Proyecto Final Sistemas de Recuperación de Información(Preentrega)

Abel Antonio Cruz Suárez A.CRUZ@ESTUDIANTES.MATCOM.UH.CU
Grupo C511

Jose Carlos Piñeira JCH@ESTUDIANTES.MATCOM.UH.CU
Grupo C511

Daniel Reynel Dominguez ceballos Dos DANIEL.DOMINGUEZ@ESTUDIANTES.MATCOM.UH.CU
Grupo C511

Tutor(es):

Resumen

Los sistemas de recuperación de información constituyen una rama de vital importancia en las ciencias de la computación. Con este trabajo se busca mostrar esa importancia mediante la creación de un modelo que transite por todas las fases de la recuperación de información. También es nuestro objetivo aplicar técnicas ya sean de procesamiento de texto, expansión de queries y/o retroalimentación que tributen a un modelo óptimo

Palabras Clave: Sistemas de recuperación de información (SRI), procesamiento del lenguaje natural(PLN), motores de búsqueda(MB), Modelo de recuperación información(MRI).
Tema: Tema, Subtema.

1. Introducción

Como se ha ido comprobando a lo largo del curso, los sistemas de recuperación de información tienen que cumplir con el único objetivo de otorgar documentos, responder preguntas, procesar paginas web y darles relevancia, entre otras muchas técnicas. Conocemos principalmente los motores de búsquedas dedicados a la tarea de recibir una consulta y entregar una serie de documentos y páginas web que considera relevantes. La explicación exacta de como funciona o qué fases realizan MBs como Chrome o mozilla va más allá del contenido que abarca este trabajo. Sin embargo, aquí se pretende desarrollar un modelo de recuperación de información clásico que permita realizar las mismas tareas que un MB cualquiera.

Debemos destacar que solo existen tres modelos clásicos y sobre ellos se han ido construyendo otros más avanzados. Estos modelos "evolucionados" tienen diversas particularidades, como el uso de redes neuronales en el modelo vectorial o redes de inferencia en el probabilístico; esto los hace más certeros en cuanto a recuperar información. Incluso con el uso de estas evoluciones de los MRI, no debemos ser absolutos, pues no hay un modelo superior a otro, todo depende de sobre que corpus estemos trabajando, quiénes serán nuestros usuarios, para qué se desarrolla la el modelo, se quiere lograr eficiencia a coste de exactitud total de términos o mayor complejidad temporal en favor de obtener documentos relevantes sin total exactitud en los términos. Muchas son las preguntas que nos hacen decantarnos por un modelo y no por otro, y solo con su respuesta y análisis es posible una buena elección de MRI.

1.1 Nota

Nota: Para probar el proyecto se deben copiar todas las carpetas del corpus 20news-18828 de la colección 20 Newsgroups dentro de "./engine/corpus", pues el sistema a partir de ahí realiza el procesamiento. Para la versión final se quiere que el usuario pueda introducir la ruta donde se encuentra el corpus dentro de su PC, y con ello ganar en usabilidad y facilitar la interacción. Para la entrega final también se ofrecerá una entorno virtual(pipenv) listo para instalar las dependencias.

2. Dependencias

Link al proyecto.

- python 3.x
- numpy
- pickle
- nltk
- nltk.corpus (stopwords)
- nltk.stem.porter (PorterStemmer)
- nltk.tokenize (word_tokenize)
- pip3 install fastapi

- pip install "uvicorn[standard]"

Para instalar los paquetes stopwords, puntk de nltk se debe colocar la l, puede abirir una terminal, ejecutar el comando python3, luego en el script the python hacer:

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
```

Figura 1: codigo para obtener paqutes de nltk.

2.1 Como ejecutar el proyecto ?

- Situarse dentro de la carpeta engine
- Abrir una terminal en esa ubicación
- Correr este comando en la terminal 'uvicorn main:app'
- Abrir el navegador: http://127.0.0.1:8000/

3. Modelo de Recuperación de Información

Luego de hacer un profundo análisis de los diferentes modelos, se decidió emplear el modelo vectorial. Primeramente nos enfocamos en el usuario, al este no tener que ser experto le sería muy complejo trabajar con un sistema de consultas basado el lógica boleana, pues en más de una ocasión tendría que llevar su consulta a la forma normal disyuntiva(FND) para que esta sea correctamente procesada. Si bien este problema es posible mitigarlo implementando una especie de traductor del lenguaje natural a la lógica boleana, incrementaría la complejidad temporal a la hora de realizar consultas, pues como mínimo un intenso PLN debería llevarse a cabo solo para no excluir términos importantes y luego aplicar un algoritmo para convertir dicha consulta procesada a la lógica boleana, que incluye la conversión a FND. Esto añadiría a la primera fase una enorme cantidad de trabajo, que luego debería ser comprobada y ajustada para lograr objetivos que con un modelo vectorial se cumplen satisfactoriamente.

El corpus empleado para esta fase fue 20Newsgroup, haciendo un análisis de dicho corpus observamos que esta compuesto por cerca de 18000 documentos; estos se encuentran divididos en 20 temas distintos y cada uno es acerca de una noticia sobre el tema. Emplear un modelo de recuperación probabilístico, parece una opción viable, ya que de antemano se conoce información sobre los conjuntos de documentos y luego de una inspección se pudieran encontrar algunos conjuntos de documentos relevantes. Aún así, seguimos considerando el modelo vectorial como nuestra mejor opción pues los documentos presentan un encabezado donde exponen de trata dicha noticia y al estar formado por palabras clave del documento, le daría mayor similitud con las consultas. Esto es basándonos en que normalmente cuando queremos obtener un documento entregado por un MB cualquiera, colocamos palabras claves en la consulta para obtener mejores resultados.

Otro análisis que nos hace decantarnos por un modelo vectorial es que ofrece una función de ranking basada en el coseno del ángulo entre vectores que representan el documento y la consulta. Aunque existen otras formas de realizar el ranking, en esta fase se emplea la del coseno, sin descartar que se pueda poner en práctica otra, tomando la que otorgue mejores resultados sobre el corpus según las métricas.

Utilizar un modelo como el vectorial nos permite mayor maniobrabilidad a la hora de recuperar documentos, ya que no está atado a la coincidencia total o a tener que hacer un análisis de que documentos puedan ser relevantes por nosotros mismos, lo que puede derivar en una mala clasificación. El único inconveniente es que no se tiene en cuenta la correlación de términos y que le otorgaría mayor precisión.

4. Fases de la Recuperación de Información

Para la elaboración de nuestro modelos dividimos el trabajo en 4 fases: Procesamiento de documentos, indexación de documentos, procesamiento de consulta y fase de recuperación de documentos relevantes.

Estas cuatro fases son ejecutadas por una instancia de la clase Pipeline. Su objetivo es solo el de ejecutar los diferentes procesos por los que transita el motor de búsqueda. Como parte propia de este pipeline existen tres métodos de vital importancia: *scan_corpus*, *save_to_disk* y $retrive\_from\_disk$.

La función *save_to_disk* es la encargada de almacenar en el dico duro todas las tablas computadas en el proceso de indexación. Con esto se guardan las tablas, *idf*, los pesos de los documentos y las normas. El objetivo de este proceso es el de reducir en gran medida el tiempo en que se obtienen los documentos, pues en el caso contrario solo calcular la norma del documento puede tardar bastante ya que se están procesando más de 18000 documentos con cientos de términos y esto aumenta enormemente la complejidad temporal. El almacenar la tabla de pesos de documentos y de *idf* también resulta vital para no tener que en tiempo de consulta hacer todos los cálculos que conlleva esto, que incluiría procesar nuevamente los documentos.

scan_corpus es la función encargada de dado un path donde se encuentra un corpus, scanear todos los directorios en búsqueda de archivos que representen a los documentos. Para ello recursivamente analizamos cada directorio, si es una carpeta lo que encontramos se realiza un llamado recursivo con el path de dicha carpeta y se devuelven todos las direcciones de documentos encontrados en el directorio que se este revisando. El código se muestra a continuación:

Como dato relevante aclarar que el proceso de indexación toma cerca de 40 minutos debido al tamaño del corpus y se tendrían que esperar con cada consulta.

```
def __scan_corpus(self,path):
  directories = listdir(path)
  files_found = []
  for e in directories:
   file_path = join(path,e)
   if not isfile(file_path):
    files_found += self.__scan_corpus(file_path)
   else:
    files_found.append(file_path)
 return files_found
```

Figura 2: Código función *scan_corpus*.

Esto se reduce a segundos de búsqueda aplicando esta técnica de guardar las tablas en el disco duro.

La función *retrive_from_disk* como su nombre indica permite recuperar las tablas del disco duro. Para estas funciones se emplea la biblioteca pickle.

Para la comodidad del usuario se creó la función *get_subjects* encargada de obtener los asuntos de cada noticia y esto será lo que se le muestre en el buscador, pudiendo ver de antemano algo acerca del documento, evitando que vea un número o un path, pues los archivos solo tienen como nombre un número.

4.1 Procesamiento de Documentos

En esta fase nos encargamos de dado un documento reducir su tamaño los más posible, de forma tal que su representación vectorial sea menor, se mantenga su idea central y términos claves. Para esta tarea empleamos la biblioteca de python NLTK, destinada al procesamiento del lenguaje natural que nos facilita el proceso de tokenizar el documento.

En la clase $Scaner$ del archivo $scaner.py$ nos encargamos de procesar el documento. El principal algoritmo que se emplea en esta fase realizan una serie de tareas, lo que reduce en gran medida la dimensión del vector documento que de por sí ya es grande.

se recibe un texto que transita por diferentes funciones y contribuyen a la reducción de dimensión. Utilizando las expresiones regulares se eliminan de los documentos aquellas secuencias que comiencen con "from:", también se creo una expresión regular capaz de reconocer y eliminar las direcciones de correo y el string "Subject" o "Subject".

Para los siguientes métodos se hace uso de la biblioteca NLTK, que permite el procesamiento del lenguaje natural y tokenizar textos. Para la tarea de tokenización se emplea el algoritmo word_tokenize, luego convertimos los tokens de mayúsculas a minúsculas, se hace un llamado a nuestro método $remove_reg$ encargado de eliminar signos de puntuación, se eliminan los tokens restantes que no sean palabras, se eliminan las stopwords del inglés y finalmente se lleva a cabo un proceso de Stemming . Esto no es más que producir variantes morfológicas de una raíz/palabra base. Los programas de Stemming se conocen comúnmente como algoritmos de Stemming o lematizadores. Un algoritmo de Stemming reduce las palabras "chocolates", "chocolatoso", "choco" a la raíz de la palabra, "chocolate" y "recuperación", "recuperado", "recupera" se reducen a la raíz "recuperar".

4.2 Indexación

En la clase el archivo *irm.py* encontramos la clase $Vector\_Space\_Model$. Como su nombre indica esta clase posee toda la lógica encargada de llevar a cabo el proceso de indexación, ranking y recuperación.

Para el cálculo de la similitud entre consulta y documento es necesario el cálculo de los pesos en los vectores que representan a la consulta y el documento. Es por ello que a la hora de realizar la indexación es necesario la construcción de las tablas tf e idf. La tabla tf en i, j indica la frecuencia con la que ocurre el término i en el documento j, luego de ser normalizada. Para la normalización se divide la frecuencia del término i entre la frecuencia del término que más se repite.

En el código se emplean diccionarios para representar tablas. Este método recibe una lista de tokens del documento y el número del documento. Luego, empleando un diccionario auxiliar, se revisa cada token, si ya se encuentra la llave $(token, \#dedocumento)$ significa que ese token ya ha sido visto en el documento y por tanto se aumenta la frecuencia en 1, en el caso contrario se añade la llave con valor 1 por ser la primera vez que se ve en el documento.

Para mantener en cuantos documentos se encuentra el token t se lleva otro diccionario $invert\_index$. $invert\_index[t]$ indica en la cantidad de documentos que se encuentra el token t. Esto se empleará luego en el cálculo de la tabla idf.

Una vez revisada la lista de tokens del documento, se pasa a encontrar la frecuencia del token que más aparece, se divide cada valor del diccionario por dicha frecuencia (normalización) y se actualiza el diccionario tf.

Para llevar a cabo el cálculo de la tabla *idf* se emplea una función muy sencilla de entender que para cada término del cuerpo de documentos se aplica la fórmula de *idf*, para la que previamente se tiene la cantidad de documentos del corpus y la lista de índices invertidos, con la que para un término sabemos en cuantos documentos aparece.

Finalmente para el cálculo de los pesos del vector documento la idea es ir por cada uno de los documentos y realizar la multiplicación $tf[t, d] * idf[t]$. Si se captura una excepción significa que el documento no posee el término y no es necesario mantener un cero o un valor extra en esa posición de la tabla. La última parte de este código refleja el método utilizado para calcular y mantener la norma del documento.

4.3 Procesamiento de consulta

En esta fase nos encargamos de procesar la consulta aplicando las mismas técnicas que a los documentos. En próximas actualizaciones del seminario se deben aplicar nuevas técnicas que faciliten la expansión de consultas.

En esta fase se lleva a cabo el proceso de vectorizar la consulta. Para ello se emplean las mismas técnicas de cálculo de $tf - idf$, solo que se le agrega una constante $\alpha = 0.5$. Estos algoritmos para calcular las tablas de tf e idf se encuentran en la clase $Vector\_Space\_Model$. Destacar que la tabla idf ya está previamente calculada desde el procesamiento de documentos.

4.4 Recuperación de Documentos

En esta fase se emplea el código para calcular la similitud (*similarity*). En este método estamos pasando por cada documento y por cada término de la query para realizar el producto escalar. Para ello vemos que se multiplica el peso del término t en el documento dj por el peso del término en la query. Destacar que como el término puede no estar en el documento se lanza una excepción donde no se realiza nada, esto es para evitar mantener un 0 en esa posición de la tabla.

Luego se realiza el cálculo de la similitud empleando la técnica del coseno del ángulo, computando también la norma del vector consulta y obteniendo la del documento. Finalmente se organizan de mayor a menor las similitudes y se toman las *threshold* primeras. Devolviendo los documentos que las representan.

5. Interfaz de Usuario

Para el desarrollo de la interfaz visual se utilizó Vue, un framework que provee muchas ventajas en los temas de desarrollo web(frontend). Se emplea una arquitectura sencilla por el hecho de que la complejidad no resulta llamativa en este escenario. La interfaz permite introducir el path donde se encuentra alojado el corpus, de igual forma pasar la query, manipulado posteriormente la respuesta del servidor. Se da la posibilidad al usuario además de enviar feedback de las respuestas que se reciben desde el servidor, para valor o no la significación de la información recuperada y trabajar en consecuencia.

5.1 Anexos

Figura 3: GUI con la realización de consulta

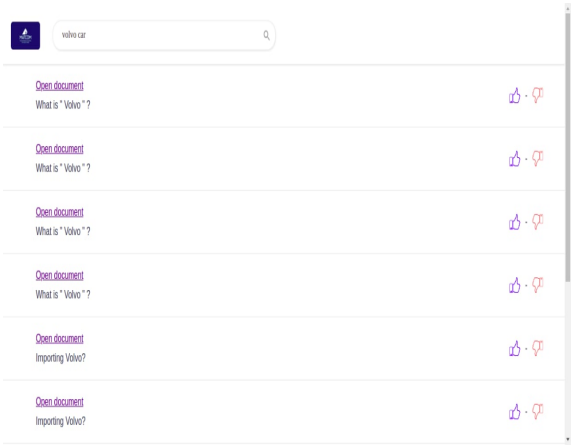

Figura 4: Resultado de la consulta

Como se aprecia en el resultado algunos asuntos de las noticias estás repetidos. Esto aparece así en el corpus y no implica que los documentos estén repetidos.