Proyecto de Sistemas de Recuperación de Información

Modelo Booleano, Vectorial y LSA

Kevin Talavera Díaz C-311

Facultad de Matemática y Computación

Universidad de La Habana

Curso 2021-2022

Link: https://github.com/KevinTD15/SRI-models.git

Resumen. Proyecto final de Sistemas de Recuperación de Información. Fueron implementados en Python los modelos Booleano, Vectorial y Análisis de Semántica Latente. Está compuesto por 5 módulos: Módulo principal que contiene la interfaz visual, Procesamiento de Texto, Modelos Recuperación de Información, Crawling y Datasets.

Keywords: LSA: Análisis de Semántica Latente, dataset: base de datos, path: dirección

- **Introducción**

  Según la mayoría de estudios que se han estado realizando en los últimos años la recuperación y organización de la información es uno de los aspectos que han cobrado mayor relevancia. En la actualidad estos estudios resaltan la vital importancia que ha cobrado ese campo. Esto se debe en gran medida a que los buscadores de internet están situados como el primer método utilizado para obtener cualquier tipo de información sea para el uso que sea (académico, empresarial).

  Debido a esto es de vital importancia conocer cuáles son los métodos o modelos de recuperación utilizados por los grandes buscadores (booleano, probabilístico, vectorial).

  El **modelo booleano** constituye el primer modelo teórico empleado para establecer el subconjunto de documentos relevantes, en relación a una consulta específica realizada a una colección de documentos, sean estos, páginas disponibles en la web o en una biblioteca digital. Está basado en el álgebra de Boole, por lo que se considera como un modelo simple y fácil de implementar, por tal motivo fue el preferido en los Sistemas de Recuperación tempranos. Se conoce como **modelo de espacio vectorial** a un modelo algebraico utilizado para filtrado, recuperación, indexado y cálculo de relevancia de información. Representa documentos en lenguaje natural de una manera formal mediante el uso de en un espacio lineal multidimensional. El **modelo LSA** es un tipo de análisis computacional que, basado en un algoritmo matemático, permite determinar y cuantificar la similitud de significado entre piezas textuales pertenecientes a un mismo dominio de conocimiento.

- **Módulo Principal**

  El objetivo de este módulo es ofrecerle al usuario una interfaz visual con el propósito facilitarle la interacción con el proyecto. Para implementar dicha interfaz se hizo uso de la biblioteca **tkinter** de python que ofrece una gran variedad de herramientas para este fin.
  La interfaz visual cuenta un menú con 4 opciones , de las cuales 3 de ellas corresponden con los modelos de recuperación de información implementados y la otra es la de salir de la aplicación.
  Al seleccionar un modelo se debe elegir si utilizar **crawling**, alguno de los **3 datasets** o hacer consultas en una dirección local de la **PC**.

- **Procesamiento de texto**

  Para poder implementar mecanismos de recuperación sobre una colección de documentos de textos es necesario obtener una representación de los mismos. Con el objetivo de lograr dicha representación se utilizó una biblioteca de **Python** llamada **nltk** la cual modifica los textos de forma tal que sólo queden palabras que aporten significado, por ejemplo: sustantivos, adjetivos, entre otras.

  La forma de trabajo con esta biblioteca, que se ve reflejado en el módulo **tokenizer.py** del proyecto fue la siguiente:

  1- Se recibe el conjunto de documentos.
  2- Se itera por cada uno de estos.
  3- Sea d_j el documento correspondiente a la iteración i-ésima de estos:
     a. Todos los términos de d_j son llevados a minúscula.
     b. Haciendo uso de la función **stopwords** de **nltk** son eliminados los artículos,

preposiciones y otros términos no deseados.

4- Al finalizar con todos los documentos, estas transformaciones son devueltas al módulo **model.py.**

- **Modelos de Recuperación**

De manera general, en este punto se recibe tanto el conjunto de documentos como la consulta.

➢ **Booleano**

1. El conjunto de documentos es procesado mediante el módulo **tokenizer.py** previamente explicado.
2. Se crea una matriz en la cual las filas son los documentos y las columnas los términos.
3. La consulta puede procesarse de 2 formas, primeramente, si es escrita en lenguaje natural se hace de igual forma que los documentos, la otra forma es que si ya es escrita en forma de álgebra booleana nos saltamos ese paso de procesamiento ya que no es necesario.
4. Haciendo uso de la biblioteca **sympy** cada término de la consulta se vuelve un símbolo, el cual esta biblioteca reconoce como una variable con la que se pueden hacer operaciones booleanas.
5. Mediante la función **to_dnf** del propio **sympy** la consulta ya reconocida como expresión booleana es llevada a **FND** (Forma Normal Disyuntiva).
6. Cada una de las componentes conjuntivas de esta es buscada en la matriz de ocurrencia previamente mencionada y los que se correspondan son documentos a ser resueltos.

➢ **Vectorial**

1. El conjunto de documentos es procesado mediante el módulo **tokenizer.py** previamente explicado.

2. Se crea una lista con todos los términos, la matriz correspondiente a los índices de frecuencia (idf_i), y la matriz de pesos (w_{ij}). Todo esto se hace en un mismo método en un único recorrido por motivos de eficiencia.
3. Se procesa la consulta mediante el módulo **tokenizer.py**.
4. Se crea la tabla de frecuencia (tf_{ij}) correspondiente a la consulta.
5. Se crea la lista correspondiente a los índices de frecuencia (idf_i) de los términos de la consulta.

6. Se calculan los pesos de cada término de la consulta ($\boldsymbol{w_{iq} = tf_{ij} * idf_i}$).

7. Se calcula la función de similitud de la consulta para cada documento:

$$\boldsymbol{sim(d_j, q) = \frac{\sum_{i=1}^{n} w_{ij}\ x\ w_{iq}}{\sum_{i=1}^{n} w_{ij}^2\ x\ \sum_{j=1}^{n} w_{iq}^2}}$$

➢ **LSA**

1. Tanto los documentos, como las consultas, son procesados como en el modelo vectorial hasta obtener el w_{ij} y el w_{iq}.
2. Se calcula la transpuesta de w_{ij}.

3. Se usa el método **svd** de **numpy.linprog** para fraccionar la matriz w_{ij} en:
    a. **S**: Matriz de valores propios.
    b. **VT**: Matriz de vectores propios ortogonales correspondiente a los documentos.
    c. **U**: Matriz de vectores propios ortogonales correspondiente a los términos.
4. Se reduce la dimensión de las matrices hasta obtener el tamaño "**k**" obtenido al comienzo de la ejecución.
5. Se calcula la transpuesta de **VT**, de donde cada una de sus columnas pasan a ser los vectores de los documentos (q).
6. Se calcula la inversa de **S** y la transpuesta de **U** para multiplicarlos con $\boldsymbol{w_{iq}}$
7. Y así obtener el vector de la consulta ($\boldsymbol{q = S_k^{-1} * U^T * w_{iq}}$).
8. Se calcula la función de similitud de la consulta para cada documento (análogo al modelo vectorial).

- **Particularidades de cada modelo**

  - **Modelo booleano:**

1- Modos de Consulta: Se implementaron 2, el **casual** y el **experto**: El **casual** es para usuarios no versados en el álgebra booleana y que puedan escribir consultas en lenguaje natural. El **experto** es aquel que tiene al menos conocimientos básicos de lógica y puede escribir consultas con el formato de expresiones lógicas.

2- Tipo de coincidencia: Se implementaron 2 tipos, la coincidencia **parcial** y la **total**.
Esto solo afecta a las consultas de usuarios casuales, ya que el algoritmo al recibir una consulta en lenguaje natural es incapaz de detectar signos de agrupación.
Ej: A y B o C puede ser interpretado como A y (B o C) ó (A y B) o C lo cual sería muy ambiguo.
La propuesta para hacer una consulta más amigable es que el usuario decida si quiere encontrar coincidencias exactas de los términos (coincidencia total) y esto en programa sería poner operadores **AND** entre los pares de términos, o si desea coincidencias parciales, es decir, cualquier término de la consulta que aparezca en un documento, es parte de los documentos recuperados, esto se lleva a cabo poniendo operadores **OR** entre todos los pares de términos de la consulta.

Tanto al modelo **LSA** como al **Vectorial** hay que indicarle cual es el valor del **umbral**, este es la cota inferior por la que regirán nuestros modelos a la hora de calcular las relevancias.

  - **Modelo LSA:**

Antes de ejecutar la consulta hay que definir cuál es el valor de 'k', el cual es usado por este modelo para hacer la reducción de dimensión.

- **Crawling**

  Implementado basado en colas.
  Este proceso comienza con un conjunto de urls semilla predefinido y el tiempo (ingresado por el usuario) que se desea que dure el proceso y es el siguiente:

  1. Mientras que la cola no esté vacía o el tiempo de ejecución no exceda el tiempo límite.
  2. Se extrae la primera url de la cola.
  3. Obtenemos la información de la página correspondiente mediante el método **get** del módulo **requests** (ver nota en requirements.txt).
  4. Mediante el uso de expresiones regulares obtenemos todos los links de la página (formato http(s)://).
  5. Eliminamos todo el contenido innecesario (scripts, styles, spans, entre otros)
     y mediante el método **get_text** del módulo **BeautifulSoap** obtenemos el contenido de la página de forma visible y lista para formar parte del corpus de documentos.
  6. El contenido previamente obtenido es guardado en una carpeta llamada **caché** en un archivo .txt para optimizar futuras ejecuciones de esta función. Todos los .txt de dicha carpeta son los que conforman el corpus de documentos a la hora de realizar las consultas con los modelos implementados.
  7. Recorremos cada link obtenido en el paso (3) y si es válido lo añadimos a la cola y luego volvemos al paso (1).

- **Datasets:**

  En este módulo se han utilizado 3 bases de datos para probar la eficiencia de los modelos implementados tomando en cuenta los criterios de **precisión**, **recobrado** y más a fondo, usando la medida **F1**, estas bases de datos son:

  - **Cranfield**
    Es un pequeño corpus compuesto por 1400 resúmenes científicos, el cual además contiene 225 consultas en lenguaje natural, así como un tester con consultas y los documentos que ellos dan como relevantes.

  - **CISI**
    Los datos fueron recopilados por el Centro de Invenciones e Información Científica ("CISI") y consisten en datos de texto sobre 1.460 documentos y 112 consultas asociadas, así como un tester con consultas y los documentos que ellos dan como relevantes.

  - **Vaswani**
    Es un corpus compuesto por 11000 resúmenes científicos el cual además contiene 93 consultas en lenguaje natural, así como un tester con consultas y los documentos que ellos dan como relevantes.

Referencias:

- Expresiones booleanas con sympy:

  https://omz-software.com/pythonista/sympy/modules/logic.html

- b. Listar archivos de un directorio:

  https://j2logo.com/python/listar-directorioen-python/#:~:text=Para%20listar%20o%20recorrer%20un,archivos%20y%20carpetas%20que%20contiene.

- Uso de numpy: https://numpy.org/doc/stable/user/index.html#usr

- Procesamiento de texto con nltk: https://www.nltk.org/

- Expresiones regulares: https://docs.python.org/3/library/re.html

- Requests: https://requests.readthedocs.io/en/latest/

- Beautiful Soup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

- Trabajo con datasets: https://ir-datasets.com/python.html