IR Exercise: Simple Retrieval Models

Simone Teufel

February 21, 2012

Part I: Building an Inverted File

In `data.gz` you will find several hundred text documents (newspaper articles). They constitute your corpus for this 3-part exercise.

The first part of this practical is to build an inverted file from the documents – the operation called "indexing" in an IR system. You will use some of the Unix tools specialised for character processing (some of them going beyond the introductory "Unix" material on the web). In the entirety of this exercise, you should not use `grep` as part of your code, but you may want to use it for testing. (Which reason might I have to discourage you from using `grep`?)

(a) For each document, create a file where all strings occurring between blanks occur on a line of their own. These strings are your approximation to terms.

(b) Collect terms from all files in one output file. Sort and count the terms in this file (tip: you need a *unique* sort to do this properly). This file constitutes a very simple word list (lexicon) representing your corpus (with various errors still contained in it).

(c) Now change your processing in a) and b), so that each term has an indication of which file it comes from. Again, sort and count the term–file combinations. The output of this process constitutes a very simple inverted file.

(d) Improve the term identification by removing punctuation marks immediately following words. This constitutes a very simple tokeniser. Recompile your inverted index (by sorting and counting).

(e) Improve the processing further by downcasing (converting capital letters to lowercase ones) all terms. Recompile the inverted index.

(f) Optional: you can explore ways of making the inverted file "cleaner" and "better". Another improvement: what is needed if you want to make the inverted file ready for

searches involving "near" by recording the term's position within the file, or for output which highlights the query term?

Part II: Boolean Model and Stemming

(a) Write a program in a language of your choice (perl?) which reads in your inverted file and stores it appropriately, e.g. in a hash.

(b) Add a line to your program which reads input (the queries).

(c) Given a single term as input, your program should now print out the files that this term occurs in. This is a very simple Boolean search engine.

(d) Add the functionality to query two terms, with either an "AND" or an "OR" between the terms. The search engine should report documents containing *both* the terms connected with "AND", and documents containing *either* term for "OR".

(e) Download the Porter Stemmer and include it in your processing pipeline from Part I, in order to create a stemmed inverted file. Remember that you also need to stem the query in order for this new, stemmed IR engine to return sensible results.

Part III: Vector Space Model

(a) Change your IR engine from Part II so that document–query similarity is calculated by the Vector Space Model instead of by Boolean set operations. For this, you first need to create a vector representation, corresponding to a Term–Document matrix (i.e., recording in which document which term occurs). Incoming queries should consist of two (implicitly) AND-ed terms. Your engine should print the name of the closest matching document in vector space.

(b) Optional?: Write a program which calculates TF*IDF values for each term in a document, according to the formula in the lecture.

(c) Change your solution to (a) in such a way that the weights from (b) can be used, i.e. calculate cosine over weighted vectors.
(d) Test your implementation systematically, e.g., with a little test suite of queries, so that others can see what your code is (supposed to be) doing.