Web Information Extraction and Retrieval - Assignment 3rd Report

Žan Ožbot, Roman Komac, Dik Medvešček Murovec

May 2019

1 Introduction

Since we already implemented a web crawler and extracted specific content from content blocks, our task is to build a simple index and implement querying against it. We started by extracting textual context from 1416 web pages which came from four different domains then performed pre-processing and stored data into an index. Finally, we were able to query against it.

2 Implementation

The implementation is divided into two independent sections, data processing and indexing 2.1 and data retrieval 2.2. Each section describes steps needed to build the end solution. We also added two optional parameters to run the implementation

- `--help`
  Shows options listed below.
- `--snippet-size` or `-s`
  Maximal size of a snippet. Defaults to 200. If first snippet longer than snippet-size it is still displayed.
- `--num-rows` or `-r`
  Number of retrieved rows. Defaults to 10.

2.1 Data processing and indexing

Firstly, we read and clean the documents with lxml. The cleaner removes all the JavaScript and style tags and the content within them. Secondly, we convert the HTML to text and remove all other HTML tags and HTML special characters. Then, we use the library nltk to tokenize the text, remove all of the stop words given in the instructions and also remove the punctuations. Finally, the tokens are inserted into the database.

2.2 Data retrieval

We implemented two kinds of data retrieval; sequential 2.2.1 and using the inverted index 2.2.2. Both approaches use the same pre-processing the only difference is that first does not store the tokens anywhere and the second one saves them inside an inverted index.

2.2.1 Sequential

This approach always reads all the documents when the algorithm runs storing the tokens in a dictionary, which dramatically slows down the query functionality. See section 3 for more details.

2.2.2 Inverted index

First, we open a connection to a database where the inverted index is stored. As soon as the query in pre-processed we find the words in the inverted index which are pointing to a document, which is stored in another database for convenience purposes.

3 Results

3.1 Database

The database consists of 47547 indexed words. In Table 1 we can see the top 5 most used words through all of the given documents.

word	**apperance**
uporabe	1399
pogoji	1398
domov	1384
portalu	1367
slovenije	1363

Table 1: Top 5 words appearing through different documents.

3.2 Comparison

We ran our implementation on the following six queries seen in Table 2 by using the sequential and sped-up approach using an inverted index. The retrieval results are the same for both approaches but the time to complete between the two is substantial. As expected the time in sequential data retrieval is quite constant as it always depends on the number of documents that need to be read and pre-processed. The speed of the inverted-index approach however depends on the number of queried elements and the number of documents in which certain token appears. The full output with snippets can be found inside *results* folder.

query	**sequential time**	**inverted index time**
“predelovalne dejavnosti”	62.42s	832ms
“trgovina”	64.55s	124ms
“social services”	65.41s	0ms
“avtomatizacija orodja”	64.14s	31mss
“novica”	64.20s	0ms
“human organs”	62.60s	0ms
“revija avtomagazin”	60.94s	328ms

Table 2: Comparing sequential with the inverted index approach.