Woogle: Mini Wiki Search Engine
Hands-on session for the Information Retrieval lecture

Clovis Galiez

November 22, 2022

We propose here to develop a mini search engine to mine documents from a wikimedia source. A skeleton of the program is provided, and we guide you in the completion of this program to achieve a search engine implementing the standard techniques of Information Retrieval.

1 Setting up your environment

Get the Git repository by running the following:

```
git clone https://gitlab.ensimag.fr/galiezc/wikisearchengineproject.git
```

You may need the follwing Python modules: `httplib2,numpy`. To install in user mode (when you do not have root access you ca use the following command:

```
pip3 install --user numpy httplib2
```

2 Crawling the data

This section describes how the data has been collected. **No coding is asked on your side.**

We developed the script `crawl.py` that requests the Wikipedia API for list of pages in a given category. By looking in the script, you will discover a straightforward querying to `http://en.wikipedia.org/w/api.php` asking for `"categorymembers"` of a given category. You can choose to retrieve the pages in the subcategories up to a depth controled by the parameter `crawlingDepth`.

After checking (quickly) the code, you should be able to answer the following questions:

Q2.1 Which Wikipedia category is crawled in this script?

Q2.2 What does this script output?

Q2.3 When running the script like `python3 crawl.py`, what should the file `wiki.lst` contain?

When setting `crawlingDepth = 2`, we get about 2 000 pages which is enough for the purpose of this hands-on session[1].

3 Downloading the data

We developed a simple bash script `dw.sh` that takes the file `wiki.lst` as argument and download the related Wikipedia pages. Due to some limitation on the Wikipedia side, we had to download the pages by batches.

By looking in the `dw.sh` script, can you tell:

Q3.1 How many pages per batch is downloaded?

Q3.2 What API of wikipedia is used to download a set of pages?

Q3.3 How does the crawling work here?

[1] When setting `crawlingDepth = 4`, we got about 70 000 pages. The documents have already been pre-processed and the results can be found in the `/matieres` directory, which you can use in case time allows for testing or optimizing your final solutions.

Q3.4 By going to the API page in your browser, and reading the documentation paragraph, can you tell in what format the pages will be encoded?

4 Parsing the data

From now on, the scripts are only partial, you will have to complete them up.

A parser of the Wikipedia documents has been developed in the `parsexml.py` that creates a *tokdoc* matrix as well as a *jump* matrix for the *random surfer model*.

Q4.1 From the code, how are encoded the two matrices (*i.e.* what type of Python object)? What is the name of this encoding?

Q4.2 Take a look at the database of Wikipedia documents in the `dws` folder, for example using the command `vi` or `less`. How are the links encoded in the wiki language?

Q4.3 Complete regular expression for extracting the links

Q4.4 Find and complete a simple regular expression for removing noisy data such as external links (outside of Wikipedia) and info boxes.

Q4.5 Implement your regular expression in Python such that the first group contains everything in the link (the target as well as its potential displayed text)[2].

Q4.6 The current implementation builds a doc-tok matrix. You need to transposes it to have a reverse sparse index. As this looks a bit underoptimal, try also to build directly the reverse version when parsing the documents (i.e. create directly a tok-doc index) and measure the performance (in per cent of execution time) you gain/loose? How do you explain that?

5 PageRank of the documents

The PageRank algorithm is implemented in the `pageRank.py` script.

Q5.1 In the *random surfer model*, at each iteration, random clicks are ”simulated” with a given probability. Complete the code with the correct probability.

Q5.2 What is the name of the effect we circumvent by adding `sourceVector` to the newly computed page rank vector `pageRanksNew`?

Q5.3 Implement the formula of the convergence δ.

Q5.4 Run the PageRank program in interactive mode[3] `python3 -i pageRank.py`, and use the Python interface to answer the following:

- How many iteration did it need to converge?
- What is the page rank of ”DNA”?
- What is the page with the highest rank?

[2] Here are some examples:

- `"([^a]+)"` will match any word having **no a**
- When parsing `Hello Bob!`, the pattern `"Hello ([^a]+)!"` will extract `Bob` in the first group
- Mind the necessity of *escaping* characters: `"(\[+)"` is the pattern to match any series of `[` character like `[[[[`

See `https://docs.python.org/3/howto/regex.html` for a description of the syntax.

[3] Interactive mode means that Python will runthe script and give you a shell afterwards to type in any code, allowin to use the functions and variables computed in the script. Syntax: `python3 -i yourScript.py`

6 Woogle!

Take a look at the file `search.py` and complete the code to produce a complete search engine.

If you run in interactive mode the `search.py`, get the best 15 results by the vector model for the query *evolution of bacteria*.

Q6.1 What type of page is selected by the vector model? By looking at the Wikipedia page, how can you explain it? What is the name of this classical *cheating*?

Q6.2 Propose and implement a way of correcting this phenomenon. Check if this correct the effect for the top 15 pages.

Q6.3 Take a look at vector model rankings for your query. What is the rank of the page *"Bacterial evolution"*? Rank the results by pageRank. What is the rank of the page *"Bacterial evolution"*? Is it expected? How would you correct for it (see extra section)?

Play around with standard queries and try to understand the behavior. You can try the following queries ("dna","darwin","crispr") while varying

- ranking and not ranking to check the difference
- Correcting or not for "classical cheating" (cf. just before)
- Varying the number of results to be ranked (say 2, 10, 20, 200)

Q6.1 What is your feeling about the right parameters? Is it better to rank with pageRank or not?

Q6.2 Can you devise a way of automating the search for the right parameters?

If you think that you have somehow limited data, try to get more data and see if it solves your problems.

7 Extras

You can pick here some ideas to improve and experiment around your search engine.

Effect of the source vector

You can tune the page source in the Random Surfer Model to put more emphasis on the source vector, as well as modifying the source vector itself.

1. Without changing anything, note down the page ranks of "DNA" and "RNA".
2. For example, set the source vector as the pages with RNA in the title and re-compute the page rank vector. What is the new page rank of "DNA" and "RNA"? How can you explain this?

Effect of stemming

In the current implementation the words *bacteria* and *bacterial* are different tokens.

1. List some simple grammar rules for stemming in English, and implement them in the right script. Does it improve the performance of the search? What if you use a standard library to do it?

Latent semantics

The current version ignore synonymy. Using the sparse implementation of SVD from the *scipy* module, implement a simple version of the latent semantics technique. Note that this method computes the correlation between tokens, and the quantity of documents is therefore critical to have an accurate estimation of co-occurring tokens. Implement and test your method with the current dataset, and use the bigger dataset in the end to achieve better results.

Embeddings

Instead of the natural embedding of the tf-idf, use standard libraries (word2vec, fasttext, doc2vec) to use as embeddings of document/queries. That's not as complicated as it looks.

New content scores

Detect special part of the documents (titles, section titles, bold text, etc.) to create a custom vector model that emphasis the important texts in pages. Think how to adapt the tf-idf model in that way, keeping the mathematical consistency of this model (which is basically a probability times an information).

Combination of content scores and ranking

The content score (*e.g.* scalar product between document and queries) and ranking are treated separately up to now. Try to create a combined score taking into account both of them. Do you improve your searches? How do you measure it?