Proyecto 9

K-means

Descripción

Implementar el algoritmo K-means para hacer clustering y aplicarlo a la compresión de una imagen.

Datos

ex7data2.mat. Contiene un conjunto de puntos para ir probando el código.
bird_small.png. Contiene una imagen de 128 X 128 pixeles en RGB.

Fuente: A. Ng. Machine Learning. Curso de Coursera-Stanford (2011).

Funciones

El algoritmo K-means se puede describir como sigue:

```
% Inicializar centroides
centroids = kMeansInitCentroids(X, K);
for iter in range (iterations):
        % Paso de asignación de cluster: Asignar cada dato al centroide más cercano.
        % idx(i) es lo mismo que c(i), el índice del centroide asignado al ejemplo i
        idx = findClosestCentroids(X, centroids);
        % Paso de mover el centroide: Calcular la media basado en la asignación de centroides
        centroids = computeMeans(X, idx, K);
end
```

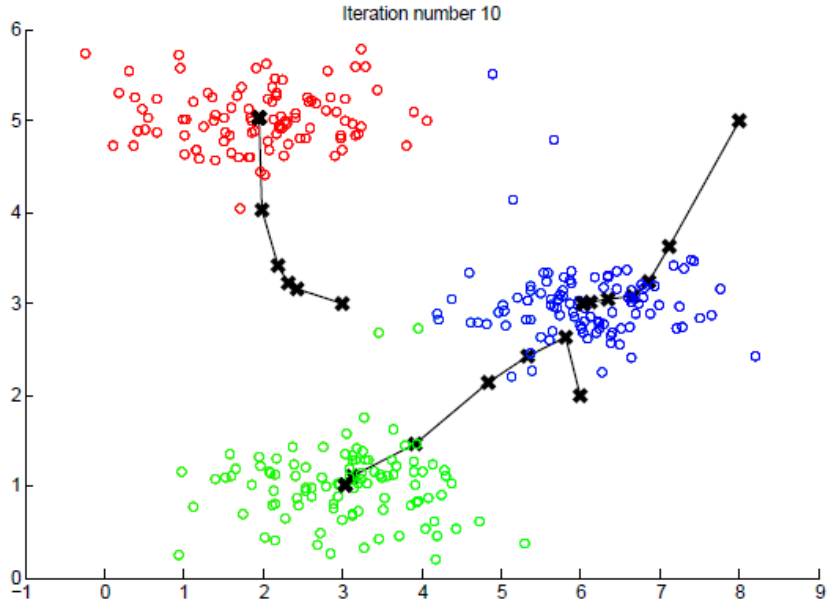
Se deben implementar las siguientes funciones:

1. **findClosestCentroids(X, initial_centroids)**: Recibe un conjunto de ejemplos **X** y los centroides iniciales **initial_centroids**. La función regresa un vector idx que contiene los índices de los clusters más cercanos a cada ejemplo. En la teoría, idx(i) es llamado $c^{(i)}$.
   Si se prueba con el conjunto de entrenamiento del archivo **ex7data2.mat**, con 3 clusters y con los centroides iniciales (3, 3), (6, 2), (8, 5), los centroides más cercanos a los primeros tres ejemplos deben ser 1, 3 y 2, respectivamente, es decir, idx(1)=1, idx(2)=3 y idx(3)=2.

2. **computeCentroids(X, idx, K)**: Recibe el conjunto de ejemplos X, los índices a los clusters más cercanos idx y el número de clusters K. La función calcula las madias basadas en los centroides más cercanos encontrados con la función del punto 1. La función regresa un arreglo que contiene las coordenadas de los nuevos centroides después de haber calculado la media de los puntos más cercanos a los mismos.

Si se corre esta función inmediatamente después de la del punto 1, el resultado con las coordenadas para los nuevos centroides deben ser: (2.428301, 3.157924), (5.813503, 2.633656), (7.119387, 3.616684).

3. **runkMeans(X, initial_centroids, max_iters, true)**: Recibe el conjunto de ejemplos **X**, los centroides iniciales **initial_centroids**, el máximo número de iteraciones en el proceso **max_iter** y un valor booleano (OPCIONAL) que si es TRUE hace que se vayan dibujando las posiciones de los centroides en cada iteración. Obviamente, esta función llama a las de los puntos 1 y 2, el número de iteraciones que se indique.
   La gráfica después de 10 iteraciones, con K=3 y los mismos centroides iniciales del punto 1 debe verse como:



4. **kMeansInitCentroids(X, K):** Recibe el conjunto de ejemplos X y el número de clusters K. Regresa K centroides seleccionados en forma aleatoria del conjunto de datos. Una buena forma de implementar este proceso es permutar aleatoriamente (reordenar en forma aleatoria) sus índices y seleccionar los K primeros:

Aplicar K-means a la compresión de imágenes (Opcional)

Una vez que se tienen todas las funciones se puede utilizar K-means para comprimir una imagen.
En una representación de 24-bits para imágenes de colores, cada pixel es representado con 3 números enteros (en un rango de 0 a 255) los cuales especifican los valores de la intensidad de Rojo, Verde y Azul de ese pixel. Esta forma de codificar se conoce como formato RGB, por sus siglas en inglés. Una imagen así contiene miles de colores y, en este ejercicio se dibujará la imagen con sólo 16 colores (los cuales se pueden guardar en sólo 4 bits por pixel).

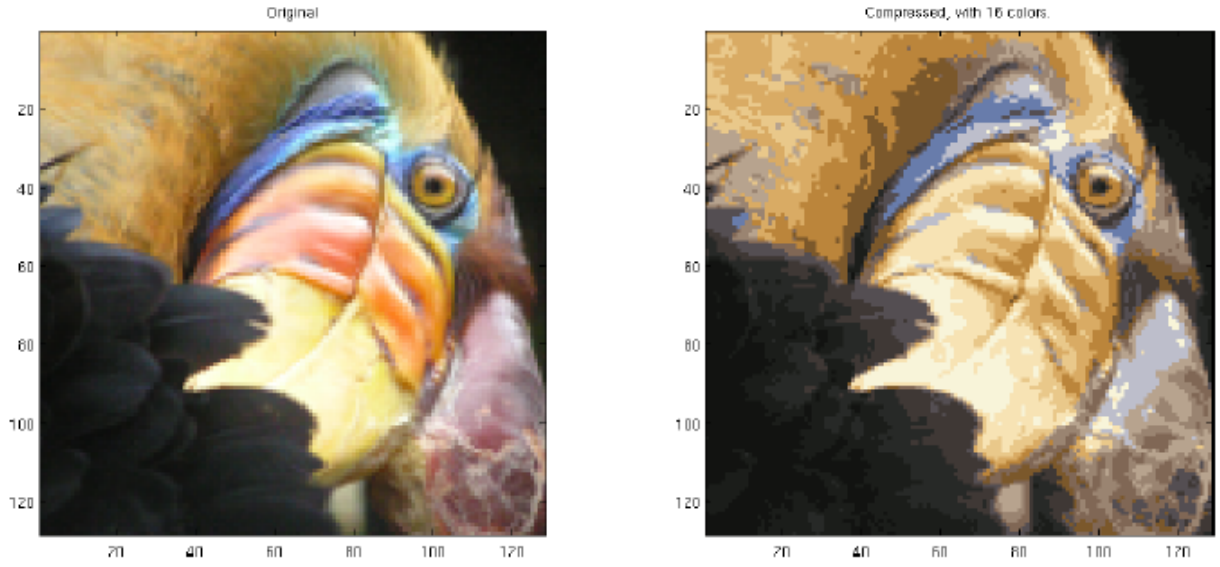
En este ejercicio, se utilizará K-means para seleccionar los 16 colores que mejor aproximan a la imagen, los cuales serán usados para crear la imagen comprimida. Cada pixel será tratado como un ejemplo y se usa K-means para encontrar los 16 colores que mejor agrupan los pixeles en el espacio tridimensional RGB. Una vez encontrados estos colores, los cuales corresponden a los centroides, se reemplazará cada pixel de la imagen original con el color de su centroide.

La imagen usada será **bird_small.png**. Buscar una función en Python que cargue la imagen a una matriz A de tres dimensiones.

La matriz A contendrá intensidades en Rojo, Verde o Azul de cada pixel. Los índices de la matriz dan información de a qué pixel y a qué color corresponde el dato. Los dos primeros índices se refieren a la posición del pixel en la imagen y el tercero dice el color al que se refiere la intensidad. Por ejemplo, A(40,14,3) contiene la intensidad de azul (3) del pixel colocado en (40, 14).

Si la instrucción encontrada carga la imagen en un vector, se tendrá que crear la matriz (hacer un **reshape**) de *m* X 3 (donde se tienen *m*= 16384=128X128 filas, correspondiendo a cada pixel y 3 columnas, correspondiendo a las intensidades del rojo, verde y azul).

La imagen original requiere 24 bits por cada uno de los 128 X 128 pixeles (394,216 bits) mientras que la imagen comprimida requerirá sólo 4 bits por cada pixel (65,920 bits). Las imágenes original y comprimida se deben ver así:



Imprima la imagen real y la comprimida para ver la diferencia.

Entrega y Revisión

- Salve su archivo con el nombre **Proyecto9.py**. Tiene que ser con ese nombre porque es el que se va a usar para la revisión automática.

- Suba sus archivos dos archivos, el del proyecto (.py) y el de la documentación (.docx o .pdf) **por separado**, NO los ponga en ZIP.
- Para la revisión se harán pruebas sobre las funciones solicitadas. Para esto, tanto el archivo que contiene las funciones (el suyo) como el de prueba (el mío), se colocarán en el mismo folder. El programa de prueba al inicio hará un:

```
from file import function
```

y luego se utilizará en el script de prueba llamándola como como:

```
function(<parámetros>)
```