Laboratorio 05 EDA

Pablo Luis Carazas Barrios

November 2023

1 Introduction

En el ámbito de datos multidimensionales, una de las principales dificultades que enfrentamos es la agrupación eficiente de datos, especialmente cuando no disponemos de una representación gráfica clara debido a la alta dimensionalidad de los mismos. En estas situaciones, se han desarrollado diversas técnicas y algoritmos para abordar este desafío. Estas técnicas varían desde enfoques simples basados en la proximidad euclidiana hasta métodos más sofisticados que consideran relaciones complejas entre valores cercanos. En este informe, se presentará una implementación del algoritmo **K-means**, uno de los enfoques más ampliamente utilizados para la agrupación de datos en el contexto de minería de datos y aprendizaje automático.

2 K-means

El algoritmo K-means es un enfoque relativamente simple pero efectivo para la agrupación de datos. En su funcionamiento, se comienza generando de manera aleatoria una cantidad `K` de centroides, que actúan como representantes iniciales de los grupos. Luego, cada centroide mide su distancia con respecto a todos los puntos de datos proporcionados. Los puntos que están más cerca de un centroide se asignan a ese grupo.

Posteriormente, se recalcula la posición de cada centroide como el punto medio de todos los puntos asignados a ese grupo. Este proceso se repite iterativamente hasta que los centroides convergen, es decir, hasta que no haya un cambio significativo en las asignaciones de puntos a grupos.

El algoritmo K-means busca minimizar la suma de las distancias al cuadrado entre cada punto y su centroide asignado, lo que resulta en una partición efectiva de los datos en `K` grupos distintos. Este enfoque es ampliamente utilizado en aplicaciones de agrupación y segmentación de datos en diversas disciplinas.

3 Código

En esta sección, se mostrará la implementación que he realizado del algoritmo K-means:

3.1 Includes

Este codigo a parte de las librerías comunes utiliza `SFML` para poder realizar la parte grafica del codigo.

```
#include <iostream>
#include <vector>
#include <random>
#include <fstream>
#include <sstream>
#include <cmath>
#include <SFML/Graphics.hpp>

using namespace sf;
using namespace std;
```

3.2 Funcion generateRandomNumber

Esta funcion se encarga de generar numeros decimales aleatorios dentro de un rango dado en la parte de parametros

```
double generateRandomNumber(double min = 0, double max = 10) {
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<double> dist(min, max);

    return dist(gen);
}
```

3.3 Struct Point

Esta implementación de K-means tiene como objetivo encontrar clusters en N-dimensiones. Para lograr esta escalabilidad en función de los datos proporcionados, hemos diseñado la estructura 'Point'. Esta estructura incluye un vector para almacenar las coordenadas del punto en el espacio multidimensional, una variable entera que registra a qué cluster pertenece el punto y un valor de tipo 'double' llamado 'ws', que almacena la distancia del punto a su cluster. Esta estructura es esencial para representar y gestionar eficazmente los puntos de datos en un espacio de alta dimensionalidad.

```
struct Point {
    vector<double> Dimensions;
    int cluster;
    double ws = INT_MAX;
};
```

3.4 Funcion Euclidean

Esta funcion es simple, recibe dos puntos y regresa la distancia que tienen, esta funcion considera todas las dimensiones de los puntos y devuelve una distancia euclidiana de estas dimensiones.

```
double Euclidean(Point a, Point b) {
    double total = 0;
    for (int i = 0; i < a.Dimensions.size(); i++) {
        total = total + pow((a.Dimensions[i] - b.Dimensions[i]), 2);
    }
    total = sqrt(total);
    return total;
}
```

3.5 Funcion Maxmin

Esta función se encarga de encontrar los valores más altos y más bajos de cada dimensión en un conjunto de puntos. Recibe como parámetros un vector de puntos en el cual se realizará la búsqueda, así como dos puntos, 'bottom' y 'top', pasados por referencia. Los valores encontrados se almacenan en estos dos puntos.

```
void maxmin(vector<Point*>& points, Point& bottom, Point& top) {
    for (int i = 0; i < points.size(); i++) {
        for (int j = 0; j < points[i]->Dimensions.size(); j++) {
            if (points[i]->Dimensions[j] > top.Dimensions[j]) {
                top.Dimensions[j] = points[i]->Dimensions[j];
            }
            if (points[i]->Dimensions[j] < bottom.Dimensions[j]) {
                bottom.Dimensions[j] = points[i]->Dimensions[j];
```

```
            }
        }
    }
}
```

3.6 Struct Kmeans

La estructura 'Kmeans' se compone de los siguientes atributos:

- 'points': Un vector que almacena todos los puntos de datos que se utilizarán en el algoritmo.
- 'k_clusters': Una variable entera que representa la cantidad de clusters que se desean usar en el proceso de agrupación.
- 'centroids': Un vector de puntos que almacena los centroides de los clusters.
- 'Kgroup': Un vector de vectores de puntos que se utiliza para asignar cada punto a su cluster correspondiente.
- 'bottom' y 'top': Dos puntos que almacenan los límites de los puntos de datos en todas las dimensiones.

La estructura 'Kmeans' se inicializa con la cantidad de clusters deseados ('K') y se ajusta a medida que se realiza la agrupación de datos.

```
struct Kmeans {
    vector<Point*> points;
    int k_clusters;
    vector<Point*> centroids;
    vector<vector<Point*>> Kgroup;
    Point bottom;
    Point top;

    Kmeans(int K) {
        k_clusters = K;
        Kgroup.resize(K);
    }
```

3.7 Funcion setPoints

Esta funcion solo se encarga de guardar los puntos en la estructura kmeans.

```
    void setPoints(vector<Point*> v) {
        points = v;
        bottom.Dimensions.resize(v[0]->Dimensions.size(), INT_MAX);
        top.Dimensions.resize(v[0]->Dimensions.size(), INT_MIN);
    }
```

3.8 Funcion CreateCluster

Esta funcion obtiene el maximo y el minimo de todos los puntos, luego, segun la cantidad de clusters que fue indicada se van creando y generan puntos aleatorios entre el maximo y minimo dado

```
    void createClusters() {
        maxmin(points, bottom, top);
        for (int i = 0; i < k_clusters; i++) {
            centroids.push_back(new Point);
            for (int j = 0; j < points[0]->Dimensions.size(); j++) {
```

```
            centroids[i]->Dimensions.push_back(generateRandomNumber
                    (bottom.Dimensions[j], top.Dimensions[j]));
        }
    }
}
```

3.9 Funcion assingCluster

La función 'assignClusters' se encarga de asignar cada punto al cluster más cercano. El proceso de asignación se realiza de la siguiente manera: se recorren todos los puntos y, para cada punto, se calcula su distancia a todos los centroides. Si la distancia a un centroide específico es menor que el valor de 'ws' (distancia actual al cluster) de ese punto, se actualiza 'ws' con el nuevo valor y se asigna el punto al índice del cluster correspondiente.

```
void assignClusters() {
    for (int i = 0; i < points.size(); i++) {
        for (int j = 0; j < centroids.size(); j++) {
            double aux = Euclidean(*points[i], *centroids[j]);
            if (aux < points[i]->ws) {
                points[i]->ws = aux;
                points[i]->cluster = j;
            }
        }
        Kgroup[points[i]->cluster].push_back(points[i]);
    }
}
```

3.10 Funcion Mv_clusters

Esta funcion recorre el vector de centroides y obtiene los puntos maximos y minimos y luego calcula el punto medio del cluster en todas las dimensiones y guarda esa nueva posicion del centroide del cluster

3.11 Funcion assingCluster

Esta funcion recorre todos los puntos, luego por cada centroide calcula la distancia, si esta distancia es menor que el `ws` de ese punto guardará ese nuevo `ws` y asignará que pertenezca al indice del cluster que tiene esa minima distancia

```
void Mv_Clusters() {
    for (int i = 0; i < centroids.size(); i++) {
        Point b;
        b.Dimensions.resize(centroids[i]->Dimensions.size(), INT_MAX);
        Point t;
        t.Dimensions.resize(centroids[i]->Dimensions.size(), INT_MIN);
        maxmin(Kgroup[i], b, t);

        for (int j = 0; j < centroids[i]->Dimensions.size(); j++) {
            centroids[i]->Dimensions[j] =(b.Dimensions[j]+t.Dimensions[j]) / 2.0;
        }
    }
}
```

3.12 Funcion centroidUnchanged

La función 'centroidUnchanged' recibe un vector de puntos que representan los centroides antiguos y un número de iteración. Si es la primera iteración (cuando 'isFirstIteration' es igual a 0), la función devuelve 'false' ya que no se realiza ninguna comparación en la primera iteración. Sin embargo, en

iteraciones posteriores, la función compara los centroides antiguos con los nuevos centroides. Si la distancia entre algún par de centroides es diferente de 0, asume que el algoritmo no ha convergido y devuelve 'false'. En cambio, si todas las distancias entre centroides son iguales a 0, la función devuelve 'true', indicando que se ha alcanzado la convergencia.

```
bool centroidUnchanged(vector<Point*>& oldcluster, int isFirstIteration) {
    if (isFirstIteration == 0) {
        return false;
    }
    for (int i = 0; i < k_clusters; i++) {
        double distance = Euclidean(*centroids[i], *oldcluster[i]);
        if (distance != 0) {
            return false;
        }
    }
    return true;
}
```

3.13 Función algorithm

La función 'algorithm' es fundamental en la implementación del algoritmo K-means. Comienza creando los clusters utilizando la función 'createClusters'. Luego, se inicia un bucle en el que se asigna cada punto al cluster más cercano mediante 'assignClusters' y se ajustan los centroides con 'Mv_Clusters'. Se verifica si los centroides han cambiado de lugar, excepto en la primera iteración, y se actualiza el contador de iteraciones. Para rastrear los centroides antiguos, se utiliza un vector llamado 'oldcentroid', que se actualiza en cada iteración. Finalmente, el bucle continúa hasta que los centroides dejen de cambiar significativamente, y se muestra el número total de iteraciones realizadas antes de converger.

```
void algorithm() {
    createClusters();
    int iteraciones = 0;
    vector<Point*> oldcentroid(k_clusters);
    while (true) {
        assignClusters();
        Mv_Clusters();
        if (centroidUnchanged(oldcentroid, iteraciones)) {
            break;
        }
        iteraciones++;
        oldcentroid.clear();
        for (int i = 0; i < k_clusters; i++) {
            Point* newCentroid = new Point;
            newCentroid->Dimensions = centroids[i]->Dimensions;
            oldcentroid.push_back(newCentroid);
        }
        Kgroup.clear();
        Kgroup.resize(k_clusters);
    }
    cout << "Total iteraciones " << iteraciones;
}
```

3.14 Funcion printWss

Esta funcion se encarga de imprimir cada distancia de puntos a su centroide y también la suma total de esta

```
void printWss() {
    vector<double> Wss(k_clusters);
    for (int i = 0; i < centroids.size(); i++) {
        for (int j = 0; j < centroids[i]->Dimensions.size(); j++) {
            Wss[i] += Kgroup[i][j]->ws;
        }
    }
    double aux=0;
    cout << endl;
    for (int i = 0; i < centroids.size(); i++) {
        aux += Wss[i];
        cout <<"El cluster " << i << " tiene un wss de " << Wss[i]<<endl;
    }
    cout << "El wss total es: " << aux;
}
```

3.15 Funcion readCSVFile

Esta funcion lee un archivo csv y crea un vector de puntos con esos valores

3.16 Main

Respecto a la parte del algoritmo esta es la parte final solo se guardan los valores en la estructura kmeans, se dice cuantos clusters se quieren y se llama a la funcion algorithm

```
vector<Point> data = readCSVFile("dataset.csv");
Kmeans kmeans(3);
vector<Point*> dataPtrs;
for (Point& p : data) {
    dataPtrs.push_back(&p);
}
kmeans.setPoints(dataPtrs);
kmeans.algorithm();
kmeans.printWss();
```

3.17 Codigo Graficador

En esta parte explicaremos de gran manera como se realizaron los graficos.

3.17.1 Función createCircle

La función 'createCircle' forma parte de la implementación que utiliza una biblioteca gráfica en C++. Esta función toma varios parámetros, incluido el punto que se desea graficar, el color deseado, el tamaño de la ventana gráfica, los valores máximos y mínimos de todos los puntos, un indicador booleano para identificar si el punto es un centroide y el tamaño.

Esta función se encarga de crear círculos proporcionales en la ventana gráfica. La razón para utilizar proporciones en lugar de las coordenadas originales radica en la necesidad de adaptar la visualización de los puntos a las dimensiones de la ventana gráfica. A continuación, se explica este concepto con la ayuda de dos figuras:

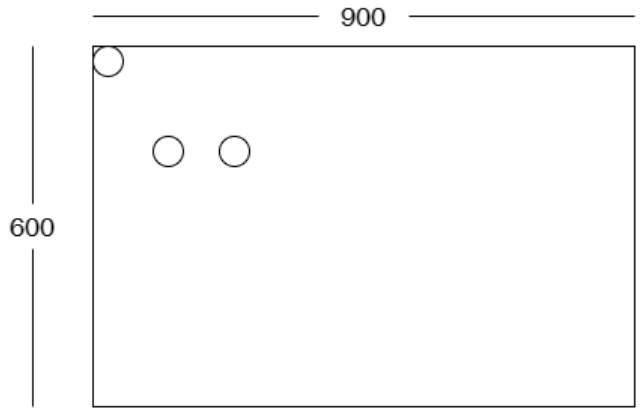


Figure 1: Distribucion normal

La Figura 1 representa una distribución normal de puntos en la pantalla, utilizando las coordenadas originales. Sin embargo, esta representación podría ser problemática cuando las coordenadas de los puntos son muy pequeñas o muy grandes. Por lo tanto, la función 'createCircle' considera las dimensiones totales de la ventana gráfica y reescala la imagen para que los puntos se visualicen adecuadamente, como se muestra en la Figura 2.

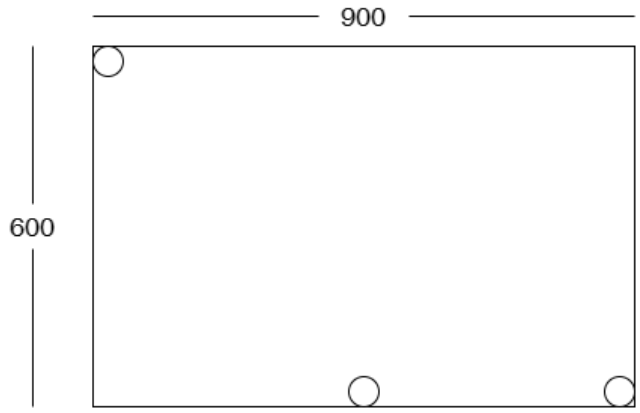


Figure 2: Distribucion reescalada

El reescalado al tamaño total de la ventana permite visualizar los mismos puntos de la figura original, pero en una escala que se adapta a la pantalla. Es importante destacar que esta transformación puede dar como resultado una deformación visual de los puntos en función del tamaño de la ventana gráfica. Sin embargo, esta distorsión no afecta la funcionalidad del algoritmo y solo impacta la representación gráfica, por lo que no representa un problema significativo en el contexto del algoritmo K-means.

```
CircleShape createCircle(Point* p, Color c, const Vector2u& windowSize, double minX,
                         double maxX, double minY, double maxY, bool cen, int tam = 10) {
    CircleShape cir(tam);
    double scaledX = ((p->Dimensions[0] - minX) / (maxX - minX));
    double scaledY = 1.0 - ((p->Dimensions[1] - minY) / (maxY - minY));
    float x = 50+(scaledX * (windowSize.x - 100));
    float y = 50+(scaledY * (windowSize.y-100));
    cir.setPosition(x, y);
    cir.setOrigin(tam / 2, tam / 2);
    cir.setFillColor(c);
    if (cen == false) {
```

```
        cir.setOutlineColor(c);
    }
    else {
        cir.setOutlineColor(Color::Black);
    }
    cir.setOutlineThickness(1);
    return cir;
}
```

3.17.2 While Graficador

SFML utiliza un bucle 'while' para graficar constantemente imágenes por segundo en una ventana. Dentro de esta función, creamos un bucle que itera a través de la cantidad de vectores dentro de 'KGroup'. Dependiendo del índice (cluster), asignamos un color diferente a los puntos de ese cluster y, finalmente, pasamos todos sus valores a nuestra función que grafica los puntos en la ventana.

```
    Color c;
    for (int i = 0; i < kmeans.Kgroup.size(); i++) {
        if (i == 0) {
            c = Color::Red;
        }
        else if (i == 1) {
            c = Color::Blue;
        }
        else if (i == 2) {
            c = Color::Green;
        }
        else if (i == 3) {
            c = Color::Yellow;
        }
        else if (i == 4) {
            c = Color::Magenta;
        }
        else if (i == 5) {
            c = Color::Cyan;
        }
        for (int j = 0; j < kmeans.Kgroup[i].size(); j++) {
            window.draw(createCircle(kmeans.Kgroup[i][j], c, window.getSize(),minX,maxX,
                        minY, maxY, false,7));
        }
        window.draw(createCircle(kmeans.centroids[i], c, window.getSize(), minX,maxX,
                        minY, maxY, true,15));
    }
```

4 Resultados

Una vez hemos utilizado el dataset Iris y clusterizar sus valores utilizando sus **4 dimensiones**, tendremos estos resultados:

Prueba	Iteraciones	Wss
1	4	10,7695
2	2	7,8451
3	3	8,0551
4	4	12,9421
5	3	12,0144
6	3	6,7726
7	3	10,8884
8	3	9,2590
9	3	8,4513
10	3	5,9665
promedio	3,1	9,2964

Table 1: Caption

A continuacion mostraremos y explicaremos lo ocurrido en las pruebas

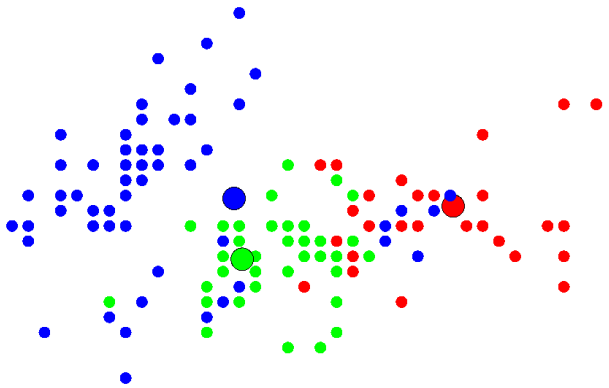


Figure 3: Mayor WSS

Lo que se puede apreciar acá es que el cluster Azul y Verde estan demasiado juntos (al menos en las 2 dimensiones visibles), esto lo que ocaciona es que al no estar bien distribuidos la distancia de uno de sus puntos al centroide sea mayor (de manera aproximada, al haber 2 dimensiones que no son visibles no se puede afirmar del todo)

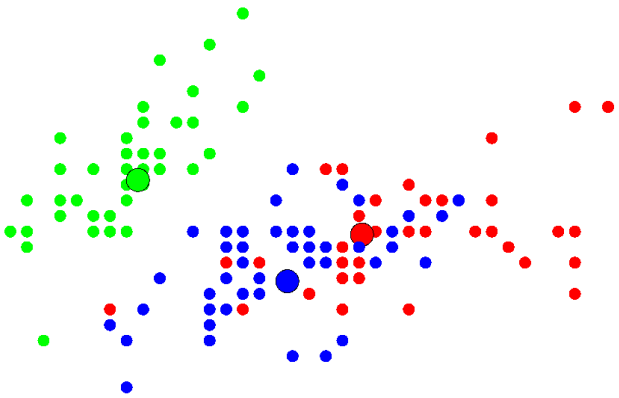
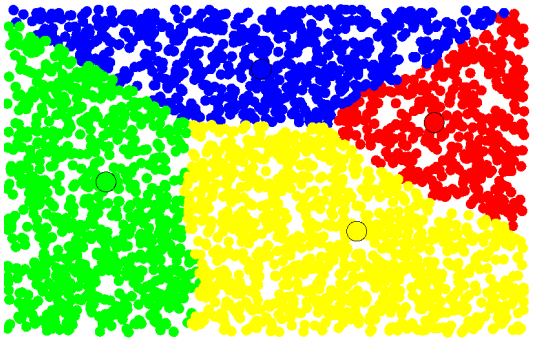


Figure 4: Menor WSS

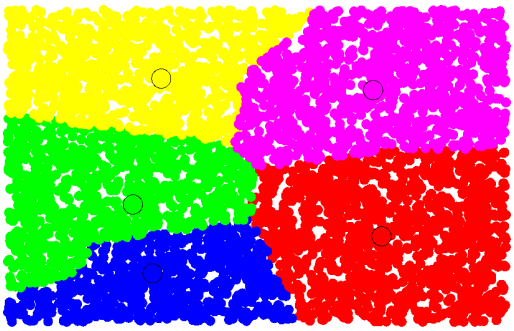
En este caso se puede ver una pequeña diferencia, hay una mayor distribucion en los clusters respectos a las dimensiones visibles, lo que puede influir para el calculo total.

4.1 Más pruebas

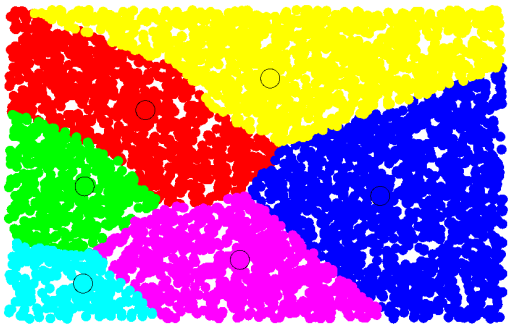
El codigo fue probado con pruebas con mayor grado de robustez respecto a cantidad de datos y clusters, consiguiendo estos resultados:



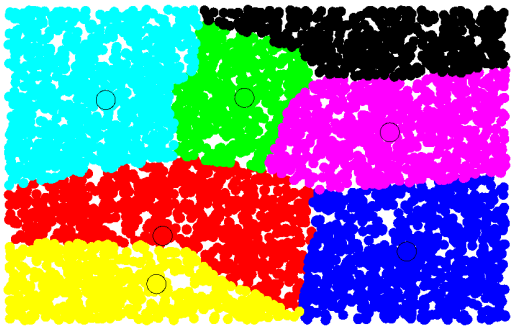
(a) 4000 puntosy 4 clusters



(b) 5000 puntos y 5 clusters



(c) 4000 puntosy 6 clusters



(d) 5000 puntos y 7 clusters

Como se puede ver el codigo es capaz de soportar grandes cantidades de datos y agrupar en varios clusters cumpliendo asi su labor.

5 Observaciones

Aunque visualmente se espera que el algoritmo agrupe por colores segun estén más cerca como en la figura 5d, en la prueba con menor WSS 4 los colores están cruzados aunque estén más cerca a otro cluster, esto se debe a que se estan considerando las 4 dimensiones para el algoritmo pero solo estamos graficando 2 dimensiones, por lo que aunque visualmente parezca estar más cerca a un centroide en realidad está mucho más cerca a otro centroide en las dimensiones que no son visibles.

6 Conclusiones

El algoritmo `K-means` se demuestra como una herramienta efectiva para agrupar grandes conjuntos de datos utilizando la proximidad euclidiana como criterio de agrupación. Su capacidad para realizar esta tarea en múltiples dimensiones es una característica valiosa para el análisis y la segmentación de datos. Sin embargo, es importante destacar que el rendimiento del algoritmo puede verse limitado en dimensiones muy altas. En tales casos, la alta dimensionalidad puede introducir desafíos adicionales, como la maldición de la dimensionalidad, que pueden afectar la calidad de los resultados. A pesar de estas limitaciones, el `K-means` sigue siendo una herramienta esencial en la agrupación de datos y puede proporcionar información valiosa en una variedad de aplicaciones.

7 Enlaces

Github con Codigo e informe