Report

Approach

I have extended my own pre-processor from the previous assignment where I used BeautifulSoup to parse the HTML documents and removed the punctuation by using RegexpTokenizer method from NLTK package. I used python because of the availability of various package which helps me manipulate my Data frames with an ease.

The stop words and the words that appear only once in the entire corpus were removed from the file before calculating TF-IDF values.

TERM WEIGHTHING CALCULATION:

After preprocessing, each output file contained the tokens and the word frequency associated with each token in a csv format.

TF:

I have calculated term frequency based on the traditional formula which states:

TF (w, d) = Number of occurrences of w in document d / Number of words in the document d

Here,

w → The tokens(terms)

d → document(file) which has set of tokens

For ease of processing, I have stored the output of each file after removal of stop words in csv format.

Using the csv format, I can calculate TF of each word based on the count column and the sum of count column (i.e.: length of document) by performing operations on Data Frames using pandas.

	A	B	C
1	term	count	tf
2	htm	1	0.03226
3	battle	3	0.09677
4	creek	3	0.09677
5	st	2	0.06452
6	philip	2	0.06452
7	neri	2	0.06452
8	fraternity	1	0.03226
9	gray	1	0.03226
10	sfo	1	0.03226
11	minister	1	0.03226
12	mail	1	0.03226
13	iserv	1	0.03226
14	net	1	0.03226
15	academy	1	0.03226
16	street	1	0.03226
17	mi	2	0.06452
18	meet	1	0.03226
19	sunday	1	0.03226
20	parish	1	0.03226
21	prayer	1	0.03226
22	capital	1	0.03226
23	ave	1	0.03226
24	ne	1	0.03226

df['tf'] = df['count']. div(df['count']. sum ()) → (In the code)

IDF:

Inverse document frequency is calculated based on:

IDF(w) = log$_e$ ((Total number of documents) / (Number of documents containing the word w))

For calculating IDF, I have created a dictionary which stores key as the word and value as the number of documents containing the word. Iterating over the dictionary, IDF is calculated based on the above formula and stored in the below dictionary. I have used Counter () to store dictionary keys and values.

idf[term]= math.log(float(N) / term_freq) → (In the code)

Where N → Total number of documents (503)

Term_freq → the number of documents containing a word

TF-IDF:

Overall formula is:

TF-IDF = TF * IDF

I have stored the results of TF-IDF in another dictionary and then wrote the term weights of each word in an output file for each input file.

Usage Guidelines:

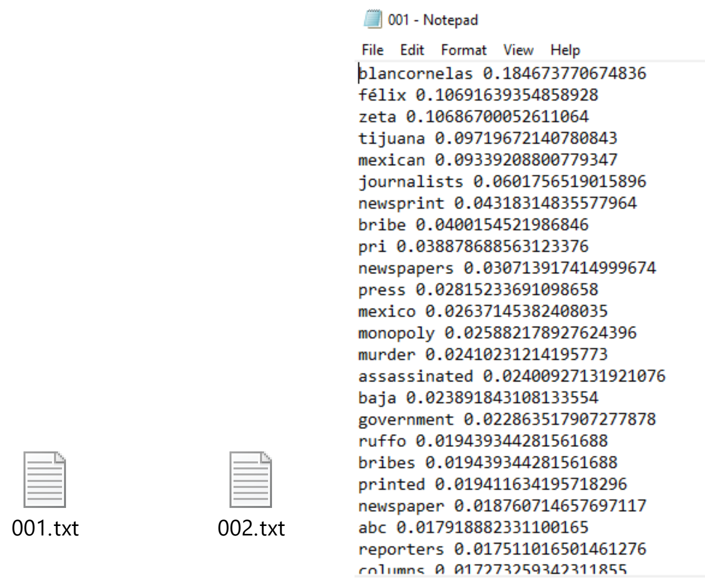
Input: python <file_name> <input_dir> <output_dir>

```
(base) C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1>python tf_idf.py C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1\files C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1\tokenized
2020-03-11 19:38:16.981583
Input directory is: C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1\files
output directory is: C:\Users\Vrindavan\Downloads\Krithika_Verma_HW1\tokenized
34
```

Output:

1) Directory containing output files of the tokenized words for each input file. (*\tokenized)
2) A file of all tokens and frequencies sorted by token (alpha.txt)
3) A file all tokens and frequencies sorted by frequency (freq.txt)
4) A directory containing tokens after removal of stop words and TF values of each word (*\stop)
5) A directory containing output files of tokens and term weights (*\TF_IDF)

Input File (001.html), Output File (001.txt)



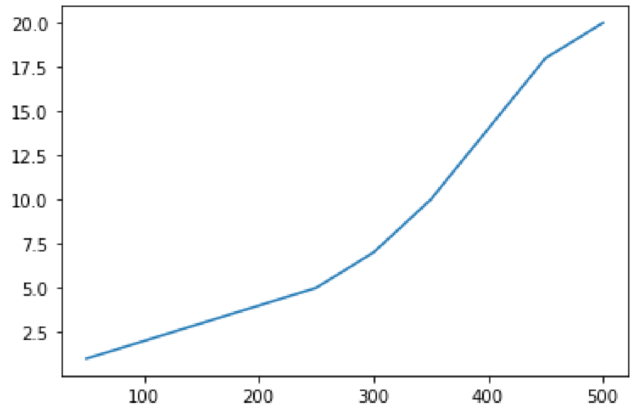

```
blancornelas 0.184673770674836
félix 0.10691639354858928
zeta 0.10686700052611064
tijuana 0.09719672140780843
mexican 0.09339208800779347
journalists 0.0601756519015896
newsprint 0.04318314835577964
bribe 0.0400154521986846
pri 0.038878688563123376
newspapers 0.030713917414999674
press 0.02815233691098658
mexico 0.02637145382408035
monopoly 0.025882178927624396
murder 0.02410231214195773
assassinated 0.02400927131921076
baja 0.023891843108133554
government 0.022863517907277878
ruffo 0.019439344281561688
bribes 0.019439344281561688
printed 0.019411634195718296
newspaper 0.018760714657697117
abc 0.017918882331100165
reporters 0.017511016501461276
columns 0.017273259342311855
```

Runtime Analysis:

Number of Input Documents	Time Takes (in seconds)
50	1
100	2
150	3
200	4
250	5
300	7
350	10
400	14
450	18
50	20

```
In [4]:  import matplotlib.pyplot as plt
```

```
In [5]:  plt.plot([50,100,150,200,250,300,350,400,450,500], [1,2,3,4,5,7,10,14,18,20])
Out[5]:  [<matplotlib.lines.Line2D at 0x7feed77ef940>]
```




```
In [ ]:
```

<u>Effect of stop words:</u>

In previous assignment, the number of tokens present after tokenization in file 1 (001.txt) was 1308 and after removal of stop words, the tokens have reduced to 401

Similarly, in file 2 (002.txt) the number of tokens were 1809 and the current generated file has 647 tokens.

Same effect was seen in all other files. Hence the effect of removing stop words was clear.

Attaching the following folders and files for your reference:

1) Tokenized folder containing tokens
2) Alpha file containing tokens and frequencies sorted alphabetically
3) Freq file containing tokens and frequencies sorted by frequency
4) Tokens without stop words folder (includes TF values as well)
5) TF_IDF (term weights) folder containing the term and its weight output file for each input file