UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

CAIM Laboratori 2: Programant amb ElasticSearch

Genís Gutiérrez Bernal

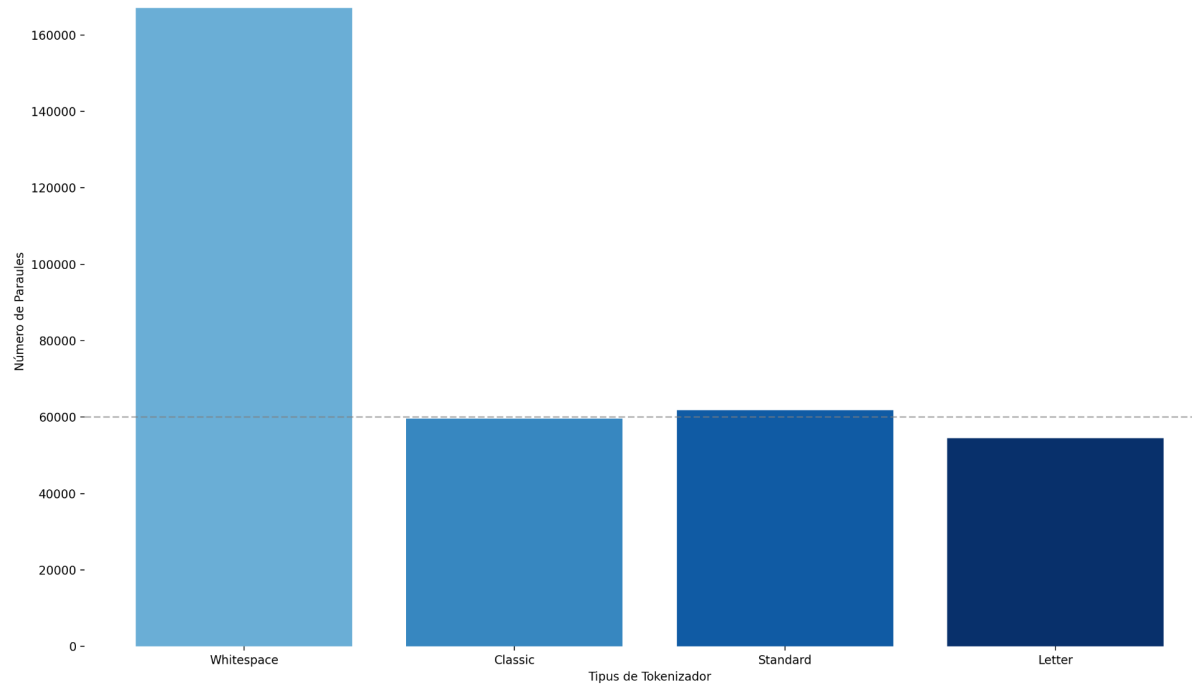
Marc Ramos Fernández

1. Modificant els índexs d'ElasticSearch

A la última pràctica, a l'hora d'indexar els diversos documents de cada col·lecció, ho fèiem amb un programa que insertava totes les paraules trobades en el text, independentment de les seves característiques. Per això, més endavant érem nosaltres els que havíem de fer aquest filtratge per eliminar paraules i/o caràcters que no ens aportaven res en el nostre experiment, com per exemple, eliminar números i stopwords.

En aquesta part, utilitzarem diferents tokenitzadors i filtres que ens proporciona ElasticSearch i que realitzen gran part d'aquesta tasca. De forma que podrem comparar-los entre ells, veure el seu comportament i quins són els més restrictius.

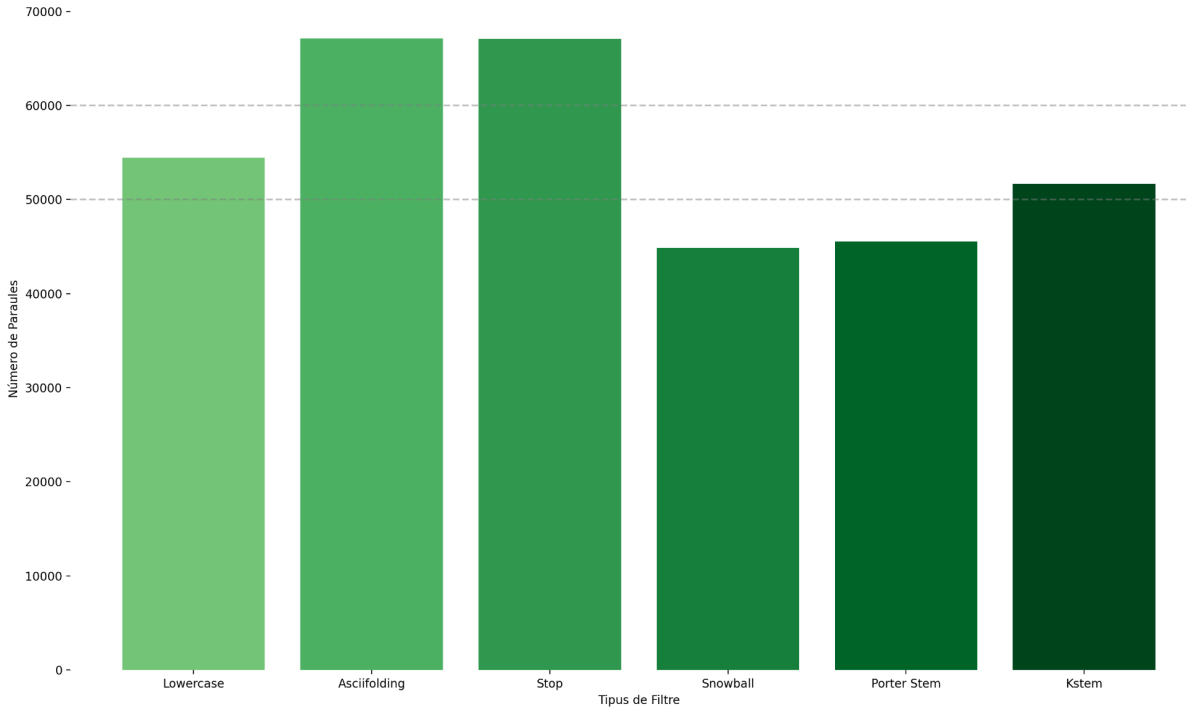
Per fer-ho, hem executat el programa *IndexFilesPreprocess.py*, donat en el .zip de la pràctica, sobre els documents de novel·les utilitzant els diferents flags: *--token* i *--filter*. Els primers índexs que hem creat han estat amb els diferents tokenitzadors, i aquest ha estat el resultat:



Com podem veure, el tokenitzador amb el qual obtenir més paraules és el *whitespace*, el que té sentit ja que aquest només divideix el text en tokens en funció dels espais en blanc. No fa cap altre processament del text, per tant, és un anàlisis molt bàsic. A continuació, al voltant de les 60.000 paraules ens trobem el *classic* i l'*standard*, que són més restrictius que l'anterior perquè, a part d'utilitzar els espais en blanc per dividir el text, realitzen una sèrie de transformacions addicionals per netejar-lo. El tokenitzador que produeix menys tokens és el *letter* i la raó és que aquest divideix el text

utilitzant qualsevol caràcter que no sigui una lletra com a delimitador, excloent així números i caràcters especials.

Seguidament, ens quedem amb el tokenitzador més agressiu, el *letter*, i ens centrem en els filtres seguint el mateix procediment.



Aquest gràfic ens mostra el nombre de paraules resultant d'aplicar cadascun dels filtres i veiem com aquest és més elevat utilitzant els filtres asciifolding i stop. Les raons poden ser que, en el cas de l'asciifolding només es reemplacen caràcters accentuats pels seu equivalent ASCII i en l'stop s'eliminen stopwords, però aquestes són paraules molt utilitzades en els textos que sovint es repeteixen moltes vegades i, per tant, el nombre de paraules diferents no canviarà excessivament. Després es troba el lowercase que converteix tots els caràcters del text en minúscula i el qual és més agressiu que els dos anteriors. No obstant, en ocasions l'aplicació d'aquest filtre pot suposar un problema en el cas d'abreviatures i pronoms, canviant per exemple "US", abreviatura de United States, per "us", el pronom anglès. En últim lloc, trobem una darrera secció amb els filtres més restrictius: l'snowball, que és el que més, seguit del porter_stem i el kstem. Amb aquests obtenim un menor nombre de paraules, ja que els tres són filtres de derivació lèxica (stemming) que, amb diferents algoritmes, redueixen les paraules a la seva arrel.

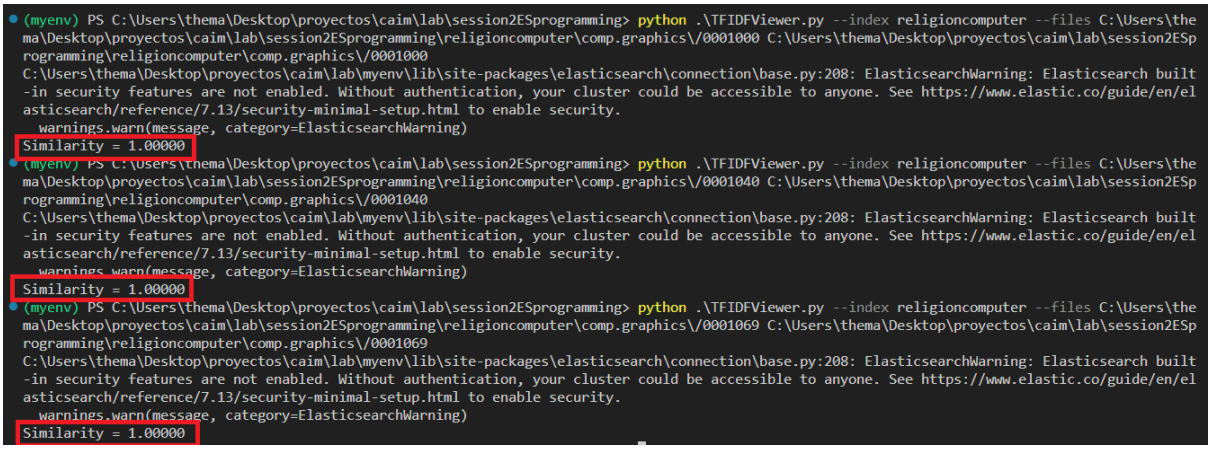
Un cop realitzades aquestes proves, hem experimentat per tal d'extreure altres resultats que no s'aprecien tan clarament a simple vista però que hem considerat importants; si més no curiosos. El primer d'ells és que en tots els casos la paraula amb més freqüència és "the", menys per quan apliquem el filtre stop, ja que eliminant les stopwords, la paraula que més apareix és "I", seguit

d'altres pronoms personals com "he" i "you" i les formes del verb "to have". Una altra conclusió a la que hem arribat és que, degut a que els filtres es poden combinar de diverses maneres, l'ordre en el qual es realitza aquesta combinació importa. Això es pot veure en el cas que hem mencionat anteriorment, on utilitzant el filtre lowercase es transformava "US" en "us". De forma que, si apliquem el lowercase i després l'stop, ens eliminaria tots els "us" incloent els "US" que abans ha transformat. D'altra banda, si executem primer l'stop seguit del lowercase, ens eliminaria només la paraula "us" i no la "US" també.

2. Similaritat del cosinus tf-idf

El primer que hem hagut de fer, ha sigut completar el codi necessari dintre de "TFIDFViewer.py" per poder disposar del programa que calcula la similaritat de dos documents dintre d'un índex. La major dificultat en aquesta tasca ha estat trobar una funció eficient en el cáclcul de la similatrit, on finalment, aprofitant que els termes als vectors estan ordenats i només son rellevants quan el terme es troba als dos, hem aconseguir una solución en temps $O(min(v1, v2))$.

Així, per tal de validar la correctesa del nostre programa primerament hem comprovat la similaritat entre diversos fitxers amb ells mateixos, donat que si el programa funciona correctament, hauria de donar 1 en tots els casos al estar comprovant vectors iguals. I efectivament:



Després de comprovar la correctessa del programa, hem volgut evaluar la sortida de diferents inputs segons la categoria dels textos comparats. Amb aquest objectiu en ment, hem creat un nou índex amb tots els fitxers d'un nou subgrup de novel·les composat per *comp.graphics*, *sci.electronics* i *tal.religion.misc*. La nostra idea és seleccionar dues novel·les aleatòries de cada un dels subgrups i comparar les sortides de totes entre totes, esperant poder distinguir patrons entre els textos i les seves categories. Per exemple, esperem que tots els textos d'una mateixa categoria siguin molt similars i també que hi hagi més semblança entre els textos de *comp.graphics*, *sci.electronics* que no pas entre qualsevol d'aquests dos i *tal.religion.misc.* donada la seva temática religiosa envers la tecnología.

Podem veure els resultats a la taula, on efectivamente podem discernir les nostres sospites. Els documents de *comp.graphics* son molt similars entre ells , després, es pot considerar que guarden certa relació amb els dos de *sci.electronics*, i finalment, que casi no estan relacionats amb els de *talk.religion.misc*.

		comp.graphics		sci.electronics		talk.religion.misc	
		00010017	0001598	0012454	0012714	0019786	0019942
comp.graphics	00010017	1	0.05473	0.02753	0.00111	0.00408	0.00562
	0001598	-	1	0.01251	0.01248	0.00244	0.03718
sci.electronics	0012454	-	-	1	0.00095	0.00373	0.01351
	0012714	-	-	-	1	0.00127	0.01883
talk.religion.misc	0019454	-	-	-	-	1	0.01439
	0018999	-	-	-	-	-	1

Tot i així, també podem veure algunes inconsistències com la poca relació entre els dos documents de *sci.electronics* o la poca similitud entre els documents 0.012714 dels graphics i el 0.0010017 de la electrónica, que fins i tot és menor que la similitud d'aquests amb els documents religiosos.

Pensem que aquestes desviacions son simplement outliers estadisics resultants del petit tamany de la nostra mostra, ja que si bé els textos entre electrònica i gràfics d'ordinador poden tenir similituds, també poden ser tan diferents com ho serien un text d'electrònica pura de microxips i un altre de qualitat gràfica en sentit artístic. Podent comprendre així les desviacions observades.

Tokenització dels paths

Responent a la pregunta final, sí, hem notat que s'han de filtrar els fitxers de manera individual pel path EXACTE especificat al crear l'índex, ja que vam tenir un problema afegint un "/" innecessari a la carpeta original dels índexs fent que no es poguessin trobar després, ja que es necessitava doble "//".

És veritat que si els paths haguessin estat tokenitzats no hauríem pogut especificar les direccions dels fitxers amb el mateix nom que a les nostres carpetes, ja que per exemple, amb l'algorisme *letter* s'haurien separat en dos paraules diferents les carpetes amb un punt al nom. Tot i així, si coneixem l'algorisme tokennitzador podriem adaptar la nostra cerca segons aquest i tenir èxit.

```
def generate_files_list(path):
    """
    Generates a list of all the files inside a path
    :param path:
    :return:
    """
    if path[-1] == '/':
        path = path[:-1]

    lfiles = []

    for lf in os.walk(path):
        if lf[2]:
            for f in lf[2]:
                lfiles.append(lf[0] + '/' + f)
    return lfiles
```

En aquest cas, no es tokentizen els paths donat que a IndexFilesPreprocess.py es pot veure com l'indexador primerament busca tots els paths absoluts cap a fitxers i després tokenitza e indexa els resultats a elasticsearch per aquests valors, permeten així la cerca per path exacte.