ElasticSearch - Documentación de la práctica

> Se detalla a continuación documentaciones de los 3 scripts contenidos en este mismo directorio. Los enunciados tanto del script como de la propia tarea de descripción y muestreo están incluídos en cada una de las siguientes partes.
>
> [IMPORTANTE] Ejecutar todos los scripts desde la carpeta base de la práctica.

- Donato A. Martín (UO288787@uniovi.es)
- Repositorios de Información, Universidad de Oviedo
- 2024

Primera parte

Documentación correspondiente a esta fase señalando al menos las decisiones tomadas en relación con los documentos (p.ej., si se decidió ignorar alguna sección de los mismos), su preprocesamiento (p.ej., decisiones sobre el algoritmo de stemming, el tamaño de los shingles, el paso a minúsculas, etc.) así como comentarios (y enlaces) sobre la utilidad (o no) del uso de ChatGPT para resolver esta tarea. **Hasta 1 punto.**

Enunciado del script

Script Python para procesar la colección de documentos y crear el índice en ElasticSearch. **Hasta 1,5 puntos.**

Contenido

Para la realización de este ejercicio lo primero que debemos hacer es entender cómo se organiza y qué en qué consiste realmente la colección CISI, para ello se deberá leer el enunciado de la práctica y hacer un par de búsquedas rápidas en internet.

Una vez hecho este pequeño research podemos determinar que CISI.ALL es un fichero muy utilizado en el campo de la recuperación de información que contiene 1.460 textos en una notación que incluye campos como identificador, título o resumen.

Cada entrada sigue el siguiente formato:

```
.I <ID>
.T <Título>
.W <Resumen>
.B <Bibliografía>
.A <Lista de autores>
.X <Lista de referencias cruzadas a otros documentos>
```

Cuando ya es conocido nuestro objetivo, el siguiente paso es determinar qué información nos es relevante para parsear e indexar en nuestro elastic. Un poco en línea con el enunciado de la práctica podemos acordar que las referencias cruzadas no iban a aportar nada en lo que a recuperación de información se refiere ya que no nos muestran información relevante sobre la temática del documento, es por ello que el campo X no será indexado.

El resto; id, título, resúmen, fecha y autores, todo ello nos permitiría localizar el documento por aportar información directamente relacionada con el mismo. Parsearlo ha supuesto un reto mayor del que me esperaba, pese a no ser demasiado complicado, la estructuración en secciones dificulta la lectura por líneas al tener que guardar un contexto de la sección en la que te encuentras y si esta pertenece o no a un nuevo documento. En cualquier caso con este heurístico el trabajo fue realizado correctamente:

```
args = {
        "settings": {
            "analysis": {
                "filter": {
                    "estematizacion_ingles": {"type": "stemmer", "name":
"porter2"},
                },
                "analyzer": {
                    "analizador_personalizado": {
                        "tokenizer": "standard",
                        "filter": ["lowercase", "estematizacion_ingles"],
                    },
                    "analizador_autores": {
                        "tokenizer": "standard",
                        "filter": ["lowercase"]
                    }
                },
            }
        },
        "mappings": {
            "properties": {
                "text": {
                    "type": "text",
                    "analyzer": "analizador_personalizado",
```

```
                    "fielddata": "true",
                },
                "author": {
                    "type": "text",
                    "analyzer": "analizador_autores",
                    "fields": {
                        "keyword": {
                            "type": "keyword"
                        }
                    }
                },
            }
        },
    }
```

Tomaremos dos campos, texto y autor, en el segundo caso, debido a que estará mayormente compuesto por nombres propios, no aplicaremos stemming. Tampoco aplicaremos fielddata, ya que si queremos hacer agregaciones, el subcampo keyword nos permitirá hacer ordenación y agregaciones de manera más eficiente en términos de memoria.

```
for line in lines:

    if line.startswith('.T'):
        current_section = 'text'
        continue
    elif line.startswith('.W'):
        current_section = 'text'
        continue
    elif line.startswith('.B'):
        current_section = 'text'
        continue
    elif line.startswith('.A'):
        current_section = 'author'
        continue
    elif line.startswith('.X'):
        current_section = None
        continue

    if current_section:
        doc[current_section] = doc.get(current_section, "") + line.strip() + " "
```

Consideramos X e I secciones nulas pues I nos permite dividir todo el fichero en las diferentes entradas y X no tiene ser indexado como propósito. Más allá de eso simplemente se añade a un diccionario cada una de las líneas en función de la sección en la que nos encontremos, si la sección es nula, será ignorada.

También cabe recalcar que el título, así como la bibliografía, son añadidos al texto para mayor facilidad de recuperación de la información y es que estas secciones en concreto no tendrían demasiado interés para analizar por separado. Todo esto lo aclaro porque en primera instancia barajé tener cada uno de los campos separados.

El campo autor en cambio, considero que sí debería tener su propio campo ya que puede resultar interesante buscar específicamente por autor.

El algoritmo de stemming utilizado, porter2 es adecuado por su balance entre precisión y efectividad del sistema de búsqueda. Reduce correctamente las palabras a sus raíces facilitando una búsqueda más amplia y relevante en ElasticSearch, además de que este lo soporta nativamente. Existen otras alternativas, otros stemmers como Lancaster pueden ser útiles para textos con restricciones de longitud.

Se utiliza en el analizador personalizado del texto un filtro lowercase para asegurarnos que todo esté en minúscula dando como resultado un índice que no sea case sensitive.

Con esto concluye la documentación de la primera parte. Respecto al uso de ChatGPT general, podemos declarar que ha sido utilizado para arreglarme un bug a la hora de parsear el código, denotando mi oxidado python, así como en la búsqueda de información sobre la adecuación del stemming porter2 o del subcampo keyword.

Segunda parte

Documentación correspondiente a esta fase señalando al menos las decisiones tomadas en relación a la creación de las listas de documentos relevantes. Téngase en cuenta que son colecciones pequeñas y que ElasticSearch puede generar listas de resultados de hasta 10.000 documentos. No hay ningún problema en ello pero dependiendo de cuántos documentos se retornen se obtendrán unas medidas de rendimiento u otras y pueden arrojar luz sobre la percepción que tendría un usuario final del buscador. **Hasta 1 punto.**

Enunciado del script

Script Python para procesar la lista de consultas y generar listas de documentos potencialmente relevantes para las mismas. Este código será el que genere las denominadas runs que se le pasarían a ranx junto con los juicios de relevancia para el cálculo de las métricas de rendimiento. **Hasta 1,5 puntos.**

Contenido

Dado que ElasticSearch puede devolver hasta 10,000 documentos en una consulta y estamos trabajando con un conjunto de datos relativamente pequeño (CISI), se optó por limitar la cantidad de documentos devueltos a 100 por consulta. Esta decisión tiene como objetivo equilibrar entre obtener suficientes resultados para análisis significativos y evitar un sobrecargo de información que podría diluir la calidad de los resultados en términos de relevancia.

```
results = es.search(
    index="cisi",
    query={
        "match": {
            "text": query_text
        }
    },
    size=100
)
```

Las consultas se extraen de un archivo estructurado similar al de los documentos, donde cada consulta está precedida por un identificador .I y seguida por su texto bajo el tag .W. Se decidió procesar y extraer únicamente el texto de las consultas ignorando cualquier otra información que no fuese el contenido textual principal, ya que este es el componente más relevante para la recuperación de información.

Para facilitar la interoperabilidad y la manipulación posterior de los datos, los resultados de las runs se almacenan en un archivo JSON. Este formato es ampliamente soportado y fácil de integrar con otras herramientas, lo cual es útil para la fase de evaluación con ranx.

La búsqueda se realiza contra el campo text de los documentos, definido en la primera parte de la práctica.

Concluyendo esta parte y respecto al uso de ChatGPT podemos decir que le he pedido que me hiciese el parseo de las querys de dandole como ejemplo mi primer script, y también le he preguntado cómo debería almacenar cada run para más tarde parsear con ranx.

Tercera parte

Documentación correspondiente a esta fase señalando al menos las decisiones tomadas en relación al procesamiento de los juicios de relevancia así como la interpretación de los resultados de la evaluación, las posibles causas para los mismos y las implicaciones para un usuario final. **Hasta 2,5 puntos.**

Enunciado del script

Script Python que utilice ranx, reciba las runs generadas por el script anterior y sea capaz de procesar los juicios de relevancia para evaluar el rendimiento del buscador según las dos métricas que le corresponda utilizar al estudiante. **Hasta 2,5 puntos.**

Contenido

En esta fase final del proyecto, el script Python ha sido diseñado para evaluar el rendimiento del buscador utilizando los datos de relevancia y las runs generadas anteriormente. La métrica principal utilizada en esta evaluación ha sido Bpref, conocida por su eficacia en entornos donde no se dispone de una lista completa de documentos relevantes. Esta métrica es particularmente útil porque enfatiza la importancia de recuperar documentos relevantes antes de los no relevantes, lo cual es crucial en aplicaciones de recuperación de información realistas.

El script comienza con la lectura de los archivos que contienen las runs y los juicios de relevancia, asegurando que los datos sean cargados correctamente para su posterior procesamiento. La función calculate_bpref se encarga de calcular la métrica Bpref para cada consulta basándose en la información de relevancia disponible. Este proceso implica contabilizar los documentos relevantes que son recuperados antes de encontrar un documento no relevante, proporcionando así una medida del rendimiento del buscador que refleja la experiencia del usuario al evitar la frustración de encontrarse con resultados no deseados antes de los relevantes.

En cuanto a los resultados obtenidos, estos varían ampliamente, lo cual es esperado en pruebas de este tipo y ofrece una visión profunda del comportamiento del buscador en diferentes escenarios de consulta. Los resultados más destacados incluyen puntuaciones de Bpref extremadamente bajas para algunas consultas, como las identificadas con los números 5, 6, 8, 14, 16, 43, 61, 84, y 101, todas con un score de 0.0, lo que indica que no se recuperaron documentos relevantes antes de los no relevantes. Por otro lado, algunas

consultas como la 62 y la 55 muestran puntuaciones de Bpref excepcionalmente altas (0.555 y 0.496, respectivamente), señalando un rendimiento muy eficaz del sistema para esas consultas específicas.

```
C:/Users/dono/.pyenv/pyenv-win/versions/3.12.2/python.exe
c:/Users/dono/Documents/Proyectos/ElasticSearch/content/script_3.py
Bpref Scores: {'1': 0.083648393194707, '2': 0.03698224852071006, '3':
0.09039256198347108, '4': 0.234375, '5': 0.0, '6': 0.0, '7': 0.0625, '8': 0.0,
'9': 0.10640138408304498, '10': 0.14201183431952663, '11': 0.030752061504123008,
'12': 0.03550295857988166, '13': 0.04190315179326168, '14': 0.0, '15':
0.022456870910172518, '16': 0.0, '17': 0.038461538461538464, '18':
0.1983471074380165, '19': 0.08413351623228166, '20': 0.006944444444444444, '21':
0.144, '22': 0.03453186187255251, '23': 0.05902777777777777, '24':
0.13424556213017752, '25': 0.07897153351698807, '26': 0.14094387755102042, '27':
0.016786389413988658, '28': 0.13, '29': 0.057291666666666664, '30':
0.036645132546224105, '31': 0.08008599838753022, '32': 0.03345752063700782, '33':
0.034999999999999996, '34': 0.07825484764542937, '35': 0.15630070308274743, '37':
0.073125, '39': 0.05555555555555555, '41': 0.256198347107438, '42':
0.07479999999999999, '43': 0.0, '44': 0.03167533818938605, '45':
0.036937088885140835, '46': 0.04154280618311534, '49': 0.02681660899653979, '50':
0.06665825022093169, '52': 0.36444444444444446, '54': 0.07381776239907728, '55':
0.49691358024691357, '56': 0.037037037037037035, '57': 0.09567901234567902, '58':
0.12381852551984876, '61': 0.0, '62': 0.5555555555555555, '65':
0.27218934911242604, '66': 0.1942857142857143, '67': 0.0537109375, '69':
0.15555555555555556, '71': 0.1111111111111111, '76': 0.0813888888888889, '79':
0.21487603305785125, '81': 0.15702479338842976, '82': 0.06, '84': 0.0, '90':
0.026938775510204082, '92': 0.10318559556786705, '95': 0.2644628099173554, '96':
0.03703703703703704, '97': 0.11111111111111112, '98': 0.23424494649227115, '99':
0.17301038062283738, '100': 0.05246913580246913, '101': 0.0, '102':
0.23958333333333334, '104': 0.024793388429752063, '109': 0.06824042848641142,
'111': 0.41666666666666674}
```

Este análisis no solo resalta las fortalezas y debilidades del sistema actual, sino que también ofrece pistas sobre posibles mejoras. Las consultas con bajo rendimiento podrían beneficiarse de una revisión en la indexación o en el procesamiento de las consultas, tal vez ajustando la forma en que se manejan los términos de búsqueda o la relevancia de los documentos. Además, las variaciones en el rendimiento pueden indicar la necesidad de adaptar mejor el sistema a las especificidades del dominio o del conjunto de datos en cuestión.

La evaluación del buscador a través de estas métricas permite a los estudiantes de informática, como aquellos de la Universidad de Oviedo, entender profundamente las interacciones entre la teoría de recuperación de información y su aplicación práctica. Aprenden no solo cómo implementar estas herramientas, sino también cómo interpretar sus resultados y aplicar esos aprendizajes para mejorar los sistemas de información que podrían desarrollar en el futuro. Además, este tipo de ejercicio prepara a los estudiantes para enfrentar desafíos reales en el campo de la tecnología de la información, donde la capacidad de analizar y mejorar continuamente un sistema es invaluable.

Sobre el uso de ChatGPT para esta parte, se destaca en la estructuración inicial del código ya que se notó una falta amplia en práctica con librerías como ranx, así que fue necesario ese empujón para empezar a realizar el script. También se ha usado para obtener información acerca de Bpref y poder argumentar esta tercera parte.