CAIM - Lab2
Programming with ElasticSearch

Pau Núñez Amorós
Víctor Vallejo Rives

14/10/2019

1 The index reloaded

En aquest primer apartat hem de comparar els efectes d'aplicar els diferents `tokens` i `filters` dels que disposa `ElasticSearch`. A continuació mostrem els resultats obtinguts amb cada un dels `tokens` amb els que hem executat `IndexFilesPreprocess.py` per indexar i tot seguit `CountWords.py` per obtenir-los.

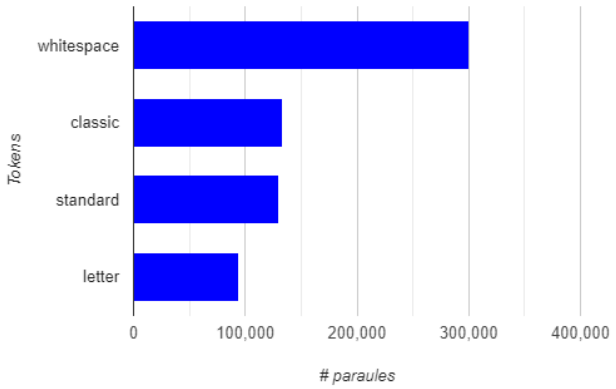

Figure 1: Nombre de paraules diferents segons el token

Com es pot observar, el `token` amb el que obtenim un nombre de paraules més reduït, i per tant el més agressiu, és `letter`. Això és comprensible perquè `letter` usa com a separador $c \mid c \notin [a..z] \cup [A..Z]$. Separant paraules més sovint que altres `tokens`, per tant creant paraules més curtes i conseqüentment creant un nombre menor de paraules **úniques** (hi ha més paraules iguals repetides).

Seguidament se'ns demana que fixem el `token` a l'opció més agressiva: `token=letter` i realitzem el mateix experiment pels `filters`. Hem tornat a executar l'script amb cada un dels filtres i hem obtingut els següents resultats:

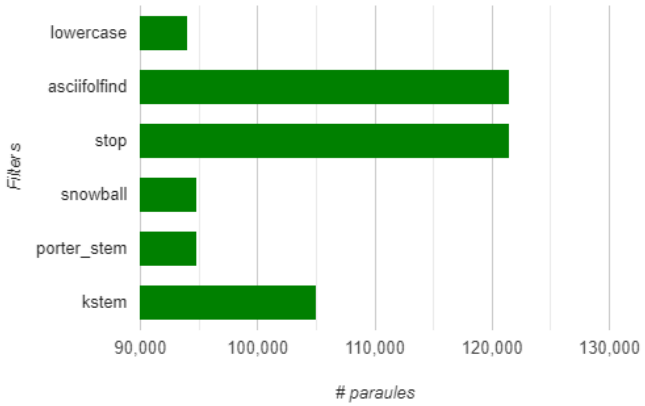

Figure 2: Nombre de paraules diferents segons el filtre (`token=letter`)

Com es pot apreciar a simple vista, el nombre de paraules resultants varia molt notablement depenent del filtre seleccionat. Per curiositat hem comparat quina és la paraula resultant més freqüent al aplicar cada un dels filtres. Com era d'esperar en la majoria dels casos ens trobem amb la paraula `the`, però al ser una *english stop word*, el filtre `stop` no la contempla. En aquest cas la paraula amb major freqüència és `I`, resultat que no se'ns havia acudit però que té molt sentit.

Amb el nostre `DataSet`, quan obtenim un nombre més reduït de paraules com a resultat, és en aplicar el filtre `lowercase`. Pero a diferència del cas dels tokens, aquí podem aplicar més d'un filtre a la vegada.

Hem provat algunes de les possibles combinacions i ens hem adonat que una de les característiques més importants a tenir en compte a l'hora d'aplicar els filtres és **l'ordre**. I si és important és degut a que els resultats poden variar significativament depenent d'aquest. Un clar exemple de cas en el que l'ordre agafa importància és quan apliquem els filtres `lowercase` i `stop`. Si ens trobéssim amb la paraula *AM*, al aplicar primer el filtre `lowercase`, es transforma en *am* i seguidament el filtre `stop` l'eliminarà al considerar-la una *english stop word*. En canvi, si primer s'aplica `stop`, *AM* no forma part de la llista de *stop words* i per tant seguirà existint quan s'apliqui el segon filtre, deixant com a resultat final la paraula *am*.

2 Computing tf-idf's and cosine similarity

En aquesta secció se'ns demana completar el codi de `TFIDFViewer.py` per tal de poder comparar la similaritat (sim) de dos documents donats. Les funcions a implementar són força senzilles amb l'explicació de teoria. Nosaltres vam dubtar però a `cosine_similarity`, ja que al principi vam implementar-la amb una funció d'ordre superior però com les longituds dels vectors que representen cada document pot diferir molt aquesta opció no funcionava correctament. Sabent doncs, que els dos vectors $(L1, L2)$ vénen ordenats alfabèticament per `term` podem fer un recorregut per ambdós alhora en $t = O(min(|L1|, |L2|))$. El següent pseudocodi explica la idea:

```
while (i1 < |L1| and i2 < |L2|):
    if L1[i1].term > L2[i2].term:
        ++i2
    if L1[i1].term < L2[i2].term:
        ++i1
    if L1[i1].term == L2[i2].term:
        sim += L1[i1].weight * L2[i2].weight
```

2.1 Experiments

El primer que hem volgut comparar ha sigut la similitud resultant de comparar un fitxer amb ell mateix. Si tot va bé hauria de ser equivalent a 1, ja que en aplicar els tokens o filtres hauria de generar les mateixes paraules. No obstant, al executar-ho per primer cop vam veure que teniem algun tipus d'error al codi, ja que vam obtenir una similitud de 0.8 amb el mateix fitxer. Vam revisar el `TFIDFViewer.py` i vam trobar un petit error a l'hora d'acumular el valor de la variable representant de la similiaritat dins la funció `cosine_similarity(tw1, tw2)`. Un cop aplicat el canvi ja hem obtingut le resultat desitjat.

Com a curiositat hem intentat comparar un fitxer qualsevol amb un de buit, esperant que ens donés una similitud de 0. Però en comptes d'això hem vist que el programa ens retornava error. Buscant el causant d'aquest hem trobat que a l'hora de calcular la paraula de màxima freqüència, la funció `max` es crida recorrent un `file_tv` buit, i això fa que el mètode `max()` es cridi sense arguments, cosa que `python` no accepta.

Un cop fetes aquestes comprovacions introductòries hem passat a comparar autors mitjançant la comparació de les seves obres. En el nostre cas hem volgut comparar la forma d'escriure que tenen *Edgar Allan Poe* i *Charles Dickens*. Són dos autors que comparteixen època històrica, i per tant es podria assumir que de base comparteixen bastantes coses. Abans de fer la comparació entre ells, i per tenir algun valor de referència, hem calculat la similitud entre dues noveles de Poe. Hem obtingut com a resultat una semblança de 0.24, que considerem que és prou elevada ja que representa que comparteixen gairebé 1 de cada 4 paraules.

Ara sí, comparem les noveles dels dos autors com a tal:

$$sim(Poe, Dickens) = average(sims) = (0.009 + 0.039 + 0.029 + 0.016 + 0.052 + 0.030)/6 = 0.029 \sim 0.03$$

Com podem veure, una novel · la de *Poe* és quasi 10 vegades més diferent d'una de *Dickens* que d'una altra de seva. Al ser coetanis esperàvem obtenir una similaritat major, però sembla que els autors tenien estils prou diferenciats.

Seguidament hem volgut comprovar si el tema del text és important a l'hora d'escollir les paraules que s'utilitzen. Hem comparat dos texts del tema *space* i hem obtingut una similitud de 0.04, que sembla bastant limitada. Però sí que s'obté un valor superior que al comparar un text d'aquest tema amb un del tema *motorcycles*, ja que ha resultat de 0.026.

Per acabar, i seguint amb la mateixa idea de temes, hem comparat un text de *baseball* amb un de *hockey*, per comprovar si el fet de ser esports era suficient per a crear una alta similaritat. El resultat ha sigut de 0.015, i per tant concloem que no, el simple fet de ser dos esports no provoca que els texts siguin molt més similars, ja que al cap i a la fi cada esport té el seu vocabulari propi.