Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Aprendizaje automático

Practica 7: Agrupamiento

Autora:
Nerea Gallego Sánchez (801950)

7 de diciembre de 2022

Índice

1. Introducción y objetivos

El objetivo de la práctica es utilizar técnicas de agrupamiento de colores (en este caso).
Para ello se utiliza la técnica de KMeans para cuantificar imágenes.

Se va a utilizar el algoritmo para la cuantificación de imágenes.
La idea es representar la imagen como un grupo de K colores más pequeño que la paleta original. Para ello, se asigna un color a cada clúster.

2. Construcción de funciones auxiliares para implementar KMeans

En el primer apartado se pide construir funciones auxiliares que para implementar el algoritmo de KMeans.

Las funciones auxiliares que son necesarias para implementar el algoritmo son: inicializarCentroides que elige los centroides iniciales, más adelante se explica el criterio elegido para inicializarlos. updateClusters actualiza los clusters a los que pertenece cada muestra en función de los centroides que se tienen. updateCentroids actualiza los centroides en función de las muestras que pertenecen a él.

La primera función implementada es inicializarCentroides. Para ello, ha sido necesario elegir un método de inicialización de los mismos. Para evitar que dos centroides iniciales representen al mismo color y, alguno de ellos se quede sin muestras durante las iteraciones del algoritmo de KMeans, se ha decidido primero, eliminar las filas repetidas del conjunto de datos inicial y se han ordenado de menor a mayor. Para elegir los centroides repartidos (que se encuentren a distancias similares). Se han dividido los píxeles ordenados en k "slots" y se ha elegido de cada "slot", el píxel que se encuentra primero. De esta manera, todos los píxeles iniciales estarán a una disntancia similar.

```
% Se eligen k centroides (uno para cada cluster)
% Los centroides elegidos son muestras del conjunto de datos que estan
% separados entre si a igual distancia
function Xm = inicializarCentroides(data, k)
    % Se ordenan los datos de forma ascendente y se quitan los datos
    % repetidos para evitar que un color tenga mas posibilidades de salir
    % en dos centroides iniciales
    X = sort(unique(data,'rows'),'ascend');
    m = size(X,1);
    % El primer centroide corresponde con el pixel que aparece primero en
    % la lista ordenada
    Xm = [X(1,:)];
    % Se eligen el resto de muestras que van a actuar de centroides
    % iniciales de manera que se encuentren a la misma distancia.
    for i = 1:(k-1)
        Xm = [Xm ; X(int32(m*i/(k-1)),:)];
    end
end
```

En la función de updateCentroids se itera sobre los k clusters, y para cada cluster, se asigna como nuevo centroide la media de todas las muestras que pertenecen al cluster (y que se le han asignado anteriormente).

```
% Devuelve en munew k filas que contienen los nuevos centroides
% Se actualizan los centroides en funcion de las muestras que pertenecen a
% ese cluster
% El nuevo centroide es la media de todas las muestras que pertenecen al
% cluster dado
function munew = updateCentroids(D,c,K)
% D((m,n), m datapoints, n dimensions
% c(m) assignment of each datapoint to a class
%
% munew(K,n) new centroids
    munew = [];
    % Para cada uno de los k clusters
    for i=1:K
```

```
        % Se obtienen las muestras que pertenecen a ese cluster
        x = D(c==i,:);
        % Se a ade la media de las muestras de ese cluster
        munew = [munew ; mean(x,1)];
    end
end
```

La función updateClusters actualiza las etiquetas de cada muestra en función de los centroides de cada cluster. Se asigna a cada muestra la etiqueta del cluster que sea más cercano.
Para ello, se ha calculado la distancia que tiene la muestra a cada cluster, finalmente, se devuelve como etiqueta el índice del menor valor.
La distancia de la muestra a un cluster se ha definido como la norma al cuadrado del vector que hay desde una muestra hata el centroide de ese cluster. Por lo tanto, la distancia se cancula de la siguiente manera: $d = \| x^{(i)} - \mu_k \|^2$

```
% Acutualiza las etiquetas del cluster al que pertenece cada muestra
% Dadas las muestras de entrada y los centroides de cada cluster, devuelve
% en Z el indice del cluster al que pertenece cada muestra.
% Una muestra se asigna a un cluster si el centroide de dicho cluster es el
% que esta a menor distancia.
% Se define la distancia como la norma del vector entre la muestra y el
% centroide al cuadrado.
function Z = updateClusters(D,mu)
% D(m,n), m datapoints, n dimensions
% mu(K,n) final centroids
%
% c(m) assignment of each datapoint to a class
    K = size(mu,1);
    m = size(D,1);
    % Guarda en la columna i, las distancias de cada muestra al cluster i
    distancia = [];
    for i = 1:K % dsearch
        distancia = [distancia sum((D-mu(i,:)).^2,2)];
    end
    % Devuelve los clusters que est n a menor distancia
    [~, Z] = min(distancia,[],2);
end
```

Para construir el algoritmo de kMeans se han utilizado las funciones construidas anteriormente. La función tiene como entrada los datos iniciales y los centroides iniciales. El algoritmo calcula unas etiquetas iniciales y recalcula los centroides en función de las muestras que se han asignado a cada cluster. El algoritmo itera recalculando las etiquetas y los centroides de los clusters hasta que en dos iteraciones consecutivas no varíen las etiquetas de las muestras. La función devuelve las etiquetas para los píxeles y los centroides de los clusters.

```
% Algoritmo de kmeans
% Dados los datos de entrada y los centroides iniciales, devuelve las
% etiquetas de las muestras (a que cluster pertenecen) y los centroides de
% dichos clusters.
% Realiza iteraaciones en las que se actualizan las etiquetas y los
% centroides hasta que en dos iteraciones consecutivas no varie la etiqueta
% de ninguna de las muestras
function [mu, c] = kmeans(D,mu0)
% D(m,n), m datapoints, n dimensions
% mu0(K,n) K initial centroids
%
% mu(K,n) final centroids
% c(m) assignment of each datapoint to a class
    c = updateClusters(D,mu0);
    mu = updateCentroids(D,c,size(mu0,1));
    c0 = c*0;
    while sum(c0 ~= c) > 0
        c0 = c;
        c = updateClusters(D,mu);
        mu = updateCentroids(D,c,size(mu0,1));
```

Figura 1: Imagen inicial del loro.

```
    end
end
```

3. Utilizar KMeans

Se pide utilizar el algoritmo de KMeans para agrupar los pixeles RGB con la matriz de prueba. La matriz de prueba corresponde con los píxeles RGB de la imagen del loro (figura 1).

Para agrupar los colores en clusters, se ha utilizado la función kMeans. Inicialmente, el código proporcionado por los profesores realizaba un agrupamiento a 16 colores distintos. El resultado de agrupar la imagen del loro a 16 colores se muestra en la figura 2. Como se puede observar, la calidad de la imagen se mantiene y se pueden distinguir las formas y texturas que tenía el loro inicialmente.

Para elegir el número de clusters adecuado, se ha decidido utilizar la regla del codo.
La regla del codo indica que hay un número de clusters para que el coste del agrupamiento comienza a disminuir más lentamente. Se considera que el valor donde se produce el codo es un valor adecuado para el número de clusters.

Para ello, se han realizado iteraciones con distinta cantidad de clusters. Con la función funcionCoste se ha calculado el coste de las muestras dadas con respecto al cluster al que han sido asignadas. El coste se define como la media de la norma al cuadrado de las muestras con respecto al cluster al que fueron asignadas. El coste se calcula de la siguiente manera: $J(c^{(1)},...,c^{(m)},\mu_1,...,\mu_k) = \frac{1}{m}\sum_{i=1}^{m} \parallel x^{(i)} - \mu_k \parallel^2$

```
function J = funcionCoste(D,mu,c)
    m = size(D,1);
    J = sum(sum((D-mu(c,:)).^2,2))./m;
end
```

De cada posible cantidad de clusters a valorar, se calcula el coste y se almacena en un array. Se han cosiderado entre 3 y 20 clusters distintos para la imagen.
En la figura 3 se puede observar como a cada iteración el coste disminuye. La regla del codo indica que hay una cantidad de clusters en la que el coste continua disminuyendo pero a menor velocidad. Como se puede observar en la imagen, el cluster óptimo a partir del cual comienza a disminuir el coste más lentamente es con 7 clusters.

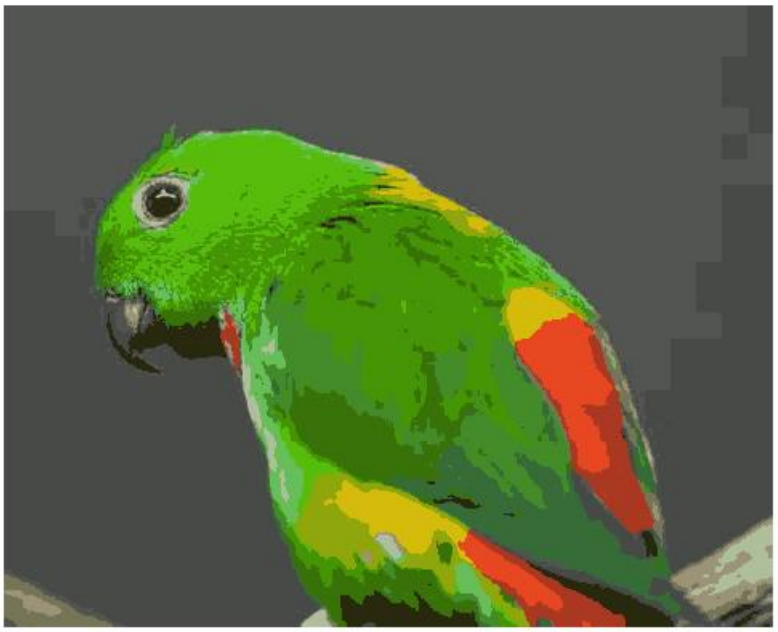

Figura 2: Imagen del loro con 16 colores.

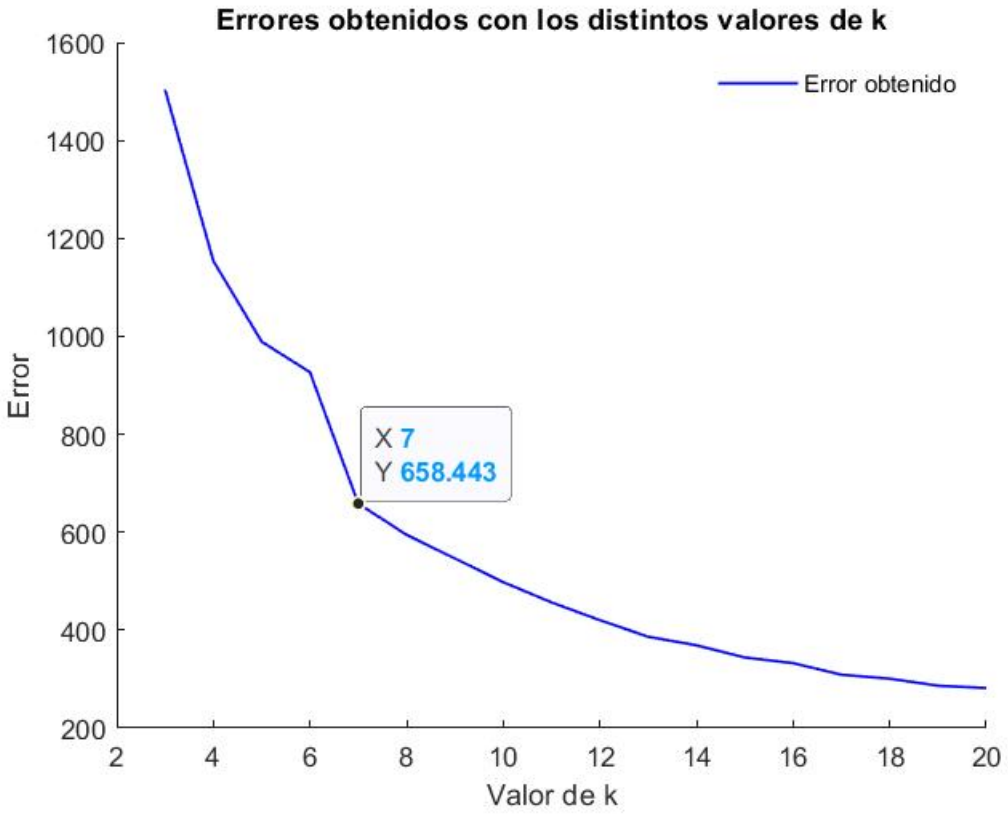

Figura 3: Curva de coste que se obtiene con la imagen del loro.

Figura 4: Imagen del loro con 7 colores.

Por lo tanto, se ha obtenido la imagen utilizando únicamente 7 colores. Como se puede observar en la imagen 4 se ve perfectamente la forma y textura del loro. Únicamente se pierde algo de información acerca del fondo y algunas texturas y sombras de la imagen.

A continuación, se pide utilizar otras imágenes para probar el algoritmo de KMeans.
Para realizar dicha prueba se ha decidido utilizar la imagen 5. Esta nueva imagen, al igual que la imagen del loro contiene muchos colores distintos y cambios de color en la misma.

Con la misma regla del codo explicada anteriormente, se ha buscado el número de clusters óptimo para esta imagen. Como se puede observar en la figura 6 el codo se encuentra cuando la cantidad de centroides elegidos es $K = 7$. Por lo tanto, a partir de este valor, el coste disminuye más lentamente y no merece la pena aumentar la cantidad de clusters.

Se ha construido, a continuación, la imagen mostrada anteriormente utilizando únicamente 7 clusters. Como se puede observar en la imagen 7 la pérdida de información en la nueva imagen es mínima y se puede distinguir la imagen con la información que contiene.

Comparando esta tecnica con la ganacia en compresión, se puede observar como realizar agrupamiento en menos colores, también produce una ganancia en espacio, ya que una vez realizado el agrupamiento solo es necesario guardar las etiquetas y la información de los centroides, cuando anteriormente era necesario guardar todas las dimensiones de las muestras.

La primera imagen tiene una dimensión inicial de 18264 píxeles con 3 dimensiones(RGB). Por lo tanto, las dimensiones iniciales son $18264 * 3 = 54792$ componentes a almacenar. Después de realizar el agrupamiento, solo es necesario guardar una etiqueta para cada píxel y la matriz de centroides. En este caso tiene 7 centroides (de tres dimensiones RGB). El ahorro que se obtiene con la primera imagen es $\frac{(18264*3)-(18264+7*3)}{(18264*3)} * 100 = 66,28\,\%$. Es decir, se almacena un $66\,\%$ menos de componentes al realizar el agrupamiento.

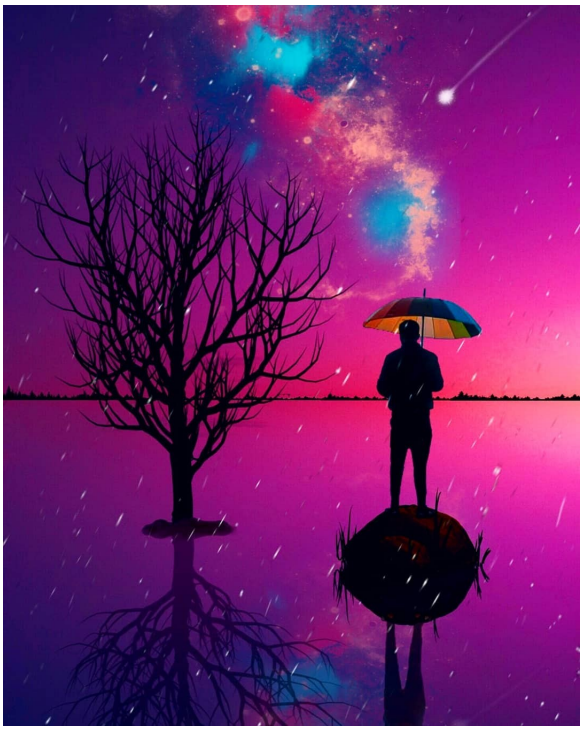

Figura 5: Imagen para realizar pruebas con KMeans.

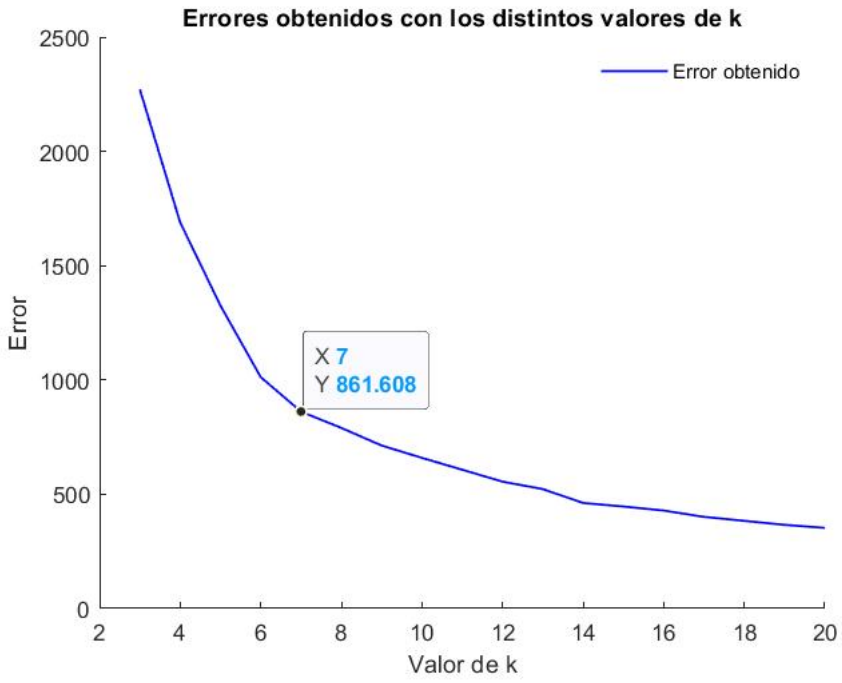

Figura 6: Curva de coste que se obtiene con la nueva imagen.

Figura 7: Imagen para realizar pruebas con KMeans con 7 colores.

Con la segunda imagen ocurre algo similar. Tiene de dimensión inicial 1458000 píxeles con 3 dimensiones (RGB). En total se almacenan $1458000 * 3 = 4374000$ componentes. Después de realizar el agrupamiento, como únicamente se seleccionan 7 clusters, se almacenan $1458000 + 7 * 3 = 1458021$ componentes. El ahorro que se obtiene con la segunda imagen es $\frac{(1458000*3)-(1458000+7*3)}{(18264*3)} * 100 = 66,66\,\%$. Es decir, se almacena un 66 % menos de componentes al realizar el agrupamiento.

Igual que ocurría con las técnicas de compresión, la imagen reconstruida tiene la misma dimensión que la imagen inicial, pero el espacio que ocupa almacenarla es notablemente menor que la imagen inicial.
Cabe destacar, que con esta técnica, a diferencia de lo que ocurría con las técnicas de compresión, el ahorro de espacio puede ser uniforme si se elige correctamente el número de clusters a almacenar.
Por último, mencionar que mediante agrupamiento por colores perdemos información sobre algunas texturas/sombras en la imagen, mientras que con las técnicas de agrupamiento se perdía información de nitidez de la imagen.