CSCI 6370: Information Retrieval Friday June 26, 2015

Search Engine Phase Two

**Objective**: To build an inverse index for our document\_corpus which takes a list of files (100 documents) as inputs and creates the inverse index as follows:

1. Tokenize each document
2. Remove Stop words and punctuation mark (using nltk library)
3. Remove special HTML tags (like <script> tag, css classes, etc)
4. Extract document title and summary (snip) save it to a “document reference” table.
5. Allow for our search engine to access the inverted index by means of a network socket.
6. Using the document reference table, the search engine (GUI) presents the results in a nicely formatted HTML webpage.

**Team and Search Engine name**: Boom

|  |  |  |
| --- | --- | --- |
| **Team member** | **Student ID** | **Role** |
| Alex Campos | 10234425 | Python Developer |
| Alvaro Leal | 10123676 | Code Reviewer, Testing and Design Improvements |
| Divya Vuppala | 20331466 | Technology research – Python Libraries and Framework |
| Akhil Choppadandi | 20331723 | Functional Testing of search engine |

**Description and Accomplishments**:

This second phase was substantially more challenging than the first phase because we had to not only create an inverted index which removes stop words and special characters but also somehow allow our search engine (graphical user interface) to access this inverted index and perform remote queries.

The structure we decided to use for the inverted index is a 2-dimentional python dictionary that lets us retrieve both the search term fast as well as update the term frequency of each word inside a given document. This dictionary uses each term as the “key” and each value is another dictionary of “document id, word frequency”. Although we are not using the document frequency to calculate page ranking, the structure we chose allows us for an easy addition of this feature in a future phase.

As far as adding search capabilities to our inverted index, we decided to experiment with distributed computing and allow our Search engine GUI to query our inverted index via a network socket. The idea here is that if needed, one can dedicate many computers to do our indexing and searching.

The challenge with this design decision was most definitely the lack of expertise doing socket programming in python so we ended up using some code we found online to both write a simple server/client application [[1]](#endnote-1) and then extend this to allow for an arbitrary message size [[2]](#endnote-2) which is recognized by our Web GUI and the indexer system.

Once a query is received by the indexer, it is parsed and looks for the special Boolean keywords “AND, OR, BUT”. If it finds them the program computes two independent vectors and then intersects them according to the Boolean logic required to produce a result vectors which is then transferred via the same socket connection as a comma separated list of matching document id’s.

On the other end, once the search engine receives the comma separated list of document id’s relevant to the search query, it proceeds to extract information from the document reference table and then used to construct the necessary data structure used to build our custom HTML response.

Here are some sample screenshots of the whole process:

1. Start our indexer/search engine:

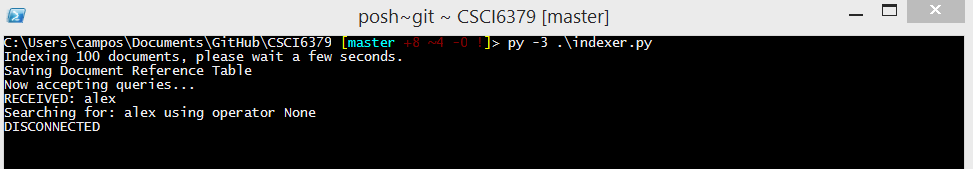


Figure 1: Our indexer program first takes care of analyzing each of the document inside our document\_corpus, creates a summary table and then waits for incomming queries via a network socket.

1. Start our Django application

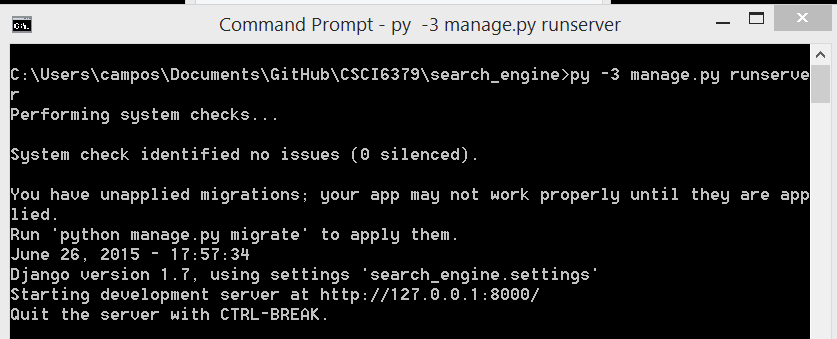
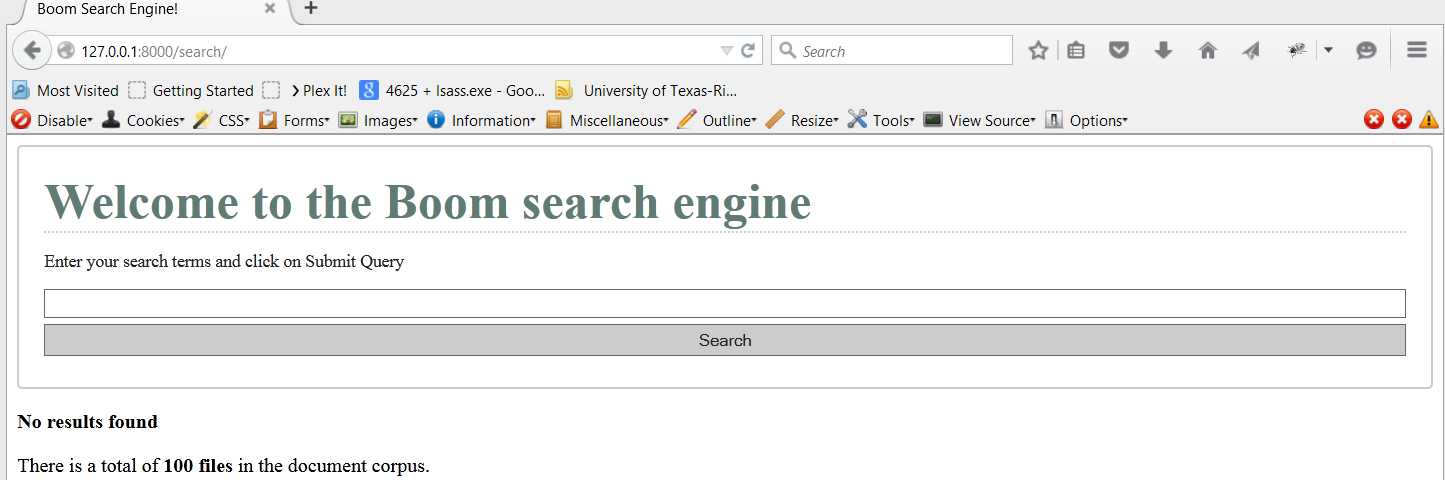
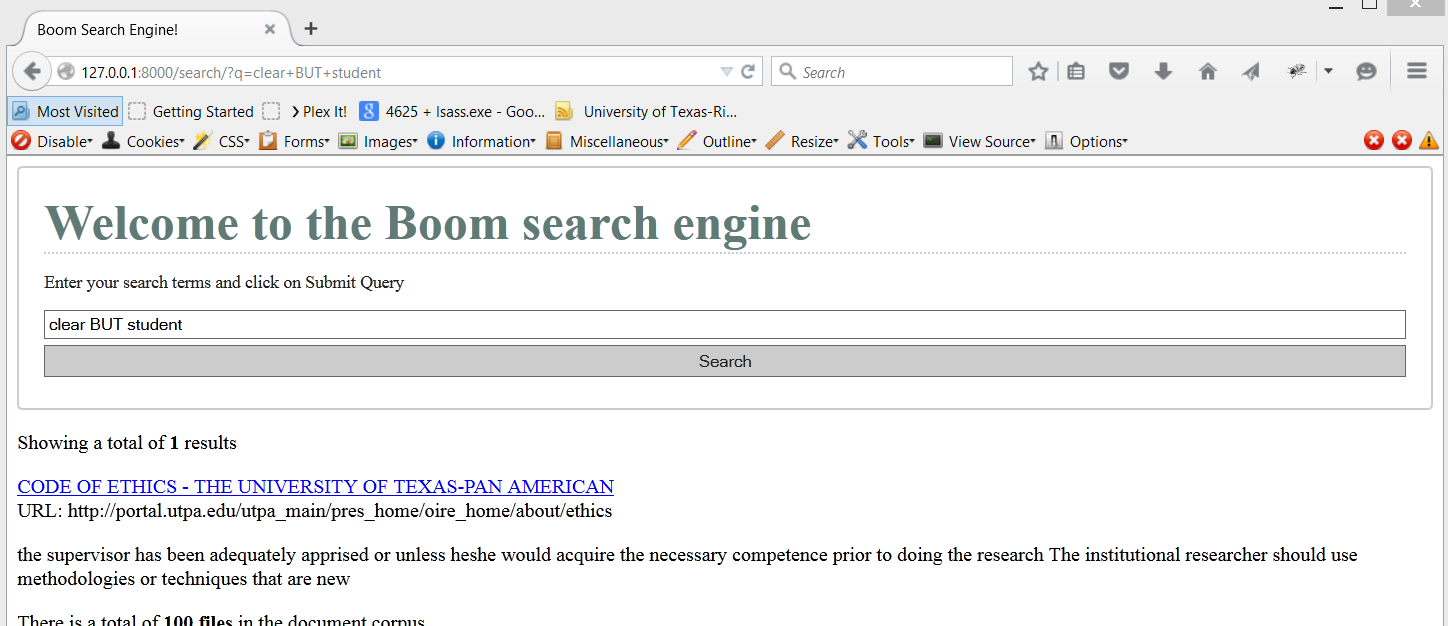


Figure 2: Starting django is as simple as running the manage.py with a runserver parameter

1. Navigate to our search engine graphical user interface accessible at <http://localhost:8000>



1. Enter keyword, in this example we used the query: clear BUT student



1. http://stackoverflow.com/questions/13979764/python-converting-sock-recv-to-string [↑](#endnote-ref-1)
2. http://stackoverflow.com/questions/17667903/python-socket-receive-large-amount-of-data [↑](#endnote-ref-2)