Lab. *K-means*

Neste exercício, você usará o algoritmo ***Kmeans***.

Exercício 1

a) Executar o algoritmo *K-means* para obter 3 clusters das amostras presentes no arquivo "data2.mat", conforme Figura 1.

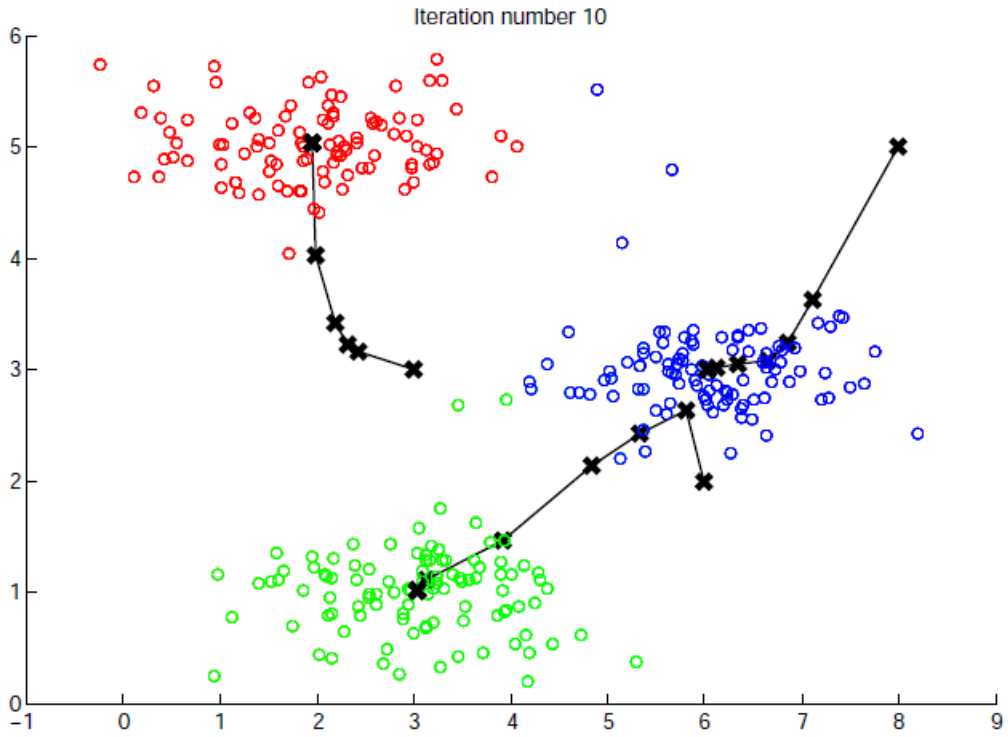

Figure 1: The expected output.

b) Agora ajuste o algoritmo *K-Means* para diferentes números de clusters (1 a 20).

c) Para cada modelo, armazene o número de clusters e o valor da inércia (custo).

d) Plote a *Inércia (custo) versus Nº de clusters*. Parece haver um número ideal de cluster (método do cotovelo - *elbow*?

e) Ajuste um modelo de agrupamento aglomerativo hierárquico (*Hierarchical Agglomerative Clustering*) com três clusters (escolha e teste diferentes tipos de *linkage*).

f) Compare os resultados com os obtidos pelo *k-means*.

g) Visualize o dendrograma produzido pelo método de agrupamento aglomerativo *Dica:* O *SciPy* tem um módulo chamado cluster.hierarchy que contém as funções de *linkage* e dendrogram necessárias para criar o mapa de ligação (*linkage*) e traçar o dendrograma resultante.

Exercício 2

Neste exercício, você aplicará *k-means* para comprimir uma imagem. Em uma representação direta de cores com 24 bits de uma imagem (Figura 2, bird_small.png.mat e bird_small.png), cada pixel é representado por três valores inteiros sem sinal de 8 bits (variando de 0 a 255) que especificam os valores de intensidade vermelha, verde e azul. Essa codificação é frequentemente referenciada como codificação RGB. Nossa imagem contém milhares de cores, e nesta parte do exercício, você reduzirá o número de cores para 16 cores.

Ao fazer essa redução, é possível representar (comprimir) a foto de forma eficiente. Especificamente, você só precisa armazenar os valores RGB das 16 cores selecionadas, e para cada pixel a imagem você precisará apenas armazenar o índice da cor daquele local (onde apenas 4 bits são necessários para representar as 16 possibilidades).

Neste exercício, você usará o algoritmo *k-means* para selecionar as 16 cores que serão usadas para representar a imagem compactada. Concretamente, você tratará cada pixel da imagem original como um amostra (instância) de dados e usará o algoritmo *k-means* para encontrar as 16 cores que melhor agrupam (clusterizam) os pixels no espaço RGB. Uma vez que você tenha computado os centroides dos clusters da imagem, você irá usar as 16 cores para substituir os pixels da imagem original.

Figure 2: The original 128x128 image.

Depois de encontrar K = 16 cores para representar a imagem, você pode atribuir cada posição de pixel ao seu centroide mais próximo. Observe que você reduziu significativamente o número de bits necessários para descrever a imagem. A imagem original exigia 24 bits para cada um dos 128×128 pixels, resultando em tamanho total de 128×128×24 = 393.216 bits. A nova representação requer um armazenamento na forma de um dicionário de 16 cores, sendo que cada uma delas requer 24 bits. No entanto, como a imagem em si requer apenas 4 bits por pixel, o número final de bits utilizados será, portanto, 16 × 24 + 128 × 128 × 4 = 65.920 bits, o que corresponde à compressão da imagem original por cerca de um fator igual a 6.

Finalmente, você poderá ver os efeitos da compressão reconstruindo a imagem baseada apenas nas atribuições dos centroides. A Figura 3 mostra a reconstrução que pode ser obtida.

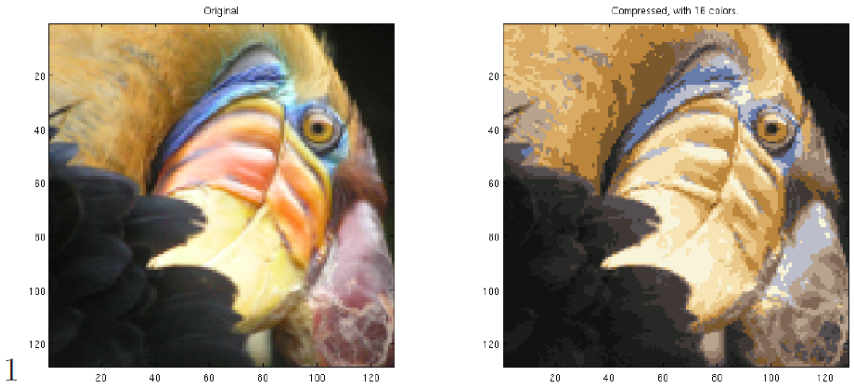

Figure 3: Original and reconstructed image (when using K-means to compress the image).