Méthodes d'accès aux données 2021–2022
Professeur : Nastaran Fatemi
Assistants : Christopher Meier
Edward Ransome

Labo 1 : Indexing and Search with *Elasticsearch*

1 Introduction

1.1 Objectives

The goal of this lab is to discover *Elasticsearch*, which is a distributed RESTful search engine used widely in the industry based on Apache Lucene (a free and open-source information retrieval software library). You will learn how to use various functionalities of *Elasticsearch* to index and search a collection of scientific publications.

1.2 Organization

The lab is realized in groups of maximum 2 students.

Deadline: See deadline on *Moodle*.

Report: You are kindly asked to submit a report containing both the answers to the questions and the API requests. In the current document, the report tag in green colour highlights the questions we are expecting to be answered in the report.

REPORT An example of report tag. The deliverable is specified with a number D.X.

Please respect the format of the API requests, the same as in *Elasticsearch* documentation:

```
PUT  cacm_raw/_doc/1

{
    "_row": "<id>\t<author 1>;<author 2>\t<title>\t<summary>"
}
```

1.3 Setup

Follow the steps described in the "Getting started with Elasticsearch" document.

2 Indexing and Searching the CACM collection

We are now going to use *Elasticsearch* to index and analyze a list of scientific publications.

In this lab, you will manually index the publication list, will perform a few queries, and will answer the questions by running API Requests.

2.1 Description of the collection

We will use the famous text corpus, CACM, which you imported in "Getting started with Elasticsearch".

As a remainder, each document in the collection contains:

- the publication id
- the authors (if any)

- the title
- the date of publication (year and month)
- the summary (if any)

There might be publications without any author or without the summary field.

2.2 Indexing

The goal of this part is to explicitly index the CACM publication collection, by manually specifying the mapping.

Elasticsearch provides many field types. Most can be extensively configured. Take a look at the documentation.

- Use the `standard` analyzer for this part (It's configured by default).
- Use the appropriate field types. Take a look at the common types and the text search types. You will essentially need: numeric, date, text and keyword
  - Enable queries on author, title, date and summary attributes
  - Keep the publication id in the index and show it in the results set of queries (without needing to read the `_source` field), but do not enable queries on this field.
  - Note how *Elasticsearch* deals with arrays.
  - For the summary field, store the offsets in the index.
  - In this lab, always activate `fielddata` for fields of type `text`.

Do the following steps:

1. Create an index called `cacm_standard` which matches the above configuration. Reindex to add the documents of the collection to the index.
2. Find out what is a "term vector" in *Elasticsearch* vocabulary. Create an index called `cacm_termvector` which matches the above configuration and allows access to the "term vector" of the `summary` field. Reindex. Use the Term vectors API to check that the "term vector" is included in the index.
3. Compare the size of the `cacm_standard` and `cacm_termvector` indices, discuss the results.

REPORT
Attach the API request for creating the indices (`cacm_standard` D.1 and `cacm_termvector` D.2).
Attach the API request to check the presence of the term vector D.3.
Answer the questions (term vector D.4, size of index with discussion D.5).

2.3 Reading Index

Use Terms aggregation to get info on the `cacm_standard` index. Aggregation on text field requires activating `fielddata`.

Hint: Setting the global search size to 0, prevents including unwanted `hits` in the response. The interesting information is inside the `aggregations` field of the response.

Answer the following questions:

1. Who is the author with the highest number of publications? How many publications does he/she have?
2. List the top 10 terms in the title field together with their frequency.

REPORT
Attach the API requests and your answers (author D.6, titles D.7).

2.4 Using different Analyzers

Elasticsearch provides different analyzers to process a document. Below, we will test some of these analyzers. Note that here indexing and searching will use the same analyzer.

Apply the same configuration of the fields as in section 2.2.

Attention: Remember to create a different index for each analyzer.

1. Index the publication list using each of the following analyzers :
   - `whitespace`
   - `english`
   - Custom analyzer based on the `standard` analyzer (lowercase filter and standard tokenizer) but outputs shingles[1] of size 1 and 2
   - Custom analyzer based on the `standard` analyzer but outputs shingles of size 3
   - `stop` with a custom stop list. A list of common words is provided in the file `common_words.txt` of the archive. Use this list as stopwords[2].
2. Reindex and save the response for each, so that you can answer questions later.
3. Explain the difference of these five analyzers.
4. Look at the index using Terms aggregation and the Index stats API and for each created index find out the following information:
   a. The number of indexed documents.
   b. The number of indexed terms in the summary field.
   c. The top 10 frequent terms of the summary field in the index.
   d. The size of the index on disk.
   e. The required time for indexing (e.g. using `took` field from response to reindex).
5. Make 3 concluding statements bases on the above observations.

REPORT For each analyzer attach the API request used to create the index D.8.
Attach your answers into the report (analyzer differences D.9, index statistics for each analyzer D.10, concluding statements D.11).

2.5 Searching

On the index with the `english` analyzer, use Query string query to perform the following queries on the `summary` field:

1. Publications containing the term "Information Retrieval".
2. Publications containing both "Information" and "Retrieval".
3. Publications containing at least the term "Retrieval" and, possibly "Information" but not "Database".
4. Publications containing a term starting with "Info".
5. Publications containing the term "Information" close to "Retrieval" (max distance 5).

Each query should only return the `id` field of the matching documents (it should not return the whole document).

REPORT For each query: provide the API request D.12, the total number of results D.13.

[1] Shingles are n-grams of words
[2] The data folder which contains the file is mounted inside the config folder of *Elasticsearch*

2.6 Tuning the Lucene Score

The goal of this part is to modify the way documents are ranked against a query.

2.6.1 Custom similarity

Create a new index based on `cacm_standard` which calculates it's similarity score based on the following formula (query boost is provided by *Elasticsearch*):

$$\begin{aligned} \text{score} &= \text{tf} \cdot \text{idf} \cdot \text{norm} \cdot \text{query boost} \\ \text{tf} &= 1 + \log freq \\ \text{idf} &= \log\left(\frac{numDoc}{docFreq + 1}\right) + 1 \\ \text{norm} &= 1 \end{aligned}$$

Compute the query `compiler program` with the default similarity function and with the similarity function using the above parameters. Show the top 10 results (ids with scores) of both similarity functions.

2.6.2 Function score

Compute the query `compiler program` with a function score that calculates the score of documents considering the two following constraints:

- The score of the document reduces linearly as a function of the distance of the publication date and a given reference date, January 1970.
- The score of a document published 90 days before or after the reference date should be halved.

Here is an example of weighted scores using the function score (considering that all months have 30 days):

1969-10	score × 0.5
1969-11	score × 0.666
1969-12	score × 0.833
1970-01	score × 1
1970-02	score × 0.833
1970-07	score × 0

REPORT

Attach the API request to create an index which uses the custom scoring D.14.
Attach the top 10 results and scores with and without custom scoring D.15.
Attach the API request to query document including the function score D.16.