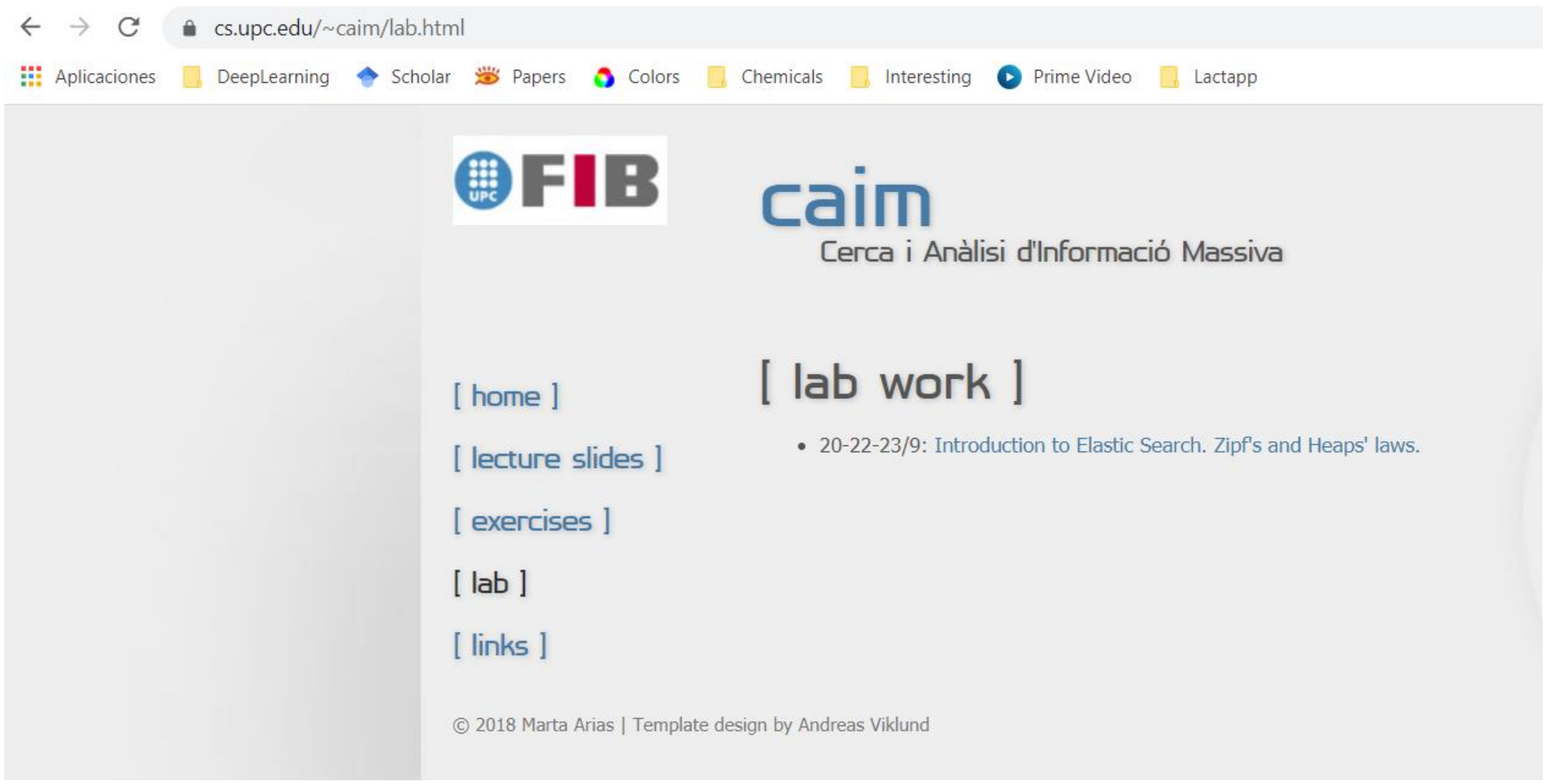
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

- CAIM Grupo 13 de Laboratorio
- Empezamos a las 12:05

Introducción

- Clases presenciales
- Presentación de la clase de laboratorio 20-30 minutos
    - Sed puntuales (empezamos a las 12:05)
    - Estad atentos, tras la presentación hay tiempo para desarrollar la sesión
- Trabajo en equipo supervisado
    - Grupos de dos personas
    - Mismo compañero para todas las prácticas
- Dudas y consultas
    - Presenciales en horario de clase
    - E-mail (Ignasi.gomez@upc.edu)
    - Sesiones individuales de google meet para consultas complejas
    - Grupo de chat en gmail
- Entregas:
    - Dos semanas desde que se hace la primera sesión de laboratorio del tema.
    - En un mundo ideal, esto coincide con el inicio de la siguiente sesión.
    - El 28 de Septiembre Comenzamos la segunda parte de Elastic Search y se entrega el trabajo de la primera, que vemos en esta sesión de laboratorio.
    - La entrega se cierra al iniciar la nueva sesión de laboratorio, en vuestro caso a las 12:00

Introducción


cs.upc.edu/~caim/lab.html

Aplicaciones DeepLearning Scholar Papers Colors Chemicals Interesting Prime Video Lactapp

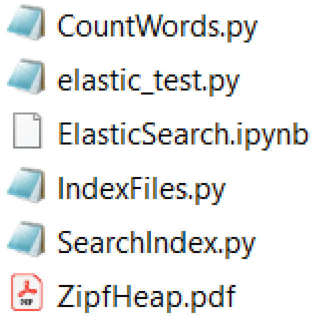
caim
Cerca i Anàlisi d'Informació Massiva

[home]
[lecture slides]
[exercises]
[lab]
[links]

[lab work]

- 20-22-23/9: Introduction to Elastic Search. Zipf's and Heaps' laws.

© 2018 Marta Arias | Template design by Andreas Viklund

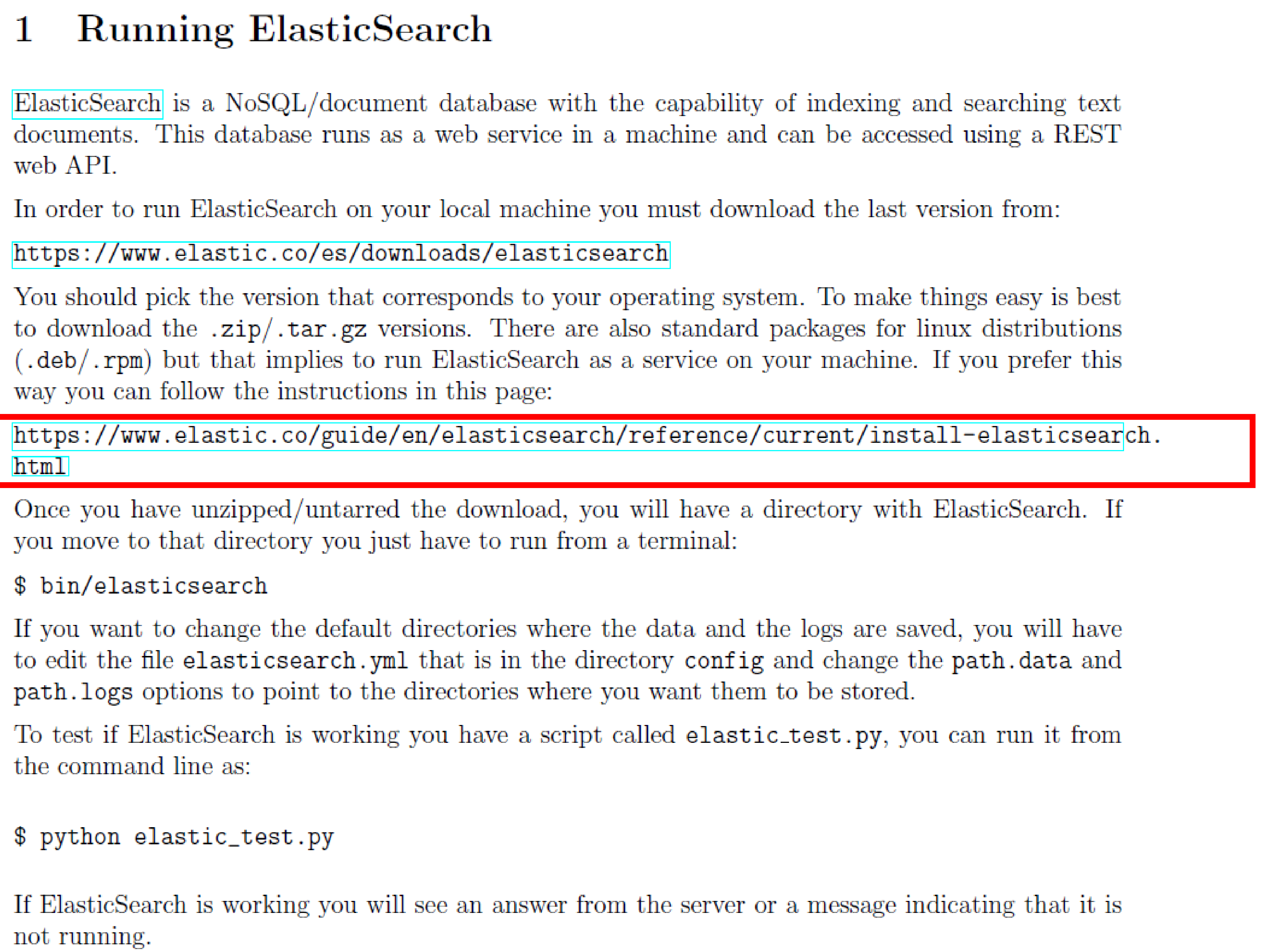

CountWords.py
elastic_test.py
ElasticSearch.ipynb
IndexFiles.py
SearchIndex.py
ZipfHeap.pdf

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.
- We will study whether the given texts satisfy Zipf's and Heaps' laws.

Instalar ElasticSearch I

Recomendado si usáis Linux, virtualbox o WSL



1 Running ElasticSearch

ElasticSearch is a NoSQL/document database with the capability of indexing and searching text documents. This database runs as a web service in a machine and can be accessed using a REST web API.

In order to run ElasticSearch on your local machine you must download the last version from:

https://www.elastic.co/es/downloads/elasticsearch

You should pick the version that corresponds to your operating system. To make things easy is best to download the .zip/.tar.gz versions. There are also standard packages for linux distributions (.deb/.rpm) but that implies to run ElasticSearch as a service on your machine. If you prefer this way you can follow the instructions in this page:

https://www.elastic.co/guide/en/elasticsearch/reference/current/install-elasticsearch.html

Once you have unzipped/untarred the download, you will have a directory with ElasticSearch. If you move to that directory you just have to run from a terminal:

```
$ bin/elasticsearch
```

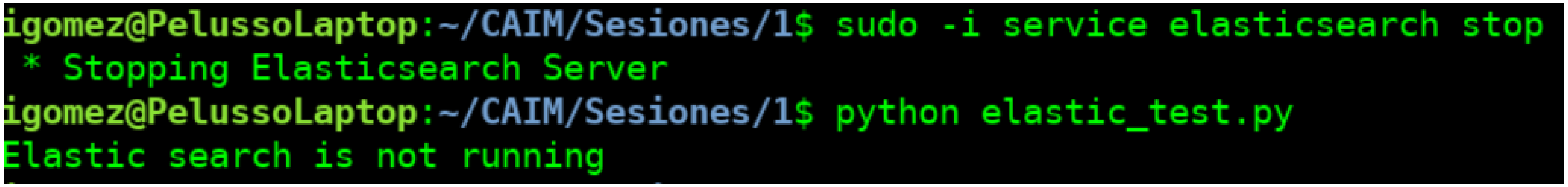
If you want to change the default directories where the data and the logs are saved, you will have to edit the file `elasticsearch.yml` that is in the directory `config` and change the `path.data` and `path.logs` options to point to the directories where you want them to be stored.

To test if ElasticSearch is working you have a script called `elastic_test.py`, you can run it from the command line as:

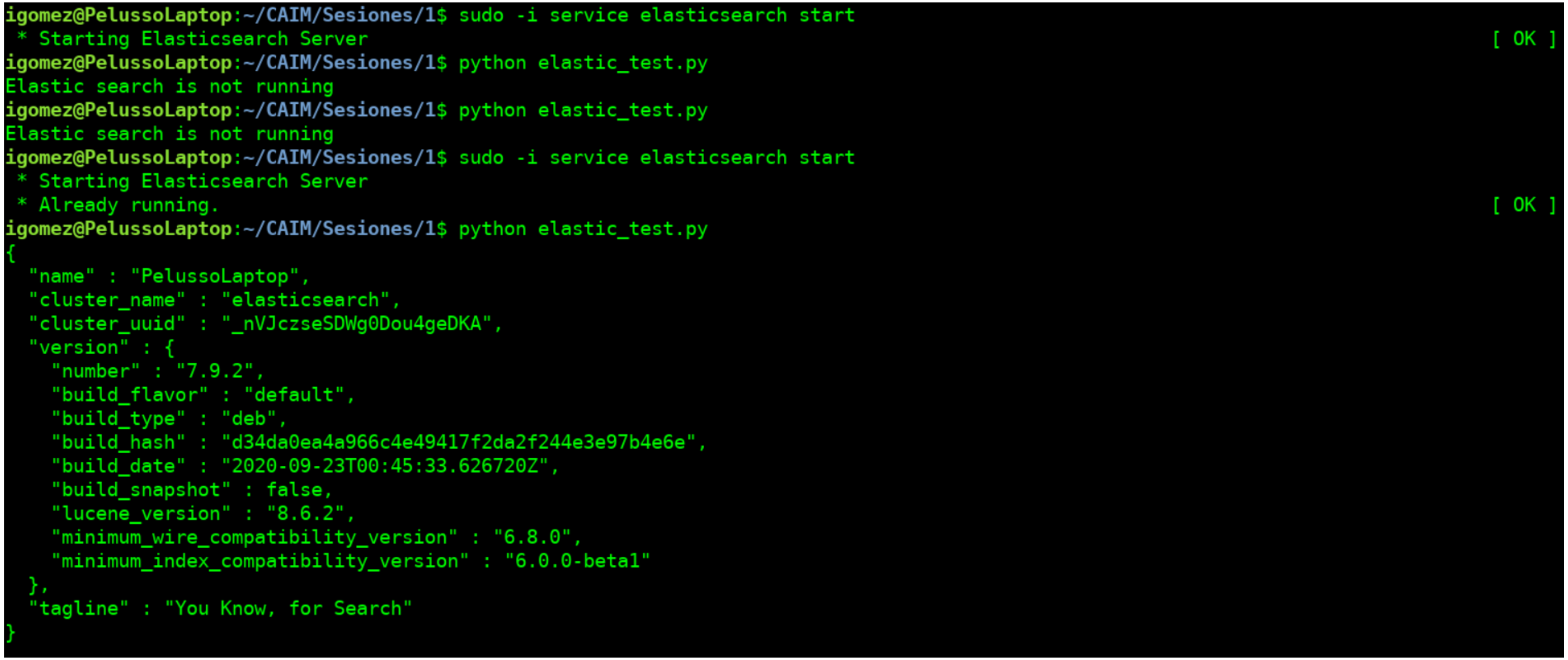
```
$ python elastic_test.py
```

If ElasticSearch is working you will see an answer from the server or a message indicating that it is not running.

Instalar ElasticSearch II

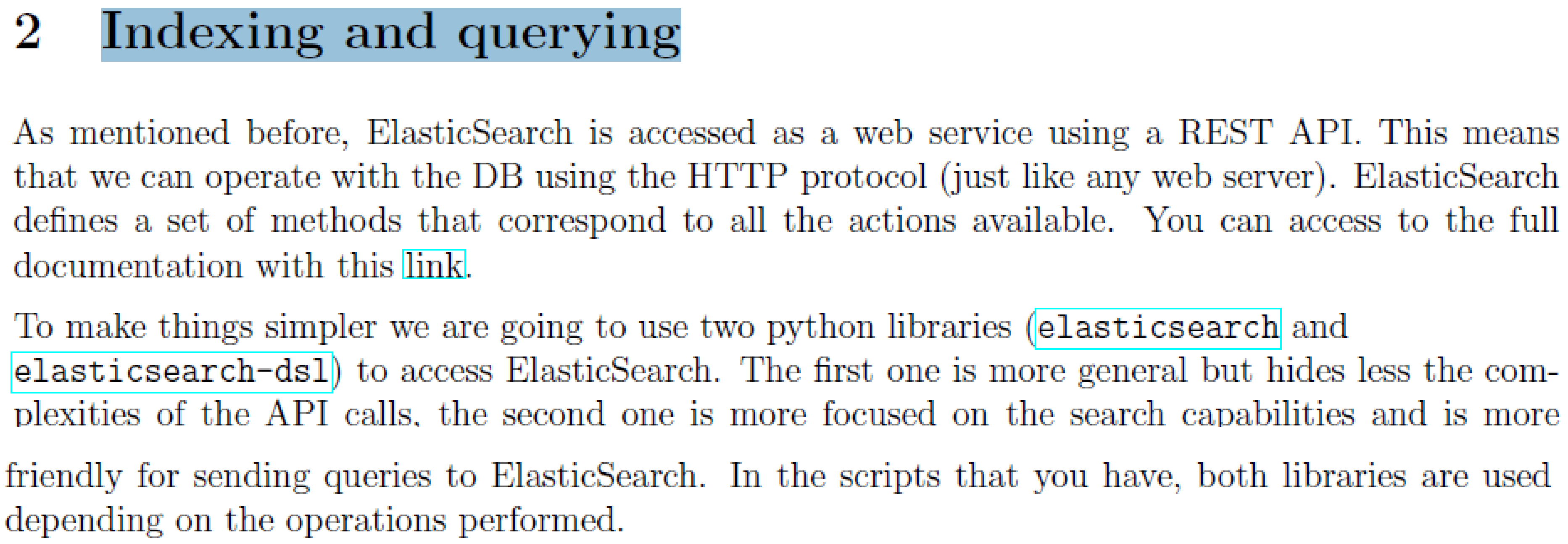


```
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ sudo -i service elasticsearch stop
 * Stopping Elasticsearch Server
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python elastic_test.py
Elastic search is not running
```



```
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ sudo -i service elasticsearch start
 * Starting Elasticsearch Server                                                              [ OK ]
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python elastic_test.py
Elastic search is not running
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python elastic_test.py
Elastic search is not running
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ sudo -i service elasticsearch start
 * Starting Elasticsearch Server
 * Already running.                                                                           [ OK ]
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python elastic_test.py
{
  "name" : "PelussoLaptop",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "_nVJczseSDWg0Dou4geDKA",
  "version" : {
    "number" : "7.9.2",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "d34da0ea4a966c4e49417f2da2f244e3e97b4e6e",
    "build_date" : "2020-09-23T00:45:33.626720Z",
    "build_snapshot" : false,
    "lucene_version" : "8.6.2",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Indexing and querying I

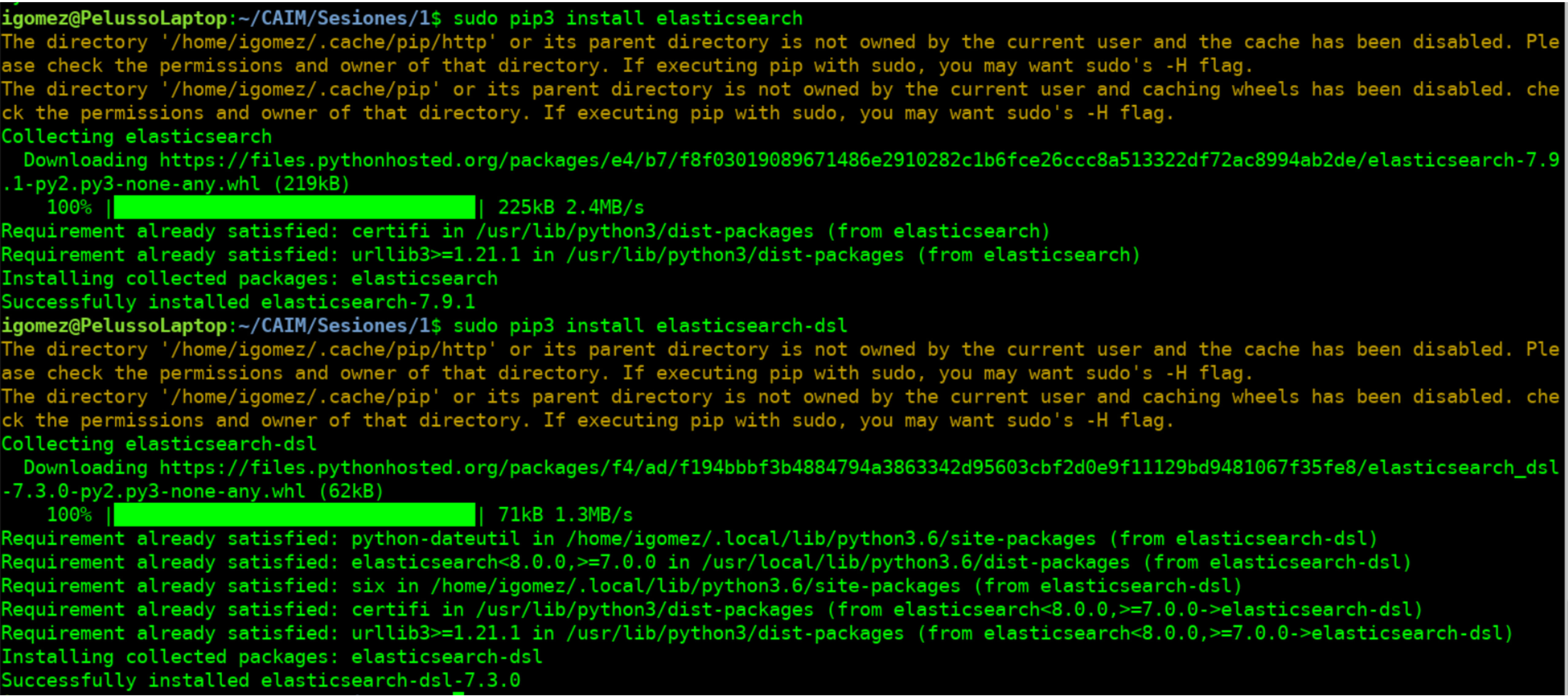


2 Indexing and querying

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. You can access to the full documentation with this link.

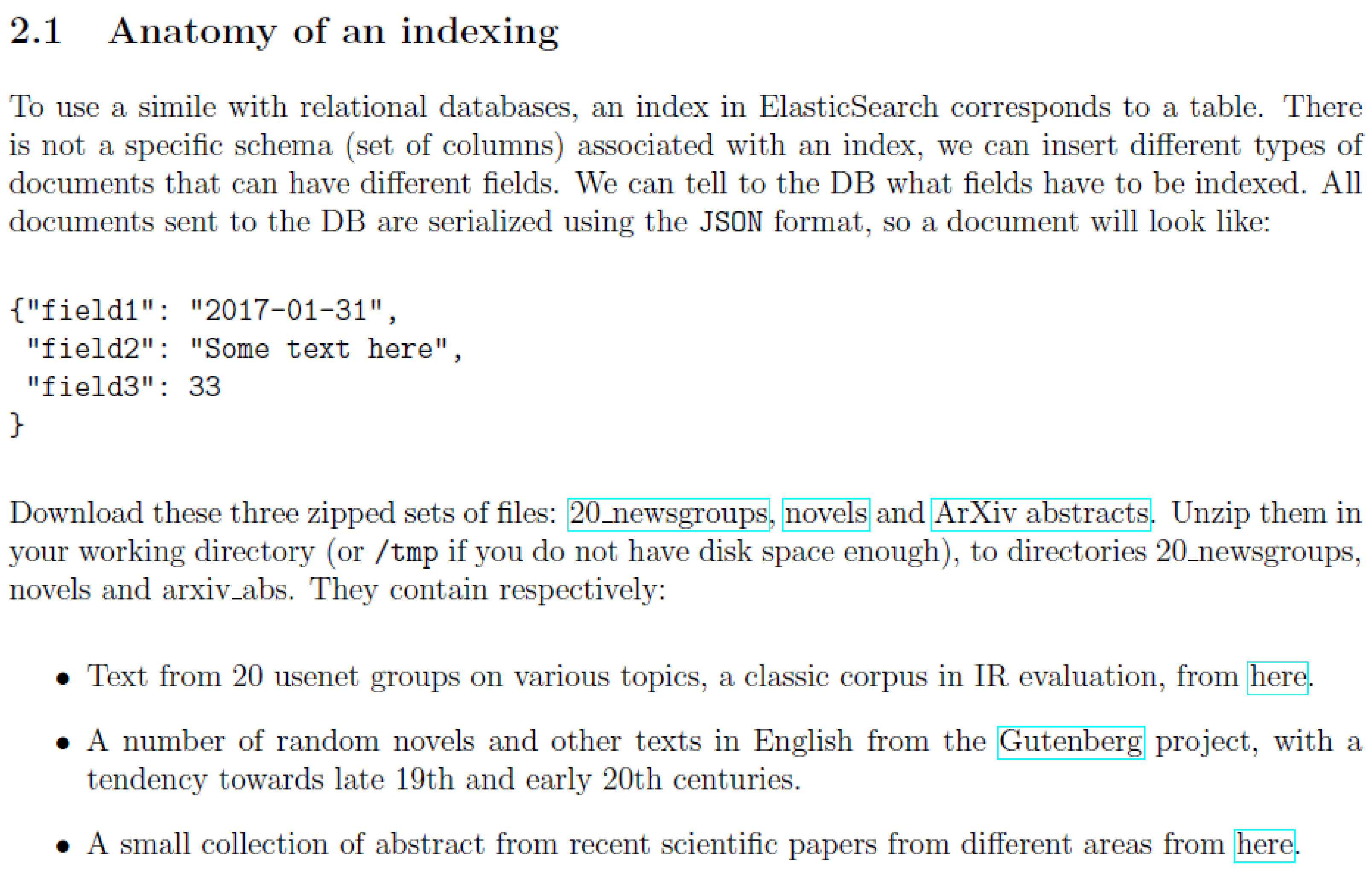
To make things simpler we are going to use two python libraries (`elasticsearch` and `elasticsearch-dsl`) to access ElasticSearch. The first one is more general but hides less the complexities of the API calls, the second one is more focused on the search capabilities and is more friendly for sending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

Indexing and querying II



```
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ sudo pip3 install elasticsearch
The directory '/home/igomez/.cache/pip/http' or its parent directory is not owned by the current user and the cache has been disabled. Ple
ase check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/igomez/.cache/pip' or its parent directory is not owned by the current user and caching wheels has been disabled. che
ck the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting elasticsearch
  Downloading https://files.pythonhosted.org/packages/e4/b7/f8f03019089671486e2910282c1b6fce26ccc8a513322df72ac8994ab2de/elasticsearch-7.9
.1-py2.py3-none-any.whl (219kB)
    100% |████████████████████████████████| 225kB 2.4MB/s
Requirement already satisfied: certifi in /usr/lib/python3/dist-packages (from elasticsearch)
Requirement already satisfied: urllib3>=1.21.1 in /usr/lib/python3/dist-packages (from elasticsearch)
Installing collected packages: elasticsearch
Successfully installed elasticsearch-7.9.1
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ sudo pip3 install elasticsearch-dsl
The directory '/home/igomez/.cache/pip/http' or its parent directory is not owned by the current user and the cache has been disabled. Ple
ase check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/igomez/.cache/pip' or its parent directory is not owned by the current user and caching wheels has been disabled. che
ck the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting elasticsearch-dsl
  Downloading https://files.pythonhosted.org/packages/f4/ad/f194bbbf3b4884794a3863342d95603cbf2d0e9f11129bd9481067f35fe8/elasticsearch_dsl
-7.3.0-py2.py3-none-any.whl (62kB)
    100% |████████████████████████████████| 71kB 1.3MB/s
Requirement already satisfied: python-dateutil in /home/igomez/.local/lib/python3.6/site-packages (from elasticsearch-dsl)
Requirement already satisfied: elasticsearch<8.0.0,>=7.0.0 in /usr/local/lib/python3.6/dist-packages (from elasticsearch-dsl)
Requirement already satisfied: six in /home/igomez/.local/lib/python3.6/site-packages (from elasticsearch-dsl)
Requirement already satisfied: certifi in /usr/lib/python3/dist-packages (from elasticsearch<8.0.0,>=7.0.0->elasticsearch-dsl)
Requirement already satisfied: urllib3>=1.21.1 in /usr/lib/python3/dist-packages (from elasticsearch<8.0.0,>=7.0.0->elasticsearch-dsl)
Installing collected packages: elasticsearch-dsl
Successfully installed elasticsearch-dsl-7.3.0
```

Indexing I



2.1 Anatomy of an indexing

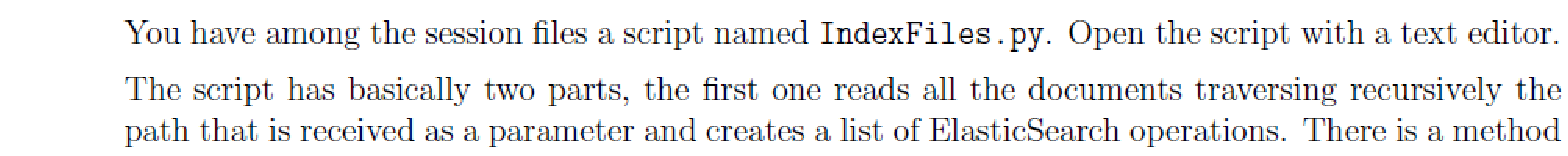
To use a simile with relational databases, an index in ElasticSearch corresponds to a table. There is not a specific schema (set of columns) associated with an index, we can insert different types of documents that can have different fields. We can tell to the DB what fields have to be indexed. All documents 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
}
```

Download these three zipped sets of files: 20_newsgroups, novels and ArXiv abstracts. Unzip them in your working directory (or `/tmp` if you do not have disk space enough), to directories 20_newsgroups, novels and arxiv_abs. They contain respectively:

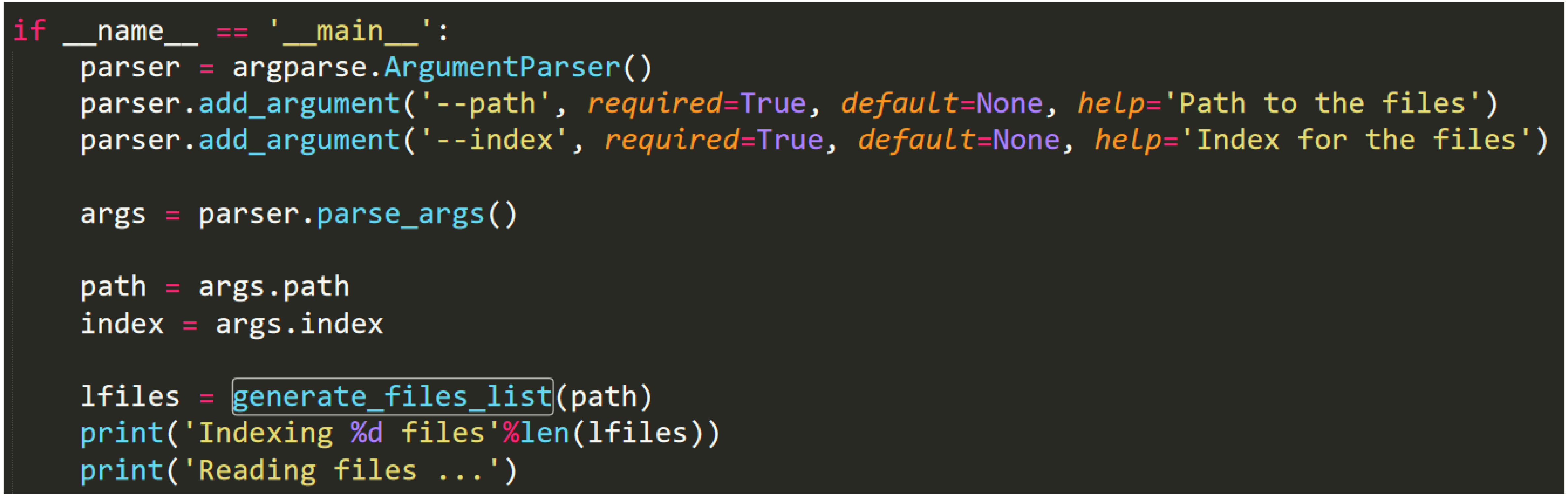
- Text from 20 usenet groups on various topics, a classic corpus in IR evaluation, from here.
- A number of random novels and other texts in English from the Gutenberg project, with a tendency towards late 19th and early 20th centuries.
- A small collection of abstract from recent scientific papers from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

Indexing II



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



```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--path', required=True, default=None, help='Path to the files')
    parser.add_argument('--index', required=True, default=None, help='Index for the files')

    args = parser.parse_args()

    path = args.path
    index = args.index

    lfiles = generate_files_list(path)
    print('Indexing %d files'%len(lfiles))
    print('Reading files ...')
```

Indexing III

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.

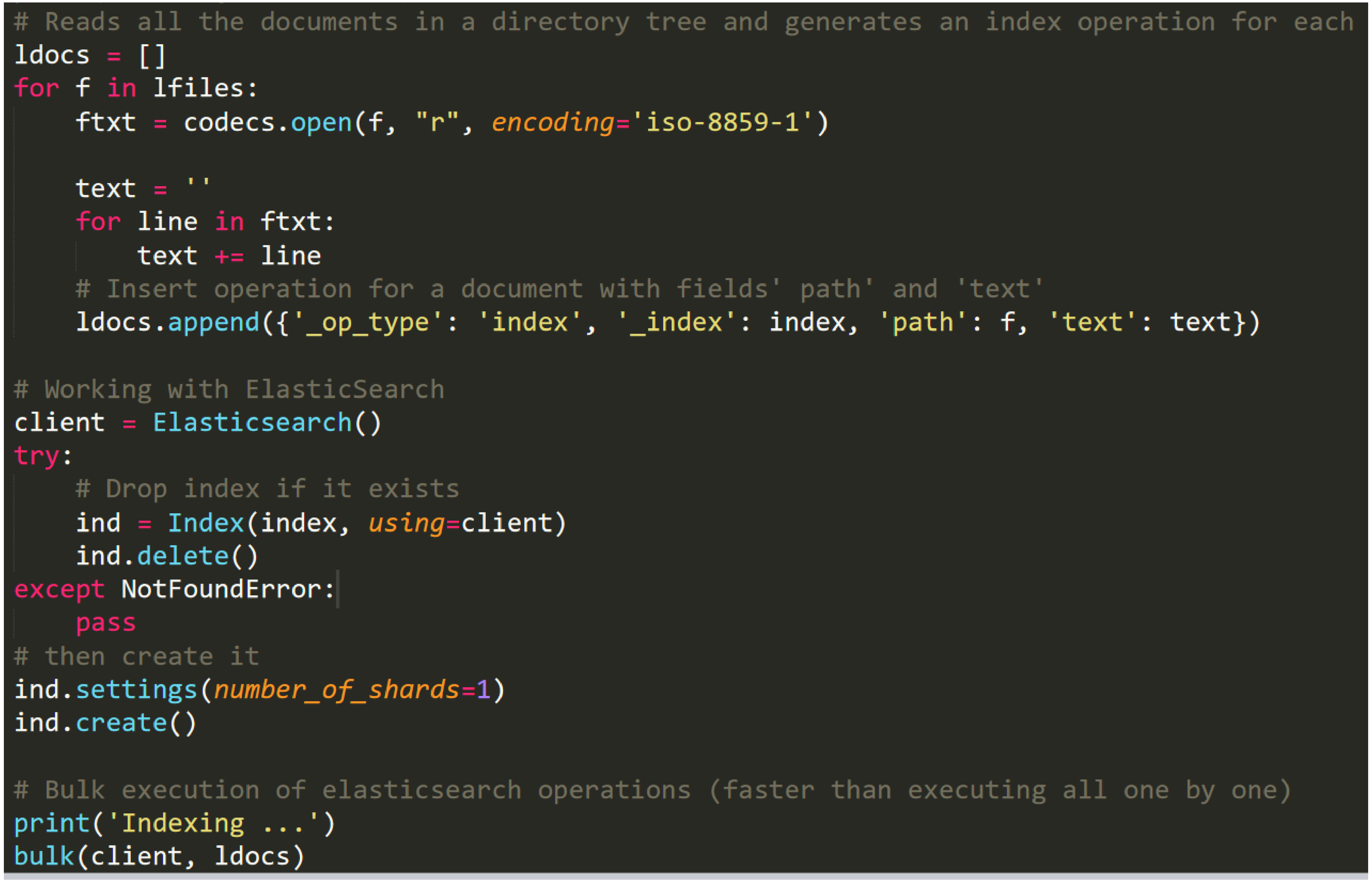
The second part of the script operates with ElasticSearch. First we need an `ElasticSearch` object that will manage the connection with the DB, if we have the service in the local machine with the default configuration, no parameters are needed, in any other case we must configure the object adequately. 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
```

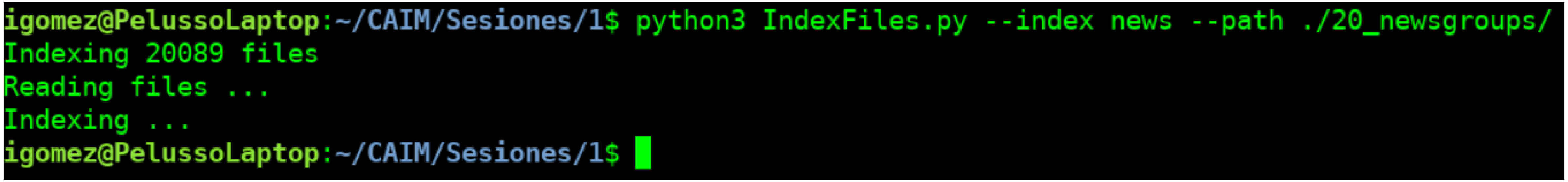
will insert all the documents from `20_newsgroups` in the index `news`

Indexing IV



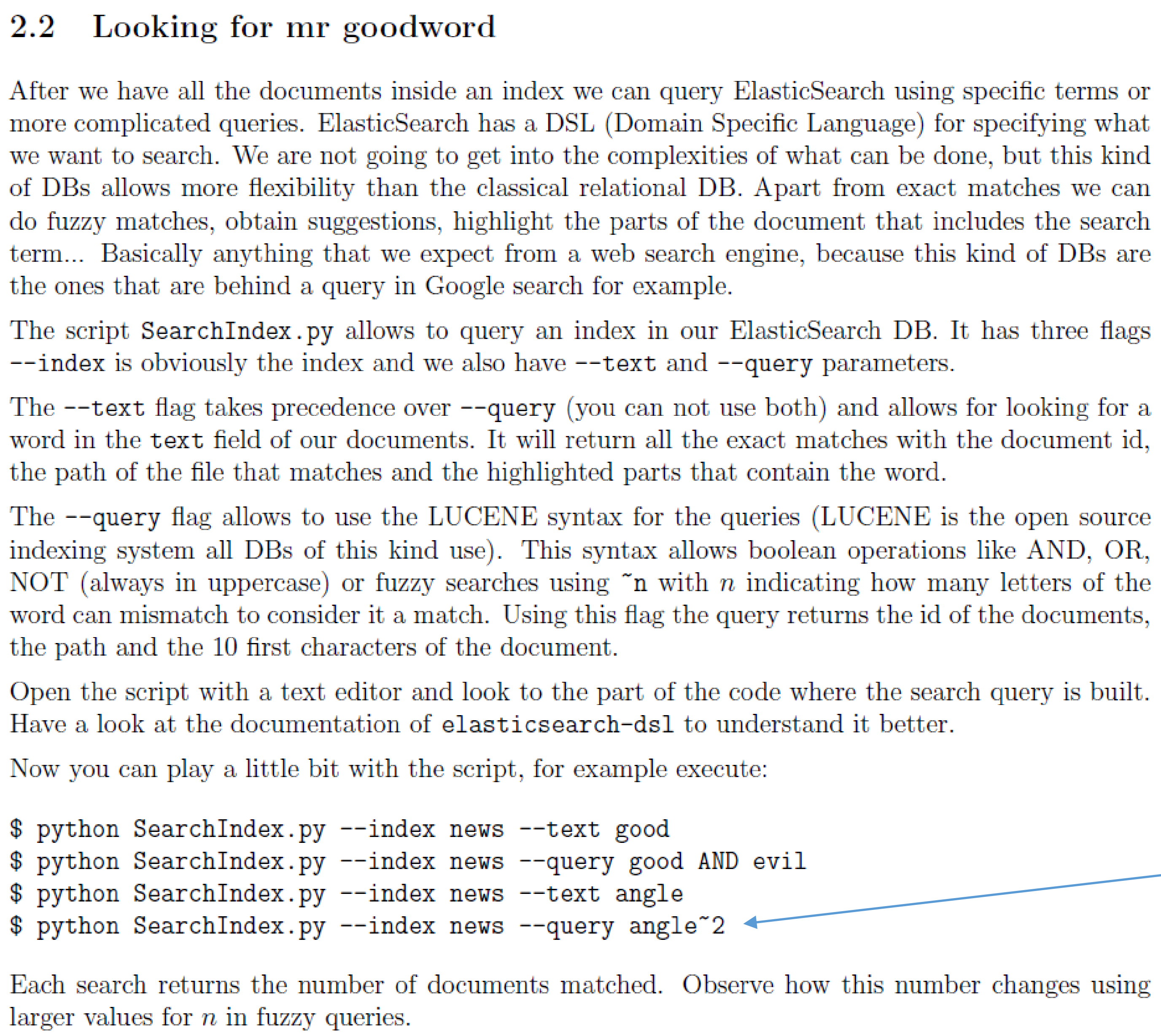
```python
# Reads all the documents in a directory tree and generates an index operation for each
ldocs = []
for f in lfiles:
    ftxt = codecs.open(f, "r", encoding='iso-8859-1')

    text = ''
    for line in ftxt:
        text += line
    # Insert operation for a document with fields' path' and 'text'
    ldocs.append({'_op_type': 'index', '_index': index, 'path': f, 'text': text})

# Working with ElasticSearch
client = Elasticsearch()
try:
    # Drop index if it exists
    ind = Index(index, using=client)
    ind.delete()
except NotFoundError:
    pass
# then create it
ind.settings(number_of_shards=1)
ind.create()

# Bulk execution of elasticsearch operations (faster than executing all one by one)
print('Indexing ...')
bulk(client, ldocs)
```

Indexing V



```
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python3 IndexFiles.py --index news --path ./20_newsgroups/
Indexing 20089 files
Reading files ...
Indexing ...
igomez@PelussoLaptop:~/CAIM/Sesiones/1$
```

Querying I



2.2 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.

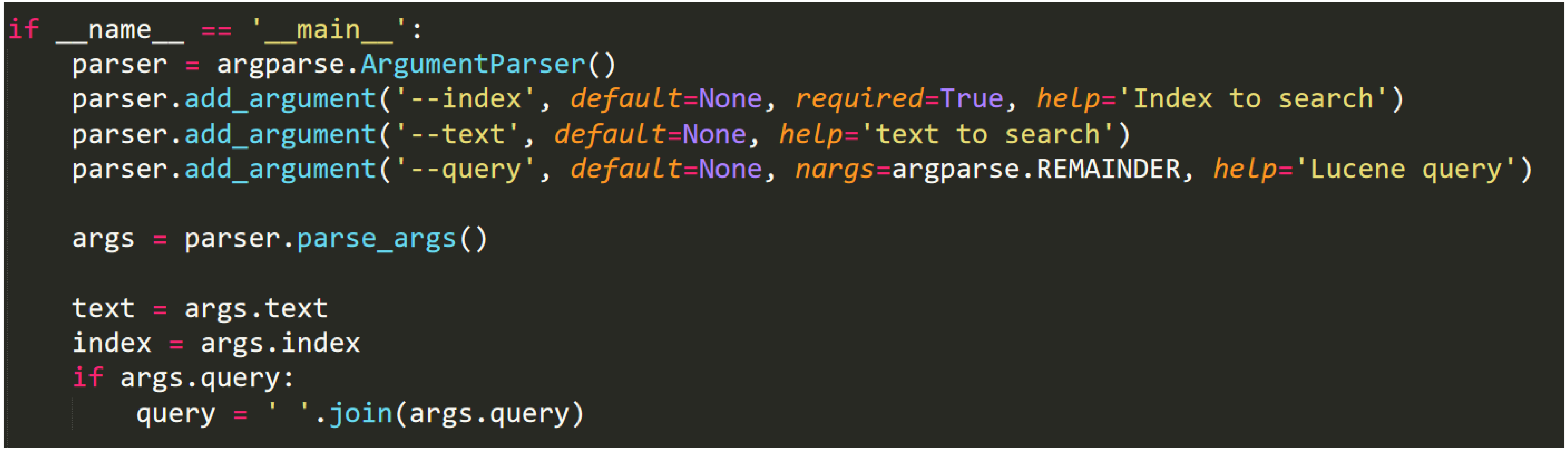
Now you can play a little bit with the script, for example execute:

```
$ 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.

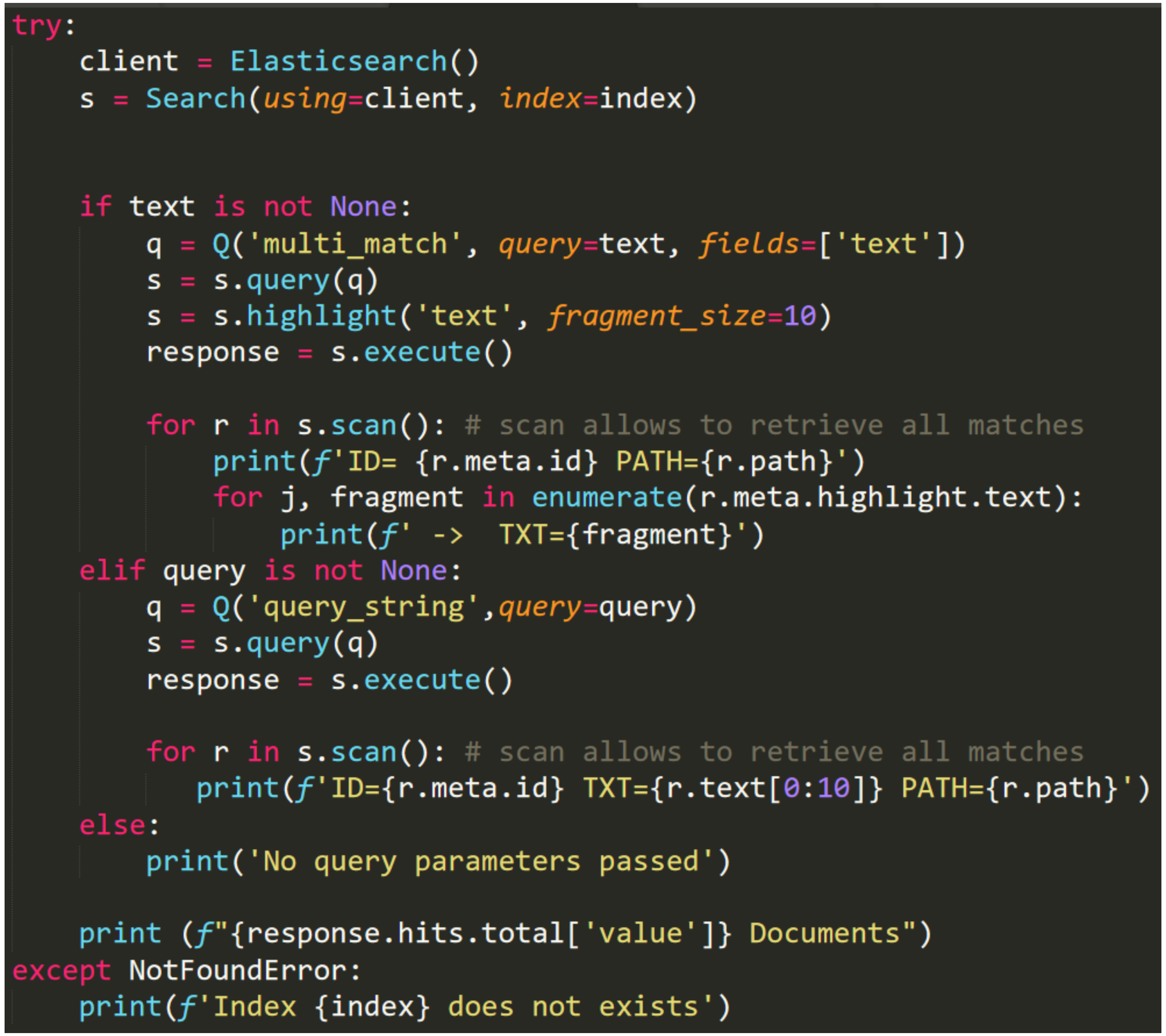
Angel OK
Engel KO

Querying II



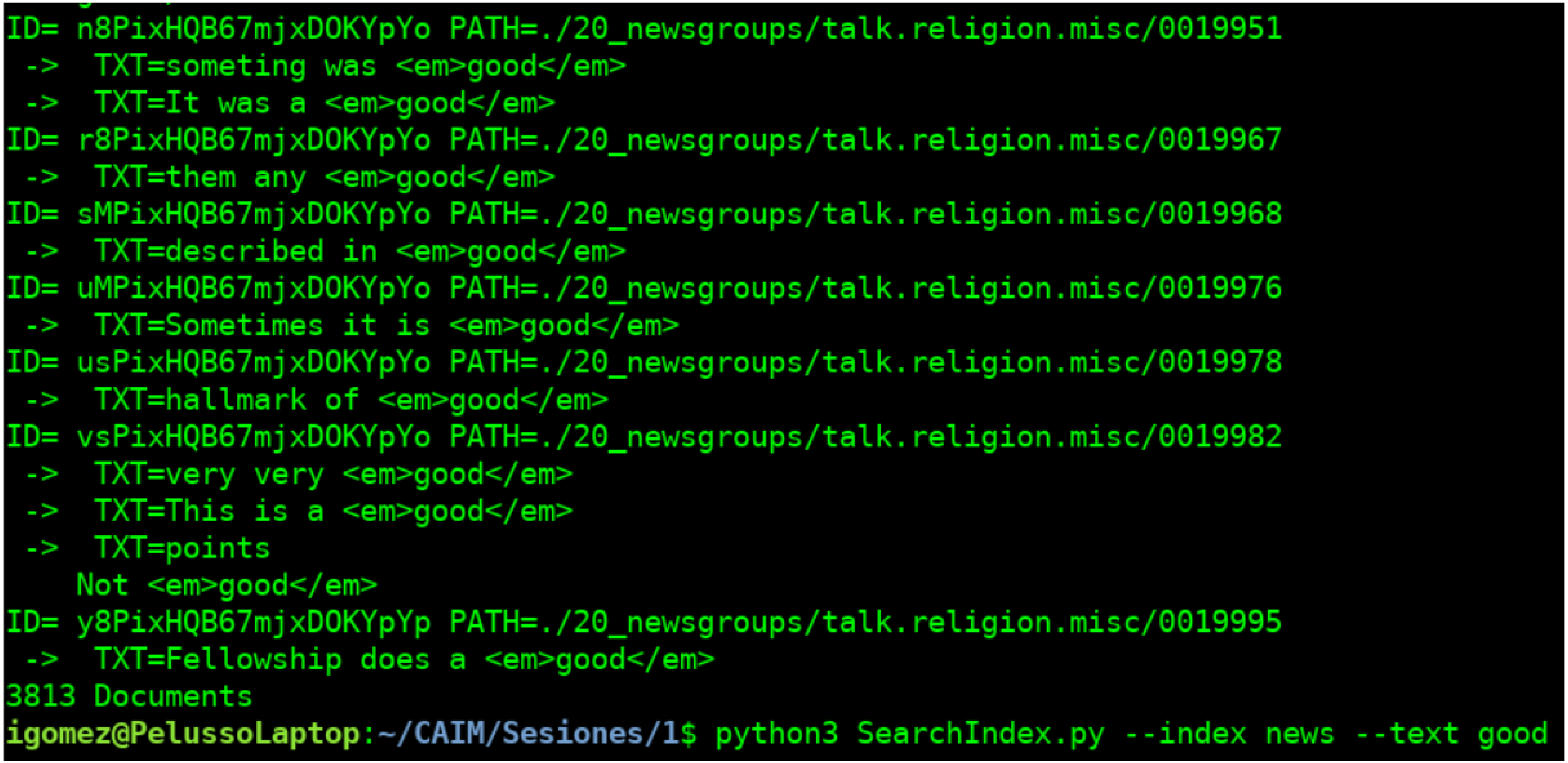
```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--index', default=None, required=True, help='Index to search')
    parser.add_argument('--text', default=None, help='text to search')
    parser.add_argument('--query', default=None, nargs=argparse.REMAINDER, help='Lucene query')

    args = parser.parse_args()

    text = args.text
    index = args.index
    if args.query:
        query = ' '.join(args.query)
```

Querying III



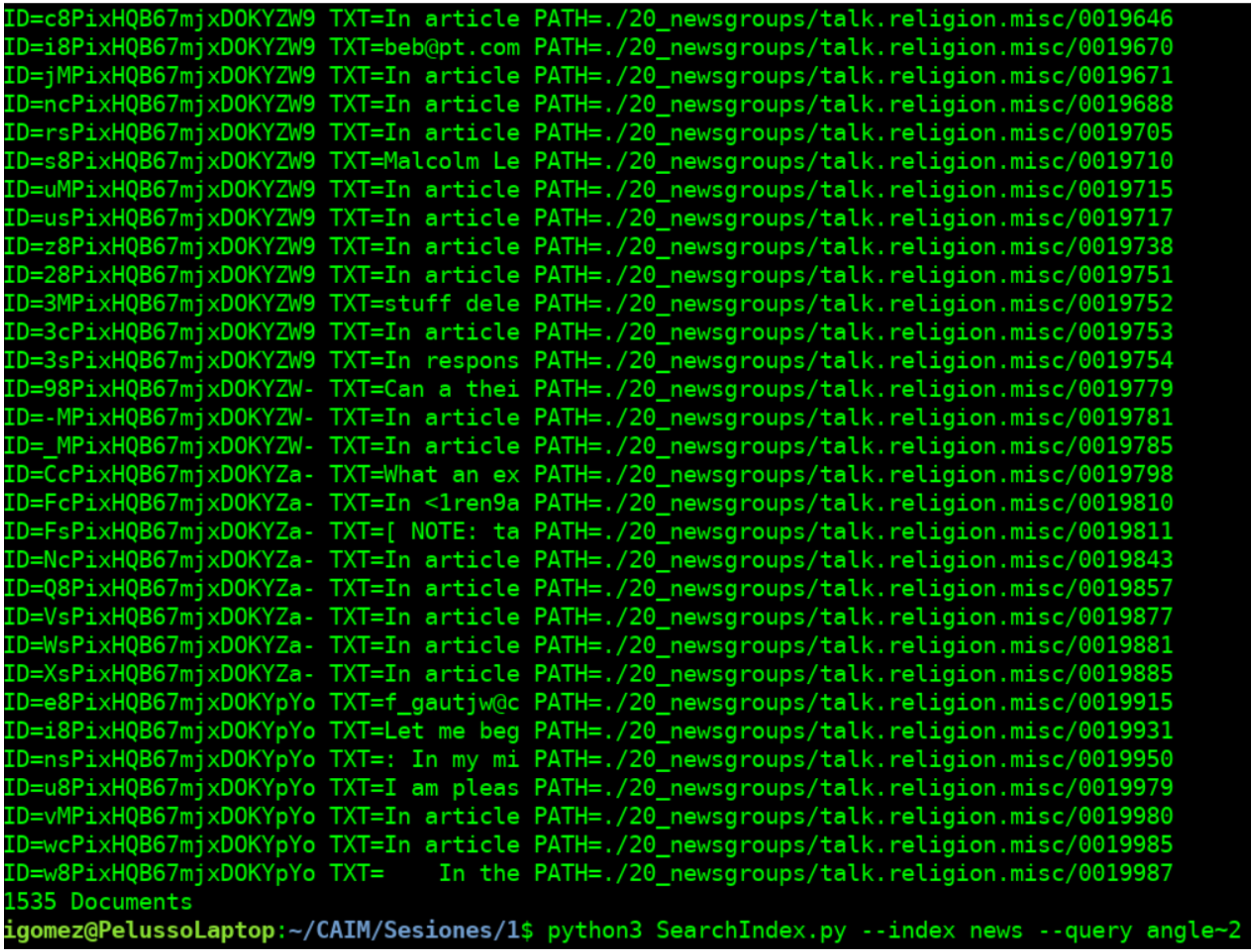
```python
try:
    client = Elasticsearch()
    s = Search(using=client, index=index)


    if text is not None:
        q = Q('multi_match', query=text, fields=['text'])
        s = s.query(q)
        s = s.highlight('text', fragment_size=10)
        response = s.execute()

        for r in s.scan(): # scan allows to retrieve all matches
            print(f'ID= {r.meta.id} PATH={r.path}')
            for j, fragment in enumerate(r.meta.highlight.text):
                print(f' ->  TXT={fragment}')
    elif query is not None:
        q = Q('query_string',query=query)
        s = s.query(q)
        response = s.execute()

        for r in s.scan(): # scan allows to retrieve all matches
           print(f'ID={r.meta.id} TXT={r.text[0:10]} PATH={r.path}')
    else:
        print('No query parameters passed')

    print (f"{response.hits.total['value']} Documents")
except NotFoundError:
    print(f'Index {index} does not exists')
```

Querying IV



```
ID= n8PixHQB67mjxDOKYpYo PATH=./20_newsgroups/talk.religion.misc/0019951
 ->  TXT=someting was <em>good</em>
 ->  TXT=It was a <em>good</em>
ID= r8PixHQB67mjxDOKYpYo PATH=./20_newsgroups/talk.religion.misc/0019967
 ->  TXT=them any <em>good</em>
ID= sMPixHQB67mjxDOKYpYo PATH=./20_newsgroups/talk.religion.misc/0019968
 ->  TXT=described in <em>good</em>
ID= uMPixHQB67mjxDOKYpYo PATH=./20_newsgroups/talk.religion.misc/0019976
 ->  TXT=Sometimes it is <em>good</em>
ID= usPixHQB67mjxDOKYpYo PATH=./20_newsgroups/talk.religion.misc/0019978
 ->  TXT=hallmark of <em>good</em>
ID= vsPixHQB67mjxDOKYpYo PATH=./20_newsgroups/talk.religion.misc/0019982
 ->  TXT=very very <em>good</em>
 ->  TXT=This is a <em>good</em>
 ->  TXT=points
    Not <em>good</em>
ID= y8PixHQB67mjxDOKYpYp PATH=./20_newsgroups/talk.religion.misc/0019995
 ->  TXT=Fellowship does a <em>good</em>
3813 Documents
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python3 SearchIndex.py --index news --text good
```

Querying V



```
ID=c8PixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019646
ID=i8PixHQB67mjxDOKYZW9 TXT=beb@pt.com PATH=./20_newsgroups/talk.religion.misc/0019670
ID=jMPixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019671
ID=ncPixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019688
ID=rsPixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019705
ID=s8PixHQB67mjxDOKYZW9 TXT=Malcolm Le PATH=./20_newsgroups/talk.religion.misc/0019710
ID=uMPixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019715
ID=usPixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019717
ID=z8PixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019738
ID=28PixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019751
ID=3MPixHQB67mjxDOKYZW9 TXT=stuff dele PATH=./20_newsgroups/talk.religion.misc/0019752
ID=3cPixHQB67mjxDOKYZW9 TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019753
ID=3sPixHQB67mjxDOKYZW9 TXT=In respons PATH=./20_newsgroups/talk.religion.misc/0019754
ID=98PixHQB67mjxDOKYZW- TXT=Can a thei PATH=./20_newsgroups/talk.religion.misc/0019779
ID=-MPixHQB67mjxDOKYZW- TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019781
ID=_MPixHQB67mjxDOKYZW- TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019785
ID=CcPixHQB67mjxDOKYZa- TXT=What an ex PATH=./20_newsgroups/talk.religion.misc/0019798
ID=FcPixHQB67mjxDOKYZa- TXT=In <1ren9a PATH=./20_newsgroups/talk.religion.misc/0019810
ID=FsPixHQB67mjxDOKYZa- TXT=[ NOTE: ta PATH=./20_newsgroups/talk.religion.misc/0019811
ID=NcPixHQB67mjxDOKYZa- TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019843
ID=Q8PixHQB67mjxDOKYZa- TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019857
ID=VsPixHQB67mjxDOKYZa- TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019877
ID=WsPixHQB67mjxDOKYZa- TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019881
ID=XsPixHQB67mjxDOKYZa- TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019885
ID=e8PixHQB67mjxDOKYpYo TXT=f_gautjw@c PATH=./20_newsgroups/talk.religion.misc/0019915
ID=i8PixHQB67mjxDOKYpYo TXT=Let me beg PATH=./20_newsgroups/talk.religion.misc/0019931
ID=nsPixHQB67mjxDOKYpYo TXT=: In my mi PATH=./20_newsgroups/talk.religion.misc/0019950
ID=u8PixHQB67mjxDOKYpYo TXT=I am pleas PATH=./20_newsgroups/talk.religion.misc/0019979
ID=vMPixHQB67mjxDOKYpYo TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019980
ID=wcPixHQB67mjxDOKYpYo TXT=In article PATH=./20_newsgroups/talk.religion.misc/0019985
ID=w8PixHQB67mjxDOKYpYo TXT=    In the PATH=./20_newsgroups/talk.religion.misc/0019987
1535 Documents
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python3 SearchIndex.py --index news --query angle~2
```

Zipf's and Heaps' laws I

Now lets have a look at the word distribution in these texts, and in particular whether Zipf's and Heaps' laws hold. The novels corpus is much cleaner than the others and probably will get better results, but it is interesting to see how the other two corpora behave.

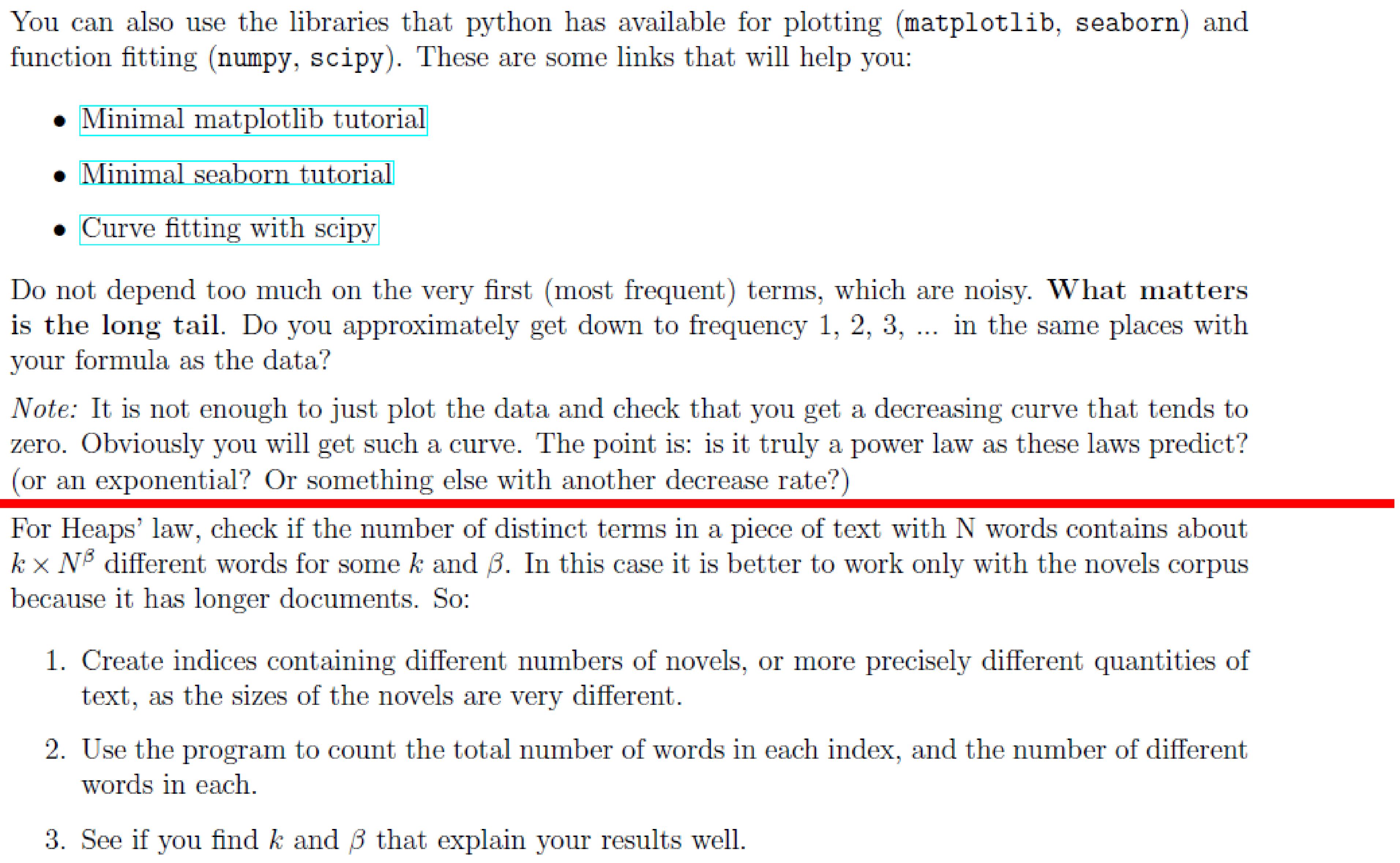
The `CountWords.py` script in the pack reads an index and writes on the standard output all the terms it contains, with their counts. Execute it redirecting the output to a file. You have two flags, the obvious `--index` and an `--alpha` flag that returns the list alphabetically instead of ascending by the number of occurrences of the word. Beware of that the last line is the number of words.

Inspect the file, sort the words lexicographically and remove stupid terms such as numbers, url's, binary or unreadable stuff, dates, etc. Leave only *proper* words. Now we are ready to analyze the data.

For Zipf's law, check if the rank-frequency distribution seems to follow a power law; what are the power law parameters ($f = \frac{c}{(rank+b)^a}$) that seem to describe best what you see?

You can use a spreadsheet for this, trying different triples (a,b,c) to imitate the number of occurrences of the word as a function of the rank. Plotting a graph (either in linear scale or in log-log scale) is also an option. The fitting will involve some trial and error.

Zipf's and Heaps' laws II



You can also use the libraries that python has available for plotting (`matplotlib`, `seaborn`) and function fitting (`numpy`, `scipy`). These are some links that will help you:

- Minimal matplotlib tutorial
- Minimal seaborn tutorial
- Curve fitting with scipy

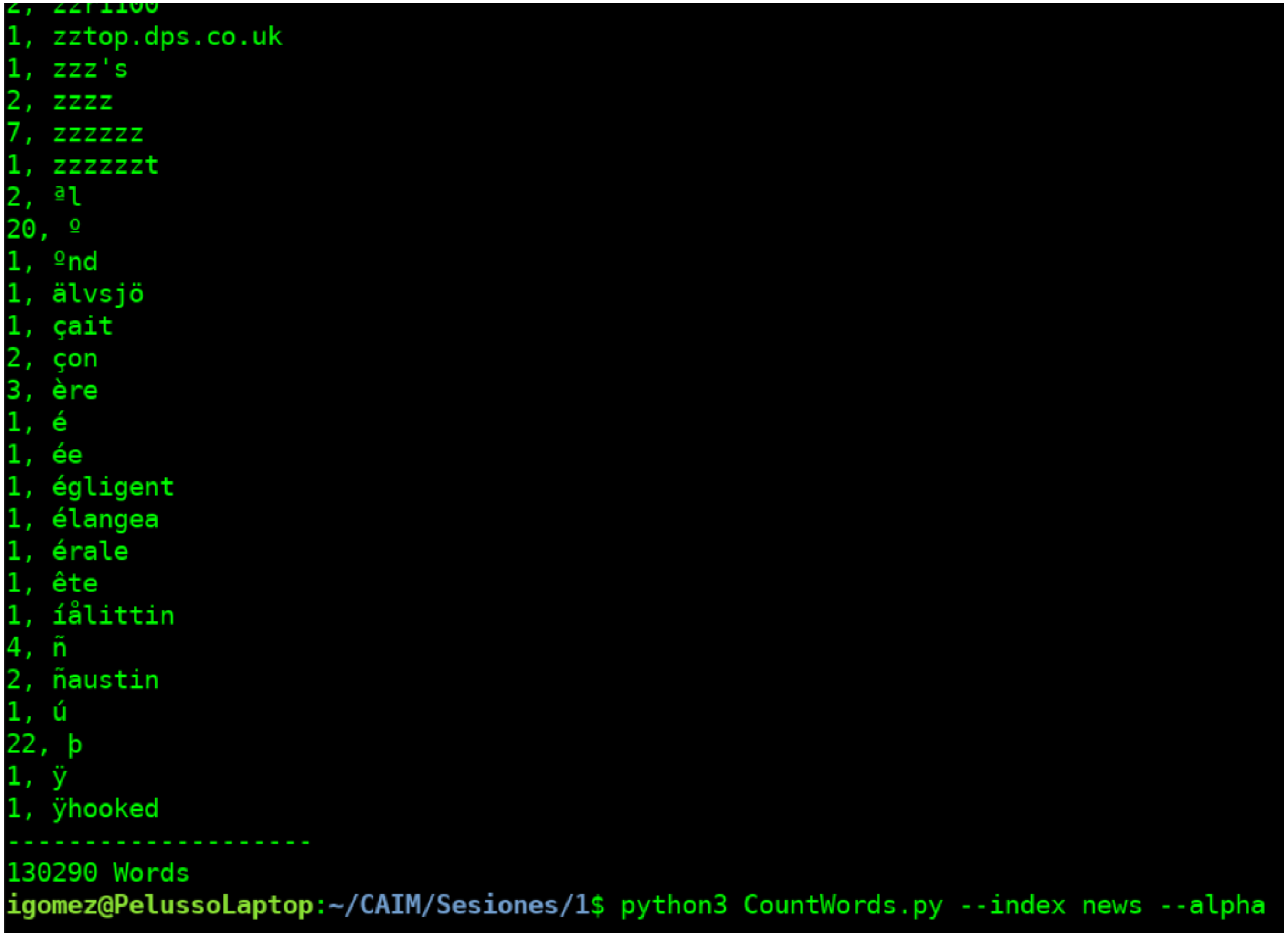
Do not depend too much on the very first (most frequent) terms, which are noisy. **What matters is the long tail.** Do you approximately get down to frequency 1, 2, 3, ... in the same places with your formula as the data?

Note: It is not enough to just plot the data and check that you get a decreasing curve that tends to zero. Obviously you will get such a curve. The point is: is it truly a power law as these laws predict? (or an exponential? Or something else with another decrease rate?)

For Heaps' law, check if the number of distinct terms in a piece of text with N words contains about $k \times N^{\beta}$ different words for some k and β. In this case it is better to work only with the novels corpus because it has longer documents. So:

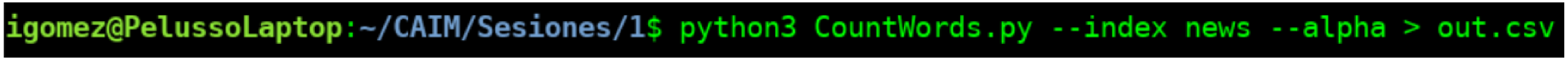
1. Create indices containing different numbers of novels, or more precisely different quantities of text, as the sizes of the novels are very different.
2. Use the program to count the total number of words in each index, and the number of different words in each.
3. See if you find k and β that explain your results well.

Zipf's and Heaps' laws III



```
2, zzr1100
1, zztop.dps.co.uk
1, zzz's
2, zzzz
7, zzzzzz
1, zzzzzzt
2, ªl
20, º
1, ºnd
1, älvsjö
1, çait
2, çon
3, ère
1, é
1, ée
1, égligent
1, élangea
1, érale
1, ête
1, íålittin
4, ñ
2, ñaustin
1, ú
22, þ
1, ÿ
1, ÿhooked
--------------------
130290 Words
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python3 CountWords.py --index news --alpha
```

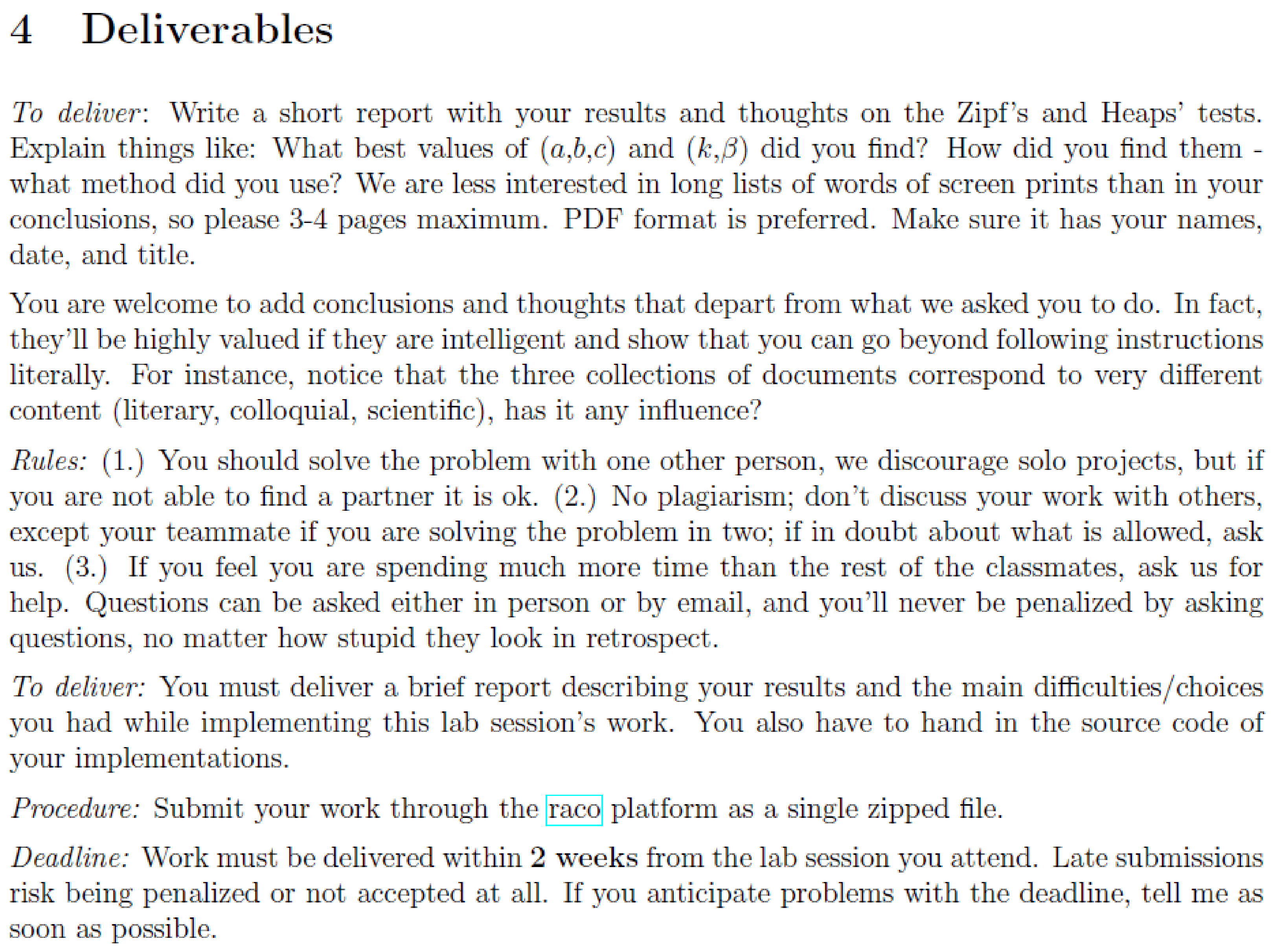
Zipf's and Heaps' laws IV



```
igomez@PelussoLaptop:~/CAIM/Sesiones/1$ python3 CountWords.py --index news --alpha > out.csv
```

1	mango.ucs.indiana.edu
53	mangoe
1	mangum
45	manhattan
1	manhole
3	manhood
3	manhunt
3	mani
11	mania
4	maniac
4	maniacal
4	maniacs
1	manias
1	manibus
3	manic

Deliverables and rules



4 Deliverables

To deliver: Write a short report with your results and thoughts on the Zipf's and Heaps' tests. Explain things like: What best values of (a,b,c) and (k,β) did you find? How did you find them - what method did you use? We are less interested in long lists of words of screen prints than in your conclusions, so please 3-4 pages maximum. PDF format is preferred. Make sure it has your names, date, and title.

You are welcome to add conclusions and thoughts that depart from what we asked you to do. In fact, they'll be highly valued if they are intelligent and show that you can go beyond following instructions literally. For instance, notice that the three collections of documents correspond to very different content (literary, colloquial, scientific), has it any influence?

Rules: (1.) You should solve the problem with one other person, we discourage solo projects, but if you are not able to find a partner it is ok. (2.) No plagiarism; don't discuss your work with others, except your teammate if you are solving the problem in two; if in doubt about what is allowed, ask us. (3.) 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.

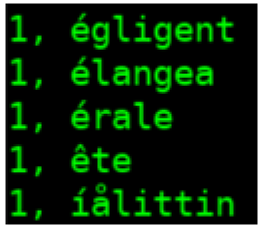
To deliver: You must deliver a brief report describing your results and the main difficulties/choices you had while implementing this lab session's work. You also have to hand in the source code of your implementations.

Procedure: Submit your work through the raco platform as a single zipped file.

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

Importante

- Preparad vuestros datos, limpiad las palabras basura y **explicad como lo habéis hecho**.

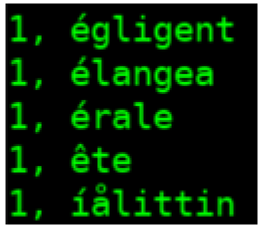


```
1, égligent
1, élangea
1, érale
1, ête
1, íålittin
```

- Ajustad la función zipf a los tres conjuntos de documentos (tres índices, tres!!)
  - Veis diferencias entre los parámetros de ajuste?
- Para heap’s law trabajad con novelas
  - Cuantas palabras distintas tengo con N = 40.000, 80.000, 120.000 etc
- Nos interesan más vuestras conclusiones que logs interminables
  - Resumenes de datos en tablas y gráficas
  - Conclusiones y procesos que os pedimos que expliquéis
  - 3-4 páginas (no es estricto)

Importante

- Preparad vuestros datos, limpiad las palabras basura y **explicad como lo habéis hecho**.
  - Más frecuentes? StopWords?
  - Menos frecuentes? Son palabras?



```
1, égligent
1, élangea
1, érale
1, ête
1, íålittin
```

aether also spelled **æther**,

- Ajustad la función zipf a los tres conjuntos de documentos (tres índices, tres!!)
  - Veis diferencias entre los parámetros de ajuste?
- Para heap's law trabajad con novelas
  - Cuantas palabras distintas tengo con N = 40.000, 80.000, 120.000 etc
- Nos interesan más vuestras conclusiones que logs interminables
  - Resumenes de datos en tablas y gráficas
  - Conclusiones y procesos que os pedimos que expliquéis
  - 3-4 páginas (no es estricto)