Web Search using IR

Manuel Montoya Catalá [1]

[1] Master in Multimedia and Communications
mmontoya@ing.uc3m.es

Abstract

In this paper, a system for implementing Information Retrieval techniques over the internet is proposed. Since the Web contains too many files for performing classical IR techniques, Web Crawling is used for downloading specific domains, obtaining targeted HTML documents and preprocessing them in order to get their relevant plain text and metadata. The documents are then transformed into TF-IDF vectors for computing classical similarity measures on them such as Euclidean distance and cosine similarity. Also, a novel similarity technique is proposed for short queries.

Index Terms: Information Retrieval, Web Crawling, text processing, similarity measures, recipes, Python.

1. Introduction

The Web has become the largest available repository of data in the world; it contains over 47 billion webpages and billions of multimedia documents. Hence, it is natural to think of using it as a source for making datasets from which to extract information. Web search engines have become one of the most used tools in the Internet since they allow us to obtain the relevant "documents" from the internet, using a simple query.

These Web search engines cannot follow the classical concept of an Information Retrieval system since comparing a single query with these billions of documents is not tractable. Web search engines use techniques such as Indexing and PageRank which assign, to each possible word and word combinations; a list of ordered Web pages ordered by relevance according to some metrics.

Traditional information retrieval systems performs a search within a relatively small and non-linked collections of documents called Corpus (i.e. collection of biomedical papers), whereas in Web Information retrieval, the search is performed within the world's largest document collection.

The solution proposed in this paper for making possible the use of IR techniques in the Internet is targeted Web Crawling, where we download and preprocess all the webpages composing the corpus and then apply classical IR techniques to this constrained selection. The IR system uses a TF-IDF vector representation of every document and computes the similarity measures Euclidean distance and cosine similarity. A novel similarity measure has been created for short queries.

The system of this project has been developed in Python, making use the libraries "Scrapy" for the Web Crawling stage and "NLTK" for the text processing step. The rest of the code has been created from scratch. The corpus is composed by around 1000 food recipes obtained from two different domains.

2. System Structure

The structure of the proposed system is can be divided into two main components. First, we need to download and preprocess the corpus in order to obtain the relevant plain text and useful metadata. Second, we perform classical Information Retrieval techniques over the preprocessed corpus.

The figure below shows the basic structure of the system proposed in this paper.

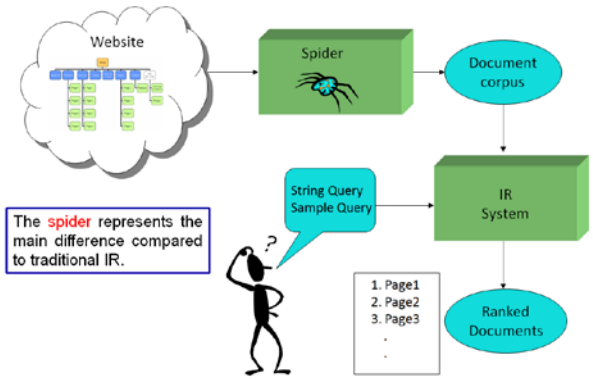

Figure 1 Structure of the system

2.1. Corpus obtaining and preprocessing

This stage is in charge of downloading and preprocessing the corpus, it can be subdivided in the following tasks:

- Web Crawling: For downloading the folder structure of several servers in order to download their recipes
- Document filtering: Removing near-duplicate documents and documents that do not contain a recipe.
- Webpage preprocessing: Getting the relevant text and metadata from the HTML documents.

2.2. IR system

The information retrieval system outputs the top ranked documents obtained given a query, it can be subdivided in the following tasks:

- Text processing: Token obtaining and normalization.
- Vector Representation: Every document is represented as a TF-IDF vector.
- Similarity computation: A similarity measure between the query and every document is computed. Corpus documents are then ranked according to this measure.

3. Web Crawling

The first step of our system is to obtain the RAW webpage documents from the internet and process them in order to get rid of the HTML markups, scripts, styles and misleading texts so that we are left with the relevant text and metadata.

For downloading the desired web pages a WebCrawler has been used, this is just some bot software that, given an initial URL, travels through the hyperlinks of the HTML document, downloading and processing the multimedia data inside a given domain. A WebCrawler has different properties to be set up like the starting URLs, allowed domains, link-following protocol, rules for scraping and the document processing function.

We tried to code our own WebCrawler from scratch but it had some issues in large-scale scraping so we decided to use the Python's scrapy framework. **Scrapy** is a scraping and web crawling framework, used to crawl websites and extract structured data from their pages.

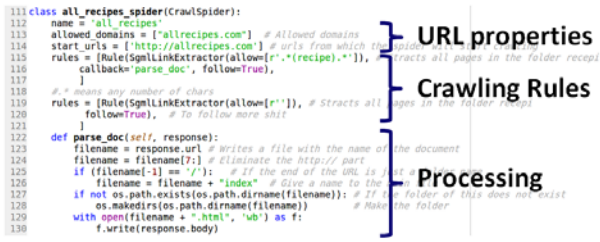

Figure 2 Web Crawler example

The figure above describes one of the Web Crawlers used in this paper, as it can be seen:

- The starting URL is allrecipes.com and so it is the allowed domain. No file from other domain will be downloaded and processed.
- The WebCrawler will only download and process the HMTL documents that contain the word "recipe" either on its name, or its path. It will follow all the hyperlinks it finds.
- The processing function writes the downloaded HTML documents into disk.

4. Webpage preprocessing

Once we have made a "carbon-copy" of the folder structure inside the server, we now have all the desired HTML documents into our disk. The final purpose of this Corpus preprocessing is to obtain a single HTML document per recipe and get the relevant plain text and useful metadata from them, discarding all other text in the webpages that could be misleading for the IR purposes.

First of all, we need to be able to "walk" through the Webpages directory and read every file, no matter what the structures of subfolders maybe. Then we remove all the documents that do not have a recipe in them and also those near duplicates documents that have the same recipe due to JavaScript generated pages.

For doing so, since every webserver has its own file structure, it was necessary to learn each specific one and remove the documents using rule-based methods making use of Unix commands and RexEx expressions.

Once we are left with a set of HTML documents, one per recipe, we cannot just apply text processing functions since the code has undesired elements of text such as:

- Scripts: Programming languages embedded in the HTML document such as JavaScript or PHP.
- Styles: CSS code to give a format to the text and other elements such as images.
- HTML markups.
- Misleading text: Like comments of the users or "Latest Recipes" hyperlinks.

BeautifulSoup is a Python library for HTML processing, it understands the HTML DOM structure allowing us to reference every element in the Webpage that has been given a name. Every server has its own specific way to reference the HMTL parts of a document so it is needed to get inside the code and find a common way of reference them.

Figure 3 Relevant text example

We can also exploit the HTML structure and obtain metadata information that can be useful for improving the ranking results or filtering the data. The following figure shows that we can obtain relevant keywords such as the title of the recipe or the category when we are able to reference those in the HTML markups and code. These keywords will have more importance than normal words for the ranking porpuse.

Moreover, we can obtain further information for filtering (and also ranking) such as the average rating of a recipe given by users (5 stars rating) or the cooking time. A way to reference these values can be found and perform a filtering strategy, i.e. filtering out all the recipes that take more than a certain time to be prepared or those that have a rating lesser than a given threshold.

Figure 4 Metadata example

All the operations described above have been performed in order to get the ultimate Web Searcher for recipes.

5. Text processing

Once the relevant plain text is obtained from the HTML structure, the NLTK 3.0 Python library is used to preprocess the data, performing a correction and normalization of the words and expressions which we will further be used for representing each document.

This preprocessing stage consists of the following steps:

- **Tokenization**: It divides the whole plain text String into separate tokens, in this case String words.
- **Lower Case**: Since upper letters are not important for the purposes of the task, every letter is turned into lower case so that the IR system is not case sensitive.
- **Removing Alpha-numeric words**: It removes all the punctuation and words that are not a combination of letters and numbers.
- **Removing stopwords**: It removes common words such as articles or junctions that are used in language to glue subjects. These words are not very relevant to the IR system because they don't offer information about the context themselves. The NLTK corpus of English stopwords is used in order to identify them.
- **Steaming**: Normalize verbs, plurals and compositions so that they are treated as the same. The Python implementation of the Snowball steamers is used.

The following figure shows an example of the outputs of this process for a String query:

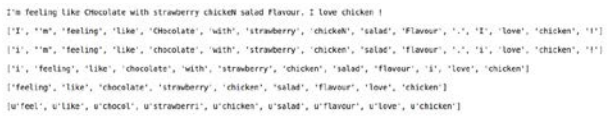

Figure 5 Text processing example

6. Document Representation

So far, every document is represented as a set of normalized relevant words. These words might also be repeated if they were written more than once in the HTML document. Each word's set is transformed into a TF-IDF vector that contains, for every different word in the set, a floating number obtained as the product of two terms:

- $TF_{(t,d)}$: Term Frecuency (BoW). Frecuency of term 't' in the document 'd'.
- $IDF_{(t,c)}$: Inverse Document Frecuency of term 't' in corpus 'c',

$$TF_{(t,d)} = \frac{Number\ of\ terms\ \boldsymbol{t}\ in\ document\ \boldsymbol{d}}{Total\ number\ of\ terms\ in\ \boldsymbol{d}} \quad (1)$$

$$IDF_{(t,c)} = \log_2\left(\frac{Total\ number\ of\ documents\ in\ \boldsymbol{c}}{Number\ of\ documents\ that\ contain\ \boldsymbol{t}}\right) \quad (2)$$

The resulting TF-IDF vector representing each document in the corpus has the form:

$$TF - IDF_{(t,d,c)} = TF_{(t,d)} \cdot IDF_{(t,c)} \quad (3)$$

The IDF term is a measure of how discriminative a word is in the whole corpus and the TF term is a measure of how representative a word is for a document.

For further understanding, let's see the TF-IDF vector resulting from the previous string query:

Figure 6 TF-IDF vector example

Since most of the terms in the query only appeared once, the TF term of all of them is the same, except for the "chicken" term which is twice as much. What it's left to see in the vector is the IDF component obtained from the Corpus, the bigger this value, the more discriminative it is since it appears in fewer documents.

A few ingredients can be seen in the vector, "chicken" has the highest value because his IDF value is multiplied by 2 instead of 1 as it happens to the other terms. The term "chocolate" has lower IDF value than "strawberry" which means there are more recipes with chocolate than with strawberries. Another set of words such as "feel" or "love" has also non-zero IDF values, which means they appear in the corpus, probably in the descriptions of the recipes. It's remarkable that the term "love" has such a low IDF value, meaning that it appears in a lot of recipes, even though you cannot buy in a grocery store.

7. Similarity measures

The TF-IDF representation of the documents, allows them to be represented as vectors in N-dimensional space, where every dimension corresponds to a possible term. Different similarity measures have been used according to the type of query provided, since for short queries, a useful frequency profile cannot be obtain and thus comparing TF-IDF vectors is nor suitable.

For example queries, the Euclidean distance and cosine similarity measure have been used. The total rank has been obtained by mixing both ranks.

$$Euclid\_distance(Q, D_d) = \left|\vec{d} - \vec{q}\right| \quad (4)$$

$$Cosine\_similarity(Q, D_d) = \frac{\vec{d}\cdot\vec{q}}{|\vec{d}||\vec{q}|} \quad (5)$$

For short queries, another two similarity measures have been implemented. The first one is just the number of words in common between the query and the document. The second method is similar but more complex; it computes the sum of the TF-IDF values of the words from the document that are common to the query:

$$Sim(Q, D_d) = \sum_{t\in(Q\cap D_d)} TF - IDF_{(t,d)} \quad (6)$$

This similarity is based on the fact that, since the TF-IDF values gives a measure of how important a word is in the corpus and document, the sum of the common terms can give an estimate of how relevant the document is, given the common terms with the query.

8. Experimental setup

The final corpus of this system is composed of around 1000 recipes obtained from the domains allrecipes.com and www.homecookingadventure.com. This number was chosen so that the Corpus did not have very close recipes which would make the ranking process too easy. The time required for performing all the preprocessing and similarity comparison is very low, so once the right similarity measures are set, the number of documents can be increase within 2 orders of magnitude without obtaining a big delay in the retrieving step.

The system accepts two kinds of queries:

- **String Queries**: These are relatively short hand-written queries given by the user.
- **Examples Queries**: An URL containing a recipe. The system will download and preprocess the webpage removing scripts, styles, hyperlinks and some other parts in order only obtain the relevant text.

As explained in the previous section, the similarity measure used depends on the type of the query. Once the corresponding similarity measure is applied to every document in the corpus, they are ranked accordingly and the index of the top relevant documents is retrieved.

The system opens an instance of the default web browser in the computer. The instance will have as many open tabs as top ranked recipes retrieved, and every tab will contain the online URL of the recipe.

9. Evaluation

In order to evaluate the performance of the proposed system, the Mean Average Precision (MAP) score has been used. Other common scores such as Recall or variants of Cumulative Gain could not be used since the Corpus is unlabeled. It is very costly to label it because of its dynamic content.

The MAP estimate was obtained from 5 examples queries and 5 short string queries, both retrieving the top 5 ranked documents. The MAP values are 0,79 for the example queries and the one for the short queries is 0,99. The reason for this is that the similarity measure of the examples queries does not suite this specific task since the number of words in the recipes is low compared to that in a regular document such as a paper or a book. Also, not all the non-relevant text can be filtered out of the example query, producing errors.

10. Conclusions and Future Work

The proposed system retrieves good results. It can perform better and more tailored searches than common web browsers; in addition, you can use example queries. The whole system demonstrates that you can make your own corpus about anything you want, in a single day, for free. The similarity measure proposed in this paper for short queries works fairly well, beating Euclidean distance and cosine similarity precision outstandingly.

Future works in the line of this project are:

- Improving the code that obtains the TD-IDF vector, so that it takes less time to compute. Other approaches describe every document with a large fixed-length TF-IDF vector, that has a position for every possible word in the corpus. This approach uses more memory to represent the document but decreases the time needed for computing similarities since the words in the vectors are already aligned.
- Obtain information not only from the HTML documents but also from the images, videos or pdf files embedded.
- Use CSS styles to weight the importance of every word (i.e. words that have bigger size could have more importance).
- This system can be used for any purpose, one interesting application at the time is finding suitable job offers in webservers such as Infojobs, Ticjobs and linkedin.

11. References

[1] Brants, Thorsten. "Natural Language Processing in Information Retrieval." *CLIN*. 2003.

[2] WU, Ho Chung, et al. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, 2008, vol. 26, no 3, p. 13.

[3] D. Jurafsky, C. Manning. “Natural Language Processing”. By Standford. Coursera: https://class.coursera.org/nlp/lecture

[4] Broder, Andrei. "A taxonomy of web search." *ACM Sigir forum*. Vol. 36. No. 2. ACM, 2002.

[5] Mansourian, Yazdan. "Similarities and differences between web search procedure and searching in the pre-web information retrieval systems." *Webology* 1.1 (2004).

[6] Langville, Amy N., and Carl D. Meyer. "Information retrieval and web search." *Supported by National Science Foundation under NSF grant CCR-0318575, USA* (2006).

[7] Information Retrieval on the Web. MEI KOBAYASHI and KOICHI TAKEDA. IBM Research

[8] Similarity measures for short queries. Ross Wilkinson, Justin Zobel. Melburne 3001, Australia. October 1995.

[9] http://ace.cs.ohiou.edu/~razvan/courses/ir6900/index.html