Elasticsearch

Davide Antonino Giorgio
Politecnico di Torino
Student id: s291477
s291477@studenti.polito.it
http://giorgiodavide.it

***Abstract*—This project consists in the analysis of the powerful searching capabilities of ElasticSearch and the creation of an ElasticSearch-powered search engine for a movie's database.**

I. Project overview

The data collection proposed for the project is related to a collection of movies' information. It can be downloaded from the "Portale della Didattica" (imdb.json). The collection includes information about movies and their associated reviews on IMDB and Rotten Tomatoes. The aim is to use elasticsearch to perform searches in this dataset.

II. Interest queries

1) Get the list of the first 10 movies that have been published between 1939 and 1945 and best matches the word "war" in the plot.

```
res = es.search(index="project_3", body={
"query": {
   "bool":{
      "must":[{
         "match":{
            "plot": "war"
         }},{
         "range": {
            "year": {
               "gte": 1939,
               "lte": 1945
            }
         }}
      ]
   }
},
"size": 10
})

for doc in res['hits']['hits']:
   print (doc["_score"], doc["_source"]["title"])
```

In this query I used the following operators:

- must: used to contribute to the matching score given a query
- match: used to find in the plot the word *war*
- range: used to return a document that contain terms within a provided range. In this case we was interested to years between 1939 and 1945.
- size: used to return the set of movies and limit it to ten.

2) Get the list of the first 20 movies that have an average IMDB rating higher than 6.0 and best matches "Iron Man" in the full plot.

```
res = es.search(index="project_3", body={
"query": {
   "bool":{
      "must":[{
         "match":{
            "fullplot": "Iron Man"
         }},{
         "range": {
            "imdb_rating": {
               "gt": 6.0
            }
         }}
      ]
   }
},
"size": 20
})

for doc in res['hits']['hits']:
   print (doc["_score"], doc["_source"]["title"])
```

In this case I did not understand the text of the request. In particular I don't know what is the average of IMDB rating for each movie. Anyway, I used the *imdb rating* instead of the *average imdb rating*. The operators used are similar to the one of the previous query. In particular, here we are looking for "*Iron Man*" in the *fullplot* and then I applied a range over the *imdb_rating* field in order to return onlyt the subset of documents where this field is greather than 6.0.

3) Search for the movies that best match the text query "matrix" in the title. Boost the results by multiplying the standard score with the IMDB rating score. Repeat the same query considering the Rotten Tomatoes score instead of IMDB. Does the order of the results change?

```
res = es.search(index="project_3", body={
"query": {
   "function_score": {
      "query": {
         "match":{
            "title": "matrix",
         },
      },
      "script_score": {
         "script": {
            "source": "doc.containsKey('imdb_rating')
               ? doc['imdb_rating'].value : 1"
         }
      }
   }
}})

for doc in res['hits']['hits']:
   print (doc["_score"], doc["_source"]["title"]+',',
        'score:', doc["_source"]['imdb_rating'])
```

```
res = es.search(index="project_3", body={
"query": {
   "function_score": {
      "query": {
         "match":{
            "title": "matrix",
         },
      },
      "script_score": {
         "script": {
            "source":
                 "doc.containsKey('tomatoes_rating')
                 ? doc['tomatoes_rating'].value : 1"
         }
      }
   }
}})

for doc in res['hits']['hits']:
   print (doc["_score"], doc["_source"]["title"]+',',
        'score:', doc["_source"]['tomatoes_rating'])
```

In order to boost the resulting score using the IMDB and Rotten Tomato rating, is possible to use *script_score*. This operator can be used to boost the score in different ways. Actually the default choice is used that is the *multiply* one. Is possible to change is using the operator **"boost_mode": "multiply"**. This operator allors us to boost the score using the following strategies:

- multiply (default): query score and function score is multiplied
- replace: only function score is used, the query score is ignored
- sum: query score and function score are added
- avg: average
- max: max of query score and function score
- min: min of query score and function score

In our case the function score is *imdb_rating* in the first query and *tomatoes_rating* in the second. The result is different between the two queries. Below two snippets about the results abtained.

```
// for the first query (IMDB rating score)
89.26549 The Matrix, score: 8.7
65.56894 The Matrix Revisited, score: 7.4
63.796803 The Matrix Reloaded, score: 7.2
59.36647 The Matrix Revolutions, score: 6.7
57.594337 Armitage: Dual Matrix, score: 6.5
50.299583 Return to Source: Philosophy & 'The
     Matrix', score: 8.0
```

```
// for the first query (Rotten Tomatoes rating score)
36.937443 The Matrix, score: 3.6
31.012335 Armitage: Dual Matrix, score: 3.5
30.12627 The Matrix Revisited, score: 3.4
30.12627 The Matrix Reloaded, score: 3.4
30.12627 The Matrix Revolutions, score: 3.4
27.664772 Return to Source: Philosophy & 'The
     Matrix', score: 4.4
```

The result obtained is correct because, as we can see by the score parameter, the two different multipliers are different (IMDB and Ronnet Tomatoes).

III. Movies search engine

An application based on the flask framework was created which could query the elasticsearch server and display the results of the requested search. Searches are made according to the following specifications:

- Create a flask-based application that is capable of performing full-text search over the provided data collection. The application should include:
    1) An highlighter that shows the first match of the full-text search in bold. The highlighter could be implemented both in plot or fullplot fields, the choice is left to your preference.
    2) A score booster that is able to increase the score using IMDB ratings (or Rotten Tomatoes ratings, the choice is left to your preference).
    3) A results.html page that presents the title and the first match of the highlighter for each of the top-20 documents after the query search.

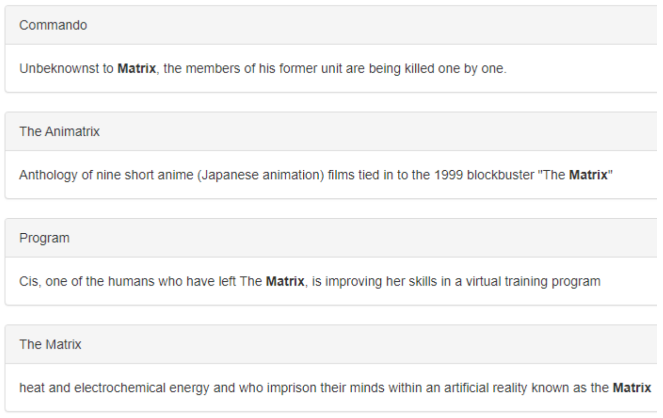
In the proposed solution I decided to:

- Apply and highlighter to the *fullplot* of the movie.
- Apply a score booster based on the IMDB rating

```
res = es.search(
      index="project_3",
      size=20,
      body={
         "query": {
            "function_score": {
               "query": {
                  "multi_match" : {
                     "query": search_term,
                     "fields": [
                        "fullplot"
                     ]
                  }
               },
               "script_score": {
                  "script": {
                     "source":
                          "doc.containsKey('imdb_rating') ?
                          doc['imdb_rating'].value : 1"
                  }
               }
            }
         },
         "highlight": {
            "fields": {
               "fullplot": {}
            },
            "pre_tags" : ["<b>"],
            "post_tags" : ["</b>"],
         }
      }
   )
   return render_template('results.html', res=res )
```

In particular, I have used the highlight operator, which makes it possible to highlight the word searched for in the text by introducing the html tags <b> and </b>. The consequences of using IMDB and Rotten Tomatoes as values to boost the score are important and obvious. As reported in the two following pictures 1 2, the results obtained by the research are different.

This result is similar to the one we reached in the last two queries of the point 3. Except that here the score also depended partly on the fullplot, and not on the title. The output of our search is then injected into the html document that is provided to the user via flask. Using the appropriate Jinja syntax (which is a templating language for Python) it is possible to populate the html document with the result coming



Commando

Unbeknownst to **Matrix**, the members of his former unit are being killed one by one.

The Animatrix

Anthology of nine short anime (Japanese animation) films tied in to the 1999 blockbuster "The **Matrix**"

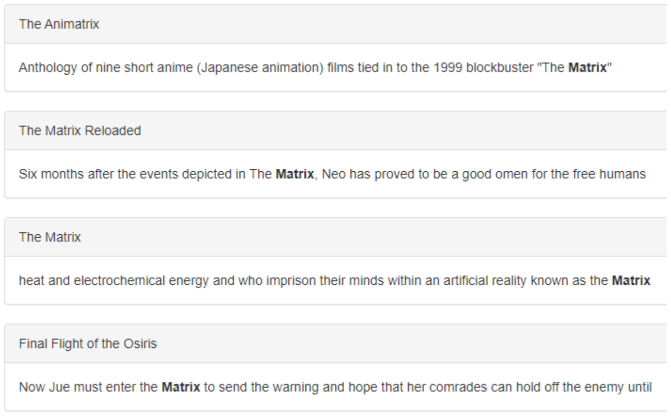
Program

Cis, one of the humans who have left The **Matrix**, is improving her skills in a virtual training program

The Matrix

heat and electrochemical energy and who imprison their minds within an artificial reality known as the **Matrix**

Fig. 1. boosting standard scores using IMDB rating value - first four records



The Animatrix

Anthology of nine short anime (Japanese animation) films tied in to the 1999 blockbuster "The **Matrix**"

The Matrix Reloaded

Six months after the events depicted in The **Matrix**, Neo has proved to be a good omen for the free humans

The Matrix

heat and electrochemical energy and who imprison their minds within an artificial reality known as the **Matrix**

Final Flight of the Osiris

Now Jue must enter the **Matrix** to send the warning and hope that her comrades can hold off the enemy until

Fig. 2. boosting standard scores using Rotten Tomatoes rating value - first four records

directly from elasticsearch. The code fragment used to show the results is shown for the reader's understanding.

```
<center>
   <p style="text-align: center;">Results: {{
        res['hits']['total']["value"] }} </p>
</center>
<br> {% for hit in res['hits']['hits'] %}

<div class="panel panel-default">
   <div class="panel-heading">{{
        hit['_source']['title']}}</div>
   <div class="panel-body">
      {{ (hit['highlight']['fullplot'][0] if
           'highlight' in hit else '') |safe}}
      <br>
   </div>
</div>
{% endfor %}
```

IV. Scoring Function Analysis

In order to calculate the scoring of the results, elasticsearch uses a function called **T**erm **f**requency — **I**nverse **d**ocument **f**requency (TF-IDF). The aim of this function is to search within a bag of words for which words are most important in our text. The score calculated by this function for each word is:

- directly proportional to the number of times the term is contained in the document
- inversely proportional with the frequency of the term in the collection

The idea behind this behaviour is to give more importance to terms which appear in the document, but which are generally infrequent. For the sake of completeness, a mathematical formulation is given here:

$$TfIdf(t,d,D) = Tf(t,d) * Idf(t,D)$$

Where the two components are:

$$Tf(\mathbf{t},\mathbf{d}) = \left[\frac{number\_of\_occurrences\_of\_term\_\mathbf{t}\_in\_document\_\mathbf{d}}{number\_of\_terms\_present\_in\_document\_\mathbf{d}}\right]$$

$$Idf(\mathbf{t}) = log\left[\frac{number\_of\_documents}{number\_of\_documents\_that\_have\_\mathbf{t}}\right]$$

And the terms are:

- **t**: the term that we want to search
- **d**: the document that we are analysing
- **D**: number of documents in the collection

This function is easy to calculate, but it only analyses words semantically and not lexically. We cannot therefore capture overall information about the words present but can only observe their semantic value. This is given by the operation of the function itself, that is based on the bag-of-words (*BoW*) model.

V. Additional Search Request

The query that I designed is presented below. In particular we want to find the first five movies that match *'mafia'* in the fullplot with *imdb_rating* greather than 7.0 and year less than 2000. Finally we want to boost the score using the *imdb_rating*.

```
res = es.search(index="project_3", body={
"query": {
   "function_score": {
      "query": {
         "bool":{
            "must":[{
               "match":{
                  "fullplot": "mafia"
               }},{
               "range": {
                  "imdb_rating": {
                     "gt": 7.0
                  }
               }},{
               "range": {
                  "year": {
                     "lt": 2000
                  }
               }}
            ]
         }
      },
      "script_score": {
         "script": {
            "source": "doc.containsKey('imdb_rating') ?
                 doc['imdb_rating'].value : 1"
         }
      }
   }
},
"size": 5
})

for doc in res['hits']['hits']:
   print (doc["_score"], doc["_source"]["title"])
```

The result of the query is:

```
80.38469 Donnie Brasco
77.17743 The Boondock Saints
75.18646 Nayakan
75.18646 Nayakan
74.71821 Confessions of a Police Captain
```

For this I used a *match* on the fullplot searching for *'mafia'* and then I used two *range* operators, one on the imdb_rating value and the other one on the year. Finally I boosted the score using the imdb_rating.