

# Homework 1 Report

*Paul Sampson Ledala*

*[pledala1@umbc.edu](mailto:pledala1@umbc.edu)*

*Department of Computer Science and Electrical Engineering*

*University of Maryland, Baltimore County*

**Date:** 2/11/21

## **I. Introduction**

Tokenization is the process of converting a piece of text into a smaller unit called a token. In this homework we were supposed to create tokens of the input files given. The input files were arranged in a directory called 'files'. These were HTML files which needed prior conversion to text from HTML format to be converted into tokens. The following report explains how different alphanumeric tokens were handled, challenges encountered while programming, analysis of the results and comparative study of another student's working model.

## **II. Specifications of the Program**

Input: Directory of HTML files named 'files'

Output: Directory of tokenized text files, a file called 'Vocabulary' containing sorted tokens with frequencies, a file called 'Token\_Frequency' of all the tokens sorted by frequency

Example command to run the code including input and output paths:

```
python tokenize_script.py
```

```
C:\Users\p\Documents\Coding\Github\IR_Tokenization\Information_Retrieval\files
```

```
C:\Users\p\Documents\Coding\Github\IR_Tokenization\Information_Retrieval\output
```

(includes a space between tokenize\_script.py and the input path, and a space between input and output path)

Coding Language used: Python 3.7.1

Parser used: BeautifulSoup

Tokenizer used: wordpunkt tokenizer of the nltk package

## **III. Methods Used in the Program:**

### **A. Parser**

The Parser used in the program was BeautifulSoup. It is a versatile package which can parse HTML and XML files. It is predominantly used in Web Scraping where the HTML

is converted into text strings by constructing a parse tree for parsed pages and then extracting data. I used the `html.parser` which is the standard python parser (included in Python's standard library). I used it because it is lenient as well as quick for newer versions of python. As I was using python 3.7.1, this parser was ideal for me.

## **B. Tokenizer**

Having tried both `nlk.word_tokenize` and `nlk.tokenize.wordpunct_tokenize` I settled on the latter. The `word_tokenize` tokenizer is based on `TreebankWordTokenizer` and `wordpunct_tokenize` tokenizer is based on the simple regex tokenizing where any special character will be stored in the next token. For example: To split the phrase "I'm a", `word_tokenize` would return `["I", "'m", "a"]` whereas `wordpunct_tokenize` would return `["I", "'", "m", "a"]`. Hence the special character (apostrophe) would be stored in a separate token rather than in the token including text. There was an issue faced while using the `wordpunct_tokenize` tokenizer which will be explained in the challenges section which ultimately helped in deciding to go with `nlk.tokenize.wordpunct_tokenize`.

## **C. Punctuations and Numbers:**

Both punctuation and numbers were removed from the list of tokens while writing into the file. As the `wordpunct` tokenizer was used, this separated the punctuations into a separate token. This was included in the tokens list but eradicated before copying the tokens into the output files. This was done intentionally as the punctuations by themselves wouldn't convey significant information. There was a filter for numbers too as numbers by themselves do not hold significance. These can be included in the output by removing the code filter.

## **D. Frequency Calculations:**

The tokens were counted using the `Counter` functionality of the python's `collections` library. Basically this is a dictionary data structure which counts the number of times the tokens are repeated in the documents storing the token in the key and the frequency in the value. `Counter` provides an efficient way to calculate the frequency as it is internally optimized and can handle large amounts of data and different iterables. It can also be sorted by key or by frequency easily.

# **IV. Incorrect Tokens**

Based on the tokenizer used (`wordpunct`), the tokens were split in such a way that punctuations were placed in separate tokens. This made the words having an apostrophe be replaced by 3 tokens. One for the text before the apostrophe, one for the apostrophe and one for the text after the apostrophe. This leads to 2 issues. One being a possessive term being stored as a noun/pronoun. The other being the text after the apostrophe being stored needlessly. I think the second issue can be handled by removing the terms `'s`, `'re`. Any one letter token can be removed from the tokens list. The stopwords could also be potentially removed in further processing. But the primary issue cannot be solved by this approach. Example: "Mexico's"

would be stored in the 'Mexico' token rather than a separate token called 'Mexicos' which can better indicate the possessive term. This issue could be resolved by using a different tokenizer.

## V. Algorithm Analysis

The program's runtime complexity depends on the following methods as they are inside the loop which iterates through all the input files:

1. BeautifulSoup
2. wordpunct\_tokenize
3. Counter

BeautifulSoup's HTML parser uses a DOM tree based approach where traversing the tree is the most complex action which takes  $O(n)$  time complexity (where  $n$  is the number of tags in HTML).

Constructing a Counter (among all the dictionary operations) takes the maximum runtime complexity which is  $O(n)$  (where  $n$  is the number of keys inserted).

Wordpunct tokenizer is a regular expression based tokenizer which takes  $O(n)$  (where  $n$  is the number of characters in the input)

Using this information, we can calculate that the most time consuming part is the tokenizer which takes  $O(m)$  where  $m$  is the number of characters in the input file of the tokenizer.

This  $O(m)$  is the runtime complexity per input file. If there are  $n$  input files in the program, and the largest file has  $m$  characters, then the runtime time complexity can be approximated to  $O(nm)$ .

Worst Case Time Complexity =  $O(nm)$ , where

$n$  is the number of input html files

$m$  is the number of characters in the largest input html file

## VI. Calculated Run-Time

Time module of python was used to calculate the run-time elapsed in the program. The elapsed time for the total program is 14.845 seconds. The processor used was I5. Refer Fig 1.

The tokenizing time taken per 50 input documents is

Input Iteration	Time Elapsed
0th	0.006
50th	0.365

100th	0.609
150th	0.876
200th	1.273
250th	1.840
300th	2.485
350th	3.362
400th	4.748
450th	13.999
500th	14.459

We see that there is a big time jump between 400th and 450th times. This can be attributed to the fact that the size (number of characters and file size) of the input html files between 400 and 450 is the highest (refer fig 2). Thus the input files are of very different sizes which makes it hard for plotting a graph and getting a best fit line for  $O(nm)$  (where  $n$  is the number of files and  $m$  is the number of characters in the largest file). We can thus experimentally say that  $O(m_1+m_2+m_3....m_n)$  is a better runtime complexity calculation (where  $m_1$  is the number of chars in file 1,  $m_n$  is the number of characters in the file  $n$ ).

## VII. Challenges

The HTML had encodings for the latin apostrophe ('0x91') which would be embedded into the text. For example "Mexico's" would be embedded as "Mexico0x91s" which was misclassified by the word\_tokenize which would store the tokens as ["Mexico", "0x91s"]. It would be hard to get rid of the second token without losing data especially when the apostrophe is used as a single quote. To workaround this I had to use the wordpunct\_tokenize tokenizer. (Refer fig. 5)

## VIII. Comparative analysis

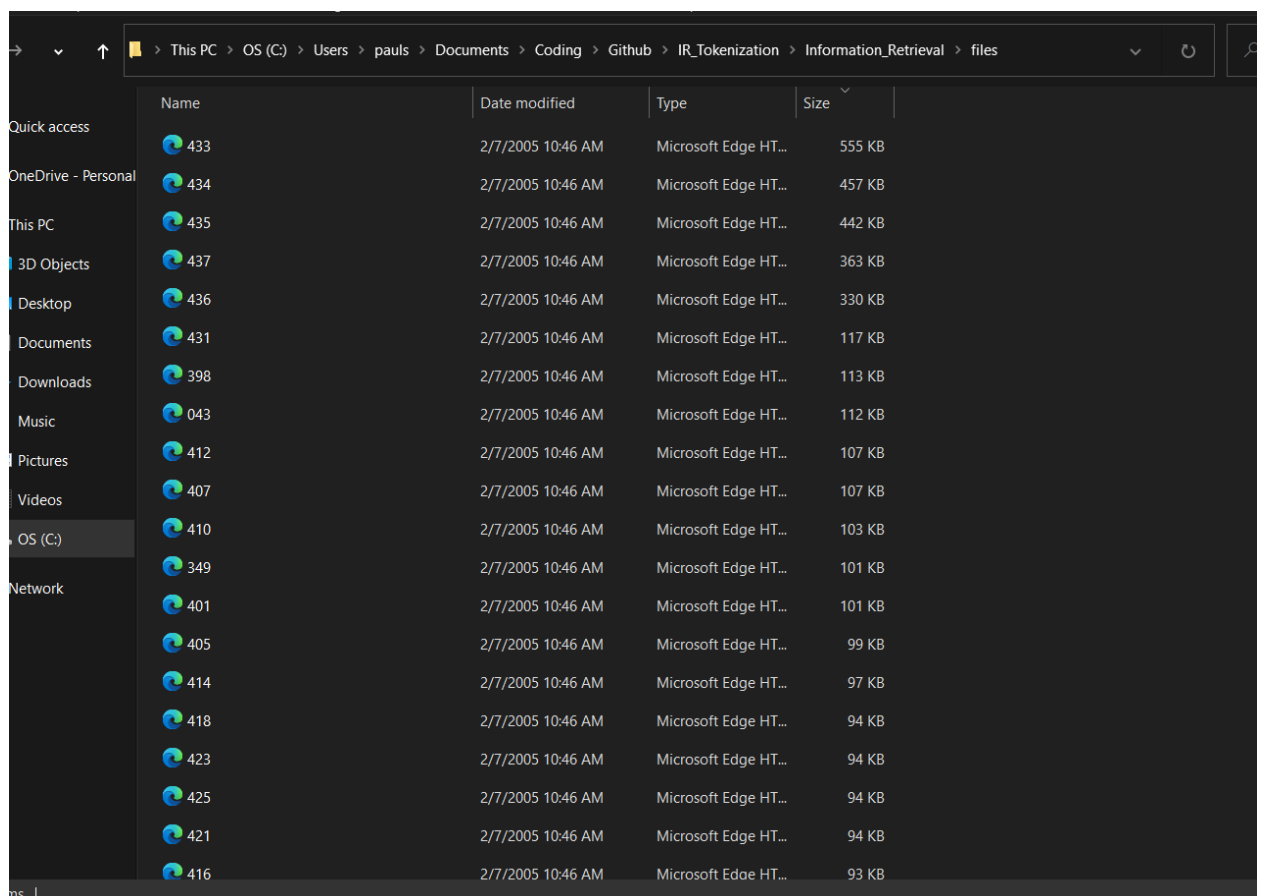
I compared my work with Pragati Mynampati's. She built a similar model using BeautifulSoup for parsing but used a general regular expression for tokenizing. I used a regex method too but it is an inbuilt nltk package (wordpunct.tokenize) which would be more optimized and time-saving. The results show this difference- my program takes 14.845 secs whereas hers tokenizes in 20.298 secs (refer fig 1 and fig 6). My program is also efficient in removing the underscores (refer figures 3 and 8) and other special characters which can be seen in her token frequency output file sorted by tokens. My frequency file also returns a greater count of the tokens than Pragathi's output file (refer fig. 4 and fig. 7)

## IX. Appendix

```
lly wanted the numpy scalar type, use np.float64 here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
eps=4 * np.finfo(np.float).eps, n_jobs=None,
Time Taken by the 0 document is --- 0.006996870040893555 seconds ---
Time Taken by the 50 document is --- 0.36544013023376465 seconds ---
Time Taken by the 100 document is --- 0.6092016696929932 seconds ---
Time Taken by the 150 document is --- 0.8765201568603516 seconds ---
Time Taken by the 200 document is --- 1.2734243869781494 seconds ---
Time Taken by the 250 document is --- 1.840122938156128 seconds ---
Time Taken by the 300 document is --- 2.485309362411499 seconds ---
Time Taken by the 350 document is --- 3.3625261783599854 seconds ---
Time Taken by the 400 document is --- 4.748638868331909 seconds ---
Time Taken by the 450 document is --- 13.999921321868896 seconds ---
Time Taken by the 500 document is --- 14.459736108779907 seconds ---
Time Taken by the program--- 14.844853639602661 seconds ---
```

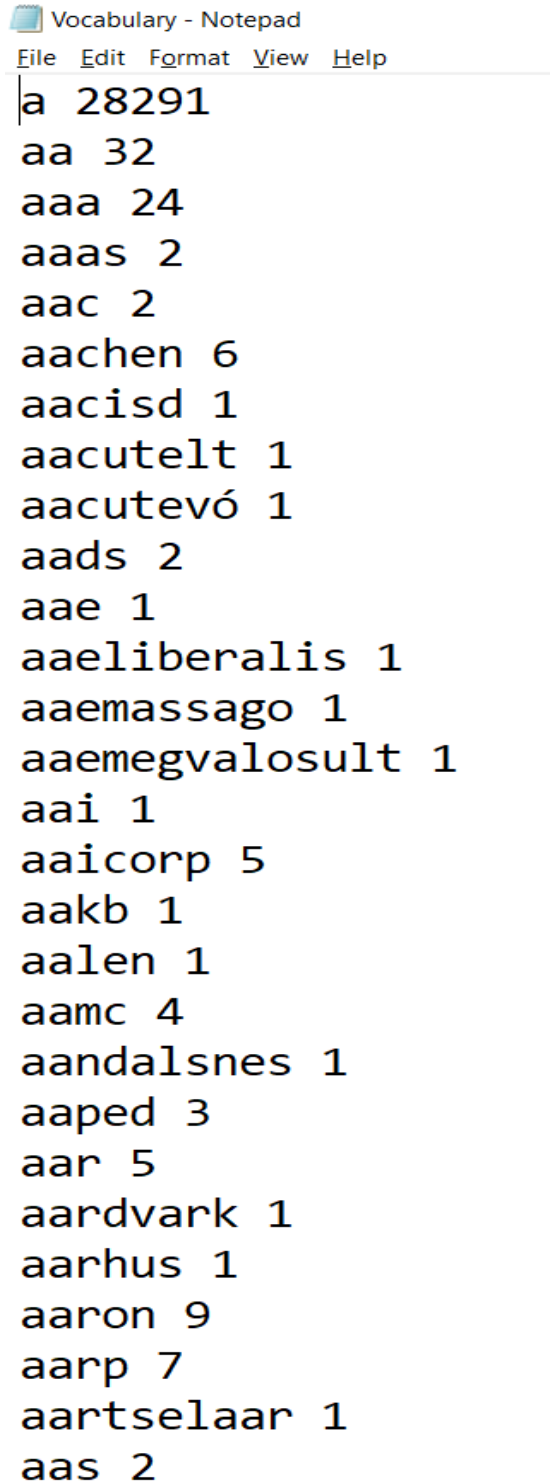
C:\Users\pauls\Documents\Coding\Github\IR\_Tokenization\Information\_Retrieval>

Fig 1: Time taken by the tokenize\_script program to tokenize document. The time is cumulative and increases in proportion to input file size.



Name	Date modified	Type	Size
433	2/7/2005 10:46 AM	Microsoft Edge HT...	555 KB
434	2/7/2005 10:46 AM	Microsoft Edge HT...	457 KB
435	2/7/2005 10:46 AM	Microsoft Edge HT...	442 KB
437	2/7/2005 10:46 AM	Microsoft Edge HT...	363 KB
436	2/7/2005 10:46 AM	Microsoft Edge HT...	330 KB
431	2/7/2005 10:46 AM	Microsoft Edge HT...	117 KB
398	2/7/2005 10:46 AM	Microsoft Edge HT...	113 KB
043	2/7/2005 10:46 AM	Microsoft Edge HT...	112 KB
412	2/7/2005 10:46 AM	Microsoft Edge HT...	107 KB
407	2/7/2005 10:46 AM	Microsoft Edge HT...	107 KB
410	2/7/2005 10:46 AM	Microsoft Edge HT...	103 KB
349	2/7/2005 10:46 AM	Microsoft Edge HT...	101 KB
401	2/7/2005 10:46 AM	Microsoft Edge HT...	101 KB
405	2/7/2005 10:46 AM	Microsoft Edge HT...	99 KB
414	2/7/2005 10:46 AM	Microsoft Edge HT...	97 KB
418	2/7/2005 10:46 AM	Microsoft Edge HT...	94 KB
423	2/7/2005 10:46 AM	Microsoft Edge HT...	94 KB
425	2/7/2005 10:46 AM	Microsoft Edge HT...	94 KB
421	2/7/2005 10:46 AM	Microsoft Edge HT...	94 KB
416	2/7/2005 10:46 AM	Microsoft Edae HT...	93 KB

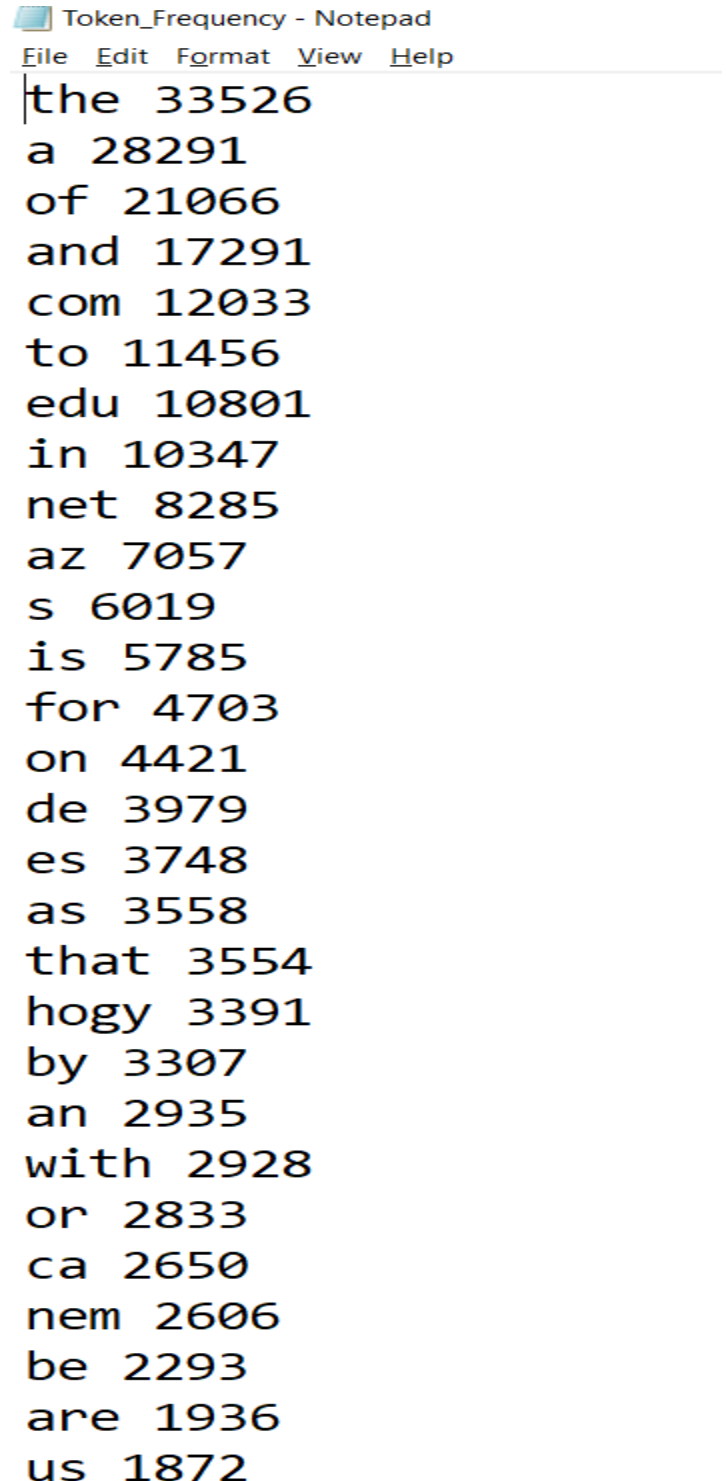
Fig 2: Input files sorted by file size shows that the files in 400-450 range are the biggest, which explains why 400-450 files iteration takes the maximum amount of time



Vocabulary - Notepad  
File Edit Format View Help

a	28291
aa	32
aaa	24
aaas	2
aac	2
aachen	6
aacisd	1
aacutelt	1
aacutevó	1
aads	2
aae	1
aaeliberalis	1
aaemassago	1
aaemegvalosult	1
aai	1
aaicorp	5
aakb	1
aalen	1
aamc	4
aandalsnes	1
aaped	3
aar	5
aardvark	1
aarhus	1
aaron	9
aarp	7
aartselaar	1
aas	2

Fig 3: Tokens sorted in the alphabetical order



Token\_Frequency - Notepad  
File Edit Format View Help

the	33526
a	28291
of	21066
and	17291
com	12033
to	11456
edu	10801
in	10347
net	8285
az	7057
s	6019
is	5785
for	4703
on	4421
de	3979
es	3748
as	3558
that	3554
hogy	3391
by	3307
an	2935
with	2928
or	2833
ca	2650
nem	2606
be	2293
are	1936
us	1872


Fig 4: Tokens sorted according to frequencies

, 'an', 'award', 'this', 'year', 'to', 'Blancornelas', '.', 'THE', 'UPHILL', 'CLIMB', 'TO', 'ESTABLISH', 'AN', 'I  
 NDEPENDENT', 'VOICE', 'IN', 'TIJUANA', 'Blancornelas', 'and', 'Félix', 'launched', 'Zeta', 'soon', 'after', 'losin  
 g', 'control', 'of', 'another', 'Tijuana', 'newspaper', ',', 'ABC', ',', 'which', 'they', 'had', 'started', 'in',  
 '1977', '.', 'The', 'daily\x92s', 'hallmark', 'was', 'in-depth', 'reporting', 'of', 'government', 'corruption', '.  
 ', 'In', 'late', '1979', ',', 'the', 'paper', 'was', 'taken', 'over', 'by', 'a', 'government-controlled', 'labor',  
 'union', 'in', 'a', 'move', 'orchestrated', 'by', 'Baja', 'Gov', '.', 'Roberto', 'de', 'la', 'Madrid', ',', 'a',  
 'member', 'of', 'the', 'PRI', '.', 'The', 'takeover', 'occurred', 'after', 'Blancornelas', 'had', 'refused', 'de',  
 'la', 'Madrid\x92s', 'demand', 'that', 'he', 'fire', 'Félix', ',', 'the', 'paper\x92s', 'star', 'columnist', '.',  
 'Fearful', 'that', 'Zeta', ',', 'which', 'picked', 'up', 'where', 'ABC', 'left', 'off', ',', 'would', 'be', 'a',  
 'target', 'of', 'government', 'reprisal', ',', 'Blancornelas', 'decided', 'to', 'edit', 'and', 'publish', 'the', 'p  
 aper', 'from', 'San', 'Diego', 'for', 'a', 'couple', 'of', 'years', 'to', 'avoid', 'a', 'shutdown', 'by', 'Mexica  
 n', 'authorities', ',', 'who', 'controlled', 'a', 'newsprint', 'monopoly', '.', 'Meanwhile', ',', 'Félix', 'stayed  
 'in', 'Tijuana', ',', 'writing', 'irreverent', 'columns', 'about', 'the', 'backroom', 'happenings', 'of', 'soci  
 al', 'and', 'political', 'life', 'in', 'Baja', 'California', '.', 'FELIX', 'MURDER', 'ENRAGES', 'MEXICAN', 'PUBLIC  
 ', 'In', '1988', ',', 'Félix', ',', 'who', 'served', 'as', 'co-editor', 'of', 'Zeta', ',', 'was', 'assassinated',  
 'while', 'driving', 'to', 'work', 'along', 'a', 'narrow', 'Tijuana', 'street', '.', 'Outrage', 'followed', 'his',  
 'murder', '.', 'Thousands', 'attended', 'the', 'funeral', ',', 'and', 'an', 'unusual', 'alliance', 'of', 'Tijuana',  
 ', 'journalists', 'and', 'political', 'officials', 'demanded', 'justice', '.', 'They', 'would', 'not', 'allow', 'th  
 is', 'murder', 'to', 'be', 'left', 'unsolved', 'as', 'had', 'those', 'of', 'many', 'other', 'Mexican', 'journalist  
 s', '.', 'Two', 'employees', 'of', 'a', 'Tijuana', 'racetrack', 'owned', 'by', 'the', 'son', 'of', 'a', 'powerful',  
 'PRI', 'politician', 'and', 'power', 'broker', 'were', 'eventually', 'convicted', 'of', 'the', 'shooting', ',',  
 'and', 'they', 'are', 'currently', 'serving', 'prison', 'sentences', '.', 'But', 'authorities', 'never', 'establis  
 hed', 'a', 'motive', 'for', 'the', 'assassination', ',', 'and', 'it', 'is', 'widely', 'believed', 'the', 'intellec  
 tual', 'authors', 'remain', 'at', 'large', '.', 'In', 'the', 'wake', 'of', 'Félix\x92s', 'murder', ',', 'Blancorne  
 las', 'continued', 'to', 'print', 'his', 'colleague\x92s', 'name', 'on', 'the', 'masthead', 'of', 'Zeta', 'and', '  
 ', 'in', 'each', 'edition', ',', 'reproduced', 'one', 'of', 'Félix\x92s', 'columns', 'opposite', 'a', 'black', 'p  
 age', 'that', 'asked', ',', 'Who', 'killed', 'me', '?', 'In', 'the', 'year', 'following', 'Félix\x92s',  
 'murder', ',', 'several', 'journalists', 'found', 'their', 'careers', 'threatened', 'when', 'they', 'criticized',  
 'how', 'the', 'case', 'was', 'being', 'handled', '.', 'Five', 'reporters', 'were', 'fired', 'from', 'the', 'dail  
 y', 'El', 'Herald', 'after', 'they', 'published', 'a', 'special', 'edition', 'critical', 'of', 'the', 'investigati  
 on', '.', 'The', 'five', 'later', 'formed', 'their', 'own', 'weekly', 'journal', ',', 'Cambio', '21', '.', 'Two',

Fig 5: The encoding issue can be seen several times in this screengrab.  
 Example: 'daily\x92s' in line 4 3rd word.

number of documents	Time elapsed(seconds)
100	01.012435
200	02.601404
300	04.079624
400	12.115761
500	20.298059

Fig 6: Time taken by Pragathi's program (which uses regular expressions), taking 5 more seconds than my code which uses wordpunct\_tokenize() of the nltk package.




```

the 33416
a 27548
of 21010
and 17249
s 12395
com 12059
to 11398
edu 10802
in 10309
net 8325
az 6608
yr 6435
pgs 6433
fn 5905
is 5669
for 4670
on 4393
de 3949
as 3540
that 3519
es 3501
by 3297
hogy 3199
an 2933
with 2913
or 2833
ca 2653
nem 2450
be 2264
are 1929

```

Fig 7: Pragathi's frequency count file returning lesser tokens than my proposed program. 'The' is counted 33526 times in my tokens file as opposed to 33416 in her file.



```

_ 1
_ 2
_ 1
_ 4
_ 1
_ 1
_ 1
_ 1
_ the 1
_ date 1
_ tar 1
_ especially_ 2
_ nextpart__bbc 3
a 27548
aa 37
aaa 24
aaaaaamaigaaaaaacwapaadadyaaaaaaqaaevnwaracaabaaaajwaaafjfoibh 1
aaaaaaawagaaaaaabaabttvrqaaaaabazabaaaajqaaenmtfluybtkqtrehfvsubtslvw 1
aaaaaabmaaaaaaabaabttvrqaaaaabahwwbaaaaaahhdldzaymmmdaw 1
aaaaealktqibvtfs 17
aaas 2
aac 2
aachen 6
aacisd 1
aads 2
aae 1
aaeliberalis 1
aaemassago 1
aaemegvalosult 1
aai 1

```

Fig 8: Pragathi's tokens file returning special characters (underscores)