CAIM - Lab3
User Relevance Feedback
Rocchio

Pau Núñez Amorós
Víctor Vallejo Rives

21/10/2019

1 User Relevance Feedback amb Rocchio

Per tal de dur a terme la pràctica hem partit del fixer base `SearchIndewWeights.py` proporcionat al `zip`. Hem afegit diverses de les funcions necessàries a la sessió anterior (amb alguna modificació) referents al càlcul de `TFIDF`. A més hem hagut de crear noves funcions com la suma dels TFIDF que ens permet calcular la `Rocchio's rule`.

Al desenvolupar aquest procés ens hem trobat amb alguna dificultat extra. Per començar, quan hem creat les noves funcions, volíem treballar amb diccionaris però les funcions que vam utilitzar la pràctica anterior ho tractaven tot com a llistes, i per tant hem hagut de fer les modificacions adients per tal de fer servir també diccionaris. L'altre gran inconvenient que ens hem trobat ha sigut a l'hora de veure els nous pesos un cop aplicada la fórmula de la `Rocchio's rule`. Quan imprimíem les noves `queries` ens indicava que el pes de qualsevol paraula era `NaN`. Això volia dir que en algun punt del nostre procés, els pesos deixaven de calcular-se bé. Per tal d'identificar quin era aquest punt hem hagut d'anar debugant funció rere funció començant per la que ens dona el resultat final i saltant cap enrere. Seguint aquest criteri hem arribat fins a la funció `normalize` ja utilitzada a la sessió anterior, i ens hem adonat que hi havia un problema a l'hora de fer una divisió i a partir d'aquell moment el resultat era considerat `NaN`. La solució ha sigut poc intuïtiva, ja que tot i que a l'imprimir el tipus del numerador i denominador ens deia que es tractava de dues variables `float`, no ha sigut fins que hem fet el `cast float()` com a tal que no ha començat a funcionar correctament. Aquest error ve del canvi comentat anteriorment, ja que quan fèiem un `append()` d'aquest mateix a càlcul a la pràctica anterior sense fer cap `cast`, no ens trobavem amb aquest problema.

2 Experimentació

2.1 Experiment 1

En aquest experiment volem comprovar com canvia el resultat que ens dóna ElasticSearch quan toquem els pesos d'un terme de la *query*. Fixem la resta de paràmetres durant l'experiment per tal que no intervinguin en l'execució. (`nhits=5, nrounds=5, R=5, alpha=1, beta=1`).

```
Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 1 --beta 1 --query toronto nyc

SCORE=45.729 | PATH=/alt.atheism/0000574
SCORE=24.427 | PATH=/sci.med/0013128
SCORE=20.273 | PATH=/talk.politics.misc/0018667
SCORE=17.930 | PATH=/talk.politics.guns/0015998


Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 1 --beta 1 --query toronto nyc^2

SCORE=72.947 | PATH=/alt.atheism/0000574
SCORE=38.966 | PATH=/sci.med/0013128
SCORE=28.602 | PATH=/talk.politics.guns/0015998
SCORE=26.910 | PATH=/talk.politics.misc/0018667
```

Com podem veure, el fet de donar més pes a `nyc`, fa que les puntuacions canviïn i els documents es reordenin (`politics.misc i politics.guns` intercanvien posicions). Això es deu a que `guns` és un document més curt que `misc` i llavors afovereix que tingui una *score* més alta tot i que el terme`nyc` aparegui en ambdós documents.

2.2 Experiment 2

En aquest experiment comprovarem com Rocchio reassigna els pesos a cada terme de la nova *query*.

```
Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 2 --beta 1 --query toronto nyc^2
```

```
i = 0, q = ['toronto', 'nyc^2']
i = 1, q = ['toronto^2.0', 'nyc^4.0']
i = 2, q = ['toronto^4.0', 'nyc^8.0']
i = 3, q = ['toronto^8.0', 'nyc^16.0']
i = 4, q = ['toronto^16.0', 'nyc^32.0']
```

Veiem que a cada iteració de Rocchio el pes de cada terme incrementa de forma exponencial en α, partint del seu pes inicial en la primera iteració. Per $\alpha = 3$ l'increment exponencial és cúbic, lògicament, i així successivament. Podem comprovar, en canvi, que β no té cap efecte sobre els termes originals perquè aquest terme afecta a l'aparició de nous termes a la `query`.

2.3 Experiment 3

En aquest experiment, cercarem paraules molt més freqüents a la nostra consulta, per tal de tenir un *pool* de documents molt més extens:

```
python Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 2 --beta 1 --query is was
python Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 2 --beta 1 --query is was^2
python Rocchio.py --index news --nhits 5 --nrounds 5 -R 5 --alpha 2 --beta 1 --query is^2 was
```

Entre aquestes línies hi ha una diferència notable en l'ordre dels documents més rellevants i en les puntuacions assignades a un perquè les paraules base són molt genèriques i el refinament de la *query* que fa Rocchio a cada iteració divergeix considerablement entre una execución i una altra. Això es deu també molt probablement a una `R` relativament alta que permet que nous termes molt variats (degut a la varietat de documents, tenim més de 6735 candidats) s'afegeixin a la nova *query*.

3 Conclusions

A partir de les observacions i experiments realitzats, hem pogut obtenir algunes conclusions.

Si ens centrem en la fórmula de la `Rocchio's rule`, els paràmetres α i β ponderen positivament la conservació de la *query* original i la importància de la nova obtinguda per `TFIDF` respectivament. Per tant, tenint en compte això, veiem que són dos paràmetres contraris per definició. En quant a `R`, com més gran es defineix menys documents finals s'obtenen. Això és degut al fet que s'està augmentant el nombre de termes nous, i per tant els documents finals que compleixin tots els requisits de paraules serà menor a la força. És a dir, seran més `precisos` però amb menys `recall`. En canvi, quan parlem de `k`, si augmentem el seu valor, obtindrem més documents tot i que aquests poden ser poc rellevants, disminuint així la `precisió` però augmentant el `recall`. En quant a `nrounds`, en determinar el nombre de vegades que s'aplica la `Rocchio's rule` i tenint en compte que cada cop que apliquem le regla augmentem el grau de modificació de la query respecte a l'original, hem de vigilar que no sigui massa gran. Ja que tot i que això ens ajudarà en termes de `recall`, igual que abans ens farà sacrificar `precisió`.