Digital content retrieval, mod. B

Project work, part II

The main goal of this project is to build a search engine that is able to tell if a word is present in a document or not. Other features have been added to the search facility in order for it to be more complete, for instance the Porter’s algorithm, that is used to ‘stem’ the words (the words ‘work’, ‘working’, ‘worked’ are transformed into ‘work’, so they are treated as the same word). Furthermore, I added a simple user interface that suggests possible words to search in case of misspelling, based on an algorithm of word correction.

1. Data

I started from a database available on Kaggle.com, that includes 10 columns and 568454 data items. The dataset contains reviews on Amazon products, and for this project I used only two relevant columns: ‘Id’, that represent the identification code of the review, and ‘Text’, that is the actual review.

1. Methodology

The first operation to perform in order to obtain an efficient search engine is to build the so-called ‘postings list’. For every (unique) word that is present in the collection of documents, I have to build a list that contains the IDs of all the documents in which that word is present. Then the user can search for a particular word through the user interface, and the search engine will investigate the postings list to obtain the list of the documents in which that word is present.

1. Postings lists

3.1 Construction of the dictionary

The algorithm used to build the postings list starts from the building of the dictionary. The first operation I made was to create a list of all the unique words in the documents. To do so, I isolated every word in the documents and applied the Porter’s algorithm on them. This list is transformed into a set to delete duplicates, then transformed again into a list to allow the sorting. The result is a sorted list of all the unique words in the documents. This operation took few minutes to complete.

3.2 Postings lists algorithm

The next operation to be performed is the actual building of the postings lists. For every word in the dictionary, the algorithm scans all the documents to find out where the word is present. If it is present, the ID of the relative document is inserted in a list. The information is stored in a Python dictionary, with {key:value} represented by {word:[list\_of\_documents]}. This operation is performed for every word in the dictionary, so the resultant Python dictionary is the postings list. This information is permanently stored in a file, named ‘postings\_list.json’. This operation took several hours to complete, since the size of the dictionary is more than 88000 words and the number of documents in the dataset is 568454.

1. Search facility

The search engine includes a simple user interface that takes in input the word to search. I added a system for the correction of the word in case of misspelling. To perform this task, I used an algorithm whose workflow is the following:

* Load the complete English vocabulary (sorted) as a list of items;
* For every unique word in the documents, compute the frequency as:
* Check if the word to search is included in the vocabulary, using binary search.
* If the word is not included, build a list with all the possible variations of the word, with maximum 2 typos.
* Compute the score. For every variation, the probability is computed as: . The weight, in this case, has been set to 0.01.
* Sort the new words based on the score.
* Provide as output the first 5 words in the sorted list.

The user interface gives the possibility to change the word to search without re-running the whole program, which stops when:

* The word is found in the vocabulary and in the postings list 🡺 output : [list of document IDs]
* The word is found in the vocabulary but not found in the postings list 🡺 output : “The word is not found in any of the available documents”
* The word is not found in the vocabulary and the user specifies that he doesn’t want to change the word to search. 🡺 no output

The searching operation in the postings lists is performed using binary search for efficiency.