# CE306 Report

## Obtaining the dataset

I found the *Signal Media One Million News Article* dataset using the following URL link: [https://research.signal-ai.com/newsir16/signalmedia-1m.jsonl.gz](https://linkprotect.cudasvc.com/url?a=https%3a%2f%2fresearch.signal-ai.com%2fnewsir16%2fsignalmedia-1m.jsonl.gz&c=E,1,mjAPC5fZv6Q3GBnb_HxdQdQC22OkLyUTb5BVr2iTeoXxFMpzRoEBel-mE9XeHRprhS9GxaSO_gU9-vveb5lpm6-1V7QP5aIlrm60s_CRPqPVOUvmOXP-lqZFPw,,&typo=1) after following two initial links in respect:

* <https://research.signal-ai.com/newsir16/signal-dataset.html>
* <http://goo.gl/forms/5i4KldoWIX>

I proceeded to fill out the form, in which I was emailed the link to the dataset.

When downloaded, the dataset is a compressed text file in JSONL format and consists of 1 million articles (265,512 Blog articles and 734,488 News articles), each represented by a JSON object on each line. The data fields within this object represent the article’s ID, title, content, source, publication date and media-type.

## Instructions

In order to run this system, there are a number of pre-requisites:

* Elasticsearch (v7.14.0) downloaded and installed
  + Must run elasticsearch.bat as Administrator (located within bin folder)
* Kibana (v7.14.0) downloaded and installed
  + Must run kibana.bat as Administrator (located within bin folder)
* pip install json (run command within CMD python directory)
* pip install elasticsearch (run command within CMD python directory)
* pip install re (if needed)

within my script, I imported **json**, **re** (regex), as well as from elastic search: **helpers** and **elaticsearch**

A picture containing logo

Description automatically generated

## Indexing

Script:

Includes an indexing function:

Text

Description automatically generated with medium confidence Graphical user interface

Description automatically generated with low confidence

(results in – apologies for poor quality photo – each line printed within the python shell as it is indexed in elasticsearch – this was to keep track of how long it took to upload each JSON object into elasticsearch.)

A picture containing text, whiteboard

Description automatically generated

After running this function and opening the kibana server (<http://localhost:5601/>), the connection to the elasticsearch server had proven successful. The indexing function was able to upload the first ~58,000 documents to the index, as shown below:

A screenshot of a computer

Description automatically generated

(It took around 4 to 5 hours to upload a total of 58,726 documents to the index using my script.)

I then created an Index Pattern using Kibana, which allowed me to navigate through the index as shown below:

A screenshot of a computer

Description automatically generated

## Searching

Using Kibana we can now explore the data and search for terms within the documents. Below are screenshots including search terms and the results:

A screenshot of a computer

Description automatically generated with medium confidence

Here, we are searching for documents that contain “Federal Government” OR “Syrian refugees”. Here are the results:

A screenshot of a computer

Description automatically generated with medium confidence

Quite a large amount of matches, due to the OR constraint. The documents that the search term ‘hits’ can contain either “federal government” or “Syrian refugees”, or both.

Second test: A screenshot of a computer

Description automatically generated with medium confidence

Our search term:

Mickey Rooney-Judy Garland musicals in which a character inevitably proclaims AND "My dad's got a barn. Let's put on a show."

Result is below:

Graphical user interface

Description automatically generated

This was a very specific search term, especially as it contains a quote, as well as an AND constraint, hence only a single match is expected.

Final test:

Search term: “Black Lives Matter” AND protest

Results:

A screenshot of a computer

Description automatically generated with medium confidence

If the query used OR instead of AND, the number of matches would be drastically higher, however this is more specific.

The design decisions to the overall architecture can be seen within the script, which is commented explaining how the index is structured to match the data fields within the dataset. A line from the dataset is converted into a JSON object in a dictionary format {data field : value} e.g. {media-type : News}. This allows the index pattern to be created much more easily, resulting in the architecture seen above when using Kibana to search and explore the data in a very structured manner.

## Building a Test Collection

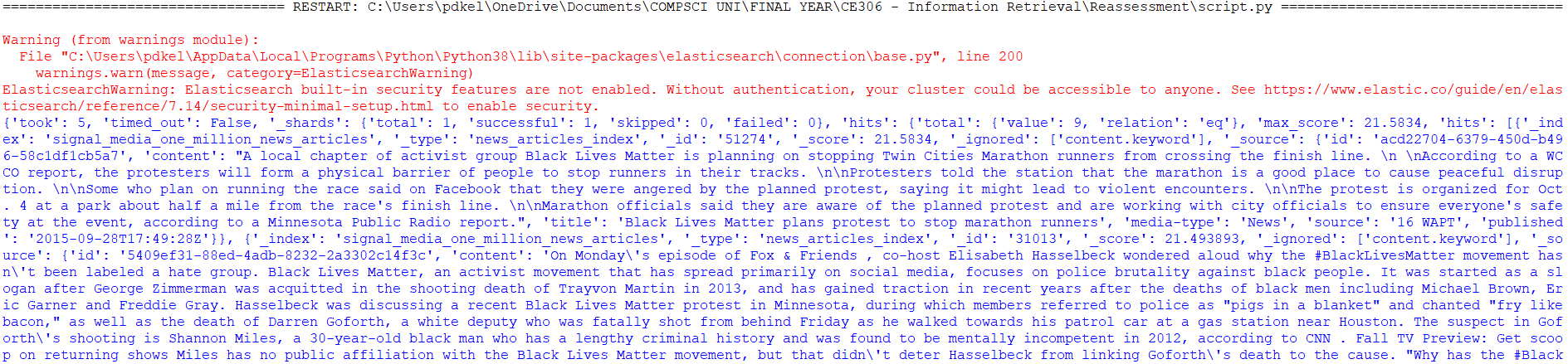
I am going to be collecting news (not blog) articles consisting of content that must include “Black Lives Matter” and “protest”. The information need is for writing about Black Lives Matter protests (could be used as a reference or for a source in an academic essay).

The query I am using to collect this data is:

A picture containing graphical user interface

Description automatically generated

This query returns 9 articles that satisfy the information need which makes up my chosen test collection. As shown by the screenshot of my python shell output below:



(continued…)

Text

Description automatically generated

*Identify three specific events covered by the collection and then compose two sample queries for each:*

### Event One

<target article>

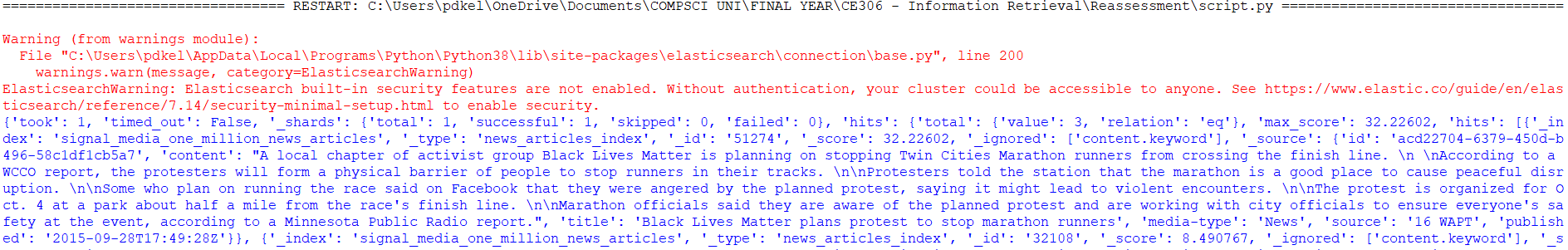
"Black Lives Matter plans protest to stop marathon runners”

***Query 1:***



This represents a typical search query: “Black Lives Matter” and “protest” and “marathon”

Result (simply output in Python Shell):



(continues)…



* Hits: 3

With no real specificity about the “marathon” event, other documents that are less relevant to this article also appear as a match. The results are sorted by score in descending order. The first article to appear (highest score) was our target article.

***Query 2:***

A picture containing text

Description automatically generated

A typical search query from a user, which is very similar to the first query, however more specific:

“Black Lives Matter” and “protest” and “Twin Cities Marathon”.

Text

Description automatically generated

* Hits: 1

With the added phrase which must match the phrase “Twin Cities Marathon”, the number of matches has decreased to 1, which is exactly the relevant document we wanted to match.

### Event Two

<target article>

“Elisabeth Hasselbeck Asks If BlackLivesMatter Should Be Classified as a Hate Group”

***Query 1:***



Typical user query: “black lives matter” and “protests” and “started” and “Minnesota”

(use regex to apply stemming to word, start: starting, started… protest: protests, protesting, protested…)

Results:

Text

Description automatically generated

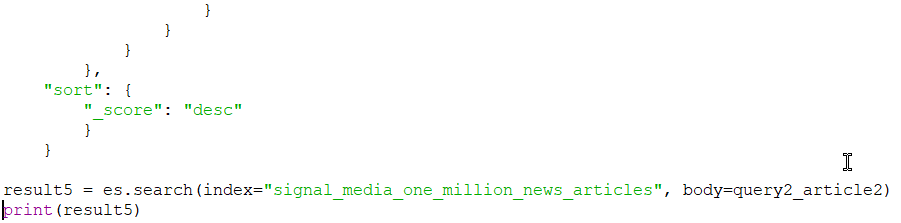
(continued…)



Results in 3 hits, however 2 articles contain identical content and title values, hence the query was effective in finding the relevant document first. The other hit is most likely due to the stemming of words: start and protest.

***Query 2:***

A picture containing text

Description automatically generated 

Typical search for a quote: “Elisabeth Hasselbeck” and “black lives matter” and “protest” and “movement” and “hate group”.

(stemming required for Elisabeth – Elisabeth/ Elizabeth)

Result:

Text

Description automatically generated

2 hits – very specific query which resulted in 2 documents – however, all data field values (apart from doc id) are identical. Hence, 1 article was found with this query.

### Event Three

<target article>

“The Latest on Gray case: Officer trials to stay in Baltimore”

***Query 1:***

Graphical user interface

Description automatically generated with medium confidence A picture containing graphical user interface

Description automatically generated

Typical query: "black lives matter" and “protest” and "Baltimore police officers" and “charged” and “death” and "Freddie Gray"

(stemming: Fred\* = Fred, Freddie, Freddy…)

Result:

A picture containing text

Description automatically generated

(continued…)

Text

Description automatically generated

2 hits – both very informative articles – the targeted article appears first, then another that is very relevant to the information need (can also be used to cover the event – goes through the full trial with timestamps) appears after.

***Query 2:***

“Baltimore Mayor Stephanie Rawlings-Blake” and “Interim Police Commissioner Kevin Davis” and “black lives matter” and “protest” and “press conference” and “trial” and “police officers”

A picture containing graphical user interface

Description automatically generated A picture containing graphical user interface

Description automatically generated

Result:

Text

Description automatically generated

(continued…)

Text

Description automatically generated

Hits 2 documents again, both of which are fully relevant to the query and satisfy the information need in intricate detail (targeted document is matched first, then the second document is matched). Both documents matched have a high score, with the second being a little less.

## Evaluation

Within my python script my evaluation function uses ranked retrieval with an evaluation metric p@5. The test collection has only 9 documents within it, and the user search queries obtain a maximum of 3 hits throughout, so I will also be using the query I used to obtain the subset of documents used for my test collection (specialised for tf-idf and boolean) in order to show the metric applied to more than 5 query results.

In order to be able to use TF-IDF and Boolean retrieval models, I used the elasticsearch function, put\_mapping. Within the “content” and “title” fields, I added the “similarity” property which defined the required retrieval models for both fields, “boolean” and “BM25”. This can be seen in my update\_settings function:

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application

Description automatically generated



(Commented out the retrieval model I did not use at the time)

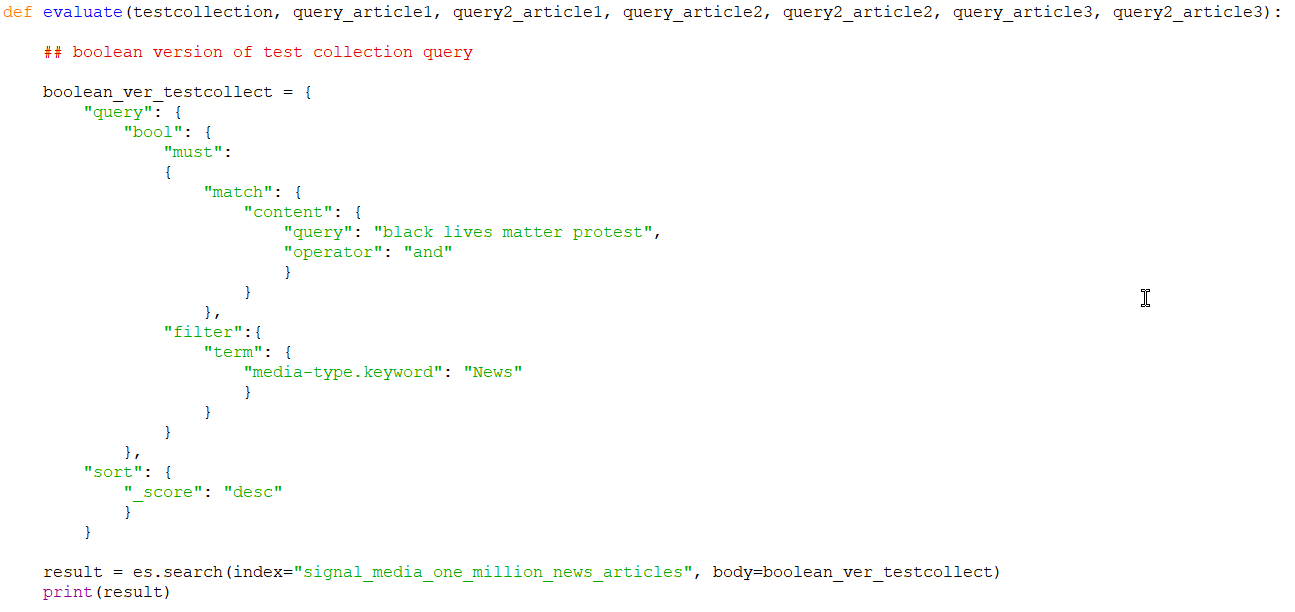
I chose to evaluate my queries using the Ranking evaluation API:

This is made up of 3 building blocks essentially:

* My test collection (a subset of documents)
* The queries I used for each even, and one that has been changed to specialise in the Boolean Similarity module, as many of my queries use full-text matches.
* A set of document ratings that represent the relevance of the documents to the information need of the user.

Below is the code showing I have satisfied the prerequisites:

(Boolean-type query included in my script’s evaluate function)

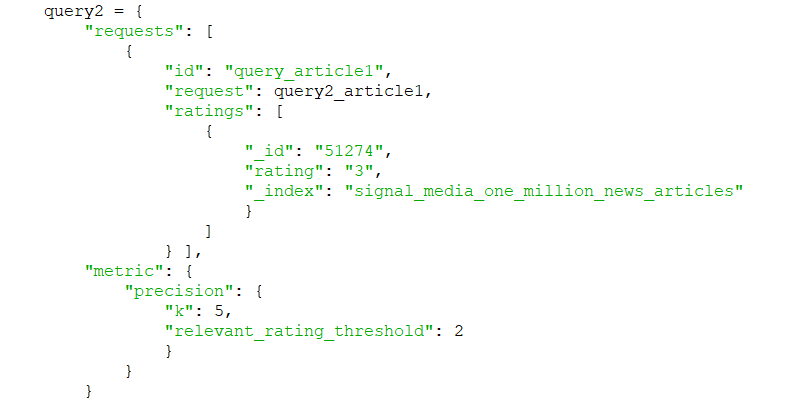
Graphical user interface, text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

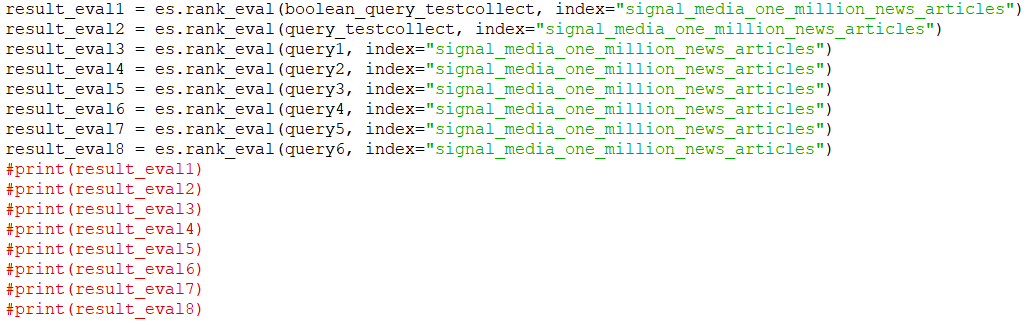
Description automatically generatedText

Description automatically generatedText

Description automatically generated

### Evaluation results

To do this I used the following code:



#### (Using Boolean Similarity)

Test Collection query (changed to include just Boolean operators):

A picture containing text

Description automatically generated

* 5/5 docs retrieved are relevant
* (Top k=5 documents retrieved are relevant, sorted by the score)

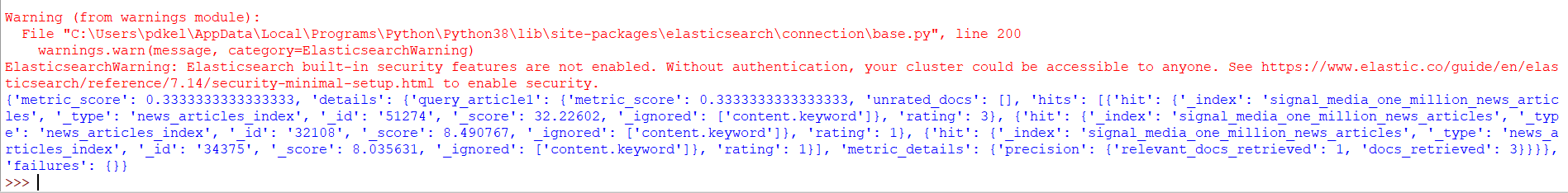
Test Collection query (original)

Text

Description automatically generated

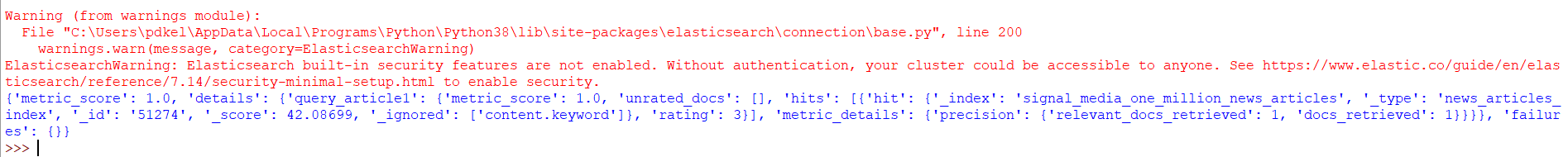
* 5/5 docs retrieved are relevant
* (Top k=5 documents retrieved are relevant, sorted by the score)

Event 1 query 1:



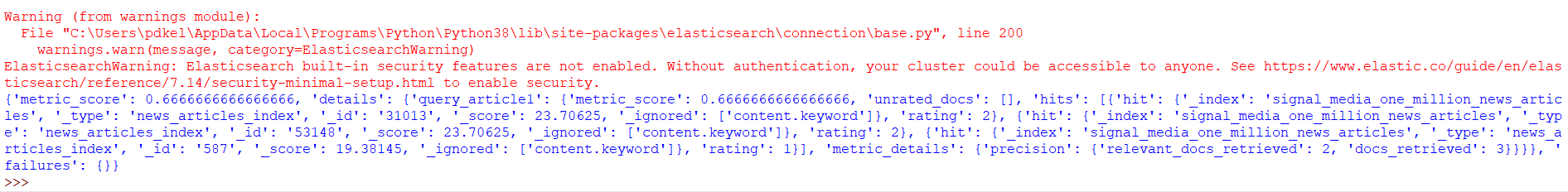
* 1/3 docs retrieved are relevant

Event 1 query 2:



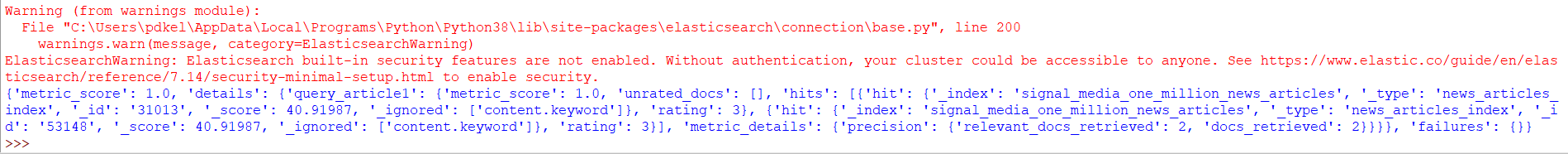
* 1/1 docs retrieved are relevant

Event 2 query 1:



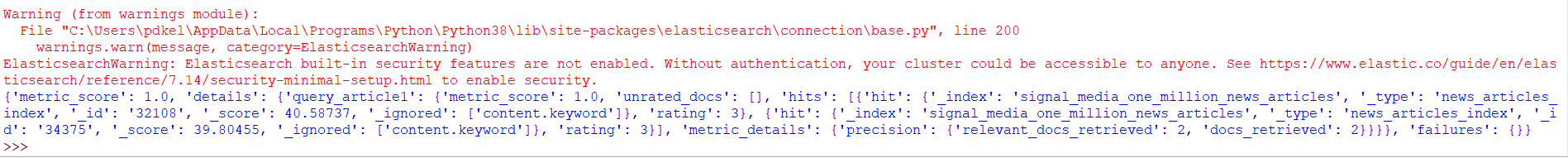
* 2/3 docs retrieved are relevant

Event 2 query 2:



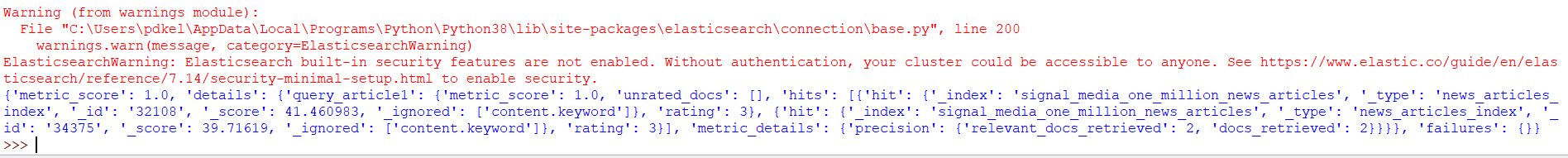
* 2/2 docs retrieved are relevant

Event 3 query 1:



* 2/2 docs retrieved are relevant

Event 3 query 2:



* 2/2 docs retrieved are relevant

#### (Using TF-IDF Similarity)

Test Collection query (changed to include just Boolean operators):

A picture containing text

Description automatically generated

* 5/5 docs retrieved are relevant
* (Top k=5 documents retrieved are relevant, sorted by the score)

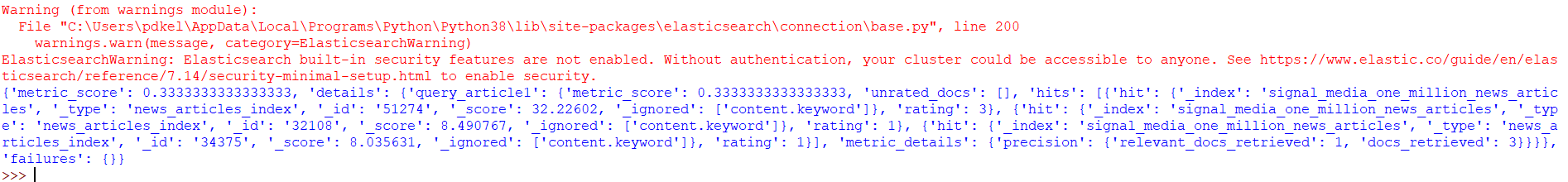
Test Collection query (original)

A picture containing text

Description automatically generated

* 5/5 docs retrieved are relevant
* (Top k=5 documents retrieved are relevant, sorted by the score)

Event 1 query 1:



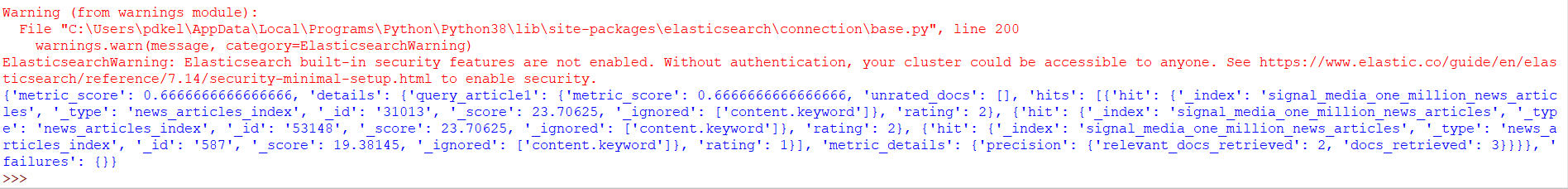
* 1/3 docs retrieved are relevant

Event 1 query 2:



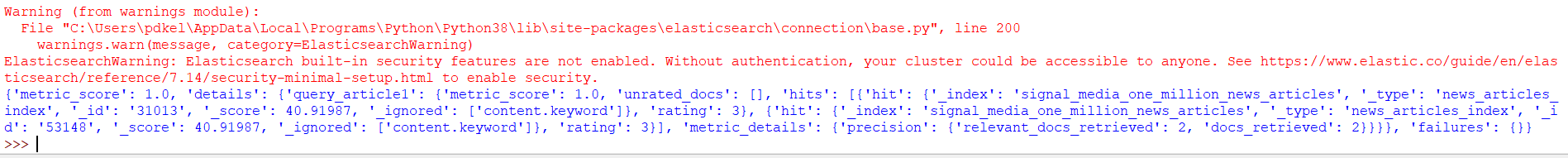
* 1/1 docs retrieved are relevant

Event 2 query 1:



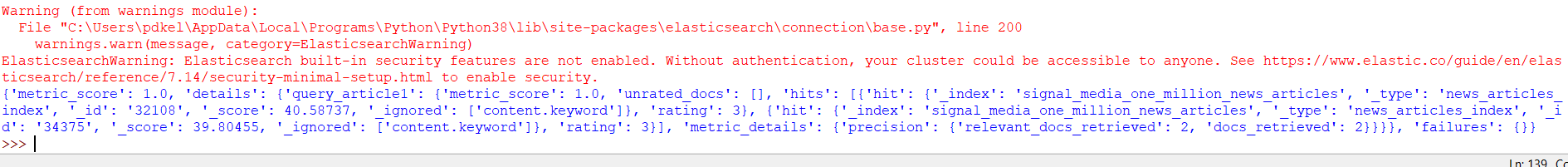
* 2/3 docs retrieved are relevant

Event 2 query 2:



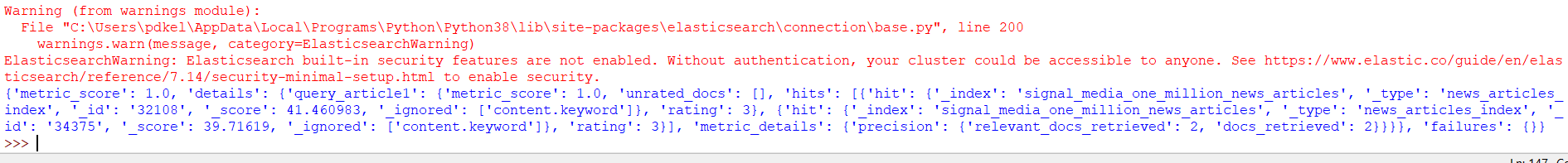
* 2/2 docs retrieved are relevant

Event 3 query 1:



* 2/2 docs retrieved are relevant

Event 3 query 2:



* 2/2 docs retrieved are relevant

As stated above, I have used the Ranked retrieval API within Elastic Search which allowed me to use the required metric P@K, where k = 5 (top 5 relevant documents). I was able to change the Similarity type on two data fields by updating the index mapping, however I could have improved by creating a dynamic index (dedicating multiple shards to the index) which would have allowed me to update the index settings – a much more efficient way of changing the Similarity type – however this was the only other successful way of implementing the Boolean and TF-IDF retrieval models for evaluating my user search queries.

From the results seen above, I can conclude that there was no difference using the stated metric to evaluate Boolean and TF-IDF retrieval models on the test queries. Therefore, to improve this, I would use a much larger test collection, and expand variety on test queries (using more Boolean operators, as opposed to full-text / phrase matches). On the test collection at hand, most queries were specific and resulted in the same number of relevant documents as documents retrieved, apart from event 1 query 1 and event 2 query 1. The test collection queries showed that the score for a document with rating 2 was higher than a document with rating 3, therefore the metric used didn’t fully prove that the documents were ordered correctly, or that in fact, the rating may have been slightly flawed. Other than this, the evaluation of both the Boolean and original test collection queries worked in ranking the most relevant documents at the top (documents with rating 2 or 3).

Links I used to help with learning topics, applying knowledge of code, and learning advanced structures of queries:

<https://www.youtube.com/watch?v=6ShIMvS7bh0>

<https://stackoverflow.com/questions/28538760/elasticsearch-bool-query-combine-must-with-or>

<https://www.elastic.co/guide/en/elasticsearch/reference/6.8/search-rank-eval.html>

<https://elasticsearch-py.readthedocs.io/en/latest/api.html>