Exercice de programmation 3 :
K-means Clustering

Dans cet exercice, vous allez implémenter l'algorithme de clustering K-means et l'appliquer pour compresser une image. Dans la deuxième partie, vous utiliserez l'analyse en composantes principales pour trouver une représentation de faible dimension des images de visage.

1 K-means Clustering

Dans cet exercice, vous allez mettre en œuvre l'algorithme des K-means et l'utiliser pour la compression d'images. Vous commencerez par un exemple d'ensemble de données 2D qui vous permettra d'avoir une idée du fonctionnement de l'algorithme des K-means. Ensuite, vous utiliserez l'algorithme K-means pour la compression d'images en réduisant le nombre de couleurs qui apparaissent dans une image à celles qui sont les plus communes dans cette image.

Implementing K-means
L'algorithme K-means est une méthode permettant de regrouper automatiquement des exemples de données similaires. Concrètement, on vous donne un ensemble d'apprentissage {x(1), ..., x(m)} (où x (i) ∈ Rn), et vous voulez regrouper les données en quelques "clusters" cohésifs.
L'intuition derrière K-means est une procédure itérative qui commence par deviner les centroïdes initiaux, puis raffine cette estimation en assignant de manière répétée des exemples à leurs centroïdes les plus proches, puis en recalculant les centroïdes en fonction des assignations.

The way kmeans algorithm works is as follows:

1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn’t changing.

- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).
- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

La boucle interne de l'algorithme exécute deux étapes de manière répétée : (i) l'affectation de chaque exemple d'apprentissage x(i) à son centroïde le plus proche, et (ii) le recalcul de la moyenne de chaque centroïde à l'aide des points qui lui sont affectés. L'algorithme K-means convergera toujours vers un ensemble final de moyennes pour les centroïdes. Notez que la solution convergée n'est pas toujours idéale et dépend du réglage initial des centroïdes. Par conséquent, dans la pratique, l'algorithme K-means est généralement exécuté plusieurs fois avec différentes initialisations aléatoires.

Vous implémenterez les deux phases de l'algorithme K-means séparément dans les sections suivantes.

Trouver les centroïdes les plus proches

Dans la phase "d'affectation des clusters" de l'algorithme K-means, l'algorithme affecte chaque exemple de formation x (i) à son centroïde le plus proche, étant donné les positions actuelles des centroïdes. Plus précisément, pour chaque exemple i, nous définissons

$$c^{(i)} := j \quad \text{that minimizes} \quad ||x^{(i)} - \mu_j||^2,$$

où c(i) est l'indice du centroïde qui est le plus proche de x (i), et μj est la position (valeur) du jeme centroïde.
Votre tâche consiste à créer une fonction "findClosestCentroids". Cette fonction prend la matrice de données X et les emplacements de tous les centroïdes à l'intérieur des centroïdes et doit produire un tableau unidimensionnel idx qui contient l'indice (une valeur dans {1, ..., K}, où K est le nombre total de centroïdes) du centroïde le plus proche de chaque exemple d'apprentissage. Vous pouvez implémenter ceci en utilisant une boucle sur chaque exemple d'apprentissage et chaque centroïde.

Calcul des moyennes centroïdes

Étant donné les affectations de chaque point à un centroïde, la deuxième phase de l'algorithme recalcule, pour chaque centroïde, la moyenne des points qui lui ont été affectés. Plus précisément, pour chaque centroïde k, on définit

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

où Ck est l'ensemble des exemples qui sont affectés au centroïde k. Concrètement, si deux exemples, disons x(3) et x(5), sont affectés au centroïde k = 2, vous devez mettre à jour μ2 =12(x(3) + x(5)).

K-means on example dataset

Après avoir exécuté les deux fonctions (findClosestCentroids et computeCentroids), l'étape suivante consiste à exécuter l'algorithme K-means sur un ensemble de données 2D pour vous aider à comprendre le fonctionnement de K-means.

K = 3; => 3 Centroids

centroids initial = [3 3; 6 2; 8 5]

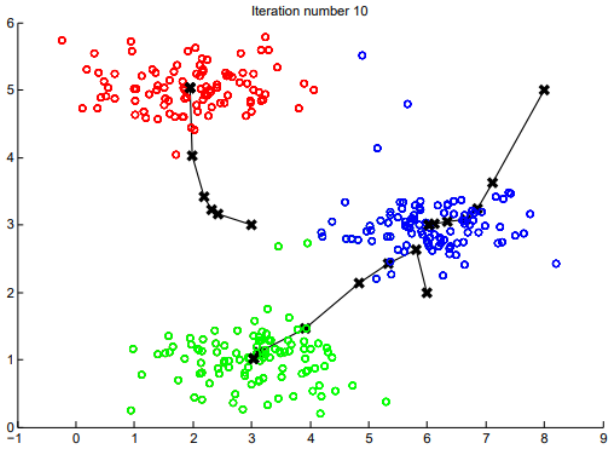

Figure 1: The expected output.

2 Image compression with K-means

Figure 2: The original 128x128 image.

Dans cet exercice, vous allez appliquer les K-means à la compression d'images. Dans une représentation directe d'une image en couleur sur 24 bits, chaque pixel est représenté par trois entiers non signés de 8 bits (allant de 0 à 255) qui spécifient les valeurs d'intensité rouge, verte et bleue. Ce codage est souvent appelé le codage RVB. Notre image contient des milliers de couleurs, et dans cette partie de l'exercice, vous allez réduire le nombre de couleurs à 16 couleurs.
En effectuant cette réduction, il est possible de représenter (compresser) la photo d'une manière efficace. Plus précisément, vous n'avez besoin de stocker que les valeurs RVB des 16 couleurs sélectionnées, et pour chaque pixel de l'image, vous n'avez plus qu'à stocker l'index de la couleur à cet endroit (où seulement 4 bits sont nécessaires pour représenter 16 possibilités).
Dans cet exercice, vous allez utiliser l'algorithme K-means pour sélectionner les 16 couleurs qui seront utilisées pour représenter l'image compressée. Concrètement, vous allez traiter chaque pixel de l'image originale comme un exemple de données et utiliser l'algorithme des moyennes K pour trouver les 16 couleurs qui regroupent le mieux (cluster) les pixels dans l'espace RVB à 3 dimensions. Une fois que vous aurez calculé les centroïdes des groupes sur l'image, vous utiliserez les 16 couleurs pour remplacer les pixels de l'image originale.

assigner chaque position de pixel à son centroïde le plus proche en utilisant la fonction findClosestCentroids. Cela vous permet de représenter l'image originale à l'aide des affectations des centroïdes de chaque pixel. Remarquez que vous avez considérablement réduit le nombre de bits nécessaires pour décrire l'image. L'image d'origine nécessitait 24 bits pour chacun des 128×128 emplacements de pixels, soit une taille totale de 128 × 128 × 24 = 393 216 bits. La nouvelle représentation nécessite un stockage supplémentaire sous la forme d'un dictionnaire de 16 couleurs, chacune nécessitant 24 bits, mais l'image elle-même ne nécessite alors que 4 bits par emplacement de pixel. Le nombre final de bits utilisés est donc de 16 × 24 + 128 × 128 × 4 = 65 920 bits, ce qui correspond à une compression de l'image originale d'un facteur 6 environ.

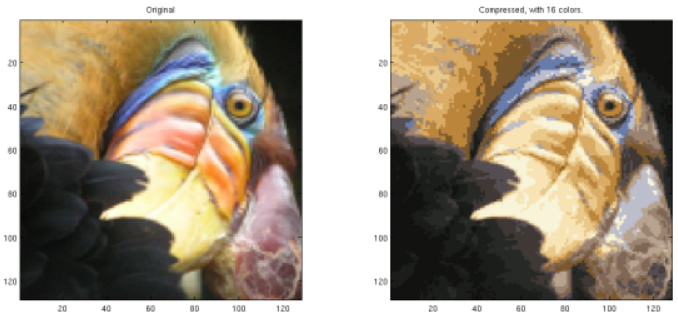

Figure 3: Original and reconstructed image (when using K-means to compress the image).

Enfin, vous pouvez visualiser les effets de la compression en reconstruisant l'image en vous basant uniquement sur les assignations des centroïdes. Plus précisément, vous pouvez remplacer l'emplacement de chaque pixel par la moyenne du centroïde qui lui a été attribué. La figure 3 montre la reconstruction que nous avons obtenue.