CSIT 5930 Search Engine and Application

HW2: Clarification and Implementation Notes

Preprocessing steps:

The following steps are required. You should use NLTK libraries to perform these steps. Points will be deducted if you do not follow them.

- Punctuation removal

- Map uppercase letters to lowercase letter

- Stop word removal and stemming

Inverted file design

- Since phrase search is supported, you need to keep word positions in the postings lists of the inverted file. Word positions within the document should be counted before stop word removal is applied, i.e., if the document contains "coffee on the table", coffee has position 0, and table should have position 3. In this example, searching the phrase "coffee table" will not match the document.

- Use the simple tf*idf formula for term weighting. No tf_max or any form of normalization is required (i.e., no bm25, etc.). Cosine similarity is used.

- Constructing the inverted file (index) and using it for searching is required. You cannot search the documents directly to answer a query.
- It is important to keep your code neat and organized. You can find some tips on this https://refactoring.guru/refactoring

- It is impossible to fully specify all requirements of a search engine. In addition to those specified in this homework, you may need to make some design decisions. If you do, you need to document all assumptions and design decisions.

A ROUGHT skeleton of HW2:

Indexing

1) (Load dataset) Dataset is given as a JSON file, which is just a long string with all the articles and their fields concatenated; load the dataset into a Python object (see HW2 description). You can loop through the papers using standard Python iterators, e.g., for ... in ...

2) (Task a) For each paper (which is a string), perform preprocessing steps using NLTK

3) (Task b) Create postings: keyword -> [{ doc-id, [keyword positions] }, ...] and insert it into the index (inverted file)

Search

4) (Task c) Create a query string, apply the same preprocessing steps using NLTK to obtain the query keywords

5) For each query keyword, look it up from your index, accumulate the tf*idf score contributed by the keyword into the partial document score (i.e., compute inner product)

6) Compute the final cosine similarity score of the article (i.e., perform normalization with ||q|| and ||d||

7) For each of the top 5 results, retrieve the required statistics and display it in the result

Building the complete interactive system

8) (Task f) Create the search API and put it into the Flask folder

9) Test and have fun!

10) In HW2, you do not need to use JavaScript or CSS, and you only need to treat JSON as a string representation of an object.