MERC/IST 2014

Extracção e Análise de Dados na Web

Lab 03: Indexing, Searching and Ranking

The goal of this exercise is to implement a simple, memory-based, version of the Vector Space Model for Information Retrieval, using the python language.

Your application should take as input a file containing several textual documents, one document per line. The documents should be read from disc and indexed into memory. Once the documents are stored, the application should read a set of keywords from *stdin* and use it as a query over the indexed documents. The documents should be presented ordered according to their similarity to the query.

1

Implement a function that reads a file from disc (as described above) and creates an in-memory *inverted index* of its contents. The index should contain, for each term, the documents where it occurs and its document frequency.

Assume that the terms in each document are separated by any sequence of non-alphanumeric characters. Each document read should be assigned a unique identifier.

2

Using the above function, implement a script to print some statistics on the documents indexed, namely:

- The total number of documents;
- The total number of terms;
- The total number of individual terms;

The script should also take a set of terms as command line arguments and, for each term, print the following statistics:

- *Document frequency* (DF);
- Maximum and minimum *term frequency*;
- *IDF*, according to the formula $\log(N/DF)$, where N is the total number of documents.

If needed, change your indexing function so that these values can be easily obtained.

Notes:

To read command line arguments, you should use the `sys.argv` list, from the `sys` module.

To find the minimum and maximum, try the `min` and `max` functions.

3

Implement a function that takes as input a list of terms and computes the *dot product* similarity between the query formed by those terms and each indexed document. The function should return a list of pairs (document id, similarity).

The following pseudo-code shows how this can be implemented efficiently.

- `Set` $A \leftarrow \{\}$
- `For each query term` $t \in Q$
  - `Set` $I_t \leftarrow$ `the inverted list of` t
  - `Set` $idf_t \leftarrow \log(N/DF_t)$
  - `For each` $(d, TF_{d,t})$ `pair in` I_t
    - `If` $A_d \notin A$ `then`
      `Set` $A_d \leftarrow 0$
      `Set` $A \leftarrow A \cup \{A_d\}$
    - `Set` $A_d \leftarrow A_d + TF_{d,t} \times idf_t$
- `Return` A`, where each` A_d `contains the similarity between the query and document` d`.`

4

With the above functions, create an application that indexes a set of documents, reads user queries from *stdin*, and produces a list of documents (or document ids) ordered by their similarity to the query.

Note: You can use function `raw_input` to read data from stdin.