Exercise 1 - Basic Search System

188.977 Grundlagen des Information Retrieval 2015w

October 21, 2015

Abstract

At the end of this exercise, you should have an understanding of the practical issues of creating an inverted index for information retrieval, as well as basic scoring. The exercise can be done in pairs, but each of the members has to be able to answer questions about the implementation provided. Make sure you go through the lecture slides and relevant book chapters at least once before starting on this exercise.

Task

Your task, should you choose to accept it, is to create a basic ranking engine and be able to write and talk about it. Your performance will be graded as follows:

Functionality (70%)

Index

Build two inverted indexes from the document collection 20 Newsgroups Subset as provided on TUWEL (*20_newsgroups_subset.zip*). One is a simple bag-of-words, the other is a bi-word index. *If there are three team members, also implement an n-gram index.* You may use any format for storing the index.

Vocabulary

Normalize the vocabulary by applying any combination of the techniques described in Chapter 2 of the *Introduction to Information Retrieval* book (case folding, removing stopwords, stemming, lemmatizing). These options should be exposed as parameters in the index creation phase.

Search

Implement a basic search functionality and provide it as a command line interface (CLI). No GUI is required. The CLI allows the user to provide search parameters and a file with a topic to search for. The system then returns a list of documents ranked by a similarity function of your choice and based on a variant of term frequency ranking. Different components (e.g. the use of the bag-of-words or the bi-word index, scoring method) need to be exposed as parameters in the command line. Two scoring methods have to be implemented, at least one of which must have document length normalization.

The search engine must take as a parameter a topic file. You will be provided with a set of 20 topics to search for. These topics are in the same format as the collection itself, so it should be easy to parse it using the same parser you created for the indexing part.

As a recommendation, also regarding the future exercises, try to implement topic processing and the actual scoring/ranking as two separate tasks. By topic processing we understand here simply getting the terms that will be search in the index. Consider the use or not use of the header elements in the topic document.

Your search should be (moderately) efficient. It is e.g. not ok to perform your collection indexing only at search time, and also loading the inverted index should be reasonably in time, or at least not happen before every search.

For each topic of the 20 provided, the result should be a ranked list of up to 100 documents, where each line is in the following format:

topic Q0 document-id rank score run-name

where

topic is "topic#" where # is a number between 1 and 20;

document-id is an identifier for the document, consisting of the folder name in the newsgroups archive and the file name (e.g. misc.forsale/76050)

rank is an integer indicating the rank of the object in the sorted list (normally, this should be ascending from the first line to the last line)

score the similarity score you calculated (normally, this should be descending from the first line to the last line)

run-name a name you give your experiment (should be the same for the same configuration of index-scoring method, across all 20 topics tested)

Here is a short example of a potential output for a topic:

```
topic1 Q0 misc.forsale\76442 1 2.813525 group5-experiment1
topic1 Q0 misc.forsale\76056 2 1.0114759 group5-experiment1
topic1 Q0 rec.autos\102958 3 0.58848727 group5-experiment1
```

Java-based: You must use Java for your solution. You are allowed to use standard Java API functionality for index and search. For the vocabulary part, you are allowed to use external algorithms (e.g. for stemming). Everything you use, you must be able to explain.

Evaluation

You must use `trec_eval`[1] to calculate the Mean Average Precision of your result lists over the 20 topics provided. `trec_eval` is a small C application, which means that you have to compile it for your own machine. In principle, you can use any C compiler (most Linux distributions already come with one), and just run the `make` command in the folder[2].

[1] `http://trec.nist.gov/trec_eval/trec_eval_latest.tar.gz`

[2] Note that last year some students could not compile the latest trec_eval on Windows, but managed to compile version 8.1 (also available from the same website). If that happens to you, you may use 8.1 as well.

```
trec_eval -q -m map -c qrels.txt your_results
```

where the parameters are[3]:

-m measure: Add 'measure' to the lists of measures to calculate and print. If 'measure' contains a '.', then the name of the measure is everything preceeding the period, and everything to the right of the period is assumed to be a list of parameters for the measure, separated by ','. There can be multiple occurrences of the -m flag. 'measure' can also be a nickname for a set of measures. Current nicknames include

'official' : the main measures often used by TREC

'all_trec' : all measures calculated with the standard TREC results and rel_info format files.

'set' : subset of all_trec that calculates unranked values.

'prefs' : Measures not in all_trec that calculate preference measures.

-c: Average over the complete set of queries in the relevance judgements instead of the queries in the intersection of relevance judgements and results. Missing queries will contribute a value of 0 to all evaluation measures (which may or may not be reasonable for a particular evaluation measure, but is reasonable for standard TREC measures.) Default is off.

-q: In addition to summary evaluation, give evaluation for each query/topic

and where

`qrels.txt` is the file available on the resources on TUWEL. It contains a concatenation of the files `topic[1-19].qrels.txt` from the `qrels.zip` file. It ignores variants of relevance judgements like `topic1b.qrels.txt`.

`your_results` is your results file using the format indicated above, where all topics are present.

In the end, your results should be in a table like:

	bag-of-words	biword	n-gram	bag-of-words v2
topic1	0.02	0.03	0.025	0.0251
...	...	...	...	...
topic19	0.01	0.023	0.035	0.0241
average	0.012	0.021	0.022	0.0151

Note that topics 8, 12 and 20 have no query relevance judgements (qrels), therefore your results will only be evaluated on the other topics.

In the table above, *bag-of-words v2* is another execution on your bag-of-words, using different parameters (see above index creation). The *n-gram* column only needs to be there if there are three team members.

[3]taken from the help menu of trec_eval. Type `trec_eval -h` to see the full list of options

Point allocation

The 70 percentual points are allocated as follows:

Document processing	
basic tokenizing	12
header filtering	2
case folding	2
special strings	2
stemming	2
Index creation	
Simple posting list, Hash or B-Tree dictionary	12
skip lists	3
Single Pass In Memory Indexing	5
Search	
scoring method 1	7
scoring method 2	7
tolerant match	3
Evaluation	
simple	12
calculate statistical significance using R (code provided on request)	2
Total	**71**

Report (15%)

Describing the prototype in a report. Describe how the index is generated and stored, how the vocabulary is processed and how the search works. The report must explain how to run the prototype. Maximum size: 4 pages or 2000 words.

Hand-in Presentation (15%)

- Prototypes are presented to the coordinator in one of the labs.
- Final deadline is December 4th at 8 am (but you may hand-in earlier). For that you must upload a zipped file to TUWEL (report, source code, and corresponding executable jar file incl. all dependencies).
- Your submission has to be self-contained.
- You must book a time on TUWEL for the presentation.
- You present on your own notebook to the coordinator in the seminar room.
- Seminar rooms for hand-in times are on TISS

Note: You must have 33% of this exercise in order to pass the course.