K-moyennes

5 mars 2020

1 Les K-moyennes

L'algorithme des *K-moyennes*, également appelé algorithme des *nuées dynamiques*, (en anglais *k-means clustering*) est un algorithme couramment utilisé en analyse de données. Il permet de partitionner une collection d'objets en K clusters, K étant un nombre fixé par l'utilisateur. On supposera dans la suite que nos objets $o_1, \ldots, o_N$ peuvent être représentés sous forme de vecteurs. L'algorithme des K-moyennes se déroule de la façon suivante :

1. Choisir K objets au hasard parmi les objets de la collection. Soient $(R_1, \cdots, R_K)$ les objets ainsi obtenus. $(R_1, \cdots, R_K)$ sont les représentants initiaux des K clusters $(C_1, \cdots, C_K)$ qui sont pour l'instant vides.
2. Affecter chaque objet de la collection au cluster dont le représentant est le plus proche, o btenu par : $\underset{1 \leq k \leq K}{argmin}\{d(o_j, R_k)\}$ où d est une distance ou une similarité entre objets.
3. Calculer de nouveaux représentants pour les clusters. Ces nouveaux représentants correspondent à la moyenne des objets du cluster :

$$\forall k \in \{1, \ldots, K\},\ R_k = \frac{1}{|C_k|} \sum_{o_j \in C_k} o_j$$

4. Retourner en 2 tant que la différence $\Delta(R)$ entre les anciens et les nouveaux représentants est supérieure à un seuil ϵ fixé (et arbitrairement petit).

La complexité de l'algorithme des K-moyennes est de l'ordre de $O(KNIs)$, où I est le nombre d'itérations de l'algorithme et s la complexité du calcul de la distance/similarité. La partition obtenue par l'algorithme des K-moyennes dépend des représentants initialement choisis (essayez de vous en convaincre sur un exemple simple). De façon à s'affranchir en partie de cette dépendance, on peut par exemple exécuter l'algorithme des K-moyennes (K et d étant fixés) avec des initialisations différentes, et on retient la meilleure partition. La qualité d'une partition est mesurée par la quantité :

$$D = \sum_{k=1}^{K} \sum_{o_j \in C_k} d(o_j, R_k)$$

qui mesure la cohésion des clusters obtenues.

2 Application aux images

Le but du TP est de restreindre à K le nombre de couleurs d'une image. Les K couleurs seront déterminées grâce à l'algorithme des K-moyennes. Voici quelques exemples d'application sur une image pour différentes valeurs de K.

original

$K = 2$

$K = 5$

$K = 10$

Exercice 1. Gestion d'image

a. Écrire un programme qui ouvre un fichier image dont le nom est donné en ligne de commande, et l'enregistre sous un autre nom.
Note : utilisez les classes BufferedImage [1] du package java.awt.image et ImageIO [2] du package javax.imageio. Par exemple, une image "mon_image.png" peut être lue grâce à ces classes de la façon suivante :

```
BufferedImage img = ImageIO.read(new File("mon_image.png"));
```

Référez vous à la *documentation en ligne* de ces classes pour déterminer comment sauvegarder une image. Pensez à gérer les diverses exceptions nécessaires à la gestion de fichiers et de la ligne de commande.

b. Sachant que l'on souhaite utiliser l'algorithme des K-moyennes pour restreindre le nombre de couleurs de l'image, quel type de vecteur allez-vous utiliser ?

c. Écrivez une fonction qui prend en paramètre une image et renvoie la liste de ses vecteurs (tels que définis dans la question précédente).

Exercice 2. Algorithme des K-moyennes

Écrivez une classe $KMoyennes$ qui implémente l'algorithme des K-moyennes sur une image comme décrit précédemment. Plusieurs paramètres sont importants dans le cadre des K-moyennes : le nombre K de clusters, l'initialisation (aléatoire) des représentants des clusters et la fonction de distance/similarité choisie. On vous demande de faire varier ces paramètres et de voir comment varient les résultats.
Remarque : La génération de nombres aléatoires peut se faire grâce à la classe Random [3] du package java.util.

Keep Calm
and
GIT GUD

1. https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html
2. https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html
3. https://docs.oracle.com/javase/7/docs/api/java/util/Random.html