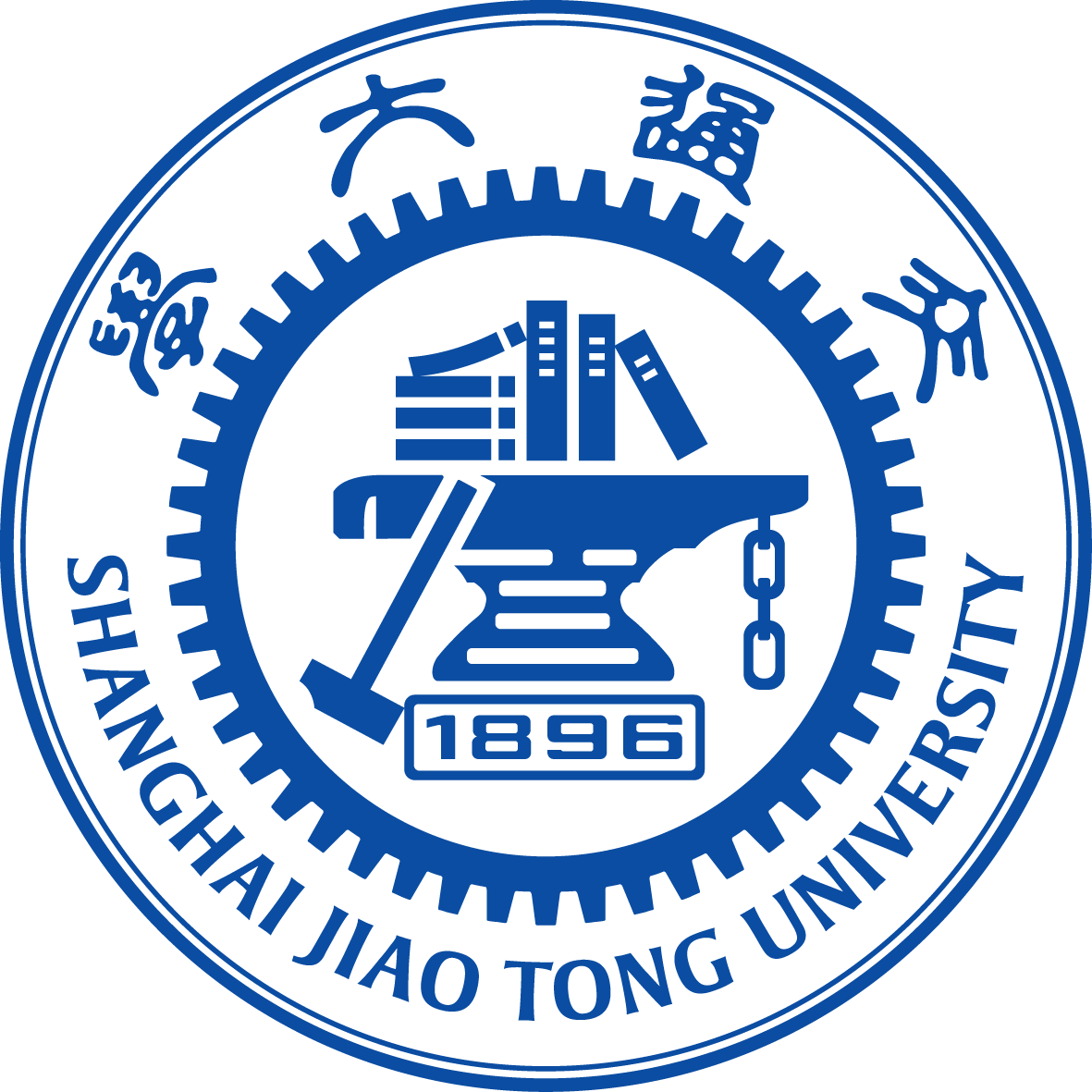
Web Search and Mining (F033583)



The Search System



Charlie Kruger J11603099010

Daniel Aldana 116030990081

Introduction

The search system consists basically of the following items listed below:

1. **The Web Crawler** which crawls the follow site (<https://www.realself.com/>) and obtain the frequently asked question (FAQ) and the doctor list and these are stored in a local directory.

2. **The Parser** will obtain the specific information of the crawled files and will make a first term list.

3. **The Intermediate** creates second list terms based on the parsed files.

4. **Query Parser** realizes spell corrections and utilize refinement techniques.

5. **Scoring table** constructs and build a full table of the scores and ranking the terms getting from the index.

6. **TF-IDF** the matrix for tf-idf scoring will sort all the documents by the respective relevance.

7. **The Intermediate** construct a term scoring by the numbers of reviews.

8. **Return Results** The results will be retrieved for the wait of the graphical interface.

9. **GUI** The Graphical User Interface will consist of a search button and a textbox for the input of the keyword.

10. **Returned Results** Number of item that can be returned and shown by page.

Detailed Description of the Search System Elements

**Web Crawler:**

The .html files are stored in the user’s directory under the folder tsg/raw once there the folders will be created per the crawled files downloaded, those categories are listed as following

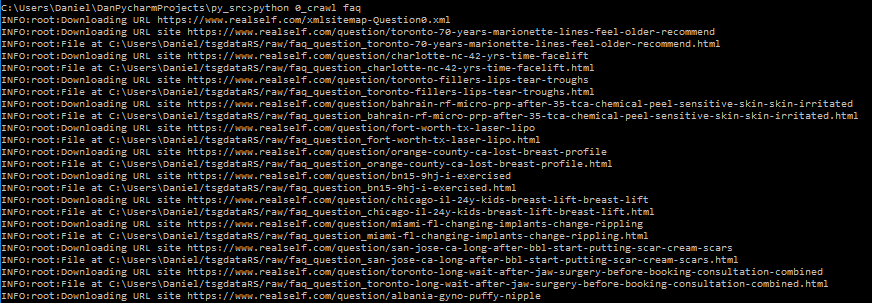
* Frequently Asked Question (FAQ)
* Doctors

**Item Names**

**Crawled Site:** SITE = '*https://www.realself.com*'

**FAQ items:**

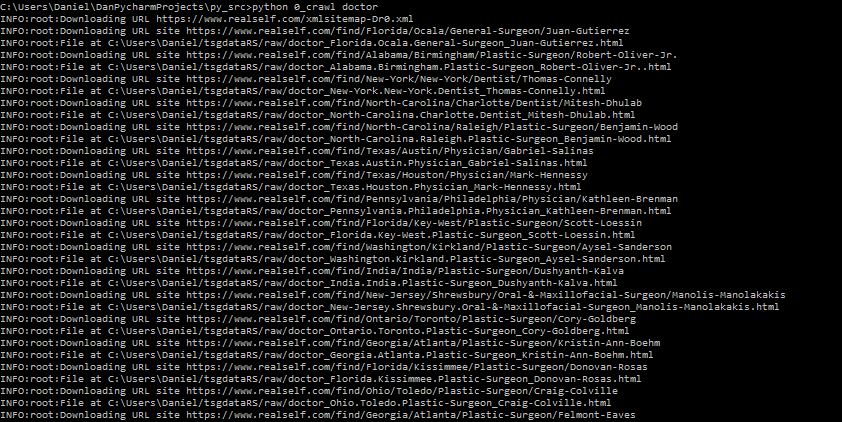
FAQ Site: */xmlsitemap-Question{}.xml*

FAQ Files*: faq\_question\_toronto-70-years-marionette-lines-feel-older-recommend.html*

**Doctor items:**

Doctor Site: */xmlsitemap-Dr{}.xml*

Doctor Files *doctor\_Anesthesiologist\_Alex-Roher.html*



**Parser:**

The parser read all the crawled html files and converts into a json file with a more structured information with parsed contents.

All the parsed contents consist of the following items:

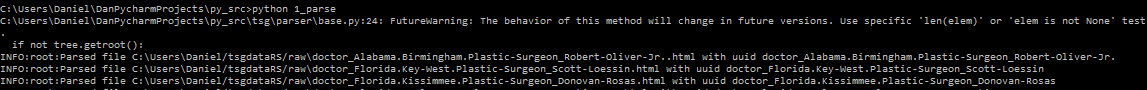
* Content: (parsed content)
* title: (doctor names or question title)
* heading: (answer titles(questions), review titles(doctor))
* text: (FAQ answer text, doctor review text)
* url:(the entire link of the html file on the realself.com site)
* uuid: (the identification of the file, how to find unique IDs by hash or content)

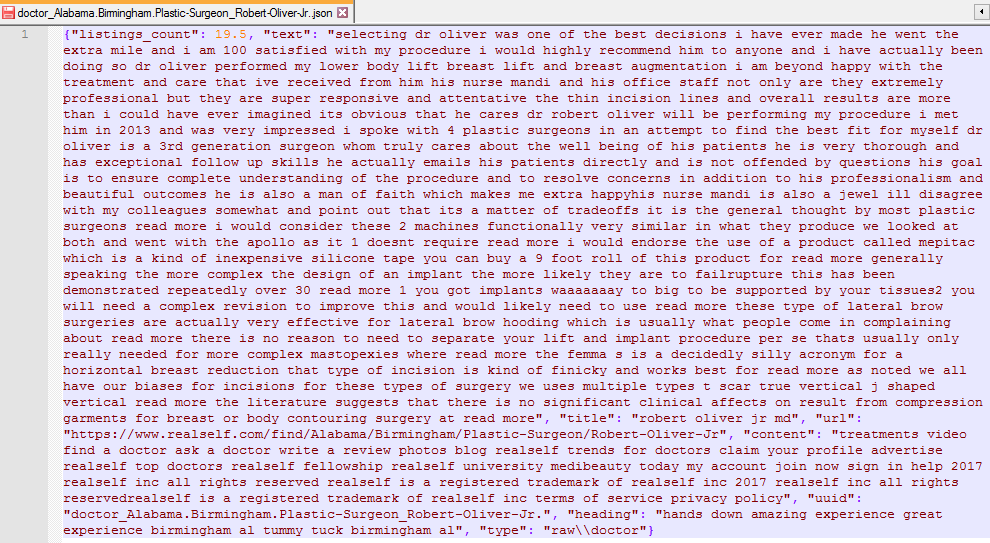
The content section is a parsed string list of the content separated by spaces and converted into lowercases.

**OUTPUT**

The folder parsed will be created inside the raw directory, the names of the files will be UUID.json

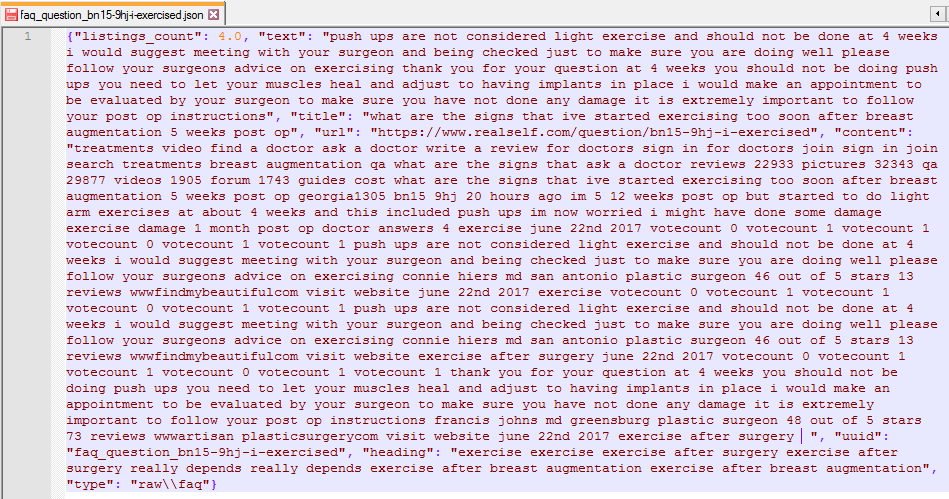
**Doctor Files**



**\*** Parts of the content section were removed, otherwise the follow image could not be displayed correctly. 

**FAQ Files**



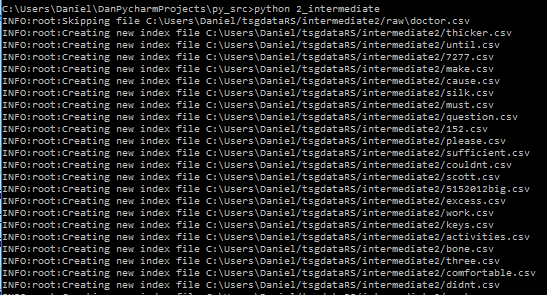
**\*** Parts of the content section were removed, otherwise the follow image could not be displayed correctly.

**The Intermediate:**

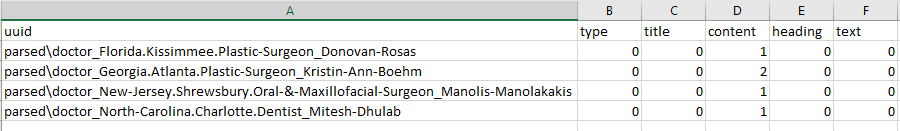
The intermediate reads all JSONs file and generates the full list of terms with their respective links to the documents and the number of occurrences.

**OUTPUT**

* The files will be stored under the directory /raw/intermediate
* For every term one .CSV file will be created
* The files will have the following items:
  1. name : term.json
  2. type
  3. title
  4. content
  5. text



Dentristry.csv



**The Indexer**

The indexer creates a term dictionary and index based on all the links of the documents, the term frequencies and the frequencies per documents, the dictionaries will be created in alphabetical order.

**Dictionaries**

The dictionaries will consist of the following structure:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| term1/*doc1*: tf-idf | term1/*doc2*:tf-idf | term1/*doc3*: tf-idf | term1/*doc4*: tf-idf | term1/*doc5*: tf-idf |

Ordering by tf-idf in continuously until the last document: term*(n)/doc(m):tf-idf*

**Term Hash**

Once the dictionary is created, a hash file should be created under the following structure:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| term 1 -  byte position | term 2 -  byte position | term 3 -  byte position | term 4 -  byte position | term 5 -  byte position |

This will produce that the search find all the position of the terms faster, which will improve the speed of the query and their retrieval.

**OUTPUT**

A file with these properties:

Name: dictionary.dat

Dictionary\_hash.dat (this will include the dictionary locations)

**PageRank:**

The input is the query with the clean string

The output dictionaries with the docs IDs (keys) and the scores values

**Method Score Calculation**

**Cosine Calculation**

* The query should be splitted by terms and for every term should perform:
  1. Calculate in the query the weight of the terms.
  2. Calculate the weight of the term in the document.
  3. Add the product of 2 and 3 to score **[doc\_id]**
  4. Add the square of product of 2 and 3 to length **[doc\_id]**
* Each score should be normalized using the following formula:

**Quality Score**

The quality score it is included already when the indexer is performed in the tf-idf phase.

**Query parser**

The Query parser takes the query input and pass to the ranker, check as well if the returned documents is lower than certainly number, in given case it will try the possible combinations of the query to fill the K docs.

In given case the query is:

” Breast reduction surgery” if < K docs, then try “breast reduction” and “reduction surgery” first, later fill K docs if don not work then fill “Breast reduction surgery” and Fill K docs.

Based on the given query, the algorithm will be executed and this will return in the indexes and get relevant information.

* tf-idf
* Ordering importance (according to the different indexes which has different weights)
  + Match title first
  + Match body later
* Semantic scoring based on query
  + Relation between the requested terms on the initial query and some specific term on the index.