UNIVERSIDAD CATOLICA SAN PABLO - AREQUIPA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Estructura de Datos Avanzados

Informe

K - Means

Advisor: Erick Mauricio Gomes Nieto
Carlos Eduardo Atencio Torres

Student: Loayza Huarachi Angel Josue

AREQUIPA, JUNE 2023

Contents

1 Introducción

En el presente trabajo, se mostrará una implementación de la estructura de datos **K-Means**, que pertenece a la familia de los **Clusters** en el lenguaje de programación C++. También, se detallará el objetivo y lo que hacen todas las clases y métodos desarrollados para su respectiva implementación; más no se explicara como funciona la estructura, pues es algo que se intuye, que se sabe. Por último, se propondrá una mejora utilizando la librería **OpenMP** para la paralelización Aun así, se compartirá el link del repositorio de GitHub para futuras pruebas y mejoras: https://github.com/angel452/

2 Metodologia y Lógica

A continuación, se describirá como se llevó a cabo esta implementación, mostrando las funciones, clases, apuntes, etc.
En el código presentado, contamos con 4 clases primordiales para tanto recibir el "Data Set" y crear la propia estructura. Entre estas clases tenemos: La clase Point, la clase MBR, la clase ClusterGroup y la clase Kmeans.

2.1 Clase Point

La clase Point nos servirá para guardar las coordenadas extraídas del Data-Set; y en algunos casos también la usaremos para los puntos "BottomLeft" y "UpperRight", los cuales servirán para sacar el MBR (Mínimum Bouding Rectangles).

Variables privadas:

- cords: Array de tipo float que se usara para 2 motivos: Guardar las coordenadas de un punto, y también para guardar el bottomLeft y el UpperRight que generan todos los puntos juntos.

2.2 Clase MBR

La clase MBR servirá para guardar el Mínimum Bouding Rectangles que forman todos los puntos luego de ser insertados en la variable dataSet de la clase Kmeans. Por ello, las únicas 2 variables de esta clase son de tipo/clase Point.

Variables privadas:

- bottomLeft: Conjunto de "n" puntos (de acuerdo a cuantas dimensiones estamos trabajando) que indican el punto más a la izquierda - abajo
- upperRight: Conjunto de "n" puntos (de acuerdo a cuantas dimensiones estamos trabajando) que indican el punto más a la derecha-arriba

2.3 Clase ClusterGroup

Luego está la clase ClusterGroup, esta clase será la encargada de guardar las caracteristicas primordiales que destacan a un cluster; y además, será en donde crearemos por primera vez nuestros primeros centroides aleatorios

Variables privadas:

- centroid: Será una variable de tipo/clase Point, pues recordemos que nuestro centroide también es un punto en nuestro espacio n-dimensional, solo que este se estará actualizando constantemente.
  Al separarlo de esta manera, podemos tener un mejor control y asegurar que cada Cluster tenga un centroide
- dataCluster: Este será un vector que guardará variables del tipo Point, es decir, que en este vector se guardaran todos los puntos que pertenecen a dicho clúster.
- dataClusterAux: Esta variable cumple la misma labor que la anterior. Nos servirá para guardar un "backup" de puntos que estaban guardados en la variable dataCluster, ya que la requerimos para comparar si algún punto se actualizó de Cluster y tengamos que volver a actualizar los centroides (Esto se verá claro en la función "checkChanges" de la clase Kmeans)

Métodos:

- createCentroid: El objetivo de esta función es crear aleatoriamente una cantidad específica de puntos (de acuerdo a cuantas dimensiones trabajemos), para colocarlos como coordenadas de los centroides.
  Ya que la aleatoriedad puede darme números inmensamente alejados, lo restringimos a que solo lo genere entre los parámetros regidos por el bottomLeft y upperRight.

2.4 Clase Kmeans

En esta clase y sus métodos, se llamarán a todas las anteriores clases y sus métodos para formar así nuestra estructura.

Variables privadas:

- kCentroides: Variable de tipo entera donde guardaré cuantos clusters/centroides generaré para mi Data-Set
- dataSet: Este es un vector donde solo guardaremos toda la información recopilada por el Data-Set.
- minimumRectangle: Es la variable ligada a la clase MBR; mediante esta, podremos acceder a los puntos bottomLeft y upperRight.
- groupClusters: Este vector guardará a todos los clusters que generemos (dependiendo de cuantos kCentroides le demos), por eso es de tipo vector<ClusterGroup>

Métodos:

- constructor: La clase constructor es el que guarda la lógica del K-means para crear, mover, etiquetar y reetiquetar "n" veces para crear estos grupos.

```
1  Input: -
2  Output: -
3  calculareMBR();
4  createClusters();
5  // -> Bucle Principal
6  bool flag;
7  while flag do
8  |   moveCentroids();
9  |   labelData();
10 |   flag = checkChanges();
11 end
```

Algorithm 1: constructor

- calculareMBR: Encargada de calcular y guardar dentro de las variables bottonLeft y upperRight los límites del Mínimum Bouding Rectangle

- labelData: Esta función lo que hace es recorrer todos los datos ingresados, y evaluar uno a uno, a qué centroide es más cercano; para esto, utiliza la función getEuclideanDistance(). Algo importante aquí, es que antes de borrar los puntos que ya tenía guardados e ingresar los nuevos, guarda en la variable dataClusterAux dichos puntos.

- createClusters: Simplemente, llama a la función createCentroid de la clase ClusterGroup y aprovecha para hacer el primer "etiquetado"de los puntos.

- moveCentroids: Esta funcion accede a las coordenadas del centroide de cada cluster generado, modificándolo para que este se encuentre en el centro de todos sus puntos. Utiliza la técnica de cálculo de promedio para todas las dimensiones.

- checkChanges: Esta función retorna una variable booleana. Retornará verdadero, si ocurrió algún cambio de puntos entre clusters; y retornara falso, si es que no lo hubo.

3 Resultados

3.1 Ejecución del Algoritmo

Ya que el K-Means utiliza la aleatoriedad como punto de partida para posicionar los centroides en el primer momento, los grupos se ven definitivamente influenciados por estos valores.
Se podría clasificar como la siguiente imagen en el mejor de los casos:

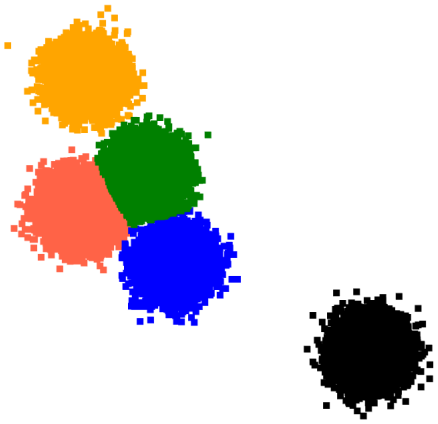

Figure 3.1: El mejor caso

Como también podríamos encontrar casos donde los puntos estén muy mal posicionados, dándonos resultados como:

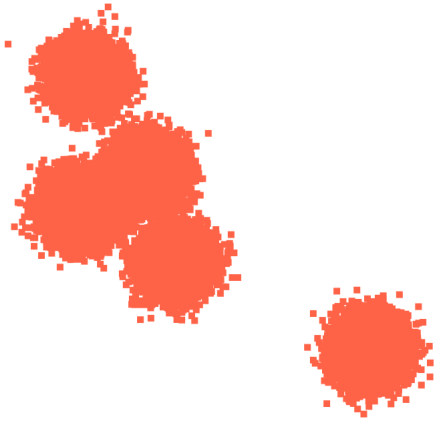

Figure 3.2: El peor caso

A continuación, les mostraremos algunos otros resultados que se crearon, indicándole al programa que deseamos que cree 5 Clusters

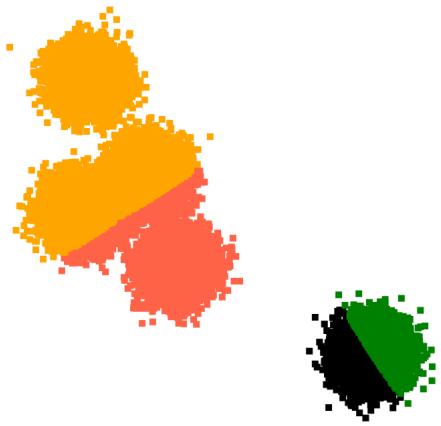

Figure 3.3: Resultado 1

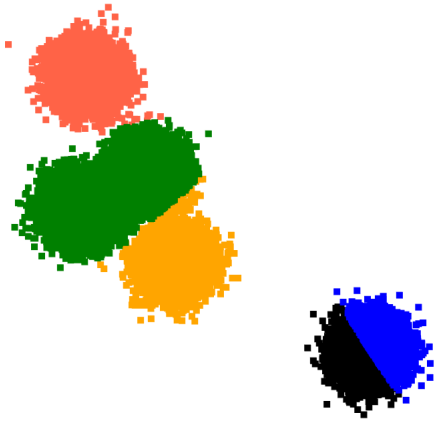

Figure 3.4: Resultado 2

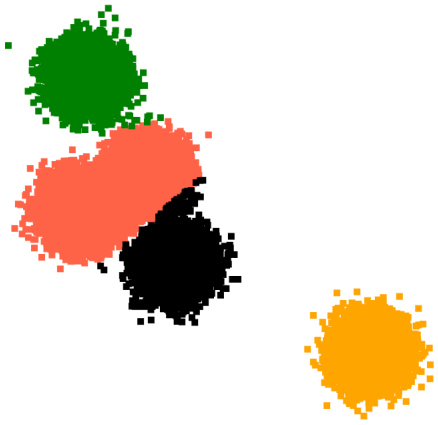

Figure 3.5: Resultado 3

3.2 Comparación lineal vs. paralela

3.2.1 Utilizando 2 centroides

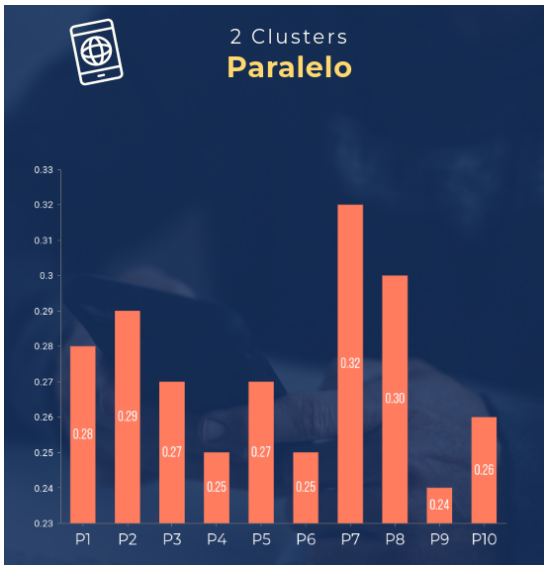

Figure 3.6: Versión Paralela

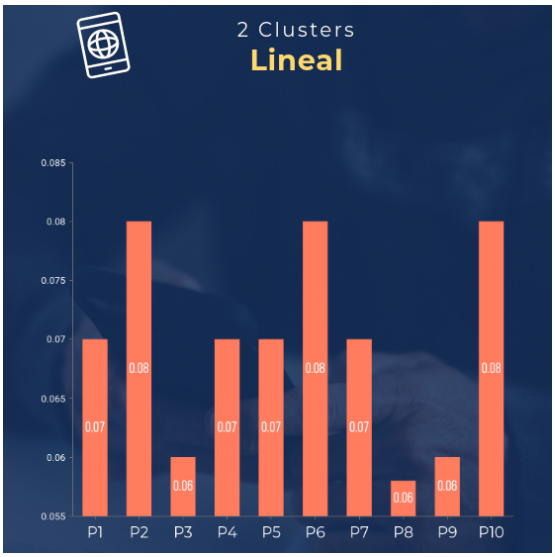

Figure 3.7: Versión Lineal

3.2.2 Utilizando 5 centroides

Figure 3.8: Versión Paralela

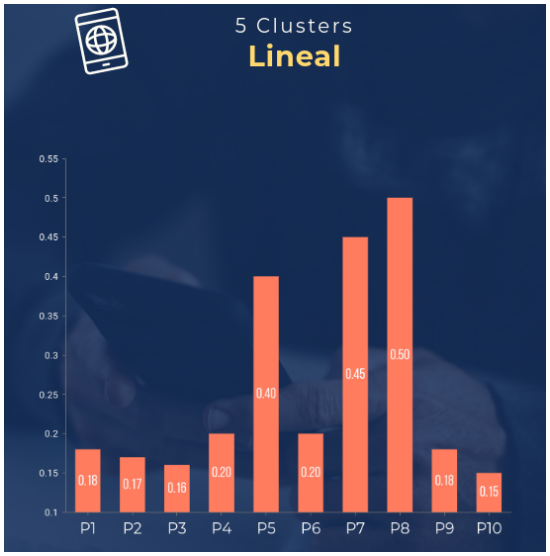

Figure 3.9: Versión Lineal

3.2.3 Resumen

Estos resultados fueron obtenidos promediando los resultados obtenidos en los graficos anteriores

	2 Centroides	5 Centroides
Lineal	0.064	0.259
	2 Centroides	5 Centroides
Paralela	0.273	0.449

References

[1] Javier Cuenca. Programación con openmp. Presentación de PowerPoint.

[2] The VTK Examples Project. Coloredpoints - vtk example. https://examples.vtk.org/site/Cxx/PolyData/ColoredPoints/. Accessed on June 1, 2023.