**CE306/CE706 (Spring 2022)**

**Lab script 2**

This lab aims at getting you familiar with tools that can be employed in the processing pipeline of an Information Retrieval (IR) application. In addition to that, it should help you get started with the first assignment.

The first part will be about using Elasticsearch’s internal text analysis and mapping features which can be used to explore a range of its build-in text analyzers to handle basic processing tasks such as markup-removal, tokenization, stemming. The remaining part will introduce you to a few state-of-the-art natural language processing open-source tools. Remember that one of the beauties of IR is the fact that there are many ways to solve your problems and as a result we would not be surprised to get many different solutions to the assignment.

In the first lab some of the students choose to use the Windows and the others use the Linux, but both should all installed the Elasticsearch and Kibana (if they are not installed, please follow the worksheet of the first lab to install them). In this lab I will assume you’ve already had both software installed and the code in this worksheet will be used in the dev-tools in Kibana. So, both Windows and Linux can use the same code.

**Mapping**

Before we get start with the actual analyzers, let’s talk briefly about the Mapping features, which can be later used to specify the analyzers for the different fields of the document. One of the common problems from the first lab is when you apply sorting or filtering sometimes you need to add “.keyword” suffixes. You might wonder why is this needed? This specific setting is because of the Elasticsearch’s default mapping. To access the mappings of your current index simply type:

GET /bank/\_mapping

in the Kibana’s Dev Tools and click the small triangle to run the code. You will get something looks like this:

Text

Description automatically generated with medium confidence

Take “address” field as an example, you will see the type of “address” filed is “text”, this means it will be processed by the text analyzer defined by the user or the default one if the analyzer is not specified. In addition to the “type” you will find there is a [multi-fields](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/mapping-types.html" \l "_multi_fields) feature enabled through “fields”. Here the first “keyword” is the name of the additional field and the other one after the “type” is the type of this field. When you define your own multi-fields you might want to use a different name (e.g. “raw”) to avoid confusing:

"fields": {

"raw": {

"type":"keyword"

}

}

The “ignore\_above” is a threshold used to avoid extremely long keyword, anything longer than the given value (in term of number of characters) will be ignored.

The “keyword” type can be used for sorting and filtering as it will preserve the original form of the field (no processing pipeline will be applied to “keyword” type), so that it can be used for those tasks. On the other hand, the “address” field (text type) will be processed by the specified pipeline. Suppose we have a document whose “address”: “22 National Drive, Colchester” for “address” field you will get four index terms analyzed by the default text analyzer: “22”, “national”, “drive”, “colchester”, where as for “address.keyword” it will create one index term: “22 National Drive, Colchester”.

This keyword feature can be used conveniently also for introduce your own index terms, e.g. processed by your python based pipeline. Please note the elasticsearch also support the array as well, you can put an array of items with the same type to a field, e.g. for “address” field you could also put [“22”, “National Drive”, “Colchester”], you will get the same index terms for “address”, but will get multiple index terms for “address.keyword”, this can be helpful if you want to find all people live in “National Drive” for instance.

To configure your own mapping rules, you could use “PUT bank” along with the “mappings” configurations, e.g. the following commend disables keywords for all the “address” field (which makes is non-sortable/filterable):

PUT bank

{

"mappings":{

"account":{

"properties":{

"address" : {"type" : "text"}

}

}

}

}

Please note the mappings can only be changed before adding any document, once the document is added you no longer be able to change the mappings, you will need to delete the old index, define the new mapping and import all the documents again.

The mapping also the place to specify the analyzers (processing pipeline), you could choose different analyzers for different fields, we will talk about this in the next section.

**Processing pipeline**

The processing pipeline in elasticsearch is called text analysis and is done by employ the analyzers. By default it uses the “standard” analyzer that apply a language independent rule based tokenizer and lower case filter which is able to perform basic processing for most languages. The analyzers are made of three components: zero or more [character filters](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/analysis-charfilters.html), exactly **one** [tokenizer](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/analysis-tokenizers.html) and zero or more [token filters](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/analysis-tokenfilters.html). The analyzers will excite all the specified character filters before feed the output into the tokenizer to get the tokens, the tokens then put through the token filters to get your final index terms. The character filters are pre-processing steps for the tokenizer, which can be used to add, remove, or change characters in the original text, e.g.: mark-up removal. The tokenizer then breaks up the processed characters into tokens, this could be as simple as just split the tokens by whitespace (whitespace tokenizer) or do noting (keyword tokenizer). The last step of an analyzer is to use the token filters to modify (e.g.: case-folding), delete (e.g.: stopwords) or add (e.g.: synonyms) tokens. There are several build-in analyzers you can use directly by refer to the name of the analyzer in the mappings. For example, using the command below you could apply the “[simple](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/analysis-simple-analyzer.html)” analyzer to the address field:

PUT bank

{

"mappings":{

"account":{

"properties":{

"address" :{

"type" : "text",

"analyzer":"simple"

}

}

}

}

}

To check what does the analyzer do to a specific text, you could use the following command:

GET bank/\_analyze

{

"field":"address",

"text":"22 National Drive, Colchester."

}

The build-in analyzers have some configurations you could specify (the configurations are different from analyzer to analyzer; you will need to check the individual page for reference). Take “standard” analyzer as an example, by default it has an empty (\_none\_) “stopwords” list (i.e. no stopwords will be filtered). The following commend will create a “standard” analyzer that removes common English stopwords:

PUT bank

{

"settings": {

"analysis": {

"analyzer": {

"my\_english\_analyzer": {

"type": "standard",

"stopwords": "\_english\_"

}

}

}

}

}

You might noticed that the creation of the configured analyzers are down under settings > analysis > analyzer, the very same position can be used to do more flexible customized analyzers. We will come to this very soon. Now let’s first take a look at what we just did to the standard analyzer. The following commends can be used to check out the difference between your new English analyzer and the original standard analyzer:

GET bank/\_analyze

{

"analyzer":"my\_english\_analyzer",

"text":"This is the first configured standard analyzer by our CE306/CE706 students."

}

GET bank/\_analyze

{

"analyzer":"standard",

"text":"This is the first configured standard analyzer by our CE306/CE706 students."

}

Did you notice the different made by additional stopwords configuration? Now let’s take one step further to create a custom analyzer. Remember the analyzer in elastic search can contain zero or more [character filters](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/analysis-charfilters.html) (char\_filter), one and exact one [tokenizer](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/analysis-tokenizers.html) and zero or more [token filters](https://www.elastic.co/guide/en/elasticsearch/reference/6.5/analysis-tokenfilters.html) (filter). Following is a template that can be used to define the custom analyzer:

PUT bank

{

"settings": {

"analysis": {

"analyzer": {

"my\_custom\_analyzer": {

"type": "custom",

"tokenizer": "the name of the tokenizer",

"char\_filter": [

"the name of the char filter",

"the name of the char filter"

],

"filter": [

"the name of the token filter",

"the name of the token filter"

]

}

}

}

}

}

Solution:

PUT bank

{

"settings": {

"analysis": {

"analyzer": {

"my\_custom\_analyzer": {

"type": "custom",

"tokenizer": "standard",

"char\_filter": [

"html\_strip"

],

"filter": [

"lowercase",

"stemmer"

]

}

}

}

}

}

**NLTK**

For more advanced text processing we need use extra libraries. If you use python, NLTK would be a handy tool. NLTK is a natural language processing toolkit consisting of Python modules that are installed in the labs. It includes taggers, tokenizers, parsers, visualisation tools and much more. You first start Python and then load the appropriate modules.

I suggest you run Python from the command line so that you can easily run your Python programs as part of a pipeline of processes. However, in the labs, you can also start IDLE (the Python GUI).

Start by exploring the NLTK site: <https://www.nltk.org/>

For a more detailed tutorial have a look at the book Natural Language Processing in Python by Steven Bird, Ewan Klein, and Edward Loper (O’Reilly, 2009). The complete book is available online at: <https://www.nltk.org/book/>

The book starts with some basics of programming in Python. It then covers simple word-level processing, part-of-speech-tagging and moves on to more complex NLE tasks.

I also suggest you download NLTK to your machine and play around with it, as you will be more flexible with installing additional modules/packages.

**Stanford CoreNLP**

If you use java, Stanford CoreNLP is perhaps the best open-source toolkit around for a variety of language processing tasks. This is where you start: <http://stanfordnlp.github.io/CoreNLP/>

The package is quite big and it might be better that you install this on your machine at home. However, you can have a go at it in the labs by following these steps:

mkdir lab2

cd lab2

curl -L -O <http://nlp.stanford.edu/software/stanford-corenlp-full-2016-10-31.zip>

unzip stanford-corenlp-full-2016-10-31.zip cd stanford-corenlp-full-2016-10-31

Then you can create a test file test.txt (containing a sample sentence such as *“Information Retrieval is even more fun than NLP.”*) and run a pipeline like this:

java -cp "\*" -Xmx2g edu.stanford.nlp.pipeline.StanfordCoreNLP \ --annotators tokenize,ssplit,pos,lemma,ner,parse,dcoref -file test.txt

Now play around with the individual processing tools and then use them to build your processing pipeline. Note that Java 1.8+ is required to get the current distribution to work.

**Exercises**

Suppose you want to process the Wikipedia example we discussed in the second lecture:

“<p><b>Gerard Salton</b> (8 March 1927 in <a href="/wiki/Nuremberg" title="Nuremberg">Nuremberg</a> - 28 August 1995), also known as Gerry Salton, was a Professor of <a href="/wiki/Computer\_Science" title="Computer Science" class="mw-redirect">Computer Science</a> at <a href="/wiki/Cornell\_University" title="Cornell University">Cornell University</a>. Salton was perhaps the leading computer scientist working in the field of <a href="/wiki/Information\_retrieval" title="Information retrieval">information retrieval</a> during his time. His group at Cornell developed the <a href="/wiki/SMART\_Information\_Retrieval\_System" title="SMART Information Retrieval System">SMART Information Retrieval System</a>, which he initiated when he was at Harvard.</p>”

Can you create a custom analyzer to remove the html tags, and tokenize the text then apply cased folding and stem to the text.

After the processing try the command below to see the output from your analyzer:

GET bank/\_analyze

{

"analyzer": "my\_custom\_analyzer",

"text":"<p><b>Gerard Salton</b> (8 March 1927 in <a href=\"/wiki/Nuremberg\" title=\"Nuremberg\">Nuremberg</a> - 28 August 1995), also known as Gerry Salton, was a Professor of <a href=\"/wiki/Computer\_Science\" title=\"Computer Science\" class=\"mw-redirect\">Computer Science</a> at <a href=\"/wiki/Cornell\_University\" title=\"Cornell University\">Cornell University</a>. Salton was perhaps the leading computer scientist working in the field of <a href=\"/wiki/Information\_retrieval\" title=\"Information retrieval\">information retrieval</a> during his time. His group at Cornell developed the <a href=\"/wiki/SMART\_Information\_Retrieval\_System\" title=\"SMART Information Retrieval System\">SMART Information Retrieval System</a>, which he initiated when he was at Harvard.</p>"

}

**Python Elasticsearch API (advanced topic)**

If you managed everything and reached here, well done! Now we can discuss how to use the python API to control Elasticsearch from the python code. Process only if you are already done with all above, if you find it already hard, don’t worry you don’t need to use this to pass your assignment. Focus on mapping and analyzers might be a better choice to maximise your assignment score.

To use python API first make sure you have Elasticsearch running and use the following command to install the python API:

pip install elasticsearch==6.8.2

Here we use 6.x.x version to make sure it works with the lab machine, if you have a 7.x.x version Elasticsearch installed you can simply use:

pip install elasticsearch

which will give you the latest version of the API.

Once you’ve installed the python Elasticsearch API, go to the IDE of your choice or simply use the command line. To connect to Elasticsearch simply use the following code:

from elasticsearch import Elasticsearch

es = Elasticsearch("http://localhost:9200")

to test your connection, simply use:

print(es.info())

this should print the basic settings of your installed Elasticsearch, e.g. the version number etc.

Now let’s create an index with the API, since we are learning the analyzers and mappings this week. The following code shows you how to create an index “bank” with the customized analyzers and mappings for “address” field.

body = ('{"settings":{'

'"analysis":{'

'"analyzer":{'

'"my\_english\_analyzer":{"type":"standard","stopwords":"\_english\_"}'

'}}},'

'"mappings":{'

'"account":{'

'"properties":{'

'"address":{"type":"text","analyzer":"my\_english\_analyzer"}'

'}}}}')

es.indices.create("bank", body=body)

Now let’s try to add few documents to the index you just created:

docs = ('{"index":{"\_id":"1"}}\n'

'{"account\_number":1,"balance":39225,"firstname":"Amber","lastname":"Duke","age":32,"gender":"M","address":"880 Holmes Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}\n'

'{"index":{"\_id":"6"}}\n'

'{"account\_number":6,"balance":5686,"firstname":"Hattie","lastname":"Bond","age":36,"gender":"M","address":"671 Bristol Street","employer":"Netagy","email":"hattiebond@netagy.com","city":"Dante","state":"TN"}\n'

'{"index":{"\_id":"13"}}\n'

'{"account\_number":13,"balance":32838,"firstname":"Nanette","lastname":"Bates","age":28,"gender":"F","address":"789 Madison Street","employer":"Quility","email":"nanettebates@quility.com","city":"Nogal","state":"VA"}\n'

'{"index":{"\_id":"18"}}\n'

'{"account\_number":18,"balance":4180,"firstname":"Dale","lastname":"Adams","age":33,"gender":"M","address":"467 Hutchinson Court","employer":"Boink","email":"daleadams@boink.com","city":"Orick","state":"MD"}\n'

'{"index":{"\_id":"20"}}\n'

'{"account\_number":20,"balance":16418,"firstname":"Elinor","lastname":"Ratliff","age":36,"gender":"M","address":"282 Kings Place","employer":"Scentric","email":"elinorratliff@scentric.com","city":"Ribera","state":"WA"}\n')

es.bulk(docs,index="bank", doc\_type="account")

The documents we use here are simply copied from the “accounts.json” we used in the first lab and the [bulk()](https://elasticsearch-py.readthedocs.io/en/v7.17.0/api.html" \l "elasticsearch.Elasticsearch.bulk) method allow us to execute multiple elastic search commends in batches, if you want index one document a time for instance you could use [es.index()](https://elasticsearch-py.readthedocs.io/en/v7.17.0/api.html" \l "elasticsearch.Elasticsearch.index) method .

Now we have documents indexed using customized analyzers and mappings, we can start to conduct some searches. The grammar for search is the same as the one used in our last lab, the main different is here we use a method to connect the Elasticsearch, and the queries are put in a parameter called “body”. For instance the following code will return the documents whose last name matches “Bond”:

query = '{"query": {"match":{"lastname":"Bond"}}}'

es.search(body=query,index='bank')

This should give you a basic knowledge about how to use the Elasticsearch python API, If you want to learn more details, you can find more reference on the [official page](https://elasticsearch-py.readthedocs.io/en/v7.17.0/index.html).

**Windows Python Elasticsearch API install guide**

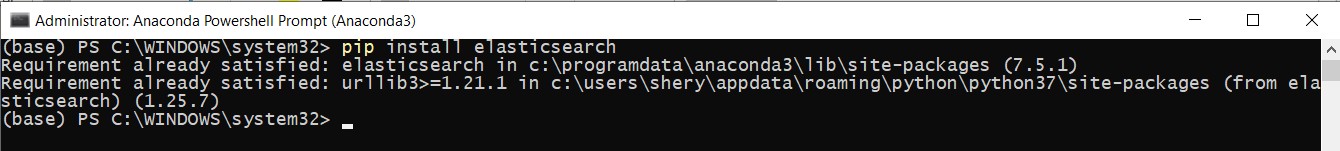
If you work on windows, I recommend using Anaconda as a tool for coding, you can either use Jupyter Notebook which is easy for coding or Spyder with more authentic coding style or other Python IDE as you desire.

\*\*Note for those using Spyder, do not upgrade to 4.1 since there’s a bug on not able to connect to kernel now\*\*

Graphical user interface

Description automatically generatedFirst, select Anaconda3 folder from your startup button then select ‘**Anaconda Powershell Prompt**’ and **Run as Administrator**

Then type: **pip install elasticsearch** on the prompt windows

****

**Now you’re ready to work on elasticsearch with python, whenever you want to work on elasticsearch, leave it on the background otherwise your python cannot connect to its port**

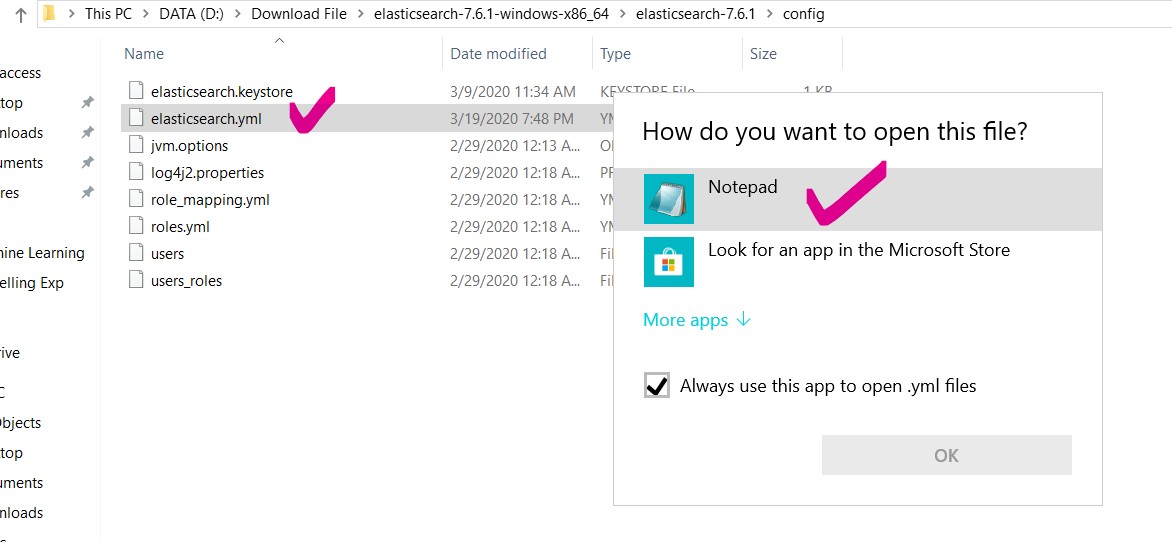
=========== Once you code with Python if you encounter with this error============

" ConnectionTimeout: ConnectionTimeout caused by

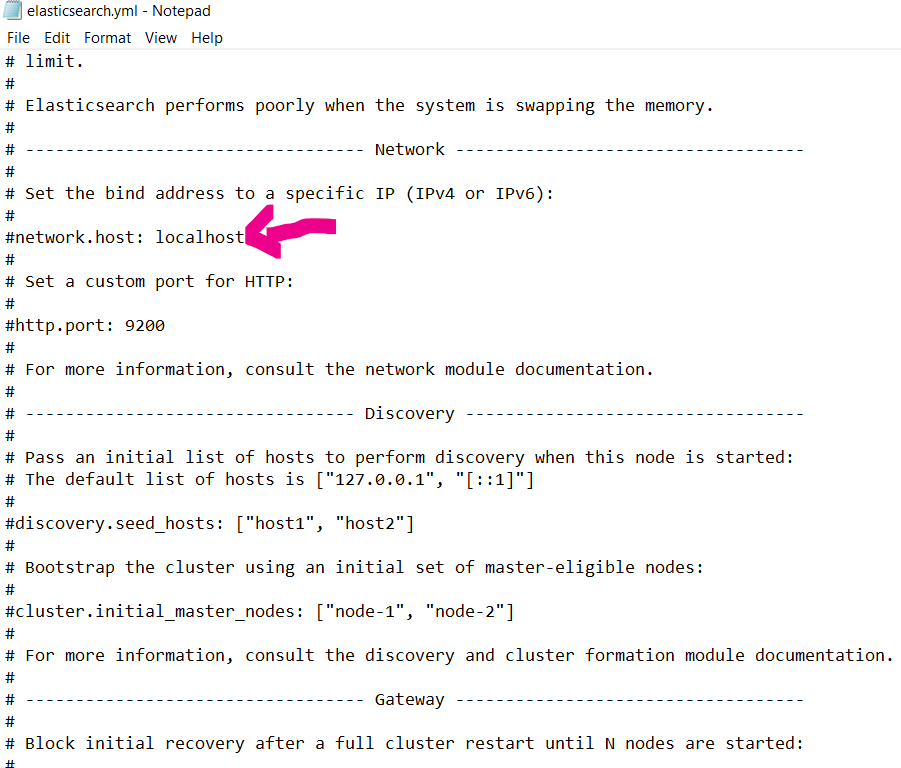
- ReadTimeoutError(HTTPConnectionPool(host='localhost', port=9200): Read timed out =30)"

Or something like this, here's the solution

You will need to go back to the elasticsearch folder on your pc again then open the config folder, select the file **elasticsearch.yml** and open it with notepad



After that look for the line "**network.host**: " in which the default before editing is **192.168.1.1**,



you will need to change it to **localhost** and **save** the file.

**Acknowledgements**

This lab script was partially prepared by Dr Shoaib Jameel.