HW #1: Analyze Documents by Numpy

Each assignment needs to be completed independently. Never ever copy others' work (even with minor modification, e.g. changing variable names). Anti-Plagiarism software will be used to check all submissions.

Instructions:

- Please read the problem description carefully
- Make sure to complete all requirements (shown as bullets) . In general, it would be much easier if you complete the requirements in the order as shown in the problem description
- Follow the Submission Instruction to submit your assignment

Problem Description

In this assignment, you'll write functions to analyze an article to find out the word distributions and key concepts.

The packages you'll need for this assignment include `numpy` and `string`. Some useful functions:

- string, list, dictionary: `split` , `count` , `index` , `strip`
- numpy: `sum` , `where` , `log` , `argsort` , `argmin` , `argmax`

Q1. Define a function to analyze word counts in an input sentence

Define a function named `tokenize(doc)` which does the following:

- accepts a document (i.e., `doc` parameter) as an input
- first splits a document into paragraphs by delimiter `\n\n` (i.e. two new lines)
- for each paragraph,
  - splits it into a list of tokens by **space** (including tab, and new line).
    - e.g., `it's a hello world!!!` will be split into tokens `["it's", "a","hello","world!!!"]`
  - removes the **leading/trailing punctuations or spaces** of each token, if any
    - e.g., `world!!! -> world`, while `it's` does not change
    - hint, you can import module *string*, use `string.punctuation` to get a list of punctuations (say `puncts`), and then use function `strip(puncts)` to remove leading or trailing punctuations in each token
  - a token has at least two characters
  - converts all tokens into lower case
  - find the count of each unique token and save the count as a dictionary, named `word_dict`, i.e., `{world: 1, a: 1, ...}`
- creates another dictionary,say `para_dict`, where a key is the order of each paragraph in `doc`, and the value is the `word_dict` generated from this paragraph
- returns the dictionary `para_dict` and the paragraphs in the document

In [1]:
```python
import string
import pandas as pd
import numpy as np

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
def tokenize(doc):

    para_dict, para = None, None

    para = doc.split("\n\n")   # to delete \n\n and separate document into p
    list_paragraphs = []
    for each in para:
        if each.count("\n")>0:  # to delete \n if there is at least \n in ea
            change=each.replace("\n"," ")
            list_paragraphs.append(change.split(" ")) # to create a list of
        else:
            list_paragraphs.append(each.split(" ")) # to create a list of to

    puncts = string.punctuation     # to get a list of punctuations
    word_dict=[]


    for paragraphs_order in range(len(list_paragraphs)):
        for word in range(len(list_paragraphs[paragraphs_order])):
            # remove leading or trailing punctuations in each token and conv
            list_paragraphs[paragraphs_order][word] = list_paragraphs[paragr

        for letters in list_paragraphs[paragraphs_order][:]:
            # to remove a token has less than two characters
            if len(letters) < 2:
                list_paragraphs[paragraphs_order].remove(letters)


        each_word = set(list_paragraphs[paragraphs_order])
        number_word = np.zeros(len(each_word))
        each_para = dict(zip(each_word,number_word))
        for count_word in each_para:  # to count how many time a token appea
            each_para[count_word] = list_paragraphs[paragraphs_order].count(
        word_dict.append(each_para)
    para_number = list(range(len(list_paragraphs)))
    # to create a dictionary where a key is the order of each paragraph in d
    para_dict = (dict(zip(para_number,word_dict)))



    return para_dict, para
```

In [4]:
```
# test your code
doc = "it's a hello world!!!\nit is hello world again.\n\nThis is paragraph
print(doc)
tokenize(doc)
```

```
it's a hello world!!!
it is hello world again.

This is paragraph 2.
```

Out[4]:
```
({0: {"it's": 1, 'hello': 2, 'it': 1, 'is': 1, 'world': 2, 'again': 1},
  1: {'paragraph': 1, 'is': 1, 'this': 1}},
 ["it's a hello world!!!\nit is hello world again.", 'This is paragraph 2.']
)
```

Q2. Generate a document term matrix (DTM) as a numpy array

Define a function `get_dtm(doc)` as follows:

- accepts a document, i.e., `doc`, as an input
- uses `tokenize` function you defined in Q1 to get the word dictionary for each paragraph in the document
- pools the keys from all the word dictionaries to get a list of unique words, denoted as `unique_words`
- creates a numpy array, say `dtm` with a shape (# of paragraphs x # of unique words), and set the initial values to 0.
- fills cell `dtm[i,j]` with the count of the `j` th word in the `i` th paragraph
- returns `dtm` and `unique_words`

In [5]:
```python
def get_dtm(doc):

    dtm, all_words = None, None
    para_dict, para = tokenize(doc)
    all_words = []
    # to get the word dictionary for each paragraph in the document
    for i in range(len(para_dict)):
        all_words += (list(para_dict[i].keys()))
    unique_words = list(set(all_words))   # to get a list of unique word
    array_shape = (len(para_dict),len(unique_words))  # to find the shape of
    dtm = np.zeros(array_shape)      # to set an array with the initial value
    for i in range(len(dtm)):      # to fills cell dtm with words in each par
        for j in range(len(unique_words)):
            dtm[i][j] = para_dict[i].get(unique_words[j],0)
    return dtm, unique_words
```

In [6]:
```python
dtm, all_words = get_dtm(doc)
dtm
all_words
```

Out[6]:
```
array([[0., 1., 2., 1., 1., 0., 2., 1.],
       [1., 0., 0., 0., 1., 1., 0., 0.]])
```

Out[6]:
```
['paragraph', "it's", 'hello', 'it', 'is', 'this', 'world', 'again']
```

In [7]:
```
# A test document. This document can be found at https://www.wboi.org/npr-ne

doc = open("chatgpt_npr.txt", 'r').read()
dtm, all_words = get_dtm(doc)
```

In [8]:
```
print(doc)
```

```
Ethan Mollick has a message for the humans and the machines: can't we all ju
st get along?

After all, we are now officially in an A.I. world and we're going to have to
share it, reasons the associate professor at the University of Pennsylvania'
s prestigious Wharton School.

"This was a sudden change, right? There is a lot of good stuff that we are g
oing to have to do differently, but I think we could solve the problems of h
ow we teach people to write in a world with ChatGPT," Mollick told NPR.

Ever since the chatbot ChatGPT launched in November, educators have raised c
oncerns it could facilitate cheating.

Some school districts have banned access to the bot, and not without reason.
The artificial intelligence tool from the company OpenAI can compose poetry.
It can write computer code. It can maybe even pass an MBA exam.

One Wharton professor recently fed the chatbot the final exam questions for
a core MBA course and found that, despite some surprising math errors, he wo
uld have given it a B or a B-minus in the class.

And yet, not all educators are shying away from the bot.

This year, Mollick is not only allowing his students to use ChatGPT, they ar
e required to. And he has formally adopted an A.I. policy into his syllabus
for the first time.

He teaches classes in entrepreneurship and innovation, and said the early in
dications were the move was going great.

"The truth is, I probably couldn't have stopped them even if I didn't requir
e it," Mollick said.

This week he ran a session where students were asked to come up with ideas f
or their class project. Almost everyone had ChatGPT running and were asking
it to generate projects, and then they interrogated the bot's ideas with fur
ther prompts.

"And the ideas so far are great, partially as a result of that set of intera
ctions," Mollick said.

 Users experimenting with the chatbot are warned before testing the tool tha
t ChatGPT "may occasionally generate incorrect or misleading information."
/ OpenAI/Screenshot By NPR
```

/
OpenAI/Screenshot By NPR
Users experimenting with the chatbot are warned before testing the tool that ChatGPT "may occasionally generate incorrect or misleading information."
He readily admits he alternates between enthusiasm and anxiety about how artificial intelligence can change assessments in the classroom, but he believes educators need to move with the times.

"We taught people how to do math in a world with calculators," he said. Now the challenge is for educators to teach students how the world has changed again, and how they can adapt to that.

Mollick's new policy states that using A.I. is an "emerging skill"; that it can be wrong and students should check its results against other sources; and that they will be responsible for any errors or omissions provided by the tool.

And, perhaps most importantly, students need to acknowledge when and how they have used it.

"Failure to do so is in violation of academic honesty policies," the policy reads.

Mollick isn't the first to try to put guardrails in place for a post-ChatGPT world.

Earlier this month, 22-year-old Princeton student Edward Tian created an app to detect if something had been written by a machine. Named GPTZero, it was so popular that when he launched it, the app crashed from overuse.

"Humans deserve to know when something is written by a human or written by a machine," Tian told NPR of his motivation.

Mollick agrees, but isn't convinced that educators can ever truly stop cheating.

He cites a survey of Stanford students that found many had already used ChatGPT in their final exams, and he points to estimates that thousands of people in places like Kenya are writing essays on behalf of students abroad.

"I think everybody is cheating ... I mean, it's happening. So what I'm asking students to do is just be honest with me," he said. "Tell me what they use ChatGPT for, tell me what they used as prompts to get it to do what they want, and that's all I'm asking from them. We're in a world where this is happening, but now it's just going to be at an even grander scale."

"I don't think human nature changes as a result of ChatGPT. I think capability did."

In [9]:
```
# To ensure dtm is correct, check what words in a paragraph have been captur

p = 6 # paragraph id

[w for i,w in enumerate(all_words) if dtm[p][i]>0]
```

Out[9]:
```
['not',
 'are',
 'shying',
 'and',
 'away',
 'from',
 'educators',
 'yet',
 'the',
 'bot',
 'all']
```

Q3 Analyze DTM Array

Don't use any loop in this task. You should use array operations to take the advantage of high performance computing.

Define a function named `analyze_dtm(dtm, words, paragraphs)` which:

- takes an array dtm and $words$ as an input, where dtm is the array you get in Q2 with a shape $(m \times n)$, and $words$ contains an array of words corresponding to the columns of dtm.
- calculates the paragraph frequency for each word j, e.g. how many paragraphs contain word j. Save the result to array df. df has shape of $(n,)$ or $(1, n)$.
- normalizes the word count per paragraph: divides word count, i.e., $dtm_{i,j}$, by the total number of words in paragraph i. Save the result as an array named tf. tf has shape of (m,n).
- for each $dtm_{i,j}$, calculates $tfidf_{i,j} = \frac{tf_{i, j}}{1+log(df_j)}$, i.e., divide each normalized word count by the log of the paragraph frequency of the word (add 1 to the denominator to avoid dividing by 0). $tfidf$ has shape of (m,n)
- prints out the following (hint: you can zip words and their values into a list so that there is no need for loop during printing):
  - the total number of words in the document represented by dtm
  - the number of paragraphs and the number of unique words in the document
  - the most frequent top 10 words in this document
  - top-10 words that show in most of the paragraphs, i.e. words with the top 10 largest df values (show words and their df values)
  - the shortest paragraph (i.e., the one with the least number of words)
  - top-5 words with the largest $tfidf$ values in the longest sentence (show words and values)

Note, for all the steps, **do not use any loop**. Just use array functions and broadcasting for high performance computation.

Your answer may be different from the example output, since words may have the same values in the dtm but are kept in positions

In [11]:
```python
def analyze_dtm(dtm, words, paragraphs):

    # to calculates how many paragraphs for each word
    df = np.where(dtm > 0,1,0).sum(axis=0)
    # to normalizes the word count per paragraph
    tf = (dtm.T/np.sum(dtm, axis=1)).T
    denominator = 1 + np.log(df)
    # to add 1 to the denominator to avoid dividing by 0
    denominator[denominator==0] = 1
    tfidf = tf/denominator
    # to creat a list of the frequent words in this document
    top_word = list(zip(words, np.sum(dtm, axis = 0)))
    # to sort the frequent words descending
    top_word.sort(key=lambda x: x[1], reverse=True)

    # to create a list of words to find the top 10 largest df values
    top_value = list(zip(words, df))
    top_value.sort(key=lambda x: x[1],reverse=True)

    # to find the position of the shortest paragraph
    shortest_para = int(np.argmin(np.sum(dtm,axis=1, keepdims = True), axis
    # to find the position of the longest paragraph
    longest_para = int(np.argmax(np.sum(dtm,axis=1, keepdims = True), axis =
    top_tfidf = list(zip(words,tfidf[longest_para]))
    top_tfidf.sort(key=lambda x: x[1], reverse= True)

    print("The total number of words:\n",np.sum(dtm),"\n")
    print(f"The number of paragraps: {dtm.shape[0]}, the number of unique wo
    print("The top 10 frequent words:\n", top_word[:10],"\n")
    print("The top 10 words with highest df values:\n", top_value[:10],"\n")
    print("The shortest paragraph :\n",paragraphs[shortest_para],"\n")
    print("the top 5 words with the highest tf_idf values in the longest par
```

In [12]:
```python
para_dict, paragraphs = tokenize(doc)

# convert words in numpy arrays so that you can use array slicing
words = np.array(all_words)

analyze_dtm(dtm, words, paragraphs)
```

```
The total number of words:
 686.0

The number of paragraps: 24, the number of unique words in the document: 312
.

The top 10 frequent words:
 [('the', 31.0), ('to', 25.0), ('and', 19.0), ('that', 13.0), ('it', 12.0),
('he', 12.0), ('in', 12.0), ('of', 11.0), ('is', 10.0), ('chatgpt', 9.0)]

The top 10 words with highest df values:
 [('the', 18), ('and', 15), ('to', 14), ('in', 11), ('it', 10), ('that', 9),
('he', 9), ('is', 8), ('chatgpt', 8), ('for', 8)]

The shortest paragraph :
 And yet, not all educators are shying away from the bot.

the top 5 words with the highest tf_idf values in the longest paragraph:
 [('testing', 0.02666666666666667), ('misleading', 0.02666666666666667), ('u
sers', 0.02666666666666667), ('occasionally', 0.02666666666666667), ('openai
/screenshot', 0.02666666666666667)]
```

Q4. Find co-occuring words (Bonus)

Can you leverage $dtm$ array you generated to find what words frequently coocur with a specific word? For example, "students" and "chatgpt" coocur in 4 paragraphs.

Define a function `find_coocur(w1, w2, dtm, words)`, which returns the paragraphs containong both words w1 and w2.

Use a pdf file to describe your ideas and also implement your ideas. Again, `DO NOT USE LOOP`!

In [13]:
```python
def find_coocur(w1, w2, dtm, words, paragraphs):

    result = None
    words_w1 = np.where(words == w1)    # to find the positions contain w1
    words_w2 = np.where(words == w2)    # to find the positions contain w2
    # to create a list of paragraphs of 2 words
    para_w_ = dtm[:,(int(words_w1[0]),int(words_w2[0]))]
    # print(para_w_)
    a = np.where(para_w_>0,1,0).sum(axis=1)
    array_para = np.array(paragraphs)
    # to find paragraphs contains both words w1 and w2
    result = array_para[np.where(a>1)[0]]


    return result
```

```
In [14]: ps = find_coocur('chatgpt','students',dtm, words, paragraphs)
         len(ps)
         ps
```

```
Out[14]: 4
```

```
Out[14]: array(['This year, Mollick is not only allowing his students to use ChatGPT,
         they are required to. And he has formally adopted an A.I. policy into his sy
         llabus for the first time.',
                "This week he ran a session where students were asked to come up with
         ideas for their class project. Almost everyone had ChatGPT running and were
         asking it to generate projects, and then they interrogated the bot's ideas w
         ith further prompts.",
                'He cites a survey of Stanford students that found many had already u
         sed ChatGPT in their final exams, and he points to estimates that thousands
         of people in places like Kenya are writing essays on behalf of students abro
         ad.',
                '"I think everybody is cheating ... I mean, it\'s happening. So what
         I\'m asking students to do is just be honest with me," he said. "Tell me wha
         t they use ChatGPT for, tell me what they used as prompts to get it to do wh
         at they want, and that\'s all I\'m asking from them. We\'re in a world where
         this is happening, but now it\'s just going to be at an even grander scale."
         '],
               dtype='<U547')
```

Put everything together and test using main block

In [15]:
```
# best practice to test your class
# if your script is exported as a module,
# the following part is ignored
# this is equivalent to main() in Java

if __name__ == "__main__":

    # Test Question 1
    doc = "it's a hello world!!!\nit is hello world again.\n\nThis is paragr
    print("Test Question 1")
    para_dict, paragraphs = tokenize(doc)
    print(tokenize(doc))


    # Test Question 2
    print("\nTest Question 2")
    dtm, all_words = get_dtm(doc)

    print(dtm)
    print(all_words)


    #3 Test Question 3

    doc = open("chatgpt_npr.txt", 'r').read()

    para_dict, paragraphs = tokenize(doc)
    dtm, all_words = get_dtm(doc)

    print("\nTest Question 3")
    words = np.array(all_words)

    tfidf = analyze_dtm(dtm, words, paragraphs)
```

```
Test Question 1
({0: {"it's": 1, 'hello': 2, 'it': 1, 'is': 1, 'world': 2, 'again': 1}, 1: {
'paragraph': 1, 'is': 1, 'this': 1}}, ["it's a hello world!!!\nit is hello w
orld again.", 'This is paragraph 2.'])

Test Question 2
[[0. 1. 2. 1. 1. 0. 2. 1.]
 [1. 0. 0. 0. 1. 1. 0. 0.]]
['paragraph', "it's", 'hello', 'it', 'is', 'this', 'world', 'again']

Test Question 3
The total number of words:
 686.0

The number of paragraps: 24, the number of unique words in the document: 312
.

The top 10 frequent words:
 [('the', 31.0), ('to', 25.0), ('and', 19.0), ('that', 13.0), ('it', 12.0),
('he', 12.0), ('in', 12.0), ('of', 11.0), ('is', 10.0), ('chatgpt', 9.0)]

The top 10 words with highest df values:
 [('the', 18), ('and', 15), ('to', 14), ('in', 11), ('it', 10), ('that', 9),
('he', 9), ('is', 8), ('chatgpt', 8), ('for', 8)]

The shortest paragraph :
 And yet, not all educators are shying away from the bot.

the top 5 words with the highest tf_idf values in the longest paragraph:
 [('testing', 0.02666666666666667), ('misleading', 0.02666666666666667), ('u
sers', 0.02666666666666667), ('occasionally', 0.02666666666666667), ('openai
/screenshot', 0.02666666666666667)]
```

```
In [ ]:
```