In this session:

- You will learn how to use the ElasticSearch database
- How to index a set of documents and how to ask simple queries about these documents
- How to do this from Python.

The goal of this session is not just to execute the scripts provided, but to understand them. Otherwise you will not be prepared for the next sessions in which you will need to write scripts of your own to access Elastic Search.

1 ElasticSearch

ElasticSearch is a NoSQL/document database with the capability of indexing and searching text documents. As a rough analogue, we can use the following table for the equivalence between ElasticSearch and a more classical relational database:

Relational DB	*ElasticSearch*
Database	Index
Table	Type
Row/Record	Document
Column	Field

An index can be thought of as an optimized collection of documents and each document is a collection of fields, which are the key-value pairs that contain your data.

2 Running ElasticSearch

First you will need to install ElasticSearch following instructions in their documentation.

This database runs as a web service in a machine and can be accessed using a REST web API; however we will interact with the database through its python libraries `elasticsearch-py` and `elasticsearch-dsl`, so you will need to install these as well. You can run ElasticSearch by typing:

```
$ elasticsearch &
```

After a few seconds (and a lot of logging) the database will be up and running; you may need to hit return for the prompt to show up. To test if ElasticSearch is working you have a script called `elastic_test.py`, you can run it from the command line as:

```
$ python elastic_test.py
```

If ElasticSearch is working you will see an answer from the server; otherwise you will see a message indicating that it is not running. You can try also throwing the URL `http://localhost:9200` to your browser; you should get a similar answer.

3 Indexing and querying

ElasticSearch is a pretty big beast with many options.

- Here is the full documentation.
- A simple intro using Python.
- Other partial intros you want to have a look at: here, here, here.
- One on using the preprocessors (called Analyzers in ES/Lucene)
- You found another one that you liked? Let us know.

As mentioned before, ElasticSearch is accessed as a web service using a REST API. This means that we can operate with the DB using the HTTP protocol (just like any web server). ElasticSearch defines a set of methods that correspond to all the actions available.

To make things simpler we are going to use two python libraries (`elasticsearch` and `elasticsearch-dsl`) to access ElasticSearch. The first one is more general but hides less the complexities of the API calls, the second one is more focused on the search capabilities and is more friendly for sending queries to ElasticSearch. In the scripts that you have, both libraries are used depending on the operations performed.

3.1 Anatomy of an indexing

To use a simile with relational databases, an index in ElasticSearch corresponds to a table. There is not a specific schema (set of columns) associated with an index, we can insert different types of documents that can have different fields. We can tell to the DB what fields have to be indexed. All documents sent to the DB are serialized using the `JSON` format, so a document will look like:

```
{
"field1": "2017-01-31",
"field2": "Some text here",
"field3": 33
}
```

We provide three zipped sets of files: `20_newsgroups.zip`, `novels.zip` and `arxiv_abs.zip`. Unzip them in your working directory, to directories `20_newsgroups`, `novels` and `arxiv_abs`. They contain respectively:

- Text from 20 usenet groups on various topics, a classic corpus in IR evaluation, from here.
- A number of random novels and other texts in English from the Gutenberg project, with a tendency towards late 19th and early 20th centuries.
- A small collection of abstract from recent scientific papers from different areas from here.

You have among the session files a script named `IndexFiles.py`. Open the script with a text editor.

The script has basically two parts, the first one reads all the documents traversing recursively the path that is received as a parameter and creates a list of ElasticSearch operations. There is a method in the API for indexing just one file, but given that we are going to index a lot of them, we can use the `bulk` method that allows executing several ElasticSearch calls as one.

A bulk operation is a special document that indicates the type of operation to execute (`_op_type`), in this case `index`, the index where the operation is performed (`_index`) and the information of the document: the type of the document and the fields of the document. In this case we have used as type `document` and we include two fields in the document, `path` for the path of the file and `text` for the content of the file.

The second part of the script operates with ElasticSearch. First we need an `ElasticSearch` object that will manage the connection with the DB, if we have the service in the local machine with the default configuration, no parameters are needed, in any other case we must configure the object adequately. With this object we create the index for the documents (if it already exists, it is deleted) and the bulk method is called for performing all the insertions.

Now you can use this script with the documents that you have downloaded, for example:

```
$ python IndexFiles.py --index news --path /path/to/20_newsgroups
```

will insert all the documents from `20_newsgroups` in the index `news`

3.2 Looking for mr goodword

After we have all the documents inside an index we can query ElasticSearch using specific terms or more complicated queries. ElasticSearch has a DSL (Domain Specific Language) for specifying what we want to search. We are not going to get into the complexities of what can be done, but this kind of DBs allows more flexibility than the classical relational DB. Apart from exact matches we can do fuzzy matches, obtain suggestions, highlight the parts of the document that includes the search term... Basically anything that we expect from a web search engine, because this kind of DBs are the ones that are behind a query in Google search for example.

The script `SearchIndex.py` allows to query an index in our ElasticSearch DB. It has three flags `--index` is obviously the index and we also have `--text` and `--query` parameters.

The `--text` flag takes precedence over `--query` (you can not use both) and allows for looking for a word in the `text` field of our documents. It will return all the exact matches with the document id, the path of the file that matches and the highlighted parts that contain the word.

The `--query` flag allows to use the LUCENE syntax for the queries (LUCENE is the open source indexing system all DBs of this kind use). This syntax allows boolean operations like AND, OR, NOT (always in uppercase) or fuzzy searches using `~n` with n indicating how many letters of the word can mismatch to consider it a match. Using this flag the query returns the id of the documents, the path and the 10 first characters of the document.

Open the script with a text editor and look to the part of the code where the search query is built. Have a look at the documentation of `elasticsearch-dsl` to understand it better.

Now you can play a little bit with the script, for example execute:

```
$ python SearchIndex.py --index news --text good
$ python SearchIndex.py --index news --query good AND evil
$ python SearchIndex.py --index news --text angle
$ python SearchIndex.py --index news --query angle~2
```

Each search returns the number of documents matched. Observe how this number changes using larger values for n in fuzzy queries. If this does not behave as expected, try to figure out what it is happening by looking into the `max_expansions` parameter of ElasticSearch.

4 Redoing Session 1

Suppose we want to redo Exercise 8 of Session 1 (extracting words from text) on a database of texts in Elastic Search.

Study the `CountWords.py` script. It reads an index and writes on the standard output all the terms it contains, with their counts.

Execute it redirecting the output to a file. You have two flags, the obvious `--index` and an `--alpha` flag that returns the list alphabetically instead of ascending by the number of occurrences of the word. Beware of that the last line is the number of words.

5 Rules of delivery

1. No plagiarism; don't discuss your work with other teams. You can ask for help to others for simple things, such as recalling a python instruction or module, but nothing too specific to the session.

2. If you feel you are spending much more time than the rest of the classmates, ask us for help. Questions can be asked either in person or by email, and you'll never be penalized by asking questions, no matter how stupid they look in retrospect.

3. Write a short report with your results and thoughts. Did you succeed in running all the scripts? Did you get the expected results? More importantly, did you understand the scripts? Make it at most 1 page, so you cannot put large tables or plots - we are not interested in these. You are welcome to add conclusions and findings that depart from what we asked you to do. We encourage you to discuss the difficulties you find; this lets us give you help and also improve the lab session for future editions.

4. Turn the report to PDF. Make sure it has your names, date, and title.

5. Submit your work through the Racó. There will be a Pràctica open for each report.

   Deadline: Work must be delivered within 2 weeks from the end of the lab session. Late submissions risk being penalized or not accepted at all. If you anticipate problems with the deadline, tell us as soon as possible.