2. Introduction to ElasticSearch

Aleix Torres i Camps, Àlex Batlle Casellas

01/10/2021

Running ElasticSearch

First of all, we have installed ElasticSearch for Microsoft Windows using the `.msi` installer, with all the default options, and without much problem. To start the ElasticSearch service, we went into its installation directory and into the binary file directory, where we were able to start the service. We ran the provided python script `elastic_test.py` to check that everything was fine and running. We saw that the answer we received from the python script is the same as the one we get when we access `https://localhost:9200`, which makes sense, as the script simply performs a URL request and prints the result to the standard output.

Indexing and Querying

To our understanding, the provided scripts `IndexFiles.py` and `SearchIndex.py` do the following:

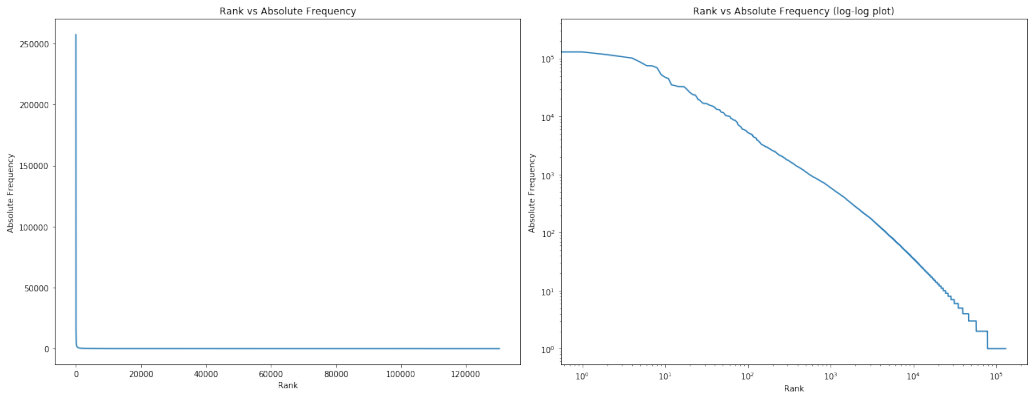
- `IndexFiles.py`: given an index name and a directory, this script creates an index in the ElasticSearch server, traversing the directory recursively and adding all of the files it finds by performing a bulk indexing operation. For each file, the script prepares an indexing operation for a document with fields `text` and `path`, which will go into the index given through a command line parameter. After preparing all of these operations, they are bulk-executed using a client for the HTTP ElasticSearch REST API, and then the index is created.
- `SearchIndex.py`: given an index name, and either a word (using the `--text` option) or a query (using the `--query` option), this script creates a query for the API to process, the result of which will match our specifications. The `--query` option has some more advanced functionalities, such as boolean operations (`AND`, `OR`, `NOT`) or fuzzy searches (`~n` where `n` is the number of characters that can be mismatched at most), and returns the document ID, its first characters (by default, the first 10, but this is obviously modifiable), and the document's path. The `--text` option is more limited but is also useful. It can only search for one word (although it can be tricked to search for more by using quotes, such as `python ... --text "word1 word2 ..."`) and returns the document ID, its path, and the words around what we wanted to find.

Redoing Session 1

To redo Exercise 8 from session 1, we execute the following line in the terminal:

```
python CountWords.py --index news > input.txt
```

Then, we manually erase the last two lines of the file `input.txt` which contain some dashes and the number of different words, and process the file with this code block, which produces the following output:



As we already concluded in the last session, we can see that the distribution of different words in a sufficiently big corpus follows a power law. We have used `CountWords.py`, which processes all of the documents in the given index by iterating over their term vectors and counting, one by one, the appearances of each term. This script also has a flag called `--alpha`, which we did not use. It returns the frequency table sorted alphabetically instead of by ascending frequency.