Proyecto Final de SRI

Integrantes:

- Ernesto Alfonso Hernández C312
- Abraham González Rivero C312
- Jorge Alejandro Soler González C312

Link del proyecto: https://github.com/Greenman44/SRI-2022

Los modelos escogidos por nuestro equipo fueron:

- modelo vectorial
- modelo booleano
- modelo de semántica latente

La implementación se realizo en Python.

Elementos generales:

Las colecciones escogidas para probar nuestros modelos son las siguientes :

- Cranfield Dataset : http://ir.dcs.gla.ac.uk/resources/test_collections/cran/cran.tar.gz
- CISI Dataset : http://ir.dcs.gla.ac.uk/resources/test_collections/cisi/cisi.tar.gz
- Vaswani Dataset : http://ir.dcs.gla.ac.uk/resources/test_collections/npl/npl.tar.gz

Cada una de estas colecciones proveen 3 archivos que nos ofrecen un conjunto de documentos, un conjunto de querys y un conjunto de parejas (query,documento) que nos indican que el documento es relevante para la query.

Nuestro sistema requiere que las colecciones posean los formatos ofrecidos por las direcciones anteriormente mencionadas.

Cada una de estas colecciones es parseada a formato .json con las siguientes características:

El conjunto de los documentos se almacena en un .json donde cada documento se representa como un diccionario que posee al menos las llaves:

- id : el id del documento en cuestión.
- body: el texto del documento en cuestión.

El conjunto de las querys se almacena en un .json donde cada query se representa como un diccionario que posee al menos las llaves:

- query number : el id de la query en cuestión.
- query : la query en cuestión.

El conjunto de las relevancias se almacena en un .json donde cada query se representa como un diccionario que posee al menos las llaves:

- query_num : el id de la query.

- id : el id del documento.
- position : un valor entre 1 y 4 que representa la relevancia del documento para la query en cuestión. Este valor solo viene por defecto en la colección Cranfield, por lo tanto, se asume que en cualquier otra colección el documento tiene relevancia 4 para la query dada.

Preprocesamiento de los datos:

Según la colección escogida por el usuario, se utiliza el .json de los documentos para obtener una representación matricial de los documentos, que tiene dimensiones (cantidad de términos, cantidad de documentos) y su posición [i , j] contiene el tf del término i respecto al documento j.
Para esto nos apoyamos en el uso de:

- expresiones regulares: nos permite depurar el texto, es decir, eliminar información innecesaria(signos de puntuación, espacios en blanco, símbolos, entre otros).
- la biblioteca nltk que nos ofrece herramientas para el procesamiento del lenguaje natural, stopwords para eliminar preposiciones y otras palabras que carecen de relevancia semántica, stemmers para obtener la precedencia de los términos una vez filtrados(notar que hacer esto mejora la eficiencia ya que elimina redundancia entre términos con procedencia similar) y word_tokenizers para tokenizar los términos restantes.

Luego de filtrar y tokenizar, se obtiene la matriz de tf y una representación del vocabulario(el conjuntos de todos los términos) en forma de diccionario donde las llaves son los términos y los valores son tuplas (índice al que apunta el término en la matriz, idf del término, frecuencia global del término). Esta representación se encapsula en una clase dataset que sirve como interfaz de comunicación entre los modelos y los documentos.

Este proceso se realiza en el archivo "process_data.py" y la implementación de la clase en el archivo "datasets.py".

Preprocesamiento de las querys:

Las querys se consideran documentos, lo que implica que su preprocesamiento es similar al anteriormente descrito, a excepción del modelo booleano, esto se tratará en el apartado dedicado a este modelo.

Representación de los modelos:

Todos los modelos se representan a través de una clase, que heredan de una clase abstracta, que cuentan con una instancia de dataset(representación de la colección escogida), la clase abstracta exige la implementación de `EvalQuery`(forma de procesar y evaluar la query) y si lo requiere una forma de representar los pesos de los términos respecto a los documentos en el método `_weightupterms`. Para esto cada modelo debe tener su propio procesador de querys. Los procesadores de querys heredan de QueryProcessor, que exige la implementación del método `processquery` y en caso de requerirlo, una implementación del método `weightupquery` que asigna pesos a los términos de la query.

Análisis del diseño del modelo vectorial:

Se decidió escoger este modelo debido a su excelente rendimiento en colecciones de temáticas variadas. Es el de mejores resultados de los modelos clásicos. Además, Python posee bibliotecas como numpy que facilitan enormemente el trabajo con vectores y sus respectivas operaciones algebraicas.

Representación vectorial de los documentos:

La instancia de dataset nos facilita obtener la matriz de tf de los términos y el diccionario que representa el vocabulario que nos permite acceder eficientemente al idf de cada término, permitiendo de los pesos de los términos. Luego, para mejorar la eficiencia del sistema, se inicializa el modelo con la matriz de pesos de la colección de documentos.

Representación vectorial de la query:

Luego de preprocesar la query, obteniendo su matriz de tf, se calcula su peso con la fórmula estándar con a = 0.4 para tener así su representación vectorial.

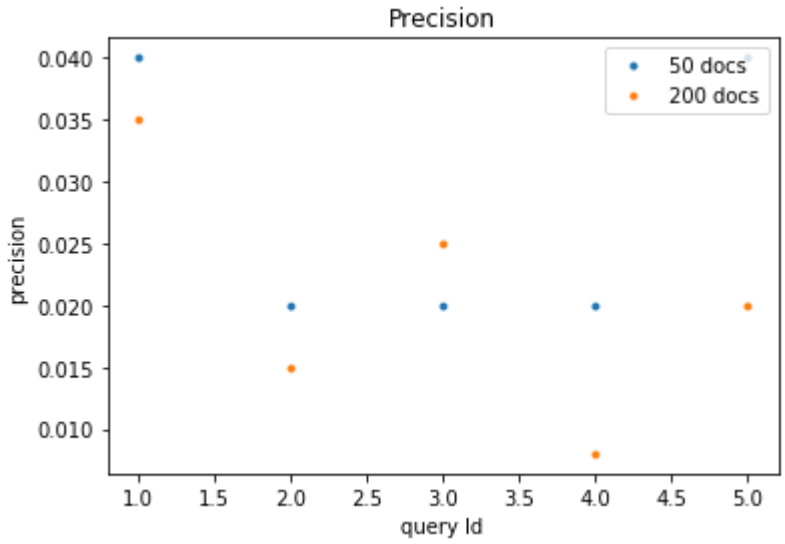
Evaluación de la query:

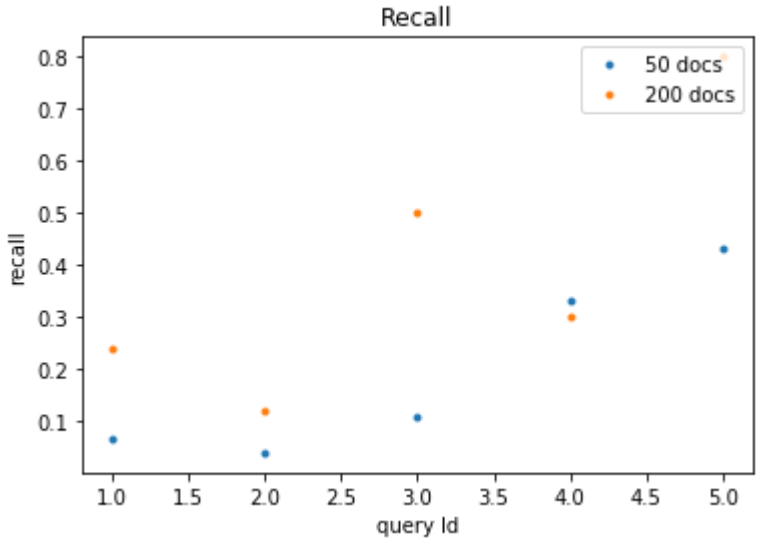
Utilizando numpy se calcula el coseno entre la query y los documentos para obtener la similitud. Los resultados se ordenan y se devuelve un ranking con los primeros 50 documentos.

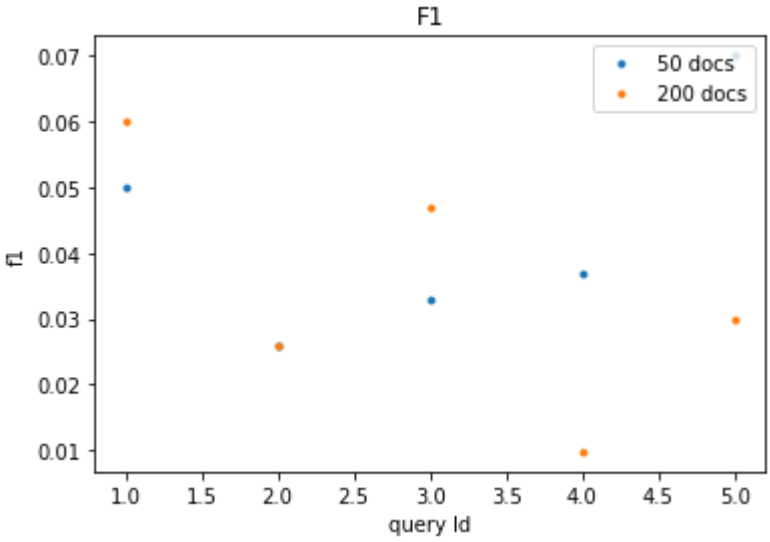
Evaluación de los resultados:

A continuación se muestran la precisión, recobrado y medida f1 de nuestro modelo para las colecciones de prueba. Se evaluaron las primeras 5 querys de cada colección

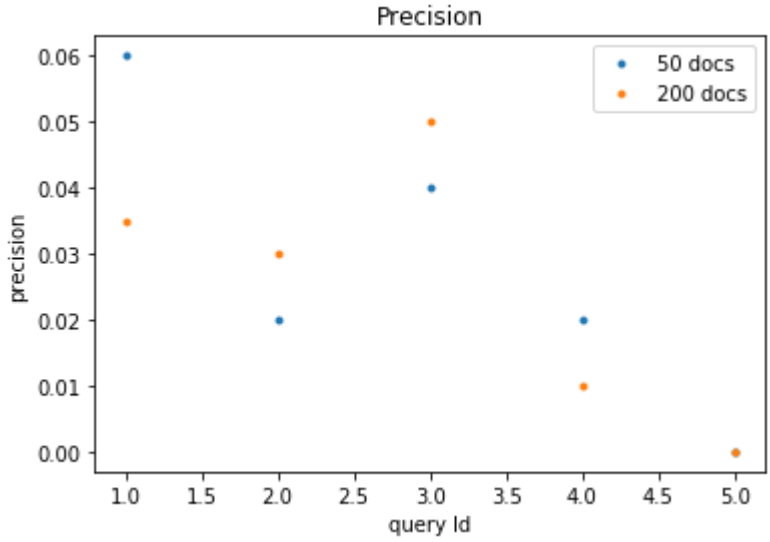
Cranfield Dataset:

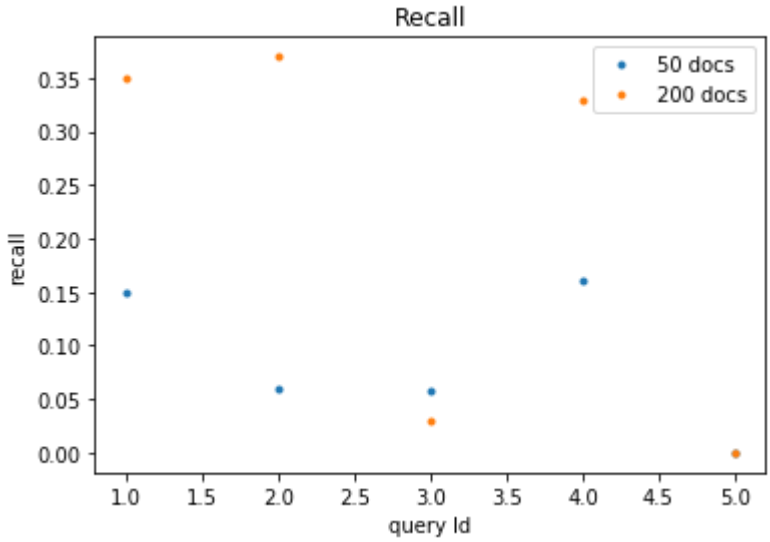


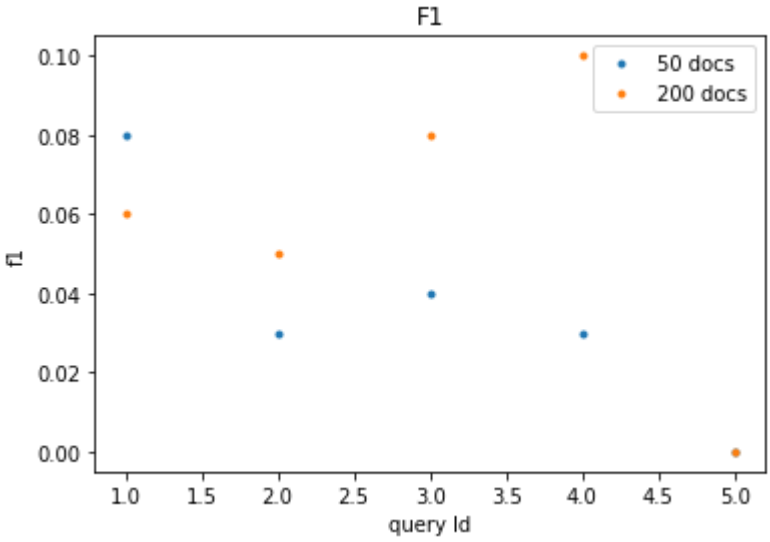




Vaswani Dataset







Análisis del diseño del modelo booleano:

Escogimos este modelo debido a su fácil implementación. Además, Python nos ofrece la biblioteca boolean para parsear expresiones booleanas y la biblioteca simpy para evaluarlas.

Representación de los documentos:

Usando la matriz de tf obtenida del preprocesado, adquirimos el vector binario teniendo en cuenta que si tf es mayor que 0 entonces el término aparece en el documento.

Preprocesamiento de la query:

La query es una expresión booleana con los operadores de negación, conjunción y disyunción. Las variables de la expresión son los términos. Se parsea la query con la función `BooleanAlgebra().parse(query)` de la biblioteca boolean y se convierte a una expresión de simpy con el método `sympify` para su posterior evaluación.

Evaluación de la query:

Se evalúa en la query la representación vectorial binaria de cada documento, esto sería un diccionario donde las llaves son las variables y los valores son los valores que tomaran esas variables. Para esto se utiliza la función `simpy.sub(simpifyObject, dict)`.

Análisis del modelo LSI:

Escogimos por depender del modelo vectorial, lo que nos permite reutilizar herramientas ya implementadas. Además, Python facilita las operaciones complejas con matrices y algebraicas que este modelo conlleva, a través de la biblioteca numpy.

Representación de los documentos:

Tras preprocesar los documentos obtenemos la matriz de tf y en el diccionario del vocabulario tenemos la frecuencia global del término. Con esto, podemos calcular el peso de los términos en cada documento, utilizando la métrica:

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

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

Con la matriz de los pesos se aplica svd(T,S,D) utilizando la función `np.linalg.svd(M)` de la biblioteca numpy. Luego se reducen las dimensiones de las matrices resultantes, el 70% en nuestro caso, obteniendo una representación vectorial de los documentos en el nuevo espacio.

Representación de la query:

Con las herramientas del vectorial se obtiene una representación vectorial de la query. Esta se proyecta en el nuevo espacio de los documentos, para esto se utiliza la fórmula:

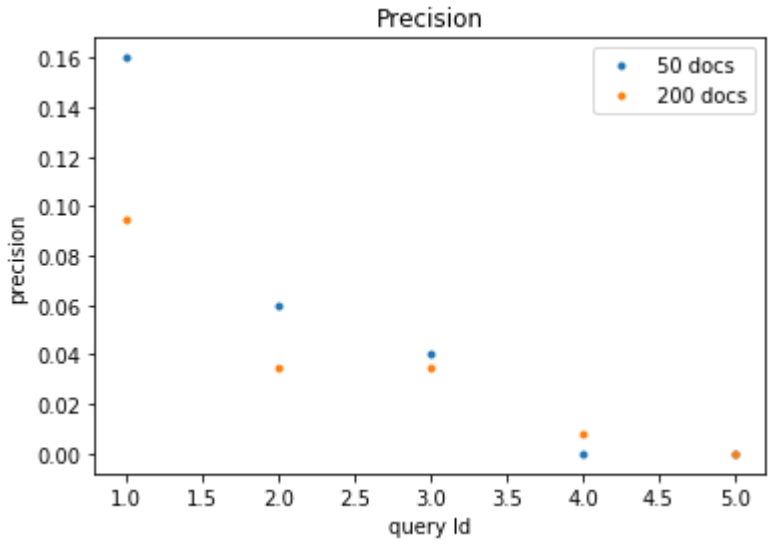
$$q_k = S_k^{-1} * T_k^T * q$$

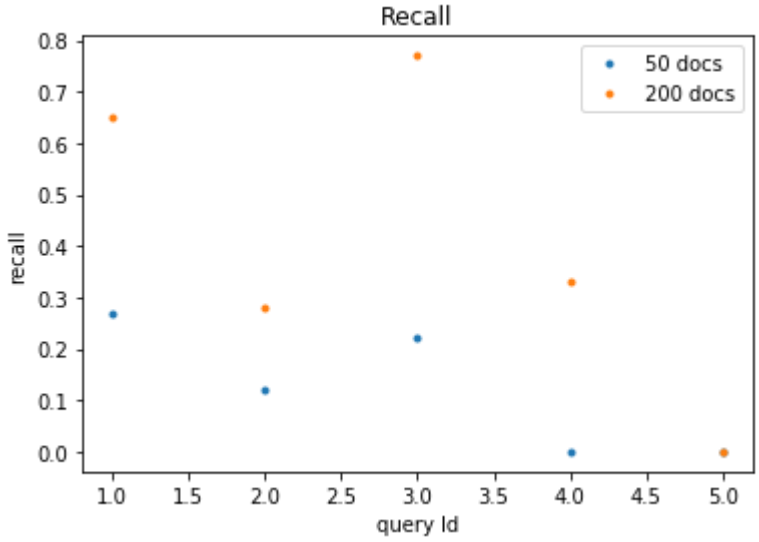
Evaluación de la query:

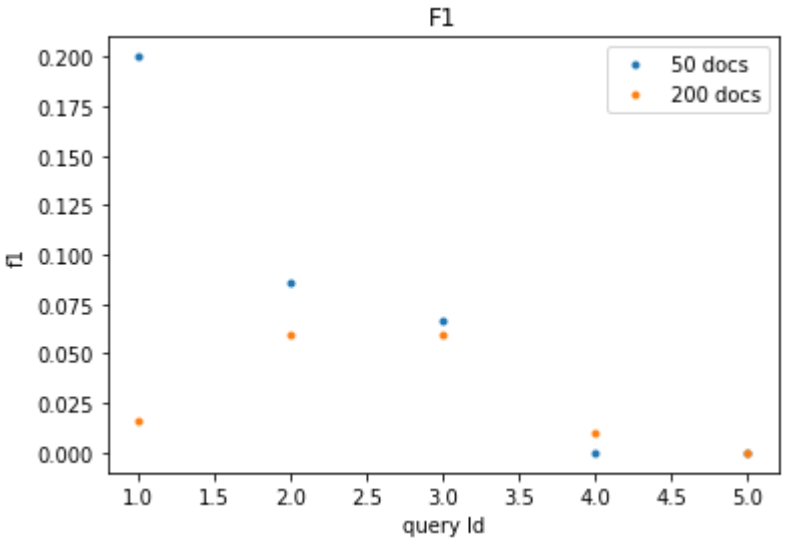
Se obtiene la similitud con el documento de la misma forma que en el vectorial, calculando el coseno entre el vector de la query y el documento, se ordenan y se devuelve un ranking con los primeros 50 documentos.

Evaluación de los resultados:

Cranfield Dataset:







Ventajas:

- El modelo vectorial es eficiente y de buenos resultados.
- La interfaz de usuario facilita la utilización de la aplicación por parte del cliente, al igual que la visualización de los resultados.

Desventajas:

- El modelo LSI es muy ineficiente para colecciones grandes.
- El proceso de parseo de colecciones desde la interfaz de usuario se demora mucho.
- El modelo booleano no es muy útil en colecciones de temáticas variadas.

A mejorar en el futuro:

- La interfaz de usuario puede mejorarse para facilitar el uso de la aplicación.
- Optimizar la implementación del modelo LSI y del proceso de parsing para colecciones muy grandes.

Para ejecutar el proyecto:

- Instalar en un entorno virtual de python(version > 3.9 preferiblemente) las dependencias, a través del comando de consola:
  `pip install -r requirements.txt`
- Ejecutar el proyecto a través del comando de consola:
  `python app_test.py`
- Jugar con el visual.