**Information Retrieval Programming Phase 1 (Divya Natolana Ganapathy)**

The assignment requires the given 503 html files to be processed. This includes tokenizing the words and placing each html file’s words in a separate text file. For this,

1. **Reading the input file**

The file directory needs to be read first. The directory is opened, and the files are listed. This is done using os.listdir . The input path directory is taken from the user as a system argument (sys arg[1]. This is performed by the function readFromDirectory. This function takes variable input path and returns the list of filenames present in that directory

1. **Html parser**

An html parser is used to convert the html file into a text file. To read each html file the encoding ISO-8859-1 is used instead of utf-8 because a few characters in the html files through errors if utf-8 is used for encoding it. This is then parsed using html.parser provided by the BeautifulSoup Library. BeautifulSoup is used for the conversion of XML to html .It understand the html format and removes the tags . for e.g. if only text needs to be read from an article then the variable.get\_text would fetch only the text part. It is performed by function htmlParser which takes list of files as input and returns the text from all those files as a list.

1. **Tokenize**

The output of the previous function is a chunk of plain text. This text is now split to obtain each and every token. For this nltk.word\_tokeinzer was used . This is a function provided by nltk(natural language toolkit) to split a string into substrings of words. It removes the overhead of specifying the split with white spaces, and full stops. Even characters like ‘,’ , ‘ ?” etc. are considered as separate tokens .

There is another approach where str.split can be done, but the disadvantage of this is that we need to specify the delimiter like tab , \n etc. and it does not understand splitting in linguistic approach for egg . ‘America’ and ‘,’ would be tokens from nltk tokenizer and ‘America,’ would be the token obtained if split is used with delimiter as space.

Hence preferred approach is tokenizer

The function used is wordTokenizer which takes plain text and custom stopwords as parameters and returns list of tokenized words

Another approach would be to use a hash table to store the word when this would be the key and the value associated with it would be the frequency(used later).

1. **Write tokens to a file**

Tokens from every file is written on to a corresponding text file. A new directory OutputDir is created to store these files. A new file is created using file handling technique. The file name is fetched from step one and the .html is replaced with .txt to form the new output file.

Therefore, now there are same number of output files as are input files (503). The path of the output directory is passed as a system argument (sys arg[2]) . In the function generateOutput

3 parameters are passed the complete list of tokenized words, destination directory path (sys arg[2]), input file directory(to fetch the name of the corresponding input file).

1. **Cleaning of Data**

Once the tokens are obtained, it is noticed that the tokens are not just words. There are many numbers ,characters like ‘,’ ,’ ‘ ‘. Then, there are a lot of words which repeat several times in a document but do not give any insight towards the document meaning. These words could include ‘the’ ,’because’ etc. These could be eliminated from the tokens.

Nltk corpus contains a set of commonly agreed upon stopwords, which could be imported and used to remove it from the total set of token. This reduces the unnecessary weightage given to stopwords in a string or document.

words = [word for word in words if word not in stopWords]

for this assignment,stopWords is a combination of nltk stopwords and punctuations.Apart from this any other words that we think is not required in the final tokens can also be added to this list and make a custom stopwords list. If the approach of not using nltk stopwords iand instead explicitly putting in the words to be removed is used then there are chances of missing out a few

and somehow a domain knowledge of the document would be required.

Punctuations can be imported from String. This is a critical step because each punctuation gets counted as a token just increasing the size of our output file.

Numbers can be removed by including only the tokens which are not word.isnumeric.

The decimal numbers can be removed using regress . ('^[0-9]+\\.[0-9]+$) . If only (not word.isnumeric is used and the decimal number removal using regular expression mentioned above is not used the we could end up in a situation where ‘0’ will be removed but ‘0.00’ will not be removed.

Words with length 1 are removed as this could be a typing error or just a ‘a’. All the words are down cased as mentioned in the question so that ‘The’ and ‘the’ are not treated as separate words, which could be easily possible if the words are not converted to lower case.

1. **Tokens with sorted frequencies**

We can count the occurrence of each token to see what is the token that is being repeated the maximum number of times. This could give us insight on the kind of data we are dealing with.

There are a few approaches to do it. A one can be added every time a word occurs and then the sum of the 1’s can be commuted for same words. This would give us an array of word and its frequency. A counter can be used for every token to count the frequency.

Another approach is to use the FreqDist class of nltk, which does the same.

FreqDist provides a few functions like most\_common(n) which can be used to obtain the frequency distribution of the most common n terms.

Here is where removing the stopwords and punctuation helps. If stopwords and punctuations are not removed then the most frequent word in most English document would be ‘the’,which gives absolutely no information about the document. FreqDist by default gives the result in frequency sorted way. First 50 obtained by fdist.most\_common () function or can be obtained by array[50 :] and last 50 tokens can be obtained by array[:50].

One more way of doing this would be a situation where hash table is used. In the hash table we can easily sort with respect to the keys or the values according to the requirement.

1. **Sorted tokens and frequency**

The tokens and frequencies list are put into an array and then sorted, this would sort the tokens in alphabetical order. The function used is freqDistribution. The input parameters to this function are the array of word and frequency, the destination path of the 2 output folders. One each to place the output of 6 and output of 7. First 50 tokens obtained by fdist.most\_common(50) or array[50 :] and last 50 by array[:50].

The calculation of frequency takes 2 seconds with the use of nltk and around 7 seconds without it.

The total running time of this program is O(n) and in this case is approx.. 10 seconds .In a program where no nltk was used and hash map for tokens and counter for frequency is used ,the run time is around 20 seconds.

Output from approach 1

Output for 6:

('0.00', 62496)('0.01', 14863)('az', 6877)("''", 4134)("'s", 4125)('0.02', 4069)('``', 3554)('es', 3457)('hogy', 3386)('de', 3367)('--', 3270)('nem', 2570)('0.03', 1808)('0s', 1766)('0fn', 1435)('magyar', 1202)('meg', 1199)('el', 1169)('7pgs', 1101)('62.65', 1101)('examines', 1076)('analysis', 1067)('79yr', 1062)('0.05', 1014)('en', 1012)('egy', 974)('80yr', 967)('la', 963)('new', 962)('0.04', 949)('law', 826)('53.70', 809)('6pgs', 807)('81yr', 801)('ha', 783)('may', 752)('ez', 687)('5pgs', 680)('44.75', 680)('political', 675)('szerint', 672)('also', 658)('78yr', 652)('government', 648)('10pgs', 647)('89.50', 647)('azt', 629)('csak', 629)('paper', 618)('one', 616)

('machevsky', 1)('giga', 1)('mis', 1)('persist', 1)('gartner', 1)('intelligently', 1)('client-side', 1)('tuner', 1)('back-end', 1)('toolkit', 1)('bongo', 1)('assembling', 1)('activex', 1)('eagerly', 1)('silicon', 1)('polese', 2)('microsystems', 1)('payne', 1)('hoff', 1)('shaio', 1)('kleiner', 1)('caufield', 1)('byers', 1)('intranets', 1)('tv/web', 1)('intelliquest', 1)('internet/online', 1)('fifty-four', 1)('16-and-over', 1)('ventured', 1)('statistic', 1)('exploit', 1)('avid', 1)('college-educated', 1)('small-business', 1)('webmasters', 1)('bookmarks', 1)('freeware', 1)("'/pages/top20.html", 1)('/pages/top20.html', 2)('watermarking', 1)('copyrighted', 3)('watermarkcreated', 1)('watermark', 6)('pixels', 1)('rescanned', 1)('photoshop', 1)('picturemarc', 1)

When compared with a program where stopwords were not removed the word ‘the’ occurred around 150000 times. The highest values in this program is of 0.00 which is 62496. This changed to az when decimal numbers were removed.

Output for 7:

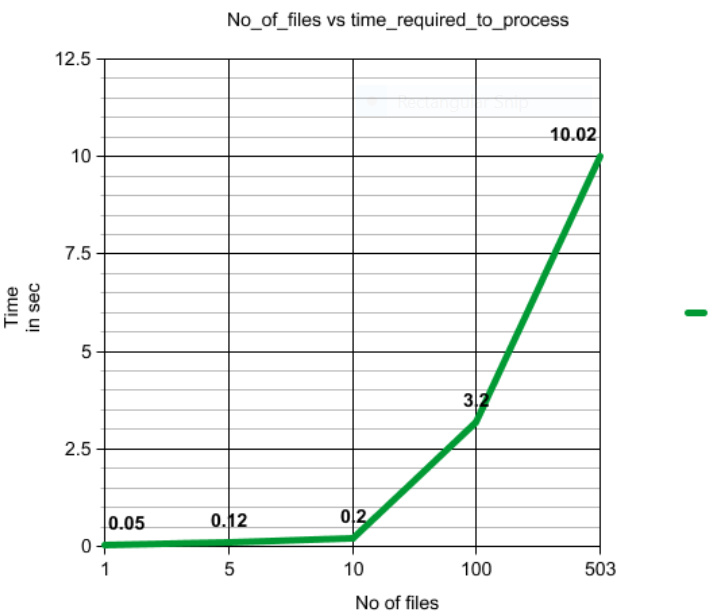
('az', 6877)("''", 4134)("'s", 4125)('``', 3554)('es', 3457)('hogy', 3386)('de', 3367)('--', 3270)('nem', 2570)('0s', 1766)('0fn', 1435)('magyar', 1202)('meg', 1199)('el', 1169)('7pgs', 1101)('examines', 1076)('analysis', 1067)('79yr', 1062)('en', 1012)('egy', 974)('80yr', 967)('la', 963)('new', 962)('law', 826)('6pgs', 807)('81yr', 801)('ha', 783)('may', 752)('ez', 687)('5pgs', 680)('political', 675)('szerint', 672)('also', 658)('78yr', 652)('government', 648)('10pgs', 647)('azt', 629)('csak', 629)('paper', 618)('one', 616)('text/plain', 612)('information', 606)('5s', 606)('6s', 587)('discusses', 586)('mar', 585)('two', 581)('social', 580)('american', 580)('state', 576)

('újrafölosztása', 1), ('újságírók', 1), ('última', 5), ('último', 3), ('últimos', 5), ('ún', 5), ('única', 2), ('únicamente', 1), ('únicas', 1), ('único', 2), ('úr', 2), ('úri', 1), ('út', 1), ('útiles', 3), ('útiokmány', 1), ('útjába', 1), ('útján', 4), ('útlevelet', 2), ('útra', 1), ('útválasztása', 1), ('ûz', 2), ('über', 4), ('überlebt', 1), ('ügy', 2), ('ügyben', 2), ('ügyes', 1), ('ügyfeleim', 1), ('ügyvédként', 1), ('ügyében', 2), ('ügyészeknek', 1), ('ügyüket.ha', 1), ('ül', 1), ('ültettek', 1), ('ülésnapján', 1), ('ülésén', 1), ('ünnepiérzést', 1), ('ünnepséggel', 1), ('ürességet', 1), ('ütemben', 1), ('ütemû', 1), ('üzemeltetni', 1), ('üzemeltetése', 1), ('üzemköltségéhez', 1), ('—a', 1), ('—because', 1), ('—bill', 1), ('—chatelaine', 1), ('—macleans', 1), ('—newsweek', 1), ('—nicola', 1), ('—the', 1)]

The first token seems to be ‘az’ which does not convey any information about the document hence, this could be handles by increases the minimum word length or by removing just the word ‘az’. The characters like ‘—’ also seems to have occurred multiple times ,this could also be handled better.

Table 1.

|  |  |
| --- | --- |
| No\_of\_documents | Time (in sec) |
| 1 | 0.05 |
| 5 | 0.12 |
| 10 | 0.20 |
| 50 | 1.5 |
| 100 | 3 .20 |
| 300 | 7.5 |
| 503 | 10.02 |

**Graph of performance time vs number of documents processed** 

**Comparison of approach 1 and approach 2 summary**

Output from approach 2:

First 50 words sorted by frequency

(| - 96104) (000 – 62498) (the – 31401) (a – 26735) (of – 21019) (and – 17081) (is – 5508) (2 – 3157)……

|  |  |
| --- | --- |
| Approach 1 | Approach 2 |
| Maximum frequency word is 0.00 | Maximum frequency word is ‘|’ because punctuations are not removed |
| 1. (decimal number is treated as the way it is )   Word length is restricted to be above 1 to avoid words like ‘a’. | 0.00 is converted into 000 ,removing the decimal number .While doing this some adjacent characters could be added or removed treating them as different word ,leading to lesser frequency of 000  Word length is not restricted ,leading to words like ‘a’ being displayed and having a high frequency |
| Beautifulsoup is used for html to text parser. This helps in efficient extraction of different parts of the html file.  Takes around 5 seconds to process 503 files | Htmltotext library is used. This converts the whole text into one single big block of text. Difficult to separate a paragraph from title.  Takes around 8-9 seconds to process 503 files |
| Nltk WordTokenizer used for tokenization of words.  As nltk understands linguistic text better it considered America, as America. | Tokenization done by using string.split with space as a delimiter.  This approach considered America, as America,  Moreover (the’ , is considered different from ‘the’. This gets handled in nltk |
| The word and frequency are put in an array | The word and frequency are put in a hash map. This helps in handling them as key values |
| Frequency distribution is calculated using nltk.FreqDist() | Frequency of each word is calculated by counting the values for each keys. Incrementing the counter or sum of values gives the frequency |
| Put the array into a list and sort the list to get the word frequency in word sorted order.  To display the first 50 . Freuencydistribution.most\_common is used .  Last 50 by taking the last 50 values of the array | Sort the Keys in alphabetical order  Display first 50 and last 50 . |
| Total run time is around 10 seconds | Total runtime is around 15 seconds |

**Steps to run the program**

Step 1 : place the 503 html input files in a directory

Step 2 : On cmdline IR1\_HW1.py InputPath OutputPath

Step 3 : Check the output in OutputPath

Contains one text file each for every html file and 2 more file op.txt for word sorted (frequency) and op2.txt for sorted(word) frequency

**Improvements**

The words obtained are not of great quality

Handling of tokens like ‘s’ and ‘- -‘ needs to be done. The processing time could be improved too.

Words that do not make sense like ‘hogy’, ‘magyr’ should be removed along with custom stopwords.