Lab5: MapReduce and Document clustering

Q1 2022-23

Integrants:
Pablo Montón Gimeno
Cristian Sánchez Estapé

Introducció

En aquesta pràctica, utilitzant el framework de python MRJob per implementar amb *map-reduce* l'algorisme de *clustering k-means*. Donat un conjunt de dades de paraules i les seves freqüències extretes amb ElasticSearch; llavors aconseguirem un conjunt de clústers que tenen similitud entre ells.

Característiques i dificultats de la implementació

A continuació, es detallen part de les característiques de la nostra implementació:

- Pel que fa a la funció *jaccard*, i seguint les instruccions de l'enunciat, s'ha realitzat una implementació basada en l'ús de llistes i *multisets*, uns objectes que corresponen a conjunts o *sets* que permeten repeticions. Aquest darrer tipus d'objecte ha estat utilitzat per implementar, de manera eficient, l'operació d'intersecció entre conjunts de paraules. A més a més, degut a aquesta implementació prescindim del fet que el paràmetre *doc* (la llista de paraules) estigui ordenada.
- Pel que fa a la funció *assign_prototype* (*map*), s'ha optat per fer una implementació molt simple: mirem, per tots els clústers, quin és el més proper, segons *jaccard*, a la llista de paraules indicada; i retornem una tupla (clúster, (doc, llista)). El retorn del valor (doc, llista) ens permetrà construïr posteriorment tant la llista de documents assignats a un clúster com les paraules assignades al mateix.
- Pel que fa a la funció *aggregate_prototype* (*reduce*), fem quelcom semblant a la funció prèvia: construïm, de manera iterativa, el conjunt total de paraules i documents assignat al clúster, dades que extraiem de la llista de tuples (doc, llista) generades pel *map*, i finalment retornem una tupla (prototip, (documents, frequencies)), on *frequencies* correspon a un diccionari, sent les claus les paraules i els valors les seves corresponents freqüències.
- Finalment, el cos del bucle del script *MRKmeans.py* s'ha implementat de la següent manera: per cada iteració recorrem el resultat de l'execució dels map-reduce, tot construïnt els diccionaris de la nova assignació de documents i paraules als clústers; seguidament, generem els seus fitxers associats (tant pels documents com per les llistes de paraules amb les seves freqüències), i finalment comprovem si la nova assignació és igual a l'anterior. Dues coses han de ser destacades: per una banda, és en escriure el fitxer *prototypes* que ordenem alfabèticament les paraules; d'altra banda, fem ús de l'operador "==" per diccionaris (que ja fa la comparació profunda, clau per clau, valor per valor) per definir la condició d'aturada del programa.

Experimentació

A part de la mateixa experimentació feta per comprovar el correcte funcionament de l'algorisme, hem documentat uns experiments sobre un cert *dataset* per tal de comprovar quin efecte tenen certs canvis en la configuració. Els resultats d'aquesta experimentació estan a l'arxiu adjunt *Experimentation.pdf.*

Efecte de canviar el nombre de clusters

Donat un índex, arxiu, nombre de paraules i iteracions fixe,
Com podem observar a les dades i als comentaris que hem indicat a l'experimentació, veiem que, a l'incrementar el nombre de clústers, augmenta la complexitat de l'algorisme, i observem que això implica que aquest trigui més iteracions (i, per tant, més temps d'execució) en convergir.

Efecte del tamany del vocabulari

Per realitzar aquest experiment, hem agafat un dataset al qual canviarem el nombre de paraules. Com podem veure, aquest fenomen implica un increment en la complexitat i, per tant, en el temps d'execució. Però, a més a més, podem veure que triga més iteracions en convergir, per tant és una altra constatació que utilitzar clústers més grans fa que trigui més iteracions en convergir.

Efecte del nombre de *cores*

Per comprovar l'efecte de canviar el nombre de cores amb els que s'executa el programa (i, per tant, el nombre d'instàncies de *maps* i *reduce*), fixem la resta de valors com hem fet als anteriors experiments.
En aquest cas, podem constatar com realment veiem que, en aquest cas, l'únic que canvia és el temps que triga a computar. Com veiem, cada iteració triga un temps molt diferent, ja que la primera iteració s'utilitza principalment per inicialitzar els *clusters* d'acord amb el *prototype* inicial creat amb l'altre programa (i que, donada la reduïda mida que inicialment té, implica un temps menor d'execució que la resta d'iteracions, a excepció de la darrera). En aquest cas, la iteració 2 és en la qual es fa la gran part de la computació de les instàncies de map-reduce, ja que a l'última principalment detecta que aquesta ha convergit i s'atura. Podem observar com, quan incrementen els *cores*, a l'inici tenim un increment del rendiment molt més gran, més que quan ja tenim un nombre de cores considerat, i la diferència és insignificant encara que els doblem. Això es pot donar a causa que hi ha un moment en el qual, encara que s'incrementi el nombre d'instàncies de *map* i *reduce*, si no s'utilitzen no té cap efecte positiu i, fins i tot, pot alentir el temps de computació.

Iter\nCore	**1**	**2**	**4**	**8**	**16**
1	97.44 s	53.16 s	31.49 s	25.24 s	27.54 s
2	320.07 s	177.90 s	105.93 s	84.59 s	86.93 s
3	4.63 s	2.96 s	2.37 s	2.46 s	2.95 s

Taula extreta del 3r apartat de Experimentation.pdf

Prova de correctesa

Per comprovar que l'algoritme s'executa correctament, el que vam fer és crear un dataset amb un nombre gran de paraules sense modificar la freqüència d'aparició, per tal d'observar els *assignments,* i així veure que els clústers es van organitzant segons el tipus de paraules i el document on apareixen. En arribar a més iteracions que l'exemple anterior, podem veure millor com van convergint els clústers.

El resultat dels *assignments* i dels *prototypes* està a la carpeta del zip *proof_of_correctness*.