CAIM - Sesión 6

Map-Reduce and Document clustering

Santiago Arxé i Carbona

Bryan Leonardo Salto Salao

15/11/2021

Introducción

Antes de comenzar a plantear la implementación, es importante conocer con profundidad los algoritmos de ***Map-Reduce*** y ***K-Means***.

La principal característica del algoritmo de ***Map-Reduce*** es la simplificación de documentos en sus características más básicas, como son sus palabras y la cantidad de veces que aparece cada una de ellas.

Esta manera de representar documentos es más que ideal para la implementación y utilización de ***K-Means***. Este algoritmo de *clusterización*, permite crear los *clusters* en función de la cercanía de los puntos (en este caso, los documentos).

Explicándolo de manera rápida y simple, este algoritmo identifica nubes de puntos (los *clusters*) que tendrán un centroide, normalmente identificado como el centro de todos los puntos de la nube. Los puntos pertenecerán a la nube cuyo *cluster* sea el más cercano a ellos. En el caso de este estudio, los documentos serán los que representarán los puntos en un espacio de tantas dimensiones como palabras se obtengan.

Esta abstracción fue posiblemente lo más complicado de la implementación, ya que entender qué eran las coordenadas de los puntos y cómo podía calcularse su “distancia” no era demasiado intuitivo. Una vez se entendió eso, la implementación no fue demasiado complicada.

Experimentación

Análisis del efecto de los intervalos seleccionados

Primero de todo, se analizará cómo afecta el intervalo de palabras que se seleccionan (min. freq y max. freq) al momento de ejecutar el ***Map-Reduce***. Para ello se han probado diferentes frecuencias con ciertos parámetros ya fijados. El número de *clusters* inicial será de 10, el número de iteraciones de 5 y el número de palabras de 250. Se analizará si se pueden obtener el número de palabras solicitado, si se reduce el número de *clusters* inicial, cuál es el tiempo medio de cada iteración (sin tener en cuenta la primera iteración), y como són los *clusters* resultantes. Al final se intentará conseguir un intervalo “ideal” que pueda dar buenos resultados.

	Caso 1	**Caso 2**	**Caso 3**	**Caso 4**	**Caso 5**	**Caso 6**
min - max	0.01 - 0.05	0.7 - 0.9	0.1 - 0-3	0.2 - 0.3	0.1 - 0.15	0.05 - 0.15
***Clusters* finales**	10	4	10	10	10	10
tiempo medio	28.25	1.6	27.24	7.5	13.63	27.1

Núm. Palabras	250	3	250	58	117	250
Obs.	Cada *cluster* tiene poca similitud, cuesta sacar una temática para cada *cluster*.	Palabras muy comunes (Ej. "*which*")	Palabras con cierta relación en cada *cluster*. Hay alguna bastante común (Ej. "*from*").	Hay alguna palabra bastante común (Ej. "*from*")	Se distingue un tematica en cada *cluster*.	Se distingue un temática en cada *cluster*.

Tabla 1

En el caso 2, se puede ver cómo afecta tener intervalos con frecuencias altas a la hora de implementar ***K-Means***. Se observa como hay muy pocas palabras con una frecuencia tan elevada; en realidad solo hay tres. Esto hace que el tiempo de ejecución sea el más pequeño y que al final solo se necesiten 4 *clusters* para dividir los documentos. Se puede ver que es una mala opción ya que los *clusters* no tendrían sentido alguno.

Por otra parte, el caso 1 es todo lo contrario, se consiguen 250 palabras y el tiempo de ejecución aumenta gravemente. No se tiene que olvidar que es un problema **NP-Completo**. El problema es que las palabras que forman parte de cada *cluster* no se asemejan tanto entre sí. Cuesta encontrar una temática que encaje en cada *cluster*, es más parece que alguna palabra estaría mejor en el otro *cluster*.

En los otros casos, se puede observar lo mismo pero no llevado tan al extremo. El caso 3 parece que tiene buenos resultados, aunque se ve que el rango es bastante grande y algún *cluster* tiene palabras que no se podrían clasificar en ninguna temática en concreto. En el caso 4, pueden observarse cosas parecidas al caso 2, con pocas palabras que son bastante frecuentes en cualquier texto. El caso cinco parece tener mejor pinta, el número de palabras se ve incrementado, aunque no llega a 250, y se ve que cada *cluster* podría relacionarse con alguna temática.

Para acabar se encuentra el caso 6, el que da mejores resultados. El número de palabras es el solicitado y las palabras de cada *cluster* tienen mucha relación y se podría deducir una temática para cada grupo a simple vista.

Echando la vista al tiempo de ejecución, se puede ver como cuando el número de palabras es menor el tiempo de ejecución también lo es, y a la inversa también sucede lo mismo. Además, durante la ejecución de cada caso se ha visto que la primera iteración siempre es la que tarda menos. Esto es debido a cómo funciona ***K-Means*** y cómo se calculan las distancias/similitudes (se hablará más de ello en el siguiente apartado). Las demás iteraciones tienen un tiempo bastante parecido entre ellas, como el número de parámetros y los cálculos entre cada iteración siempre són los mismos, tiene sentido que sea así, excluyendo la primera iteración.

Para acabar, analizando los *clusters* / grupos resultantes, se ve que, excepto en el caso 2, donde solo había 3 palabras, siempre se podía obtener 10 grupos. Tiene sentido, ya que arxiv está dividido en 8 apartados y que seguramente cada uno de ellos se pueda dividir en más.

Para acabar de ver la importancia de escoger un buen valor de ***K*** y un buen número de palabras, se han analizado dos casos más con 100 y 250 palabras respectivamente.

	Caso 7	Caso 8
K inicial	20	15
freq. min - freq. máx	0.05 - 0.15	0.05 - 0.15
Núm. Iteraciones	10	10
Núm. cores	4	4
Núm. Palabras	100	250
K. final	20	15
Tiempo medio (sin iter 1)	22.12	48,26
Observaciones	Es difícil encontrar una temática para cada grupo.	Grupos con cierto sentido. Se pueden distinguir temas como probabilidad astrología o matemáticas

Tabla 2

En el caso 7 puede verse como el tiempo de cada iteración no ha aumentado en comparación con el caso 5, analizado anteriormente, aunque el número de *clusters* ha aumentado a 20 el número de palabras se ha reducido a más de la mitad. Por lo que parece, gran parte del coste se ha visto equilibrado con un menor número de palabras.

Como se ha aumentado el número de iteraciones también se ha aumentado el número de núcleos pasando de 2 a 4 con tal de mejorar el rendimiento (comentado con más profundidad en el siguiente apartado). En cuanto al resultado, se puede ver cómo se ha reducido la calidad. El programa es capaz de crear 20 grupos pero muchos de ellos no tienen mucho sentido.

En el caso 8 se ve como el tiempo de cada iteración es mayor en comparación al caso 7. El principal motivo es el número de palabras, más del doble. Otra vez pasa lo mismo, el programa es capaz de crear 15 grupos. No obstante, en esta ocasión los grupos tienen mucho más sentido. Es posible distinguir ciertos temas con cierta facilidad y no son temas demasiado generales como “ciencia” sino más específicos como por ejemplo “astronomía”.

En conclusión, si se tiene en cuenta el resultado, el caso 8 es el mejor con un valor más pequeño para ***K****,* mientras que el caso 7 obtiene mejores tiempos. La mejor opción sería una combinación de ambas, 100 palabras o alguna más y 15 *clusters* o alguno menos.

Análisis del efecto del número de *threads*

Con algoritmos tan costosos como los de esta práctica, es importante buscar siempre la manera más óptima de ejecutarlos. Para ello, una de las opciones más habituales es la paralelización. Por suerte, los cálculos a realizar pueden paralelizarse y distribuirse en *threads* que podrán trabajar de manera independiente.

Para analizar el efecto del número de *threads* en la ejecución, se han realizado diversas ejecuciones con una cantidad diferente de hilos de procesamiento. Como se ha comentado anteriormente, el tiempo que tardan las iteraciones (sin contar la primera) dentro de un mismo caso es muy similar, con lo que solo se han ejecutado 5 iteraciones en cada caso para reducir

un tiempo de espera totalmente innecesario. Cabe destacar que este estudio se ha realizado con la configuración del **Caso 6** del apartado anterior (min. freq: 0.05, max. freq 0.15, número de palabras: 250).

	Caso 1	**Caso 2**	**Caso 3**	**Caso 4**	**Caso 5**	**Caso 6**
# threads	1	2	4	8	16	24
tiempo primera iteración (s)	13.47	7.43	4.46	3.64	3.51	4.03
tiempo medio demás iteraciones (s)	72.08	37.69	20.39	14.31	11.95	12.49

Tabla 3

Como puede observarse en la Tabla 3, el tiempo de ejecución va disminuyendo cuando se aumenta el número de *threads*, pero cada vez la mejora es menor. Uno de los puntos clave de la paralelización es que, pese a que es cierto que distribuir la carga de trabajo entre *threads* es siempre algo positivo, eso viene con un precio a pagar. Cada vez que se crean *threads* y se reparte trabajo entre ellos, hay que tener en cuenta un *overhead* que, a partir de un momento dado, hace que seguir dividiendo el problema sea algo negativo.

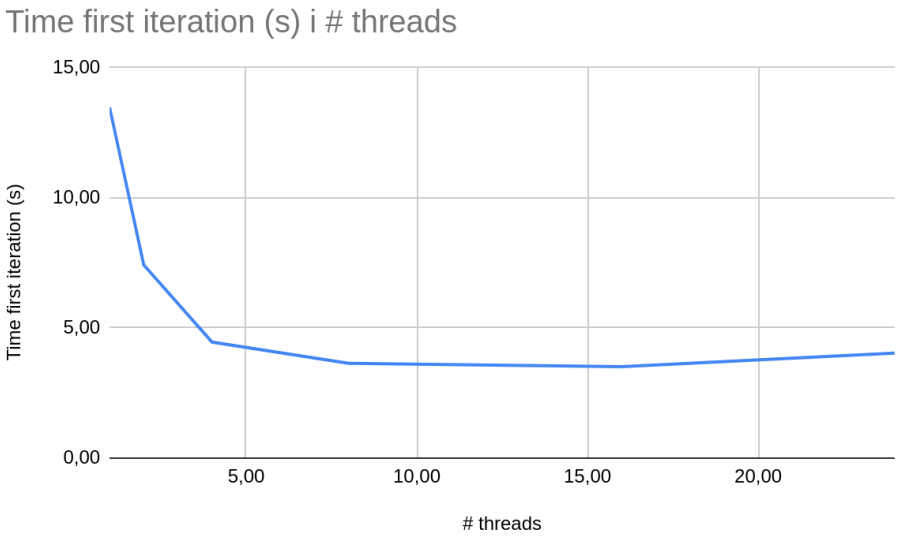

Figura 1: Tiempo de la primera iteración en función del número de *threads*

Como puede observarse en la Figura 1, el tiempo de ejecución va disminuyendo conforme aumenta el número de *threads*… Pero solo hasta un punto. Cuando se pasa de 16 hilos de procesamiento a 24, la mejora que viene de dividir más el problema es superada por el *overhead* de crear dichos hilos, con lo que el tiempo acaba aumentando.

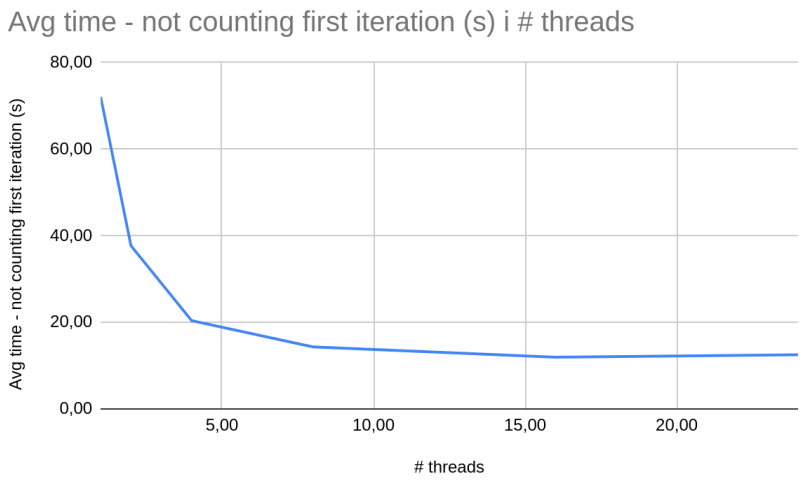

Figura 2: Tiempo medio de ejecución de las iteraciones (sin tener en cuenta la primera) en función del número de *threads*

El aumento del tiempo de ejecución a partir de los 24 *threads* que se veía en la primera iteración, puede observarse también en el resto de iteraciones (Figura 2). Además, la forma de la progresión parece ser la misma. No obstante, es importante destacar que todas las iteraciones tardan mucho más que la primera (en todos y cada uno de los casos).

Esto se debe, principalmente, a que los centroides utilizados en la primera iteración son un documento concreto (no un punto medio entre diversos documentos). Esto tiene dos resultados claros.

- No hay que calcular el los centroides en la primera iteración.
- Los cálculos son más complicados cuando el centroide es un “documento teórico” con valores reales (y no 0s y 1s como los documentos utilizados).

Clusters ideales

Para acabar, con los conocimientos recogidos por todos los experimentos anteriores. Se ha ejecutado ***Map-Reduce*** con los siguientes parámetros: frecuencias mínima y máxima de 0.05 y 0.1, 100 palabras, 12 *clusters* iniciales, 50 iteraciones y 16 cores.
El tiempo medio de iteración (sin contar la primera) ha sido de 4,5 segundos, y la primera ha tardado 2,1s. Como se puede ver, el output de *script* **ProcessResults.py** devuelve grupos con *tokens* muy parecidos entre ellos. Por ejemplo CLASS3 se podría relacionar con la informática, con diferentes palabras como *task*, *artificial* (intelligence), *input*, *machine*, *neural*(IA). CLASS4 se puede relacionar con la física, CLASS5 con la astronomía…

CLASS3	->	neural	task	art	machin	input
CLASS11	->	galaxi	stellar	emiss	sim	spectral
CLASS5	->	distanc	galaxi	veloc	constant	stellar
CLASS7	->	classic	finit	establish	constant	probabl
CLASS9	->	flow	particl	behavior	analyt	critic
CLASS1	->	free	finit	particl	classic	constant
CLASS8	->	complet	finit	open	contain	group
CLASS2	->	doe	probabl	input	assum	constant
CLASS4	->	magnet	electron	wave	frequenc	particl

CLASS10	->	research	tool	futur	open	support
CLASS0	->	constant	probabl	finit	assum	establish
CLASS6	->	scheme	minim	cost	finit	signal