Exercise 5: Text retrieval

Multimedia systems

2018/2019

Create a folder `exercise5` that you will use during this exercise. Unpack the content of `exercise5.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as the *Matlab/Octave* scripts to `exercise5` folder. In order to complete the exercise you have to present these files to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the presentation as well. The tasks that are marked with ★ are optional. Without completing them you can get at most 75 points for the exercise (the total number of points is 100 and results in grade 10). Each optional task has the amout of additional points written next to it, sometimes there are more optional exercises and you do not have to complete all of them.

Introduction

In this exercise you will implement some indexing operations used to retrieve information from a corpus of text documents. As the size of readily available text documents grows beyond all measures, methods for fast and user-friendly querying of information from text files are needed. Due to the fact that *Matlab/Octave* is relatively poorly suited for working with text, you will write this exerise in Python.

Assignment 1: Setup of `nltk` and corpuses

In this assignment, you will install the `nltk` library, then load and preprocess a corpus of documents.

(a) First, install the Natural Language Toolkit [1], which is a Python library for processing language data. It includes various corpuses of text and functions for its processing such as tokenization. The easiest way for the installation is probably using the `pip` package manager but you can install `nltk` in any way you wish.

(b) The library includes mechanisms for loading a number of different text corpuses. Load the Gutenberg corpus by using the function `nltk.download()`. Then, with the help of the NLTK book[2], familiarize yourself with the contents and structure of the corpus. If at any point of the exercise you find the Gutenberg corpus too small or otherwise unsuitable for your needs you are welcome to try other corpuses available on the internet.

[1] `https://www.nltk.org/`

[2] `http://www.nltk.org/book/`

(c) Preprocess the raw data of each document in the Gutenberg corpus using the `nltk` inbuilt tokenizer. You can use the function `word_tokenize`. Also remove stop words to further reduce the amount of data you will need to process later on.

(d) ★ 15 Write your own tokenizer for preprocessing. You can use regular expressions. Show the difference in the results from the tokenizer you used in the previous task. What kind of tokens did you keep/ignore relative to the in-built tokenizer?

Assignment 2: Indexing

In order to simplify and speed up operations on sets of documents, they need to be indexed. This allows a fast look up if and where a query word or phrase appears in our corpus.

(a) Build an inverted index. For each unique token appearing in your corpus, make a list of the documents it appears in. Make some queries using Boolean logic (operation AND is an intersection of lists etc.).

(b) To allow querying of tokens that occur together (i.e. common phrases), a positional index can be used. Build a positional index on your corpus. This is an extension of the inverted index where each of the list elements containing the document index also stores a list of positions in the document where the token appears. Make sure you properly removed stop words in order to keep your computation relatively fast.

(c) Use the positional index to query phrases. That is, return the positions in documents where each of the words in your phrase occurs at approximately the same position in the document.

Assignment 3: Text statistics

(a) The relevance of the documents in your corpus to the user's query can be measured by the term frequency, i.e. how many times the query term appears in each document. Implement a function that counts the number of appearances of each token in each document.

(b) The absolute number of appearances is biased, therefore a different metric called TFIDF (short for term frequency–inverse document frequency) is commonly used to rank the relevance of documents containing a query. TFIDF is computed as follows:

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t, \tag{1}$$

where $tf_{t,d}$ is the frequency of the term t in document d and idf_t equals to

$$\log_{10} \frac{N}{df_t}, \tag{2}$$

where N is the number of documents in the corpus and df_t is the number of documents of the corpus in which the term t appears.

(c) Implement a system that returns the first 5 most relevant documents from the corpus given a query. Note that your queries can contain more than one word. The score in that case is calculated as

$$s(q, d) = \sum_{t \in q} tfidf_{t,d}, \tag{3}$$

where q is your query.

(d) ★ 10 Implement a system for handling typographical errors of queries on the user's part. The choice of the method is up to you. You need to show that your system returns relevant results for misspelled queries.