FIT5196 Assessment 1

Student Name: DISHI JAIN

Student ID: 30759307

Libraries used:

- langid (for verifying the language of a string)
- pandas (for creating dataframe and using pandas features on dataframe)
- nltk (Natural Language Toolkit, included in Anaconda Python 3.6)
- nltk.collocations (for finding bigrams)
- nltk.tokenize (for tokenization, included in Anaconda Python 3.6)
- nltk.probability (for using FreqDist functionality)
- nltk.stem (for stemming words using PorterStemmer)
- nltk.util (for using ngrams)
- nltk.metrics (for using BigramAssocMeasures in collocations)
- itertools (for using chain functionality)

1. Introduction

This assignment focuses on writing a Python code to preprocess a set of tweets and convert them into numerical representations (which are suitable for input into recommender-systems/ information-retrieval algorithms). This task involves extracting the required data i.e. the tweets daywise where each day is represented by the sheets in the excel file. For each day after extracting the english tweet text certain steps involving tokenization, stopwords removal, removal of tokens with length less than 3, removal of context dependent tokens and rare tokens are involved. Then unigrams, bigrams, vocab and vector files need to be generated.

More details of this task will be seen in the following sections.

2. Libraries importing

In [1]:
```
#imporing the required libraries
import langid
import pandas as pd
import nltk
from nltk.util import ngrams
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from nltk.probability import FreqDist
from nltk.stem import PorterStemmer
from nltk.tokenize import RegexpTokenizer
from itertools import chain

stem = PorterStemmer()
bigram_measures = nltk.collocations.BigramAssocMeasures()
```

In [2]:
```
from __future__ import division
```

3. Loading and cleaning the data

The implementation is as follows -

- In this task the xml file 30759307 is loaded into an excel object called excel using pandas and ExcelFile function. This excel object then iterates through the sheets of the excel file using sheet_names functionality.
- For each sheet now we perform ceratin operations for cleaning of the file. First the columns and rows with NA as their values in each cell i.e. in all the cells are removed as they are of no use to us. This is done by using dropna and arguments as axis and how.
- Next I have resetted the index of the dataframe using reset_index functionality. After doing this the original indexes are obtained as a separate column. Hence to remove them I have used drop function and given the column name 'index' to it along with the axis. Inplace = True is used to make the changes permanent to the dataframe.
- Next I check if the term 'text' is present in the columns of the dataframe or not. If it is not present in the columns then I create a header variable corresponding to the first row of the dataframe. This will contain the values text,id and date. Then I create a new dataframe with this header as the column names and the rows as the previous dataframe from 2nd row. If however the term 'text' is present already in the column, then I don't create any new dataframe.
- Next I iterate through the column text and check if the values are of string type. If not a string I have removed the row from the dataframe.
- Next I iterate through the text column of the dataframe to check if the text is in english or not. If the text is in english then I use a dictionary called d and give the date as its key and the remaining text as its value. The text part will get appended. Hence a final dictionary d is created with date as its key and tweet's text as it's values where each text is separated by a newline character.

In [4]:
```
headers = ''
excel = pd.ExcelFile('30759307.xlsx')

d = {}

#transversing through sheets in the excel object
for x in excel.sheet_names:
    df = excel.parse(x)

    #dropping the rows with NA as all its vales
    df = df.dropna(0, how = 'all')
    #dropping the columns with NA as all its vales
    df = df.dropna(1, how = 'all')

    #resetting the index
    df.reset_index(inplace=True)

    #removing the generated column 'index'
    df.drop('index',axis = 1,inplace=True)

    #checking for term 'text' in columns
    if 'text' not in df.columns:

        #taking the first row as headers
        headers = df.iloc[0]
        #creating a new dataframe
        new_df  = pd.DataFrame(df.values[1:], columns=headers)

        #iterating through the text column
        for i in new_df.text.values:
            #if type of text is not string then removing it
            if type(i) != str:
                index_names = new_df[ new_df['text'] == i ].index
                new_df.drop(index_names, inplace = True)

        #iterating through the text column
        for i in new_df.text.values:
            #check for language of text string
            if langid.classify(i)[0] == 'en':
                if type(new_df.text.values) != str:
                    if x in d:

                        #appending the string to dictionary value
                        d[x] = d[x] + '\n' + str(i)
                    else:
                        d[x] = str(i)
                else:
                    if x in d:
                        d[x] = d[x] + '\n' + i
                    else:
                        d[x] = i

    else:
        #iterating through the text column
        for i in df.text.values:
            #if type of text is not string then removing it
            if type(i) != str:
                print(i)
                index_names = df[ df['text'] == i ].index
                print(index_names)
                df.drop(index_names, inplace = True)

        #iterating through the text column
        for i in df.text.values:

            #check for language of text string
            if langid.classify(i)[0] == 'en':
                if type(new_df.text.values) != str:
                    if x in d:

                        #appending the string to dictionary value
                        d[x] = d[x] + '\n' + str(i)
                    else:
                        d[x] = str(i)
                else:
                    if x in d:
                        d[x] = d[x] + '\n' + i
                    else:
                        d[x] = i
```

After this step we get a dictionary that has key as date and value as a string containing all the tweets for that date.

4. Creating tokens

In this step we create tokens using the RegularExpression given to us. We iterate it through each value in the above created dictionary and create tokens of it after converting it into lowercase. This data is then stored in another dictionary called tokens_dicti which has date as key and list of tokens as values

In [5]:
```
#creating dictionary for storing tokens date wise into a list
tokens_dicti = {}

#generating a RegexpTokenizer object with the RE as the one given to us
tokenizer = RegexpTokenizer(r"[a-zA-Z]+(?:[-'][a-zA-Z]+)?")

#iterating inside the dictionary created before to create tokens of each value af
for key,val in d.items():
    tokens_dicti[key] = tokenizer.tokenize(d[key].lower())
```

5. Generating Bigrams

In this process bigrams are generated from the tokens given to us date wise. We get the first 200 bigrams using the nbest feature. Also we use the pmi measure collocations while creating the bigrams to get the most meaningful bigrams. Then I have created a dictionary which will contain the dates as keys and bigrams generated into a list.

Making use of this final dictionary then the file sample_100bi.txt is created. A string is then generated that will be in the same format as expected, where each row represents a document and then each document has a list of bigrams along with its frequency/count in the document. Only the first 100 bigrams are taken for this task. Finally the string is written to the output file.

In [6]:
```python
#100bi.txt

bi_final = {}

#iterating in the dictionary created above
for i in tokens_dicti:

    #Creating bigrams from values of tokens_dicti
    finder = nltk.collocations.BigramCollocationFinder.from_words(tokens_dicti[i]

    #getting top 200 bigrams
    bigram_list = finder.nbest(bigram_measures.pmi,200)
    #selecting only the top 100 bigrams and inserting into value of bi_final
    bi_final[i] = (sorted(finder.ngram_fd.items(), key=lambda t: (-t[1], t[0]))[:


final_string_bi = ''

#iterating in bi_final dictionary to create the string in the required format
for i in bi_final:
    final_string_bi = final_string_bi + str(i) + ':' +  str(bi_final[i]) + '\n'

#writing the string into the file sample_100bi.txt
f2 = open("30759307_100bi.txt" , 'w', encoding = 'utf8')
f2.write(final_string_bi)
f2.close()
```

After this step we have successfully generated a bigram text file

6. Generating Unigrams

Implementation is as follows -

- In this below step I have first read the stopword file given to us and generated a list stopwords_list containing all the stopwords. I have then created a set using this stopwords_list to get unique stopwords.
- Then I have removed all the stopwords from the tokens_dicti dictionary using list comprehension and stored the final terms in stopwords_removal_final where key is date and value is a list of tokens after removal of stopwords
- Then I have removed the tokens whose lenth is less than 3. For this I have created another dictionary called greater_than_3 which contains key as date and values are appended from previous step and after removing tokens with lenth less than 3.

- Then I have handled rare tokens removal and context dependent tokens removal. For this I have used the dictionary created in above step and have given it as an argument to chain.from_iterable function. I have also used list comprehension inside it and FreqDist which calculates the document frequency of the tokens. This is stored in document_freq. Next I have iterated inside this to get keys(tokens) and values(document frequency). If the value is greater than 60 or less than 5 then a final_document_freq dictionary is created which contains the valid tokens only.
- Then I have created another dictionary called rare_removal which will contain the dates as keys and tokens as values. These tokens are the ones which have undergone - stopwords removal, length less than 3 removal, rare tokens removal and context dependent removal.
- After this I have stemmed the words using PortStemmer() stem. Finally the dictionary stemmed_words is created containing dates as keys and stemmed versions of valid tokens.
- Now to get the frequency of each token date wise I have used FreqDist in stemmed_words and used most_common functionality to get top 100 stemmed tokens date wise. This is finally stored in uni_final
- Now using the dictionary uni_final I have written the data as the required format into the string final_string_uni. This string is then written to the file sample_100uni.txt

In [7]:
```
# 100 uni.txt

#FINAL DISHI JAIN


#stopwards removal
with open('stopwords_en.txt' , 'r') as f:
    stopwords_list = f.read().splitlines()


stopwords_removal_final = {}
#creating set of stopwords
stopwords_set = set(stopwords_list)
#iterating inside tokens_dicti to remove stopwords
for i in tokens_dicti:
    stopwords_removal_final[i] = [w for w in tokens_dicti[i] if w not in stopworc


#length < 3 removal
greater_than_3 = {}
for i in stopwords_removal_final:
    for j in stopwords_removal_final[i]:
        #check for length of the tokens
        if len(j) >= 3:
            if i in greater_than_3:
                greater_than_3[i].append(j)
            else:
                greater_than_3[i] = [j]




#context dependent removal > 60 and rare tokens removal < 5
count_list = list(chain.from_iterable([set(value) for value in greater_than_3.val

#document frequency of each token
document_freq = FreqDist(count_list)
final_document_freq = document_freq.copy()
for key,val in document_freq.items():
    if val > 60 or val < 5:
        #storing valid keys only after removal of rare tokens and context depende
        final_document_freq.pop(key)



rare_removal = {}
for i,j in greater_than_3.items():
        #creating date wise valid tokens
        rare_removal[i] = [k for k in j if k in final_document_freq.keys()]


#stem the words
stemmed_words = {}
for i in rare_removal:
    for j in rare_removal[i]:
        if i in stemmed_words:
            #stemming of the tokens date wise
            stemmed_words[i].append(stem.stem(j))
        else:
             stemmed_words[i] = [stem.stem(j)]


#count of tokens date wise
uni_final = {}
for i in stemmed_words:
    #FreqDist to get count of each token
    word_fd_uni = nltk.FreqDist(stemmed_words[i])
    #storing date wise top 100 unigrams only alongwith its count
    uni_final[i] = word_fd_uni.most_common(100)



final_string_uni = ''
#iterating inside uni_final
for i in uni_final:
    #generating final_string_uni in the required format
    final_string_uni = final_string_uni + str(i) + ':' +  str(uni_final[i]) + '\r


#writing the final_string_uni to the file sample_100uni.txt
f3 = open("30759307_100uni.txt" , 'w', encoding = 'utf8')
f3.write(final_string_uni)
f3.close()
```

After this step we have created the file containing top 100 unigrams for each date

7. Generating Vocab

Implementation is as follows -

- In this step I have generated bigrams from the entire original tokens_dicti values hence the entire set of tokens(Not date wise). I have stored this in finder_new and then extracted the top 200 of it to store in the bigram_list. Hence bigram_list contains the top 200 bigrams extacted from all the tokens in the entire data.
- Then I have created a list called vocab_final_list containing the valid bigrams with '_' between them.
- Next I have used the stemmed_words dictionary created above to get just the valid unigrams. This dictionary contains valid unigrams and their dates hence I have appended just the unigrams into the list vocab_final_list.
- Next I have created a set of this list to remove any repetitions using the set()
- Next I have sorted this to get write_list as the final vocab using sorted().
- Next I have created a string final_string_vocab to get the ouput in the required format. I have used range and len function to get the indexes/index of each token in vocab.
- Finally I have written this string into the file sample_vocab.txt

In [8]:
```
#VOCAB.txt


# FINALLLLL DISHIIIII JAINNNN

vocab_final_list = []
vocab_final = {}
all_words = list(chain.from_iterable(tokens_dicti.values()))
finder_new = nltk.collocations.BigramCollocationFinder.from_words(all_words)
bigram_list = finder_new.nbest(bigram_measures.pmi,200)
for x in bigram_list:
    vocab_final_list.append(str(x[0]) + '_' + str(x[1]))



for i,j in stemmed_words.items():
    for x in j:
        vocab_final_list.append(x)

vocab_final_set = set(vocab_final_list)
write_list = sorted(vocab_final_set)


final_string_vocab = ''
for i in range(len(write_list)):
        final_string_vocab = final_string_vocab + str(write_list[i]) + ':' + str(



f4 = open("30759307_vocab.txt" , 'w', encoding = 'utf8')
f4.write(final_string_vocab)
f4.close()
```

After this step I have generated a file called sample_vocab.txt containing the vocab

8. Generating CountVector

In this step I have to create a CountVector text file containing the date wise count of each token where the token is seen as its index value from the vocab file. Implementation is done as follows -

- First I have created date wise bigrams from the original tokens_dicti dictionary. I have created all the possible bigrams in this and then I have sorted them. This gives us a dictionary d1 which contains date as key and a list of tuples containing bigrams and its count.
- Next I have created a copy of d1 dictionary and created d2 dictionary to get only those bigrame date wise which are valid and present in the list bigram_list. I have used list comprehension to do so.
- Next I have appended the valid bigrams date wise into the dictionary stemmed_words. Hence stemmed_words will finally contain date as key and a list of valid tokens(uni and bi grams) as its value.

- Then I have created a file sample_countVec.txt to get the output in the desired format. For this I have first created a list of the final vocab's file. Then a new dictionary vocab_vector_dic is created that contains each vocab token as key and a counter as its value.
- Next I have iterted inside stemmed_words dictionary to get the date wise index/counter of each token and its count. Hence d_idx is a list containing index of tokens as many times as the token was present. Using FreqDist will then give key as the token index and value as the frequency/count of that token. This is then stored in string_vector in the required format.
- The string_vector is then written to the file sample_countVec.txt to get the final output.

In [9]:
```
## COUNT VECTOR

d1 = {}

#iterating inside original tokens_dicti
for i in tokens_dicti:
    finder = nltk.collocations.BigramCollocationFinder.from_words(tokens_dicti[i]
    #creating bigrams from all tokens date wise
    nope = bigram_measures.pmi
    d1[i] = (sorted(finder.ngram_fd.items(), key=lambda t: (-t[1], t[0])))



d2 = d1.copy()

#getting only the valid bigrams after comparing from the list bigram_list
for i,j in d1.items():
    d2[i] = [w for w in d1[i] if w[0] in bigram_list]


for i,j in stemmed_words.items():
    for y,z in d2.items():
        if i == y:
            for x in z:
                #appending valid bigrams to stemmed_words
                j.append(str(x[0][0]) + '_' + str(x[0][1]))


#writing the data to the file sample_countVec.txt
out_file = open("./30759307_countVec.txt" , 'w')
#creating the vocab_vector as a list of vocab
vocab_vector = list(write_list)
vocab_vector_dic = {}
i = 0
for w in vocab_vector:
    #indexing each token in vocab starting from 0
    vocab_vector_dic[w] = i
    i = i + 1

string_vector = ''
#iterating in stemmed_words dictionary
for c,d in stemmed_words.items():
    #adding the date into the string
    string_vector = string_vector + str(c)
    #creating d_idx to get occurences of each token index
    d_idx = [vocab_vector_dic[w] for w in d]
    #using freqdist to get count of each token index
    for k,v in FreqDist(d_idx).items():
        string_vector = string_vector + ',' + str(k) + ':' + str(v)
    string_vector = string_vector + '\n'

#writing the string_vector to the file
out_file.write(string_vector)
out_file.close
```

Out[9]:
```
<function TextIOWrapper.close()>
```

In the above step I have created a file sample_countVec.txt in the required format as needed.

9. Summary

In this assessment I have created 4 files. For creation of sample_100bi.txt I have created bigrams date wise using collocations and pmi measure. The top 100 bigrams are taken into consideration for each date. For creating the sample_100uni.txt file I have first removed the stopwords from the original tokens_dicti dictionary date wise. Then I have removed the tokens with length less than 3. Then I have removed the rare tokens and context dependent tokens. Then I have stemmed the tokens and used FreqDist to count occurence of each token date wise. For creation of sample_vocab.txt file I have used the already generated valid unigrams and generated new bigrams generated from the entire list of tokens. I have then combined these two into a final vocab list where they are arranged alphabetically and contains index/counter. For creation of the file sample_countVec.txt I have found the valid bigrams and unigrams date wise and combined them together. Then used the vocab file to get index of each token and counted its occurence using FreqDist. This way the final file is also generated