



|  |
| --- |
| ElasticSearch  2019 |
|  |
| 15 diciembre  Information repositories – Uniovi 19-20  Written by: Óscar Sánchez Campo (UO265078)  Daniel Finca Martínez (UO264469) |

# Prologue

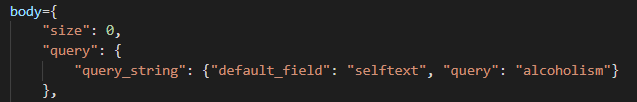
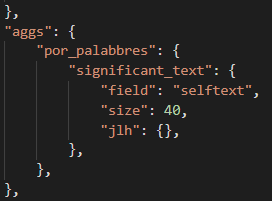
For this project, the provided script “bulkIndexer4.py” has been used to generate the index. Process such as stemming, removing stopwords and n-grams generation are used in such script.

Note: Lowercase operation has been commented out and the reason why this has been done will be explained later on (Exercise 3).

# Exercises

## Exercise 1

In this exercise “alcoholism” topic has been chosen. As an initial query to retrieve significant terms, a simple structure has been used as shown here:



To expand the results provided with such request, Elasticsearch query language aggregations feature is added to above body as stated on the image to the right. Using this, most common words used in posts retrieved from above results are grouped by frequency. These new outcomes then serve as the expanded set of terms about the topic chosen for the theme of the exercise.

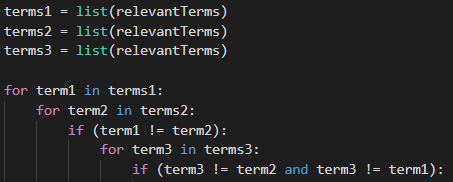
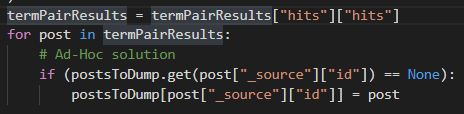
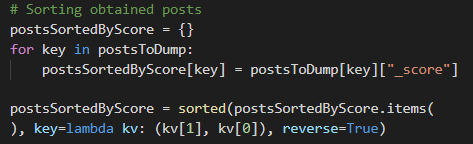
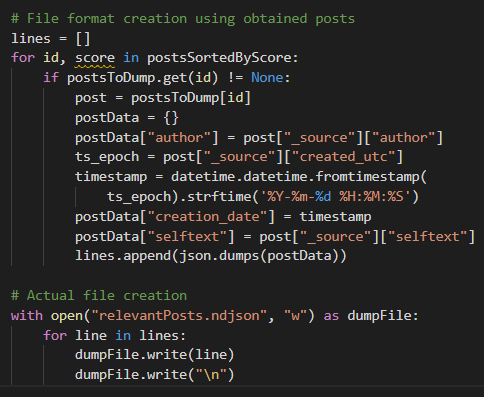
In addition to above aggregation snippet, different similarity metrics have been used. These are gnd, chi\_square and jlh. Different aggregation sizes have been tested to assess their performance. Jlh metric was used with above query. This will be explained later on.

A table can be found below with the relation of the metrics used to the number of words obtained to query the index. To our criteria, jlh has one of the highest precision rates obtained from execution experience.

|  |  |  |  |
| --- | --- | --- | --- |
| Similarity metric  to  Aggregation size | GND | JLH | Chi-square |
| 20 | 7 words – 85% | 7 words – 100% | 7 words – 100% |
| 30 | 8 words – 95% | 8 words – 100% | 9 words – 100% |
| 40 | 13 words – 100% | 14 words – 100% | 13 words – 95% |
| 50 | 16 words – 95% | 17 words – 95% | 17 words – 95% |

As mentioned above, in jlh metric has been used to obtain the results (cell marked).

From the exercise statement, it can be read that the set of posts related to the chosen topic must be obtained and extracted to a file. To do this, these steps have been followed:

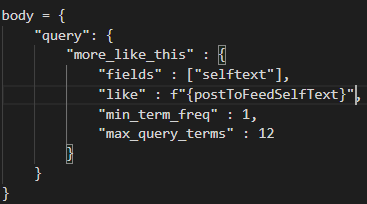
1. Significant terms are taken out from the aggregation previously shown.
2. Then, these are combined in triplets to query the index. In this way, the posts that talk about such combination of terms are retrieved. These tuples are created using three for-loops which avoid same term being queried at the same time, hence being always different.
3. As triplets are generated, they are fed to an Elasticsearch query that looks like this: 
4. As different combinations might generate repeated posts as outcome, this part of the script is used to avoid such issue. Now a dictionary with id to post relation is on our hands to process.
5. Once the previous relation is obtained, the posts are sorted based on their relevance score; using yet again a fresh new dictionary. This can be used whenever the nth most relevant posts want to be obtained.
6. Once sorted, format and output of the posts can be produced. Author, selftext and creation date have been added to the output to comply with the statement of the exercise. This output has been written to an ndJson file. Each line represents a post in json format obtained from stringifying python dictionary objects; by means of json package.

## Exercise 2

In this exercise, the requested task asks for a suggestion on how to implement “*More like this*” queries. This should pretty much simulate the behaviour of gnd similarity metric used in exercise 1. Somewhat alike to what google does when searching any term, suggesting new complete searches like the one on the input.

On top of the terms found from a query to expand relevant terms to our topic, we can feed in the text of the posts found to obtain related ones. This would be essentially the same as doing it using the terms to feed into the “like” parameter. Anyways, here is an explanation of kind of a pseudocode that might be used.

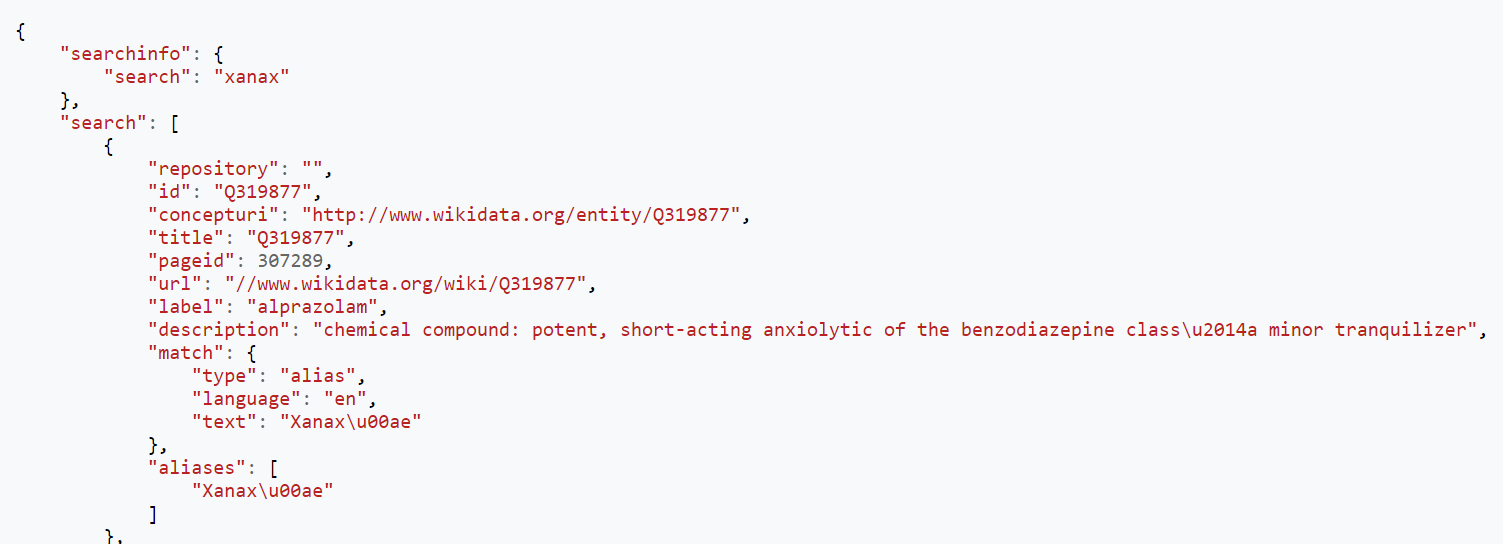
These are the series of steps followed:

1. To obtain relevant documents for the selected topic a single and normal search query is used. To limit the results, the nth most relevant ones might be extracted.
2. Once done, they can be fed into an Elasticsearch request that uses more\_like\_this dsl’s feature. Where the text of each obtained post from step 1 is introduced into the “like” parameter. The syntax was obtained from the links provided in some document from the assignment material. These queries are easy to create using python f”” formatting syntax as shown in the picture included. Where postToFeedSelfText could be a loop iterator taking out text from previously retrieved posts.

## Exercise 3

In this exercise, the problem to solve talks about obtaining a list of medications used and/or mentioned by users of reddit who appear in our post collection. In this case Wikidata has been used as a source of information to distinguish medications from other terms that might get retrieved on accident. The information is retrieved using json format from these two URLs:

* [https://www.wikidata.org/w/api.php?action=wbsearchentities&search=%SEARCH%&language=en&format=json](https://www.wikidata.org/w/api.php?action=wbsearchentities&search=%25SEARCH%25&language=en&format=json)
  + To search for the actual word retrieved by the first issued query which will be mentioned later on. This URL allows us to include a search word or phrase as a parameter.
  + It is also configured with a “format” parameter to transform the output to json format.
  + “%SEARCH%” string has been placed there to allow the script to later on the code replace it by currently iterated term to check. It returns a list of possible registered items in their databases that refer to the provided search string.
  + These items in such list contain a code that will be used later to verify its precedence using second mentioned query.

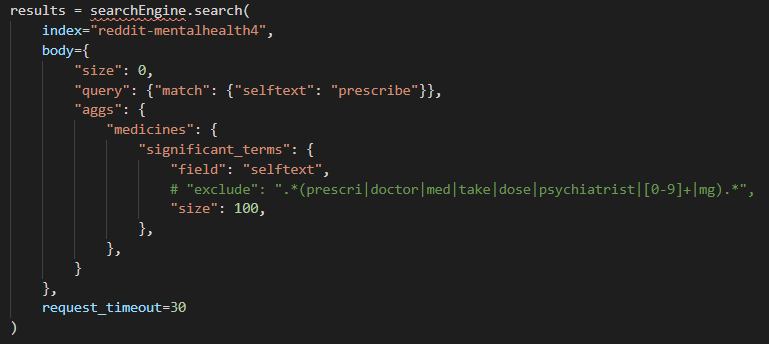


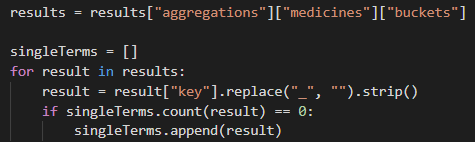
* [https://www.wikidata.org/wiki/Special:EntityData/%ITEM\_ID%.json](https://www.wikidata.org/wiki/Special:EntityData/%25ITEM_ID%25.json)
  + Given an item id (“Q[0-9]+”), this query returns the structured data for the associated item in json format. These ids are obtained from the previous URL.
  + This structure also includes the Wikidata classes from which such item is a subclass. We look for P31 property in this output file which retains the “instance of” information. Object properties information is held inside “claims” key in obtained json file. We are interested in a specific class which is “medication”. Its id is: “Q12140”
  + Same as in previous query, “%ITEM\_ID%” string is used as an anchor to replace by the id of the item to request to Wikidata.

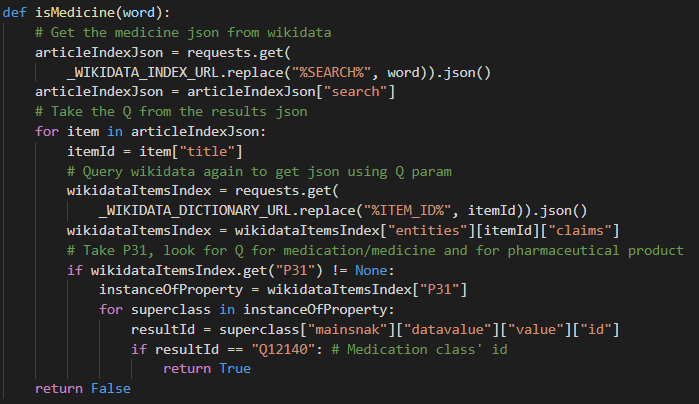
 

To solve this exercise, these steps are followed:

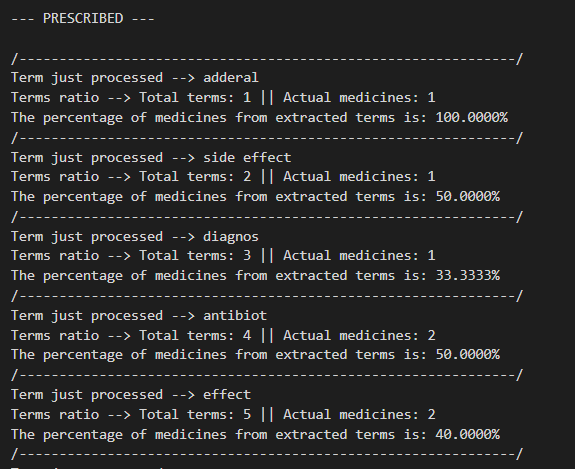
1. We start by issuing the query included below. The starting word to retrieve posts from the index is “prescribe”. We thought this would propose several posts with the context of prescribing medicine/medications and would so represent a solid starting point. We can also see that an aggregation has been included to group the most common words in such posts. These might contain actual medications but also terms that are not actual medicines. Which will be filtered down in the script; more on that later.



1. Once we obtained such terms, we need to obtain unique terms because they might be repeated inside the results from above query.
2. Filtering process can now be started. This is done through a function that checks whether or not a term is actually a medicine. This performs several operations.
   1. Firstly, requests data from Wikidata related to the word passed as a parameter. Once obtained, the ids may be processed.
   2. Secondly, al the ids returned by the request in GET mode are iterated through and serve as parameters for the second mentioned URL.
   3. Thirdly, given an id, yet another request in GET mode is issued to Wikidata so information from the object can be obtained. Once obtained, “P31” property can be looked up.
   4. And lastly, if the index has returned data referring to the instance of class “medication” inside mentioned property (id --> Q12140) in any of the search results, then the function returns True. Otherwise, when all are processed, if none contained such instance, the function returns False.



1. Finally, to follow the process, the script logs the operations performed and its results as it progresses through the terms obtained in the first place. Then this log is dumped to a file at the end of the execution for further reference. It should be noted that the success rate is logged in each iteration of the term checking.



**Disclaimer**: This screenshot has been taken from the first execution output of the script created. Where some words were excluded after experience with obtained terms. As we didn’t know whether this was allowed or not, we decided to leave it commented out, so the results were genuinely original and unmodified by previous experience. Both files, with exclusion of terms and without any exclusion have been included in the folder for this exercise’s solution; just in case they need to be checked. It must be also said that, obviously, the success rate increased a little bit when excluding undesired terms.

Hence, this means that the query used to search for terms could be refined as the program progresses by updating it with terms that don’t represent actual medicines.

But we thought that it might spoil the purpose of this exercise.