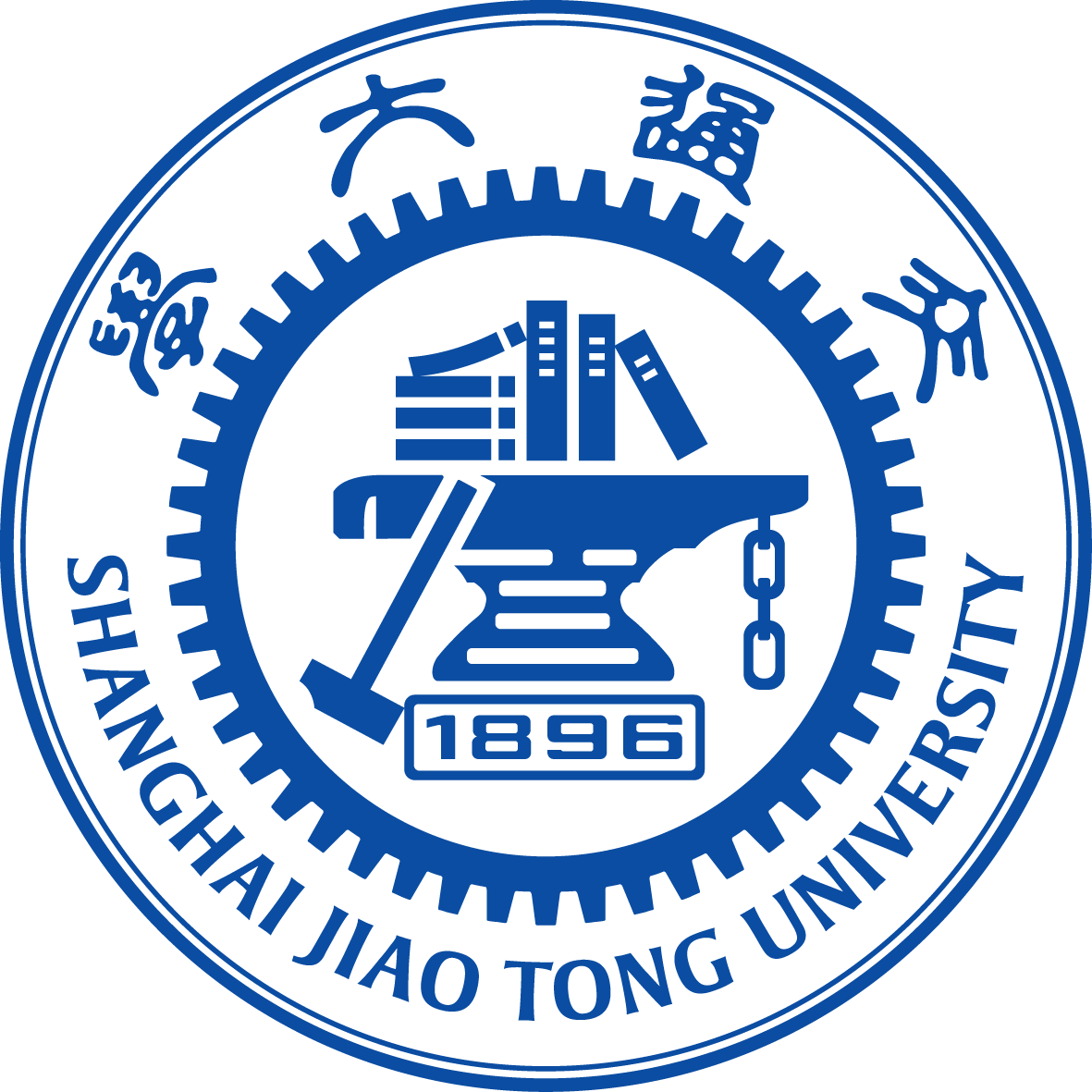
Web Search and Mining (F033583)



The Search System



Charlie Kruger J11603099010

Daniel Aldana 116030990081

Introduction

The crawled website it is [*https://www.realself.com/*](https://www.realself.com/), it will be crawled two specific sections: the doctor section which is under /xmlsitemap-Dr{}.xml and questions section, contained in /xmlsitemap-Question{}.xml both of these sections consists of the titles of the questions or doctors names, the heading which is the answer title of the question and the title of the reviews that a doctor is answering along with the content of the question and doctor information

The system consists of the following parts:

1. The Web Crawler which crawls the follow site and obtain the frequently asked questions and the doctor list.

2. The Parser obtains extracts specific information of the crawled files in bulk and gives a first structure.

3. The Intermediate Part creates from the extracted information preliminary files per word and connects it to the webpages where those words occur.

4.1 The PageRanker calculates the popularity of a webpage based on the number of links which point to it from other webpages.

4.2 The QualityScorer assigns a quality to a webpage. The quality is determined by the number of answers on a question website and by the amount of reviews in case of a doctor’s page.

4.3 Term Frequency-Inverse Dictionary Frequency( TF-IDF) is the matrix for tf-idf scoring will sort all the documents by the respective relevance.

5. The Indexer finally creates the final dictionary using the combined score of PageRanker, Quality Score and TF-IDF.

6. This can then be used via a Graphical User Interface (GUI ) that consists of a search button and a textbox for the input of the keywords.

7. The Query Parser converts the keywords to an appropriate format which can be matched against the dictionary.

8.The returned Results are presented on the graphical interface in a ranked fashioned and with links.

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

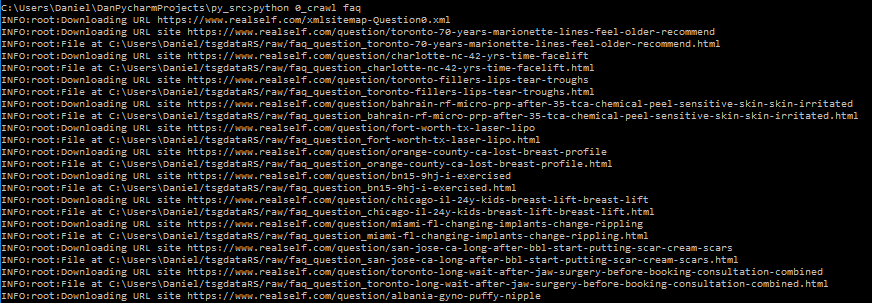
* Questions
* Doctors

**Item Names**

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

**Questions items:**

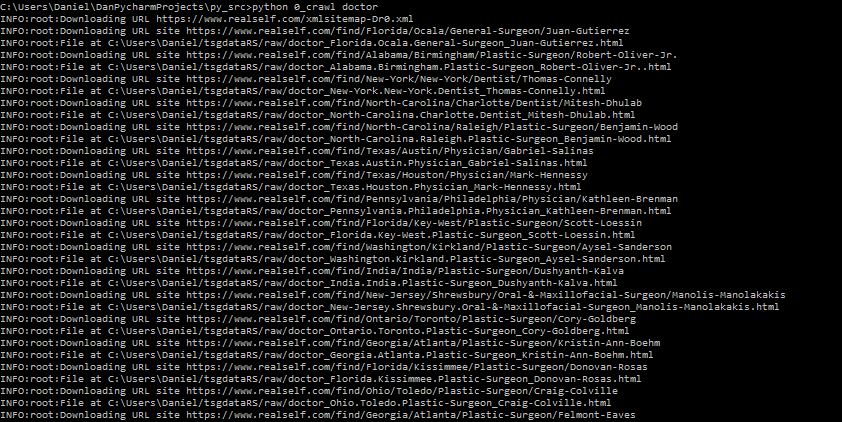
Questions Site: */xmlsitemap-Question{}.xml*

Questions 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.

The output files contain the following list of attributes:

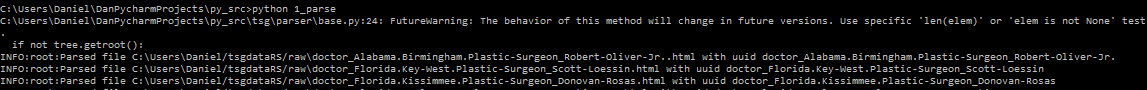
* type - the category of a website; either a question or a doctor
* title – the question for a question page or the name of a doctor
* content – the entire text content of a website
* uuid - the identification of the file, how to find unique IDs by hash or content)
* link (the entire link of the html file on the realself.com site)
* listing

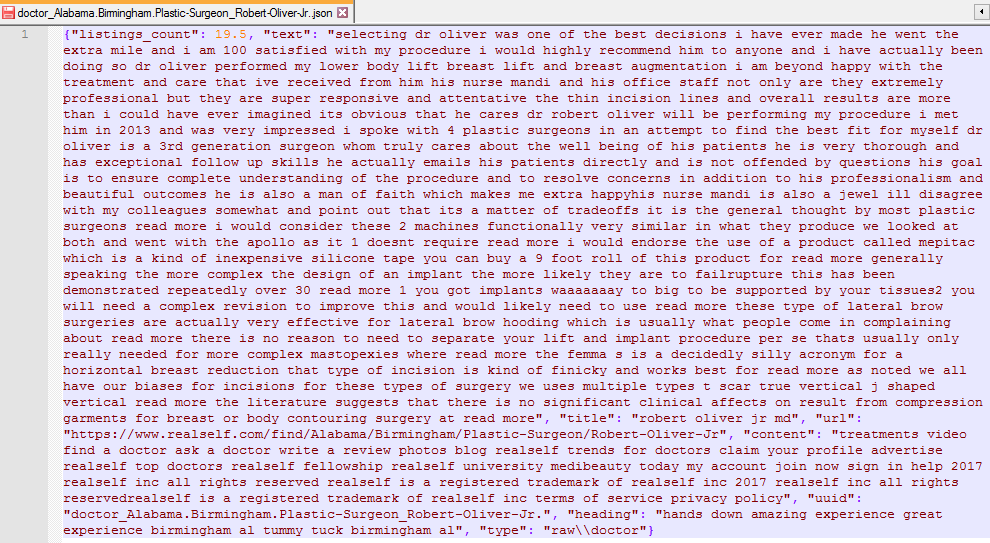
The content section is a parsed string list of the content separated by spaces and converted into lowercases.

**OUTPUT**

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

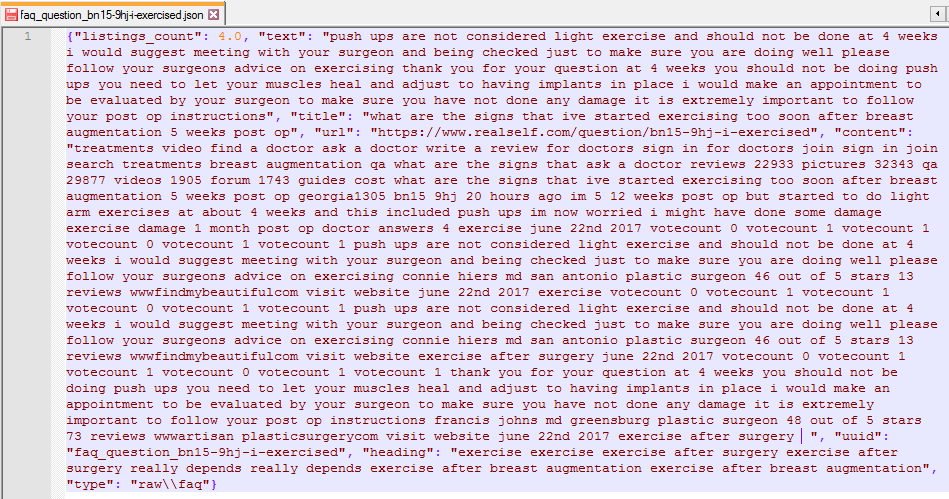
**Doctor Files**



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

**Question Files**



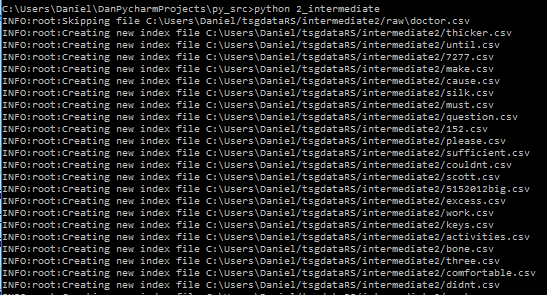
**\*** Parts of the content section were removed, otherwise the follow image could not be displayed correctly.

**The Intermediate:**

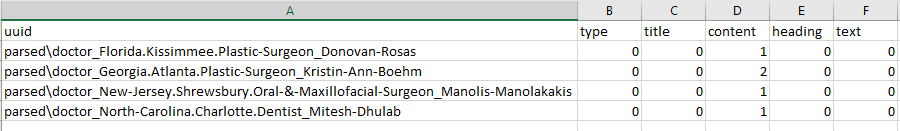
The intermediate reads all JSONs file and generates the full list of terms with their respective links to the documents and the number of occurrences.

**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 are structured as follows:

term{j} doc{i,1}:weight\_doc{i,1}, doc{i,2}:weight\_doc{i,2}, …  
term{j + 1} doc{i + 1, 1}:weight\_doc{i + 1, 1}, doc{i + 1, 2}:weight\_doc{i + 1, 2}, …  
term{j + 2} doc{i + 2, 1}:weight\_doc{i + 2, 1}, doc{i + 2, 2}:weight\_doc{i + 2, 2}, ...

**Page Rank**

The page rank will take in first place the links and will obtain the doc file string and the type of this document: doctor or question, once there the list will be unified and converts all the keys to uuid, the link dictionary will be created and sorted alphabetically.

The output is finally the 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:

**The Quality Score**

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

**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)

**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.

**Accepted Queries**

Apart from the simple and pure keyword search a range of filters can be used.

The keywords can be aimed at:

* either only doctors or only question pages (‘filter: {doctor, question}’)
* zones on the pages, like the title (question or name of a doctor), the heading (answer titles or review titles) or the text (answers or reviews) (‘filter: {title, heading, text}’).
* the specialization of doctors (‘specialization:< a specialization >’).
* the quality / rating of the targeted source. Doctors with a review rating of more or less than x or answers to question with a certain number of upvotes (‘filter: rating{<,=,>}<number>’).

After the query got parsed those arguments are forwarded to the ranking part.

There, either an appropriate separate dictionary gets used (title, heading, text) or the results of the standard dictionary get further filtered (doctor, question, specialization, rating).

**Future Work**

- Terms can be further processed to stem forms to match a wider range of relevantly similar words.

- Allow combinations of query filters.

**Technology used at the RealSelf search system.**

The follow components were using during the construction of the search system

**Python 3.6.1**

**Plugin****s:**

* Requests: (connect to realself.com and their branches)
* Lxml: (crawling and data scrapping)
* Pandas: (data structure and data analysis)
* Validators: (different validations thought the search sustem)
* Scipy & Numpy: (used for mathematics, numeric calculations / PageRank )
* Flask: (Graphical User Interface)

**Docker:**

Container platform for run the program on any machine, regardless the OS and other factors, eliminate the need to install multiple plugins across all the machines the program may run of.