Prácticas de Recuperación de Información

Curso 2024-2025

1 Introducción a la búsqueda léxica

En este *notebook*, implementaremos un sistema de búsqueda basado en BM25 para un *dataset* de pizzas. Usaremos la librería bm25s para crear un índice de búsqueda eficiente.

BM25S es una implementación rápida y ligera de Python de BM25, construida sobre NumPy y SciPy. Nos permite experimentar con la búsqueda léxica sin la configuración compleja que se requiere cuando usamos Elasticsearch.

Puedes obtener más información sobre BM25S en https://bm25s.github.io/.

2 Indexar con BM25S

En este ejemplo, cargaremos una colección de documentos y crearemos un índice utilizando bm25s. Más adelante, veremos cómo cargar el índice y realizar búsquedas en él.

2.1 Configuración

En primer lugar, vamos a instalar los paquetes necesarios de Python:

```
pip install bm25s[full]
```

Ahora, vamos a importar las librerías que necesitamos:

```
import bm25s # Para crear índices y buscar documentos con BM25
import Stemmer  # Para estematizar términos
import json  # Para cargar el corpus que nos viene en formato JSON
```

2.2 Preparar el corpus

Descargamos el fichero pizzas.json de
https://drive.google.com/uc?id=1_FRU7C04R3EczS3RrXuqnywd5uNJgtmo

Ahora, lo parseamos:

```
with open("pizzas.json", "r", encoding="utf-8") as f:
    corpus_content = f.read()
    corpus_content = json.loads(corpus_content)
```

Necesitamos preparar nuestro corpus para BM25S en dos formatos:

1. Una versión literal que mantenga la estructura original de los "documentos".
2. Una versión de texto plano para tokenización e indexación.

```
corpus_verbatim = list()
corpus_plaintext = list()
for entry in corpus_content:
    # Nota: nuestros "documentos" de pizza solo tienen 'id'
    # (el nombre de la pizza) y los "ingredientes",
    # pero BM25S espera 'id', 'title', y 'text'
    document = {"id": entry["id"], "title": entry["id"].lower(),
        "text": entry["ingredients"].lower()}
    corpus_verbatim.append(document)
    corpus_plaintext.append(entry["ingredients"].lower())
```

Vamos a procesar el texto. Para ello, aplicamos un *stemmer* para el idioma inglés y eliminamos palabras vacías al realizar la tokenización:

```
stemmer = Stemmer.Stemmer("english")

corpus_tokenized = bm25s.tokenize(corpus_plaintext, stopwords="en",
    stemmer=stemmer, show_progress=True)
```

2.3 Crear el *retriever* BM25S e indexar la colección

Ahora crearemos nuestro *retriever* basado en BM25 e indexaremos el corpus tokenizado.

Hay que tener en cuenta que tanto la función de ranking BM25 como la función para obtener el IDF tienen diferentes "sabores" que se pueden combinar. Las opciones disponibles son robertson, atire, bm25l, bm25+ y lucene (la implementación utilizada por Elasticsearch y OpenSearch).

```
# Escogemos lucene, pero puedes cambiarla y probar otras opciones
bm25_flavor = "lucene"
idf_flavor = "lucene"

retriever = bm25s.BM25(corpus=corpus_verbatim, method=bm25_flavor,
    idf_method=idf_flavor)
retriever.index(corpus_tokenized, show_progress=True)
```

Si utilizas BM25S “de verdad” en un script, deberías guardar tu índice para reutilizarlo. Obtendrás una carpeta con el índice:

```
retriever.save("pizzas", corpus=corpus_verbatim)
```

3 Buscar con BM25S

Aunque estamos demostrando el uso de BM25S en el mismo *notebook*, la indexación y la búsqueda son tareas separadas. En esta sección, mostraremos cómo enviar consultas a un índice BM25S.

3.1 Configuración

En primer lugar, instalamos los paquetes necesarios (sí, ya se instalaron para este *notebook*):

```
pip install bm25s[full]
```

Ahora, importamos las librerías necesarias:

```
import bm25s # Para crear índices y buscar documentos con BM25
import Stemmer # Para estematizar términos
# Ten en cuenta que no necesitamos importar json, los documentos se
# almacenan en el índice, por lo que no necesitamos acceder al conjunto de
# datos original
```

Necesitamos crear una instancia de un *stemmer* para aplicarlo a las consultas:

```
stemmer = Stemmer.Stemmer("english")
```

3.2 Cargar el *retriever* BM25S

Cargamos el índice **y** el corpus/colección/*dataset* original; se almacenará en retriever.corpus.

```
retriever = bm25s.BM25.load("pizzas", load_corpus=True)
```

3.3 Enviar consultas

En este momento tenemos solo una consulta, pero puedes agregar más...

```
consultas = [{"id": "craving", "ingredients": "tomato oregano olives"}]
```

Iteramos sobre cada consulta para enviarla al *retriever* y obtener resultados.

```
for query in queries:
    # Para evaluar después el rendimiento, es útil conservar el id de la
    # consulta, de forma que comparemos los resultados obtenidos con el
    # retriever con los resultados esperados, segúnlo determinado por
    # los juicios de relevancia
    #
    query_id = query["id"]

    # Recuerda que el contenido está en el campo 'ingredients' ...
    #
    query_string = query["ingredients"].lower()

    # Imprimimos la consulta (solo para saber que hemos "pedido" al sistema
    # de recuperación)
    #
    print(f"---\nQuery {query_id}: {query_string[0:80]}...")

    # Tokenizamos la consulta. ¡Atención! Debemos tokenizar la consulta
    # con la misma configuración que usamos al indexar el corpus. En este
    # caso, eliminando las palabras vacías del inglés y aplicando el
    # stemmer de inglés
    #
    query_tokenized = bm25s.tokenize(query_string, stopwords="en",
        stemmer=stemmer, show_progress=False)

    # Retornamos los k resultados más "top" como una tupla con nombre.
    # Por favor, lee la documentación de BM25S para otras alternativas.
    #
    # ¡Atención! Si pides más documentos que los que hay en la colección,
    # tendrás una excepción.
    #
    results = retriever.retrieve(query_tokenized, corpus=retriever.corpus,
        k=5, return_as="tuple", show_progress=False)

    # Los documentos y sus puntuaciones se guardan en dos campos
    # diferentes de la tupla.
    #
    returned_documents = results.documents[0]
    relevance_scores = results.scores[0]

    # Imprimimos los resultados
    #
    print("\nResults:")

    for i in range(len(returned_documents)):
        print(f"\t{i}\t{relevance_scores[i]}\t{returned_documents[i]
            ['title']}\t{returned_documents[i]['text'][0:80]}...")
```

El resultado que obtenemos es:

```
Query craving: tomato oregano olives...

Results:
        0       0.5406627058982849      capricciosa     artichokes mozzarella mushrooms oil olives tomato...
        1       0.2751682996749878      marinara        garlic oil oregano tomato...
        2       0.25040316581726074     romana  anchovies mozzarella oil oregano tomato...
        3       0.25040316581726074     viennese        mozzarella oil oregano sausage tomato...
        4       0.03480454906821251     quattro formaggi        fontina gorgonzola mozzarella tomato stracchino...
```

4 Conclusión

En este *notebook*, hemos explorado la aplicación de la librería bm25s, un motor de búsqueda léxica, utilizando un *dataset* de juguete. Hemos recorrido todo el proceso, desde la configuración del entorno y la preparación del corpus hasta la creación de un índice y la realización de búsquedas.

En el siguiente *notebook*, aplicaremos el mismo enfoque a una colección de evaluación estándar, LISA, y examinaremos cómo evaluar el rendimiento utilizando juicios de relevancia.