CS F469 - Information Retrieval

Assignment 1 -

Passage or Paragraph Retrieval From Given Documents

Vishesh Mehta - 2020A7PS2194H

Shaury Trivedi - 2020A7PS1690H

Kathan Naik - 2020A7PS2082H

Kedarnath Senegavaram - 2020A7PS0245H

Introduction

In the given assignment, we were provided with a set of pdf documents related to automobile insurance and another set related to property insurance, and the objective was to take a random word or phrase from the user and retrieve the entire passage or phrase that contains the word from the dataset given to us.

Implementation

Preprocessing of Dataset and Vocabulary Formation

The first step was to preprocess all our documents. The documents provided to us were in pdf format, and these documents were converted into .txt format using online tools. All the converted .txt files were saved in the same folder, with "property" mentioned before the names of documents relevant to property insurance and "auto" before the names of documents relevant to automobile insurance, in case we needed to differentiate. Files were imported and Porter Stemming was used to stem the words to root form. NLTK was used to tokenize the stemmed words, and to remove any stop words such as "and", "am", "the" and the punctuations such as ".", "......", "," etc. which are redundant and appear in every other sentence. From the list of remaining words after cleaning, a set was created to remove duplicates, and this set forms the vocabulary of the given dataset.

Paragraph Storage on the Backend

Since the requirement is to retrieve entire paragraphs/passages that contain the user query, we needed a way to store paragraphs. But the definition of a paragraph is pretty loose and hence a little difficult to implement accurately. We have considered 15 lines of text to be a "paragraph" in our implementation. We created individual lists for each document, and each entry of a list is the text of one paragraph. With 8 documents relevant to property insurance and 7 to automobile insurance, 15 total lists were made, and these 15 lists were put into another list. This therefore became a list of lists where all our paragraphs are stored.

Term-Incidence Matrix

Using the vocabulary we had created earlier, and the paragraph list, we have created a term incidence matrix with the following dimensions: the number of rows is equal to the number of vocabulary entries, and the number of columns will be 15. Each entry of the term-incidence matrix is a dictionary, where the key-value pairs are the paragraph number and the number of occurrences in the paragraph respectively.

dict{x:y}

- ***x :*** paragraph number
- ***y :*** number of occurrences in paragraph x

The number of entries in an individual dictionary are obviously not fixed and depend on how many times a word in the vocabulary appears in a document, and how many times in which paragraph of the document. The final matrix is a therefore a data structure of the form:

list[list[dict{ string:int }]]

with dimensions ***n ✖ 15***, where ***n*** is the number of words in the vocabulary and ***15*** represents the total number of documents (8 related to property and 7 related to automobile policy)

Retrieval of Information

After taking the query from the user, the query sentence will be tokenized, stemmed and stored in a list. We will then take each word stored in this list and check from the **Term-Incidence Matrix** mentioned above**,** the presence of this word in each of the paragraphs of each document. We are then creating a new common paragraph list which contains the numbers of all the paragraphs in which the words of the input phrase are present. Using this common paragraph list, and the list of paragraphs mentioned earlier, we retrieve the paragraphs relevant to the input query and store it in two different .txt files, depending on the type of entry

Additional Implementations and Improvements

1) We have implemented an attractive GUI so that the user can enter their search queries with ease. We have provided a defined space for the user to enter their query in. Along with this we have created a log for the user, which is a file which contains all the queries searched by the user, using which we display the last three queries searched by them. So, in case the user wants to obtain any previous search results, they can check the last three searches with a single glance and if they want, they can trace their exact search history as well.
2) We have ensured the functionality of retrieving documents containing the exact phrase as the user search query. In order to optimize the search for documents with an exact match to the user input, we are first searching for paragraphs containing words in the input, which are loosely relevant. Among these paragraphs we are searching for the exact user input. If we were to search for the exact query first instead, it would loop through each of the paragraphs, and considering our estimate of 15 lines per paragraph, the code would loop for a large number of iterations. We tried this execution as well, with an upper cap of 10000 iterations, and we found that this took slightly longer to execute (around 10 seconds). After the optimization we are able to cut down the execution time to 1 to 2 seconds. This was a major optimization we were able to make in terms of execution time.

Precision and Recall Scores

Precision = (Number of relevant documents retrieved) ÷ (Total documents retrieved)

Recall = (Number of relevant documents retrieved) ÷ (Total number of relevant documents in database)

Precision is 1 in our implementation because all documents that were retrieved were relevant to the input phrase

Recall is also 1 in our case as all the documents retrieved are relevant documents.

The unusually perfect score for precision and recall may probably be because of the small size of the database (only 15 documents) and also because we have implemented term-incidence matrix and boolean retrieval, which usually result in high scores in small document corpus.

Bibliography

1) https://www.nltk.org/
2) https://docs.python.org/3/library/tk.html
3) Porter Stemming Algorithm (tartarus.org)
4) https://stackoverflow.com/questions/18171739/unicodedecodeerror-when-reading-csv-file-in-pandas-with-python