Cerca i Anàlisi d'Informació Massiva
Pseudo-Relevance Feedback

Marc Canals Riba Joaquin Figueira Chacón

11 de novembre de 2021

Índex

1 Implementació

Implementem un algorisme de Pseudo-Relevance Feedback, en què s'aplica la fórmula de Rocchio. Aquesta fórmula permet introduir nous termes a la consulta i redistribuir-ne els pesos per tal que els documents de la resposta definitiva continguin els termes dels documents més similars i, per tant, la resposta no contingui documents irrellevants i sigui més precisa.

```
def expand_query(client, index, documents, alpha_weights, alpha, beta, R):
    beta_weights = document_weight_vector_sum(client, index, documents)
    n = len(documents)
    weights = dict()
    for term in alpha_weights:
        weights[term] = alpha * alpha_weights[term]
    for term in beta_weights:
        alpha_weight = weights.get(term, 0.0)
        weights[term] = alpha_weight + beta * beta_weights[term] / n
    query = dict()
    for term in nlargest(R, weights, key = weights.__getitem__):
        query[term] = weights[term]
    return query
```

Figura 1: Codi de la funció que aplica la fórmula de Rocchio.

En essència, sumem els pesos dels termes dels documents, d'un en un —per a reduir l'ús de memòria—, en un diccionari, a continuació calculem la nova consulta, també en un diccionari i n'excloem els termes de menor pes. En comptes d'ordenar els parells de termes i pesos en temps $\mathcal{O}(n \log n)$ —on n és el nombre de termes— i després agafar-ne els R primers, emprem la funció `nlargest` que utilitza un *maxheap*, on hi desa els R termes de major pes i permet obtenir, suposant que $R \ll n$, els termes de major pes amb temps $\mathcal{O}(n)$.

Analitzem el cost temporal de sumar els vectors de pesos dels k documents segons les estructures de dades utilitzades. Si representem la suma dels vectors $v_1, \ldots, v_k$ de parells de termes i pesos, ordenats per terme, amb una llista, en podem fer la suma mitjançant un algorisme de fusió de llistes, amb el qual a cada iteració visitem l'element mínim dels elements mínims de cadascuna de les k llistes; en el cas pitjor cal visitar-los tots i el cost és $\sum_{i=1}^{k} (k \cdot (|v_i| - 1) + i)$ —sigui i el nombre de llistes en les quals sols hi resta per a visitar el darrer element, per a cadascun dels elements restants ($1 \leq i \leq k$) només fem el mínim amb i elements—. Ara bé, en comptes de fer el mínim per a cada element visitat, podem tindre l'element mínim de cadascuna de les k llistes en un *minheap*; llavors, per exemple, amb un *binary minheap*, el cost es redueix a $\sum_{i=1}^{k} (log_2\ i + log_2\ k \cdot (|v_i| - 2) + 1)$ o, de manera equivalent, $log_2\ k! + log_2\ k \cdot \sum_{i=1}^{k} (|v_i| - 2) + k$ —sigui i l'i-èsim element inserit al *heap*, a l'hora d'inserir els primers elements al *heap* ($1 \leq i \leq k$), aquest conté i elements i per als darrers element de cada llista només consultem l'element mínim del *heap*, però no n'inserim cap—.

Si representem la suma dels vectors amb un diccionari, podem fer la suma visitant tots els elements i el cost mitjà és $\sum_{i=1}^{k} |v_i|$.

Per tal de simplificar les expressions, podem substituir la longitud de cadascuna de les llistes per la longitud mitjana n, llavors, el primer cost esdevé $log_2\ k! + log_2\ k \cdot k \cdot (n - 2) + k$ i el segon $k \cdot n$.

Tot i que el cost temporal de l'algorisme de fusió de llistes ordenades no en fa, aparentment, inviable l'ús, cal tenir en compte el cost espacial: l'algorisme pot intercalar els accessos a les llistes, per tant, cal que totes les llistes siguin accessibles alhora —podríem generar-les progressivament, però no és fàcil implementar-ho—.

2 Estudi dels paràmetres

Per a estudiar el comportament dels paràmetres realitzem consultes amb un terme aleatori, a més de consultes el·laborades manualment. Com que no disposem d'un conjunt realista de documents rellevants per a les consultes és difícil de mesurar la precisió i el reclam de les respostes; sí que podem mesurar altres quantitats en què basar l'anàlisi quantitatiu, com ara el nombre de documents de la resposta i la distància d'edició de Levenshtein entre dos resultats. Deixem oberta la possibilitat d'indexar documents d'un cercador com ara Google i definir els conjunts de documents rellevants per a una consulta segons els documents de la resposta del cercador. També es podria mesurar el temps d'execució.

2.1 Estudi del nombre màxim de termes (R)

Cerquem termes aleatoris aplicant la fórmula de Rocchio un sol cop ($r = 1$), limitant el nombre de documents al límit del servidor ($k = 10000$), amb $\alpha = 1$, amb $\beta = 0.1$ i amb el nombre màxim de termes (R) variable.

Observem que el nombre de documents de la resposta disminueix ràpidament quan R augmenta; no és gens estrany: com més gran és R, més termes conté la consulta, més restrictiva esdevé i menys documents conté la resposta. En principi, com que descartem documents que no contenen els termes que contenen els documents més semblants, com més gran és el nombre de termes més augmenta la precisió i més disminueix el reclam —si el nombre de documents de la resposta disminueix, també ho fa el nombre de documents rellevants de la resposta—.

Podríem trobar-ne un valor òptim al rang de valors entre 3 i 5, tot i que caldria fer-ne un anàlisi més rigorós que ens permetés, per exemple, poder afirmar que amb una confiança determinada el nombre de documents d'una resposta es troba entre un mínim i un màxim establerts.

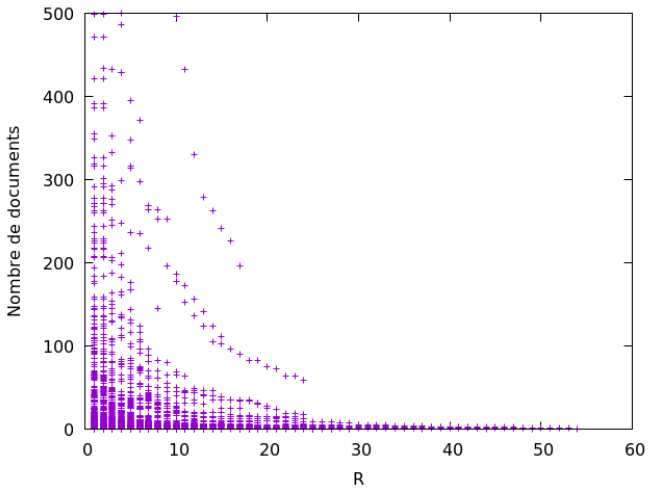

Figura 2: Nombre de documents de la resposta a una consulta en funció del nombre màxim de termes a la consulta.

2.2 Estudi del nombre d'aplicacions (r)

Cerquem termes aleatoris aplicant la fórmula de Rocchio un nombre de vegades (r) variable, limitant el nombre de documents al límit del servidor ($k = 10000$), amb $\alpha = 1$, amb $\beta = 0.1$ i amb un nombre màxim de termes raonable ($R = 4$).

Observem que la distància d'edició de Levenshtein, una mesura de la diferència entre dues seqüències, entre una resposta i l'anterior disminueix abruptament quan r passa de 1 a 2 i esdevé gairebé 0, és a dir, la resposta no varia.

Per tant, com que els resultats no milloren gaire si s'aplica la fórmula més de dues vegades, podem considerar $r = 1$ òptim.

De fet, en general amb $r > 1$ no canvia quins documents apareixen a la resposta, sinó en quin ordre hi apareixen. L'ordre canvia quan s'han redistribuït prou els pesos dels termes, sovint després de bastantes aplicacions de la fórmula.

Respecte a la precisió i al reclam, en principi, la precisió augmenta i el reclam disminueix, tot i que gairebé gens després de la primera aplicació.

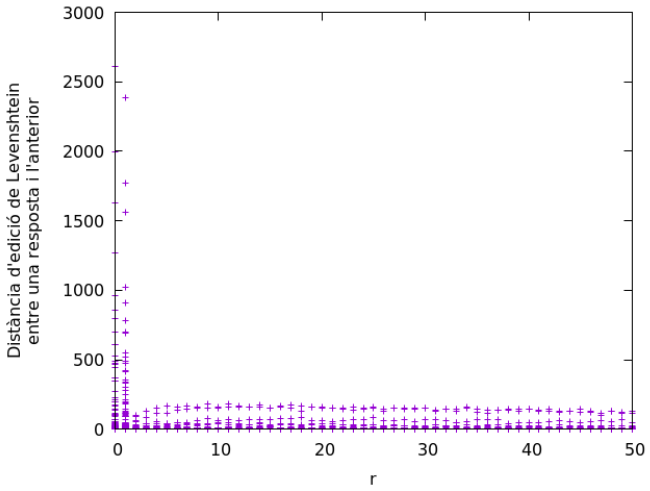

Figura 3: Distància d'edició de Levenshtein entre una resposta i l'anterior en funció del nombre d'aplicacions de la fórmula de Rocchio.

2.3 Estudi del nombre màxim de documents (k)

Cerquem termes aleatoris aplicant la fórmula de Rocchio un sol cop ($r = 1$), variant el límit del nombre de documents (k), amb $\alpha = 1$, amb $\beta = 0.1$ i amb un nombre màxim de termes raonable ($R = 4$).

No observem resultats clars. Sembla que el nombre de documents de la resposta final augmenta lleugerament fins a convergir, segurament perquè amb valors petits de k incloem menys documents que els que coincideixen amb la consulta inicial i en haver inclòs tots els documents possibles la resposta final només en pot contenir menys o els mateixos que la inicial —també per aquesta raó, es veu clarament una diagonal a les gràfiques—. És raonable de pensar que el nombre de documents de la resposta final està més relacionat amb R que amb k. Caldria estudiar el comportament d'aquest paràmetre amb valors diferents de R.

En alguns casos, quan el valor de k és petit, trobem un clúster de documents prou gran com perquè els termes més rellevants provinguin d'aquests documents i el resultat els inclogui, en canvi, quan el valor de k és massa gran n'hi trobem més d'un i els termes més rellevants són massa diversos i el resultat conté menys documents. En alguns casos l'augment de k canvia completament la resposta, com ara en el cas del terme *harmoniously*: amb $r = 48$ molts dels documents similars són sobre religió, els termes més rellevants també ho són i, per tant, la resposta final conté documents sobre religió; amb $r = 49$ apareix un document de criptografia que fa que un dels termes més rellevants canviï a *encrypt* i la resposta final sigui buida, perquè no hi ha documents amb tots aquests termes, tant diversos.

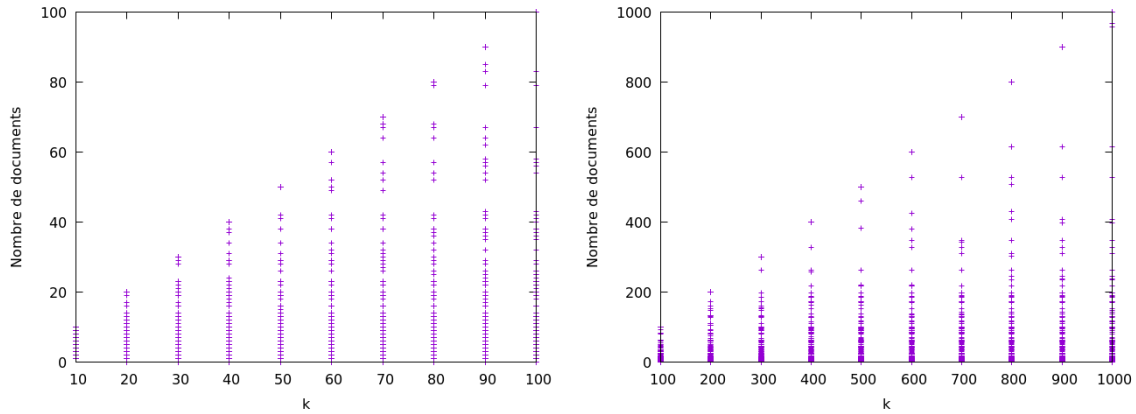

Figura 4: Nombre de documents de la resposta final en funció del nombre màxim de documents de la resposta inicial. A l'esquerra per a $10 \leq k \leq 100$ i a la dreta per a $10 \leq k \leq 100$.

2.4 Estudi del pes α

El paràmetre α determina la importància de la *query* original en el *pseudo-relevance feedback loop*. Per això el que ens interessa especialment amb aquest paràmetre es veure la semblança de la *query* final produïda pel *feedback loop* amb la *query* original. Teòricament mentre major sigui α, més s'assemblarà o relacionarà la *query* final amb la original.

Per tal de comprovar el comportament abans explicat experimentalment hem fet una sèrie d'execucions de l'*script* fixant la resta de paràmetres amb els següents valors seleccionats arbitràriament: $r = 10$, $\beta = 0.5$, $k = 5$, $R = 4$ i $query = toronto^3 detroit^5$.

Per a les cinc execucions de l'*script* què hem fet hem utilitzat els valors d'α següents: 0.01, 0.5, 1.0, 5.0 i 10. Els resultats es resumeixen a la següent taula:

α	*Query* final
0.01	$0^{0.2946}$, $1^{0.2784}$, $2^{0.0939}$, $pp^{0.0789}$
0.5	$1^{0.4847}$, $0^{0.4432}$, $detroit^{0.1058}$, $2^{0.0824}$
1.0	$detroit^{5.8671}$, $toronto^{3.7934}$, $1^{1.5220}$, $0^{1.1126}$
5.0	$detroit^{4913}$, $toronto^{2957}$, 1^{2886}, 0^{2059}
10.0	$detroit^{50144015427.6517}$, $toronto^{30132615368.1150}$, $1^{125997420.5381}$, $0^{89478380.6197}$

Tal com havíem dit abans, sembla ser que la tendència en els resultats és que mentre major sigui la α més similar és la *query* final a l'original (si bé amb els pesos escalats uns quants ordres de magnitud per la multiplicació successiva que pateix en el bucle). A més, com la *query* original només té dos termes i el programa pot fabricar *queries* de quatre termes amb els paràmetres seleccionats, s'afegeixen dos termes addicionals. També s'ha de dir que un cop arribats a certa alfa el *query* resultat convergeix als mateixos termes amb pesos escalats. Com també observem aquest comportament amb altres *queries*, sembla ser que aquest efecte es una tendència general del *relevance feedback-loop* amb la regla de Rocchio. Per últim, és important destacar que quan α és molt similar a β o inferior com és el cas de 0.01 o 0.5 la regla de Rocchio i la poda del termes que fa l'*script* fa que la *query* final quasi no tingui res a veure amb l'original.

2.5 Estudi del pes β

Pel paràmetre β hem fet la mateixa experimentació que amb el paràmetre anterior, però aquesta vegada variant β i fixant la resta de paràmetres. Com que aquest paràmetre controla el pes dels nous termes en el *feedback-*

loop, amb un major valor de β hauríem d'observar *queries* menys relacionades amb la *query* original i amb més termes nous amb majors pesos. Dit això els valors de beta que hem utilitzat en les execucions són: 0.1, 0.5, 0.7, 1.0 i 1.5. Per altre banda el valor de la resta de paràmetres utilitzats va ser: $r = 10$, $\alpha = 1$, $k = 5$, $R = 4$ i $query = toronto^3 detroit^5 gilmour^{0.5} team^{1.5}$ (hem escollit una *query* amb més termes per veure més clarament l'aparició de termes diferents en la *query*.

Tal com podem observar en la taula de sota els resultats no són el que esperàvem. Per un costat els pesos han canviat molt poc, i per altre la *query* final en tots els casos segueix tenint tots els termes de la *query* original. Pensem que la causa d'això és que tots els termes de la *query* són també molt freqüents en els documents cercats i per això mai son reemplaçats per termes nous (és a dir, en totes les iteracions són sempre els R termes amb major pes). A més, observem que els seus pesos creixen una mica amb el creixement de β ja que en cada iteració sumem una fracció d'aquest paràmetre als seus pesos com indica la regla de Rocchio.

β	*Query* final
0.1	$detroit^{5.2203}$, $toronto^{3.1986}$, $team^{1.6420}$, $gilmour^{0.6749}$
0.5	$detroit^{6.1015}$, $toronto^{3.9933}$, $team^{2.21019}$, $gilmour^{1.3745}$
0.7	$detroit^{6.5421}$, $toronto^{4.3906}$, $team^{2.4942}$, $gilmour^{1.7244}$
1.0	$detroit^{7.2030}$, $toronto^{4.9866}$, $team^{2.9203}$, $gilmour^{2.2491}$
1.5	$detroit^{8.3046}$, $toronto^{5.9799}$, $team^{3.6305}$, $gilmour^{3.1237}$