

Proyecto Sistema de Recuperación de Información

Integrantes:

Luis Alejandro Lara Rojas C-512

Richard García De la Osa C-512

Carlos Alejandro Arrieta Montes de Oca C-512

Desarrollo y composición del proyecto

El proyecto está desarrollado en python y sigue los siguientes pasos:

- 1-Procesamiento del lenguaje.
- 2-Implementación de los modelos escogidos.
- 3- Implementacion de la interfaz visual.
- 4-Pruebas y evaluación de métricas.
- 5-Conclusiones.

Desarrollo de los modelos

En nuestro caso nos decidimos por el modelo vectorial al principio, que, a pesar de ser un poco más complicado de implementar que el modelo booleano, para uso general ha demostrado brindar mejores resultados, además de brindar un ranking. Luego implementamos el modelo booleano para comparar resultados en escenarios distintos.

No es objetivo de este trabajo explicar cómo funciona el modelo vectorial, pero estaremos explicando brevemente qué hicimos en el archivo **vectorial_model.py**, en el cual se encuentra toda la lógica del modelo. La clase **VectorialModel** se debe crear pasándole a su constructor un diccionario del mismo formato de **dataset.json** y esta contiene las funciones **tf** e **idf**, **calculateWQuery** que se encarga de calcular los pesos de una consulta dada, la función **sim** es la que calcula la similitud dada una consulta y un documento, etc.

La función **search** tiene como labor devolver dada una consulta los documentos relevantes, para ello por cada documento llama a la función **sim**, y considera devolver un documento siempre que su similitud con la consulta sea mayor o igual a un umbral al cual le dimos valor 0.6.

En caso del modelo booleano se utilizaron consultas de la forma "palabra" o "no palabra". En este caso en el archivo **boolean_model.py**, el metodo **boolean_retrieve** se encarga de dada la consulta revisar los documentos que se van a recuperar. En este caso se devuelven dos colecciones la primera resultado de aplicar estrictamente el modelo y la segunda aplicando la modificación sobre relajar la condición del modelo booleano.

Al igual que el modelo vectorial, recibe un diccionario con el formato del **dataset.json** y tiene una función **search** que es la que devuelve las dos colecciones con los documentos recuperados en el formato mencionado anteriormente.

Procesamiento de los documentos.

Nuestra primera tarea fue procesar los documentos y transformarlos en una estructura más facil de utilizar para nosotros, para ello transformamos el archivo cran.all.1400 a un archivo en formato .json que nos permitiera operar mejor sobre los documentos. Auxiliándonos de la biblioteca spacy eliminamos del texto elementos que no nos aportaran mucha información, tales como signos de puntuación, conjunciones, preposiciones, etc (stopwording y stemming).

Finalmente exportamos, luego del procesamiento, en el archivo dataset.json toda la información sobre los documentos, donde incluimos por cada documento un diccionario con las palabras y su frecuencia en cada texto

La implementación de todo este proceso se encuentra en el archivo **text_processing.py**, encapsulado en las siguientes funciones:

processing_text: función encargada dado un texto removerle los signos de puntuación y los stopwords.

read_docs: función encargada de leer el archivo cran.all.1400 y transformarlo en una estructura computacional (diccionario), en el camino procesaba el texto usando la función **processing_text** mencionada anteriormente.

dataset_preprocessing: esta funciona como un wrapper de las 2 anteriores, ya que se encarga de devolver el **dataset.json** previamente creado, o, en caso de este no existir crearlo haciendo todo el procesamiento anteriormente explicado, también tiene la opción de modificar el **dataset.json** en caso de se hayan modificado los datos, esta última pasándole el parámetro **override** en "True".

query_processing: básicamente su tarea es, utilizando **processing_text** procesar las queries para eliminar todos los elementos que nos resulten indiferentes, tales como signos de puntuación, etc.

Análisis de las métricas

Para medir la efectividad del modelo implementado se decidió utilizar las métricas **precisión**, **recobrado** y **F1**. En el módulo **evaluation.py** se tienen los métodos **recover_value** y **precision_value** que calculan para una consulta los valores anteriormente mencionados respectivamente. Como estos valores se calculan respecto a alguna de las consultas almacenadas, reciben como párametro el número de la consulta sobre la cual se quiere calcular la métrica. Luego, apoyándose en las funciones para calcular los conjuntos de Documentos Recuperados, Documentos Relevantes y Documentos Recuperados Relevantes devuelve el valor de la métrica correspondiente.

Además de estas funcionalidades se agregaron los métodos **recover_mean**, **precision_mean** y **f_mean**, que calculan el promedio de los valores de las métricas con respecto a todas las consultas almacenadas.

Luego de este análisis obtuvimos una media en la tasa de recobrado de un 68% y una pobre media en la precisión de un 7%, esto se debe a que en nuestro parámetros sacrificamos en precisión en pro del recobrado, ya que retornamos un documento si su similitud con la consulta es mayor o igual a 0.6, colocando un mayor valor en este parámetro obteníamos mejor precisión pero se veía afectado el recobrado, y decidimos por priorizar este último.

Modelo Vectorial

Cranfield

- Cantidad de documentos recuperados: 113
- Precisión: 0.0660901356010108
- Recobrado: 0.6740821865660946

Vaswani

- Cantidad de documentos recuperados: 28
- Precisión: 0.0005662856577880375
- Recobrado: 0.011045235787031453

En el caso del Boleano los valores para las metricas son bastante variables dado que este modelo suele o devolver muchos documentos o casi ninguno por lo que la precisión y el recobrado no tienen resultados muy esperados. Por esto decidimos "modificar" el modelo para que no solo devolviese estrictamente los documentos que cumpliesen el modelo booleano, sino cuantificar que tan cercanos a la consulta estaban y devolver acorde a esto. Al hacer este cambio hay una mejora en cuanto al recobrado pero la precisión sigue siendo muy mala. Podemos observar que en estos casos se recuperaron 0 documentos usando el modelo estrictamente.

Modelo Boleano

Cranfield

Recuperados estricto

- Cantidad de documentos recuperados: 0
- Precisión: 1
- Recobrado: 0

Recuperados por partes

- Cantidad de documentos recuperados: 593
- Precisión: 0.015112175391930341
- Recobrado: 0.8560052494050359

Vaswani

Recuperados estricto

- Cantidad de documentos recuperados: 0
- Precisión: 1
- Recobrado: 0

Recuperados por partes

- Cantidad de documentos recuperados: 971
- Precisión: 0.0006521159629718105
- Recobrado: 0.0926281150756899

Conclusiones

Nuestros modelos obtuvieron una buena tasa de recobrado, en nuestra opinión, pero no fue el caso con la tasa de precisión, para mejorar esto deberíamos probar con otros valores en los parámetros de los modelos, así como mejorar el procesamiento del texto usando otras bibliotecas y/o técnicas de machine learning. También utilizar otras técnicas de retroalimentación para mejorar los resultados y talvés usar otras métricas para ampliar el espectro de comparación entre ambos modelos y poder determinar con mayor seguridad en que casos sería mejor utilizar alguno de ellos.