Implementación de un Sistema de Recuperación de Información

Dianelys Cruz Mengana,
Jordan Pla González,
Felix D. Acosta Rodríguez.

Abstract. En el presente artículo se exponen las ideas principales empleadas en la implementación, evaluación y análisis de un Sistema de Recuperación de Información. Se emplean tres modelos de Recuperación de Información, cada uno sobre tres colecciones de documentos distintas. De esta manera es posible comprobar en la práctica qué modelos o variaciones en los modelos son mejores para obtener resultados óptimos en la Recuperación de Información.

Keywords: Modelo Vectorial, Modelo Indexación de Semántica Latente, Modelo Booleano.

Introducción

Para la implementación del sistema se utiliza el Modelo Vectorial sobre la colección Vaswani[1], el Modelo de Indexación de Semántica Latente sobre la colección Cranfield[2] y el Modelo Booleano sobre la colección Trec-covid[1]. En cada uno de los casos se realizan todas las fases del proceso de recuperación, como se verá más adelante. Además, es posible usar cada modelo de recuperación con cualquier otro *dataset*.

1 Implementación

1.1 Ejecución del proyecto

Para ejecutar el Proyecto, es necesario tener FastAPI previamente instalado, un marco web basado en Python que nos permite desarrollar API web rápidamente. FastAPI es fácil de usar, liviano y se puede emplear en el desarrollo web y de aplicaciones a pequeña escala. Por tales motivos decidimos usar este *framework* en el diseño de nuestro motor de búsqueda. Además, es preciso instalar el servidor ASGI

[1] ir-datasets.com/index.html
[2] datasets de los profesores

(Asynchronous Server Gateway)[3] uvicorn. Para ejecutar el software se introduce en la terminal la línea de comando:

`uvicorn main:app --reload`.

Luego debemos acceder a **http://127.0.0.1:8000**. Para realizar una consulta es preciso navegar por la URL generada e introducir **`/search?qry=<msg>`** donde "<msg>" sería la consulta que se desea realizar, y para evaluar el modelo se introduciría **`/evaluate`**. De esta manera es posible realizar consultas, teniendo en cuenta el modelo y el corpus especificados en la línea 6 del archivo **`main.py`** del proyecto. Si se desea cambiar el modelo y la colección de documentos especificados es necesario modificar esta línea de la siguiente forma:

```
system = Reindj(<corpus_type>, <model_type>)
```

1.2 Diseño del sistema

El siguiente esquema presenta la arquitectura general que refleja el motor de búsqueda.

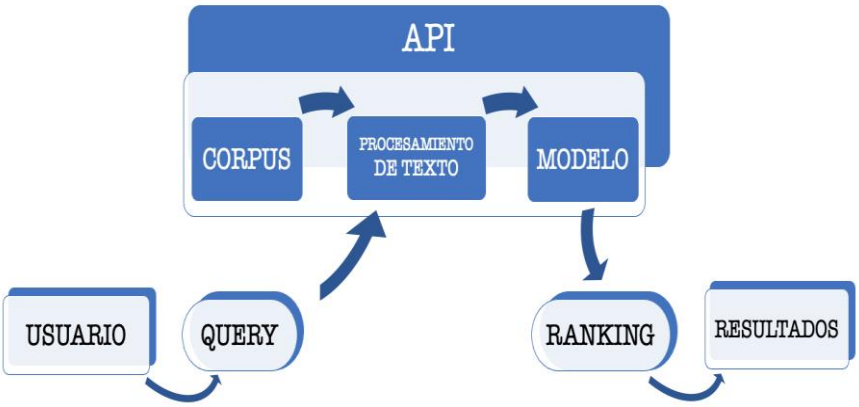

Figure 1. Arquitectura del Motor de Búsqueda

El flujo del funcionamiento se desarrolla de la siguiente forma:

El usuario realiza una consulta en lenguaje natural, que luego es preprocesada. Teniendo ya el corpus de documentos también preprocesado de forma *offline*, se aplica un modelo de recuperación de información para devolver un conjunto de documentos.

[3] Tiene como objetivo proporcionar una interfaz estándar entre servidores web, marcos y aplicaciones Python con funciones asincrónicas.

1. Preprocesamiento del corpus de documentos: Consta de dos partes fundamentales:

   - ***Parser***: Dado un corpus de documentos permite obtener una lista que representa el conjunto de documentos. Dado que este proceso depende en gran medida del formato del corpus, hemos decido tener una clase por cada uno de los tipos de corpus que se trabajaron: Cranfield, Vaswani y Trec-Covid.

   - ***Indexer***: Permite obtener los términos indexados que se tendrán en cuenta en el modelo de recuperación de información. En este caso aplicamos algunas técnicas de procesamiento del lenguaje natural, mediante el empleo de la librería de Python NLTK (*Natural Language Toolkit*), como:

     - Eliminación de *stopwords* o palabras de paso:
       Se refiere a la eliminación de palabras que puede que no tengan utilidad semántica dentro de un uso o contexto determinado. La elección de estas palabras depende del idioma (en este caso inglés).
       Es posible obtener una lista de palabras consideradas como stopwords a través del método `stopwords.words` de NLTK

     - Lematización:
       Técnica computacional para determinar el lema de una palabra. En este caso usamos el método lemmatize, que se encuentra en el módulo `nltk.stem.wordnet.WordNetLemmatizer`.

     - Etiquetado gramatical:
       Consiste en asignar a cada palabra su categoría gramatical. NLTK dispone de la función `pos_tag`, que permite obtener dicha categorización para cada uno de los *tokens* en una lista de entrada.

   A partir del empleo de estas técnicas, solo consideramos como términos indexados a aquellos que no representen palabras de paso y cuya categoría gramatical sea sustantivo, verbo o adjetivo.

   Para aportar eficiencia al motor de búsqueda, en esta fase de preprocesamiento se computan todos los datos necesarios para la aplicación de cada uno de los modelos respectivamente.

2. Procesamiento de la consulta: Se realiza de manera análoga al procesamiento de los documentos. Solo se tienen en cuenta los términos indexados que se obtienen en el preprocesamiento de la colección de documentos.

3. Aplicación de un Modelo de Recuperación de Información:

   I. Modelo Vectorial:

   Aplicando las fórmulas correspondientes al modelo vectorial se calculan:

   - Los pesos de los términos en las consultas y en los documentos. En el caso de las consultas se usa 0.5 como amortiguado.
   - La similitud de cada uno de los documentos con la consulta.

   En particular, el valor idf se calcula de la forma:

$$idf_i = \log \frac{N+1}{n_i}$$

   Esta modificación con respecto al modelo original se realiza para otorgar un peso algo mayor a los términos que aparecen en todos los documentos respecto a los que no aparecen.

   II. Modelo Booleano:

   El modelo booleano, visto en conferencias, permite aplicar reglas del álgebra booleana a la recuperación de documentos. En nuestro proceso para implementar el mismo, con el objetivo de permitir hacer consultas menos técnicas se toman los términos relevantes de una consulta y se hace un AND entre ellos y los términos indexados de los documentos. Solo se devuelve un documento si todos los términos indexados aparecen en el mismo.

   Para indexar los términos de un documento se tiene por cada uno un vector binario donde la posición i-ésima del vector es 0, si no aparece y 1, si sí. El análisis de los términos indexados se hace eliminando las *stopwords* del idioma y quedando solo con el lema de los términos que representen sustantivos, adjetivos o verbos para mejor comprensión del sistema.

   III. Modelo Indexación de Semántica Latente

   Este modelo es una alternativa al modelo vectorial. La idea fundamental es mapear los términos y documentos a un espacio de conceptos con menor dimensión; teniendo

en cuenta que las palabras que se utilizan en el mismo contexto suelen tener significados similares. El proceso de reducción de dimensión y de agrupar por conceptos se lleva a cabo a través de la Descomposición en Valores Singulares (SVD) de la matriz de términos-documentos, llamémosle A. Existen distintas formas de calcular los términos de dicha matriz. En función de contribuir a la eficiencia del motor de búsqueda la forma escogida fue:

$$a_{ij} = g_i \times l_{ij} \qquad g_i = \frac{1}{\sqrt{\Sigma_j tf_{ij}^2}} \qquad l_{ij} = tf_{ij}$$

Donde:

- tf_{ij} es la frecuencia del término i en el documento j

El parámetro k utilizado para reducir las dimensiones, luego de hallar la descomposición SVD de la matriz A, fue 150.

4. Establecimiento de Ranking: Usando un *heap*, mantenemos ordenados los documentos en orden creciente atendiendo a su similitud respecto a la consulta. Solo se recuperan los documentos con similitud mayor o igual a umbral determinado (se escogió en cada caso un valor para el umbral conveniente) y la máxima cantidad de documentos recuperados es 500.

2 Evaluación del Sistema

Modelo/Corpus	Vectorial	Semántica Latente	Booleano
Vaswani	P: 0.323 R: 0.029 F: 0.058 F1: 0.044 Umbral: 0.8	P: 0.230 R: 0.056 F: 0.104 F1: 0.085 Umbral: 0.8	
Cranfield	P: 0.680 R: 0.116 F: 0.247 F1: 0.190 Umbral: 0.8	P: 0.457 R: 0.079 F: 0.163 F1: 0.127 Umbral: 0.8	
Trec-Covid	P: 0.520 R: 0.025 F: 0.066 F1: 0.046 Umbral: 0.8	P: 0.190 R: 0.013 F: 0.032 F1: 0.023 Umbral: 0.8	

3 Análisis del Sistema

a) En el caso del modelo Indexación de Semántica Latente es probable que si se usa alguna de las otras fórmulas para calcular la matriz de términos-documentos, mejoren los resultados al aplicar este modelo. Por ejemplo, podría utilizarse la siguiente fórmula, que es la recomendada, de acuerdo con los resultados empíricos:

$$a_{ij} = g_i \times l_{ij} \qquad g_i = 1 + \sum_j \frac{p_{ij} \times \log p_{ij}}{\log n} \qquad l_{ij} = \log\left(tf_{ij} + 1\right)$$

Donde:

- $p_{ij} = \frac{tf_{ij}}{gf_i}$
- gf_i es el número de veces que i ocurre en toda la colección.
- n es la cantidad de documentos de la colección.

b) El umbral a partir del cual se consideran los documentos como relevantes es configurable en el archivo **`config.json`**, modificando la llave "umbral". Se podría añadir una llave para cada uno de los pares de modelos-corpus analizados, con valores que arrojen mejores resultados para cada caso en particular.

c) No fue posible comprobar los resultados de la aplicación del Modelo Indexación de Semántica Latente sobre la colección de documentos Vaswani en su completitud. En este caso, no se puede alojar memoria para la matriz de términos-documentos, dado que la cantidad de documentos es 11429. La solución encontrada fue reducir la colección a 2000 documentos para poder aplicar dicho modelo.