Student Name: Mai Ngo

Course Name and Number: DSC 575 Intelligent Information Retrieval - SEC 801

Assignment 2 - Inverted index

Date: 1/28/2024

1. Position Indices

a. Which document(s) (if any) match each of the following queries where each expression within quotes is a phrase query? At which positions do the queries match?

i. "fools rush in"

```
Matching found in document 2 at position index 1.
Matching found in document 4 at position index 8.
Matching found in document 7 at position index 3.
Matching found in document 7 at position index 13.
```

ii. "fools rush in" AND "angels fear to tread".

```
Matching found in document 4 at position indices 8 and 12.
```

b. There is something wrong with this positional index. What is the problem?

```
Document 7: Duplicate position 15 for terms 'where' and 'in'.
```

Comment

- It was not difficult for me to do this problem manually, took me a bit of time to figure out part b since I need to write the positional index on paper to trace it.
- However, attempted to code the positional index took me a lot of time. I want to store it as a nested dictionary, the code to process posting values as dictionary was very tricky, mainly the step to determine at which character(s) to split or strip. Overall, what I did was correctly extract each positing element into a list, then re-arrange it into a dictionary.
- As for queries, I use basic logic: positional index of first term + 1 must match positional index of the second term, and so on. I just need the correct syntax for dictionary iteration, so the code will not look lengthy.

- As for troubleshoot problem. I basically re-arrange the master dictionary using positional indices as keys. Set a print statement where if there is a term within a document and same index, the code would raise flag.

2. TFiDF Weighting

a. Compute the new weights for all the terms in documents DOC3 and DOC7 using the tf x idf approach. Use the raw term frequency for TF (for both documents and query) and the log of base 10 of N/df for IDF.

Comment

- I already have my own TF-IDF code to start with. For every class I took, most of the times I try to create reusable codes (help me along the way), so when it comes to the same problem I can just use it. I want to state that this is MY OWN CODE, I established it using it my own logic and understanding of the concept.
- What I did was reviewing the concept and calculation of TF-IDF, then match each input/output to each step of my own code.

New weights for all the terms in documents DOC3.

```
doc3 = doc_termTFIDF_pd.loc['Doc3']
print('New weights for all the terms in documents DOC3:')
doc3
```

```
New weights for all the terms in documents DOC3:
Term1    0.464706
Term2    0.000000
Term3    1.204120
Term4    0.903090
Term5    1.591760
Term6    0.000000
Term7    0.000000
Term8    0.774510
Name: Doc3, dtype: float64
```

New weights for all the terms in documents DOC7.

```
doc7 = doc_termTFIDF_pd.loc['Doc7']
print('New weights for all the terms in documents DOC7:')
doc7
```

```
New weights for all the terms in documents DOC7:
Term1    0.309804
Term2    1.505150
Term3    0.000000
Term4    0.903090
Term5    0.000000
Term6    0.221849
Term7    1.204120
Term8    0.309804
Name: Doc7, dtype: float64
```

Steps – Using NumPy Array

Create document-term matrix:

- Import exactly how document-term table comes with. In this case, documents as rows and terms as columns. It is important to determine which attribute is column/row.

First - Tackle IDF

Get document frequency df_t: the number of documents that contain term t.

- Transpose since I need the table to be term-document matrix. Now, terms are row and documents are columns.
- Get count of how many non-zero element(s), applied row wise then reshaped the output into one column vector.

```
Number of documents a term exists in:
[[7]
 [5]
 [5]
 [5]
 [4]
 [6]
 [5]
 [7]]
```

Create base matrix represents total number of document N:

- Generate matrix of ones with size of the term-document matrix. Then multiply each element with N.
- N should just be the total number of columns of term-document matrix aka. shape[1].

```
[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]
```

Calculate IDF following formula from class PowerPoint #6: $idf_t = \log_{10}(N/df_t)$

- IDF = np.log10(np.divide(documentNum_Matrix, docNum_perTerm))

```
IDF:
[[0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549 ]
 [0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103]
 [0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103]
 [0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103]
 [0.39794 0.39794 0.39794 0.39794 0.39794 0.39794 0.39794 0.39794 0.39794 0.39794]
 [0.22185 0.22185 0.22185 0.22185 0.22185 0.22185 0.22185 0.22185 0.22185 0.22185]
 [0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103 0.30103]
 [0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549  0.1549 ]]
```

Second - Calculate TFxIDF with TF as raw term frequency.

```
#Compute the TFxIDF values for document-term entry.
TFIDF = term_docMatrix * IDF
print(TFIDF)
```

```
[[0.      0.77451 0.46471 0.1549  0.      0.3098  0.3098  0.46471 0.      0.1549 ]
 [0.90309 0.      0.      2.40824 0.30103 0.      1.50515 0.90309 0.      0.     ]
 [0.30103 0.      1.20412 0.      0.      0.60206 0.      0.      0.90309 1.50515]
 [0.      0.      0.90309 0.90309 0.      0.      0.90309 0.60206 0.90309 0.     ]
 [0.      1.19382 1.59176 0.      0.      0.      0.      0.      1.19382 0.79588]
 [0.4437  0.      0.      0.22185 1.10924 0.88739 0.22185 0.      0.      0.88739]
 [0.30103 0.      0.      1.20412 1.20412 0.      1.20412 0.30103 0.      0.     ]
 [0.      0.3098  0.77451 0.      0.3098  0.1549  0.3098  0.46471 0.      0.3098 ]]
```

Convert back to same format as the original document-term matrix.

- Note: I have the output print as 5 decimals.
- Convert to pandas data frame because I am more comfortable working with it.

```
columnNames = ['Term1', 'Term2', 'Term3', 'Term4', 'Term5', 'Term6', 'Term7', 'Term8']
indexNames = [f'Doc{i+1}' for i in range(len(doc_termTFIDF))]
doc_termTFIDF_pd = pd.DataFrame(doc_termTFIDF, columns=columnNames, index=indexNames)
print('TF-IDF weights:')
doc_termTFIDF_pd
```

TF-IDF weights:

	Term1	Term2	Term3	Term4	Term5	Term6	Term7	Term8
Doc1	0.000000	0.90309	0.30103	0.00000	0.00000	0.443697	0.30103	0.000000
Doc2	0.774510	0.00000	0.00000	0.00000	1.19382	0.000000	0.00000	0.309804
Doc3	0.464706	0.00000	1.20412	0.90309	1.59176	0.000000	0.00000	0.774510
Doc4	0.154902	2.40824	0.00000	0.90309	0.00000	0.221849	1.20412	0.000000
Doc5	0.000000	0.30103	0.00000	0.00000	0.00000	1.109244	1.20412	0.309804
Doc6	0.309804	0.00000	0.60206	0.00000	0.00000	0.887395	0.00000	0.154902
Doc7	0.309804	1.50515	0.00000	0.90309	0.00000	0.221849	1.20412	0.309804
Doc8	0.464706	0.90309	0.00000	0.60206	0.00000	0.000000	0.30103	0.464706
Doc9	0.000000	0.00000	0.90309	0.90309	1.19382	0.000000	0.00000	0.000000
Doc10	0.154902	0.00000	1.50515	0.00000	0.79588	0.887395	0.00000	0.309804

b. Using cosine as the measure, which document is the most relevant to DOC7? Show your calculations.

```
The most relevant document to Doc7 is Doc4 with a Cosine similarity of 0.9280
```

Comment

- I already have my own Cosine *distance* function code to start with. Again, I want to state that this is MY OWN CODE. What I did was modifying to output Cosine similarity, just removing one line of code. I know that there are other bult-in functions to calculate this, but I like to use my own code.
- I also want to state that I took a lot of times to fully understand and build my own reusable codes, so this doesn't mean I went through this assignment easily. I want to make sure my codes are correctly set up and have something I can confidently use as my own resources.

```
def cosineSimilarity (instance1, instance2):
    '''Return Cosine similarity distance between two instance.'''

    #Find the vector norm for each instance.
    instance1_norm = np.linalg.norm(instance1)
    instance2_norm = np.linalg.norm(instance2)

    #Compute Cosine: divide the dot product of the predicted instance versus each document instance by the product of the two norms.
    cosine = np.dot(instance1, instance2)/(instance1_norm * instance2_norm)

    return cosine
```

Following formula: dot product of 2 instances divided by the product of length-normalized of each instance.

- Then I created a function to iterate over each document instance and calculate Cosine similarity with doc7, this was pretty straight forward. The function is applicable for pandas data frame inputs.

```
def cosine_simDoc(doc, weightMatrix):
    '''Get the most similar document based on Cosine similarity.'''
    '''Using raw term frequnecy and pandas data frame.'''

    cosSim_res = []

    for index, row in weightMatrix.iterrows():
        cosineSim = cosineSimilarity(doc, row.values)
        cosSim_res.append((index, cosineSim))

    #Sort the similarities in descending order.
    cosSim_res.sort(key=lambda sim: sim[1], reverse=True)

    #Return 2nd similar item. First item is the document itself.
    return cosSim_res[1]
```

```
doc_termMatrix_pd = pd.DataFrame(doc_termMatrix)
doc7Raw = doc_termMatrix_pd.iloc[6]
cosineSimilar = cosine_simDoc(doc7Raw, doc_termMatrix_pd)
print(f"The most relevant document to Doc7 is Doc{cosineSimilar[0] + 1} with a Cosine similarity of {cosineSimilar[1]:.4f}")
```

```
The most relevant document to Doc7 is Doc4 with a Cosine similarity of 0.9280
```

3. Inverted Index

Comment

- I use pandas data frame as a benchmark to approach everything in this problem. For example, how can I apply token pre-processing for the 'description' column then assign output to a new column.
- By using pandas, I can basically apply cross tabulations as much as I want. Overall, solve this problem pretty smoothly and efficiently.

Set up pre-processing text per requirement.

- I used the same code in assignment 1, this is to ensure I can keep track of same number of total tokens and unique vocabulary.

```
#Set to store English stop word for efficiency.
stopwordsSet = set(stopwords.words('english'))

#Porter stemmer object.
porterStemmer = nltk.PorterStemmer()

def processDescription(description):
    '''Apply Word tokenization + Case folding (lower-case) + Stopword filtering + Non-alphabet filtering + Porter stemming to a string.'''

    tokens = [porterStemmer.stem(token) for token in word_tokenize(description.lower())
              if token.isalpha() and token not in stopwordsSet]

    #Using Counter for dictionary output.
    return Counter(tokens)
```

Set up master term-document data frame.

- Not just for this one, for every problem, I will pull necessary columns, apply required modifications, and store it as a master data frame to apply cross tabulation and further tasks.
- Extract 'description' and 'url' column.
- For every description, apply text pre-process. Then iterate through each output, store keys (term) and values (term count) as separate columns.
- The goal is to have pdWord represents each term in each document and its respective counts.

```
#Look at processed desciption per row, then iterate through each word.
#Assign to tuple (A SINGLE WORD, corresponding word count within URL document, corresponding URL).

pdWord = pd.DataFrame([(word, word_count, tedShort.loc[index, 'url']) for index, row in tedShort.iterrows()
                       for word, word_count in row['processedDesc'].items()], columns=['word', 'word_count', 'url'])
pdWord.head(3)
```

	word	word_count	url
0	sir	1	https://www.ted.com/talks/ken_robinson_says_sc...
1	ken	1	https://www.ted.com/talks/ken_robinson_says_sc...
2	robinson	1	https://www.ted.com/talks/ken_robinson_says_sc...

a. A file that maps each term name to the term ID and the document frequency (i.e., the number of files the term occurred in). Assign a series of positive integers (1-based) to the term names. Store each term name per line. Name the file "TED_term_index.csv".

Comment

- Apply cross tabulation on pdWord. Group by 'word' and count how many rows are associated to each term. Term_ID is just index + 1.
- Super convenient because 'groupby' automatically sort the grouped attribute in ascending order. In this case, terms are automatically sorted alphabetically.

```
#size() - count occurrences grouped by one word.
pdWordcounts = pdWord.groupby('word').size().reset_index(name='count')
pdWordcounts['term_ID'] = pdWordcounts.index + 1
pdWordcounts.head(3)
```

	word	count	term_ID
0	a	2	1
1	aakash	1	2
2	aala	1	3

CSV file

word, term_ID, count
a, 1, 2
aakash, 2, 1
aala, 3, 1
aamodt, 4, 1

b. A file that maps each document ID to the document name. Assign a series of positive integers (1-based) to the documents. Store each document ID per line. Name the file "TED_doc_index.csv".

Comment

- As straight forward as the requirement. I extracted the 'url' column individually and created doc_ID column as index + 1.

```
#Add document index column.
tedURL['doc_ID'] = tedURL.index + 1
tedURL.head(3)
```

	url	doc_ID
0	https://www.ted.com/talks/ken_robinson_says_sc...	1
1	https://www.ted.com/talks/al_gore_on_averting_...	2
2	https://www.ted.com/talks/david_pogue_says_sim...	3

CSV file

```
doc_ID, url
1, https://www.ted.com/talks/ken_robinson_says_schools_kill_creativity
2, https://www.ted.com/talks/al_gore_on_averting_climate_crisis
3, https://www.ted.com/talks/david_pogue_says_simplicity_sells
4, https://www.ted.com/talks/majora_carter_s_tale_of_urban_renewal
```

c. The inverted index file that maps each term ID to its postings list. Each posting should contain a document ID and the term frequency in that document. Store one term ID per line. Name the file "TED_inverted_index.csv".

Comment

- Using master pdWord, created a copy pdWord_inverted to work with.
- Perform left join with both pandas data frame from previous tasks. Basically, adding term_ID and doc_ID columns.

```
#Create a copy for this problem.
pdWord_inverted=pdWord

#Merge each word to correpoding term_ID.
pdWord_inverted = pdWord_inverted.merge(pdWordcounts[['word', 'term_ID']], on='word', how='left')

#Merge each doc to correpoding doc_ID.
pdWord_inverted = pdWord_inverted.merge(tedURL[['url', 'doc_ID']], on='url', how='left')

#Sorted in ascending order.
pdWord_inverted = pdWord_inverted.sort_values(by=['term_ID', 'doc_ID'])

#Display the updated pdWord DataFrame
pdWord_inverted.head(3)
```

	word	word_count	url	term_ID	doc_ID
25373	a	1	https://www.ted.com/talks/lucien_engelen_crowd...	1	1146
65127	a	1	https://www.ted.com/talks/todd_scott_an_interg...	1	2429
45972	aakash	1	https://www.ted.com/talks/aakash_odedra_a_danc...	2	1878

- Instead of applying further cross tabulation, I decided to create a template for each line to write out to the csv file. I have this thought when I saw your suggested output, not ideal to creating new data frame with empty cells.
- Group by term_ID, combine each doc_ID and associated word count to a tuple. The template would be 'term_ID, doc_ID, word count' with each (doc_ID, word_count) tuple keep appending for each term_ID.

```
invertedIndex_lines = []

#Create template line.
for termID, postings in pdWord_inverted.groupby('term_ID'):
    template = f"{termID}, " + ', '.join(f"{docID}, {wordCount}" for (docID, wordCount) in postings[['doc_ID', 'word_count']].itertuples(index=False))
    invertedIndex_lines.append(template)

with open('TED_inverted_index.csv', 'w', newline='', encoding='utf-8') as outFile:
    outFile.write('\n'.join(invertedIndex_lines))

print("Successfully written!")
```

CSV file

```
1, 1146, 1, 2429, 1
2, 1878, 1
3, 2381, 1
4, 1655, 1
5, 810, 1, 943, 1, 951, 1, 1717, 1
6, 1991, 1
7, 1611, 1
8, 923, 1
9, 973, 1, 1386, 1, 1451, 1, 1604, 1, 1818, 1, 1945, 1, 2129, 1, 2139, 1, 2267, 1, 2337, 1, 2409, 1
```

Boolean conjunctive queries.

Comment

- I use pdWord_inverted pandas data frame from previous part.
- Created a function that stemmed queried terms, and only retrieve rows that containing the stemmed terms. To check if it existed in more than one document, groupby doc_ID and check if there is at least 2 rows per doc_ID. Compile output to a new data frame with matching doc_ID per row, followed by associated url, and both terms with their frequencies.
- This logic was not easy to come up with. I tried several ways and the code ended up very long and redundant (lots of errors). After a while, I think this is the most efficient approach I can get to.

i. 'climate' AND 'change'

```
Number of documents have both terms 'climate' AND 'change' are 35.
```

ii. 'climate' AND 'fuel'

```
Number of documents have both terms 'climate' AND 'fuel' are 1.
```

iii. 'artificial' AND 'intelligence'

```
Number of documents have both terms 'artificial' AND 'intelligence' are 5.
```

iv. 'giant' AND 'troll'

```
Number of documents have both terms 'giant' AND 'troll' are 0.
```

Boolean conjunctive queries – Code verification.

Comment

- I printed out the intermediate data frame to verify by sorting doc_ID in ascending order. My expectation was every doc_ID should have at least 2 rows with each row represents each term and it satisfied.

```
#climateChange2 should give all doc_ID that both query words existed in.
climateChange2.head(4)
```

	word	word_count	url	term_ID	doc_ID
27	chang	1	https://www.ted.com/talks/al_gore_on_averting_...	1561	2
2032	chang	1	https://www.ted.com/talks/john_doerr_sees_salv...	1561	110
2946	chang	1	https://www.ted.com/talks/david_keith_s_surpri...	1561	161
3964	chang	1	https://www.ted.com/talks/al_gore_s_new_thinki...	1561	215

```
#Sort by doc_ID, expectation would be every two rows same doc_ID.
validate = climateChange2.sort_values(by='doc_ID')
#At this point I can confirm my code and logic are correct.
validate.head(10)
```

	word	word_count	url	term_ID	doc_ID
27	chang	1	https://www.ted.com/talks/al_gore_on_averting_...	1561	2
26	climat	1	https://www.ted.com/talks/al_gore_on_averting_...	1739	2
2032	chang	1	https://www.ted.com/talks/john_doerr_sees_salv...	1561	110
2031	climat	1	https://www.ted.com/talks/john_doerr_sees_salv...	1739	110
2946	chang	1	https://www.ted.com/talks/david_keith_s_surpri...	1561	161
2945	climat	1	https://www.ted.com/talks/david_keith_s_surpri...	1739	161
3964	chang	1	https://www.ted.com/talks/al_gore_s_new_thinki...	1561	215
3963	climat	1	https://www.ted.com/talks/al_gore_s_new_thinki...	1739	215
9687	climat	1	https://www.ted.com/talks/gordon_brown\n	1739	492
9688	chang	1	https://www.ted.com/talks/gordon_brown\n	1561	492

Overall comment:

- Through this assignment, I think now you can tell how much I love using pandas data frame. I think it is super-efficient and straight forward to deal with. Almost every approach I can think of, there is a way to tackle it with pandas data frame syntax.
- Once I have a solid master pandas data frame to work with, I just need to think properly how to cross tabulate it, but the code is not hard to establish anymore. So, if I can import and work with data as pandas data frame, I always go for that option.
- I think overall, once I have my data type and structure set up comfortably (to my own understanding), I can start thinking how to solve a problem step-by-step. Make it much easier and simpler for me.