TP n°4.5 - Algorithme des k moyennes

Notions abordées

- Algorithme des k moyennes
- Struct en C

Exercice 1 : Algorithme des k-moyennes en C

Dans cet exercice on se propose d'implémenter l'algorithme des k-moyennes en C. L'algorithme prend en arguments un entier k et un ensemble de vecteurs $\mathcal{V}$ et essaie de regrouper les vecteurs en k groupes de sorte à minimiser la somme des carré des distances des vecteurs au vecteur moyen de leur groupe. On se donne donc (dans `compagnon.c`, que vous trouverez sur cahier de prépa), un type de vecteurs :

```c
struct vector_s {
  int     taille ; /* taille du tableau content */
  float* content ; /* tableau d'éléments        */
};
typedef struct vector_s *vector;
```

Q. 1 Définir les fonctions de création et de libération d'éléments de type `vector` :

- `vector vector_create(int n)`
- `void vector_free(vector v)`

Dans le reste de cet exercice, les groupes de vecteurs du jeu de données seront représentés par un entier de l'intervalle $[\![0, k[\![$ puisqu'il y a k groupes à former. Ainsi un partitionnement du jeu de données est la donnée d'un numéro dans $[\![0, k[\![$ pour chaque vecteur du jeu de données, ce qui est facilement représentable par un tableau. Ainsi l'algorithme des k-moyennes manipule 3 grandeurs :

- un tableau `data` de taille n qui n'est jamais modifié : c'est le jeu de données, ici un tableau de vecteurs ;
- un tableau `clustering` de taille n indiquant dans sa case i l'indice du groupe auquel appartient la donnée `data[`i`]` ;
- un tableau `centers` de taille k contenant les centres des groupes courants.

Q. 2 Définir une fonction `int compute_closest_point(int k, vector* centers, vector input)` prenant en arguments : un entier `k` qui est la taille du tableau de vecteurs `centers`, et un vecteur `input` et calculant l'indice dans `centers` du vecteur de `centers` minimisant la distance à `input`.

Q. 3 Définir une fonction `bool recompute_clusterings(int k, int n, vector *centers, vector *data, int *clustering)` prenant en arguments : un entier `k` qui est la taille du tableau de vecteurs `centers`, un entier `n` qui est la taille du tableau `data` et du tableau `clustering`. La fonction doit remplir le tableau `clustering` pour y stocker, dans chaque case d'indice i, l'indice dans `centers` du vecteur le plus proche de `data[`i`]`. Votre fonction devra renvoyer `true` si et seulement si une des données a changé de regroupe.

Q. 4 Définir une fonction `void recompute_centers(int k, int n, vector *centers, vector *data, int *clustering)` prenant en arguments les mêmes données que la question précédente et mettant à jour les centres de chacun des groupes. Aussi le tableau `centers` ne doit pas être lu par votre algorithme, chaque case `centers[i]` doit recevoir le vecteur moyen des vecteurs se trouvant dans le regroupe d'indice `i` d'après le tableau `clustering`.

Q. 5 Définir l'algorithme des k-moyennes dans une fonction `void kmeans(int k, int n, vector *centers, vector *data, int *clustering)`, prenant en arguments les mêmes données que les questions précédentes et itérant :

- mise à jour des centres ;
- mise à jour des groupes ;

jusqu'à stabilisation des groupes.

Q. 6 Mettre en place de quoi tester l'algorithme implémenté. On pourra :

- proposer une génération aléatoire de jeu de données de taille n ;
- proposer une description textuelle d'une partition, en calculant par exemple pour chaque classe son diamètre ;
- faire un programme qui prend en ligne de commande les paramètres k et n ;
- proposer une transcription dans un fichier texte d'un jeu de données d'une part et d'une partition d'autre part, puis coder des fonctions de lecture de ces fichiers en OCaml afin de réutiliser les fonctions d'affichage du TP correspondant en OCaml.