Compilador de colores ultilizando el algoritmo k-means

Joaquín Farias, *Estudiante*

Resumen—En Este informe se presenta una aplicación para compilar los colores de una imagen mediante clustering utilizando el algoritmo k-means. Se explicara el algoritmo, además de mostrar los resultados de pruebas realizadas. También se hará un análisis del error con distintos valores de K.

Palabras clave—Compilador, RGB, k-means, clustering.

1 Introducción

El *clustering* o agrupamientos son algoritmos que se utilizan en distintos ámbitos para agrupar cierta cantidad de datos de acuerdo con un criterio dado. Generalmente, los datos (que en general, son vectores) de un mismo grupo comparten propiedades en común. Así, se busca, luego de asignar cada dato a un grupo específico, cambiar las características de los individuos de cada grupo por la de un represéntate característico de ese grupo.

Estos algoritmos son ampliamente usados en distintas ramas de la ciencia, así como también en el marketing. Algunos campos y aplicaciones distintivas son:

- Medicina, identificar enfermedades.
- Biología, clasificar animales y plantas.
- Marketing, identificar personas con hábitos de compa similares.

En particular, el algoritmo k-means es ampliamente usado por su fácil implementación para gran cantidad de datos. Además, también se usa para reprocesamiento para otros algoritmos.

Dentro de estos preprocesamientos mencionados existe la compilación de imágenes, que sirve especialmente para disminuir la memoria que ocuparía una imagen para su procesamiento, además de disminuir la el tiempo de procesamiento de algoritmos que analizan imágenes para cierto requerimiento.

2 Motivación y descripción del algoritmo k-means

El programa realizado es una aplicación para compilar colores de imágenes. La motivación para realizar este proyecto es que el análisis de imágenes para extraer sus características tenga un menor requerimiento computacional, disminuyendo la información dentro de la imagen, pero sin destruir la esencial que nos interesa. Además, también para disminuir la memoria que ocuparía una imagen guardada en un equipo de almacenamiento.

El programa consta de tres funciones principales:

1. **Compilador de imagen**.
2. **Recompilado de imagen**.
3. **Calculo de error**.

2.1 Función de compilador de imagen

Esta función realiza una compilación de la imagen utilizando el algoritmo *k-means*. Se separa la imagen original en dos matrices: la primera con la información de la cantidad de pixeles que posee la imagen original, además de contener la asignación de a qué grupo se asocia cada pixel. Mientras, que la segunda contiene la información de cada grupo.

El algoritmo *k-means* tiene cinco pasos principales:

1. **Paso inicial:** donde se asignan los K centroides de cada *cluster* o grupos iniciales de forma aleatoria dentro del espacio vectorial de nuestros datos.
2. **Paso de asignación inicial:** donde se asigna cada vector de nuestros dato a un *cluster*. Para

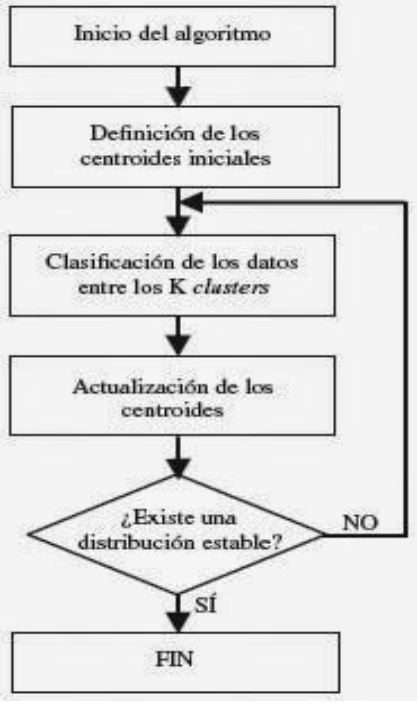

Fig. 1. Diagrama de flujo de algoritmo *k-means.*

lograr lo anterior se utiliza la fórmula de distancia euclidiana entre vectores, donde se asigna un vector al grupo que tenga menor distancia.

3. **Paso de actualización:** Donde se vuelve a calcular el valor de los centroides obteniendo el promedio de cada *cluster* con los miembros de este. En general se ocupa el promedio aritmético.
4. **Paso de reasignación:** donde se reasignan los vectores de nuestros datos a los nuevos centroides obtenidos.
5. **Obtención de resultado:** donde se extrae la información de la pertenencia de cada vector a su *cluster*.

La salida de esta función son dos matrices: una de dimensiones mxn, donde m seria el número de pixeles del alto de la imagen y n número de pixeles del ancho de la imagen. Y, la otra de dimensiones Kx3 donde K seria el número de *cluster* escogidos y 3 porque la imagen está compuesta de vectores RGB.

2.2 Función de recopilador de imagen

Esta función se encarga principalmente de recompilar la imagen uniendo las dos matrices obtenidas en el paso anterior. Esto lo hace, recorriendo la matriz de dimensiones mxn y de acuerdo a los datos dentro de ella asignarle un centroide k de la matriz de dimensiones Kx3.

Con lo anterior se forma la matriz de la imagen recompilada que se puede mostrar en pantalla para compararse con la original.

2.3 Función de cálculo de error de imagen recompilada

Esta función se encarga de calcular el error entre la imagen recompilada y la imagen original. Esto lo logra obteniendo el error relativo en cada vector RGB dentro la matriz de pixeles, es decir, se resta le resta la matriz de la imagen recompilada a la imagen original, luego se divide por 256 (número de bits que tiene cada vector RGB). Luego de esto, se suman los errores de todo los pixeles y se obtiene el promedio aritmético de errores.

Ahora, también sería interesante obtener el error de cada grupo por separado, pero no se hará dentro de este proyecto.

3 Código

En este capítulo se describirá de manera concisa los códigos principales de cada función ocupada.

El código principal de nuestro programa pide que se ingrese el valor de K al usuario para empezar su ejecución.

3.1 Código de función de compilador de imagen

En la *fig. 2* se puede apreciar el código empleado para esta función. Primero se toma una posición al azar para cada centroides k (*fig. 2 (a)*), luego se asigna cada vector a un *cluster* (*fig. 2 (b)*), para pasar al recalculo de centroides (*fig. 2 (c)*) y a la reasignación en los *cluster*. Los últimos dos pasos anteriores se repiten la cantidad de veces necesarias

```
%Establecer los primeros centroides
for i=1:K
    a = randperm(size(IMG_ORIGINAL,1),1);
    b = randperm(size(IMG_ORIGINAL,2),1);
    centros_inicial(i,:) = [R(a, b),G(a, b),B(a, b)];
end
```

(a)

```
%asignacion inicial
for i=1:size(IMG_ORIGINAL,1)
    for j=1:size(IMG_ORIGINAL,2)
        for k=1:K
            distancia(k) = sqrt(power(centros_inicial(k,1)-R(i,j),2) +
            power(centros_inicial(k,2)-G(i,j),2) +
            power(centros_inicial(k,3)-B(i,j),2));
            if (distancia(k)<cont_dist)
                    contador_cluster = k;
                    cont_dist = distancia(k);
            end
        end
        cluster(i,j) = contador_cluster;
        cont_dist = 9999999999999;
    end
end
```

(b)

```
%obtener nuevos centros de grupos
for k=1:K
    for i=1:size(IMG_ORIGINAL,1)
        for j=1:size(IMG_ORIGINAL,2)
            if (cluster(i,j)==k)
                sumaR = sumaR + R(i,j);
                sumaG = sumaG + G(i,j);
                sumaB = sumaB + B(i,j);
                cont = cont + 1;
            end
        end
    end
    if cont~=0
        promR = sumaR / cont;
        promG = sumaG / cont;
        promB = sumaB / cont;
        centros_new(k,:) = [round(promR), round(promG), round(promB)];
    end
    sumaR = 0;
    sumaG = 0;
    sumaB = 0;
    cont = 0;
end
```

(c)

```
%obtener nuevas asignaciones
cont_dist = 9999999999;
for i=1:size(IMG_ORIGINAL,1)
    for j=1:size(IMG_ORIGINAL,2)
        for k=1:K
            distancia(k) = sqrt(power(centros_new(k,1)-R(i,j),2) +
            power(centros_new(k,2)-G(i,j),2) +
            power(centros_new(k,3)-B(i,j),2));
            if (distancia(k)<cont_dist)
                    contador_cluster = k;
                    cont_dist = distancia(k);
            end
        end
        cluster(i,j) = contador_cluster;
        cont_dist = 99999999;
    end
end
```

(d)

Fig. 2. (a) Código de algoritmo para asignación de centroides iniciales. (b) Código para la asignación inicial en cada *cluster*. (c) Código para la actualización de centroides. (d) código para reasignación en cada *cluster*.

para que el algoritmo converja.

Después se extraen los resultados que serían la **cluster** y la matriz **centros_new**.

3.2 Código de fusión de recopilador de imagen

En la *fig. 3* se puede apreciar el código empleado para esta función, que es esencialmente la reunión de las dos matrices de la imagen compilada, recorriendo la matriz **IMG_COMP** y asignarle valores de los centroides contenidos en la matriz **IMG_IDX**.

```
%obtener los nuevos vectores RGB
for i=1:size(IMG_COMP,1)
    for j=1:size(IMG_COMP,2)
        for k=1:size(IMG_IDX,1)
            if IMG_COMP(i,j)==k
                Rp(i,j) = IMG_IDX(k,1);
                Gp(i,j) = IMG_IDX(k,2);
                Bp(i,j) = IMG_IDX(k,3);
            end
        end
    end
```

Fig. 3. Código de la recopilación de la imagen compilada.

3.3 Código de función de cálculo de error de imagen recompilada

En la *fig. 4* se puede apreciar el código empleado para esta función, que básicamente calcula el error relativo en cada posición de la matriz **IMG_RECON,** para luego calcula un promedio de errores de toda la matriz.

```
%calculo de error promedio de la imagen compilada
for i=1:size(IMG_ORIGINAL,1)
        for j=1:size(IMG_ORIGINAL,2)
            sum_errores = sum_errores + img_prom_erroresRGB(i,j);
        end
end
%calculo de errores en cada elemento de las matrices RGB
img_restaR = abs(R - Rc);
img_restaG = abs(G - Gc);
img_restaB = abs(B - Bc);
for i=1:size(IMG_ORIGINAL,1)
        for j=1:size(IMG_ORIGINAL,2)
            img_erroresR(i,j) = img_restaR(i,j) / 256;
            img_erroresG(i,j) = img_restaG(i,j) / 256;
            img_erroresB(i,j) = img_restaB(i,j) / 256;
        end
end

%calculo de error promedio en cada vector RGB
for i=1:size(IMG_ORIGINAL,1)
        for j=1:size(IMG_ORIGINAL,2)
            img_prom_erroresRGB(i,j) = img_erroresR(i,j) +
            img_erroresG(i,j) + img_erroresB(i,j);
        end
end
```

Fig. 4. Código de cálculo de error de la imagen recompilada.

4 PRUEBAS Y ANÁLISIS

Dentro de este capítulo se presentaran los resultados de la aplicación de nuestro programa, el análisis del tiempo de ejecución de las distintas funciones y el análisis de error.

4.1 Pruebas, resultados y ejemplos

Los resultados de pruebas hechas con una imagen se mientras en la *fig. 5* donde se aprecia cómo cambia la imagen con distintos valores de K. Se nota que a menor el valor de cada menos resolución tienen los colores de la foto y, por lo tanto, más alejada esta de la original en cuanto a coloreado.

4.2 Análisis de tiempo de ejecución de funciones

Se puede notar por el grafico de la *fig. 6* que el tiempo de ejecución sube a medida que se aumente los valores de K. Esto ocurre con todas las funciones de nuestro algoritmo en distintas medidas.

4.3 Análisis de error

Se nota por el grafico de la *fig. 7* que a medida que el K disminuye, el error aumenta. Esto es debido que como

(a)

(b)

(c)

(d)

Fig. 5. Imagen recompilada con distintos valores de K. (a) Imagen original. (b) K=32. (c) K=16. (d) K=8.

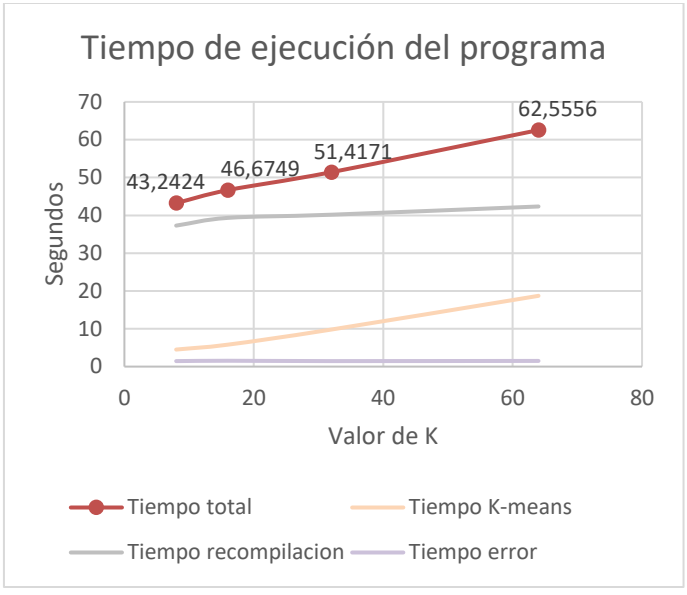

Fig. 7. Grafico que muestra los tiempos de ejecución del programa.

hay menos posibilidades de posibles colores se fuerza al algoritmo a agrupar ciertos colores en grupos que no son tan cernos a él, es decir, mientras más alejado un color del centroide de su *cluster* menos fidedigno es el nuevo color.

5 Conclusiones

Los resultados obtenidos están dentro de lo esperado de la ejecución de esta aplicación. Además, se nota cierta-

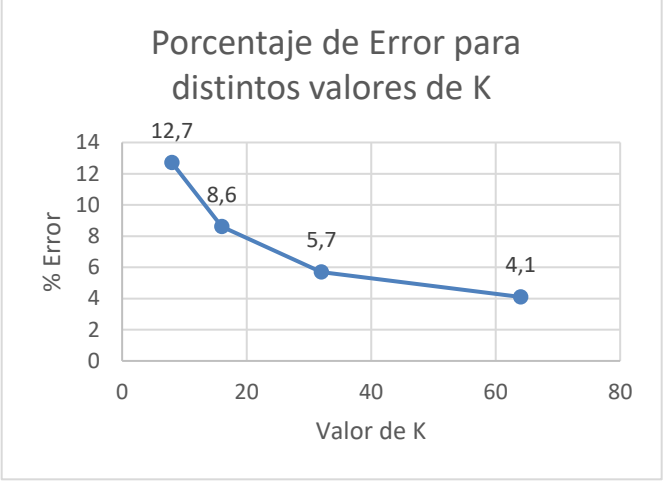

Fig. 7. Grafico que muestra el porcentaje de error para distintos valores de K.

mente la incidencia del valor de K en el error y el tiempo de ejecución del algoritmo, este tiempo sube de gran manera cuando el K tiene un valor alto.

Se nota, además, que los tiempos de ejecución del algoritmo serían menores si se dispusiera de un método para elegir de mejor manera los centros iniciales, ya que estos inciden en cuanto demora el algoritmo en converger.

Por último, se obtuvo experiencia en la aplicación del algoritmo k-means y se incurrió en pensar aplicaciones nuevas para él para proyectos futuros.

Referencias

[1] Boyd, S., & Vandenberghe, L. (2018). Introduction to applied linear algebra: vectors, matrices, and least squares. Cambridge university press.

[2] Norvig, P., & Russell, S. (2004). Inteligencia artificial. Editora Campus, 20.)

Joaquín Farias Muñoz Estudiante de ingeniería civil electrónica, Pontificia Universidad catolica de Valparaíso.