Pontificia Universidad Católica de Chile
Instituto de Ingeniería Matemática
IMT2112

Tarea 1

31 de agosto de 2020

Fernando De Diego Ávila

Resumen

En esta tarea, se pidió implementar de tres maneras distintas el algoritmo *k-means*. La primera, que llamaré "Método 1: Sólo Loops", consiste en utilizar sólo bucles de Python para realizar los cálculos. La segunda manera (Método 2), consiste en utilizar la librería externa NumPy, para optimizar los cálculos que involucran vectores (practicamente todos). Finalmente, la tercera forma (Método 3), consiste en paralelizar los bucles de Python con la librería externa muliprocessing. Es importante notar que en este último método, se prefirió usar un bucle para hacer append, ya que es mucho más eficiente que paralelizarlo (los distintos procesadores tendrían que pedir el mismo objeto todo el tiempo). De todas formas, se dejó comentada la implementación en paralelo.

Aparte de los tres métodos mencionados en enunciado, implementé una variante del Método 1 (Loops y Map), que consiste en una mezcla de bucles y mapeos, y que ocupa las mismas funciones que el Método 3, solo que sin multiprocessing. Esto, para comparar el tiempo de ejecución de este método y el Método 3.

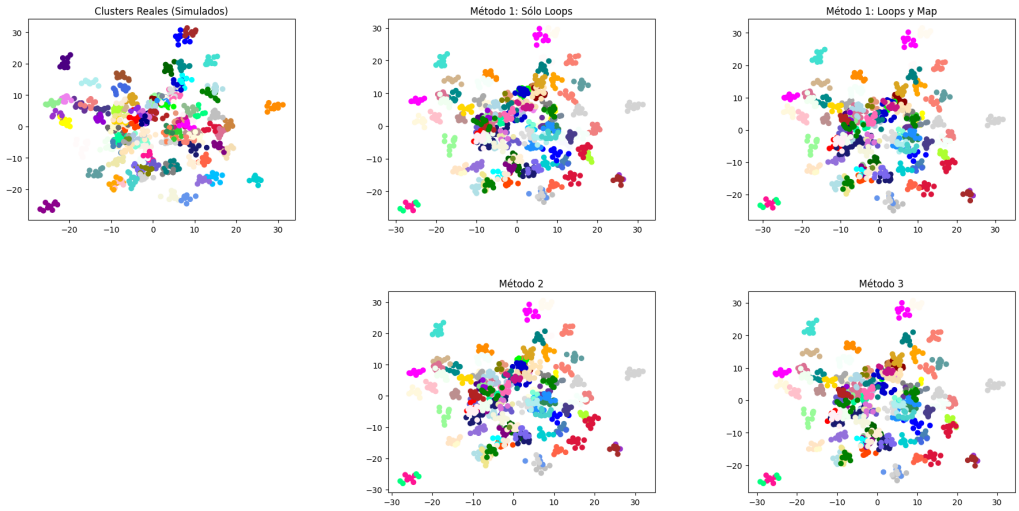
Algunas observaciones con respecto a los resultados son las siguientes:

- NumPy es extremadamente rápida para problemas con dimensiones grandes (> 100). Para problemas con baja dimensión, suele tomar un tiempo mayor que los otros métodos. Esto, debido a que NumPy está realmente optimizada para optimizar problemas de alta dimensión.
- El Método 1: Sólo Loops suele tener peor desempeño que el Método 1: Loops y Map, salvo en los casos en que el número de datos y la dimensión es baja. Esto, debido a que el *overhead* de generar las tuplas para llevar a cabo la suma de vectores y la iteración sobre el conjunto de datos es mayor que la velocidad que aporta la función *map*.
- El Método 3 suele tener mejor desempeño que el Método 1, para casos con un número

de *clusters* grande (> 50) y con un número de datos grande (> 2000). Para los casos que no cumplen esas condiciones, suele tener un peor desempeño (en ocasiones, por mucho), debido a las funciones paralelizadas están en función del número de datos y el número de *clusters*. En estos últimos casos, el *overhead* de separar los datos y comunicar los distintos procesos supera con creces el beneficio que trae paralelizar el código.

Resultados

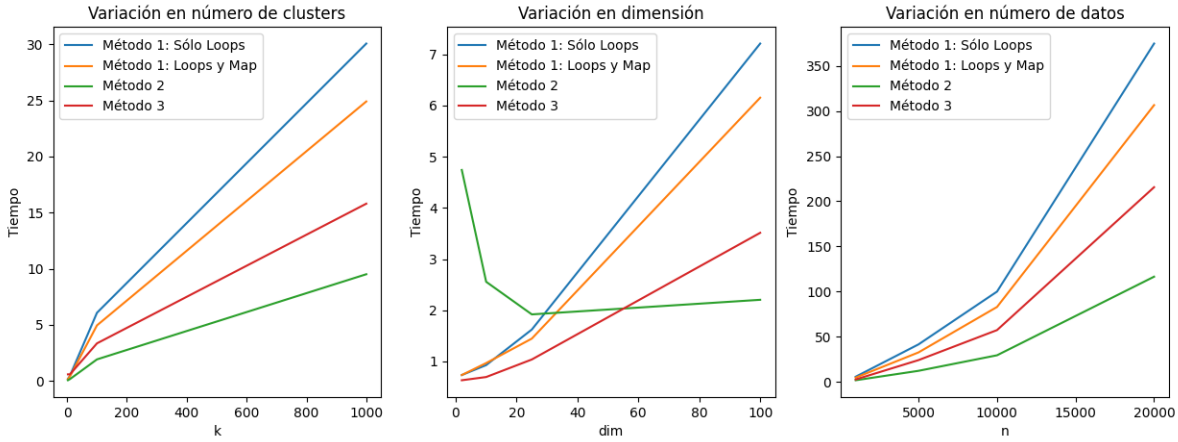
A continuación, se presentan un gráfico de las componentes principales de los puntos simulados, junto a gráficos con los resultados obtenidos al aplicar el algoritmo *k-means* usando los cuatro métodos mencionados anteriormente. Claramente, los resultados obtenidos deben ser iguales (y así resultaron):



Además, se calculó el comportamiento de los distintos métodos a medida que varían los parámetros (dimensión, número de datos, y número de *clusters*). El caso base se definió con los siguientes valores:

- Número de datos (n): 1000.
- Dimensión de los datos (dim): 100.
- Número de clusters (k): 100.

El tiempo tomado por cada método es el siguiente:



Claramente, el Método 2 suele ser más rápido que el resto de los métodos, probablemente debido a la optimización explicada anteriormente. El Método 1: Sólo Bucles, presenta en casi todos los casos, resultados peores que el Método 1: Loops y Map. Finalmente, el Método 3 presenta mejores resultados que el Método 1, sobretodo a medida que los valores de los parámetros crecen.