```
# -*- coding: utf-8 -*-
"""
Created on Tue May 24 12:01:16 2016

@author: Amit_Mac
"""

#********************************************************* Packages Import
*****************************************************************

from __future__ import division                                                 # Handling mathematics division
part
from datetime import datetime                                                   # Handle Timer part use to
calculate the running time of the program
import textmining                                                               # For Document_Term_Matrix
import numpy as np                                                              # Numpy package for python
arrays,matrix etc
from nltk.corpus import stopwords                                               # Handle stop words in the file
import peach as p                                                               # For creation of Topic/cluster
matrix
from numpy.random import random                                                 # For creation of random number
import codecs                                                                   # Handle the unicode
compatibility problem of the code
import enchant                                                                  # For the use of dictionary
import heapq                                                                    # Caculation of n-largest
indexes in an topic
import re                                                                       # For the regular expression
import os                                                                       # Removing unnecessary files

#********************************************************** Clock/Timer start
*****************************************************************

start_time = datetime.now()                                                     # Start of a clock to calculate
the time taken by the program to execute

#********************************************************** Filteration of a file
*************************************************************

with codecs.open("d_input.txt") as ins,open("Filtered_file1.txt","w") as inst:
# takes an input file open it in read mode and another file in write mode to insert the data
    print "Select an option and enter the number for filteration : "
# provide an option to the user to select the filteration scheme
    print " 1.  File without any numeric digit "
    #print " 2.  File without any character i.e. only digits and special characters or both combinations "
    print " 2.  File with character and number together and along  with special characters "
    print " 3.  File with special characters and its combination with digit , character or both "
    print " 4.  File without any filteration "

    try:
# Exception Handling use to handle when user proceed without entering any number "Just press enter"
        Filteration_choice = int(raw_input("Enter the number from 1-4 for your choice : "))
        if (Filteration_choice not in (1,2,3,4)):
# check for vlid entry by user
                print "Enter you choice according to the number associated with choice "
                Filteration_choice = int(raw_input(" Enter your choice once again : "))
    except ValueError:
        Filteration_choice = 1
    print '\n'

    print "Select an option for your choice of using Dictionary to check word (consume more time) : "
# provide an option to user whether to use dictionary or not
    print " 1. Use Dictionary"
    print " 2. Without Dictionary"
    try:
# Exception handling cases when user proceed without valid input then default 2(without dictionry) is
initialized
        Dictionary_choice = int(raw_input(" Input : "))
        if (Dictionary_choice not in (1,2)):
# check for vlid entry by user
                print "Enter you choice according to the number associated with choice "
                Dictionary_choice = int(raw_input(" Enter your choice once again : "))
    except ValueError:
        Dictionary_choice = 2
    print '\n'

    Special_characters = "[~\!\?\@#\$%\^&\*\(\)\=\|\/\>\<\,\._\+{}\"':;'\[\]]"
# Special character variable for regular expression
    Alphabets = "[a-z]"
# Alphabet variable for regular expression
    Numbers = "[0-9]"
# Numberic variable for regular expression
    Alpha_Num = "[a-z0-9]"
# Alphabet and number  variable for regular expression this will check for both alphabets and number
    Alpha_Num_Special = "[~\!\?\@#\$%\^&\*\(\)\=\|\/\>\<\,\._\+{}\":;'\[\]a-z0-9]"
# special character,Alphabet and number variable for regular expression check for anything from this three
```

```
things

    for line in ins:
# run the loop for every line read from the input text file
        text = ' '.join([word for word in line.lower().split() if word not
in(re.findall(r"^"+Special_characters+"+$", word))])                                                    #
remove word consist of only special characters like "@##,@@ etc"
        text = ' '.join([word for word in text.split() if not word.startswith("http")])
# remove words staring with http
        if (Filteration_choice == 1):
# filter the file and create the file which consist of letter and specail characters i.e no number
            text = ' '.join([word for word in text.split() if word
in(re.findall(r"^"+Alphabets+"*"+Special_characters+"*"+Alphabets+"*$", word))])
        #elif (Filteration_choice == 2):
        #    text = ' '.join([word for word in text.split() if word
in(re.findall(r"^"+Numbers+"*"+Special_characters+"*"+Numbers+"*$", word))])
        elif (Filteration_choice == 2):
# file contain the words which consist of both characters and numbers
            text = ' '.join([word for word in text.split() if word in(re.findall(r"^.*(?=.*"+Numbers+").(?
=.*"+Alphabets+")"+Alpha_Num_Special+"+$", word))])
        elif (Filteration_choice == 3):
#  contain the words which starts with the special characters
            text = ' '.join([word for word in text.split() if word
in(re.findall(r"^"+Special_characters+"+"+Alpha_Num+"*$", word))])
        else:
# no filteration at all
            text = ' '.join([word for word in text.split()])
        text = ' '.join(re.sub(Special_characters," ",word)for word in text.split())
# replace the special characters in the word with space
        text = ' '.join([word for word in text.split() if word not in (stopwords.words('english'))])
# remove stop words
        if (Dictionary_choice == 1):
# check for dictionary words if user want to
            text = ' '.join([word for word in text.split() if (enchant.Dict("en_US").check(word) == True)])
        else :
# proceed with dictionary checking of the word
            pass
        text = ' '.join([word for word in text.split() if (len(word) > 2)])
# remove the words whose length is 2 or less than that
        inst.write(text + '\n')
# write the line after filteration
inst.close()

#******************************** removing empty lines
*********************************************************************************

with codecs.open("Filtered_file1.txt") as input,open("Filtered_file.txt","w") as output:
    for line in input:
        if not line in ['\n', '\r\n']:          # Look  for empty lines in a file
            output.write(line)                  # write the line if line is not empty one
output.close()
os.remove("Filtered_file1.txt")                 # remove the file which created above after filteration

#********************************** print the completion of a process
*****************************************************************

print "Filteration completed"
print '\n'

#*****************************  open the files to read and write data
*****************************************************************

fileinput  = open("Filtered_file.txt").readlines()              # read Filtered file for furtur use
words_textfile = open("words_textfile.txt","w")                 # open file words_textfile in write mode
to write all words/documents in the documentword matrix
Topics_file = open("Topics_file.txt","w")                       # open Topics_file in write mode to write
top topics
Words_file = open("Words_file.txt","w")                         # open word_file in write mode to write top
words
#****************************** document matrix creation
*********************************************************************************

def term_document_matrix():
    num_documents = 0   # initialization of variable to count number of documents
    for line in fileinput:
       num_documents = num_documents + 1                                        # read the documents it
increase it by one as document encounter
       reading_file_info = [item.rstrip('\n') for item in fileinput]            # remove the leading white
spaces till it encounter new line
       tdm = textmining.TermDocumentMatrix()                                    # define Termdocument matrix
       for i in range(0,num_documents):
           tdm.add_doc(reading_file_info[i])                                    # creating the matrix rows
one by one
    tdm.write_csv('TermDocumentMatrix_1.csv',cutoff=1)                          # creation of a CSV file of
the matrix
```

```
    temp = list(tdm.rows(cutoff=1))                                                    # changing the matrix to
list for furthur use
    vocab = tuple(temp[0])                                                             # storing the words in a
tuple and assign to a vaiable vocab
    num_words = len(vocab)                                                             # calculate the number of
words

#************************************** dispplay completion and total number of words and documents
*******************************

    print "Document Matrix created"
    print "\n"
    print " There are %d words and %d documents in the file " %(num_words,num_documents)                #
displaying total number of documents and words
    print "\n"
#************************************ calculation of probabiltiy related to the TermDocument matrix
*********************************

    print " Select an option of Matrix  which you want to create ultimately : "                         #
Asking an user to proceed with which matrix W or D
    print " 1. probability of word by term P(W/T)"
    print " 2. probabiliy of document by term P(D/T) "
    try:
        Matrix_choice = int(raw_input(" Input : "))                                                     #
assign the user input to the matrix_choice variable
        if (Matrix_choice not in (1,2)):
# check for vlid entry by user
                print "Enter you choice according to the number associated with choice "
                Matrix_choice = int(raw_input(" Enter your choice once again : "))
    except ValueError:                                                                                  #
if user donot enter any input like press enter the default 1 is assign
        Matrix_choice = 1

    print '\n'

    if (Matrix_choice == 2):                                                                            #
this will proceed when user opt for D matrix
        x = np.array(temp[1:])                                                                          #
changing the list to numpy array
        x=np.transpose(x)                                                                               #
transpose the matrix to optain the documents in the columns and words in rows
        for i in range (0,len(x)):
            words_textfile.write("%d. document%d"  %((i)+1,(i)+1))                                      #
write all documents in a text file name Words_textfile
            words_textfile.write( '\n' )

    else :                                                                                              #
respond in W matrix
        x = np.array(temp[1:])                                                                          #
conversion of an list to array
        words_textfile.write('All words of the Document Term frequency Matrix are :' + '\n')
        words_textfile.write( '\n' )
        for i in range (0,len(vocab)):
            words_textfile.write("%d. %s"  %((i)+1,vocab[i]))                                           #
write all words present in the file after filteration to a text file
            words_textfile.write( '\n' )

    rows_document_matrix = x.shape[0]                                                                   #
calculate the number of rows in X matrix
    columns_document_matrix = x.shape[1]                                                                #
calculate the number of columns in X matrix
    summation_document_matrix_rows = x.sum(axis = 1)
# calculate the sum of each row of a matrix
    summation_document_matrix_columns = x.sum(axis = 0)
# calculate the sum of each columns of a matrix
    summation_document_matrix_rows = summation_document_matrix_rows[:, None]
# changing from horizontal to vertical rows sum data for calculation later
    summation_document_matrix_columns = summation_document_matrix_columns[:, None]
# changing from horizontal to vertical columns sum data for calculation later
    Sum_of_array = x.sum()
# calculate the sum of a matrix
    probability_of_each_word_in_a_specific_document_matrix =
np.zeros((rows_document_matrix,columns_document_matrix))   # initialize the P(w/d) matrix with zeros

    if (Matrix_choice == 2):
       probability_of_word_matrix = np.zeros((1, rows_document_matrix))                         # initialize
the p(w) matrix
       probability_of_document_inmatrix = np.zeros((1,columns_document_matrix))                 # initialize
the p(d) matrix

       for i in range(0,rows_document_matrix):
           probability_of_word_matrix[0][i] = summation_document_matrix_rows[i][0] / Sum_of_array
# calculation of p(w)
           for j in range(0,columns_document_matrix):
               probability_of_document_inmatrix[0][j] = summation_document_matrix_columns[j][0] /
```

```
Sum_of_array  # calculation of p(d)
               probability_of_each_word_in_a_specific_document_matrix[i][j] = x[i][j] /
summation_document_matrix_columns[j][0]   # calculation of p(w/d)

    else :

       print " Select an option to calculate the probability of a document in a Document_Term_Matrix p(D) :
"                    # choice for calculation p(d) in W matrix
       print " 1. Summation of distribution of word frequency in a particular document / summation of word
distribution frequency in entire matrix "
       print " 2. Individual document / total number of documents (1 / total number of documents) "
       try:
           probability_of_document_calculation_choice = int(raw_input(" Input : "))
           if (probability_of_document_calculation_choice not in (1,2)):
# check for vlid entry by user
               print "Enter you choice according to the number associated with choice "
               probability_of_document_calculation_choice = int(raw_input(" Enter your choice once again :
"))
       except ValueError:
           probability_of_document_calculation_choice = 2
# in case of enter 2 option is selected
       print '\n'

       probability_of_word_matrix = np.zeros((1, columns_document_matrix))
# initialize the p(w) matrix
       probability_of_document_inmatrix = np.zeros((1,rows_document_matrix))
# initialize the p(d) matrix

       for i in range(0,rows_document_matrix):
            if (probability_of_document_calculation_choice == 1) :
# calculation of the p(d) matrix based on user input
               probability_of_document_inmatrix[0][i] = summation_document_matrix_rows[i][0] /
Sum_of_array


            else :
               probability_of_document_inmatrix[0][i] = 1/num_documents


            for j in range(0,columns_document_matrix):
               probability_of_word_matrix[0][j] = summation_document_matrix_columns[j][0] / Sum_of_array
# calculation of p(w)
               probability_of_each_word_in_a_specific_document_matrix[i][j] = x[i][j] /
summation_document_matrix_rows[i][0]     # calculation of p(w/d)

#********************************** showing of completon of task
**************************************************************************

    print " Probability related to Term Document Matrix completed "
    print "\n"

#****************************** creation of topic matrix
*******************************************************************************

    try :
        Number_of_cluster = int(raw_input("Enter the number of clusters/topics you want to create : "))
# input the number of topics/clusters from the user

    except ValueError:
        Number_of_cluster = 10   # default topic/cluster is 10

    Assign_number_of_cluster = Number_of_cluster     # Assign the topics to new variable to use in the last
part of topic matrix

    print " Select an option of Reduction Technique : "    # option to select the deduction technique
    print " 1. SVD approach "
    print " 2. Peach approach "
    try:
        Deduction_technique = int(raw_input(" Input : "))               # assign the choice to the variable
        if (Deduction_technique not in (1,2)):
# check for vlid entry by user
               print "Enter you choice according to the number associated with choice "
               Deduction_technique = int(raw_input(" Enter your choice once again : "))
    except ValueError:
        Deduction_technique = 2   # default technique is 2

    if (Matrix_choice == 2) :
        num_lines = int(num_words)                  # numlines for random number generation as a input for
creation of topic matrix based on the user choice
    else:
        num_lines = int(num_documents)

    if (Deduction_technique == 1) :                                 # SVD approach
        Method_SVD = np.linalg.svd(x, full_matrices=True)         # SVD implementation and output assign to
a list variable
```

```
        Matrix_S = Method_SVD[0]                                    # extraction of s matrix
        Matrix_S_Rows = Matrix_S.shape[0]                           # calculation of rows of s matrix
        Array_for_Fcm = np.zeros((Matrix_S_Rows, 2))                # initialize the array for storing two
rows of s matrix
        for i in range(0,Matrix_S_Rows):
            for j in range(0,2):
                Array_for_Fcm[i][j] = Matrix_S[i][j]                # shifthing the values to array from s
matrix
        fcm = Array_for_Fcm

    else :                                                          # peach approach
        mu = random((num_lines,Number_of_cluster ))                 # random number as a input for peach
library function to calculate the topic matrix and initial cluster mention by user
        fcm = p.FuzzyCMeans(x, mu, 2)                               # calculate the fuzzycmeans where x is a
initial documentterm matrix , mu random number
        fcm = fcm.mu                                                # access the fcm matrix with the random
number extension
        if (Number_of_cluster < 10):                                # if cluster are less than 10 then in
each step reduce the size of input cluster by 1 till 2 is encounter with previous fcm matrix as input
            b = 1
            while (Number_of_cluster > 2):
                Number_of_cluster = Number_of_cluster - b
                mu = random((num_lines, Number_of_cluster))
                fcm = p.FuzzyCMeans(fcm, mu, 2)
                fcm = fcm.mu

        elif (Number_of_cluster > 10 and Number_of_cluster <= 200 ):    # clusters between 10 and 200,
reduce the clusters by number get by divide by 10 n take upper value till
            b = Number_of_cluster / 10
            b = np.ceil(b)
            while (Number_of_cluster > 2):
                Number_of_cluster = Number_of_cluster - b
                if (Number_of_cluster > 2):                                         # run till cluster
greater than 2
                    mu = random((num_lines, Number_of_cluster))
                    fcm = p.FuzzyCMeans(fcm, mu, 2)
                    fcm = fcm.mu
            mu = random((num_lines, 2))                                             # final reduction with
2 clusters for better result
            fcm = p.FuzzyCMeans(fcm, mu, 2)
            fcm = fcm.mu

        else:                                                                       # cluster greater than
200 and reduce clusters by number divide by 10
            b = Number_of_cluster / 10
            b = np.ceil(b)
            while (Number_of_cluster > 10):                                         # run till greater than
10
                Number_of_cluster = Number_of_cluster - b
                if (Number_of_cluster > 10):
                    mu = random((num_lines, Number_of_cluster))
                    fcm = p.FuzzyCMeans(fcm, mu, 2)
                    fcm = fcm.mu

            mu = random((num_lines, 10))                                            # reduction with 10
clusters
            fcm = p.FuzzyCMeans(fcm, mu, 2)
            fcm = fcm.mu

            mu = random((num_lines, 2))                                             # reduction with 2
clusters
            fcm = p.FuzzyCMeans(fcm, mu, 2)
            fcm = fcm.mu

    mu = random((num_lines, Assign_number_of_cluster))                              # final topic matrix
with initial cluster input by user
    fcm = p.FuzzyCMeans(fcm, mu, 2)
    fcm = fcm.mu

#******************************************************* display of completion of task
************************************************************

    print " Cluster/Topic matrix created "
    print "\n"

#************************************************* probabilities related to the topic matrix n final
w/t or t/d matrix ***********************************************

    num_arra =  fcm                                 # assign of a topic matrix to num_arrra variable
    summation = num_arra.sum(axis = 1)              # summation of rows of topic matrix for
normalization
    summation_vertical = summation[:, None]         # transefer of horizontal result to vertical
    rows = num_arra.shape[0]                        # rows of topic matrix
    columns = num_arra.shape[1]                     # columns of topic matrix
    num_arra=num_arra.astype(float)                 # initialize the matrix as a float to store result
```

```
after normailzation

    for rows_count in range (0,rows):
        divide_sum = summation_vertical.item(rows_count,0)                    # take out each item one by
one of rows sum to divide for normalization
        for i in range(0,columns):
            replace_division = num_arra.item(rows_count,i)/divide_sum        # normalization of an value of
topic matrix
            num_arra[rows_count,i]  = replace_division                       # relacing the value of topic
matrix by normalized one

    probability_document_by_term_cluster = np.zeros((rows, columns))              # initializatio of p(d,t)
    probability_document_by_term_cluster_normalize = np.zeros((rows, columns))    # iniatialize of p(d/t)

    for i in range(0,rows):
        for j in range(0,columns):
            if(Matrix_choice == 1):
                probability_document_by_term_cluster[i][j]=num_arra[i][j] *
probability_of_document_inmatrix[0][i]    # calculation of p(d,t) base on the user choice which he decide
to proceed with W or D matrix
            else :
                probability_document_by_term_cluster[i][j]=num_arra[i][j] * probability_of_word_matrix[0]
[i]

    Sum_of_cluster_rows = probability_document_by_term_cluster.sum(axis = 1)
# calculate the sum of rows of p(d,t) matrix for normalization
    Sum_of_cluster_rows = Sum_of_cluster_rows[:, None]
    probability_of_each_word_in_a_specific_document_matrix =
np.transpose(probability_of_each_word_in_a_specific_document_matrix)          # initialization of p(d/t)
normalized matrix with zeros

    for i in range(0,rows):
        for j in range(0,columns):
            probability_document_by_term_cluster_normalize[i][j]=probability_document_by_term_cluster[i][j]
/ Sum_of_cluster_rows[i][0]    # calculate the p(d/t) divide the element by its row sum as a normailze


    array_w =
np.dot(probability_of_each_word_in_a_specific_document_matrix,probability_document_by_term_cluster_normaliz
e)                 # matrix multiplication of p(w/d)*p(d/t) to get p(w/t) or either p(t/d)
    summation = array_w.sum(axis = 1 )
# calculate the sum of rows for normalization
    summation = summation[:, None]
    rows = len(array_w)
    columns = len(array_w[0])
    array  = np.zeros((rows,columns))
# initialize with zeros array to store after normalization
    for i in range(0,rows):
        for j in range(0,columns):
            array[i][j] = array_w[i][j] / summation[i][0]
# store the normalized value in array
    np.savetxt('Words_topics.txt',array)
# create the txt file of this matrix
#************************************** Display of completion of task
********************************************************************************

    print "probability releated to topic/cluster matrix created"
    print "\n"
#************************************* display of topics
*****************************************************************************************************
    topic_list = []
    print "Select a choice to display the topics of Document : "
# option to display all document topics or any particular document
    print " 1. All Document Topics "
    print " 2. Single Document Topics "

    try:
        Topic_choice = int(raw_input(" Input : "))
        if (Topic_choice not in (1,2)):
# check for vlid entry by user
            print "Enter you choice according to the number associated with choice "
            Topic_choice = int(raw_input(" Enter your choice once again : "))
    except ValueError:
        Topic_choice = 1


    rows = len(array)
    columns = len(array[0])
    List_for_topic = []

    if (Topic_choice == 2) :
        print " You have %d number of documents to choose " %rows
# ask for document to display topics
        try:
            Document_number = int(raw_input(" Enter the document number to display topics : " ))
```

```
            if (Document_number > rows):
# if the user enter greater than number of documents it ask it for again
                print (" please enter the number less than %d as you have this specific document " %rows)
                Document_number = int(raw_input(" Enter the document number again : "))
            else :
                pass

        except ValueError:
            Document_number = 1
# default document is 1

        print "Select an option whether you want to Display for document : "
# option for topic display for particular entered document
        print " 1. All topics "
        print " 2. Specific number of Topics "
        try :
            Number_of_Topic = int(raw_input(" Input : "))
            if (Number_of_Topic not in (1,2)):
# check for vlid entry by user
                print "Enter you choice according to the number associated with choice "
                Number_of_Topic = int(raw_input(" Enter your choice once again : "))

        except ValueError:
            Number_of_Topic = 1
# default is all topics of document

        for i in range(0,rows):
                if (i+1 == Document_number):
# matches the document entered by the user
                    for j in range(0,columns):
                        List_for_topic.append(array[i][j])
# entering all topics value of document to list to retrive the decreasing order values

                    if (Number_of_Topic == 2):
                        print " You have %d number of topics to choose " %columns
                        print " Enter the number of topics to display : "
# ask for number of topics to display if user ask for
                        try:
                            Number_of_topic_display = int(raw_input(" Input : "))
                            if (Number_of_topic_display > columns):
                                print (" please enter the number less than %d as you have this specific
topic " %columns)
                                Number_of_topic_display = int(raw_input(" Enter the topics number again :
"))
                        except ValueError:
                            Number_of_topic_display = 1
# default is 1 topic
                        indexes = heapq.nlargest(Number_of_topic_display, range(len(List_for_topic)),
List_for_topic.__getitem__)           # retrieve the top topic index number as topic user want to display
                        Topics_file.write('Top %d topic for Document %d is' %
(Number_of_topic_display,Document_number))                    # writing the document number in the text
file
                        Topics_file.write('\n')

                        for i in range(0,len(indexes)):
                            topic_list.append(str('%d. t%d - %f ' %((i)+1,indexes[i]+1,
(List_for_topic[indexes[i]])))))
                            #Topics_file.write('%d. t%d - %f ' %((i)+1,indexes[i]+1,
(List_for_topic[indexes[i]])))                              # writing the topic in the text file as topic
user want to display
                            #Topics_file.write('\n')

                    else :
                        indexes = heapq.nlargest(len(List_for_topic), range(len(List_for_topic)),
List_for_topic.__getitem__)               # retrieve the index number of all topic in decreasing order n
store in list
                        Topics_file.write('Topic for Document %d is' %Document_number)
# writing the document number in the text file
                        Topics_file.write('\n')
                        for i in range(0,len(indexes)):
                            topic_list.append(str('%d. t%d - %f ' %((i)+1,indexes[i]+1,
(List_for_topic[indexes[i]])))))
                            #Topics_file.write('%d. t%d - %f ' %((i)+1,indexes[i]+1,
(List_for_topic[indexes[i]])))                              # writing the topics in tet file in decreasing
order
                            #Topics_file.write('\n')
                        Topics_file.write("  ".join(topic_list))
                        Topics_file.write('\n')
    else :
        for i in range(0,rows):
            for j in range(0,columns):
                List_for_topic.append(array[i][j])
# transfer the each topic value to list one by one
            indexes = heapq.nlargest(len(List_for_topic), range(len(List_for_topic)),
List_for_topic.__getitem__)                           # retrieve the index in decreasing oreder for
```

```
transfered topic values
            Topics_file.write('topic for Document %d is : ' %((i)+1))
# writing document number in text file
            Topics_file.write('\n')
            for i in range(0,len(indexes)):
                topic_list.append(str('%d. t%d - %f ' %((i)+1,indexes[i]+1,(List_for_topic[indexes[i]]))))
                #Topics_file.write('%d. t%d - %f ' %((i)+1,indexes[i]+1,(List_for_topic[indexes[i]])))
# wriitng the topic in the text file
                #Topics_file.write('\n')
            Topics_file.write("  ".join(topic_list))
            Topics_file.write('\n')
            del topic_list[ : ]
            del List_for_topic[ : ]
# delete the list valus so new topic values transfer for back steps follow again

#***********************************  task completed
*****************************************************************************************************
*

    print "Topic listed file created"
    print "\n"
###***************************  top words retrieve
*****************************************************************************************************
*****

    rows = len(array)                   # calculate the rows of the p(w/t) or p(d/t) matrix
    columns = len(array[0])             # calculate the columns of the matrix
    List = []
    Display_list = []
    l = 1
    print " Select an option on view of top  word/document  print in topic :"           # option to print
all words or specific number of words to display
    print " 1. All Words/documents in the topic"
    print " 2. Specific number of words/document"
    try:
        Word_choice = int(raw_input(" Input : "))                                       # assign choice to
variable
        if (Word_choice not in (1,2)):                                                  # check for vlid
entry by user
            print "Enter you choice according to the number associated with choice "
            Word_choice = int(raw_input(" Enter your choice once again : "))
    except ValueError:
        Word_choice = 1                                                                  # default print
all words

    if (Word_choice == 2):                                                               #  if user want
specific number of words then proceed this
        print " You have %d number of words/documents to choose " %rows
        print "Enter the number of the top words/documents you want to display : "
        try :
            Number_of_words = int(raw_input(" Input : "))                                # take the number
of words user want to display put as a input
            if (Number_of_words > rows):
                print " You exceed the number of words/documents present "
                print " you have only %d words/documents so enter within it " % rows
                Number_of_words = int(raw_input(" Enter the number of words/documents again : "))
            else:
                pass

        except ValueError:
            Number_of_words = 2                                                                  #
default is 2 words/documents

        for j in range(0,columns):
            for i in range(0,rows):
                List.append(array[i][j])                                                         #
transfer each topic values in list
            ab = heapq.nlargest(Number_of_words, range(len(List)), List.__getitem__)             #
retrieve the top index of word entered by user
            Words_file.write('top %d words/documents for cluster %d is : ' %(Number_of_words,l))        #
write the document/word number in the text file
            Words_file.write('\n')
            for k in range(0,len(ab)):
                if (Matrix_choice == 2):
                    Display_list.append(str('%d. document%d - %f' %((k)+1,(ab[k]+1),List[ab[k]])))
                    #Words_file.write('%d. document%d - %f' %((k)+1,(ab[k]+1),List[ab[k]]))              #
write document in the text file
                    #Words_file.write('\n')
                else :
                    Display_list.append(str('%d. %s - %f' %((k)+1,vocab[ab[k]],List[ab[k]])))
                    #Words_file.write('%d. %s - %f' %((k)+1,vocab[ab[k]],List[ab[k]]))                  #
write  words in the document
                    #Words_file.write('\n')
            Words_file.write("  ".join(Display_list))
            Words_file.write('\n')
```

```
            l = l + 1
            del Display_list [ : ]
            del List[ : ]


    else :
        for j in range(0,columns):
            for i in range(0,rows):
                List.append(array[i][j])                                                    #
transfer each topic values in list
            ab = heapq.nlargest(len(List), range(len(List)), List.__getitem__)              #
retrieve the index of the list in decreasing order
            Words_file.write('All words/documents for cluster %d is : ' %l)                 #
write the document/word number in the text file
            Words_file.write('\n')
            for k in range(0,len(ab)):
                if (Matrix_choice == 2):
                    Display_list.append(str('%d. document%d - %f' %((k)+1,(ab[k]+1),List[ab[k]])))
                    #Words_file.write('%d. document%d - %f' %((k)+1,(ab[k]+1),List[ab[k]]))      #
write document in the text file
                    #Words_file.write('\n')
                else :
                    Display_list.append(str('%d. %s - %f' %((k)+1,vocab[ab[k]],List[ab[k]])))
                    #Words_file.write('%d. %s - %f' %((k)+1,vocab[ab[k]],List[ab[k]]))           #
write  words in the document
                    #Words_file.write('\n')
            Words_file.write("  ".join(Display_list))
            Words_file.write('\n')
            l = l + 1
            del Display_list[ : ]
            del List[ : ]                                                                   #
delete the list for furthur transfer
#************************************* completion of task
*****************************************************************************************
    print "word listed file created"
    print "\n"
#************************************************ closing functions and open file
*************************************************************************
term_document_matrix()
words_textfile.close()
Topics_file.close()
Words_file.close()
#************************************************ stop the timer and display the time taken to complete
the whole job ***********************************
end_time = datetime.now()

print '\n'
print('Duration: {}'.format(end_time - start_time))
print '\n'
```