**X.y Análisis del Código: “Algoritmo K-Means con mlpack en c++”**

Se incluye los siguientes archivos de encabezado propios del mlpack más una librería básica del c++, asimismo se usarán los namespaces para evitar saturar el código mas adelante

#include <mlpack/core.hpp> // incluye la base componentes necesarios

#include <mlpack/methods/kmeans/kmeans.hpp> // libreria mlpack for k-means

#include <armadillo> // dependencia de mlpack con armadillo

#include <bits/stdc++.h> // incluye librerias de c++ básicas

using namespace arma;

using namespace mlpack;

using namespace std;

A continuación, se declara algunas variables básicas para establecer, el número de muestras totales de nuestro dataset (n), el número de clústeres (k), y la cantidad máxima de iteraciones que queremos hacer. ¿Por qué? Porque K-means es un proceso iterativo.

Además, se crea una variable tipo mat para contener los centroides de cada cluster, y un Row para guardar el cluster al cual pertenece cada muestra (cada cluster es representado por un número).

const int n = 500, k = 3, max\_iter = 2;

mat dataset;

mat centroids;

Row<size\_t> clusters;

Luego declaramos el prototipo de la función que nos permitirá apreciar las agrupaciones hechas por el programa, gracias a un gráfico que se verá en la parte final.

void imprimir\_grafica();

Ahora, trabajaremos en la función main. En primer lugar cargamos nuestro dataset bajo del nombre de "data.csv" y lo alojamos en nuestra matriz dataset.

data::Load("data2.csv", dataset);

Una vez cargada la data, necesitamos crear una instancia de la clase KMeans, para luego inmediatamente especificar la cantidad de iteraciones máximas y pasarlo al constructor.

kmeans::KMeans<> mlpack\_kmeans(max\_iter)

Así que ahora podemos seguir adelante y hacer el clustering. Llamaremos a la función de miembro Cluster de esta clase K-means. Necesitamos pasar los datos,

el número de clústeres, y luego también necesitamos pasar el objeto del clúster y el

objeto del centroide.

Ahora, esta función de clúster ejecutará K-means en estos datos con un número específico de clústeres, y luego inicializará estos dos objetos: clústeres y centroides

mlpack\_kmeans.Cluster(dataset, k, clusters, centroids);

.

Para ver los resultados por parte de los centroides, imprimiremos en la consola los resultados, para ello realizamos simplemente lo siguiente:

centroids.print("Centroides");

Asimismo se llama a la función imprimir\_grafica(), el código de la función se explica a continuación.

En primer lugar se puede notar que la funcion es de tipo void y no se necesita ingresar ninguna variable, esto debido a que las variables que usaremos serán de tipo global, declaradas estratégicamente al inicio del programa.  
Al inicio se declara las variables internas max, x, y, p, c; donde cada una me representa un par de coordenadas auxiliares, una matriz de enteros inicializada con ceros, en la que cada elemento me representará el cluster al cual pertenece cada muestra (raw=absica, col=ordenda), y un arreglo tipo char para cada cluster que se imprimirá en la consola, respectivamente.

void imprimir\_grafica() {

const int max = 25;

int x, y, p[max][max] = {};

char c[5] = { 207,206,244,170,177 };

Luego, se guarda la información de cada cluster respectivo en la matriz p, es importante mencionar que el objeto cluster diferencia a cada grupo según un número iniciando a partir de 0, pero para nuestro caso representaremos al 0 como un dato vacío, es por ello que se le suma una unidad al final.

for (int i = 0; i < n; i++) {

x = dataset(0, i);

y = dataset(1, i);

p[x][y] = clusters(i) + 1;

}

Para ver los centroides de cada cluster en la consola, asignaremos según su ubicación el elemento k+1 (reescribiendo sobre una muestra si es necesario) en nuestra matriz p.

for (int i = 0; i < k; i++) {

x = int(centroids(0, i));

y = int(centroids(1, i));

p[x][y] = k+1;

}

Para un mejor entendimiento se muestra el Recuadro 1.1

p[i][j]

0 ***Dato Vacío***

1 Cluster 1

2 Cluster 2

3 Cluster 3

...

k-1 Cluster k-1

k *Centroide*

Finalmente imprimimos en la consola el símbolo del cluster que le corresponde a cada muestra, en caso que no existe (p[i][j]=0), simplemente imprimiremos espacios en blanco.

cout << "\n\t Grafica \n\n";

for (int j = max-1; j >= 0; j--) {

for (int i = 0; i < max; i++) {

switch (p[i][j]) {

case 1:

cout << " " << c[1]; break;

case 2:

cout << " " << c[2]; break;

case 3:

cout << " " << c[3]; break;

case 4:

cout << " " << c[4]; break;

default:

cout << " ";

}

} cout << endl;

} cout << endl;

}