Detecció de similitud de documents amb hashing

Arnau Farràs
David Marín
Xavi Lopez

1. Implementació

Main

El programa rep com a entrada dos documents, cada un es emmagatzemat en un vector de paraules i s'envien per referència a cada un dels algorismes calculant cada cop el temps d'execució.

Algorisme Jaccard amb k-shingles

L'algorisme rep com a paràmetre dos vectors de paraules (un per cada document) i el parametre "**k**" que determina el tamany de cada *set* de paraules dins l'algorisme.

Primer de tot, recorrem el vector del primer document de tamany "**n**" i creem "**n**" strings formats per "**k**" paraules, que seran emmagatzemats en un *map*, on la clau és l'string, i li posem valor "-1".

A continuació recorrem el vector del segon document de la mateixa manera, pero aquesta vegada, en el cas de que la clau ja es troba en el *map,* actualitzem el valor a "0". En el cas contrari, posem el valor "1".

D'aquesta manera, al final de l'execució en la taula de hash, els *sets* que apareguin unicament en el primer document tindran el valor "-1", els que apareixen en el segon document tindran el valor "1", els que apareixen en els dos documents tindran el valor "0".

El resultat a retornar serà la quantitat de elements "1" entre el nombre d'elements totals.

Algorisme Jaccard amb minhash

L'algorisme Jaccard amb minhash permet calcular una aproximació a la similitud de Jaccard de forma eficient. La nostra implementació rep com a paràmetre una instància de HashFunctions, dos vectors de shingles, i el paràmetre t que determina el nombre de funcions de hash que s'utilitzaran per fer el càlcul de la similitud de Jaccard.

Una vegada s'han enmagatzemat i inicialitzat totes les variables necessàries, l'algorisme comença a calcular el minhash de cada document mitjançant t diferents funcions de hash diferents, i compara els seus valors: Si el hash més petit de ambdos documents representa la mateixa paraula, diem que s'ha produït una intersecció.

Per a obtenir el valor final només cal calcular la divisió entre el nombre de interseccions i el nombre de funcions de hash que s'han utilitzat.

Algorisme Locality-Sensitive Hashing

Degut a que la primera versió del nostre algorisme de LSH no ens acabava de donar els resultats que ens esperàvem, varem implementar una segona versió del algorisme.

PRIMERA IMPLEMENTACIÓ

La primera implementació d'aquest algorisme reb com a paràmetres una instància de HashFunctions, els dos documents a comparar, i dos enters k y b, que serán el nombre de hash functions a realitzar als documents en primera instància i el nombre de bands en el que dividirem els minhash dels documents.

Aquesta versió computarà la similitud de dos documents de la següent manera:
Primerament la funció rebrà dos arxius i generarà els k-shingles de cada document mediant els enters k1 y k2 (que normalment prendran el valor 5 per arxius menors a 9 paraules o 9 per als documents majors). Una vegada ha generat aquets k-shingles, calcularà els k minhashes a cada document, i els emmagatzemarà en dos vectors de pairs per a conserver el seu valor string de cada shingle.

Després, l'algorisme agafarà quets k minhashes i els separarà en bukets, amb b bands i k/b rows per band.
un cop realitzada aquesta tasca, l'algorisme podrà computar els minhash d'aquets buckets, tot finalitzant el Locality Sensitive Hashing al qual finalitzarà calculant la similitud dels diferents buckets dels dos arxius d'entrada.

SEGONA IMPLEMENTACIÓ

La segona implementació del algorisme LSH reb com a paràmetres una instància de HashFunctions, els dos documents, i les variables k, t i b, que representen el tamany dels shingles, el nombre de funcions de hash a utilitzar, i els bancs en que es dividirà la signature matrix.

Per a calcular la similitud, primer de tot aplica el k-shingle als dos documents. Una vegada s'han convertit en vectors de shingles, s'obté la signatura minhash de cada vector per les t funcions de hash, de manera que obtenim un nou vector per cada document amb t valors de hash. Amb aquests vectors, construim una signature matrix, la qual utilitzarem per obtenir el bucket map: Dividim la signature matrix en b bancs, de manera que cada banc tindrà t/b files, cadascuna representant una funció de hash diferent, per cada columna, que representa els documents. Una vegada dividida, s'aplica una funció de hash amb col·lisió per proximitat a les columnes dels bancs, que provocarà que vectors similars vagin a parar al mateix bucket. Aquestes funcions de hash han de donar valors unics per a cada banc. Si dos o més hash arriben al mateix bucket, significarà que hem obtingut una intersecció.
Finalment calculem la similitud de Jaccard dividint el nombre de interseccions obtingudes amb el nombre de buckets als que ha anat a parar algun hash.

1. Experiments

Per a la realització dels experiments, hem fet servir un conjunt de llibres i documents en format txt extretes d'Internet (veure carpeta "novels").
També hem utilitzat 20 documents de 50 paraules permutades aleatoriament (veure carpeta "randomDocs").

Prova 1
Documents: *DarwinOriginofSpecies.txt* y *DarwinOriginofSpecies2.txt*
El primer document es un llibre sencer, per tant, la quantitat de text es considerable, el segon document es una copia del primer amb la diferencia de que hem desordenat algun paràgraf i també hi hem afegit algun fragment de text d'un altre document, no obstant això, els canvis que li hem fet son molt petits en comparació a la mida del document.

Funcions de hash: 200
shingles: 9
bands:3

	Jaccard-shingles	Jaccard-minhash	LSH(V1)	LSH(V2)
similitud	0.937	0.94	0.666	0.0226
temps(s)	0.34	3.62	5.32142	0.026

Funcions de hash: 200
shingles: 9
bands:3

	Jaccard-shingles	Jaccard-minhash	LSH(V1)	LSH(V2)
similitud	0.939	0.94	0.666	0.0226
temps(s)	0.63	3.63	27.66	0.076

Funcions de hash: 200
shingles: 9
bands:50

	LSH(V1)	LSH(V2)
similitud	0.64	0.56
temps(s)	28.97	0.076

Els algorismes de Jaccard-shingles i Jaccard-minhash ens han donat el resultat que esperàvem (una similitud del 94%), no obstant això, els de LSH bastant del resultat correcte, creiem que és degut a un error en la implementació, podem observar, que al canviar les shingles, la similitud es manté però el temps d'execució augmenta considerablement, sobretot en el cas de LSH(V1), això es normal ja que incrementant les shingles, el numero de strings a comparar augmenta considerablement, i en el cas de documents grans com es el cas, la diferencia es bastant gran.

Pel que fa al nombre de bands, en la primera implementació no sembla variar gaire el resultat, en el cas del segon, creiem que es normal l'increment degut a com que te moltes mes bands, tindrà també mes oportunitats de trobar coincidencies.

Prova 2

Documents: *DickensGreatExpectations.txt* y *KiplingJungleBook.txt*
Aquesta vegada hem agafat dos llibres que no guarden cap relació entre ells, al ser diferents, no creiem que la similitud sigui gaire alta.

Funcions de hash: 200
shingles: 2
bands:50

	Jaccard-shingles	Jaccard-minhash	LSH(V1)	LSH(V2)
similitud	0.071	0.31	0.08	0.27
temps(s)	0.34	2.81	4.8	0.019

Funcions de hash: 200
shingles: 9
bands:50

	Jaccard-shingles	Jaccard-minhash	LSH(V1)	LSH(V2)
similitud	0.012	0.31	0.02	0.64
temps(s)	0.577	2.68	20.725	0.058

Podem apreciar, uns resultats bastant coherents en el cas de Jaccard-shingles i LSH(V1), pel que fa a Jaccard-minhash, creiem que una similitud del 31% es un resultat massa elevat tenint en compte que son dos documents totalment diferents. Podem apreciar també que al augmentar el nombre de shingles estem perdent part de la similitud, cosa que es normal perquè amb shingles petites es mes fàcil que coincideixin fragments mes petits.

Prova 3

En aquest cas, farem servir els documents doc1.txt -....- doc20.txt que contenen tots 50 paraules permutades de manera aleatoria.

Documents: doc1.txt, doc1.txt
Funcions de hash: 200
shingles: 2
bands:50

	Jaccard-shingles	Jaccard-minhash	LSH(V1)	LSH(V2)
similitud	1	1	1	1
temps(s)	0.00007	0.0016	0.0048	0.00038

En aquest experiment estem agafant dos documents identics, per tant, la similitud resultant es de 1.

Documents: doc5.txt, doc10.txt
Funcions de hash: 200
shingles: 2
bands:50

	Jaccard-shingles	Jaccard-minhash	LSH(V1)	LSH(V2)
similitud	0.009	0.905	0.007	0.7
temps(s)	0.000164	0.0026	0.0078	0.000565

Documents: doc5.txt, doc10.txt
Funcions de hash: 200
shingles: 2
bands:50

	Jaccard-shingles	Jaccard-minhash	LSH(V1)	LSH(V2)
similitud	1	1	1	0.2
temps(s)	0.00007	0.0026	0.0034	0.0004

En aquests dos casos en canvi, podem veure que si agafem dos documents que tenen les mateixes paraules, cuan el numero de *shingles* es 1, encara que aquestes estiguin desordenades, a la hora de compararlos tots els elements d'un document estaran en l'altre i per tant la similitud serà 1, a mida que incrementem el numero de *shingles* la similitud anirà disminuint.

2. Conclusions

En algunes ocasions, hem obtingut valors que diferien bastant del que nosaltres ens esperavem, es probable que sigui degut a algun error en la implementació, no obstant això es pot apreciar el paper que tenen les variables *shingles* i *bands*.

En documents molt petits com ara els de 50 paraules, no te gaire sentit fer servir un nombre de *shingles* gaire gran ja que en aquests casos normalment voldrem comparar el nombre de paraules coincidents. En documents mes grans com es el cas de llibres si que ens interessa fer servir un nombre mes gran ja que només amb les *shingles* a nivell de paraula no podem acabar de saber realment si els documents son similars de veritat.

Pel que fa a les *bands* hem arribat a la conclusió de que a un nombre mes elevat de *bands* mes posiblitats hi haurà de que dos documents coincideixin ja que les signatures seràn fraccionades mes vegades i tindran mes oportunitats de coincidir.

3. Bibliografía

- https://en.wikipedia.org/wiki/Jaccard_index
- https://en.wikipedia.org/wiki/W-shingling
- https://en.wikipedia.org/wiki/N-gram
- http://infolab.stanford.edu/~ullman/mmds/ch3.pdf
- https://en.wikipedia.org/wiki/MinHash