Information Retrieval · Assignment 1

Authors

- Daniel Martins, 80026
- João Ferreira, 80305

1 Introduction

This report explains the results regarding the first assignment of Information Retrieval. The assignment aim is to create a document indexer, in which the main components consist in a corpus reader, a document processor, a tokenizer and an indexer (Fig. 1).
These components will be explained in detail in this report, with the help of the architecture diagram and the data flow between them. It also contains details about execution instructions, external libraries used, efficiency measurements and final conclusions.

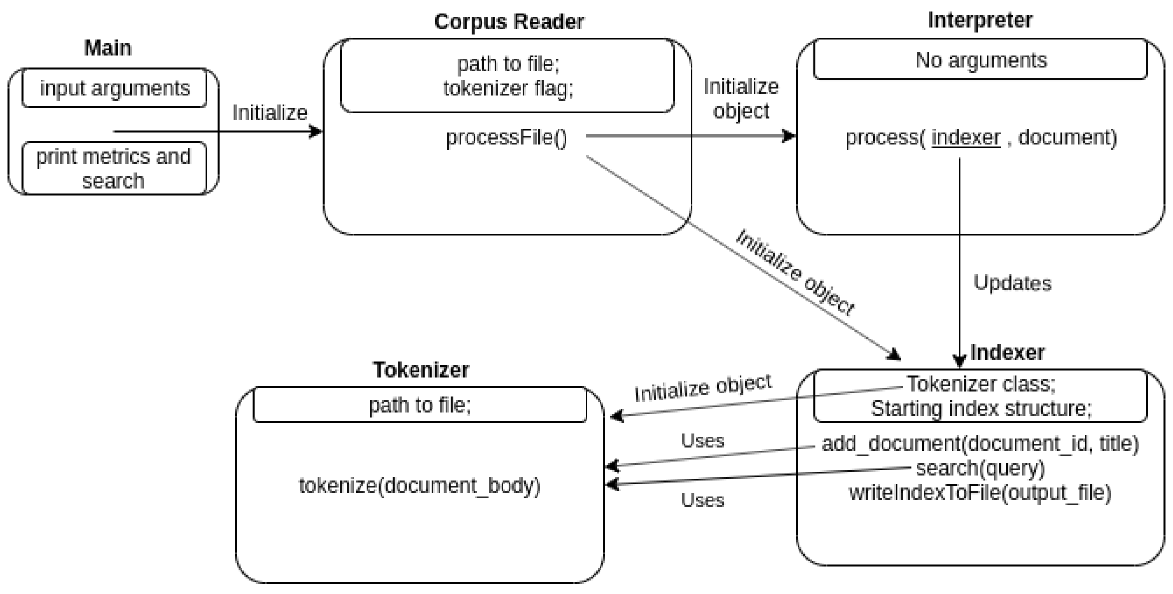

Figure 1: Class diagram.

2 Main

The main file is responsible for printing the measurements about the results and initialize the 'Corpus Reader', passing as arguments the user options, like

- Flag 'tokenizer' : what tokenizer will be used. If the flag is present in the execution options, the most sophisticated tokenizer will be used. If not, and for default, the most simple one, will be used.
- Path : what is the data file.
- Flag 'delete' : if he wants delete the older tokenization result file. If the flag is present in the execution options, the older file is deleted. If not, and for default, doesn't erase.
- Flag 'write' : if he wants to write in a file the indexing results. If present in the execution options, the results are written, if not, and by default, doesn't write.
- Flag 'search' : all that is written followed by this flag is the terms you are looking for.

2.1 Command Example

```
python3 main.py -f 2004_TREC_ASCII_MEDLINE_1 -d -t -w -s protein
```

-f indicate the file to process

-d delete previously generated index before processing the file, without this option, the script wont build the index again and will just load the previous generated binary structure.

-w write the indexing result in clear text

-t indicate which tokenizer will be used

-s anything that appears after this flag is considered to be the search terms.

3 Corpus Reader

The Corpus Reader class deals with the opening, reading and pre-processing of the documents. It receives in the construction of a new object the path to the file where the corpus is, the custom class tokenizer that will be used for the interpreter and indexer.

The method processFile(), initialize an indexer to index each token after the tokenization of each document, to pass as argument into the interpreter. As each line of a document is read, the Corpus Reader should be able to group them into documents for the interpreter to process.

4 Interpreter

The interpreter class is responsible for reading a document. It processes each document line by line and retrieves the relevant information to update the indexer. The interpreter function process(document) is called multiple times with the same index as argument.

5 Tokenizer

The tokenizer class is responsible to transform any sentence or phrase into valuable key words that can reference that phrase. There are several option to build a tokenizer in order to be memory efficient or precision efficient.

The first and the most simple Tokenizer, replaces all non-alphabetic characters by a space, lowercases tokens, splits on whitespace, and ignores all tokens with less than 3 characters.

The second does the same as the first, except ignore all tokens that are less than 3 characters long, but reduce inflected words to their word stem and ignore words that belong to a stopword list.

6 Indexer

The indexer class is responsible to build a structure that can efficiently store the key words of the documents in disk designed to search faster.

For the indexer structure we used a python dictionary, the relative performance is equally comparable to an hashtable for the task in hand.

Each document body is issued as an argument, then it is converted to tokens with the help of the tokenizer and each the retrieved token is updated in the index to refer that document.

The indexing result(binary structure) is saved in a bin file, in order to save time for testing the search functionality. Loading the indexing structure result file take approximately 8 seconds(SSD disk).

This class contains 3 functions:

1) add_document(document, body) - add document to the dictionary

2) search(query) - search for query statement in the index

3) writeIndexToFile(output_file) - write the indexer in a file in clear text, compressed compared to binary format.

7 Results

Regarding the document: 2004_TREC_ASCII_MEDLINE_1

7.1 The First Tokenizer

a) Index Time: ≈160 seconds

b) Index Size on disk: 270.9MB in python binary format, in 206,6 MB clear text

c) Vocabulary size: 246 533 words, size: 2.3MB

d) The ten first terms (in alphabetic order) that appear in only one document (document frequency = 1) :

   'aaahc';
   'aaah';
   'aaaga';
   'aaaction';
   'aaact';
   'aaab';
   'aaaat';
   'aaaasf';
   'aaaai';
   'aaaa';

e) The ten terms with highest document frequency

   'with': 311814;
   'from': 117323;
   'patients': 112027;
   'human': 106054;
   'cell': 90208;
   'cells': 85435;
   'study': 84058;
   'treatment': 78641;
   'protein': 72435;
   'disease': 70258;

7.2 The Second Tokenizer

a) Index Time: ≈250 seconds

b) Index Size on disk: 275.8MB in python binary format, 211.2 MB in clear text

c) Vocabulary size: 204 971 words, size: 1.7MB

d) The ten first terms (in alphabetic order) that appear in only one document (document frequency = 1):

   'aaahc';
   'aaah';
   'aaaga';
   'aaaction';
   'aaact';
   'aaab';
   'aaaat';
   'aaaasf';
   'aaaai';
   'aaaa';

e) The ten terms with highest document frequency:

  'cell': 175645;

  'patient': 142005;

  'effect': 138578;

  'human': 113224;

  'studi': 108482;

  'activ': 99867;

  'protein': 92121;

  'use': 89058;

  'diseas': 83289;

  'rat': 82656;

8 Discussion about the results

Although using stemming and stop-word removal reduce vocabulary size, the second tokenizer have a higher index size because in average each term will have more documents to refer too, documents strings will consume more space.