Lab 1 Boolean and Ranked Retrieval

Information Retrieval, 2020

Introduction

Goal To build a simple Boolean Retrieval model for a small test collection of documents

Instructions

We will use movie subtitles as our test collection. Copy the example data from:

/local/kurs/ir/lab1/movies.txt

This file contains a collection of movie subtitles with 10 sentences per line. We will simply treat each lines as a document in this exercise. Note that the corpus is already white space tokenized, but still contains quite a bit of noise which you don't have to worry about.

Inverted index

The first task is to convert the data into an inverted index. You can extract the terms together with document IDs corresponding to the line number using the following command:

```
perl -ne '$a++;s/\s+/ $a\n/g;print $_;' < movies.txt > term-list.txt
```

As described in the lectures this list should be sorted and multiple occurrences of the same token in a document should be ignored. This can be done using the sort command:

```
sort -t ' ' -k1,1 -k2,2n -u < term-list.txt > index.txt
```

You can put these commands together in a simple pipe to avoid intermediate files like the term-list.txt file.

Alternative A: Implement (in your choice of programming language) a program that reads through the list of sorted terms and creates an in-memory inverted index. Use appropriate data structures for the dictionary and the postings. Keep it simple and don't forget to add comments to your code to explain what you are doing.

Your program should return the size of the dictionary (number of terms in the vocabulary) and the sum of the length of the postings lists.

Lowercase the data before indexing using tr "[:upper:]" "[:lower:]" in your pipeline command when preparing the list of terms. Compare the size of the dictionary and the postings before and after lowercasing

Get the 10 most frequent words from the lowercased test collection and treat them as stop words. How much does this reduce the size of the index (in terms of postings)?

Alternative B: Look at the index file and describe how you could use it to create an inverted index from the data and what kind of data structures would be appropriate.

Count the size of the vocabulary (using appropriate command line tools such as cut and uniq).

Lowercase the data before indexing using tr "[:upper:]" "[:lower:]" in your pipeline command and count the size of the vocabulary after this conversion. How much is the vocabulary reduced in this way?

Get the 10 most prominent terms (terms with the largest document frequency) in the lower-cased index (again, using command line tools such as cut, uniq and sort). How much would you reduce the size of your postings in the inverted index when removing these terms from the index (treating them as stop words)?

Boolean queries

Now we will test some simple boolean queries with our index data.

Alternative A: Extend your program from the first part to allow simple conjunctive boolean queries. Start by retrieving the result for a single word query like 'school' and show the result of that query. Check with the original file (movies.txt) if the document IDs you obtain correspond to correct matches in the original text. You can use sed for retrieving specific lines from a text using this command (for retrieving line 13):

```
sed -n 13p movies.txt
```

Implement a function that performs the intersection between two posting lists according to the procedure described in the book and the lectures. Try your procedure with the query: **school AND kids**. Check again if the result is correct.

Try now to support even conjunctive queries with more than 2 query terms. Try your implementation with the query **really AND kids AND school**. Check your results.

Adjust your query processing routines to return the number of comparisons needed to perform the intersections. Add a simple query optimization feature and measure the number of comparison steps after adding this feature

Alternative B: The task is to process simple boolean queries using the index file create above. We will simulate the dictionary lookup by using grep:

```
grep '^school ' index-low.txt
```

This gives you the postings (document IDs) for the query 'school'. Count the number of matched documents and check for some of them if the term really appears on that line. You can do that using sed, for example retrieving line 13:

```
sed -n 13p movies.txt
```

The next task is to test conjunctive queries consisting of more than 2 query terms. Try the following query: **school AND kids**. First retrieve the posting lists for both terms as described above. Then, manually produce the intersection of the posting lists and check the results (whether the lines you obtain really include both terms).

Try also to compute the result of the query **really AND kids AND school**. Calculate the number of comparisons you need to perform when intersecting posting lists in the given order of query terms and compare this with an optimized version of processing this query.

Ranked queries

Now we will test ranked queries using standard TF/IDF weighting

Compute the inverted document frequencies for the query terms school, kids and really. You can use the grep commands again and count the number of postings (= number of lines) to obtain the document frequencies for each term.

Create a new file with term frequencies for each document and each term. You can do that by modifying the sort command when creating the original index-file: Change

```
sort -t ' ' -k1,1 -k2,2n -u
```

to

```
sort -t ' ' -k1,1 -k2,2n | uniq -c
```

This should give you frequency counts in the first column before the actual index term. Do this for a lower-cased version of the corpus!

Compute the TF/IDF values for the query terms school, kids and really for the documents that contain all three terms. Use the result from the boolean query to see what kind of documents match that conjunctive query.

Lab report

Report and discuss your results for all assignments above. For Alternative A: also hand-in your code with appropriate comments and with short instructions how to run your implementation.

VG part

Familiarize with two python libraries:

https://www.crummy.com/software/BeautifulSoup/bs4/doc/
http://docs.python-requests.org/en/master/user/quickstart/

An example of usage of the libraries, for The New York Times website:

```
import requests
from bs4 import BeautifulSoup

r = requests.get('https://www.nytimes.com')
soup = BeautifulSoup(r.text, 'html5lib')
```

Now we can look at the elements that BS has parsed for us:

```
print("PAGE TITLE", soup.title.string)
```

```
PAGE TITLE The New York Times - Breaking News, World News & Multimedia
```

or

```
for h2 in soup.find_all('h2', class_="story-heading"):
    print("HEADLINE", h2.get_text().strip())
```

```
HEADLINE Democrat Just Misses Victory in Georgia House Race
HEADLINE Analysis From Times Reporters
HEADLINE O'Reilly's Future at Fox Dims as Support of Murdochs Erodes
HEADLINE Carrier Wasn't Sailing to Deter North Korea
HEADLINE Trump Signs Order That Could Curb Foreign Workers
HEADLINE How Trump's Order May Affect Tech Worker Visas
HEADLINE Trump Raised $107 Million for Inauguration, Doubling Record
HEADLINE New York Politicians Offer Tax Returns and Pokes at Trump
HEADLINE British Leader Calls for Election, Seeking a Boost for 'Brexit'
HEADLINE What Is a Snap Election and What Would Follow?
HEADLINE Your Wednesday Briefing
HEADLINE Listen to 'The Daily'
HEADLINE Sometimes, Spending Brings a Bigger Return Than Saving
HEADLINE Birth Control Causes Depression? Not So Fast
HEADLINE Ivanka Trump Brand Poses Ethical Issues for New Post
```

Now create a program to crawl a corpus of advertisement messages from blocket.se or other alternative websites. Learn the structure of the website and tailor both libraries to your project needs. Process and store only clean data in text format. The corpus should be a single file (any format: raw text with your own structure, CSV, XML, JSON, ...). What is important that each advertisement should contain information like: ID, title, time, location, name, advertisement text. Submit a short report describing your approach to collect advertisements, your corpus and commented code.