

Proyecto Final Sistemas de Recuperación de Información

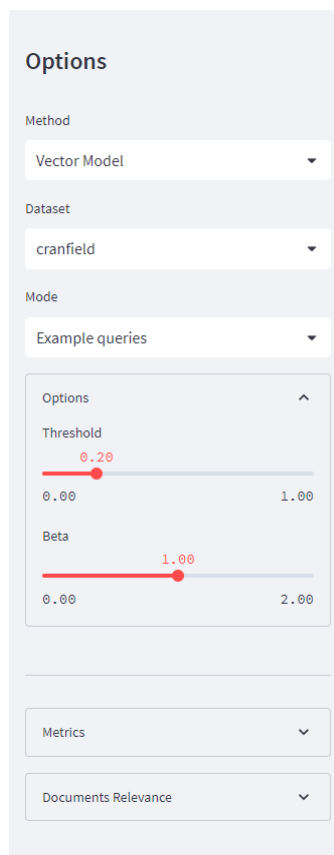
Lauren Guerra Hernández Dennis Fiallo Muñoz Paula Rodríguez Pérez

21 de diciembre de 2022

Presentación Visual

La interfaz visual se implementa usando las facilidades del módulo de Python **streamlit**. De esta se presentan a continuación sus funcionalidades.

Presenta a su izquierda una barra de opciones como se ve en la imagen.



Aparece primeramente un **selectbox** para alternar entre los modelos de recuperación de información presentados.

Luego aparece otro **selectbox** para seleccionar el set de datos del cual se quiere hacer la consulta.

El tercer **selectbox**(Mode) nos brinda la opción de alternar entre diferentes modos para usar la aplicación. Con la entrada clásica de texto, entrada de consultas de ejemplo y un modo especial para calcular las métricas del modelo seleccionado en un **dataset**.

Más abajo se presenta una ventana desplegable con las opciones para la recuperación.

Bajo la línea va a estar una ventana desplegable que cuando se tengan los resultados de la consulta nos dará los resultados de las métricas.

Mientras que más abajo se podrán ver de la consulta actual los documentos que son realmente relevantes, cuan relevantes con en realidad además de decir si este fue recuperado por el modelo seleccionado o no.

Mientras, del lado derecho de esta barra se mostrará la selección de consulta y los resultados de esta en ventanas desplegables en las cuales se da es título del documento, su contenido y su similitud con la consulta.

Options

Method

Vector Model

Dataset

cranfield

Mode

Example queries

Options

Metrics

P

0.104

R

0.414

F

0.167

F1

0.167

Execution Time

1.852 s

Documents Relevance

Search

1: what similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft .

Search

0 1 2 3 4 5 6 7 8 9 10 11

Document: 1141

Document: 359

Document: 154

Document: 56

Document: 1368

Document: 68

Document: 13

Document: 979

Document: 327

Document: 1186

Modelo Base

La implementación de los tres sistemas que presentamos van a seguir los mismos pasos en la recuperación de información pues heredan de la clase `base Model`. Por tanto, los sistemas van a seguir las etapas que se presentan a continuación:

- **Etapas 1:** Verificar si ya se tienen almacenada la información recuperada del set de datos actual, en caso de que no se hayan recuperado los datos de estos documentos con anterioridad se pasa a la Etapa 2 para extraer los datos, sin embargo, si ya se tienen los datos recuperados se pasa a la Etapa 5 para lograr mayor velocidad al mostrar los resultados ya que se cuenta con la información necesaria de los documentos.
- **Etapas 2:** Tomar los documentos del set de datos.
- **Etapas 3:** Separar los términos de cada documento y eliminar los signos de puntuación y las *stopwords*, luego se *lexemizan* los términos para quedarnos con su forma canónica. Después de tener la lista de términos se pasa a aplicar la expansión de consulta buscando sinónimos del término y agregándolo a la consulta.
- **Etapas 4:** Calcular y almacenar la relación entre término con los documentos en que aparece (cada modelo define su modo de hacer esta etapa).
- **Etapas 5:** Calcular y almacenar los pesos de las consultas en los documentos del siguiente modo: término: w, ...
- **Etapas 6:** Calcular la similitud de la consulta con los documentos usando los datos recopilados anteriormente.
- **Etapas 7:** Devolver *ranking* de documentos.

Implementación del código y herramientas usadas

El código va a tener separado en diferentes archivos las clases principales de la ejecución que serían:

Model: Este se encuentra en `base_model.py` y será la base de todos los modelos de recuperación de información que se presentan.

Datasets: Aparece en `dataset.py`, este se va a inicializar desde **Model**. Se encarga de extraer la información de los **datasets**; desde los términos en los documentos, hasta las consultas de ejemplo y la relevancia real de los documentos con respecto a las consultas. Además, tiene un conjunto de propiedades y métodos estáticos para tomar datos necesarios en diferentes formas según el uso que se le va a dar. Se apoya en el módulo de python `ir_datasets` para facilitar el acceso a la información de los **datasets** de ejemplo.

Lexemizer: Este se encuentra en `lexemizer.py`. Apoyándose de las facilidades del módulo `nltk`, este está destinado a normalizar los términos de los documentos y de las consultas; separando las palabras en *tokens* independientes, eliminando las *stopwords*, eliminando las mayúsculas y *lematizando* los términos, y para el caso de las consultas también se le aplica una expansión de consulta usando `wordnet` para agregar sinónimos de los términos.

Visual: Se encuentra en `visual.py` y se encarga de llevar todo el proceso visual del proyecto.

Evaluación

Para todos los modelos es posible evaluar los resultados obtenidos por cada consulta con las siguientes métricas:

$$P = \frac{|RR|}{|RR \cup RI|} \quad (1)$$

$$R = \frac{|RR|}{|RR \cup NR|} \quad (2)$$

$$F = \frac{1 + \beta^2}{\frac{1}{P} + \frac{\beta^2}{R}} \quad (3)$$

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} \quad (4)$$

Además de medir el tiempo de ejecución del modelo. También es posible evaluar todo el conjunto de consultas del set de datos y calcular las métricas para la unión de los resultados. Esto es posible en el proyecto en la opción `All queries` de **Mode**.

Modelo Vectorial

Como ya vimos anteriormente, la implementación del Modelo Vectorial va a heredar de la clase **Model**, siguiendo pasos similares al resto de modelos. En un principio, al inicializar el modelo se crean 3 `dict` vacíos, los cuales van a ser usados más adelante, siendo estos:

doctterms: un diccionario que tendrá como llaves los términos de todos los documentos, teniendo como **valid** almacenado cada uno los documentos en que aparecen y de estos la frecuencia, tf , idf y w . Se usa esto pues la mayor parte de los términos aparecen en una pequeña parte de los documentos, por tanto, al eliminar los elementos que no son necesarios se puede lograr ahorrar algo de tiempo de procesamiento.

queryterms: diccionario de términos de la *query* con su respectivo peso.

quersim: diccionario de documentos que tienen como valor su similitud con la consulta.

Luego se siguen una serie de etapas que en general se siguen en todos los modelos. Se comienza verificando si ya se encuentran recopilados los datos de los términos de los documentos, en este caso en **doctterms**, si se encuentra vacío se extrae la información apoyándose en la clase **Datasets**. Para los cálculos de estos datos se usan las formulas clásicas de:

$$tf_{ij} = \frac{freq_{ij}}{max_l freq_{lj}} \quad (5)$$

$$idf_i = \log \frac{N}{n_i} \quad (6)$$

$$w_{ij} = tf_{ij} \cdot idf_i \quad (7)$$

Mientras que para la consulta se aplica un suavizado variable con valor predeterminado de 0.5:

$$w_{ij} = \left(\alpha + (1 - \alpha) \frac{freq_{iq}}{max_l freq_{lq}} \right) \log \frac{N}{n_i} \quad (8)$$

De estos se tiene que:

- $freq_{ij}$ sería la frecuencia del término t_i en el documento d_j .
- Se calcula el máximo de los documentos.
- Si no aparece el término t_i en el documento d_j entonces $tf_{ij} = 0$.
- N es la cantidad de documentos en el sistema.
- n_i es es la cantidad de documentos en que aparece el término t_i .

Luego de tener los datos de los términos en los documentos y de la E se pasa a calcular la similitud entre estos usando:

$$sim(d_j, q) = \frac{\sum_{i=1}^n (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^n w_{ij}^2} \sqrt{\sum_{j=1}^n w_{jq}^2}} \quad (9)$$

Luego se devuelve un *ranking* organizado por similitud y eliminando los que sean cero, o en caso de existir un umbral los que estén por encima de este.

Evaluación

Para el *set* de datos *Cranfield*, al calcular las métricas para todas las consultas tuvimos los siguientes resultados: Un tiempo de ejecución promedio de unos 0.115 segundos por consulta. Según la medida *F1* se logra una mejor calificación, de 0,161 para un umbral de 0,4 con *Precisión* = 0,13 y *Recobrado* = 0,213, probando desde umbral 0,1 hasta 0,4 ganando 10 veces más precisión pero pasando de recobrado 0,87 a 0,213.

Para el caso de del set de datos *Vaswani* se hicieron pruebas de umbral 0,2 a 0,45 donde se pudo notar un gran aumento en precisión en umbral 0,4, mientras que a partir de este punto era mucho mayor la pérdida de documentos que la precisión que aumentaba.

Por tanto se decidió usar para este caso umbral 0,4, donde se logra un *Precisión* = 0,043 y *Recobrado* = 0,253 para una calificación de *F1* = 0,073 con un tiempo de ejecución promedio de 0,75 segundos.

Ventajas

- Tiene en cuenta el peso de *tf* - *idf* para la representación de los documentos.
- Modelo eficiente para generar *rankings* en colecciones de gran tamaño.

Desventajas

- Necesita de la intersección de los términos de la consulta con los documentos, en caso contrario no se produce la recuperación de información.
- Asume que los términos son independientes entre ellos.

Modelo Probabilístico

La implementación del Modelo Probabilístico hereda de la clase `Model`. En la inicialización del modelo se tendrán 3 diccionarios (`dict`) y un conjunto (`set`):

term_docs: diccionario para almacenar los términos de la colección y su respectiva lista de documentos en los que aparece.

queryterms: diccionario para almacenar los términos de la consulta.

query_doc_sim: diccionario para almacenar por cada documento de la colección su similitud con la consulta.

term_p_r: diccionario para almacenar las probabilidades de ocurrencia de un término en un documento relevante o no relevante a la consulta. Estas probabilidades serán almacenadas en una tupla donde el primer valor corresponde a la probabilidad de ocurrencia del término en un documento relevante a la consulta y en la segunda posición la probabilidad de que ocurra en un documento no relevante a la consulta.

Al igual que en los otros modelos implementados se verifica si ya se analizó la ocurrencia de los términos de la colección en los documentos, en este caso en **term_docs** es donde se almacena dicha información, si se encuentra vacío se extrae la información apoyándose en la clase `Datasets`.

Para la implementación de nuestro modelo inicialmente la probabilidad de ocurrencia de un término en un documento relevante será tomada como una constante (0.5) y la probabilidad de ocurrencia de un término en un documento no relevante como:

$$r_i = \frac{n_i}{N} \quad (10)$$

Donde:

- N es la cantidad de documentos en la colección.
- n_i es la cantidad de documentos en que aparece el término t_i .

Posteriormente con estos valores de probabilidades iniciales se procede a calcular la similitud entre un documento y la consulta a través de la siguiente fórmula:

$$sim(d_j, q) = \sum_{i=1}^n \left(w_{i,j} \cdot w_{i,q} \cdot \log \frac{p_i(1 - r_i)}{(1 - p_i)r_i} \right) \quad (11)$$

Donde:

- $w_{i,j}, w_{i,q}$ son la ocurrencia del término i en el documento j y la consulta respectivamente.
- p_i, r_i serán las probabilidades de que el término i ocurra en un término relevante o no relevante respectivamente.

Una vez obtenidos estos valores de similitud, estos se normalizarán y se presentará un *ranking* con los mismos.

Luego se procederá a realizar una pseudo-retroalimentación y para ello los valores de p_i, r_i serán calculados de la siguiente forma:

$$p_i = \frac{|V_i| + 0,5}{|V| + 1} \quad (12)$$

$$r_i = \frac{n_i - |V_i| + 0,5}{N - |V| + 1} \quad (13)$$

Donde:

- V es el conjunto de documentos recuperados.
- V_i es el conjunto de los documentos recuperados en los que aparece el término i .

Serán aplicadas constantes de suavizado en cada una de las fórmulas (0.5 en el numerador y 1 en el denominador) para evitar división por cero o logaritmos indeterminados.

Una vez obtenidos los nuevos valores de p_i, r_i la similitud será computada nuevamente y se retornará un *ranking* en base a esta.

Se realizarán dos iteraciones donde se calculan nuevamente las probabilidades de ocurrencia de los términos en documentos relevantes o no, se analiza la similitud y se devuelve un *ranking*. El último *ranking* calculado será retornado como respuesta del algoritmo y se tendrá en cuenta para este si es establecido un umbral para la similitud.

Evaluación

Se analizaron todas las consultas con los valores de umbral: 0.2, 0.3 y 0.4.

Para el *set* de datos *Cranfield* se obtuvo un promedio de tiempo de ejecución de 0.419s por consulta con el valor del umbral igual a 0.2. En cuanto a las métricas objetivas se obtuvieron los siguientes valores: medida F1 igual 0.03, precisión igual 0.016 y recobrado igual a 0.196, este último el mayor alcanzado en las pruebas.

Para el umbral igual a 0.3 con tiempo de ejecución promedio de 0.338s obtuvimos como medida *F1*: 0.031, mientras que de precisión y recobrado valores iguales a 0.019 y 0.083 respectivamente.

En el caso del umbral igual a 0.4 con tiempo de ejecución promedio de 0.281s, de precisión se obtuvo: 0.02 (mejor valor obtenido), de recobrado un 0.037 y medida *F1* un 0.026.

Para el *set* de datos *Vaswani* se obtuvo un promedio de tiempo de ejecución de 3.947s por consulta con umbral igual a 0.2. En cuanto a las métricas objetivas se obtuvieron los siguientes valores: medida F1 igual 0.034, precisión igual 0.018 y recobrado igual a 0.2. En cuanto a la medida *F1* y el recobrado fueron los mejores valores logrados.

Para el umbral igual a 0.3 con tiempo de ejecución promedio de 1.314s obtuvimos medida *F1*: 0.031, mientras que de precisión y recobrado valores iguales a 0.019 y 0.073 respectivamente.

En el caso del umbral igual a 0.4 con tiempo de ejecución promedio de 0.773s, fue con el que mayor valor de precisión se obtuvo: 0.02, el recobrado fue igual a 0.036 y la medida *F1* resultante fue de 0.026.

Ventajas

- Devuelve un *ranking* de los documentos.
- No requiere mucha memoria para su procesamiento.

Desventajas

- Asume la independencia entre términos.
- No considera la frecuencia de los términos en los documentos.

Modelo Indexación Semántica Latente (LSI)

El modelo de Indexación Semántica Latente es un modelo de recuperación de información derivado del modelo vectorial. Por lo que en nuestra implementación la clase que define este modelo hereda de la clase que define al modelo vectorial y sigue todas las etapas que este sigue.

La ISL se basa en el principio de que las palabras que se utilizan en el mismo contexto tienden a tener significados similares. La característica fundamental de la ISL es su habilidad para extraer el contenido conceptual de un documento, estableciendo asociaciones entre aquellos términos que ocurran en contextos similares.

Para el análisis ISL primero se construye una matriz A_k donde las filas representan los términos y las columnas los documentos, esta matriz establece las relaciones término documento por lo que cada elemento x_{ij} representa el peso del término i en el documento j . Estos pesos en

este caso son el número de ocurrencias del término i en el documento j y están contenidos en la matriz \mathbf{A} que se calcula en la clase `dataset`.

El objetivo fundamental de LSI es encontrar una matriz A_k que constituya una aproximación a la matriz Términos-Documentos A . En esa aproximación se va a obtener información que no estaba disponible directamente en la matriz A , sino que se encontraba latente en esta. Esta matriz debe cumplir que el rango debe ser al menos k , donde k es mucho menor que el rango de A . En este caso decimos que A_k es una aproximación de rango bajo.

La descomposición en valores singulares (SVD) puede ser usada para resolver el problema de la matriz de aproximación de rango bajo. Para esto se realiza el siguiente procedimiento:

Hallar la SVD de la matriz Términos-Documentos. En otras palabras, siendo $A \in R^{m \times n}$ y rango r , se obtiene como resultado $A = TSD^T$, donde:

- $T \in R^{m \times m}$ es una matriz cuyas columnas son vectores propios ortogonales de AA^T . Representa los términos en el espacio de términos.
- S es la matriz diagonal de los valores propios
- $D^T \in R^{n \times n}$ es una matriz cuyas columnas son vectores propios ortogonales de $A^T A$. Representa los documentos en el espacio de documentos.

Esto se realiza en nuestro código obteniendo las matrices \mathbf{T} , \mathbf{S} , \mathbf{DT} que son la descomposición en valores propios de \mathbf{A} obtenidos a partir de la función `linalg.svd(A)` implementada en la librería `numpy`.

Luego se realiza el proceso de truncar los $r - k$ menores valores singulares de S ayuda a generar una aproximación $rango - k$ de menor error, ya que A_k queda de la forma:

$$A_k = TS_k D^T$$

$$A_k = T \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & \dots \\ 0 & \sigma_2 & 0 & 0 & \dots \\ 0 & 0 & \dots & 0 & \dots \\ 0 & 0 & 0 & \sigma_k & \dots \\ 0 & 0 & 0 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} D^T \quad (14)$$

$$A_k = \sum_{i=1}^k \sigma_i \vec{u}_i \vec{v}_i^T$$

Donde \vec{u}_i y \vec{v}_i son las i -ésimas columnas de T y D , respectivamente. Por tanto, $\vec{u}_i \vec{v}_i^T$ es una matriz rango-1, y A_k queda expresada como la suma de k matrices rango-1, donde el coeficiente de cada una es un valor singular.

En nuestro código guardamos como parte de la clase LSI las matrices \mathbf{Tk} , \mathbf{Sk} , \mathbf{DTk} donde ya se realizó el truncamiento de tamaño k . Estamos asumiendo un k de tamaño 200 para colecciones de documentos grandes, en caso de colecciones más pequeñas se asume un $k = \min(|docs|, |terms|)/5 + 1$.

Para trabajar con la *query* esta se lleva a un vector del tamaño de la cantidad total de términos en los documentos, con 0 en los lugares en que ese término no se encuentre en la consulta y luego se realiza una transformación del vector de consulta \vec{q} a su representación en el espacio LSI mediante: $\vec{q}_k = \vec{q}^T T_k S_k^{-1}$. Esto se computó utilizando las funciones de multiplicación de matrices, cálculo de la inversa y la transpuesta implementadas en la librería `numpy`.

Para hallar la similitud término-documento se utiliza, al igual que en el modelo vectorial, el cálculo del coseno del ángulo entre los vectores de los documentos y el vector de la *query* \vec{d}_j y \vec{q}_k respectivamente, tomando como cada vector \vec{d}_j las columnas de **DTk**.

Los resultados de este modelo son generalmente superiores a los que tienen en cuenta solamente la frecuencia de términos. Sin embargo, es necesario un mayor desarrollo para que este enfoque sea lo suficientemente rápido para su uso en grandes sistemas de recuperación.

Evaluación

Para el set de datos *Cranfield*, al calcular las métricas para todas las consultas tuvimos los siguientes resultados:

- Un tiempo de ejecución promedio de unos 1394 segundos por consulta contando el tiempo que se toma hacer la descomposición SVD inicial.
- Se hicieron pruebas para un umbral de 0.2 y 0.4 obteniéndose : $Precisión_{0,2} = 0,07$, $Recobrado_{0,2} = 0,41$, $Fmedida_{0,2} = 0,12$, $Precisión_{0,4} = 0,214$, $Recobrado_{0,4} = 0,104$, $Fmedida_{0,4} = 0,14$ donde se puede notar que en dependencia del umbral que se utilice existe una diferencia notable entre los valores de precisión y recobrado, pero en el caso de la medida F no existe tan grande diferencia, por lo que por ella se puede estimar con menor error la calidad de la recuperación.

Para el set de datos *Vaswani*, al calcular las métricas para todas las consultas tuvimos los siguientes resultados:

- En este caso la demora de la ejecución de la recuperación es considerablemente alta debido a la gran cantidad de documentos que tiene este *dataset* unido a la complejidad temporal del modelo, tomando un tiempo estimado de 14 425s por consulta.
- Tomando los umbrales 0.2 y 0.4 se obtuvieron los siguientes resultados $Precisión_{0,2} = 0,029$, $Recobrado_{0,2} = 0,528$, $Fmedida_{0,2} = 0,054$, $Precisión_{0,4} = 0,079$, $Recobrado_{0,4} = 0,186$, $Fmedida_{0,4} = 0,112$ donde además de la gran diferencia entre las precisiones calculadas para los respectivos umbrales también la medida F indica que para esta colección grande es mejor exigir una mayor similitud entre los documentos y las consultas utilizando el valor de umbral 0.4 para maximizar esta medida sin que represente una disminución significativa del recobrado.

Ventajas

- Una gran ventaja de LSI es que es un enfoque estrictamente matemático, sin conocimiento del significado de los documentos o palabras que analiza. Esto lo hace una técnica poderosa y genérica capaz de indexar cualquier colección coherente de documentos en cualquier lenguaje.

- El modelo tradicional del modelo vectorial se asume independencia de términos. Mientras que en LSI si se trabaja con dependencia entre los términos y más que eso, la mayor ventaja de LSI es la sinonimia y la polisemia.
- La solución LSI solo necesita construirse una única vez para toda la colección y el rendimiento en el momento de la recuperación no sufre.
- En el espacio semántico latente, una consulta y un documento pueden tener una alta similitud incluso sin términos comunes, siempre que sean semánticamente similares.

Desventajas

- El costo espacial y temporal es muy alto comparado con otras técnicas como en este caso el modelo probabilístico y el vectorial, donde LSI ocupa mucho más espacio en memoria y demora más tiempo en devolver el *ranking* que estos modelos. De esta manera resulta imposible aplicar el modelo a grandes colecciones de elementos, aunque es cierto que con procesadores de alta velocidad y de gran memoria disminuiría un poco este problema.
- El valor óptimo de la nueva dimensión a utilizar, nuestro k , es otro problema, pues para colecciones de más o menos documentos puede ser mejor la utilización de un k que de otro.