Lab 7 – Wikipedia Search with Elasticsearch

CC5212-1 – May 5, 2021

Today we will build a search engine for Wikipedia.

First we're going to index the URL and the keywords in the title and abstract (first paragraph) of all the articles in Spanish Wikipedia using Elasticsearch (a distributed RESTful search and analytics engine that supports distributed inverted indexing of text in documents). Then we'll implement the search function: you type in a keyword, you get back the articles that match.

- Check the sample data here: `http://aidanhogan.com/teaching/data/wiki/es/es-wiki-articles-1k.tsv`. This contains the first 1000 lines of the data, with an article encoded in each line. Each line has the URL, title, and abstract (delimited by tab) of an article. The full data (which we will use later) are available on the server, and contain 964,838 lines of the same format (encoding 964,838 articles).
- Next, we introduce Elasticsearch, which at its core offers a distributed inverted index (along with other features for indexing JSON-like documents). Elasticsearch can be interacted with through a HTTP-based RESTful API (based on `GET`, `POST`, `PUT`, `DELETE`, etc.) using JSON format requests and responses. The `curl` command used herein is a general command to send a request over HTTP and receive a response. Log into the cluster as usual. The Elasticsearch REST API is hosted at `cm:9200`.
  - To see the cluster status: `curl -X GET "cm:9200/_cat/health?v&pretty"`
  - To add a simple document to a "`customer`" index (the index will be created if it does not exist; note that this command is all one line):

    ```
    curl -X PUT "cm:9200/customer/_doc/1?pretty"
      -H 'Content-Type: application/json' -d '{ "name": "Fin Shepard" }'
    ```

  - To get an indexed document by id: `curl -X GET "cm:9200/customer/_doc/1?pretty"`
  - To show the existing indices: `curl "cm:9200/_cat/indices?v"`
  - To delete an index: `curl -X DELETE "cm:9200/customer?pretty"`
  - To create an index called `mywiki` with a single field `TITLE` of type `text`, with `store` set to true, and analyser `standard` (we will explain this in a second):

    ```
    curl -X PUT "cm:9200/mywiki?pretty" -H 'Content-Type: application/json' -d'
    {
        "mappings" : {
           "_doc" : {
               "properties" : {
                   "TITLE" : {
                        "type" : "text",
                        "store" : "true",
                        "analyzer" : "standard"
                   }
               }
           }
        }
    }
    '
    ```

So what does that last command mean? Actually there are some important concepts to clarify:

* A *field* is an attribute of a document, such as title, URL, last-updated, abstract, etc. We can later select which fields we want to search document by, so for this reason we give fields names, like `TITLE`, so we can reference them later. The content of fields can be of a `"type"`; for example, a field `TITLE` might contain text, a field `LAST-UPDATED` might contain dates, etc.
* Next, recall that inverted indexes will typically parse a piece of text – say a title or an abstract – into multiple words. Also words can be normalised (e.g., `Chilean` → `chile`), losing information about the original text. From the inverted index alone it is thus often impossible to recover the original text from the inverted index. If we want to be able to retrieve the original content later – for example, we want to display the title or abstract in the search results – we should set `"store"` to `"true"`, which asks to store the exact original content (with the cost of a larger index). While we can still search over fields with `"store"` set to `"false"`, we will not be able to recover the original content of those fields.
* Finally, the `"analyzer"` is what parses the words from the text, normalises them (including capitalisation, accents, stemming, etc.), and filters stopword. The `"standard"` analyser works generically for all languages, but we can choose more specific analysers, such as those for specific languages, like `"spanish"`. As a small example, a standard analyser will not consider a word such as `"ese"` as a stopword while the Spanish analyser will. It is important to use the same analyser for indexing and querying; otherwise, if different parsing and/or normalisation are used, "mismatches" for words normalised in different ways might arise. For this reason, Elasticsearch configures the analyser at the level of a particular index.

- While we could use the REST API to index the documents one-by-one, it would make sense to construct the index programatically. For this we will use a Java client for Elasticsearch (we could alternatively code up a client for the REST API if we wished). Download the code project from u-cursos and open it in Eclipse or your Java IDE of choice. The first Java class we'll work with will index the large data file into the Elasticsearch inverted index. The second class we'll work with will take your keyword searches and use the index to find relevant Wikipedia articles.
- In both classes, the string `@TODO` will guide you to places where you need to complete the code (there are three parts to complete overall). Also note that we use an enumeration to make sure we keep the names of fields consistent when querying and indexing; specifically, we use:
  - `FieldNames.URL.name()` for the url (which is the string `"URL"`)
  - `FieldNames.TITLE.name()` for the title (which is the string `"TITLE"`)
  - `FieldNames.ABSTRACT.name()` for the abstract (which is the string `"ABSTRACT"`)
  - `FieldNames.MODIFIED.name()` for the time (which is the string `"MODIFIED"`)

  Using `FieldNames.URL.name()` rather than the string `"URL"` (for example) makes it easier to change field names later, keep field names consistent, avoid spelling mistakes, etc.
- First open up `BuildWikiIndexBulk`. The main method is already written for you, as is some of the indexing code. Instead of indexing just the URL of the document (`tabs[0]`), however, you should also index the title of the document (`tabs[1]`), the abstract of the document (`tabs[2]` – if available, which you can check with `tabs.length > 2`), and the time at which the document was indexed (`System.currentTimeMilliseconds()`). Extend the code to index these fields.
- Next open up `SearchWikiIndex`. Again, the main method is already written for you, as is some of the searching code. Instead of searching just on the title of the document, however, you should also search on the abstract (if available). Also you need to retrieve and print details of each result to standard-out: its title (already retrieved), its URL and its abstract. Extend the code along these lines.

- Once you're finished, time to test it! Build the jar with `build.xml`. Copy the jar to your own folder on the cluster. Run the jar with a unique INDEX-NAME (better to use **lowercase** for INDEX-NAME as Elasticsearch can have problems with uppercase names):

  – `java -jar mdp-elasticsearch.jar BuildWikiIndexBulk -i` DATA `-igz -o` INDEX-NAME

  where the DATA are in `/data/uhadoop/shared/wiki/es-wiki-articles.tsv.gz`. Note that if the index was not previously defined, it will be created with default settings (including the standard analyser); any fields in indexed documents will likewise be configured with default settings.
- After the indexing has finished (it should read 964,838 lines), you can run the jar with:

  – `java -jar mdp-elasticsearch.jar SearchWikiIndex -i` INDEX-NAME

  to search the index.
- Try a search like "`obama`". Do the results make sense? Copy the query and the results.
- Run four other searches (with non-empty results) and copy the results.
- SUBMIT: your two classes – `BuildWikiIndexBulk` and `SearchWikiIndex` – to u-cursos, along with a text file containing the results of the search "`obama`" and four other searches of your choice.
- OPTIONAL: Try vary the boost factors in `SearchWikiIndex` between title and abstract and see how it affects the results of a few queries. How do the results change?
- OPTIONAL: First try a search for some Spanish stopwords like "`ese`" on your index with default settings (standard analyser). Now index the data again, but before you call `BuildWikiIndexBulk`, define the index using the command outlined at the start of the lab along with the fields it will contain. You can choose to store all fields. The `"type"` of `MODIFIED` will be `"long"` and the `"type"` of `URL` will be `"keyword"` (an exact string that will not be parsed into words); you should omit the analyser attribute for these fields as it does not apply to such values. Switch the analyser to `"spanish"` for the two other fields. Index again and try the searches with the same stopwords. Any changes? Can you detect other differences, maybe in how words are stemmed, how accents are handled, etc.?