Informe Proyecto SRI
Querch

Lázaro Daniel González Martínez C311
Francisco Octavio Ayra Cáceres C312

1 Introducción

El presente informe contiene una breve explicación del proyecto final orientado en la asignatura "Recuperación de Información". De forma general consiste en la implementación de un sistema de recuperación donde el usuario pueda interactuar mediante consultas y tal vez retroalimentación. Se implementaron los modelos Booleano, Vectorial e Indexación Semántica Latente . A continuación se explicará la implementación técnica del mismo, se indicarán los archivos donde se encuentren las principales implementaciones y se describirá la arquitectura concebida para el sistema completo.

2 Arquitectura

Para implementar el Sistema de Recuperación de Información se consideró dividir la recuperación en 2 procesos: Preprocesado de documentos, y Modelación del sistema. Cada fase funciona de manera independiente, aunque en la práctica se haga un Preprocesado primero y luego el Modelado. Sin embargo, se consideran independientes porque ninguno depende técnicamente del otro.

La fase de Preproceso se encarga de extraer datos de los conjuntos de documentos y crear estructuras con estos, que se almacenarán en la memoria física del sistema. Este proceso suele ejecutarse una sola vez teniendo el corpus entero, o se ejecuta sobre cada subconjunto de documentos que se agregue al corpus. Entonces no solo se considera el espacio del corpus, sino también el espacio de datos extraídos propio de un modelo particular. Esto posibilita, que no haya necesidad de estar extrayendo los datos[1] que necesita el Modelo, durante la petición de la consulta.

En el Modelado, el sistema de búsqueda carga los datos extraídos del Preprocesado y pone a funcionar el modelo, que tendrá los datos cargados en memoria RAM. Notemos que son los datos extraídos y almacenados en la memoria física; por esto es que se consideran procesos independientes, ya que teniendo los datos extraídos, podemos ejecutar el Modelo sin tener que hacerle un preprocesamiento primero a los documentos. Durante esta fase, se espera constantemente recibir una consulta, para luego pasarla al modelo, y obtener los resultados de la recuperación de información.

3 Preprocesado del corpus

Para una mayor comodidad se trabajó con el conjunto de datos que proporciona el módulo `ir_datasets` de `Python`. Son conjuntos enfocados para la creación y evaluación de SRI de documentos. De la base de datos que contiene el módulo se utilizó:

- `cranfield`
- `beir/cqadupstack/gaming: 45301,`
- `beir/cqadupstack/mathematica: 16705,`
- `beir/cqadupstack/physics: 38316,`

Se consideró preprocesar conjuntos de datos fuera de este módulo, tales como los proporcionados por los profesores: `reuters` y `20news`. Sin embargo al ser conjuntos con estructuras distintas, requieren un cargado de documentos particular, así que por comodidad se utilizaron estos.

3.1 Preprocesado de un documento

Teniendo cargado un documento, analizamos su cuerpo para extraer los datos que necesitamos. Si trabajamos con el Modelo Booleano, solo necesitaremos extraer la ocurrencia de términos. Para ello, se almacenará en un `set` dichos términos. Sin embargo el cuerpo del documento pasa por un proceso de normalización, donde llevaremos las mayúsculas a minúsculas, y con ayuda del módulo `nltk`[2] normalizaremos algunos de los términos, por ahora de forma sencilla (por ejemplo, convirtiendo

[1] Solo los datos que son independientes de la consulta. Por ejemplo, la ocurrencia de términos en documentos, la frecuencia de términos en documentos, ...

[2] Módulo enfocado en el procesamiento de lenguaje natural

plurales en singulares,..., pero no haciendo correcciones a los términos mal escritos). Normalmente se suele eliminar las *palabras vacías* (o *stop-words*), sin embargo se consideró no eliminarlas primeramente del Preprocesamiento y ya luego sí se eliminaron por los resultados que se obtuvieron. Sin embargo, durante el Modelado, podemos obviarlas o no, a nuestra conveniencia. De ahora en adelante cada vez que se diga término, se considerará a las palabras extraídas de los documentos que pasaron por el anterior proceso de normalización. Una vez tengamos los términos de cada documento se hace un diccionario invertido de la ocurrencia de términos; es decir, un diccionario donde las llaves son los términos y sus valores correspondientes serían el conjunto de índices de los documentos en los que aparece.

Similar a esto se hace el preprocesado de los datos que requieren los modelos vectoriales y de indexación semántica latente. Lo que uno lo que procesa es la frecuencia y los valores $tf * idf$, y el otros, extrae otros valores, construye matrices y las procesa, guardando nuevas matrices con la información que necesita el modelo. El preporcesado del texto siempre es el mismo. Se debe mencionar que solo se hace una lematización, aunque primeramente se hizo tambien un steaming pero no tuvo buenos resultados

4 Modelado del sistema

Una vez preprocesado los documentos, el sistema cargará los datos de ocurrencia de una colección específica, y los tendrá en memoria RAM. Note que leer estos datos la primera vez tiene un costo temporal menor que calcular las ocurrencias, frecuencias y demás valores de los términos en documentos. Estando esta información en memoria se realizarán las operaciones necesarias para poner a funcionar el modelo.

4.1 Modelo Booleano

Para comenzar decidimos implementar un Modelo Booleano. Es sabido que este modelo considera los documentos como vectores binarios por cada término en la consulta; y la consulta es una expresión booleana con los términos como sus átomos. La idea es simple: dada una consulta evaluar los vectores documentos con la expresión booleana. La similitud entre documento y consulta es 1 si existe algún componente de la Forma Normal Disyuntiva (FND) que sea igual al vector del documento. En caso contrario 0. Note que la similitud expresada de esta forma trabaja con la FND de la consulta, lo cual presupone un proceso extra. Sin embargo, es posible obviar esto si lo que se hace es evaluar directamente el vector en la consulta; este resultado será 0 o 1, si no coincide la consulta o si sí coincide (respectivamente).

De manera general este modelo se define por:

1. D: Conjunto de de términos indexados
2. Q: Expresión booleana sobre términos indexados
3. F: Álgebra booleana sobre términos y documentos
4. R: Función booleana que indica si el término d_j es relevante a la consulta q

En la práctica una forma de evaluar cada similitud entre la consulta y los documentos, se puede hacer hallando el vector binario del documento y evaluándolo en la expresión booleana. Sin embargo eso tiene el inconveniente de recorrer todos los documentos. Una forma que suele ser más eficiente es considerar (abusando de la informalidad) los documentos en los que solo aparecen los términos. Por ejemplo, la consulta $t_1 \& t_2$ se puede limitar al espacio del conjunto de documentos a los que ellos pertenecen, donde solo nos interesarían los documentos que pertenezcan a ambos conjuntos.

Un análisis más formal nos permite verificar que una operación *or* entre dos términos en la consulta produce como resultado, la unión de los conjuntos de documentos a los que ambos términos pertenecen. La operación *and* en cambio, requeriría la intersección de los documentos a los que pertenecen ambos términos. Por último la operación *not* sería hacer el complemento del conjunto de documentos al que pertenece el término en cuestión. Note que para realizar esta última operación entre conjuntos se necesita tener un conjunto universo. Como estamos trabajando sobre colecciones de documentos, se puede considerar el conjunto universo como el propio corpus.

Para implementar este modelo se tuvo en cuenta estas últimas consideraciones. Teniendo procesado los documentos con la información de su frecuencia en los mismos, lo que se hace es crear los conjuntos de documentos de cada término (cada vez que nos refiramos a conjunto de documentos, por lo general nos estamos refiriendo al conjunto de los índices de los documentos). Con trabajamos con Python, las operaciones entre conjuntos son más eficientes con la clase 'set()'. Entonces se crea un diccionario de términos-conjunto_documentos, y esta estructura nos permitirá hacer las operaciones de unión, intersección y diferencia, necesarias para evaluar el modelo.

Otra cosa a tener en cuenta en este modelo, es el formato de la consulta en la práctica. Para ello debemos permitir que el usuario introduzca una consulta válida perteneciente al álgebra booleana. Por lo tanto se creó una gramática para reconocer consultas booleanas, definido de la siguiente forma:

$$\begin{aligned} E &\to \quad TX \\ X &\to \quad \wedge E \quad | \quad \vee E \quad | \quad \epsilon \\ T &\to \quad t \quad | \quad (E) \quad | \quad \neg T \end{aligned}$$

En la práctica se definió un parser recursivo descendente muy sencillo para parsear y evaluar al mismo tiempo la consulta. Además los caracteres que se usan para las operaciones *and*, *or*, y *not*, son &, | y $\sim$, respectivamente.

4.1.1 Ejemplos de consultas

Como este modelo tiene la peculiaridad de que las consultas están definidas en su propio lenguaje, solo se aceptarán consultas en dicho formato. Por lo tanto si quisiéramos hacer la siguiente consulta: `note on creep buckling of columns`, no sería válida. Para ello debemos definir algún tipo de operación binaria entre sus términos. Digamos que nos interesan los documentos donde al menos aparece alguno de los términos. En dicho caso la consulta sería:

`note | on | creep | buckling | of | columns`

Esta consulta sí es válida y luego de procesada, algunos de los resultados serían los documentos con los títulos siguientes:

1. note on creep buckling of columns
2. stagnation point heat transfer in partially ionized air
3. piston theory - a new aerodynamic tool for the aeroelastician
4. on transverse vibrations of thin, shallow elastic shells

entre muchos otros títulos.

Realmente este respuesta dada no está en algún orden estrictamente fijo, y no tiene un ranking para ordenarlos. De esta forma el primer resultado podría ser mucho menos relevante para el usuario que el décimo resultado.

4.1.2 Medidas

La colección `cranfield` cuenta con consultas y respuestas validadas para la comprobación de modelos de recuperación de información. Estas se utilizaron, para extraer las medidas de evaluación del modelo. Se consideró principalmente el Recobrado, la Precisión y sus medias. Así como el tiempo de ejecución de cada consulta. Las consultas al estar en un formato inválido para el modelo, se mapearon a una consulta válida del modelo. Se hizo 2 tipos de mapeo: uno considerando operadores *or* entre cada término, y otro usando *and* de la misma forma. Los resultados obtenidos fueron los siguientes:

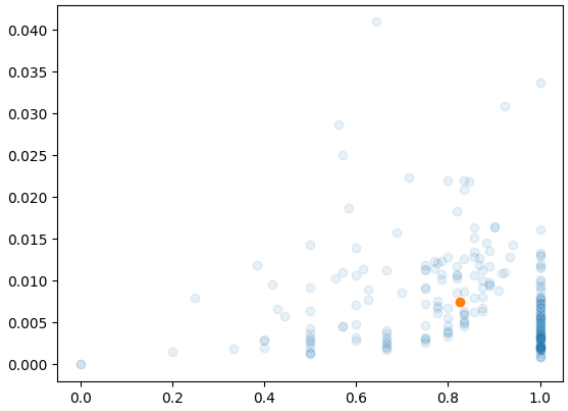

Figure 1: Recobrado/Precisión, del Modelo Booleano con consultas *or*

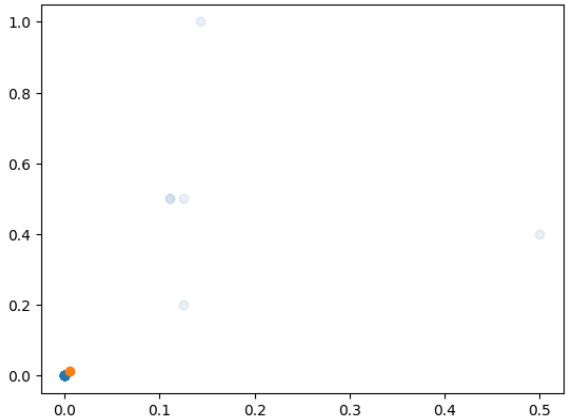

Figure 2: Recobrado/Precisión, del Modelo Booleano con consultas *and*

Puede notar primeramente, que los resultados son pésimos en precisión. Sin embargo las consultas *or* ofrecen mejor recobrado que las *and*, y es preferible decantarse entonces por las consultas *or*, ya que al menos le van a mostrar al usuario un fragmento de la colección en donde habrán al menos documentos relevantes. También resulta más natural para el usuario

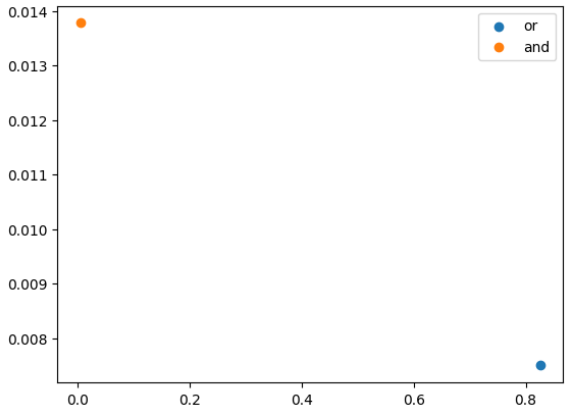

Figure 3: Recobrado/Precisión solamente las medias, del Modelo Booleano con consultas *or* y *and*

común hacer una búsqueda donde pudieran no estar algunos de los términos, por lo tanto una consulta de *or* tiene mayor sentido

En cuanto al tiempo de consulta, el promedio de la espera de cada consulta fue de 0.0007115989261203342 segundos. Lo cual es un buen indicador, para la espera del usuario.

4.2 Modelo Vectorial

El modelo vectorial de forma general en contraste al booleano, crea un ranking al hacer la consulta. A grandes rasgos le asigna a los términos en los documentos pesos, que influirán directamente en el cálculo de la similitud de la consulta. Para esto se necesitan los pesos locales por documentos tf, que no son más que las frecuencias normalizadas, y los pesos globales de los términos que vienen dados por el idf.

Lo que se debe tener principalmente en cuenta en este modelo es la estructura que almacene esta información. Note que una forma válida pero ineficiente espacialmente es una matriz de términos y documentos. El acceso es rápido pero consumiría en muchos casos, demasiados recursos. Claramente esto empeora a medida que crece la colección de documentos, ya que si además los temas son muy variados entonces la cantidad de términos crecería muy rápido y la dimensión de la matriz aumentería tanto por filas como por columnas. La solución a esto, sacrificando un poco de tiempo, es tener un diccionario de la siguiente forma $diccionario < trminos, diccionario < documentos, tf * idf >>$. Note esta estructura nos permite indexar en $(trmino, documento)$, como mismo ocurriría con una matriz. Pero la diferencia es que aquellos términos que no existían en un documento, tenían 0 en su posición correspondiente en la matriz, en cambio en la estructura de diccionarios, estos elementos no existirían; en colecciones donde los documentos tratan de temáticas muy similares, tal vez la cantidad de 0 en la matriz original es muy poca, y podría resultar más cómodo esa representación. La colección principal de documentos con la que trabajamos no tiene esta característica por lo que mucho menos costoso la estructura de diccionarios.

La fórmula de similitud de este modelo es calculando el ángulo entre los vectores de documentos y el vector de consulta. Los vectores de documentos se extraen del diccionario de términos-documentos. El vector de consulta pasa por un proceso similar pero está influenciado por una constante de suavidad α. Tras algunas pruebas que hicimos con valores cercanos a 0.5, se decidió utilizar $\alpha = 0.4$ para suavizar este vector.

Algo que se debe tener en cuenta en la implementación es que el modelo vectorial que hicimos no calcula los valores de $tf * idf$, ya que esto podría presuponer un alto costo de tiempo en dependencia del tamaño del corpus. Para ello en la primera fase de la recuperación se extraen estos valores y se almacenan en el disco duro para que sean solamente cargados al iniciar el modelo.

4.2.1 Ejemplo de consulta

Si nuestra consulta fuera:

`material properties of photoelastic materials`

simplemente le pide al modelo, esa misma consulta sin modificar. Mientras mas palabras fuertemente ponderadas tenga la consulta, la extracción será mejor. En cambio con consultas excesivamente grandes (que llegan a ser incluso tan grandes como algunos de los documentos) los resultados pueden verse afectado. Esto último nos sucede con los otros corpus que hemos añadido. Si probamos esta consulta en nuestro modelo vectorial un fragmento del resultado son los títulos:

1. photo-thermoelasticity
2. note on creep buckling of columns
3. a free-flight investigation of ablation of a blunt body to a mach number of 13 .1.
4. a theoretical study of stagnation point ablation

Este modelo, sí devuelve los resultados de forma ordenada, por lo que el primer resultado sería el que mayor similitud tiene con la consulta, el siguiente sería el segundo más similar y así sucesivamente. Esto ayuda al usuario a revisar los documentos en este orden, donde se supone que los documentos más relevantes para el usuario estén cercanos al inicio.

4.2.2 Medidas

En este caso cuando hicimos las consultas al modelo vectorial sobre el corpus de cranfield teniendo en cuenta los resultados que se obtienen de tomar los documentos con relevancia mayor a 0, 0.1, 0.2, 0.3, o tomando los primeros 25, 50, 75 y 100 documentos del ranking, obtuvimos los siguientes resultados:

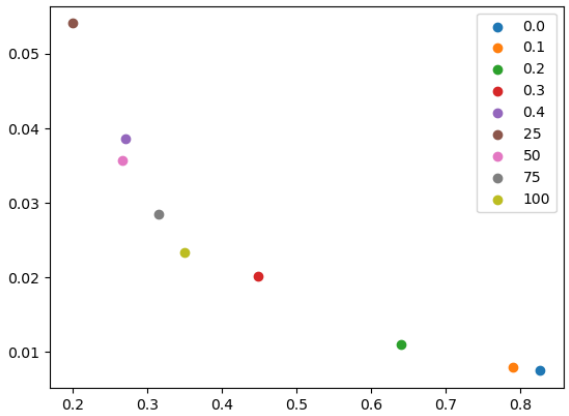

Figure 4: Recobrado/Precisión de las medias, del Modelo Vectorial

Aquí se puede ver una mejoría con respecto al modelo booleano. La precisión sigue siendo mala, pero mejora. En cambio el recobrado tiene mayor diversidad, pero evidentemente a menor cantidad documentos recuperados, o una exigencia mayor de relevancia, el recobrado disminuye. Los 2 grupos que nos llamaron la atención para una posible implementación final, fueron aquellos resultados con relevancia mayor a 0.3 y los primeros 100 resultados. La ventaja de esto con respecto a los otros es que dan un buen recobrado, sacrificando precisión pero no obviando esta. Las siguientes figuras muestran a mas detalles las medidas de todas las consultas con ambos grupos de respuestas.

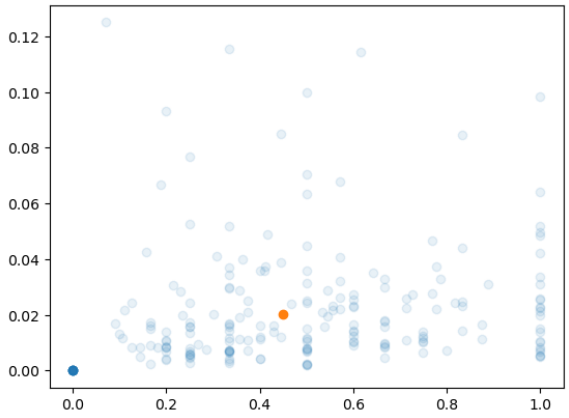

Figure 5: Recobrado/Precisión, del Modelo Vectorial con relevancia mayor a 0.3

También se calcularon estas medidas para otras colecciones pero los resultados fueron pésimos. Estas colecciones son las que sus consultas de prueba son muy grandes. Las medias por ejemplo de 'beir/cqadupstack/gaming' quedaron:

y las de 'beir/cqadupstack/physics' quedaron:

Luego de obtener estos resultados empíricos decidimos optar por la implementación del modelo considerando los documentos con mas de 0.3 de similitud en la consulta.

4.3 Modelo Indexación Semántica Latente

Quisimos implementar este otro modelo, en primer lugar porque a gusto personal nos interesa el álgebra (y este modelo hace buen uso de esta), y en segundo lugar porque la bibliografía plantea que en general este modelo supera al vectorial (y queríamos probar por nosotros si esto era real con el corpus 'cranfield').

Este modelo se enfoca en hacer una reducción de dimensión de una matriz de términos y documentos, para obtener otra matriz que sea similar a la original, la cual tendrá información latente de estas relaciones. Trabaja bajo el álgebra lineal, y se utiliza la descomposición SVD, y el algortimo de reducción de su dimensión. Para ello una vez obtenidas las matrices:

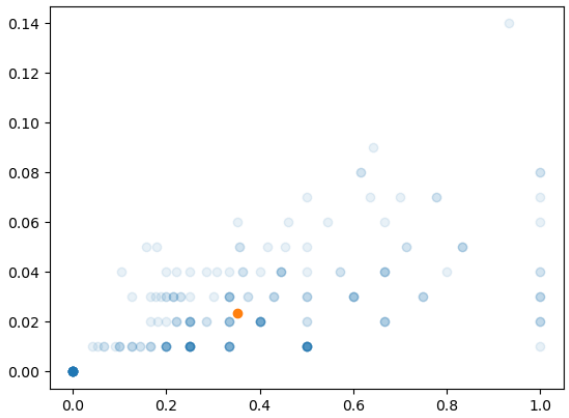

Figure 6: Recobrado/Precisión, del Modelo Vectorial tomando los 100 primeros documentos relevantes

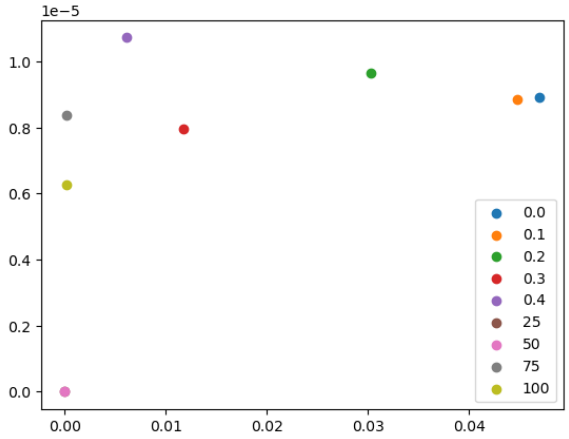

Figure 7: Recobrado/Precisión, de las medias del Modelo Vectorial 'beir/cqadupstack/gaming'

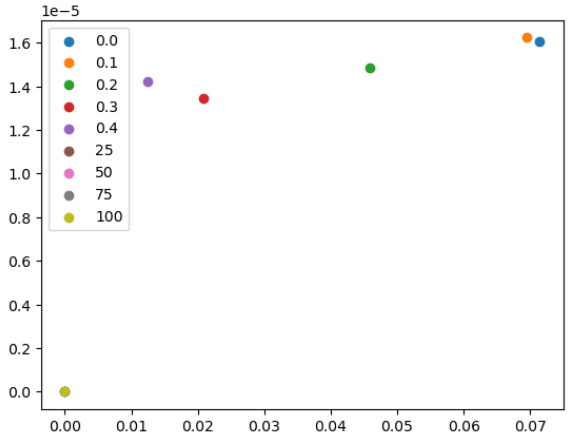

Figure 8: Recobrado/Precisión, de las medias del Modelo Vectorial 'beir/cqadupstack/physics'

$$A = TSD^t$$

donde D es la matriz de relación de los documentos en su espacio, y T lo mismo pero de los términos, y S contiene las raíces de los valores singulares; se eliminan los menores de estos valores hasta quedarnos con los primeros k. Luego de hallar esta matriz S_k reducida, se hallan T_k y D_k^t, reduciendo sus columnas y filas respectivamente. Luego la multiplicación de estas nos conformaría la matriz A_k que sería similar a A. Ahora bien, nuestros vectores de documentos sería las filas de D_k y la matriz $R_k = S^{-1}T_k^t$ sería nuestra matriz de reducción del vector consulta. Finalmente nuestra función de similitud será:

$$sim(q, d_k) = sim_v(R_k q)$$

donde sim_v es la misma función de similitud del modelo vectorial.

Algo que se debe considerar primeramente es la conformación de la matriz A, la cual se suele conformar de forma similar a los pesos del modelo vectorial, donde multiplicamos pesos locales de los términos en los documentos, con los pesos globales

de los términos. La bibliografía sugiere varios de estos valores. Nosotros usamos los similares al tf e idf, pero al final nos quedamos con:

$$l_{ij} = \log(tf_{ij} + 1)$$

$$g_i = 1 + \sum_j \left(\frac{p_{ij} \log(p_{ij})}{\log(n)} \right)$$

ya que estas eran las que por lo general daban mejor resultados empíricos. Y en la práctica fue así.

Este modelo lleva una importantísima observación en su implementación. La matriz A puede ser muy grande, sobre todo en corpus grandes como explicábamos en el modelo vectorial. Y aunque su objetivo es reducir esta dimensión el proceso para obtener la descomposición SVD requiere mucha memoria RAM, y poder de cómputo. Por lo tanto no es posible hacer esto sobre todos los corpus si tenemos limitada=o esto. En nuestro caso fue posible hacer con la colección principal ya que esta solo cuenta con 1400 documentos.

Una característica buena de este modelo es que solo necesita hacer un cálculo vectorial para reducir la consulta, y hallar la similitud de estas con los vectores de los documentos que ya están guardados. Por lo tanto su evaluación suele ser rápida por consulta. Otra característica de este modelo es que al reducir la dimensión de A, algunas palabras medianamente similares en cuanto a lexema o semántica pudieron haber reunido en un mismo concepto, por lo que la recuperación de información debe verse aumentada ya que aparecen pinceladas de sinonimia entre términos.

4.3.1 Ejemplo de consulta

Un ejemplo de esto último lo podemos ver si hacemos la consulta:

`photo`

En el modelo vectorial solo obtenemos 5 resultados y en cambio en el modelo de indexación semántica latente, obtenemos 12. En este caso ocurrió lo mencionado. La consulta coincidió con información latente en la matriz reducida.

4.3.2 Medidas

Con respecto al tiempo en que se tarda la realización de una consulta, el promedio obtenido fue de 0.000466314951578776 segundos por consulta. lo cual es un muy buen valor para la espera impaciente del usuario.

En cuanto a las medidas de recobrado y precisión se mejoró mucho con respecto a los dos anteriores modelos:

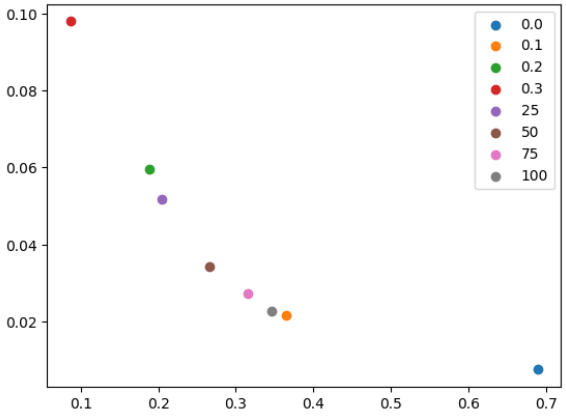

Figure 9: Recobrado/Precisión, de las medias del Modelo LSI

Puede notar que el precisión de este modelo es mejor que en los anteriores sacando las mismas medidas sobre los conjuntos de respuestas con las que se midieron también los anteriores modelos. Pero las respuestas con mayor precisión pierden en recobrado, por lo tanto los valores que decidimos analizar fueron, donde se toma una cantidad fija de los 50 primeros y los 75 primeros. Las medidas de cada consulta en sus respectivos grupos

Al final nos quedamos con la implementación que extrae los 50 primeros documentos del ranking ya que aunque el recobrado es un poco menor que el de 75 documentos, la precisión es mejor y sabemos que siempre extraerá a lo sumo los 50 documentos que a nuestro entender es una buena cantidad de documentos para que el usuario pueda revisar.

5 Resumen del análisis de los modelos

Después de explicado los modelos y sus implementaciones en código podemos sacar algunas conclusiones sobre las ventajas y desventajas que tienen estos.

El modelo booleano que es bastante sencillo en teoría, en la práctica no resultó del todo así. La parte sencilla realmente si fue hacer el cálculo de las similitudes, pero la dificultad vino al tener que definir una grámatica para el espacio de consultas. Aunque la grámatica implementada era bastante sencilla, hacer un parser para esto es un trabajo que requiere conocimientos

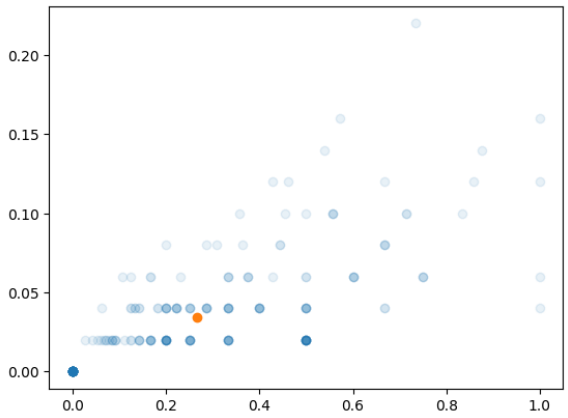

Figure 10: Recobrado/Precisión, del Modelo LSI para los primeros 50 resultados

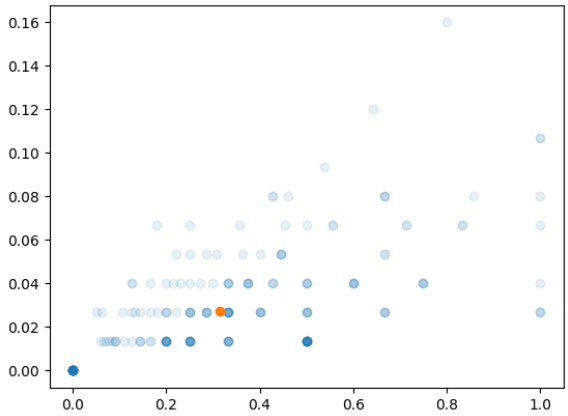

Figure 11: Recobrado/Precisión, del Modelo LSI para los primeros 75 resultados

de Compilación para hacerlo eficientemente; además de que toma tiempo de implementación, y si esta parte es comparada con los otros modelos, ellos carecen de este paso (para beneficio del programador). Algo bueno de este modelo es su velocidad, ya que las operaciones entre conjuntos cuando se usan estructuras eficientes para esto, la respuesta es casi instantánea. Sin embargo este modelo contiene en este escenario más debilidades que fortalezas. La recuperación que realiza es completa, y un usuario común suele preferir que las respuestas sean bastante buenas aunque la búsqueda sea parcial. Para consultas muy grandes al emplear componentes *and*, el recobrado es pésimo ya que la probabilidad de que cada término esté en el documento disminuye con el aumento de la cantidad de esos términos. Por otra parte hacer emplear componentes *or* entre una cantidad grande términos, tenderá a darnos un muy algo recobrado que pudiera llegar a ser casi la colección completa de documentos y esto ofrecerá muy mala precisión. A grandes rasgos el modelos booleano para consultas de usuarios comunes y bases de datos variadas, no es buen modelo de recuperación.

El modelo vectorial es un poco más complejo de implementar en teoría, pero la práctica demostró que no está tan lejos del booleano. Básicamente los datos que necesita este modelo de los términos y documentos es la frecuencia, mientras que en el booleano la necesidad es similar, pero de ocurrencia. El trabajo radicaría en hacer la evaluación de la consulta, pero no es muy complicado realmente. La desventaja inicial es que se necesitan los $tf * idf$ y es un cálculo que se necesita hacer el cual requiere tiempo, y si lo hacemos cada vez que se nos pide una consulta, entonces emplearíamos este tiempo para calcular nos valores invariante del $tf * idf$. Sin embargo esto deja de ser desventaja cuando los cálculos los ejecutamos una primera vez, y los almacenamos para luego utilizarlo. Esto aporta mucha velocidad. La otra cosa que puede ser desventajosa es el almacenado de estos valores. Si se guardan en una matriz esto requería mucho espacio en memoria, pero la solución a esto es como vimos, utilizar alguna estructura de datos reducidos. Una ventaja que tiene también es que incorpora un ranking a los documentos recuperados.

Por último el modelo de indexación semántica latente tiene una super desventaja con respecto a los otros 2 modelos: la capacidad de cómputo y espacio para procesar los datos en una etapa inicial. Este es la principal limitante del modelo, que se ve afectada cuando se utilizan corpus grandes y poca capacidad espacial y computacional. En sí tampoco es tan complicado implementarlo, siempre que se utilicen herramientas que faciliten el cálculo matricial; como en nuestro caso que utilizamos la librería `numpy` de Python. Ahora una gran ventaja es que además de que la coincidencia es parcial, existen conceptos que engloban términos, permitiendo recuperar documentos con términos similares semánticamente. Además como el cálculo de la similitud, solo requiere una reducción de consulta mediante un cálculo vectorial sencillo, el resto es calcular similitudes con los vectores de documentos que ya están previamente calculados. Por lo tanto la ejecución de las consultas de este modelo son sencillas y rápidas si se tienen las matrices que utiliza, ya preprocesadas. Por último con este modelo se obtienen muchos mejores resultados que con los anteriores, debido a todo el análisis que hemos hecho.

6 Elementos adicionales

Otra funcionalidad que agregamos es la sugerencia de palabras parecidas una a una con los términos de la consulta. La forma de buscar las palabras a sugerir es la siguiente:

- Utilizando un método que nos devuelve la distancia de Levenshtain entre dos palabras, hallamos los términos más parecidos a los de la query (individualmente), iterando por los términos contenidos en el corpus, y resolviendo la palabra de menor distancia de Levenshtain para cada término de dicha consulta.
- Teniendo en cuenta que al usuario le interesa buscar resultados interesantes para el corpus en el que se encuentra, no solo se le sugieren palabras por el parecido a las de la query, sino que también se tienen en cuenta los pesos de los términos en el corpus, incluyendo en el proceso de selección de las palabras de la sugerencia, el *idf* del término en el corpus, utilizando un α ($0 \leq \alpha \leq 1$), es directamente proporcional a la distancia de Levenshtain, e inversamente proporcional al *idf* del término
- Por tanto, la fórmula del peso de un término del corpus t respecto a una palabra de la consulta t_q, con un factor α quedaría de la siguiente manera: $sim(t, t_q) = \alpha * idf(t) + (1 - \alpha) * Levenshtain(t, t_q)$

Este algoritmo se usa en la práctica en consultas que no tuvieron una buena recuperación de información para mostrarle al usuario palabras similares que pudiera utilizar para mejorar la consulta.

7 Recomendaciones

Las recomendaciones que queremos compartir son:

1. Si se replican estos modelos se recomienda tener en cuenta todos los valores que se le dieron a los constantes que se tuvieron que utlizar en los distintos modelos.
2. Utilizar un sistema hosting para calcular la descomposición SVD del modelo semántica latente si se utilizan corpus grandes, para obtner los datos preprocesados
3. Hacer una extracción de documentos de un corpus para luego hacer la reducción de dimensión con el modelo de semántica latente, y utilizarlo para el corpus entero, insertando cada documento poco a poco en la matriz reducida.
4. Hacer una ampliación de consultas, donde se agreguen o modifique palabras, que sean sinónimos de algunas de las palabras principales de la consulta.
5. Incluir elementos del modelo booleano en el modelo vectorial, para hacer una pequeña búsqueda total primero y luego una parcial sobre un espacio más reducido del corpus.