Prácticas de Recuperación de Información

Curso 2024-2025

1 Búsqueda léxica y evaluación del rendimiento

En este *notebook*, implementaremos un sistema de búsqueda basado en BM25 para una colección de evaluación estándar: LISA.

La colección se ha reformateado en archivos JSON y, además de las consultas originales, contiene consultas parafraseadas junto con sus traducciones al español. Esto te permitirá examinar las limitaciones de la búsqueda léxica en comparación con la búsqueda semántica.

Al igual que en el *notebook* anterior, utilizaremos bm25s.

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 LISA-collection.zip de
https://drive.google.com/uc?id=1dIKh0BWK1yyEkGEfUTtx7wPu1LDB8-Sc

Posteriormente, lo descomprimimos y parseamos lisa-corpus.json:

```
with open("lisa-corpus.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 sencillo para tokenización e indexación.

```
corpus_verbatim = list()
corpus_plaintext = list()
for entry in corpus_content:
    # Nota: nuestros documentos tienen 'id', 'title' y 'content'
    # Vamos a concatenar title y content porque nos interesa para el
    # indexado
    document = {"id": entry["id"], "title": entry["title"].lower(),
        "text": entry["content"].lower()}
    corpus_verbatim.append(document)
    corpus_plaintext.append(f"{entry['title'].lower()}
        {entry['content'].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("LISA", corpus=corpus_verbatim)
```

3 Buscar con BM25S

Aunque estamos demostrando 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("LISA", load_corpus=True)
```

3.3 Enviar consultas

Vamos a cargar todas las consultas: lisa-queries.json, lisa-queries-paraphrased.json y lisa-queries-spanish.json.

```
all_queries = dict()

for filename in ["lisa-queries.json", "lisa-queries-paraphrased.json",
    "lisa-queries-spanish.json"]:
    with open(filename, "r", encoding="utf-8") as f:
        content = f.read()
        data = json.loads(content)
        all_queries[filename] = data
print(all_queries)
```

Vamos a crear una función para enviar las consultas y obtener los resultados asociados a ellas, para posteriormente calcular el rendimiento (precisión, exhaustividad y F1-score).

```
# Función para cargar consultas desde un fichero json para enviar a una
# colección dada y obtener listas de resultados.
def submit_queries_and_get_run(queries, stemmer, retriever,
    max_results=10):
    run = {}

    # Iteramos para enviar cada consulta al retriever para obtener
    # resultados
    for query in queries:
        # Para evaluar después el rendimiento, es útil conservar el id de
        # la consulta
        query_id = query["id"]

        query_string = query["query"].lower()

        # 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.
        #
        results = retriever.retrieve(query_tokenized,
            corpus=retriever.corpus, k=max_results, 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]

        returned_ids = []
        for i in range(len(returned_documents)):
##            print(f"\t{i}\t{relevance_scores[i]}\t{
##                returned_documents[i]['id']}\t
##                {returned_documents[i]['title'][0:80]}...")
            returned_ids.append(str(returned_documents[i]["id"]))
        run[query_id] = returned_ids

    return run
```

3.3.1 Consultas originales de la colección LISA

Estas son las consultas originales en la colección de evaluación; fueron recopiladas por los autores de la colección, por lo que el desajuste de vocabulario entre las consultas y los documentos debería ser mínimo.

```
original_queries = all_queries["lisa-queries.json"]
original_run = submit_queries_and_get_run(original_queries, stemmer,
    retriever)
```

3.3.2 Consultas parafraseadas

Estas consultas se han parafraseado utilizando un LLM, por lo que deberían transmitir un significado similar a las consultas originales pero el desajuste de vocabulario va a ser mayor, en comparación con dichas consultas originales.

Por ejemplo, la consulta 18 originalmente era:

I WOULD BE PLEASED TO RECEIVE ANY INFORMATION ON TELECOMMUNICATIONS NETWORKS. THIS COULD INCLUDE BOTH WIDE AREA NETWORKS (MESH) AND LOCAL AREA NETWORKS (BUS, RING). TELECOMMUNICATIONS, LOCAL AREA NETWORKS, WIDE AREA NETWORKS, DATA TRANSMISSION, DISTRIBUTED PROCESSING, DISTRIBUTED ROUTING, PACKET-SWITCHED NETWORKS, MESH, BUS, RING.

Y se parafraseó como:

I'd like information on how computers talk to each other over long and short distances. This includes big networks that cover wide areas and smaller networks within buildings or campuses.

Los humanos podríamos parafrasear la misma consulta de formas muy diferentes, por lo que te animamos a que pruebes tus propias versiones. Sin embargo, el único propósito de esto es ilustrar las diferencias significativas entre el vocabulario utilizado en las consultas comunes de los usuarios y el que se encuentra en los documentos de la colección.

```
paraphrased_queries = all_queries["lisa-queries-paraphrased.json"]
paraphrased_run = submit_queries_and_get_run(paraphrased_queries,
    stemmer, retriever)
```

3.3.3 Consultas en español

Estas consultas han sido traducidas automáticamente al español, por lo que es muy poco probable que un motor de búsqueda léxica produzca resultados significativos. Por lo tanto, el propósito de este experimento es demostrar que la recuperación de información interlingüística es prácticamente imposible si solo se depende de la búsqueda léxica.

Por ejemplo, la consulta 18 en español es:

ME ENCANTARÍA RECIBIR CUALQUIER INFORMACIÓN SOBRE REDES DE TELECOMUNICACIONES. ESTO PODRÍA INCLUIR TANTO REDES DE ÁREA AMPLIA (MESH) COMO REDES DE ÁREA LOCAL (BUS, RING). TELECOMUNICACIONES, REDES DE ÁREA LOCAL, REDES DE ÁREA AMPLIA, TRANSMISIÓN DE DATOS, PROCESAMIENTO DISTRIBUIDO, ENRUTAMIENTO DISTRIBUIDO, REDES DE CONMUTACIÓN DE PAQUETES, MALLA, BUS, ANILLO.

```
spanish_queries = all_queries['lisa-queries-spanish.json']
spanish_run = submit_queries_and_get_run(spanish_queries, stemmer,
    retriever)
```

3.4 Evaluar el rendimiento

Vamos a calcular la precisión y la exhaustividad (https://en.wikipedia.org/wiki/Precision_and_recall) y F1-score (https://en.wikipedia.org/wiki/F-score); posteriormente, obtendremos micropromedios y macropromedios (https://danigayo.prof/downloads/blindlight-dgayo-disertacion-glosario.pdf).

```
# Función para calcular micro y macropromedios de la precisión,
# exhaustividad y F1-score
def compute_precision_recall_f1(run, relevance_judgements):
    # Inicializar listas para guardar la precisión, exhaustividad y
    # F1-score de cada consulta
    precision_values = []
    recall_values = []
    f1_values = []

    # Initializar contadores globales para micropromedios
    global_retrieved = 0
    global_relevant = 0
    global_retrieved_and_relevant = 0

    # Calcular precisión, exhaustividad y F1-score para cada consulta
    for query_id in run.keys():
        retrieved_results = run[query_id]
        relevant_results = relevance_judgements[query_id]
        relevant_and_retrieved = set(retrieved_results) &
            set(relevant_results)

        # Actualizar contadores globales
        global_retrieved += len(retrieved_results)
        global_relevant += len(relevant_results)
        global_retrieved_and_relevant += len(relevant_and_retrieved)

        # Calcular precisión y exhaustividad
        precision = len(relevant_and_retrieved) / len(retrieved_results)
            if len(retrieved_results) > 0 else 0
        recall = len(relevant_and_retrieved) / len(relevant_results)
            if len(relevant_results) > 0 else 0

        # Calcular F1-score si la precisión y la exhaustividad no son cero
        if (precision + recall) > 0:
            f1 = 2 * (precision * recall) / (precision + recall)
            f1_values.append(f1)

        # Añadir precisión y exhaustividad para la consulta actual
        precision_values.append(precision)
        recall_values.append(recall)

    # Calcular macropromedios
    macro_average_precision = sum(precision_values) /
        len(precision_values) if precision_values else 0
    macro_average_recall = sum(recall_values) / len(recall_values)
       if recall_values else 0
    macro_average_f1 = sum(f1_values) / len(f1_values)
       if f1_values else 0

    # Mostrar los macropromedios
    print(f"Macro-averaged Precision:{round(macro_average_precision,3)}")
    print(f"Macro-averaged Recall: {round(macro_average_recall,3)}")
```

```
    print(f"Macro-averaged F1: {round(macro_average_f1,3)}")
    print("")

    # Calcular micropromedios
    micro_average_precision = global_retrieved_and_relevant /
        global_retrieved if global_retrieved > 0 else 0
    micro_average_recall = global_retrieved_and_relevant / global_relevant
        if global_relevant > 0 else 0
    micro_average_f1 = (2 * (micro_average_precision *
        micro_average_recall) / (micro_average_precision +
        micro_average_recall))
        if (micro_average_precision + micro_average_recall) > 0 else 0

    # Mostrar micropromedios
    print(f"Micro-averaged Precision:{round(micro_average_precision,3)}")
    print(f"Micro-averaged Recall: {round(micro_average_recall,3)}")
    print(f"Micro-averaged F1: {round(micro_average_f1,3)}")
```

3.4.1 Cargar los juicios de relevancia

Solo podemos comprobar el rendimiento si tenemos juicios de relevancia.

```
# Cargamos los juicios de relevancia
with open("lisa-relevance-judgements.json", "r", encoding="utf-8") as f:
    relevance_judgements = json.load(f)

# El formato de los juicios de relevancia necesita ser ligeramente
# transformado
relevance_judgements_reformat = dict()
for entry in relevance_judgements:
  query_id = entry["query_id"]
  rel_docs = entry["rel_docs"].split(", ")
  relevance_judgements_reformat[query_id] = rel_docs
```

3.4.2 Rendimiento cuando se utilizan las consultas originales

```
compute_precision_recall_f1(original_run, relevance_judgements_reformat)
```

Presta atención a estos valores de salida porque te sorprenderás cuando evalúes la búsqueda semántica:

Macro-averaged Precision: 0.303
Macro-averaged Recall: 0.394
Macro-averaged F1: 0.332
Micro-averaged Precision: 0.303
Micro-averaged Recall: 0.28
Micro-averaged F1: 0.291

3.4.3 Rendimiento cuando se utilizan las consultas parafraseadas

```
compute_precision_recall_f1(paraphrased_run,
    relevance_judgements_reformat)
```

El impacto del desajuste de vocabulario es **enorme**:

Macro-averaged Precision: 0.143
Macro-averaged Recall: 0.208
Macro-averaged F1: 0.213
Micro-averaged Precision: 0.143
Micro-averaged Recall: 0.132
Micro-averaged F1: 0.137

3.4.4 Rendimiento cuando se utilizan las consultas en español

```
compute_precision_recall_f1(spanish_run, relevance_judgements_reformat)
```

La búsqueda léxica simplemente no puede manejar la recuperación de información interlingüística:

Macro-averaged Precision: 0.003
Macro-averaged Recall: 0.001
Macro-averaged F1: 0.032
Micro-averaged Precision: 0.003
Micro-averaged Recall: 0.003
Micro-averaged F1: 0.003

4 Conclusión

En este *notebook*, hemos implementados un sistema de búsqueda basado en BM25 para la colección LISA, utilizando la librería bm25s.

Hemos mostrado el proceso de indexación de la colección, la realización de búsquedas con diferentes conjuntos de consultas y la evaluación del rendimiento del sistema.

Los resultados resaltan algunos aspectos cruciales de la búsqueda léxica y sus limitaciones.

Nuestros experimentos destacaron que, si bien BM25 funciona razonablemente bien con las consultas originales de la colección, su efectividad disminuye significativamente cuando se enfrenta a consultas parafraseadas. Esta disminución subraya el desafío del desajuste de vocabulario en los modelos de recuperación de información léxica.

El fracaso absoluto al tratar con consultas escritas en un idioma diferente (español) del de la colección (inglés) no es nada sorprendente para un sistema léxico; y, de hecho, la recuperación de información interlingüística requiere técnicas más avanzadas, como la búsqueda semántica utilizando modelos de lenguaje multilingües previamente entrenados.

El próximo *notebook* cubrirá esos aspectos.