FIT5196 Task 2 in Assessment 1

Student Name: Gayatri Aniruddha

Student ID: 30945305

Date: 29/08/2020

Version: 1.0

Environment: Python 2.7.11 and Jupyter notebook

Libraries used: please include the main libraries you used in your assignment here, e.g.,:

- re (for regular expression, included in Anaconda Python 2.7)
- langid (to classift the language of a tweet)
- nltk 3.2.2 (Natural Language Toolkit, included in Anaconda Python 3.6)
- nltk.collocations (for finding bigrams)
- nltk.tokenize (for tokenization, for both single words and multi-word expressions)
- nltk.stem (for porter stemmer)
- itertools (for iteration methods)
- nltk.collocations (for unigrams and bigrams)
- sklearn.feature_extraction.text (for CounterVectorizationn, in order to generate sparse matrix representation)

1. Introduction

- We have the following dataset :
  - We have an xlsx file which has around 80 sheets.
  - Each sheet has information about COVID-19 tweets.
  - Each sheet has around 2000 tweets.
  - We have the following information about each tweet :
    - tweet_id
    - text
    - created_at
- Tasks performed:
  - Generation of a corpus vocabulary with the same structure as sample_vocab.txt.
  - Calculation and presentation of the top 100 frequent unigrams for a particular day i.e sheet.
  - Calculation and presentation of the top 100 frequent bigrams for a particular day i.e sheet.
  - Generation of a sparse representation as per sample_countVec.txt

2. Methodology

- I have followed the following set of steps for data extraction and data manipultion.

2.1 Importing Libraries

- Here, I have imported the following libraries for our assignment.

- I have even mentioned the need, usage and purpose of each of the libraries.

In [1]:

```
# Library for regular expression
import re

# Library to work with dataframes
import pandas as pd

# Library to filter out the non english tweets
import langid

# Library for Natural Language Processing
import nltk

# Importing the required packages from langid package
# These packages are imported for separating out the non english tweets
# For identifying English tweets
from langid.langid import LanguageIdentifier

# Used as a parameter so that it comes in 0 to 1 range
from langid.langid import model

# For Unigram generation
from nltk.tokenize import RegexpTokenizer

# Multiword Expressions
from nltk.tokenize import MWETokenizer

# For calculating document frequency
from nltk.probability import *

# For iteration methods
import itertools
# In order to join all the tokens later on in the document
from itertools import chain

# For Porter Stemming
from nltk.stem import PorterStemmer

# For Bigram generation
from nltk.collocations import *

# For creating count vectors
from sklearn.feature_extraction.text import CountVectorizer
```

2.1 Identifier Initialisation

In [2]:

```
# Initialising the identifier with normal probability True
identifier = LanguageIdentifier.from_modelstring(model, norm_probs=True)
```

2.2 Excel Sheet Parsing

- Here, I have loaded all the sheets into one dataframe using read excel function

- Here :
  - Key : Sheet Name
  - Value : Each sheet in the excel

In [3]:

```
# Reading our excel table into a dataframe
# sheet_name and header are NONE : In order to merge all the sheets into one dataframe

# Here we open our Excel file by creating a Pandas ExcelFile object named excel_data
#excel_data = pd.read_excel('sample.xlsx', sheet_name = None, header= None)
excel_data = pd.read_excel('30945305.xlsx', sheet_name = None, header= None)

# Key : Sheet Names
# Value : Each sheet in the excel
```

2.3 Generating a list of stopwords

- Here, we create a list of stopwords.
- This list has all the stopwords given in the stopwords text file provided.

In [4]:

```
# For unigrams - Regular Expression Tokenizer
from nltk.tokenize import RegexpTokenizer

# Creating a list to add all the given stop words
stop_words_list = []

# Opening the file in read mode and utf encoding
with open('stopwords_en.txt', 'r', encoding = 'utf8') as stopwords:

    # We iterate through the stopwords text file
    for line in stopwords:

        # Adding the words to the stop_words_list after stripping the '\n'
        stop_words_list.append(line.rstrip('\n'))

# Creating a set of stopwords in order to eliminate duplicate words
stop_words_set = set(stop_words_list)
```

2.4 Initilising our tokenizer

- This regular expression is as per the documentation provided.

In [5]:

```
# As per the document specification
# Here, we are initiliazing our tokenizer
tokenizer = RegexpTokenizer(r"[a-zA-Z]+(?:[-'][a-zA-Z]+)?")
```

2.5 Data Cleaning, Analysis and Tokenization

- I have performed the following **13** data cleaning actions :
  - Removal of columns which have all Null Values.
  - Removal of rows which have all Null Values.
  - Generation of tokens for every tweet for every sheet.
  - Removal of Non English Tweets.
  - Removal of context independent stopwords.
    - We remove the tokens whose length is less than 3.

In [6]:

```
%%time
# Basic Data Cleaning of the Excel Sheets
# Iterating through every df i.e sheet
for key, df in excel_data.items():

    # Step 1 : we are dropping all columns with null values
    df.dropna(how = "all", axis = 1, inplace = True)

    # Step 2 : we are dropping all rows with null values
    df.dropna(how = "all", axis = 0, inplace = True)

    # Step 3 : we are Changing columns names
    df.columns = ['text','id','created_at']

    # Step 4 : we are dropping the header
    df.drop(df.head(1).index, inplace=True)

    # Step 5 : we are removing duplicates
    df.drop_duplicates('id', keep = 'first', inplace=True)

    # Step 6 : Here, Converting the string to lower case
    # To remove redundant tokens
    df['text'] = df['text'].apply(lambda x:str(x).lower())

    # Adding column for langugage
    df['language'] = ""

    # THIS STEP IS TAKING TIME
    # Step 7 : Adding the tweet language
    for index, row in df.iterrows():
        row['language'] = identifier.classify(row['text'])[0]

    # Step 8 : Drop all rows with non english tweets
    df.drop(df.loc[df['language']!='en'].index, inplace=True)

    # Step 9 : Adding column for tokens
    df['tokens'] = ""

    # Step 10 : Adding a column which has tokens generated for that row
    df['tokens'] = df['text'].apply(lambda x: tokenizer.tokenize(x))

    # Step 11 : Removing the context independent stopwords from the tokens
    df['final_tokens'] = df['tokens'].apply(lambda x: [token for token in x if token not in

    # Step 12 : Removing tokens with the length less than 3
    df['final_tokens'] = df['final_tokens'].apply(lambda x: [token for token in x if len(to

    # Step 13 : Dropping the old index and keeping only the new one
    df.reset_index(drop = True, inplace = True)
```

```
Wall time: 10min
```

2.6 Removal of Context Dependent words

- Here, we first create a list of all the tokens using chaining.
- We then create a set of these unique words.
- We then generate a document frequency.

    - This gives us the number of times a given word appears.
- Removal of tokens which appear in lesser than **5 documents.**
    - Here, these are rare tokens.
    - They have less variance.
    - They also have lesser correlation with the topic!
- Removing tokens which appear in more than **60 documents.**
    - Here, these tokens give repetitive information.
    - They have high correlation!

In [7]:

```
# Removing Context Dependent words

# For creating chain of tokens
from __future__ import division
from itertools import chain

# List of lists, chaining everything
# List of every token - tokenized all the sheets
# And then add all the tokens to the same list
words = list(chain.from_iterable([set(list(chain(*df['final_tokens']))) for key, df in exce

# Creating unique set of tokens
vocab = set(words)

# Creating document frequency
# Word : Number of documents in which the word appears
fd = FreqDist(words)
#fd.most_common(5)

# Removing rare tokens
# Here : Each document represents a day

# Words appearing in lesser then 5 documents i.e 5 days - 1 : for trials
# Less Correlation!
less_than_5_days = list(filter(lambda x:fd[x] < 1, fd))

# Words appearing in greater then 60 documents i.e 60 days - 6 : for trials
# Over information!
greater_than_60_days = list(filter(lambda x:fd[x] > 6, fd))
```

2.7 Porter Stemming, Unigram and Bigram generation

- Here, we use Porter Stemming to further compress our data.
- I have defined a bigram_generator function :
    - This is as per the content taken from tutorial 5.
    - This is to generate the top 100 bigrams.
- Finally, I have created an unigrams and bigrams dictionary.
    - In Unigrams, individual words are considered.
    - In Bigrams, group of two words are considered.

In [8]:

```
# Removing Context Dependent words to get the final set of tokens
%%time

# Initilising our Porter Stemmer
stem = PorterStemmer()

# Unigrams Dictionary
uni_grams = dict()

# Bigrams Dictionary
bi_grams = dict()

# Bigram Generator
# Function to generate top 200 bigrams using PMI Measure - Taken from Tutorial 5.
def bigrams_generator(l):
    bigram_measures = nltk.collocations.BigramAssocMeasures()
    bigram_finder = nltk.collocations.BigramCollocationFinder.from_words(l)
    bigram_finder.apply_freq_filter(20)
    #bigram_finder.apply_word_filter(lambda w: len(w) < 3)# or w.lower() in ignored_words)
    top_100_bigrams = bigram_finder.nbest(bigram_measures.pmi, 100) # Top-100 bigrams
    return top_100_bigrams

# Iterating through every df i.e sheet
for key, df in excel_data.items():

    # Removing words appearing in greater then 60 sheets i.e 60 days
    df['final_tokens_2'] = df['final_tokens'].apply(lambda x: [token for token in x if toke

    # Removing words appearing in lesser then 5 sheets i.e 5 days
    df['final_tokens_2'] = df['final_tokens'].apply(lambda x: [token for token in x if toke

    # Creating a new column for Porter Stemming
    # Porter stemming of the final set of tokens
    df['stem_tokens'] = df['final_tokens_2'].apply(lambda x:[stem.stem(token) for token in

    # Creating Bigrams
    top_100_bigrams = bigrams_generator(list(chain(*df['tokens'].tolist())))
    mwetokenizer = MWETokenizer(top_100_bigrams)

    df['bigrams'] = df['tokens'].apply(lambda x: mwetokenizer.tokenize(x))
    df['bigrams'] = df['bigrams'].apply(lambda x: [token for token in x if '_' in token])

    # Create Unigrams dictionary
    uni_grams[key] = sorted(list(dict(FreqDist(list(chain(*df['stem_tokens'].tolist())))).i
    # Create Bigrams dictionary
    bi_grams[key] = sorted(list(dict(FreqDist(list(chain(*df['bigrams'].tolist())))).items(

    # Joining all the tokens together for generating the vocab text
    df['joint_tokens'] = df['stem_tokens'].apply(lambda x: " ".join(x))
```

```
UsageError: Line magic function `%%time` not found.
```

In [ ]:

```
#uni_grams
#bi_grams
```

In [ ]:

```
# Here, we change the structure of our bigrams dictionary
# This as per the sample output provided
# We are supposed to remove the _ from the bi-grams generated

# Creating a new dictionary to store the bigrams as per this format
formatted_bigrams = {}

# Iterating through the bigrams
for key, value in bi_grams.items():

    # Creating a temporary list of words to store the separated bigrams
    temp_words_list = []

    # Iterating through the values
    for word in value:

        # Remvoving the _ from the bigram words
        separate_words = word[0].split("_")

        # Adding the words with the count back to the temporary list
        temp_words_list.append(((separate_words[0], separate_words[1]),word[1]))

    # Finally adding the changed formatted list back to the formatted bigrams
    formatted_bigrams[key] = temp_words_list
```

2.8 Output File 1 : Unigram and Bigram File

- Here, Each line in the txt file contains the top 100 most frequent uni/bigrams of one day of the tweet data.
- Format used : sample_100uni.txt and sample_100bi.txt
- Naming used : student_number_100uni.txt and student_number_100bi.txt

In [ ]:

```
# Generating the unigrams as per the format provided
# <student_number>_100uni.txt
with open('30945305_100uni.txt','w+', encoding = 'utf-8') as unigrams_file:
    for key, value in uni_grams.items():
        unigrams_file.write('%s %s\n' % (key, value))

# Generating the bigrams as per the format provided
# <student_number>_100bi.txt
with open('30945305_100bi.txt','w+', encoding='utf-8') as bigrams_file:
    for key, value in formatted_bigrams.items():
        bigrams_file.write('%s %s\n' % (key, value))
```

2.9 Corpus Vocabulary and Output File 2 : Vocab Text File

- Corpus Vocabulary :
  - Here, using CountVectorizer, we generate our vocabulary list.

    - We then assign a unique number to each word as per the sample_vocab.txt.
- Output File :
    - It contains the bigrams and unigrams tokens.
    - Format : sample_vocab.txt
    - Naming : student_number_vocab.txt

In [ ]:

```
# For creating count vectors
# Resource : Study Material Week 5

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(analyzer = "word")

data_features = vectorizer.fit_transform([" ".join(df['joint_tokens'].tolist()) for key, df
#print (data_features.shape)

# Creating a vocab dictionary
vocab = vectorizer.get_feature_names()

# Creating a dictionary to store all the vocab
vocab_dict = dict()

# For assigning an unique number to every word
# This is for generating the sample vocab text file
num = 0
for each_word in vocab:
    vocab_dict[each_word] = num
    num = num + 1

# Generate the corpus vocabulary with the same structure as sample_vocab.txt
# <student_number>_vocab.txt
with open('30945305_vocab.txt','w+', encoding ='utf-8') as vocab_file:

    for key,value in vocab_dict.items():

        vocab_file.write('%s:%s\n'%(key,vocab_dict[key]))
```

2.10 Sparse Representation and Output File 3 : Count Vector Text File

- Here, we Generate the sparse representation of the excel file
- Format : Date, unique_number for the word : Number of times the word occurs in that document
- **Methodology followed** :
    - We separate out all the dates which are sheet names.
    - We create a word count dictionary to store the word count.
    - Using the values provided in the data features,
        - We zip the vocab list and the word count list.
        - This generated a list of tuples according to every index.
    - We finally have a dictionary in the following format :
    - Key :Date
    - Value :Dictionary with { Key : word, Value : Number of times the word occurs in that document }
- We then **Clean and Re-format and Re-structure** this dictionary.
- Final Format :
- Key : Date
- Value : {unique_number for the word : Number of times the word occurs in that document}

In [ ]:

```
# Generate the sparse representation of the excel file according to sample
# Format : Date, unique_number for the word : Number of times the word occurs in that docum
%%time

# We first store our results in a dictionary
# Key : Date, # Value : {word : Number of times the word occurs in that document}
count_vector = dict()

# Changing the count_vector according to our format : formatted version
# Key : Date, # Value : {unique_number for the word : Number of times the word occurs in th
final_count_vector = dict()

# For easier iteration
array_of_data_features = data_features.toarray()

# Getting the dates and adding all the dates to generate the matrix
dates = list(excel_data.keys())

# In order to get the word count for every word in a given sheet
for each_row in array_of_data_features:

    # Dictionary for word count
    word_count = dict()

    # Popping out the first element from the list
    date = dates.pop(0)

    # Adding the word count to the corresponding word
    # Zip : We get a list of tuples corresponding to every index
    for word_i, count_i in zip(vocab, each_row):
        if count_i > 0:
            word_count[word_i] = count_i

    # Finally appending the word_count dictionary corresponding to every date
    count_vector[date] = word_count
#count_vector

# Iterating through our count_vector
for key,value in count_vector.items():

    # keys correspond to words
    words = value.keys()

    # Dictionary for the final count of occurences of all the words
    final_count_dict = dict()

    for each_word in words:
        if each_word in vocab_dict:
            final_count_dict[vocab_dict[each_word]] = value[each_word]

    final_count_vector[key] = final_count_dict

#final_count_vector
```

In [ ]:

```
# Generating the sparse matrix
# <student_number>_countVec.txt

with open('30945305_countVec.txt','w+') as count_file:

    for key, value in final_count_vector.items():

        # As per the given format, we do not need these characters in the final representat
        count_file.write('%s,%s\n' % (key,str(value).strip('{}')))
```

3. Conclusion

- Thus, in summary, we have successfully completed the following :
  - We have parsed the data sheets and performed the necessary basic data cleaning and tokenisation.
  - We have kept only the english tweets.
  - We have removed context independent and context dependent words.
  - We have further compressed the tokens using Porter Stemmer.
  - We have removed words that occur rarely as they have less variance.
    - These words do not really give much information to us.
  - We have even removed the words which occur way too much!
    - These words give redundant information.
  - We have generated a sample vocabulary of our data.
  - We have even management to get a document frequency of our tokens.
  - In addition, we have been able to generate unigrams and bigrams from our words.
  - Finally, we have been able to successfully generate our sparse matrix.