

# Programmation séquentielle - K-Means

## 1 But

- Le but de ce travail pratique est d'implémenter la méthode des k-moyennes pour partitionner des données de façon non supervisée.

## 2 Théorie

### 2.1 Le partitionnement de données

De façon générale, le partitionnement de données (*clustering*) consiste à grouper des données en *clusters* de façon à ce que les éléments d'un même groupe soient proches les uns des autres ou d'un élément virtuel représentant l'élément "moyen" de la partition.

### 2.2 Méthodes des k-moyennes

La méthode k-moyennes (k-means en anglais) est un algorithme qui permet de partitionner un groupe de données par rapport à une métrique définie. Chaque *cluster* devra contenir des données homogènes. De plus tous les *cluster* doivent également être hétérogènes les uns des autres.

Ce processus de regroupement de données s'effectue de manière non supervisée, c'est-à-dire que l'algorithme utilisé s'applique sur des données non étiquetées et sans avoir recours à un retour externe sur ses résultats.

### 2.3 Initialisation des clusters

Avant de pouvoir appliquer l'algorithme des k-moyennes, il faut déterminer le nombre de *clusters* souhaité ( $k$ ) ainsi que la position de leur *centroïde* (point représentant le centre de chaque *cluster*).

Dans un premier temps la valeur de  $k$  peut être choisie arbitrairement, mais il faudra par la suite déterminer la valeur la plus optimale (nous y reviendrons plus tard).

La position des *centroïdes* sera déterminée aléatoirement mais il faut s'assurer qu'ils ne soient pas trop proches les uns des autres afin d'avoir des *clusters*

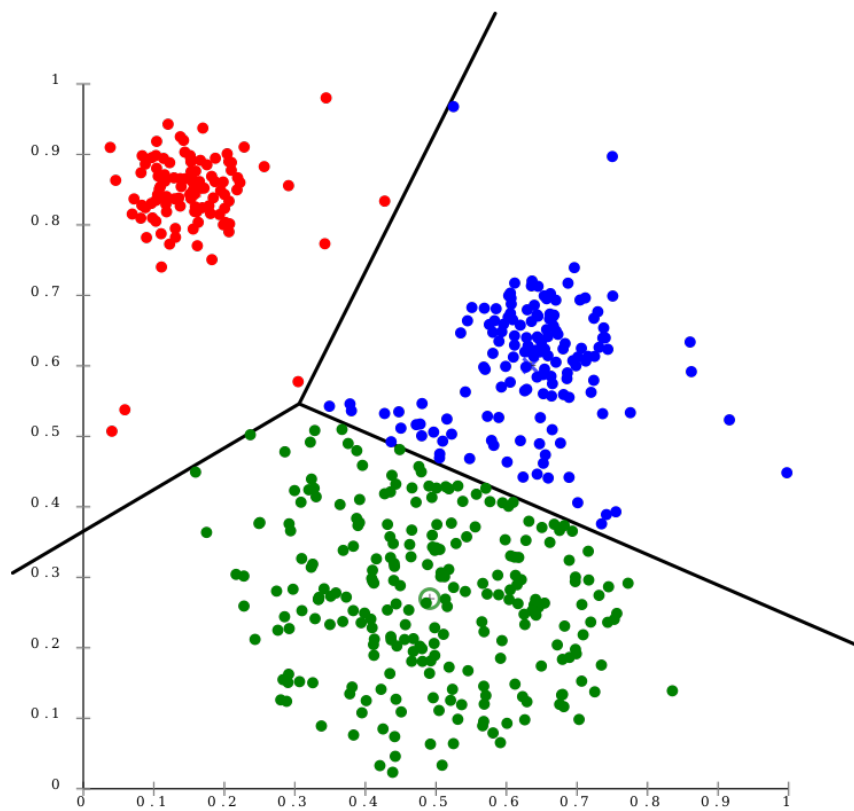


Figure 1: kmeans clustering

cohérents. Vous devrez déterminer une distance  $\delta$  minimale séparant chaque *centroïde* entre eux.

## 2.4 Calcul de la distance

Vous travaillerez avec des points à 2 dimensions contenant des coordonnées sous la forme de nombres à virgule flottante. Ainsi vous devrez les assigner à des *clusters* par rapport à la distance séparent les points. En effet, chaque point devra appartenir au *cluster* dont son *centroïde* est le plus proche.

Afin de quantifier la proximité de deux éléments entre eux, nous avons besoin de définir une fonction distance, qui sera notre métrique.

Cette fonction distance peut retourner simplement une distance de Manhattan ou une distance euclidienne et prend en argument deux vecteurs éléments de notre univers.

$$d(x, y)$$

Une grande valeur de retour indiquera que  $x$  et  $y$  sont éloignés. Inversement, une petite valeur de retour indiquera qu'ils sont proches.

Ainsi, l'assignation de chaque point à un *cluster* se déroule de la manière suivante :

1. Calculer la distance entre le point et les *centroïdes*.
2. Comparer les distances pour déterminer le *centroïde* le plus proche.
3. Assigner le point au *cluster* du *centroïde* le plus proche .

### 2.4.1 Distance euclidienne

La distance euclidienne ( $D_e$ ) entre un point  $x$  et  $y$  correspond à :

$$\vec{D}_e = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{(x - y)^2}$$

### 2.4.2 Distance de Manhattan

La distance de Manhattan ( $D_m$ ) entre un point  $x$  et  $y$  correspond à :

$$\vec{D}_m = |(x_1 - y_1)| + |(x_2 - y_2)|$$

## 2.5 Déterminer le $k$ optimal

Afin de tirer les meilleures conclusions du résultat de notre algorithme des  $k$ -moyennes, il faut déterminer le  $k$  (nombre de *cluster*) optimal.

Pour cela il existe de nombreuses méthodes, les plus connues étant : - La méthode du coude (Elbow Method) - L'analyse par silhouette (Silhouette analysis)

## 3 Énoncé

À partir d'un fichier contenant des points à 2 dimensions (chaque coordonnées étant un nombre à virgule flottante), vous devez implémenter l'algorithme k-moyennes et permettre la visualisation des différents clusters créés.

Cet algorithme peut s'appliquer facilement à tous types de données pouvant être représentées sous forme de vecteurs de valeurs. Cependant, nous avons décidé de limiter ce travail aux vecteurs à valeurs décimales, en 2 dimensions, pour en simplifier la visualisation et la vérification de notre algorithme.

### 3.1 Interface utilisateur

Le programme fonctionnera entièrement en ligne de commande, avec possibilité de spécifier un fichier d'entrée. À cette fin nous utiliserons des appels à `fscanf(...)` et la lecture des arguments en ligne de commande. Par défaut, le programme lira les données dans la console / sur l'entrée standard (permet les "pipe").

### 3.2 Déroulement de l'algorithme

Une fois nos données correctement chargées, il faudra appliquer l'algorithme des k-moyennes à ces dernières :

0. Déterminer le  $k$ .
1. Placer  $k$  *centroïdes* aléatoirement, appartenant chacun à un *cluster*.
2. Déterminer les points appartenant à chaque *cluster* selon leur distance avec les *centroïdes*.
3. Une fois tous les points (ré)assignés à un *cluster*, calculer la nouvelle position des *centroïdes* (il s'agira du nouveau centre du *cluster* suite à la modification des points).
4. Recommencer à partir de l'étape 2 si la position d'au moins 1 *centroïde* a été modifiée.
5. Afficher les points et *clusters* à l'écran (avec des couleurs !)

Il faudra utiliser la librairie `gfx` pour afficher vos résultats.

### 3.3 Fonctions à implémenter

Votre programme doit implémenter les fonctions suivantes : - Définir  $k$  centroïde aléatoirement - Calculer la distance entre le centre d'un centroïde et d'un point - Calculer la nouvelle position d'un centroïde (à partir de la moyenne des points) - Lire un fichier contenant des points - Déterminer les points appartenant aux *clusters* - Déterminer le  $K$  optimal (Elbow optimal, silhouette method, ...)

Ainsi que les structures ci-dessous :

```
typedef struct _cluster {  
    struct _point* centroid;
```

```

    int color;
} cluster;

typedef struct _point {
    float x;
    float y;
    char* label; // for tests
    struct _cluster* cluster;
    int color;
} point;

typedef struct _kmeans {
    int k;
    struct _cluster* clusters_array; // k size
    struct _point** points_array;
} kmeans;

```

### 3.4 Vérification du travail réalisé

Afin de s'assurer que votre algorithme a été correctement implémenté vous devez mettre en place des fichiers de tests.

Étant donné que nous utilisons un algorithme non supervisé, il est difficile de vérifier si le résultat obtenu est “correct”. Néanmoins, si nous appliquons notre algorithme sur des données dont on connaît le résultat attendu, alors il est possible de s'assurer que notre implémentation est correcte.

Le fichier *data\_for\_tests.txt* contient une dizaine de points directement assignés à un *cluster*. Vous devez vous assurer que votre implémentation des k-moyennes vous donne un résultat identique aux données de test.

## 4 Travail à rendre

- Le repos git contenant le code réalisé.
- Vous devrez également présenter votre travail à l'aide d'un support (Pow-erpoint par exemple).

## 5 Travail supplémentaire possible

### 5.1 Affichage en temps réel

Au lieu d'afficher directement le résultat final de votre implémentation de l'algorithme des k-moyennes, vous devez afficher chaque étape effectuée afin que l'on puisse constater comment l'assignation des points aux *clusters* est effectuée.

## 5.2 Implémentation avec vecteurs à $n$ dimensions

Étendre l'implémentation pour pouvoir utiliser des vecteurs à  $n$  dimensions avec  $n$  passé en paramètre au début de la procédure.

Afin de pouvoir visualiser les résultats, vous devrez également implémenter l'algorithme PCA pour la visualisation de vecteurs à plus de 2 dimensions.