Luis Alejandro Lara Rojas
Richard Garcia de la Osa
Carlos Alejandro Arrieta Montes de Oca

C512

Procesamiento de los documentos

Nuestra primera tarea fue procesar los documentos y transformarlos en una estructura más amigable para la computadora, 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.

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_prcessing: esta función 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: esta función aún no le damos uso en esta sección, pero 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

Desarrollo del modelo

En nuestro caso nos decidimos por el modelo vectorial, 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 ranking

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 query dada, la función *sim* es la que calcula la similitud dada una query y un documento, etc

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

Análisis de las métricas

Para medir la efectividad del modelo implementado se decidió utilizar las métricas *precisión* y *recobrado*.

En el módulo *evaluation.py* se tienen los métodos *recover_value* y *precision_value* que calculan para una consulta el recobrado y la precisión 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 Recobrados, Documentos Relevantes y Documentos Recuperados Relevantes devuelve el valor de la métrica correspondiente.

Además de estas funcionalidades se agregaron los métodos *recover_mean* y *precision_mean*, que calculan el recobrado y precisión promedio 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

Conclusiones

Nuestro modelo obtuvo una buena tasa de recobrado, en nuestra opinión, pero no fue el caso con la tasa de precisión, nos queda de tarea para la entrega final mejorar esta

métrica, para ello debemos probar con otros valores en los parámetros del modelo, así como mejorar el procesamiento del texto usando otras bibliotecas y/o técnicas de machine learning