REPORT

Name: Krithika Verma
Campus ID: TR12417

Approach:

I used term and term weights generated in HW2 and the scores were normalized in the range of 0 to 1 using the traditional TF-IDF formula:

TF (w, d) = Number of occurrences of w in document d / Number of words in the document d
IDF(w) = loge ((Total number of documents) / (Number of documents containing the word w)
TF-IDF = TF * IDF

Logic:

I have created the term and term weights and stored in the folder TF_IDF in csv format for ease of working with data frames and pandas. While calculating the TF_IDF I have also stored the 'DocId' of each word in the file with a counter variable.

I have then put all the data of the file in one single file named 'tf_idf_all_files.csv' containing Term, DocId and TfIdf. I have performed this with the concatenation operation using pandas.

```
connect = pd.concat(df1_of_all_files, axis=0)
connect.to_csv('tf_idf_all_files.csv',index =None)
```

	A	B	C
1	Term	DocId	TfIdf
397	build	1	0.005119
398	recognize	1	0.005488
399	pioneer	1	0.00692
400	rare	1	0.007554
401	provincial	1	0.007554
402	website	1	0.005119
403	cdt	2	0.023027
404	testimony	2	0.003804
405	med	2	0.002093
406	privacy	2	0.124273
407	statement	2	0.001292
408	janlori	2	0.006473
409	goldman	2	0.00533

Next, I created a term document matrix from 'tf_idf_all_files.csv' file using pivot_table feature of pandas. I have replaced the **NaN** values to zero using fillna feature.

```python
# converting the word, tf_idf value of the word, DocID into a Term document matrix
df4 = df3.pivot_table(values = 'TfIdf',index= 'Term',columns='DocId')
# replacing NaN values in matrix with value 0
df5 = df4.fillna(value= 0, method= None, axis= None)
```

From this term document matrix, I created a nested dictionary which contains key as the term or token and values as DocId and term TF-IDF weight. The values are another dictionary where key is DocId and weight as the value.

```python
df6 = df5.to_dict('index')
# converting the matrix into nested dictionary
df7 = {k:{k1:v1 for k1,v1 in v.items() if v1 != 0} for k,v in df6.items()}
```

Below is the screenshot of a portion of nested dictionary with 10 files as input.

```
{'aaa': {4: 0.09031014128967096}, 'ab': {5: 0.002343359495280899}, 'abba': {5: 0.003098907569858877}, 'abc': {1: 0.017918
882331100162}, 'aber': {5: 0.00283899403902031}, 'abide': {1: 0.007553587201531014}, 'ability': {2: 0.001180721358750422
4, 7: 0.002333118353171669 3}, 'able': {7: 0.0007584842305919355}, 'abnormal': {2: 0.002875880847835481}, 'abq': {5: 0.002
```

Using this nested dictionary, we can find the posting and dictionary files as it contains all the data needed.

```python
#writing DocID and the normalized weight from the nested dictionary to posting file
with open('posting.txt','w', encoding = 'ASCII', errors = 'ignore') as f6:
    for key, value in df7.items():
        for key1,val1 in sorted(value.items()):
            f6.write(str(key1)+','+str(val1)+'\n')
```

The first for loop iterates over the term and its values (DocId and weight). The second for loop iterates over the items in second dictionary which is nothing but posting list (DocId, weight). The entries are then written onto the posting file.

The frequency of the term was calculated and stored in another dictionary and using the value in this dictionary, the position was calculated. The position is initialized by 1 and further incremented each time by the length of posting list for each term.

```python
for k3, v3 in d.items():
    d[k3] = position
    position = position + v3
    f7.write(str(k3)+'\n'+str(v3)+'\n'+str(d[k3])+'\n')
```

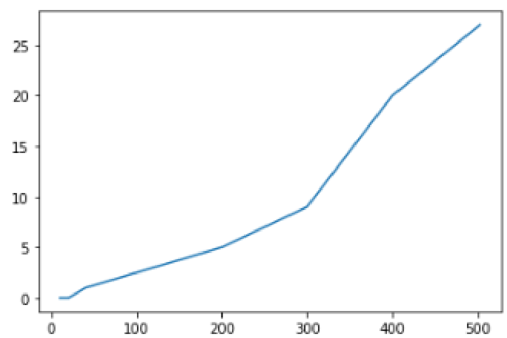
Runtime and Memory Analysis:

Number of files	Execution time (seconds)	Posting file size (kB)	Dictionary file size (kB)	Dictionary + Posting size (kB)
10	0	50	23	73
20	0	90	35	125
40	1	202	61	263
80	2	398	98	493
160	4	725	138	863
200	5	890	155	1045
300	9	1896	224	2120
400	20	4928	615	5543
503	27	6300	797	7097

Num_Files VS Execution Time:

```
plt.plot([10,20,40,80,160,200,300,400,503],[0,0,1,2,4,5,9,20,27])
```

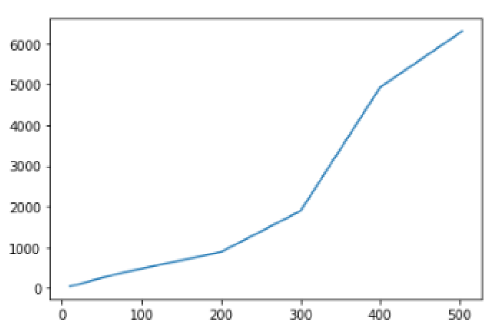
```
[<matplotlib.lines.Line2D at 0x1d124e76e48>]
```



Num_Files VS Postings Size:

```
plt.plot([10,20,40,80,160,200,300,400,503],[50,90,202,398,725,890,1896,4928,6300])
```

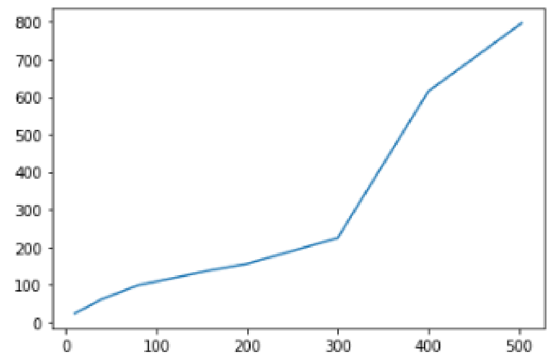
```
[<matplotlib.lines.Line2D at 0x1d125529048>]
```



Num_Files VS Dictionary_Size:

```
plt.plot([10,20,40,80,160,200,300,400,503],[23,35,61,98,138,155,224,615,797])
```

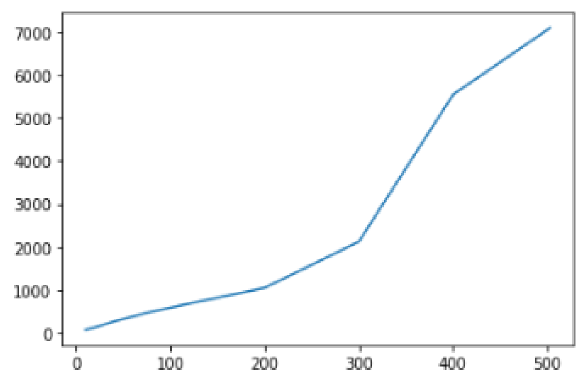
```
[<matplotlib.lines.Line2D at 0x1d12559adc8>]
```



Num_Files VS Total_Output_Size (Posting and Dictionary):

```
plt.plot([10,20,40,80,160,200,300,400,503],[73,125,263,493,863,1045,2120,5543,7097])
```

```
[<matplotlib.lines.Line2D at 0x1d12560ae88>]
```



From the graphs above, we can see that the running time increases as the corpus size increases. A sharp increment is seen between corpus size from 300 to 400. It implies that these documents have more terms (more memory space) compared to others.

The average size of the files between 300 and 400 appears to be 38.13 kB and is 1.9 times the average size of the whole corpus which is 19.6 kB. Hence, such an increment is observed.

Usage Guidelines:

Input: python <file_name> <input_dir> <output_dir>

```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python index.py C:\Users\Vrindavan\Downloads\Krith
iles C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1\tokenized
2020-03-31 02:25:28.968331
Input directory is: C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1\files
output directory is: C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1\tokenized
7
```

Output:

1) Directory containing output files of the tokenized words for each input file. (*\tokenized
2) A directory containing tokens after removal of stop words and TF values of each word (*\stop)
3) 5) A directory containing output files of tokens and term weights (*\TF_IDF)
4) File containing document ID and TF_IDF weight (posting.txt)
5) File containing word, frequency of word in posting file and first location of the term in posting file

dictionary - Notepad

File Edit Format View

```
aa
7
1
aaa
7
8
aaas
2
15
aac
2
17
aachen
4
19
aads
2
23
```

posting - Notepad

File Edit Format View Help

```
266,0.007901441813390805
433,0.0007582446468434131
434,0.0010790559184764422
435,0.0011231424122554983
436,0.0010131374109247913
437,0.0016044375414107762
494,0.07007672165646599
4,0.09031014128967096
12,0.1781116675435177
21,0.0814224765913224
43,0.004519750962100545
225,0.12953575821346747
282,0.0016100489721447928
376,0.0008766776089098495
434,0.00023254840294247944
436,0.00032751336075960137
433,0.00019609206008016863
437,0.0002963776401898012
```