CSCI 6403 - IR Assignment (Boolean Search Engine)

Hatem Nassrat
Dalhousie University - Faculty of Computer Science

October 19, 2008

Confirmation of Independent Work

I, the undersigned, certify that the work I have submitted for Assignment 2 (Boolean Search) is the product of my own work. Any additional software packages that I have used are listed as follows and have been previously approved by Dr. Shepherd:

External Resources

1. **Programming Languages**: Python
2. **Tools**: Berkeley DB 4.4, Porter Stemmer (w/python wrapper), Pylons Web Framework
3. **Other**: Public Stop Word list

Hatem A. Nassrat _____ _____
B00393388 date

1 Resource Description

Python
An object oriented, interpreted programming language.

Berkeley DB
A database engine with B-Tree, Hash, Queue, and Recno access methods. Here we use the BDB Hash access method.

Porter Stemmer
An algorithm, after the inception of the Porter2 or English stemmer is mainly used in IR research to compare with previous results. We have also used the port to this stemmer into python downloaded from http://snowball.tartarus.org/wrappers/PyStemmer-1.0.1.tar.gz.

Pylons
A growing web framework for python that allows for creating web applications design using Model-View-Controller (MVC) architectures.

Stop Words
Used a public set of stop words, to both create the inverted index and parse user queries. Stop word list retrieved from http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words

2 Data Pre-Processing

For this assignment, the data was preprocessed similar to the previous assignment. Therefore the following assumptions/processes were made to the input dataset. Similarly this pre-processing has been run on all the queries.

1. A word is denoted as a string of letters (digits are considered word boundaries)
2. HTML entities (e.g. &*quot*;) and tags (e.g. $< p >$) are removed
3. All punctuation/underscores are collapsed/imploded (e.g. my-term_rank's → mytermranks)

3 Word Frequencies

The following subsections describe some word frequencies after processing of the dataset.

3.1 Stop Words

From the public stop word list mentioned earlier, $1,546,513$ instances after preprocessing were found and removed. This denotes around 40% of the initial keyword set was filtered. In terms of unique words, from an initial set of $75,087$ words, $74,780$ remained after stop word removal.

3.2 Keyword Counts

From an initial group of $3,865,496$ keywords, $74,780$ unique words were found after preprocessing and stop word removal. After stemming they were reduced $56,566$ unique stemmed keywords.

4 Word Stemming

When retrieving information, it is very unlikely that users would be searching for a given words in particular form. The following use case displays this problem. A user may place a query for "speed car", a basic search engine would only consider documents containing terms "speed" and "car", however documents that discuss "speeding cars" would be ignored.

Such a problem may be solved by use of a Stemming Algorithm. Using such algorithms the root form of the query words would be passed to the Query Engine. Since all documents would be indexed using the stemmed forms of their words, the problem would disappear. For example passing the above words to the Porter Stemmer Algorithm would map the query and the displayed terms to "speed car", thus finding the document in the example.

We have chosen the Porter Stemmer [4], since it is widely used in information retrieval research and allows comparison of our search engine with previous work. There exists a newer version of the Porter Stemmer Algorithm (Porter 2), which would replace this stemmer, if we are to release this search engine.

5 Dataset Parsing

The dataset was parsed based on the $< newsitem >$ tags. These denoted the start and end of each document which was recorded in the forward document index. From each document, only $< title >$ and $< p >$ were looked at for words to index since these contain most of the article's *beef*. These words were pre-processed and stemmed as mentioned above, then placed in an inverted index as described in the next section.

On the Search Engine Interface, the full document is displayed in its original XML form with all the relevant fields, and attributes. For information on how the queries were parsed see Section 7.1.

6 Index Generation

All indices were placed in a Berkeley DB hash structure. The following subsections describe why BDB was chosen over GDBM and the details of each of the file structures (Forward and Inverted Indices).

6.1 Back-end Choice

A brief experiment was run to check whether Berkeley DB's *dbhash* module is more suitable than the Gnu DBM. Since this project is python based, a python experiment was set-up. It was clearly seen that *BDB* had a clear advantage in write speed over *GDBM*. However, they were extremely close in random access read times. After looking at the generated file sizes BDB also had smaller file sizes for the generated test databases. This experiment was verified by the results found at [1]. According to that study, *QDBM* or its successor *Tokyo Cabinet* are the fastest DBM clones, However, there are currently no functional python bindings available.

Moreover, it seems there are bug reports filed against GDBM for corrupt files if they are not properly closed, or kept open for too long [3]. Although many of these deal with writing issues, this was another reason for avoiding GDBM, as our search engine will keep the DB open while it is running.

6.2 Forward Index

The forward index, is the index were we store information about each document in our dataset. In this index we have stored the following information:

- **Doc ID**: A unique ID for the document
- **Offset**: Offset of the document within the dataset file
- **Length**: Length in bytes of each document
- **Title**: The document's title for display

The way they are stored within the database, is basically a mapping of *Doc ID*, to a serialized hash mapping of the other fields. This mapping is serialized into JSON [2] such that the DB will be programming language independent, and can be easily read by any language that has a JSON parser.

Since we are implementing a boolean search engine, the forward index, need not contain the indexed terms for each document as they will not be useful.

6.3 Inverted Index

The inverted file, is one which stores a mapping of *terms* to *documents* which contain these terms. In a boolean search engine, the number of occurrences of each term need not be associated with each document, but since the infrastructure was available it was included. Therefore our inverted index contained the following informations associated with each term:

- **Doc ID**: Unique ID for documents containing *term*
- **Word Count**: The number of occurrences of *term* within document

Each term pointed to a hash map, that had its key being the document and the value being the word count. This mapping was serialized into JSON [2], and placed into the DB. Thus giving us a DB which contained terms mapped to the documents that contained the term.

As mentioned, all the terms were pre-processed and stemmed prior to indexing. On Torch (torch.cs.dal.ca), building the full inverted file took on average around 25 mins prior to stemming and 15 mins when applying stemming.

7 Query Engine

The Query Engine, was built to be able return results for user queries. It was built in a modular fashion, such that each task was distributed to a specialized function.

7.1 Query Parsing

This search engine was to deal with simple boolean queries only, and therefore need not handle bracketed queries. A parsing function was written such that it handled simple boolean queries.

The parser is handed a list of words, which are split up using the pre-processing assumptions. Moreover, we have only assumed two reserved words *NOT* & *OR*, both in uppercase, such that any two words are assumed to be implicitly joined with an *AND* operator. *NOT* has ultimate precedence while the other two operators are given the same precedence.

The parser then processes the query to return a set of queries each of which contains words and negated words that are joined by an *AND* operation, and the entire set of queries is joined by *OR* operations. More information on the query parser can be found at `http://torch.cs.dal.ca:5000/help/parser`.

8 Search Interface

Built using an MVC architecture, the indexer and Query Engine (section 7) would be considered the Model. The templates that render the HTML would be considered the View. The URL dispatcher, and user management modules would be considered the Controller (partially coupled to query engine for allowing separate user queries). Working together they allow for this Web Search Application.

At http://torch.cs.dal.ca:5000/help/all is a brief *Help* page on how to use the search interface, along with a brief description of its modest features.

9 Code & File Sizes

The code that makes up this search engine is downloadable as a bzipped tarfile http://www.cs.dal.ca/~nassrat/6403/assign2/kid.tar.gz.

The following table shows a summary of the file sizes associated with this search engine:

File name	Size (KB)	Description
forward.bdb	5088.00	The Forward Index
inverted.bdb	25144.00	The Inverted Index
csci6403.txt	52804.68	Reuters dataset (for comparison)
stop_words.txt	1.87	Stop Word List
kid/model/indexer.py	14.30	Indexing and query engine code (Model)
kid/commands/index.py	2.17	Command line runner for the indexer
kid/controllers/search.py	3.02	Controller code for the search engine
kid/templates/base.mako	2.53	Base template for the views
kid/templates/main.mako	3.02	Main template containing search enging view
kid/templates/help.mako	2.09	Template containing help text
OTHERS	72.00	Configurations, and other files for the pylons framework

NOTES

- An electronic copy of this document is available at http://www.cs.dal.ca/~nassrat/6403/assign2/report.pdf.
- The mentioned search engine is running at http://torch.cs.dal.ca:5000

References

[1] Qdbm: Quick database manager benchmark. http://qdbm.sourceforge.net/benchmark.pdf.

[2] Douglas Crockford. RFC4627: Javascript Object Notation, 2006.

[3] Darren Gamble, John Dalbec, and Howard Chu. Re: Corrupt index files. http://www.openldap.org/lists/openldap-software/200208/msg00437.html.

[4] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter. New models in probabilistic information retrieval. 1980.