TD IPT Apprentissage automatique

Algorithme k-moyennes

On considère avoir les données sous la forme d'une liste `X_donnees`, chaque élément de `X_donnees` étant une liste de réels de longueur d indépendante de l'élément considéré (*donc* `X_donnees` *est une liste de listes de réels*).

Pour les exemples, on prendra

```
X_donnees = [[-0.65, -0.4 ], [-0.85, 0.25], [-0.6, -0.18], [-0.5, -0.23], [0.04, 0.9], [0.25, 0.7],
[0.3, 0.55], [0.01, 0.8], [0.6, -0.05], [0.25, 0.5], [-0.35, 0.08], [-0.8, -0.2],
[-0.78, -0.2], [0.9, -0.05], [-0.5, 0.2], [0.09, 0.6], [0.15, 0.4], [0.9, -0.45],
[0.7, -0.3], [0.6, -0.4], [0.8, -0.25], [-0.8, 0.], [0.5, -0.4], [0.03, 0.85], [0.05, 0.35]]
```

Exercice 1. Ecrire une fonction `initialiser_centres(X_donnees, k)` qui prend en entrée une liste de listes de réels `X_donnees` comme ci-dessus, `k` un entier naturel non nul (c'est le k de l'algorithme des k-moyennes), et qui renvoie une liste formée de `k` éléments différents pris au hasard dans `X` (*on pourra utiliser la fonction* `choice`).

Remarque 1. *On supposera* `k` $\leq$ `len(X_donnees)`.

Exercice 2. Ecrire une fonction `trouver_cluster(X, centres)` qui prend en entrée un élément `X` de `X_donnees`, une liste de points `centres` (les centres des clusters à déterminer), et qui renvoie l'indice j tel que `centres[j]` soit le point le plus près de `X` parmi les points de `centres` (*on utilisera la fonction* `d_carre` *du TP précédent*).

Remarque 2. *On pourra, si besoin, taper* `from math import inf`, *et utiliser* `inf` *pour* $+\infty$.

Exercice 3. En déduire une fonction `clusters(X_donnees, centres)` qui prend en entrée une liste de listes de réels `X_donnees` comme ci-dessus, une liste de points `centres` (les centres des clusters à déterminer), et qui renvoie une liste d'entiers `L` telle que `centres[L[j]]` soit le point le plus près de `X_donnees[j]` parmi les points de `centres`.

Remarque 3. *Ainsi, pour* $r \in [\![0, k-1]\!]$, *le cluster numéro* r *est donné par*

$$C_r = \{\texttt{X\_donnees[j]}, \text{ pour } \texttt{j} \text{ tel que } \texttt{L[j]} = r\}.$$

Il se peut qu'un (ou plusieurs) cluster ainsi formé soit vide. Dans ce cas, on ne pourra pas calculer le nouveau centre correspondant. Il y a plusieurs façons de gérer ceci. J'ai choisi le plus simple ici : on va gérer une variable booléenne qui vaudra `True` s'il n'y pas eu dispartion d'un (ou de plusieurs) cluster, `False` sinon. Et, si elle vaut `False`, on stoppera l'algorithme des k-moyennes : il ne renverra rien de pertinent.

Exercice 4. Ecrire une fonction `nouveaux_centres(X_donnees, L, k)` qui prend en entrée `k` un entier naturel (*le même qu'à l'exercice 1*), `X_donnees` comme ci-dessus, une liste d'entiers `L` comme celle renvoyée par la fonction précédente (*ainsi* `L` *est à valeurs dans* $[\![0, \texttt{k}-1]\!]$), et qui renvoie une liste (*de longueur* `k`) dont le j-ème élément est l'isobarycentre de C_j (*donc la fonction renvoie une liste de liste de réels*), ainsi qu'un booléen valant `True` s'il n'y pas eu dispartion d'un (ou de plusieurs) cluster, `False` sinon.

Exercice 5. Ecrire une fonction `k_means(X_donnees, k)` qui prend en entrée une liste de listes de réels `X_donnees` comme ci-dessus, `k` un entier naturel non nul, et qui, s'il n'y a pas disparition d'un cluster, renvoie la liste `L` donnant la partition obtenue par l'algorithme des k-moyennes (*c'est la liste renvoyée par la fonction* `clusters` *lorsque la liste renvoyée par* `nouveaux_centres` *n'évolue plus*), ainsi que la liste des centres associés (*pour éviter de les recalculer par la suite*), ainsi qu'un booléen valant `True`. Par contre, s'il y a eu disparition d'un cluster, il renvoie les deux listes en cours (`L` et `centres`), ainsi que `False`, et dans ce cas, les deux listes renvoyées n'ont aucune signification.

Exercice 6. Taper

```
couleur = ['blue', 'red', 'green', 'yellow', 'black', 'grey', 'pink', 'purple', 'brown',
                                                                        'cyan'].
```

Dans cet exercice, on va supposer $d = 2$ (autrement dit, les données sont des points du plan, c'est le cas de notre exemple) et `k` ≤ 10 (10 car on a donné 10 couleurs ci-dessus).

Ecrire ensuite une fonction `affiche(X_donnees, L)` qui prend en entrée une liste de listes de réels `X_donnees` comme ci-dessus, `L` une liste d'entiers à valeurs dans $[\![0, \texttt{len(couleur)}-1]\!]$ (*typiquement la première liste renvoyée par* `k_means(couleurs, k)`), et qui affiche les points de `X_donnees` sous forme de croix dans le plan, en mettant à `X_donnees[j]` la couleur numéro `L[j]` de la liste `couleur`.

Comme la partition renvoyée par l'algorithme des k-moyennes n'est pas forcément optimale, on va exécuter plusieurs fois l'algorithme et choisir la meilleure des partitions. Pour cela, il faut d'abord créer la fonction qui calcule la somme des moments d'inertie – ou score – puisque c'est elle qui sert de critère.

Exercice 7. Ecrire une fonction `Inertie(X_donnees, L, centres)` qui prend en entrée une liste de listes de réels `X_donnees` comme ci-dessus, `L` une liste d'entiers naturels non nuls, `centres` la liste des isobarycentres des clusters associés à `L`, et qui calcule la somme des moments d'inertie associée.

Exercice 8. En déduire une fonction `k_means_optimise(X_donnees, k, nb)` qui renvoie :

- la liste `L` donnant la partition obtenue par l'algorithme des k-moyennes, ainsi que la liste des centres associées, de somme des moments d'inertie minimale parmi `nb` appels différents de la fonction `k_means` **qui n'ont pas eu de disparition de cluster**,
- ainsi que la valeur de cette somme.

Enfin, pour choisir le bon k :

Exercice 9. Ecrire une fonction `choisir(X_donnees, nb)` qui prend en entrée une liste de listes de réels `X_donnees` comme ci-dessus et un entier naturel non nul `nb`, et qui trace en fonction de k en abscisse la somme des moments d'inertie minimale renvoyée par `k_means_optimise`.