Session 3: User Relevance Feedback

Q1 2022-23

Integrantes:
Pol Pérez Castillo
Daniel Ruiz Jiménez

Fecha de entrega: 20-10-2022

Índice

Introducción

En esta práctica vamos a implementar un programa en Python que dada una query nos devuelva aquellos documentos que son considerados "relevantes", emulando el concepto de User Relevance Feedback usando la regla de Rocchio. Para ello, nos hemos ayudado de implementaciones de sesiones previas y de programas que se nos han suministrado.
En el sistema de Relevance Feedback tomamos unos resultados dada una consulta inicial, y usamos esta información para ver si éstos son suficientemente relevantes para el usuario, para que de esta forma no tengamos que volver a realizar esta misma consulta, esperando mejores resultados.
La regla de Rocchio consiste en obtener una nueva consulta a partir de una inicial, un conjunto de términos acompañados de "pesos", que denotan la "importancia" de que ese término aparezca en el documento resultante, que tendrán asociado un grado de confianza (alfa), y, para esta sesión, un conjunto de documentos que son considerados "relevantes" asociados a un parámetro beta, que se trata del peso de los falsos positivos (los que no aparecen en la consulta pero que son considerados relevantes).

$$\text{Query'} = \alpha \cdot \text{Query} + \beta \cdot \frac{d_1 + \cdots + d_k}{k}$$

Imagen 1: Fórmula de la Regla de Rocchio

Proceso de implementación

Para implementar este programa hemos seguido los pasos que había en el enunciado. Primero hemos reutilizado el código del fichero *SearchIndexWeights.py* para guardar una consulta dado un conjunto de palabras y pesos. A continuación, para un número de veces (parámetro nrounds), obtener los k documentos más relevantes, y sobre estos, aplicar la regla de Rocchio. Para aplicar este algoritmo, hemos generado los vectores tf-idf de los documentos del índice ayudándonos de funciones de la sesión anterior. Una vez generados, devolvemos los k documentos más relevantes de acuerdo a la consulta.

Para mejorar la eficiencia de la implementación hemos usado diccionarios, en los que hemos metido como clave los términos de la consulta y como valores, su peso. La razón detrás de esta elección radica en la mejora del coste en tiempo sumando diccionarios. Si en su lugar hubiéramos implementado el programa con vectores, el coste algorítmico sería, siendo x el tamaño de un documento, y el tamaño de otro documento, el coste de sumar 2 vectores ordenados es $O(x+y)$. Repetir esta operación k veces para los k documentos nos daría un coste de $O(k(x+y))$.

En el caso de los diccionarios, el coste de sumar dos diccionarios sería $O(y*logx)$, siendo y<x. En la práctica de Python, la operación | (merge de diccionarios) fusiona los valores de los diccionarios, y en caso de que dos valores sean iguales, sobreescribe el valor del segundo diccionario en el primero. Sabiendo que la mayoría de términos se repiten, y es $O(1)$ en comparación a x, ya que sólo se fusionan los no repetidos. Finalmente podemos afirmar que el coste final será $O(logx)$, y repetir esta operación k veces sería $O(klogx)$. Algorítmicamente $O(klogx) < O(k(x+y))$.
Queda demostrada la mejora en eficiencia.

Experimentación

Para la fase de experimentación, hemos hecho varias consultas cambiando los valores de alfa, beta, nrounds, r y k, y hemos observado la variación en los resultados. Cabe destacar que ElasticSearch funciona haciendo una AND de las palabras de la query, por lo que considerará relevantes a aquellos documentos que contengan las R palabras de la nueva consulta. En nuestro caso, hemos creado un índice a partir de una subcarpeta del conjunto de textos coloquiales, más concretamente la subcarpeta rec.autos de 20_newsgroups.

1) Alfa = 2, Beta = 0.1, nrounds = 4, k = 4, r = 4, query = "engine diesel"

```
8 Documents
['diesel^2.5388293693545863', 'engine^2.0937005548478544', 'emissions^0.34277349913788097', 'diesels^0.2875778283010771']
5 Documents
['diesel^5.579310812866354', 'engine^4.24088768434459', 'emissions^1.1554162892288122', 'diesels^0.9844010276459947']
5 Documents
['diesel^11.66027369988989', 'engine^8.53526194333806', 'emissions^2.7807018694106747', 'diesels^2.3780474263358298']
5 Documents
['diesel^23.822199473936962', 'engine^17.124010461325003', 'emissions^6.0312730297743995', 'diesels^5.1653402237155']
5 Documents
```

Imagen 2: Output 1 de Rocchio.py

Al ser alfa grande, el resultado de la nueva consulta depende de la inicial, por lo que diesel y engine se mantienen en las primeras posiciones, y al ser beta pequeña, no le damos importancia a los valores que calculan los vectores tf-idf.

2) Alfa = 0.1, Beta = 2, nrounds = 4, k = 4, r = 4, query = "engine diesel"

```
8 Documents
['diesel^10.876587387091721', 'emissions^6.855469982757619', 'diesels^5.7515565660215415', 'particulate^4.671798637563978']
5 Documents
['diesel^11.12070022185281', 'emissions^10.082932817336767', 'diesels^8.760063077478964', 'particulate^6.924231578702145']
5 Documents
['diesel^11.145111505328918', 'emissions^10.405679100794682', 'diesels^9.060913728624707', 'particulate^7.149474872815961']
5 Documents
['diesel^11.14755263367653', 'emissions^10.437953729140474', 'diesels^9.090998793739281', 'particulate^7.171999202227343']
5 Documents
```

Imagen 3: Output 2 de Rocchio.py

Habiendo invertido los valores de Alfa y Beta, podemos ver cómo ahora el término engine desaparece, ya que la query original tiene un valor ínfimo comparado con la de los vectores tf-idf. Por lo tanto, se añade a la nueva consulta un nuevo término, "particulate".

3) Alfa = 1, Beta = 0.5, nrounds = 2, k = 4, r = 4, query = "engine diesel"

```
8 Documents
['diesel^3.6941468467729304', 'emissions^1.7138674956894047', 'engine^1.4685027742392731', 'diesels^1.4378891415053854']
5 Documents
['diesel^6.202407217558839', 'emissions^4.0632139504546565', 'diesels^3.484115996724588', 'engine^1.7359356474836787']
5 Documents
```

Imagen 4: Output 3 de Rocchio.py

En este caso, siendo nrounds un valor pequeño, sólo aplicamos 2 veces el algoritmo de Rocchio, de modo que los pesos de los términos más relevantes no cambia demasiado de una iteración a otra.

4) Alfa = 1, Beta = 0.5, nrounds = 10, k = 4, r = 4, query = “engine diesel”

```
8 Documents
['diesel^3.6941468467729304', 'emissions^1.7138674956894047', 'engine^1.4685027742392731', 'diesels^1.4378891415053854']
5 Documents
['diesel^6.202407217558839', 'emissions^4.0632139504546565', 'diesels^3.484115996724588', 'engine^1.7359356474836787']
5 Documents
['diesel^8.710667588344748', 'emissions^6.412560405219908', 'diesels^5.53034285194379', 'engine^2.0033685207280842']
5 Documents
['diesel^11.218927959130657', 'emissions^8.76190685998516', 'diesels^7.576569707162992', 'engine^2.27080139397249']
5 Documents
['diesel^13.727188329916565', 'emissions^11.111253314750412', 'diesels^9.622796562382195', 'engine^2.5382342672168954']
5 Documents
['diesel^16.235448700702474', 'emissions^13.460599769515664', 'diesels^11.669023417601398', 'engine^2.805667140461301']
5 Documents
['diesel^18.743709071488382', 'emissions^15.809946224280916', 'diesels^13.715250272820601', 'engine^3.0731000137057065']
5 Documents
['diesel^21.25196944227429', 'emissions^18.159292679046168', 'diesels^15.761477128039804', 'engine^3.340532886950112']
5 Documents
['diesel^23.7602298130602', 'emissions^20.508639133811418', 'diesels^17.807703983259007', 'engine^3.6079657601945176']
5 Documents
['diesel^26.268490183846108', 'emissions^22.857985588576668', 'diesels^19.85393083847821', 'engine^3.875398633438923']
5 Documents
```

Imagen 5: Output 4 de Rocchio.py

En cambio, ahora que nrounds tiene un valor grande, se ve el incremento del peso de los valores más importantes.

5) Alfa = 1, Beta = 0.5, nrounds = 4, k = 2, r = 4, query = “engine diesel”

```
8 Documents
['diesel^3.974183615792339', 'engine^1.7072113759129832', 'particulate^1.3389398080363262', 'emissions^1.2709579181516935']
5 Documents
['diesel^6.523483857900058', 'emissions^4.236526393838978', 'particulate^3.187951923896015', 'diesels^2.433350854855268']
5 Documents
['diesel^9.072784100007777', 'emissions^7.202094869526262', 'particulate^5.036964039755704', 'diesels^4.866701709710536']
5 Documents
['diesel^11.622084342115496', 'emissions^10.167663345213548', 'diesels^7.300052564565803', 'particulate^6.885976155615392']
5 Documents
```

Imagen 6: Output 5 de Rocchio.py

Cuando la k es pequeña, se le da más importancia al peso de los términos en el conjunto de k vectores tf-idf.

¿Pero qué pasa si ponemos un valor de k grande?

6) Alfa = 1, Beta = 0.5, nrounds = 4, k = 10, r = 4, query = “engine diesel”

```
8 Documents
['diesel^2.4650243657751663', 'engine^1.2790343297019595', 'emissions^1.012364748657626', 'diesels^0.998095209946046']
5 Documents
['diesel^3.532061020142223', 'emissions^2.024729497315252', 'diesels^1.9556345723111708', 'engine^1.394723809683243']
5 Documents
['diesel^4.59909767450928', 'emissions^3.037094245972878', 'diesels^2.9131739346762955', 'engine^1.5104132896645266']
5 Documents
['diesel^5.666134328876337', 'emissions^4.049458994630504', 'diesels^3.87071329704142', 'engine^1.6261027696458101']
5 Documents
```

Imagen 7: Output 6 de Rocchio.py

Cuando k es grande, sucede el efecto contrario, el cómputo de los pesos de los términos se divide entre una k mayor, por lo que el peso de los vectores tf-idf es menor.

7) Alfa = 1, Beta = 0.5, nrounds = 4, k = 4, r = 2, query = “engine diesel”

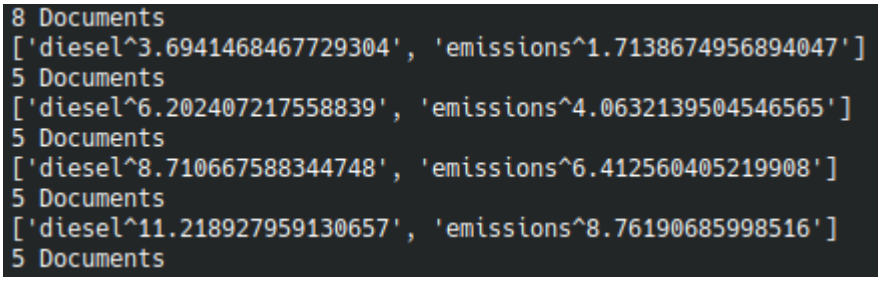

Imagen 8: Output 7 de Rocchio.py

Al ser R un valor pequeño, nos quedamos con los R términos más relevantes en la nueva consulta, por lo que es improbable encontrar nuevos términos. Sin embargo, en este caso, la palabra “engine” tiene menos importancia que la palabra “emissions”, por lo que lo sacamos del resultado.

8) Alfa = 1, Beta = 0.5, nrounds = 4, k = 4, r = 11, query = “engine diesel”

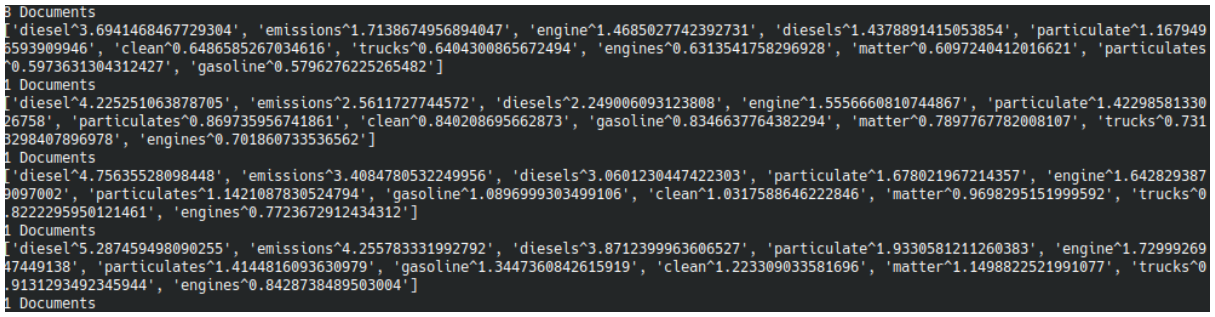

Imagen 9: Output 8 de Rocchio.py

Por último, como sabemos la forma en la que funciona ElasticSearch, podemos esperar que se recuperen un número muy reducido (llegando incluso a ser 0) de documentos relevantes cuando R comienza a tener un valor importante. En nuestro caso, se buscan las 11 palabras más relevantes en la nueva consulta.

Conclusiones

α

Si alfa es grande, la consulta final está altamente relacionada con la consulta inicial, puesto que el valor de la consulta inicial se multiplica por esta constante, y se le atribuye más prioridad. En caso contrario, el resultado final estará poco relacionado con la consulta inicial.

β

Si beta es pequeña, le damos más prioridad a la query original, y si es grande, damos más importancia a los vectores tf-idf de los k documentos, por lo que pueden aparecer nuevos términos en la nueva query, diferentes de los originales.

Nrounds

Si nrounds es pequeño, el resultado de la consulta inicial se parecerá mucho al de la consulta final, ya que habrá aplicado menos veces la regla de Rocchio y no habrá podido buscar documentos relevantes. En cambio si es grande, se itera más veces en el programa y se aplica más veces el algoritmo, por lo tanto se compararán más documentos.

R

Si la R es grande, hay más posibilidades de que la consulta no devuelva documentos relevantes, ya que la forma en que ElasticSearch lo implementa es mediante una búsqueda haciendo una operación AND entre todos los R términos.

K

El valor de k está estrechamente relacionado con el recall de la consulta, ya que cuando aumentamos su valor, aumenta el número de documentos recuperados. No obstante, disminuye la precisión, puesto que los resultados no tienen mucho que ver con la consulta inicial. Es decir, los resultados tienen más probabilidad de ser falsos positivos.