UNIVERSITAT POLITÈCNICA DE CATALUNYA

GRADO EN INGENIERÍA INFORMÁTICA

CERCA I ANÀLISI D'INFORMACIÓ MASSIVA

Q1 Course 2020-2021

User Relevance Feedback

GRUP 21

Autores:
Daniel CANO CARRASCOSA

0 INTRODUCCIÓN

Para esta entrega el objetivo es implementar con éxito el algoritmo de user relevance feedback y hacer pruebas con este para así sacar conclusiones.

1 FUNCIONAMIENTO BÁSICO DEL PROGRAMA

En cuanto al programa el funcionamiento de este sigue esencialmente la estructura presentada en el propio enunciado, pidiendo en primer lugar un conjunto de palabras, después un número de rondas y finalmente después de ese número de iteraciones devuelvo los resultados obtenidos de la query teniendo en cuenta los K documentos más importantes (Recall) y cogiendo finalmente solamente las R palabras más destacadas para la nueva query (Precisión). Yo mismo separo la query inicial por espacios para crear un array y sin filtrar nada (ya que a la hora de crear el índice aplique el token de whitespace) y si lo he hecho de esta manera es, si recordamos la última entrega donde según el análisis de la cantidad de palabras diferentes obtenidas el token de whitespace era con diferencia el que más tokens generaba, entonces esto repercutirá en que nuestra cantidad de palabras diferentes será mucho mayor, lo que implica la posibilidad de poder hacer búsquedas más ricas en contexto, por ejemplo si aplicara el filtro de ascii folding ya no podría hacer búsquedas que implicarán cantidades, y lo mismo con los demás filtros (aunque después muestro alguna prueba que he hecho para ver cómo afecta el hecho de aplicar estos filtros).

1.1 RESULTADOS OBTENIDOS

Una vez implementado con éxito me he dispuesto a empezar con las pruebas y he observado los siguientes puntos a destacar, los cuales me parecen interesantes.

- En primer lugar me gustaría mencionar los resultados obtenidos del subconjunto de ficheros de **novelas**. En este caso lo que he podido observar es una increíble tendencia a quedar atrapado en unos documentos en concreto, ya que cuando aumentaba K, la cantidad de documentos que analizaba aumentaba de forma correcta, no obstante el problema venía dado por R, que era quien me indicaba al final coger las R palabras más importantes o de más peso dentro de estos K documentos, ya que en el corpus de Novelas debido a sus características, provocaba una increíble tendencia a ciertas palabras, como por ejemplo PICKWICK, ya que había una sola novela que contenía esta palabra muchísimas veces, pero dentro de las 33 novelas solo existía en 1, por lo que el vector de pesos tf-idf por parte del tf premiaba como de concurrente era esta palabra en el texto y idf premiaba que solo existiera en un único documento, por lo que al añadir esta palabra en el top R, y hacer una busqueda con esta, automáticamente la consulta se reducía a 1 solo documento. Por lo que para paliar esto decidí reducir R y poner K al máximo (33 documentos) ya que no suponía una carga de cómputo muy pesada (Es decir aumente el Recall al máximo posible, aunque limitado por R, para así poder obtener una lista más extensa de documentos, por lo cual la precisión de mis consultas bajo). No obstante, después de todo esto también tuve que aplicar filtrado a la hora de crear índices debido a otras palabras como Cap'n que también me suponían un problema. Después de todo esto pude obtener resultados interesantes ya que al buscar Wizard, pude encontrar que los documentos pg4358.txt, pg22566.txt, KiplingJungleBook.txt, PoeWorksVol2.txt contenían referencias a la magia.

- Por parte de las **news**, el set de documentos era muchisimo mas extenso, pero esto no era problema, es más era mejor ya que de esta forma no pasaba lo dicho previamente al ser más documentos las consultas no quedarían atrapadas en términos concretos, con lo cual en pos de tener más diversidad de tokens decidí no aplicar ningún filtro y

tokenizar por espacios en blanco. Una vez hecha la prueba pude identificar que R me condicionaba nuevamente la salida, ya que al ser R el numero de términos escogidos y como en elasticsearch se indica la importancia con el operando '^' pero no indico en ningún momento el siguiente operando '~' ya que es un parámetro que si bien permite flexibilidad implica al mismo tiempo una falta de control que no me interesaba, ya que al no tener ningún criterio de base para aplicarlo debería introducirlo a mano, lo cual podía llevar a errores, cuanto mayor era R, menor era la cantidad de documentos encontrados, lo cual era lo esperado, por lo que tuve que reducir R a unos valores entre 4 y 5 para así recibir más cantidad de respuestas a la vez que aumentaba la variabilidad y poder sacar mayores conclusiones. A lo cual pude ver que la cantidad de temas diversos que se tratan es enorme, ya que ya sea buscando amigos, juegos o cocina siempre encuentro algún resultado aproximado de lo que estaba buscando.

Por ejemplo para la frase "How to become rich" y K = 60 y R = 4 me recomendaba textos de como volverse rico, o consejos de gente que lo era e incluso algún que otro texto de religión o política sobre el dinero.

- Finalmente para el conjunto de documentos de **arxiv** he observado resultados altamente parecidos a los de news, es decir que cuando buscaba un concepto este me lo relacionaba con otros documentos, y a parte de responder mi pregunta me enlazaba con otros temas, no obstante debido al contenido de estos si buscaba conceptos relacionados con "rich" se indicaba principalmente riqueza léxica, matemática o parecida, lo cual no sorprende ya debido al contenido del propio documento. En cambio si escribo algo como "exponential algorithm" automáticamente me empieza a redirigir a documentos de ciencias de la computación y alguno suelto de matemáticas, dejando claro donde está el contenido que estoy buscando.

1.2 ANALISIS DEL CODIGO E IMPLEMENTACIÓN

Para poder hacer todo este análisis he tenido que implementar un código eficiente para la ocasión, no obstante en el enunciado ya se indicaba el uso de diccionarios para aí poder sumar más rápido los vectores tf-idf, ya que al hacerlo con diccionarios se puede hacer una unión de las claves y sumar los diccionarios en una sola línea:

```
words = {t: words.get(t, 0) + Res.get(t, 0) for t in set(words) | set(Res)}
```

Y se hiciera con vectores el coste sería de O(n1 + n2) en el peor de los casos ya que tendríamos que recorrer los dos vectores de forma simultánea, en cambio aquí el coste pasa a ser max(n1,n2) ya que solo accedemos a las claves del diccionario con más entradas y ya nos sirve de cara al otro diccionario.

Por lo cual el coste es bastante menor, debido a que el acceso a una posición de los diccionarios es inmediata. No obstante después de pensarlo un poco y de implementar la versión con diccionarios, decidí implementar también una versión usando la librería de Pandas, ¿ por qué ? Pues tenía dos motivos principales:

1. Pandas es una librería que usa Series por debajo y que a su vez se basa en numpy, el cual está escrito en C++, con lo que obtenemos un rendimiento altamente superior comparado a python nativo, ya que al ser una librería basada en C++ y dedicada al ámbito científico está altamente optimizada para este tipo de tareas. Por ejemplo, el propio indexado está optimizado de una forma especial para el rápido acceso a los datos.

2. Debido al funcionamiento de Pandas este guarda toda la información en la memoria RAM, lo que evita lecturas a disco y hace más ágiles las operaciones (este último punto lo considere en caso de que pudiera dispararse el número de documentos consultados o guardados en nuestro DataFrame)

Y como suponía después de implementar las dos versiones, adaptando todas las funciones a las dos estructuras de datos se puede observar una increíble diferencia entre pandas y sus dataFrames y los diccionarios de python, siendo estos últimos increíblemente lentos comparados con la versión implementada en Pandas.