ElasticSearch

Informe de la Cuarta Práctica de Laboratorio

Implementando búsquedas en modelos de vector espacio

Diego Moreno y Daniel Lagos

Grupo 13 CAIM

Universitat Politècnica de Catalunya

Octubre 2025

Índice

Ejemplo Toy: Versiones Lenta y Rápida

Objetivos

El objetivo principal de esta práctica es describir el número de operaciones realizadas en las versiones lenta y rápida para el siguiente ejemplo toy, es decir, un ejemplo pequeño, con un tamaño 4 de vocabulario y 4 documentos.

- q = 0, 1, 1, 0

- Matriz de documentos

	t1	t2	t3	t4
d1	1,2	0	0	0
d2	0,7	1,3	1,5	0,1
d3	0	0	0	0,7
d4	2	0	0	0

Resultados

En este ejemplo, la consulta es q = [0,1,1,0], lo que significa que sólo los términos t2 y t3 están presentes en la búsqueda.

En el **slow search** el algoritmo compara la consulta con todos los documentos y todos los términos del vocabulario sin tener en cuenta si los términos aparecen o no en la consulta. Dado que hay 4 documentos y un vocabulario de 4 términos, se realizan **4×4=16** operaciones en total.

En el **quick search** el sistema utiliza el índice invertido para acceder únicamente a los documentos que contienen los términos presentes en la consulta. En este caso los términos t2y t3 solo aparecen en el documento d2 por lo que únicamente se evalúa ese documento y para esos dos términos. Por tanto se realizan solo **1×2=2** operaciones.

Implementar versión rápida

Objetivos

1. Primero hay que implementar una versión rápida del Excruciatingly Long Search que nos dan en el enunciado. Para implementar la versión eficiente, será necesario diseñar un índice invertido.

2. El código se puede hacer más eficiente si se implementa una ordenación top-r en el resultado final mediante un minheap.

3. A continuación, deberemos comparar los tiempos de ejecución de la versión rápida con el Excruciatingly Long Search. Asegúrate que ambos métodos devuelven los mismos resultados.

Resultados

Esta ha sido nuestra implementación del **quick search:**

```
def quick_search(query_tokens, inverted_index):
    sims = {}
    l2query = np.sqrt(len(query_tokens))
    for w in query_tokens:
        if w in inverted_index:
            for docid, weight in inverted_index[w]:
                sims[docid] = sims.get(docid, 0.0) + weight
    for d in sims:
        sims[d] /= l2query
    return heapq.nlargest(10, sims.items(), key=lambda kv: kv[1])
```

Utilizando esta implementación y el **slow search** dado por el enunciado hemos creado un scrypt que ejecuta ambos códigos y devuelve los resultados, que para la query: "**magic searching**" són los siguientes:

```
Preprocessing time: 36.83s

[('./arxiv/quant-ph.updates.on.arXiv.org/000650',
  np.float64(0.435797834780048)),
 ('./arxiv/quant-ph.updates.on.arXiv.org/000677',
  np.float64(0.43569542159714947)),
 ('./arxiv/cond-mat.updates.on.arXiv.org/003482',
```

```
  np.float64(0.4168723102044495)),
 ('./arxiv/quant-ph.updates.on.arXiv.org/001475',
  np.float64(0.3079756643218586)),
 ('./arxiv/cs.updates.on.arXiv.org/006235', np.float64(0.2771544913488471)),
 ('./arxiv/astro-ph.updates.on.arXiv.org/002083',
  np.float64(0.26842339084103656)),
 ('./arxiv/math.updates.on.arXiv.org/002825',
np.float64(0.26388066017358236)),
 ('./arxiv/astro-ph.updates.on.arXiv.org/001294',
  np.float64(0.25707275800793045)),
 ('./arxiv/hep-th.updates.on.arXiv.org/000265',
  np.float64(0.25615510052415014)),
 ('./arxiv/math.updates.on.arXiv.org/000955',
np.float64(0.25615510052415014))]

[('./arxiv/quant-ph.updates.on.arXiv.org/000650',
  np.float64(0.435797834780048)),
 ('./arxiv/quant-ph.updates.on.arXiv.org/000677',
  np.float64(0.43569542159714947)),
 ('./arxiv/cond-mat.updates.on.arXiv.org/003482',
  np.float64(0.4168723102044495)),
 ('./arxiv/quant-ph.updates.on.arXiv.org/001475',
  np.float64(0.3079756643218586)),
 ('./arxiv/cs.updates.on.arXiv.org/006235', np.float64(0.2771544913488471)),
 ('./arxiv/astro-ph.updates.on.arXiv.org/002083',
  np.float64(0.26842339084103656)),
 ('./arxiv/math.updates.on.arXiv.org/002825',
np.float64(0.26388066017358236)),
 ('./arxiv/astro-ph.updates.on.arXiv.org/001294',
  np.float64(0.25707275800793045)),
 ('./arxiv/hep-th.updates.on.arXiv.org/000265',
  np.float64(0.25615510052415014)),
 ('./arxiv/math.updates.on.arXiv.org/000955',
np.float64(0.25615510052415014))]

Tiempo slow:  0.04s
Tiempo quick: 0.00s
Coincidencia en top 10 resultados: 10/10
```

Vemos que la mayoría del tiempo de ejecución se dedica al preproceso de los documentos para hacer el cálculo del vector td-idf. Observamos que para una query tan simple y con un hardware potente como del que disponemos se observa un tiempo de ejecución de los algoritmos muy bajo, aunque ya se aprecia la diferencia entre el slow y el quick search.

Comparar versiones de queries

Objetivos

Compara los resultados que recibes al aplicar ciertas queries mediante el método de búsqueda por defecto de ElasticSearch y el método de Excruciatingly Long Search que nos da el enunciado.

Tendremos que determinar la similitud encontrada en los resultados y, con estos datos, averiguar cuál es el más rápido y eficiente.

Resultados

Hemos ejecutado los dos métodos con diversas queries y a continuación adjuntamos los resultados:

Query: 'searching magic'

Tiempo slow: 0.04s
Tiempo quick: 0.00s

—----------

Query: 'machine learning neural networks'

Tiempo slow: 0.06s
Tiempo quick: 0.01s

—----------

Query: 'animals hunting safari'

Tiempo slow: 0.04s
Tiempo quick: 0.00s

—----------

Query: 'blue house whale computer train car people hair at UPC'

Tiempo slow: 0.09s
Tiempo quick: 0.02s

—----------

Query: 'this is a random large text just to test limits on the query to see if there is a big difference between algorithms I like animals houses with roofs and windows in the city and the countryside i do not know what else to write'

Tiempo slow: 0.96s
Tiempo quick: 0.38s

En los resultados vemos que el tiempo de ejecución es proporcional al tamaño de la query, a la vez que el slow search es siempre mucho más lento que el quick search.

Conclusiones:

Gracias al trabajo realizado con esta práctica, hemos descubierto y mejorado nuevas técnicas de análisis de grandes grupos de documentos, técnicas que podremos usar en algoritmos de detección de plagios y buscadores masivos, por ejemplo. Cabe destacar que el punto más complicado de la práctica ha sido la instalación de elasticsearch en nuestro dispositivo en casa, ya que si no viene instalado hay que seguir unos pasos no explicados en el notebook.