TP 1-2 - A Simple Search Engine

The objective of the next two sessions is to make you practice and familiarize with the main features of the Python programming language : use of existing modules, function definition, input from files and standard input, data structures and iterations, printing results.

You shall work in pair (or alone) and send your code at `mailto:ferre@irisa.fr` *for evaluation. The evaluation criteria are the correction of the code, the fulfilling of the objectives, and the readability of the code (function and variable names, comments).*

1 Objectives

The goal of this TP is to implement a simple *search engine* in Python. Your program should take as input a CSV file describing a collection of documents (more details below), pre-process it, and then start an interaction loop where the user enters a query and the program displays the query results.

We assume that the document collection is represented as a simple CSV file with one column containing the URL of documents, and another column containing the contents of the document as a single string. We suggest to use the TED talk transcripts provided by Raounak Banik[1] on Kaggle as a document collection. The CSV file is available on Teams (as a ZIP file). It contains the transcripts of 2467 TED talks on all sorts of topics. Your search engine will allow you to find TED talks relevant to your topics of interest ! Of course, you are free to use another document collection.

Although it would be possible to answer queries by traversing the collection of documents in raw form, especially for relatively small collections, we require that your program computes an *index* of the collection, like real search engines do. The index should allow to efficiently retrieve for any keyword the list of documents (URLs) containing that keyword. it should also allow to retrieve the number of occurences of a keyword per document. For instance, the index should record the fact that keyword 'climate' occurs 3 times in document X, 5 times in document Y, etc.

The query semantics should be conjunctive, i.e. the results should be the documents that contain ALL keywords in the query. As an exception, if a keyword does not occur in the collection, it can be ignored but this should be made explicit in the displayed results (like striked-through keywords in Google). It is encouraged to normalize keywords and index entries somehow in order to avoid mismatch like "climate" and "Climate" (capitalization). It is also encouraged to compute a score for each result so as to rank them. A common score is tf-idf.

The interaction loop will be simply in text form in the interpreter (standard input and output). However, it should be made explicit what is expected from the user, and what is displayed.

1. `https://www.kaggle.com/rounakbanik/ted-talks`

2 Useful Resources and Tips

To facilitate the coding of your program, we suggest to use the following modules, available in the Python standard library[2] :

— `math` : for mathematical functions, useful for the computation of scores like tf-idf ;
— `re` : regular expressions, useful for custom string processing like splitting strings, etc. ;
— `itertools` : utilities for iterations ;
— `csv` : CSV files, useful for reading and writing CSV files ;
— `json` : JSON files, useful to save and load data structures, like an index or results.

3 Work Plan

In order to help decompose the global objectives into simpler tasks, we suggest the following work plan in order to have a first basic but complete program :

1. **Document collection.** Transform the input CSV file into a Python data structure that contains the set of documents, where each document has a URL and a text. Define functions that display excerpts of this data structure in order to test the transformation. Is the CSV file correctly read ? Are the URLs and texts correctly represented ? Is the number of documents consistent with the input file ?
2. **Index.** Compute an index from the document collection. An index is the collection of the words occuring in documents. Each word should map to the collection of documents/URLs that contain it, as well as to the number of occurences of that word in the document/URL. To make it more progresssive, you can ignore the number of occurrences in a first version. Python dictionaries are the natural match for representing the index. Again, define functions that display excerpts of the computed index, in order to test your code.
3. **Query answers.** Given a user query, input as a string, compute the set of documents that match the query, i.e. that contain all keywords in the query.
4. **Interaction loop.** Program an interaction loop that repeatedly prompts the user for a query, and displays the N first results. Find a way to allow the user to close the search engine in a clean way.

Many improvements are possible relative to the basic search engine suggested above. We propose a few significant improvements. Feel free to add more !

— **Words and keywords.** Words and keywords, as used in documents and queries, can take various forms for different reasons : capitalization (e.g., “Climate” vs “climate”), flexion (e.g., “dogs” vs “dog”, “running” vs “run”), abbreviation (e.g., “US” vs “United States”). This has the effect to increase the index vocabulary, and to miss results. There are also the *stop-words* like ’a’, ’the’ that are very frequent but useless. Find ways to improve the index by a better extraction of words from texts and queries. Introduce a function dedicated to that task. Do

2. https://docs.python.org/3.7/library/index.html

not try to make complex transformations, keep it simple ! Compare the size of the index before and after those improvements.

— **Ranking results.** Simply selecting documents that contain all keywords imply that all results are equal. Compute a score for each result, and sort results according to that score. We suggest to define the score of a document as the sum of the tf-idf scores of each query keyword in the document. See the Wikipedia page for formulas and explanations[3].

3. `https://en.wikipedia.org/wiki/Tf%E2%80%93idf`