Práctica 7: Clustering

Fecha de entrega: 15 de enero de 2015, 16.00h

Material proporcionado:

Fichero	Explicación
`ex7data2.mat`	Conjunto de datos para probar k-means.
`bird_small.png`	Imagen de ejemplo.
`plotDataPoints.m`	Dibuja un paso intermedio de k-means, asignando un color diferente a los puntos de cada cluster.
`plotProgresskMeans.m`	Dibuja el progreso del algoritmo de k-means.
`drawLine.m`	Función auxiliar que dibuja una línea recta entre dos puntos dados.
`runkMeans.m`	Ejecuta el algoritmo de k-means.

1. K-means

En primer lugar, completarás la implementación del algoritmo de clustering k-means y verificarás su correcto funcionamiento sobre un conjunto de datos sencillo. A continuación, lo aplicarás para reducir el tamaño de una imagen disminuyendo el número de colores que utiliza.

1.1. Implementación de k-means

El algoritmo k-means agrupa un conjunto de ejemplos $x^{(1)}, \ldots, x^{(m)}$ (donde $x^{(i)} \in \mathbb{R}^n$) en k clases o *clusters* mediante un proceso que comienza con una inicialización aleatoria del centroide de cada clase, para luego iterativamente asignar a cada ejemplo la clase del centroide más cercano y recomputar los centroides en base a esa asignación. La función `runkMeans` incluida en la práctica implementa este proceso:

```
% Inicialización de los centroides
for iter = 1:iterations
    %A cada ejemplo le asigna la clase del centroide más cercano.
    %idx(i) es el índice del centroide asignado al ejemplo i
    idx = findClosestCentroids(X, centroids);
    %Computa los centroides como la media de los puntos de cada clase
    centroids = computeMeans(X, idx, K);
```

```
end
```

En primer lugar, debes implementar la función `findClosestCentroids` con la siguiente cabecera:

```
function idx = findClosestCentroids(X, centroids)
```

donde la matriz `X` $(m \times n)$ contiene los ejemplos, uno por fila, y la matriz `centroids` $(k \times n)$ contiene las coordenadas de los k centroides. La función ha de devolver en el vector `idx` $(m \times 1)$ el índice (un número en el rango $[1..k]$) del centroide más cercano a cada ejemplo. Es decir, para cada ejemplo i calculamos

$$c^{(i)} := j \quad \text{que minimiza} \quad \|x^{(i)} - \mu_j\|^2$$

donde $c^{(i)}$ es el índice del centroide más próximo a $x^{(i)}$ y μ_j son las coordenadas del centroide j-ésimo. $c^{(i)}$ se corresponde con `idx(i)` en la función `findClosestCentroids`.

A continuación, debes implementar la función `computeCentroids`, que calcula la nueva posición de los centroides, con la siguiente cabecera:

```
function centroids = computeCentroids(X, idx, K)
```

donde la matriz `X` $(m \times n)$ contiene los ejemplos, uno por fila, y el vector `idx` $(m \times 1)$ contiene el índice (un número en el rango $[1..k]$) del centroide más cercano a cada ejemplo. La función ha de devolver la matriz `centroids` $(k \times n)$ con las coordenadas de los k centroides calculados de la siguiente forma:

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

donde C_k es el conjunto de ejemplos asignados al centroide k-ésimo.

Para verificar que has implementado correctamente las funciones, puedes cargar los datos del fichero `ex7data2.mat` que define una matriz `X` e invocar a la función `runkMeans`

```
function [centroids, idx] = runkMeans(X, initial_centroids, max_iters, ...
                                      plot_progress)
```

con dicha matriz `X`, la posición inicial de los centroides `[3 3; 6 2; 8 5]`, 10 iteraciones y `true` en el parámetro `plot_progress` para que se visualice una gráfica como la que se muestra en la Figura 1.

Para completar la implementación del algoritmo k-means debes implementar la inicialización de los centroides con K elementos de X elegidos aleatoriamente, que se pueden obtener como:

```
randidx = randperm(size(X, 1));
centroides = X(randidx(1:K), :);
```

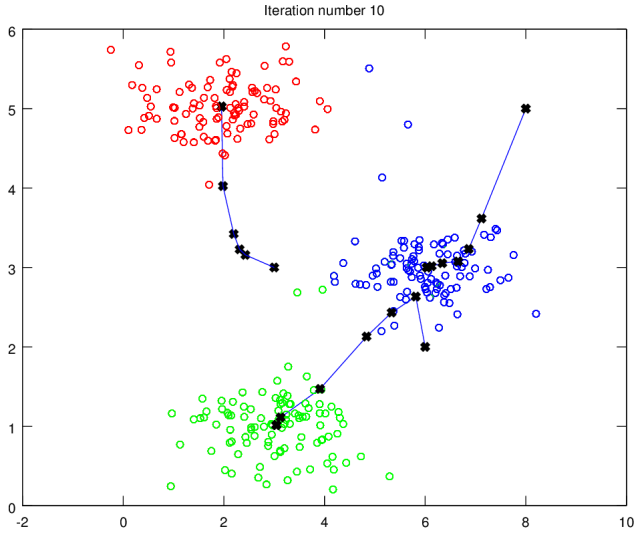

Figura 1: Resultado de aplicar k-means sobre los datos del fichero `ex7data2.mat`

1.2. Compresión de imágenes

A continuación aplicarás el algoritmo k-means para comprimir una imagen en color por el procedimiento de reducir el número de colores que utiliza.

Para leer la imagen puedes utilizar la función `imread` que devuelve una matriz de dimensión $filas \times columnas \times 3$ donde las posiciones $(i, j, 1), (i, j, 2)$ e $(i, j, 3)$ representan respectivamente los porcentajes de rojo, verde y azul (RGB) del pixel que está en la posición (i, j) expresados como un número entero entre 0 y 255. Para visualizar la imagen puedes utilizar la función `imagesc` que permite dibujar la imagen leída por `imread`, aunque los porcentajes de color se han de representar como números reales entre 0 y 1. De esta forma, el siguiente fragmento de código mostrará la imagen de la parte izquierda de la Figura 2:

```
A = double(imread('bird_small.png'));
A = A / 255;
imagesc(A);
```

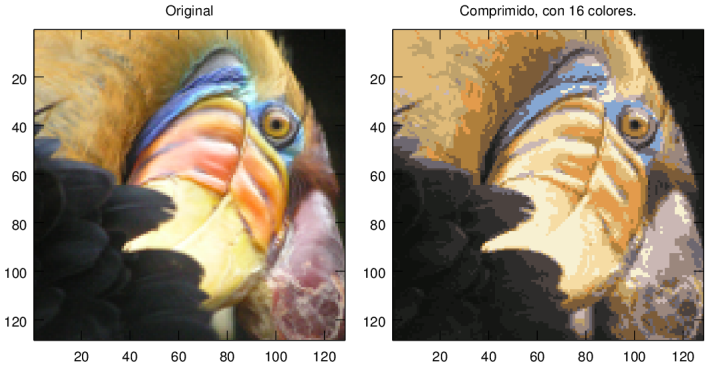

Figura 2: Imagen `bird_small.png` sin comprimir e imagen comprimida a 16 colores

El procedimiento de compresión consiste en elegir los k colores más representativos de la imagen y sustituir el color de cada punto por el más parecido entre esos k. De esta forma, si transformamos la imagen a 16 colores, en lugar de utilizar 24 bits para representar el color de cada punto (3 enteros de 8 bits cada uno), bastará con almacenar los 24 bits del color RGB de cada uno de los 16 colores, y, para cada punto, utilizar 4 bits para representar a cuál de los 16 colores corresponde.

Para elegir los k colores más representativos de la imagen, se puede utilizar el algoritmo k-means sobre los valores RGB de los pixeles de la imagen y encontrar los k centroides para ese conjunto de valores. De esta forma, los centroides darán los valores RGB de los k colores a utilizar en la representación comprimida, y cada color será sustituido por el centroide del cluster al que pertenece.

Para poder aplicar el algoritmo k-means que has implementado en el apartado anterior, primero tienes que transformar la matriz `A` de tres dimensiones, $filas \times columnas \times 3$, en otra de dos dimensiones, $filas * columnas \times 3$, que almacene en cada fila la representación RGB del color de un punto de la imagen. Para ello puedes utilizar la función `reshape` de Octave.

En la parte derecha Figura 2 se muestra el resultado de comprimir la imagen original a 16 colores.

2. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicadas en la cabecera de la práctica.

Se entregará un único fichero en formato pdf que contenga la memoria de la práctica, incluyendo el código desarrollado y los comentarios y gráficas que se estimen más adecuados para explicar los resultados obtenidos.