Session 5: MapReduce and Document clustering

Q1 2022-23

Integrantes:
Pol Pérez Castillo
Daniel Ruiz Jiménez

Fecha de entrega: 02-12-2022

Índice

Introducción

En esta práctica vamos a implementar el algoritmo de KMeans utilizando el método map-reduce usando la librería MRJob de Python. Nuestro objetivo con este algoritmo es agrupar los documentos de un índice en k conjuntos (clusters) según su similaridad. Para calcular la similaridad de un documento al centroide de un cluster hemos utilizado el índice de Jaccard.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Imagen 1: Fórmula del índice de Jaccard

Esta fórmula calcula la similitud entre 2 documentos, con un valor entre 0 y 1, pero para nuestra práctica usaremos la distancia al centroide, es decir, el complementario de la similitud (d = 1-s).

Implementación

Hemos implementado 2 programas: el primero, *MRKmeansStep.py*, estaba incompleto. Hemos terminado de implementar las funciones vacías, más en concreto la que calcula la similitud con la fórmula del índice de Jaccard vista más arriba. La segunda, el mapper, calcula el cluster (prototype) más cercano a un documento, y finalmente devuelve el cluster al que se asigna junto con un par de valores, que consiste en el ID del documento y una lista de todos sus tokens. La última, el reducer, calcula para todos los tokens del documento nuevo añadido al cluster, su frecuencia con la fórmula del TF (nº ocurrencias en el documento/tamaño total del documento), y devuelve el cluster con una lista de sus documentos ordenada, y la lista de frecuencias ordenada también.

El segundo programa, *MRKmeans.py*, ejecuta k veces el programa anterior, y finalmente, genera unos outputs para visualizar el correcto funcionamiento del programa. Los ficheros assignmentsi.txt, siendo i el número de cada iteración, contiene todos los IDs de los documentos, ordenados alfabéticamente. Los ficheros prototypesi.txt, por otro lado, contienen una lista de todos los tokens de cada cluster junto con su frecuencia.

Experimentación

Para hacer la parte de experimentación de esta práctica, vamos a utilizar el conjunto de textos científicos (ArXiv). Ejecutaremos varias veces el programa de KMeans con diferentes frecuencias máximas y mínimas, para observar la composición de cada cluster. Usamos siempre 10 clusters y ejecutamos 20 iteraciones como máximo. Además, ejecutaremos primero con las 100 palabras más frecuentes del conjunto, y a continuación con las 250 palabras más frecuentes.

100 Palabras

1-10%

Debido a que son las palabras menos frecuentes de cada documento, el algoritmo no ha convergido. La media de duración de cada iteración ha sido de 8,3 segundos.
Además, como son palabras menos compartidas entre documentos, podemos observar una ligera categorización en los clusters.
Por ejemplo:
CLASS2
[(0.39577296744503154, 'magnet'), (0.36952445883756607, 'electron'), (0.34958241009033575, 'particl'), (0.18442133969660815, 'flow'), (0.15663882733935572, 'free')]
Este cluster contiene términos presentes en el campo de la física, más concretamente de las partículas y ondas.
CLASS3
[(0.39414654988250375, 'galaxi'), (0.3326212347788934, 'stellar'), (0.27536851100192267, 'emiss'), (0.24193548387096775, 'sim'), (0.21138645588549454, 'veloc')]
Y este contiene términos del espacio

10-30%

Subiendo la frecuencia, vemos que hay cada vez más palabras que se repiten en cada cluster. Por lo que a partir de esta frecuencia se nos ha hecho complicada la tarea de categorizar los clusters. Una vez más, el algoritmo no ha convergido, pero ha tardado menos en ejecutarse cada iteración, con una media de 7,2 segundos.

30-50%

Subiendo todavía más la frecuencia, nos seguimos encontrando con el problema de que las palabras son muy comunes, véanse conectores, adverbios y demás categorías gramaticales. No converge. Media de 3,1s/iteración.

50-70%

En este caso, las palabras son tan frecuentes que casi cada cluster está conformado por los mismos términos, por lo que el algoritmo ha convergido muy rápido, en la tercera iteración. Media de 1,3s/iteración

Mostraremos un pequeño ejemplo del contenido de los clusters al escoger frecuencias tan altas:

CLASS5

[(1.0, 'model'), (0.5465890183028286, 'our'), (0.20861064891846923, 'show')]

CLASS6

[(1.0, 'our'), (1.0, 'model'), (1.0, 'can'), (0.5342152124141439, 'result')]

CLASS7

[(1.0, 'result'), (1.0, 'can'), (0.5008917954815696, 'our')]

CLASS2

[(1.0, 'result'), (1.0, 'model'), (0.5523765574526995, 'can'), (0.44639286263651745, 'show')]

El programa detecta que los términos son tan parecidos que ni siquiera genera el número de clusters introducido por parámetro, en lugar de 10 clusters genera 8.

250 palabras

Ahora ejecutaremos otra vez el programa pero aumentando el número de palabras recogidas de cada documento, manteniendo los demás aspectos (número de clusters e iteraciones máximas). Esperamos que el programa se comporte de la siguiente manera:

1. Que categorice mejor los clusters, ya que habrá más variedad léxica.
2. Que tarde más en ejecutarse el algoritmo, por lo que el tiempo de ejecución será mayor en cada iteración.
3. Que el algoritmo necesite más iteraciones para converger.

Veamos:

1-10%

Para este par de frecuencias hemos notado un aumento considerable en el tiempo de ejecución del programa. Ahora tarda 19 segundos en ejecutarse cada iteración. No obstante, la categorización de cada cluster es sorprendentemente precisa. No ha convergido.

Por ejemplo:

CLASS7

[(0.996629213483146, 'constant'), (0.1556179775280899, 'univers'), (0.14157303370786517, 'matter'), (0.14101123595505619, 'assum'), (0.13230337078651686, 'distanc')]

En este cluster es fácil deducir que habla del espacio.

CLASS5
[(0.48140669695440014, 'electron'), (0.38515901060070673, 'magnet'), (0.3255931347804139, 'materi'), (0.25643614336193843, 'spin'), (0.19266363789331986, 'induc')]
Y este habla de electromagnetismo.

10-30%

Aquí, al igual que en el caso anterior, notamos un aumento del tiempo de ejecución con su contraparte de 100 palabras, 20 segundos por iteración. Esperábamos un descenso respecto a las frecuencias anteriores, pero deducimos que no hay 250 palabras en el caso anterior, por lo que coge menos. A partir de este punto ya coge 250 palabras. En cuanto al contenido de los clusters, ya no es tan instantáneo deducir una categoría nada más ver el contenido, pero sigue siendo fácil. No ha convergido.

30-50%

Con estas frecuencias, el tiempo de ejecución ya se ha normalizado. Ahora tarda una media de 4 segundos por iteración. Las palabras de cada cluster ya se empiezan a repetir entre clusters y pasan a ser palabras más comunes. No converge.

50-70%

Finalmente, con estas frecuencias ha convergido nada más en 2 iteraciones. Ha tardado 1,7 segundos en cada iteración. Los resultados son muy similares a los de su contraparte de 100 palabras.