection of words, frequency pairs that count the number of occurrences of every word in the data.

The word is used in sparse matrics is extracted according to the following rules:

1. 200 most common bigram are generated using `PMI measure` and `MWETokenizer` is used to tokenize the bigram from the list of words
2. Unigrams are generated after removing context-dependent and independent stop words and stemming the word.
3. Then the vocabulary list is used to create count vectors.
4. After that tocond() function is used to extract the row, columns, and data information from the sparse matrix.
5. In the end, the data is stored in a text file.

In [10]:
```
from nltk.tokenize import MWETokenizer
from sklearn.feature_extraction.text import CountVectorizer

with open('30745012_vocab.txt','r') as f:
    vocab = f.read().split('\n')[:-1]
vocab = {x.split(':')[0]:x.split(':')[1] for x in vocab }
vocab

mwe_tokenizer = MWETokenizer(pmi_bigram_200)
def mwe_generator(token_list):

    mwe_token =  mwe_tokenizer.tokenize(token_list)
    mwe_token = list(filter(lambda x: len(x)>2, mwe_token))
    mwe_token = list(filter(lambda x: x not in content_independent_stop_words, mwe_token))

    bigram_tokens =  list(filter(lambda x: '_' in x, mwe_token))
    unigram_tokens = [ps.stem(x) for x in mwe_token if '_' not in x]
    mwe_token =  bigram_tokens + unigram_tokens
    return mwe_token
mwe_tokens = dict(map(lambda x: (x[0],mwe_generator(x[1])), tokenised_data.items()))
```

In [11]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(analyzer = "word",vocabulary = vocab.keys() )
data_features = vectorizer.fit_transform([' '.join(value) for value in mwe_tokens.values()])
print(data_features.shape)
```

```
(8, 1836)
```

In [12]:
```
dates = list(mwe_tokens.keys())
import itertools
def generate_countvec_vec_file(features):
    cx = features.tocoo()
    count_vector = []
    for i,j,v in itertools.zip_longest(cx.row, cx.col, cx.data):
        count_vector.append([dates[i],str(j), str(v)])

    count_vector = pd.DataFrame(count_vector, columns=['date','token','count'])
    with open('30745012_countVect.txt','w') as f:
        for date in dates:
            token = count_vector[count_vector.date == date][['token','count']].set_index('token').to_dict()['count']
            f.write("{},".format(date))
            length = len(token)
            i = 1
            for k,v in token.items():
                if(i<length):
                    f.write("{}:{},".format(k,v))
                else:
                    f.write("{}:{}".format(k,v))
                i +=1
            f.write("\n")
generate_countvec_vec_file(data_features)
```

10 Summary

This task uses the natural language preprocessing toolkit to extract useful information regarding the COVID-19 twitter data set. The main outcomes achieved while applying these techniques were:

- **Data extraction**: Reading the Excel file in python and store the data into a pandas data frame.
- **Data frame manipulation**: By using the `pandas` package, removing NaN rows and columns then convert the pandas data frame into a dictionary.
- **Exporting data to specific format**: By using built-in functions like `open` it was possible to export dictionary into `.txt` files with additional formatting and transformations.
- **Tokenization, collocation extraction**: RegexpTokenizer's function is used to tokenize the data and obtain letter only words.
- **Stop word**: Context-dependent and independent stopwords are removed from the data.
- **Stemming**: `Porter stemmer` is used to stem the tokenized words.
- **Bigrams**: For bigram generation, MWETokenizer and PMI measure is used and generate 200 most common bigrams.
- **Vocabulary and sparse vector generation**. A vocabulary covering words from different abstracts was obtained by removing stop words, the top 200 most frequent ones. Filtering based on `nltk`'s frequency distribution function `FreqDist()` and also the built-in functions `set()` and `enumerate()` were used to get the final vocabulary dictionary. Finally, a sparse vector was calculated for every abstract by counting the frequency of vocabulary word occurrences.

In [ ]:
```

```

In [ ]:
```

```