CAIM Lab, Session 3: User Relevance Feedback

Moreno Oya, Daniel
Cidraque Sala, Carles

Rocchio.py: User Relevance Feedback(URF) utilizando la regla de Rocchio

En el script Rocchio.py implementamos un pseudo URF utilizando ElasticSearch y con la ayuda de la regla de Rocchio para llevar a cabo los cálculos matemáticos. Decimos pseudo porque no vamos a estar pidiéndoles a los usuarios que seleccionen los documentos más relevantes que hay en la query en cuestión, sino que lo que hacemos es asumir que los *k (*siendo *k* un parámetro que escogemos nosotros *)*primeros documentos que devuelve la query son los más relevantes.

Para hacer este script mencionado nos hemos basado en la estructura del script SearchIndexWeight.py proporcionado para esta sesión. Concretamente hemos reutilizado el código que nos permite montar una lista a través de los términos pasados, y a partir de esta lista recuperar los *k* documentos más relevantes.

Para poder calcular los pesos de cada uno de los *k* documentos más relevantes que recuperemos necesarios para realizar la regla de Rocchio hemos utilizado el código de la sesión 2 referente al cálculo de TFIDF.

1) Para empezar lo que hacemos es recibir como parámetro la query, el conjunto de palabras que vamos a utilizar como consulta.
2) Se llama a ES para hacer la consulta original y coger los *k* documentos más relevantes del índice pasado como parámetro (--index), transformamos la query a un Diccionario (por el motivo que comentamos más adelante) y llamamos a la función TFIDFdict para obtener todos los tfidf's de los primeros *k* documentos más relevantes.
3) Calculamos el vector tf-idf de cada uno de los documentos, luego tenemos que aplanar ese vector, tenemos que juntarlos, poner los vectores de cada uno de los *k* documentos juntos.
4) A partir del parámetro *nrounds* ejecutaremos este número de veces la regla de Rocchio. Obtendremos los *k* documentos más relevantes según el parámetro que tenemos especificado y calculamos una nueva consulta aplicando la regla de Rocchio a la consulta actual un total de *nrounds*.
   En cada una de estas *nrounds* iteraciones se llama a la función Rocchio, se transforma el Diccionario de los nuevos pesos de los términos utilizando la función transfQuery, después volvemos a llamar a ES para hacer una nueva consulta y recalculamos los tfidf de los nuevos *k* documentos más relevantes.
   Acabamos el bucle y en la última iteración retornamos los *k* documentos más relevantes (--nhits).
5) Para hacer el promedio necesario en la regla de Rocchio ($\frac{(d1+d2+...+dk}{k}$) tenemos que hacer sumas de vectores con bastantes elementos, además el algoritmo hace que la suma se tenga que hacer también bastantes veces (una por iteración del bucle, es decir *nrounds*). Para ello utilizamos los Diccionarios. Siempre será mucho más eficiente buscar por un índice, en este caso por una key que es un término, que no recorriendo una determinada estructura de datos buscando un valor concreto secuencialmente.

Las nuevas funciones creadas en esta sesión para calcular la regla de Rocchio son las siguientes:

- **transfDict**: donde pasamos como parámetro la query y la transformamos a un Diccionario [word, weight] teniendo también en cuenta los posibles operandos puestos en la query de ^ y ~ .
- **transfQuery:** lo mismo que la función anterior pero a la inversa.
- **TFIDFdict**: Crea un diccionario donde guarda en cada elemento, término, la suma de los tfidf de los *k* documentos más relevantes.
- **Rocchio:** donde una vez tenemos todos los ingredientes para calcularla se los pasamos a esta función, hacemos el bucle recorriendo el diccionario creado y aplicamos la regla de Rocchio donde lo primero que hacemos es calcular el promedio ya mencionado (($\frac{(d1+d2+...+dk}{k}$)).

Experimenting

Los siguientes ejemplos están realizados sobre el corpus de notícias, indice news.

Experimento 1:
Comparamos los resultados que nos da la ejecución del script SearchIndexWeights.py con la misma query pero con nuestro script mejorado elaborado Rocchio.py. Vemos que obtenemos mejores resultados en los ejemplos a continuación :

Con SearchIndexWeights.py:
python3 SearchIndexWeights.py --index news --nhits 5 --query toronto nyc
ID= SCMVhXUBqzhnB4EQtyvi SCORE=4.9770656
PATH= ...\20_newsgroups\sci.med/0013128

ID= ViMVhXUBqzhnB4EQv0Hz SCORE=4.19135
PATH= ...\20_newsgroups\talk.politics.misc/0018667

ID= ZCMVhXUBqzhnB4EQvDYA SCORE=3.6505594
PATH= ...\20_newsgroups\talk.politics.guns/0015998

Con Rocchio.py:
python3 Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 1 --beta 1 --query toronto nyc
ID=SCMVhXUBqzhnB4EQtyvi SCORE=25.519218
PATH=...\20_newsgroups\sci.med/0013128

ID=ViMVhXUBqzhnB4EQv0Hz SCORE=21.557566
PATH=...\20_newsgroups\talk.politics.misc/0018667

ID=ZCMVhXUBqzhnB4EQvDYA SCORE=18.717743
PATH=...\20_newsgroups\talk.politics.guns/0015998

Experimento 2:
En este experimento queremos comprobar cómo cambian realmente los resultados de la ejecución del script solo variando el peso de los términos pasados en la query con el **operador de potenciación ^**. Procedemos de la misma forma que se nos invita a hacer al inicio de la sesión con el script SearchIndexWeights.py.

Vemos como desde un inicio ya le damos el doble de peso al término que le aplicamos la potenciación de 2. si comparamos la segunda ejecución con la tercera vemos que obtenemos ligeramente mejor score con la segunda, science es una palabra que seguramente aparezca más que computer, y en un mismo documento obtenemos mejor score otorgándole más peso a science que con la tercera ejecución otorgándole a computer.

python3 Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 2 --beta 1 --query computer science

python3 Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 2 --beta 1 --query computer science^2
Resp:
i = 1, q = ['computer^2.1042028408694766', 'science^4.165506576745799']
i = 2, q = ['computer^4.31260852260843', 'science^8.496519730237399']
i = 3, q = ['computer^8.729419886086337', 'science^17.158546037220596']
i = 4, q = ['computer^17.56304261304215', 'science^34.48259865118699']
ID=RyIVhXUBqzhnB4EQq_-z SCORE=533.9905
PATH=...\20_newsgroups\comp.graphics/0001677

python3 Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 2 --beta 1 --query computer^2 science
Resp:
i = 1, q = ['computer^4.188115536704749', 'science^2.0989023987031805']
i = 2, q = ['computer^8.564346610114248', 'science^4.2967071961095415']
i = 3, q = ['computer^17.316808756933245', 'science^8.692316790922265']
i = 4, q = ['computer^34.82173305057124', 'science^17.48353598054771']
ID=RyIVhXUBqzhnB4EQq_-z SCORE=527.1494
PATH=...\20_newsgroups\comp.graphics/0001677

Experimento 3:
Miramos si el cambio únicamente del parámetro ***nrounds*** afecta demasiado en la calidad del resultado obtenido. Nos fijamos en el que tiene score más alto, vemos que como más iteraciones hacemos, más score obtenemos.

Con *nrounds = 1*
ID=SCMVhXUBqzhnB4EQtyvi SCORE=10.017521
PATH=...\20_newsgroups\sci.med/0013128
Con *nrounds = 25*
ID=SCMVhXUBqzhnB4EQtyvi SCORE=143.44342
PATH=...\20_newsgroups\sci.med/0013128
Con *nrounds = 50*
ID=SCMVhXUBqzhnB4EQtyvi SCORE=326.5052

PATH=...\20_newsgroups\sci.med/0013128

Experimento 4:
Vemos como afecta la variación de los parámetros **R** y **nhits**.
Si variamos únicamente el valor de R, vemos que obtenemos menos *recall* pero aumentamos la *precision*. Concuerda con lo visto a teoria dónde en general son inversamente proporcionales.

Hacemos también la prueba de solo ir modificando y aumentando el valor de *nhits*. Vemos que el *recall* aumenta, però la precision vemos que baja:

python3 Rocchio.py --index news --nhits 60 --nrounds 5 -R 5 --alpha 1 --beta 1 --query computer science
ID=RyIVhXUBqzhnB4EQq_-z SCORE=62.340145
PATH=...\20_newsgroups\comp.graphics/0001677
python3 Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 1 --beta 1 --query computer science
ID=RyIVhXUBqzhnB4EQq_-z SCORE=66.70268
PATH=...\20_newsgroups\comp.graphics/0001677

Experimento 5:
Los parámetros **alfa** y **beta** ayudan a dar importancia a diferentes partes de la fórmula de Rocchio. Como más peso le demos a alfa más importancia tendrá la query original, y si va acompañada de una beta más bien baja la regla de Rocchio en sus iteraciones no mejorarán mucho los resultados respecto a la query original. Beta en cambio ayuda a dar importancia a la query formada por tdfidf, podemos ayudar más a la evolución de la query original. En el siguiente ejemplo vemos como aumentando alfa a 5 y dejando beta a 1 y a la inversa vemos como en el primer caso obtenemos mucho mejor score, en el segundo la query se ve demasiado alterada respecto a la query original (con alfa = 1) y perdemos mucho score. Por lo tanto, no es nada interesante tener valores muy dispares de alfa y beta si queremos sacar provecho de la aplicación de la regla de Rocchio.

python3 Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 5 --beta 1 --query computer science
ID=RyIVhXUBqzhnB4EQq_-z SCORE=8535.729
python3 Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 1 --beta 5 --query computer science
ID=RyIVhXUBqzhnB4EQq_-z SCORE=125.67557