

FIT5196 Task 2 in Assessment 1

Student Name: Sarthak Sareen

Student ID: 30761182 

Date: 13/9/2020

- pandas (for dataframe, included in Anaconda Python 2.7)
- langid (to check string language, included in Anaconda Python 2.7)
- 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)

1) Introduction

In the task we are analyzing textual data, i.e., converting the extracted data into a proper format. we will preprocess a set

of tweets and convert them into numerical representations (which are suitable for input into recommender-systems/ information-

retrieval algorithms). The data-set that we provide contains 80+ days of COVID-19 related tweets (from late March to

mid July 2020). We have extract and transform the information of the excel file performing the following task:

1) making a vocab .txt

2) For each day calculate the top 100 frequent unigram and top-100 frequent bigrams and saving it into a file 100bi.txt and

100.uni.txt

3) Generate the sparse representation (i.e., doc-term matrix) of the excel file according to the structure of the sample_countVec.txt

1. Import libraries

```
In [ ]: #importing the librarbies for the preprocess of the text.
from __future__ import division
import langid
import pandas as pd
import nltk
from nltk.tokenize import RegexpTokenizer
from itertools import chain
from nltk.stem import PorterStemmer
from nltk.probability import FreqDist
stem = PorterStemmer()
```

```
In [ ]: #reading the file
excel = pd.ExcelFile('30761182.xlsx')
#making empty dictionary
di = {}
#looping in the range of the length of the excel sheets
for i in range(len(excel.sheet_names)):
    #making a dataframe of the first excel sheet
    df = excel.parse(i)
    #dropping the NA's from the dataframe
    df = df.dropna(how='all',axis=1)
    df = df.dropna(how='all',axis=0)
    #check the 1st column is text or not
    if df.columns[0] != 'text':
        ##renaming the column from row 1
        df.rename(columns=df.iloc[0], inplace = True)
        #dropping the 1 st row
        df.drop(df.index[0], inplace = True)
    #looping in the dataframe text column values
    for y in df.text.values:
        try:
            #checking the text of the dataframe text column is english
            if langid.classify(y)[0]=='en':
                #check if key in the dictionary or not
                if excel.sheet_names[i] in di:
                    #appending the values
                    di[excel.sheet_names[i]] = di[excel.sheet_names[i]] + '\r\n' + y
                else:
                    #making new keys with values
                    di[excel.sheet_names[i]] = y.lower()
            # text of the dataframe text column is not english
        except:
            pass
```

- 1) In the above cell the excel file is being loaded with 80+ sheets which is being saved in the excel named variable.
- 2) Making a new empty dictionary di.
- 3) Looping till the range of the length of the excel.sheet_names i.e 0,1,2...

i) making a dataframe of the excel sheet number

ii) dropping the NA's from all the columns

iii) dropping the Na's from all the rows

iv) checking if the dataframe columns is not "text" then

--rename the columns of the dataframe with the first row of the dataframe and making it inplace to true that is saving the dataframe changes.

--then dropping the first row as it will duplicate as we have made the first row as columns.

-- looping in the dataframe text column values

*trying if the text column value is in english


A) checking if the key is present in the dictionary or not

-- adding the elements in the same key and lowering the case of the text also

B) if key is not present then making a new key as the excel sheet date as key and value of its text column

* except if the text is not en using langid library then pass. It will not add that row into the dictionary.

4) getting the desired dictionary with keys as dates of the excel sheets and values as the data of the text column.

```
In [ ]:  #making new dictionary
new_dict = {}
#initialize the tokenizer
tokenizer = RegexpTokenizer(r"[a-zA-Z]+(?:['-][a-zA-Z]+)?")
#making tokens and making a new dictionary
for i,k in di.items():
    new_dict[i] = tokenizer.tokenize(di[i])
```

In the above we are creating tokens of the dictionary from the regex we have given and making a new dictionary with the keys as the date and values as tokens

```

In [ ]: #reading the stopwords and saving it in a list called stopped
with open('stopwords_en.txt') as infile:
    stopped = infile.readlines()

#stripping the list
stopped = set([line.rstrip('\n') for line in stopped])

#making an empty dictionary
uni_dict_new = {}

#looping the new_dict and saving into new dic where the values are not present
for i,j in new_dict.items():
    uni_dict_new[i] = [w for w in j if w not in stopped and len(w) >= 3]

#making bag of bag of tokens
freq_list = FreqDist(list(chain.from_iterable([set(value) for value in uni_dict_new.items()])))

#making new dictionary
new_freq_dic = {}

#looping in the freq_list dic if the value is greater than 60 and smaller than 5
for i,j in freq_list.items():
    if j > 60 or j < 5 :
        pass
    else:
        new_freq_dic[i] = j

#making new dict empty stemmed
uni_dict_stem = {}

#making new dict empty stemmed and freqdist
uni_dict_stem_final = {}

#looping in the dic where stopped words and length >= are removed from the dictionary
for i,j in uni_dict_new.items():
    #stemming the values
    uni_dict_stem[i] = [stem.stem(x) for x in j if x in new_freq_dic.keys()]

    #looping in the dic where the values are stemmed
for i,j in uni_dict_stem.items():
    #using freq dist to calculate count
    uni_dict_stem_final[i] = FreqDist(j).most_common(100)

```

In the Above cell :

- 1) Reading the stopped words file
- 2) splitting the list to make list with only stop words
- 3) Looping in the dictionary of the data that we have created and adding the key as the date and the value which are not in stopwords list and the length of the value is greater than 3.

4) Making freq_list that is the dictionary with words and the count of that word using the function freq dist. the argument of the freq dist I have given as the values of the dict i.e dictionary removed from stopwords and the len of the values of the dictionary is greater than 3.

5) now removing the words which have count > 60 and <5 and making a new dict named new_freq_dict which has keys as words and value as the count of that word.

6) Now creating two dictionary with the stemmed words of the values of the dictionary i.e unit_dict_stem and as dictionary with stem words and count of that words using Freqdist i.e uni_dict_stem_final.

7) We get a dictionary which have date as the key and values which have the words and the count of that word.

```
In [ ]: ▶ #initilizing the string
string_uni = ''
#looping in final dic of the uni grams
for i,j in uni_dict_stem_final.items():
    string_uni = string_uni + str(i) + ":" + str(j)
    #string_uni = string_uni + str(x)
    string_uni = string_uni + "\n"
```

In the above cell making a string which is according to the output using the dictionary of unigram that I have made above i.e uni_dict_stem_final

```
In [ ]: ▶ #wring the string into the file
with open("30761182_100uni.txt", "w", encoding='utf-8') as text_file:
    text_file.write(string_uni)
```

Wrting the string into the output file

```
In [ ]: ▶ ##bigrams
bgm = nltk.collocations.BigramAssocMeasures()
#new dictionary for bigrams empty
bi_dict = {}
#looping in the new_dict
for i,j in new_dict.items():
    #passing the values of the dictionary to the bigram collaction finder
    finder = nltk.collocations.BigramCollocationFinder.from_words(j)
    #taing 200 bigrams
    freq = finder.nbest(bgm.pmi,200)
    lst = (sorted(finder.ngram_fd.items(), key=lambda t: (-t[1], t[0]))[:100])
    #making a new dict of the bigrams
    bi_dict[i] = lst
```

In the above cell:

1) making the object of nltk collactions bigram measures

2) making new bi_dict emotv for the bigrams

--, making new bi_dict entry for the bigrams

3) looping into the main dictionary which have key as the date and the tokens as the values

--passing the value of the key into the nltk collocations bigram collaction finder

-- taking top 200 bigrams

--making a new dictionary with key as the date and key include the bigram adn the count of that bigram

4) The dictionary which is bi_dict is the desired dictinoary which is used to write int the file.

```
In [ ]:  ▶ #empty string
          string_bi = ''
          #string concate looping in the bi_dict
          for i,j in bi_dict.items():
              string_bi = string_bi + str(i) + ":" + str(j) + "\n"
```

In the above cell making a string which is according to the output using the dictionary of bigram that I have made above i.e bi_dict

```
In [ ]:  ▶ with open("30761182_100bi.txt", "w", encoding='utf-8') as text_file:
          text_file.write(string_bi)
```

Wrting the string into the output file

```

In [ ]: ##VOCAB
#empty list of the vocab
bi_list_vocab1 = []

#making bag of bag from the values of the dictionary
words = list(chain.from_iterable(new_dict.values()))

#passing the bag of bag to the bigram collocations
finder = nltk.collocations.BigramCollocationFinder.from_words(words)

#taking top 200 bigrams
freq = finder.nbest(bgm.pmi,200)

#empty string
s = ''
#looping in the top 200 bigrams
for i in freq:
    #adding in the string with "_"
    s = s + i[0] + "_" + i[1] + "\n"

#splitting the string
list_final = s.split('\n')
list_final = list_final[:-1]

#looping in the unigram list which is stemmed
for i,j in uni_dict_stem.items():
    for x in j:
        #appending bigrams in the list
        list_final.append(x)
#making set of the list of vocab
last = set(list_final)
#sorting the list
sorted_list_final = sorted(last)

```

In the above cell:

- 1) making a list of vocab empty
- 2) making a bag of bags of the words with the argument of the values of the dictionary in which there are all the tokens of the text without removing anything.
- 3) taking the top 200 bigrams to freq
- 4) Now looping into the freq and adding "_" into the both the words of the bigrams because we have to represent the bigram like this.
- 5) splitting the string and making it into list
- 6) slicing the list to remove "\n".
- 7) Now looping into the dictionary of unigram where the values are stemmed that is being created where we have created unigrams

and appending it into the list of bigrams.

8) Now making the list a set to remove the repeated words

9) sorting the list

10) the list is the list of vocabs.

```
In [ ]: ▶ count_vocab = 0
          #initilise empty string
          string_vocab=''
          #looping in the list of vocabs
          for i in sorted_list_final:
              #string concate
              string_vocab = string_vocab + str(i) + ":" + str(count_vocab)+ "\n"
              #index of the word
              count_vocab=count_vocab+1
```

In the above cell:

iterating into the list of the vocabs and concating the string with the index of the number which is being calculated by the variable count_vocab

```
In [ ]: ▶ with open("30761182_vocab.txt", "w", encoding='utf-8') as text_file:
          text_file.write(string_vocab)
```

Wrting the string into the output file


```

In [ ]: ▶ ##counter vector
#making empty counter_vector dic
counter_bi_dict = {}

#looping in the original dic where nothing is being removed
for i,j in new_dict.items():
    #giving the values of the list to bigram collactions
    finder = nltk.collocations.BigramCollocationFinder.from_words(j)
    frequency = bgm.pmi
    lst = (sorted(finder.ngram_fd.items(), key=lambda t: (-t[1], t[0])))
    #making new dict with key as date and values as bigrams
    counter_bi_dict[i] = lst

#new empty final_counter_bi_dic
final_counter_bi_dic ={}

#looping in the freq which contains bigrams which are in the vocabs
for i in freq:
    #looping in the above created dic
    for k,l in counter_bi_dict.items():
        #iterating in the values of the dict
        for x in l:
            #check bigram in freq == counter_bi_dict
            if i == x[0]:
                #check if key present
                if k in final_counter_bi_dic:
                    #adding into the new final dict append in the present key
                    final_counter_bi_dic[k].append(i)
                else:
                    #adding into the new key and values
                    final_counter_bi_dic[k] = [i]

#adding the bigrams in the uni_dict_stem dictionary
for i,j in final_counter_bi_dic.items():
    for l,m in uni_dict_stem.items():
        if i==l:
            for x in j:
                uni_dict_stem[l].append(str(x[0]) + "_" + str(x[1]))

#making counter vector
vocab_dict = {}
vector_dict ={}
last_dict={}
#making dict of vocab having key as words and value as index
i=0
for w in sorted_list_final:
    vocab_dict[w] = i
    i = i+1

stem_dict = {}
string_counter_final=''

#looping in uni_dict_stem
for x,y in uni_dict_stem.items():
    #occurence of the tokens id
    last_val = [vocab_dict[k] for k in y]

```

```

#string concate
string_counter_final = string_counter_final + str(x) + ","
#calculating the occurence of token
for l,m in FreqDist(last_val).items():
    #string concate for output
    string_counter_final = string_counter_final + str(l) + ":" + str(m) + ","
string_counter_final = string_counter_final[:-1]
string_counter_final = string_counter_final + '\n'

```

In the above cell:

1) First we have looped in the new_dict dictionary which has the date as the key and tokens as the values of the keys without any removal of the stopwords or anything.

2) Then the values of the above dictionary is being pasted to the nltk.collocations.BigramCollocationFinder to find the bigrams in the tokens

3) putting it into a new dict keys as the date and the values as the bigrams.

4) looping in the freq which contains the valid bigrams and another loop inside it in the dictionary which contains the bigrams created in 2nd step.

checking if the bigram in the freq is equal to the bigram in the dictionary then : --check key is present in the new dict if satisfy then append in the existing key into values -- if not then make new key and add its values to the key

5) Looping into the final_counter_bi_dic and inside loop in uni_dict_stem if the keys are same then the bigrams get appended to the uni_dict_stem.

6) making dict of vocab having key as words and value as index

7) looping into the uni_dict_stem calculating the occurrence of each id of the vocab in last_vocab. Then inside loop in the

last_val calculating the Freq dist of the keys of the values of the dict and concating into the string of the counter vector

output.

```

In [ ]: with open("30761182_countVec.txt", "w", encoding='utf-8') as text_file:
        text_file.write(string_counter_final)

```

Writing the string into the output file

Summary

Read all the excel files sheet and add the sheet into dataframes to make a dictionary of the text column in the sheet.

For Unigrams.txt:

Read stopwords file and then removed the stopped words and the words which are less than 3 length from the dictionary. Then removed the rare tokens and the context dependent i.e the words that comes <5 or greater than 60 times in the documents. Then stemmed the values of the dictionary values and then count the occurrence of the words using FreqDist. The dictionary at the end the desired dict which have dsate as a key and values as the unigrams.

For Bigrams.txt

make an object of the nltk.collocations.BigramAssocMeasures(). Then make a dictionary with the topp 200 bigrams using the finder.nbest function and then adding them into a dictionary.

For countvector.txt

Making bigrams from the new_dict which have all the bigrams of the text according to the date. Then checking the bigrams created from the valid bigrams that created in the bigram.txt and making a new dict. Then appending the unigrams to the dictionary from the stemmed dict that created in the creating unigram.txt steps. Then using Freq dist calculated the occurrence of each value and desired output file.

For vocab.txt used the unigrams from the above steps while creating unigram.txt then created bigrams. Then add both the unigrams and bigrams into one list and add then into a list. Made a set of that list and sort them. Then concated into the string and write the string into the file.