CS 6322: Information Retrieval

Ashika Prakash Acharya
axa190084

HomeWork- 1
Program Description

PART 1: TOKENIZATION

- The program is written in a way that it is very easy and simple to understand. Python language is used for all coding.
- The program will read every available file in the defined directory path (`path = "cranfield/"`)
- For each file, the program will read it line by line and filters if it is a SGML tag that follows the style

  `<[/]?tag> | >[/]?tag (attr[=value])+>`

  A function `is_tag()` is defined which takes string as input and uses python regex functionality to search for tag expression.
- The program also filters special characters and creates tokens otherwise.
  A function remove_special_characters() is defined to remove special characters.
- The tokens are stored in python data structure list named **list_of_tokens.**
- Python Collections.Counter is used to convert list to a dictionary of the form

  <key, value> where

  key = unique tokens of the list,

  value = the number of times it occurs in the list.
- The dictionary (**dict_of_tokens**) that is created will help us with the text characteristics.

1. **How long the program took to acquire the text characteristics.**
   Time taken for tokenization 3.6970150470733643
   This is calculated by measuring time before reading files and once all characteristics are acquired. This is done with the help of python module 'time'.

2. **How the program handles:**
   A function **remove_special_characters**() is defined to remove below special characters
   [, / . - \ ( ) + *]
   a. **Upper- and lower-case words (e.g. "People", "people", "Apple", "apple")**
      All words are converted to lowercase while they are tokenized.
      Ex: Apple → apple, apple → apple

b. **Words with dashes (e.g. "1996-97", "middle-class", "30-year", "tean-ager")**
The dash (-) are stripped from words when they are tokenized.
Ex: middle-class → middleclass

c. **Possessives (e.g. "sheriff's", "university's")**
The apostrophe (') are stripped from words when they are tokenized.
Ex: university's → university

d. **Acronyms (e.g., "U.S.", "U.N.")**
The dot (.) in the acronyms are removed and treated as one word.
Ex: U.S → US

The program is now ready to answer the below questions.

1. The number of tokens in the Cranfield text collection
   TOKENIZATION Q1 : The number of tokens 231179
   Logic : Calculate the length of [list_of_tokens] using len() function.

2. The number of unique words in the Cranfield text collection
   TOKENIZATION Q2 : The number of unique words :: 12699
   Logic: Since dictionary maintains unique keys, calculate the length of unique keys in the dictionary.

3. The number of words that occur only once in the Cranfield text collection
   TOKENIZATION Q3: The number of words that occur only once :: 6345
   Logic: Iterate through each <key, value> in dict_of_tokens and increment the counter if value is equal to 1.

4. The 30 most frequent words in the Cranfield text collection – _list them and their respective frequency information
   TOKENIZATION Q4 : Top 30 most frequent words are :: {'the': 19442, 'of': 12672, 'and': 6659, 'a': 5919, 'in': 4627, 'to': 4529, 'is': 4111, 'for': 3490, 'are': 2427, 'with': 2263, 'on': 1940, 'at': 1834, 'by': 1747, 'flow': 1736, 'that': 1565, 'an': 1386, 'be': 1271, 'pressure': 1132, 'from': 1116, 'as': 1112, 'this': 1080, 'which': 974, 'number': 964, 'boundary': 897, 'results': 885, 'it': 852, 'mach': 816, 'theory': 775, 'layer': 728, 'was': 698, 'method': 683}

   Logic: Iterate through {dict_of_tokens} which is sorted on values in reverse order and store first 30 items of the dictionary.

5. The average number of word tokens per document.
   TOKENIZATION Q5 : The average number of tokens per document :: 165.12785714285715
   Logic: While reading each crandfield file, a {word_per_doc} dictionary is maintained which stores number of tokens retrieved per document. We will take sum of all values of {word_per_doc} and divide with number of items in dict to compute the average.

PART 2: STEMMING

- We have chosen Porter Stemmer implementation that is available with '**nltk'** module.
- The tokens that was created in part 1 are fed to the nltk function **stem**() and the results are stored in another list named **[stemmed_tokens].**
- We will use Collections.Counter functionality which will take **stemmed_tokens** as input and create a **{dict_of_stems}**

The program is now ready to answer the below questions

1. The number of distinct stems in the Cranfield text collection
   STEMMING Q1 : Number of distinct stems :: 9919
2. The number of stems that occur only once in the Cranfield text collection
   STEMMING Q2 : Number of stems that occur only once :: 5188
3. The 30 most frequent stems in the Cranfield text collection – _list them and their respective frequency information
   STEMMING Q3 : Top 30 most frequent stems are :: {'the': 19442, 'of': 12672, 'and': 6659, 'a': 5919, 'in': 4627, 'to': 4529, 'is': 4111, 'for': 3490, 'are': 2427, 'with': 2263, 'flow': 1965, 'on': 1940, 'at': 1834, 'by': 1747, 'that': 1565, 'an': 1386, 'be': 1368, 'number': 1337, 'pressur': 1307, 'from': 1116, 'as': 1112, 'result': 1086, 'thi': 1080, 'it': 1039, 'effect': 988, 'which': 974, 'boundari': 926, 'method': 883, 'theori': 868, 'layer': 859, 'solut': 847}
4. The average number of word-stems per document.
   STEMMING Q4 : The average stems per document :: 165.12785714285715