K-Means

Victor Gerardo Rodríguez Barragán

24 de Octubre de 2023

1 Descripcion del ejercicio y solucion

1.1 Ejericio

Implementar el algoritmo de agrupamiento K-means, el cual es un algoritmo de aprendizaje no supervisado que se encarga de agrupar un conjunto de datos en k grupos, donde cada grupo tiene un centroide, el cual es el promedio de todos los puntos del grupo. El algoritmo funciona de la siguiente manera:

1. Seleccionar k puntos aleatorios como centroides.
2. Asignar cada punto al centroide más cercano.
3. Actualizar los centroides.
4. Repetir los pasos 2 y 3 hasta que los centroides no cambien.

1.2 Solucion

La solucion se implemento en Rust y se utilizo OpenGL para visualizar los resultados.

El algoritmo se implemento de la siguiente manera:

1. **Generacion de datos:** Se generaron 1000 puntos de datos aleatorios en el rango de 0 a 800 en x y de 0 a 600 en y. Esos datos se utilizaron como datos de entrada para el algoritmo.
2. **Creacion de clusters:** Se crearon 6 clusters iniciales con centroides aleatorios.
3. **Asignacion de puntos:** Se asignaron los puntos a los centroides mas cercanos.
4. **Actualizacion de centroides:** Se actualizaron los centroides con el promedio de los puntos asignados.

2 Codigo

```rust
use engine::graphics::color::Color;
use engine::graphics::window::Window;
use engine::algorithms::fill::draw_array_circle;

const WIDTH: f32 = 800.0;
const HEIGHT: f32 = 600.0;

#[derive(Debug, Clone)]
struct Point {
    x: f32,
```

```
    y: f32,
}

#[derive(Debug)]
struct Cluster {
    points: Vec<Point>,
    centroid: Point,
}

// Function to calculate the distance between two points
// It will return a vector with the distance between the two points
fn kmeans() -> Vec<(f32, f32)>
{
    let mut points: Vec<Point> = Vec::new();
    let mut clusters: Vec<Cluster> = Vec::new();

    // Create 2000 random points
    for _ in 0..1000
    {
        let x = rand::random::<f32>() * WIDTH;
        let y = rand::random::<f32>() * HEIGHT;
        points.push(Point { x, y });
    }

    // Create 12 random clusters
    for _ in 0..6
    {
        let x = rand::random::<f32>() * WIDTH;
        let y = rand::random::<f32>() * HEIGHT;
        clusters.push(Cluster {
            points: Vec::new(),
            centroid: Point { x, y },
        });
    }

    // Iterate 100 times
    for _ in 0..100
    {
        // Assign the points
        for point in &points
        {
            let mut distance = std::f32::MAX;
            let mut cluster_id = 0;

            for (id, cluster) in clusters.iter().enumerate()
            {
```

```
                let dx = point.x - cluster.centroid.x;
                let dy = point.y - cluster.centroid.y;
                let d = dx * dx + dy * dy;

                if d < distance
                {
                    distance = d;
                    cluster_id = id;
                }
            }

            clusters[cluster_id].points.push(point.clone());
        }

        // Update centroids
        for cluster in &mut clusters
        {
            let mut x = 0.0;
            let mut y = 0.0;

            for point in &cluster.points
            {
                x += point.x;
                y += point.y;
            }

            cluster.centroid.x = x / cluster.points.len() as f32;
            cluster.centroid.y = y / cluster.points.len() as f32;
        }
    }

    let mut result: Vec<(f32, f32)> = Vec::new();

    for cluster in &clusters
    {
        result.push((cluster.centroid.x, cluster.centroid.y));
    }

    result
}

fn main()
{
    let mut window = Window::new(WIDTH, HEIGHT, "K-means");
    window.init();
```

```
    while !window.should_close()
    {
        unsafe
        {
            gl::ClearColor(0.0, 0.0, 0.0, 1.0);
            // gl::Clear(gl::COLOR_BUFFER_BIT);
            let v = kmeans();
            for (x, y) in &v
            {
                draw_array_circle(*x, *y, 10.0, Color::new(1.0, 1.0, 1.0, 1.0));
            }
        }
        window.update();
    }

}
```

3 Conclusion

3.1 Aprendizajes

Durante la implementacion aprendimos la mecanica detras del algoritmo de K-means y sus diferentes usos en la agrupacion de datos. Tambien experimente un poco con OpenGL para la visualizacion y utilice teoria de otras materias para implementarlo de manera mas eficiente.

3.2 Implementacion del algoritmo

Podríamos considerar implementar una mejor inicializacion para los centroides. Además, podria explorar métodos para determinar de manera automática el número óptimo de clusters (K) para un conjunto de datos.

3.3 Retos y dificultades

El algoritmo salio bien sin embargo visualizar los clusters no resultaba como esperaba.